Using Reinforcement Learning for Optimization of a Workpiece Clamping Position in a Machine Tool

Vladimir Samsonov[1], Chrismarie Enslin[1], Hans-Georg Köpken[2], Schirin Baer[2] and Daniel Lütticke[1]

[1]*Institute of Information Management in Mechanical Engineering, RWTH Aachen University, Aachen, Germany*

[2]*Siemens AG, Digital Factory Division, Nuernberg, Germany*

Keywords: Reinforcement Learning, Soft Actor-Critic, Supervised Learning, Industrial Manufacturing, Process Optimisation, Machine Tool Optimisation.

Abstract: Modern manufacturing is increasingly data-driven. Yet there are a number of applications traditionally performed by humans because of their capabilities to think analytically, learn from previous experience and adapt. With the appearance of Deep Reinforcement Learning (RL) many of these applications can be partly or completely automated. In this paper we aim at finding an optimal clamping position for a workpiece (WP) with the help of deep RL. Traditionally, a human expert chooses a clamping position that leads to an efficient, high quality machining without axis limit violations or collisions. This decision is hard to automate because of the variety of WP geometries and possible ways to manufacture them. We investigate whether the use of RL can aid in finding a near-optimal WP clamping position, even for unseen WPs during training. We develop a use case representing a simplified problem of clamping position optimisation, formalise it as a Markov Decision Process (MDP) and conduct a number of RL experiments to demonstrate the applicability of the approach in terms of training stability and quality of the solutions. First evaluations of the concept demonstrate the capability of a trained RL agent to find a near-optimal clamping position for an unseen WP with a small number of iterations required.

1 INTRODUCTION

Computer Numerical Control (CNC) machining is widely used in the manufacturing industry. To produce a part of a certain geometry, a CNC program defining relative movements of the cutting tool to the workpiece (WP) is created as a first step. Additionally, a clamping position of the WP in the milling machine space has to be chosen manually, solely based on the human expert knowledge and experience.

The controller translates the relative toolpath into real axis moves based on the machine kinematics and the chosen WP clamping position. Therefore, depending on the choice of the WP clamping position, the tool movements can be conducted by various machine axes resulting in different levels of processing speed, quality, machine wear and energy efficiency.

In this study we investigate whether a deep reinforcement learning agent (from here on referred to as the RL agent) can learn a strategy to find a clamping position close to the optimum for a given WP. Furthermore, we investigate whether an RL agent can not only reproduce learned optimal solutions, but come up with abstract solution strategies capable of efficiently generalising to new, previously unseen, WPs. The RL agent is expected to learn WP positioning strategies addressing two tasks simultaneously: avoiding axis collisions and finding optimal clamping positions in terms of axis accelerations and traveled distances.

2 RELATED WORK

Over years many efficient approaches have been proposed to tackle optimisation tasks. Random search (Anderson, 1953) and simulated annealing (van Laarhoven and Aarts, 1987) are conventional search methods which attempt to approximate the optimal solution for optimisation problems. Due to the large search space for complex problems, calculation time and computational efforts are enormous and

Samsonov, V., Enslin, C., Köpken, H., Baer, S. and Lütticke, D.
Using Reinforcement Learning for Optimization of a Workpiece Clamping Position in a Machine Tool.
DOI: 10.5220/0009354105060514
In *Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS 2020) - Volume 1*, pages 506-514
ISBN: 978-989-758-423-7

the solutions are rather use-case specific than generic. Genetic search algorithms (Hajela and Lin, 1992) are more efficient in finding an approximately optimal solution, but there is still a large amount of engineering effort needed for every different problem definition. These methods search for the global optimum for a certain problem while inserting randomness to try and circumvent possible local optimums. The further disadvantages of these methods, besides computational efforts, include not considering the current state of the environment during the search, not "remembering" previous knowledge gained in the optimisation process and having to be started from scratch every time.

We are aiming for a solution that is able to generalise to various problems with the possibility of transferring existing knowledge to new scenarios. The recent success of RL and its powerful ability to solve complex tasks by trial and error (Vinyals et al., 2019) inspired the use of this method for our optimisation problem. As discussed in detail in (Sutton and Barto, 2018), an RL agent has a defined environment that it interacts with and receives feedback from. The RL agent will find itself in a state (S) and performs an interaction with the environment by choosing a specific action (a), from a finite set of possible actions. A feedback signal will be returned in the form of a reward (R) and a definition of the new state (S') will be provided. The aim is to learn a function approximation for the policy that chooses the optimal actions. Q-values are estimations of the future value that each action could bring and helps in the decision-making of which action to choose. The value function is an estimation of the total possible future values.

In this work we use a novel RL Algorithm referred to as Soft Actor-Critic (SAC) (Haarnoja et al., 2018). It is an off-policy algorithm based on an actor-critic architecture. The main peculiarity of the algorithm is the use of an entropy term as part of the value function that intensifies exploration in an efficient way. Comparing to other off-policy algorithms, SAC is fairly robust and stable, thus reducing the effort of its application to a certain task considerably.

More and more research focuses on the usage of RL methods in the manufacturing domain, for example to solve robot control (Johannink et al., 2019), for online optimisation of flexible manufacturing systems in terms of energetic load management (Bakakeu et al., 2018) and for online job shop scheduling in flexible manufacturing systems (Baer et al., 2019). At the same time RL is proved to be an efficient tool for solving a number of optimisation problems in other domains. (Li and Malik, 2016) and (Li and Malik, 2017) demonstrate that RL can be used as an optimisation algorithm for high dimensional stochastic optimisation problems. Using an RL agent as an optimizer for training shallow neural networks or finding an optimal solution for a given objective function converges faster and/or reaches better optima comparing to commonly used optimisation approaches. A number of RL Architectures are proposed to solve combinatorial optimisation problems in (Nazari et al., 2018), (Bello et al., 2016) and (Kool et al., 2018).

To conclude RL is widely used across various manufacturing engineering and other domains and proves to be a capable tool for addressing various optimisation tasks.

3 EXPERIMENTAL SETUP

The whole experimental setup is built around the WP clamping position optimisation problem, formalised as a sequential decision making process and covers the setup for the RL agent training and evaluation. The ultimate goal is to train and evaluate an RL agent capable of finding a valid and near-optimal clamping position for seen, as well as unseen WPs. A valid clamping position is defined as a position that causes no axis collisions during the milling process. A near-optimal clamping position for a given geometry offers close to minimal energy consumption and machine wear via minimising axis accelerations and total traveled distances along the machine axes.

3.1 Milling Process Description

All investigations are conducted on the simulation of a 3-Axis milling process using SinuTrain software. Within the simulation setup various WP geometries can be generated with a few examples displayed in Figure 1. The WP geometry changes by varying the orientation of the slot to be milled into the WP.

For every simulation run, a clamping position for the WP needs to be defined within the working space of the milling machine. With respect to the defined clamping position, SinuTrain simulates the movements of the machine's axes during the milling process, recognises any axis collisions and returns information about the speed, acceleration and travelled distance recorded during the milling process along the machine axes.

3.2 RL Task Formalisation

The WP clamping position optimisation task can be formalised as a *Markov Decision Process (MDP)* (Howard, 1960). According to the Markov property

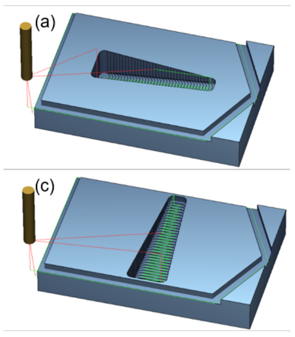

Figure 1: Examples of the Shape of the WP with Slot Angles (SA) (a) 0° (B) 45° (C) 90° (C) 180°.

the following state (S') in an MDP is dependent only on the current state (S) and action (a) pair.

The RL agent can optimise the WP clamping position by iteratively changing the WP position in the work space of the milling machine. After a predefined number of WP position changes is conducted, the optimisation process is terminated. The last position before the termination is considered to be the final WP position proposed by the agent. Such a cycle from random initialisation of the WP position to the termination of the optimisation process is called an *episode*. A number of allowed WP position changes within the episode is referred to as a number of *steps* within an episode.

The addressed task is non-stochastic, meaning that a certain action (a) in a given state (S) will always result in the same following state (S'). The clamping position problem is designed as a sequential decision making process with a finite number of steps (i) within an episode and with the goal, that the trained agent finds a near-optimal position within a few steps.

3.2.1 State-Action Representation

At every optimisation step the RL agent is allowed to move the WP in the X and Y-axis directions, as well as to rotate the WP. The magnitude of the changes made to the WP position at each optimization step is controlled by the action of the RL agent. Undertaken step sizes can range from zero up to the allowed maximum limit. In our experiments we limit the maximum single move along X and Y-axis to 40 mm. The maximum step size for the rotation angle is equal to 35°.

The agent's current state plus the action defines the new clamping position of the WP, with which the milling process simulation is executed. This is summarised in Figure 2, such that the WP is initially placed within the work space at $(X^0, Y^0, RotationAngle^0)$. At each step ($i$) the RL agent incrementally moves the WP by $(\Delta X, \Delta Y, \Delta RotationAngle)$ into a new position

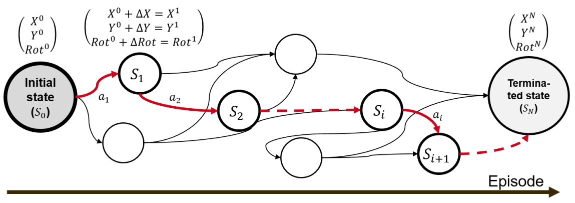
$\begin{pmatrix} X^0 \\ Y^0 \\ Rot^0 \end{pmatrix}$
$\begin{pmatrix} X^0 + \Delta X = X^1 \\ Y^0 + \Delta Y = Y^1 \\ Rot^0 + \Delta Rot = Rot^1 \end{pmatrix}$
$\begin{pmatrix} X^N \\ Y^N \\ Rot^N \end{pmatrix}$
Initial state (S_0)
a_1
S_1
a_2
S_2
S_i
a_i
S_{i+1}
Terminated state (S_N)
Episode

Figure 2: Formalisation of the Clamping Position Problem as an MDP with Initial State, Intermediate States and Terminated State within One Episode.

$(X^{i+1}, Y^{i+1}, RotationAngle^{i+1})$, with which the simulation is executed again and a feedback is given back to the RL agent. This feedback states how much of an improvement the change brought to the overall milling performance in terms of avoiding axis collisions, optimising the total traveled distance and minimising acceleration over all machine axes in the form of reward (R_i).

The observation space consists of eleven dimensions. Firstly, features describing the WP clamping position in the machine working space $(X, Y, Rotation\,Angle)$, the current WP geometry $(Slot\,Angle)$, as well as a number of process parameters captured during the milling process at the chosen clamping position by the agent is included. These process parameters are the integrals of the squared accelerations in the X- and Y-axis directions (called (e_X, e_Y) since they correspond to energy), total traveled distances in the X- and Y-axis directions (d_X, d_Y) and indicators of reaching axis limits and axis collisions $(limit\,X, limit\,Y)$. Lastly, we introduce the number of steps left within an episode before termination (*steps left*) as an additional input, as proposed by (Pardo et al., 2018). This helps the agent to determine whether it can reach a certain point in the working space with the steps left in the episode and is especially beneficial if the number of steps allowed within an episode is small. The final input is a tuple defined as *(X, Y, Rotation Angle, Slot Angle, d_X, d_Y, e_X, e_Y, limit X, limit Y, steps left)*.

3.2.2 Reward Function

A reward function is required for the RL agent to guide it through the optimisation process. It incentivises the RL agent to learn the optimal clamping position. Higher rewards result from the minimisation of accelerations and travelled distances. All observations have to be combined into a total reward. The following equations invert the observations (lower values correspond to higher rewards) and normalises them:

$$e_{\text{X, norm}} = (e_{\text{X, max}} - e_{\text{X}})/(e_{\text{X, max}} - e_{\text{X, min}}) \quad (1)$$
$$e_{\text{Y, norm}} = (e_{\text{Y, max}} - e_{\text{Y}})/(e_{\text{Y, max}} - e_{\text{Y, min}}) \quad (2)$$
$$d_{\text{X, norm}} = (d_{\text{X, max}} - d_{\text{X}})/(d_{\text{X, max}} - d_{\text{X min}}) \quad (3)$$
$$d_{\text{Y, norm}} = (d_{\text{Y, max}} - d_{\text{Y}})/(d_{\text{Y, max}} - d_{\text{Y, min}}) \quad (4)$$

where:
$e_{.,norm}$ – normalised values for the integrated squared accelerations along the X- and Y-axis
$e_{.,min}$, $e_{.,max}$ – minimum and maximum values for e_X and e_Y
$d_{.,norm}$ – normalised values for the travelled distances along the X- and Y-axis
$d_{.,min}$, $d_{.,max}$ – minimum and maximum values for d_X and d_Y

For the considered machine, the X-axis is heavier than the Y-axis. Therefore, $e_{X,norm}$ and $d_{X,norm}$ is weighted higher than $e_{Y,norm}$ and $d_{Y,norm}$ in the combined reward components e and d:

$$e = 2e_{\text{X, norm}} + e_{\text{Y, norm}} \quad (5)$$
$$d = 2d_{\text{X, norm}} + d_{\text{Y, norm}} \quad (6)$$

The total reward after every action is a combination of the acceleration related term e from eq. (5) and the distance related term d from eq. (6), such that:

$$R_i = \begin{cases} 0.7e + 0.3d & \text{no axis collision} \\ -1 & \text{axis collision} \end{cases} \quad (7)$$

The weights in eq. (7) give a higher importance to the acceleration term e than to the distance term d, because accelerations also cause disturbances in the machining process. When an axis collision occurs a reward of -1 is immediately awarded. The tuples for the state-action-reward representation are briefly summarised in Table 1.

Table 1: Summary of the Main Parameters of the Optimisation Task Formalised as an MDP.

State	*(X, Y, Rotation Angle, Slot Angle, d_X, d_Y, e_X, e_Y, limit X, limit Y, steps left)*
Action	*(ΔX, ΔY, ΔRotation Angle)*
Reward	$R_i = 0.7e + 0.3d$ $e = 2e_{X,norm} + e_{Y,norm}$ $d = 2d_{X,norm} + d_{Y,norm}$

3.3 RL Agent Training and Validation

While learning the WP position optimisation strategy the RL agent conducts a number of training episodes. The training and evaluation process employed in this study is summarised in Figure 3. At the start of each episode a new WP geometry is generated and the WP position is randomly initialised. WP geometry stays unchanged within one episode. During training the RL agent encounters various WP geometries (slot angles (SA)) with the exception of the angles ranging between 40° and 50°. The trained agent is evaluated on an unseen WP with the slot angle of 45°. This allows to determine if the given RL agent is not overfitting to solutions seen during the training, but learns a generic optimisation strategy capable of interpreting and solving unseen WP geometries.

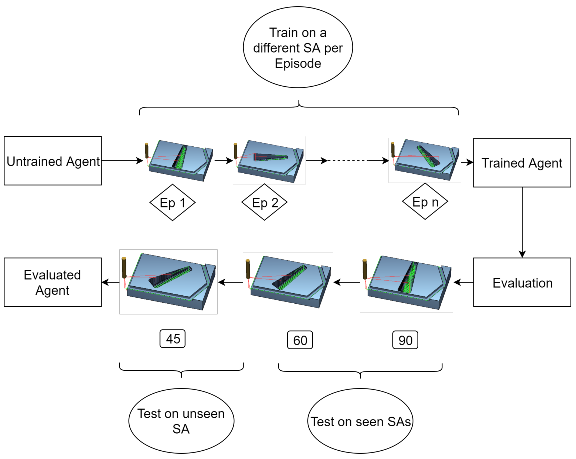

Figure 3: Training and Evaluation Process of the RL Agent.

Since the last WP position within an episode is considered to be the final solution, the last reward in an episode is an important metric. It provides an indication whether the agent found the optimal clamping position. A trained RL agent is considered to be capable of generalisation if it can find a near-optimal clamping position for an unseen WP with no additional learning within one episode.

The WP optimisation task is implemented as an OpenAI Gym environment (Brockman et al., 2016). We use the SAC implementation from stable baselines (Hill et al., 2018) for solving the optimisation task in the created environment. All training and evaluation is conducted in docker containers (Merkel, 2014) to ensure the independence of the experiments from the used software and hardware. Each RL setup is independently trained and evaluated three times with three fixed random seeds shared across all experiments. This allows to evaluate and compare not only the final performance, but also the stability of learning while ensuring the reproducibility of the results.

3.3.1 Approximation of the Simulation with Machine Learning

The training environment for the RL agent is a SinuTrain simulation incorporating various WP geometries and machine kinematics. Depending on the task, a SinuTrain simulation takes at least 4 seconds for a successful run and 1 second for a failed run. In this case a failed run would refer to an axis collision and early termination of the run. During the training process thousands of interactions between the RL agent and the simulation may be required, which results in impractically long training times. A considerably faster environment for RL agent testing is created to mimic the SinuTrain simulation with a combination of machine learning (ML) models. These models predict simulation outcomes for a given set of input parameters. The steps to building these ML models are data generation, ML-ensemble creation for prediction and ML-ensemble training and validation.

3.3.2 Data Generation

Predictive models approximating the milling process behaviour have to be built from the SinuTrain simulation runs with a set of input parameters (X, Y, $Rotation\ Angle$, $Slot\ Angle$) that covers the entire parameter space. These parameters are generated according to *randomised design*. Random sampling achieves good performance for large sample counts (Garud et al., 2017). In total 180000 points are generated.

3.3.3 ML Ensemble for Prediction

The ML models are structured as a waterfall of a classifier, followed by a set of regression models. Figure 4 provides an overview of the ML-ensemble architecture used to mimic the functioning of a 3-axis milling machine. The classifier has the purpose of determining whether a given WP position in the machine space would cause an axis collision during the milling process or complete successfully, and furthermore returns the corresponding error message. These error messages contain the axis of the machine that caused the collision and whether it was in the positive or negative direction of movement.

Only the set of inputs that will lead to successful runs are propagated further to the regression models. Four regression models are trained to predict the e_X, e_Y, d_X, d_Y values (defined in section 3.2.1) during the WP processing. These values are used as input to the RL agent and to calculate the reward.

3.3.4 ML-Ensemble Training and Validation

Ten-fold cross-validation (CV) is used to validate the ML models (Kohavi, 1995). The data is split into ten equal parts, nine parts are joined together and used as the training set and the tenth part is used as the validation set. This process is repeated until each of the parts was used once as the validation set and each repetition of the process is called a CV-fold. This process ensures that there is never any overlap between the training and the validation data. For each of the folds a prediction accuracy is calculated, and the average of these accuracies is returned as the final performance evaluation of the ML model.

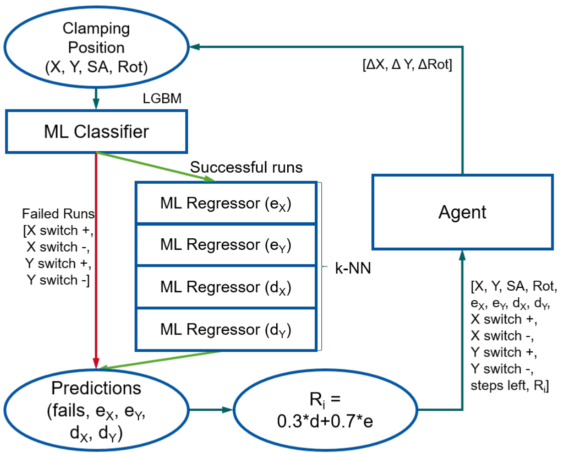
Clamping Position (X, Y, SA, Rot)
LGBM
ML Classifier
Successful runs
Failed Runs [X switch +, X switch -, Y switch +, Y switch -]
ML Regressor (e$_X$)
ML Regressor (e$_Y$)
ML Regressor (d$_X$)
ML Regressor (d$_Y$)
k-NN
Agent
[ΔX, Δ Y, ΔRot]
[X, Y, SA, Rot, e$_X$, e$_Y$, d$_X$, d$_Y$, X switch +, X switch -, Y switch +, Y switch -, steps left, R$_i$]
Predictions (fails, e$_X$, e$_Y$, d$_X$, d$_Y$)
R$_i$ = 0.3*d+0.7*e

Figure 4: Architecture of the ML-Ensemble. A Classifier Predicts If the Simulation with the given Input Parameters Is Going to Fail or Run Successfully. In the Case of a Positive Result, Regression Models Predict Output Parameters (One Model per Parameter). Gradient Boosting, with a Degree of 2 Polynomial Feature Basis Expansion, Is Used as a Classifier and K-Nearest Neighbors (K-NN), with $k = 75$ and Uniform Weighting, Is Used for the Regression Models.

Table 2: Summary of the Classifier Accuracy.

	F1-Score	**Totals**
Success	0.990	34943
Limit X +	0.949	4200
Limit X -	0.901	36608
Limit Y +	0.960	16892
Limit Y -	0.932	46730
Weighted Average	0.942	139373

The accuracy of the classifier is determined by the weighted F1-score, as it is important to have good performance in the prediction of each of the classes. Table 2 summarises the F1-score for each of the classes and the overall classification is evaluated by a weighted average according to the sizes of the different classes. This overall F1-score is 94.2% and the fact that 99% of the successful runs are correctly identified means that the regression models receive the information they need to make accurate predictions.

For the regression models, the R-squared metric is used to compare models and to choose the optimal model. The R-squared value for each regression

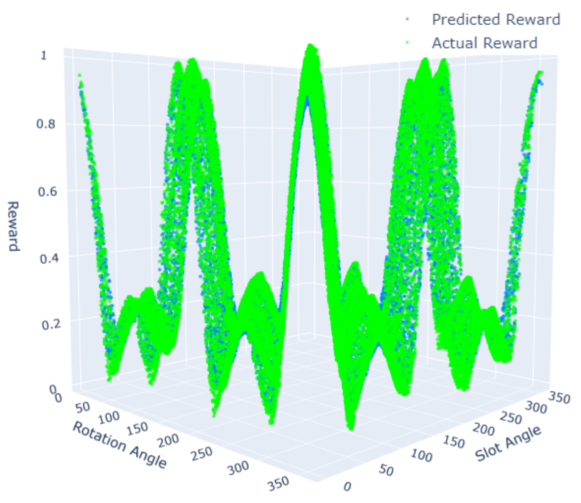

Figure 5: Predicted versus Actual Rewards for Successful Simulation Runs.

model is quoted in Table 3 and Figure 5 depicts the reward surface comparing predicted and actual rewards. The reward surface represents only the part of eq. (7) corresponding to no axis collisions and is a combination of the predictions for e_X, e_Y, d_X, d_Y. These predictions are summarised as described in section 3.2.2. The deviation of the predicted values from the actual values are almost negligible, therefore the surface produced by the ML-ensemble is a good representation of the true surface.

Table 3: Summary of the Regression Models Accuracy.

	R-Squared
e_X	0.951
e_Y	0.947
d_X	0.840
d_Y	0.835

The resulting ML-ensemble can predict the outcome of any simulation iteration under 0.01 seconds. This offers up to a 400-fold improvement in the training speed of the chosen RL algorithms.

4 EXPERIMENTAL RESULTS

The training of the RL agent is conducted on various WP geometries and validated on two seen WP geometries (slot angle of 90° and 60°) and an unseen WP geometry (slot angle of 45°) as described in section Section 3. It tests whether the RL agent "remembers" good solutions seen during the training without overfitting to them and whether it can generalise to unseen WPs.

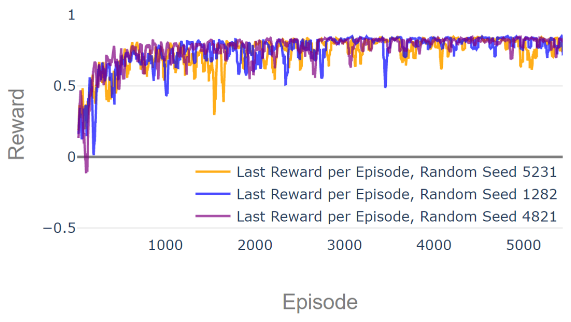

Figure 6: Last Reward per Episode during Training, Episode Length - 110 Steps, Total Training Steps - 600.000.

Firstly, we investigate if the RL agent is capable of finding an optimal clamping position for an unseen WP within 110 steps per episode. It is a fairly large number of steps which allows the RL agent to conduct more exploration and recover from mistakes, such as not heading straight to the optimal point from the beginning of an episode.

Figure 6 and Figure 7 show the last reward per episode, with a smoothing window of 30 episodes, during training and validation respectively. The RL agent is trained over 600.000 steps before the validation starts. The three lines of different colours represent the achieved last reward per episode for independent training (white background area) and validation (the light-grey background area for the 90° slot angle, the grey background area for the 60° slot angle and the dark-grey background area for the 45° slot angle) runs with different random seeds. The maximum possible reward depends on the slot angle because of the implicit differences in the milling process. This explains the changes in general reward level seen at different validation stages for each slot angle. At all validation runs the RL agent never gets negative rewards at the last episode step. It proves that the RL agent learns how to avoid axis collisions for different initialisation positions across the whole machine space. Moreover, the last reward per episode stays close to the maximum for both seen WP configurations with 90° and 60° slot angles, as well as for the unseen 45° configuration. The RL agent is capable of consistently placing the WP with an unseen geometry in positions close to the optimum.

Figure 8 shows position changes of the WP in the three-dimensional space of $(X, Y, Rotation\ Angle)$ at each optimisation step over some randomly sampled validation episodes from all three independently trained RL agents. This allows for understanding of the actions of the RL agent and to validate if the learned strategy optimally solves the task. We provide the visualisation only for the WP with a 45° slot angle, since understanding the generalisation capabil-

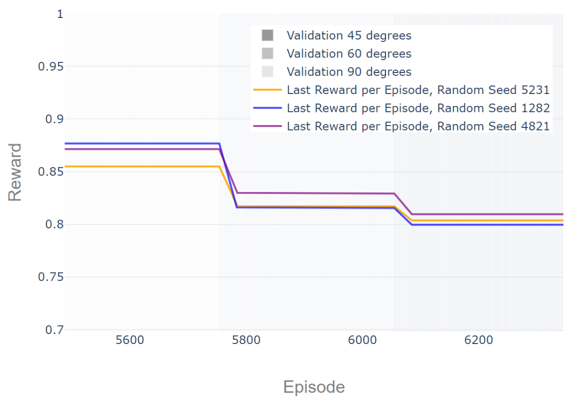

Figure 7: Last Reward per Episode during Validation, Episode Length - 110 Steps, Total Training Steps - 600.000.

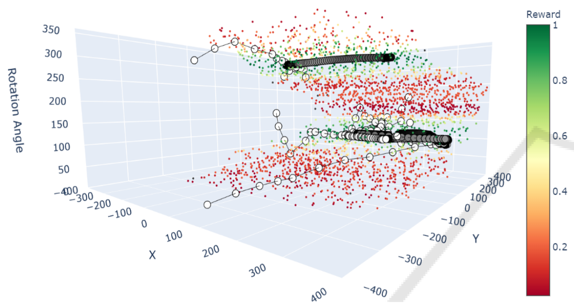

Figure 8: Moves of the RL Agents (110 Steps per Episode) in the Machine Space. The Episode Trajectories Are Randomly Sampled across the Three Trained RL Agents.

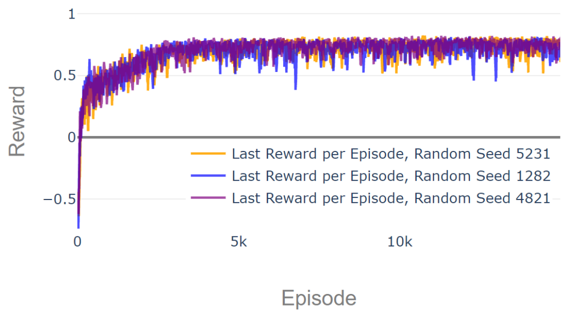

Figure 9: Last Reward per Episode during Training, Episode Length - 20 Steps, Total Training Steps - 300.000.

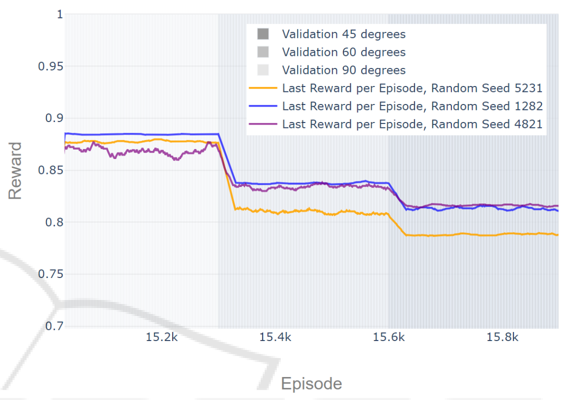

Figure 10: Last Reward per Episode during Validation, Episode Length - 20 Steps, Total Training Steps - 300.000.

ities of the RL agent is the main purpose of this study. One connected line in the figure corresponds to one optimisation episode. Points show positions of the WP in the machine working space, while lines depict the transitions following the RL agent's chosen actions. The colour of the points is changing gradually from white to black throughout the episode such that the start and end points can easily be recognised. The cloud of points represents the area where the milling process runs through without axis collisions and the empty space surrounding the point cloud represents the WP positions where the simulation fails to run. The colour of every point in the cloud encodes the reward for placing the WP in the given position. It can be seen, that across all initialisation points the RL agent first moves the WP out of the axis collision area in the shortest possible way to minimise the penalty. Afterwards, it gradually moves the WP towards one of the two optimal areas with the highest rewards. As soon as the high reward area is achieved the RL agent significantly decreases the step-size and stays within the high-reward area.

Looking at Figure 8, the RL agent mostly needs a small fraction of the available steps to reach areas of high reward. The overall goal is to find an optimal WP clamping position while keeping the number of iteration steps at a minimum. Therefore we reduce the number of steps per episode to 20. This requires the agent to deviate as little as possible from the direction of the optimal point during the episode, in order to reach it before or at the last step.

With 20 steps per episode, up to 300.000 training steps are required to achieve the best result (Figure 9). This can be explained by the fact that the RL agent does not need to learn to stay in the area of high reward for many steps before the termination of the episode. At the same time, longer training leads to a slight performance decrease.

Figure 10 shows that three independently trained RL agents have higher differences in the final performance comparing to the 110 steps per episode RL agents. This points towards less stable training. During two training runs the RL Agents were able to learn optimal solutions for all three validation slot angles. Depending on the initialisation point, the RL Agent does not always move the WP into the same place as for 110 steps per episode. Nevertheless, all chosen points are very close to the optimum. The third trained RL agent underperforms on all three validation angles.

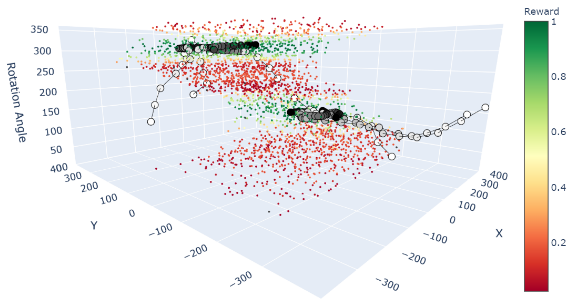

Figure 11: Moves of the RL Agents (20 Steps per Episode) in the Machine Space. The Episode Trajectories Are Randomly Sampled across the Three Trained RL Agents.

The learned optimisation strategy is visualised in Figure 11. Similarly to the case with 110 steps per episode, the RL agent moves the WP out of the axis collision area in big steps first and then gradually approaches the optimal points.

A further decrease of the training steps per episode leads to a decrease of the stability of the results because depending on the WP initialisation point, the RL agent may not always be capable of reaching the optimal point.

5 CONCLUSION AND OUTLOOK

We formalised the problem of finding an optimal WP clamping position as an MDP problem, set up a training environment for an RL agent using an approximation of the SinuTrain milling simulation with stacked ML models and successfully demonstrated that an RL agent can solve the WP clamping position optimisation task. Through a number of evaluations we demonstrated that a trained RL agent is capable of generalisation to new, previously unseen (but similar in geometry) WPs. In the case of a 3-axis machine tool, it is capable of finding a valid and near-optimal clamping position within a limited number of optimisation steps for an unseen WP without additional training when the WP geometry is explicitly described as a part of the state space.

This paper is a proof-of-concept work demonstrating that RL can be applied in complex optimisation tasks from the field of mechanical engineering, such as the search for an optimal WP clamping position. In this study, we elaborated only on a simple 3-axis milling machine use case. In future work, the demonstrated results can be transferred to more complex WPs requiring processing on a 5-Axis milling machine. Another important aspect of the possible future research focuses on a solution able to handle a variety of WPs, without providing a set of hand-crafted features describing the WP. For example, we can define the optimisation task as a Partially Observable Markov Decision Process (POMDP) where WP related features can be learned indirectly by the RL agent through simulation feedback at the beginning of the optimisation process. Possible solutions for such a POMDP optimisation task includes the use of frame stacking, RL with Recurrent Neural Networks or a meta-learning method. To progress further in the direction of a production capable solution, we would need to improve the RL training efficiency and transferability of RL-learned solutions to new WPs and machine tools with complex axis movements with no or little extra training required.

REFERENCES

Anderson, R. L. (1953). Recent advances in finding best operating conditions. *Journal of the American Statistical Association*, 48(264):789–798.

Baer, S., Bakakeu Romuald Jupiter, Meyes Richard, and Meisen Tobias (25.09.2019-27.09.2019). Multi-agent reinforcement learning for job shop scheduling in flexible manufacturing systems. In *2019 Second International Conference on Artificial Intelligence for Industries (AI4I)*. IEEE.

Bakakeu, J., Tolksdorf, S., Bauer, J., Klos, H.-H., Peschke, J., Fehrle, A., Eberlein, W., Bürner, J., Brossog, M., Jahn, L., and Franke, J. (2018). An artificial intelligence approach for online optimization of flexible manufacturing systems. *Applied Mechanics and Materials*, 882:96–108.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Garud, S. S., Karimi, I. A., and Kraft, M. (2017). Design of computer experiments: A review. *Computers and Chemical Engineering*.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

Hajela, P. and Lin, C.-Y. (1992). Genetic search strategies in multicriterion optimal design. *Structural Optimization*, 4(2):99–107.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2018). Stable baselines.

Howard, R. A. (1960). Dynamic programming and markov processes.

Johannink, T., Bahl, S., Nair, A., Luo, J., Kumar, A., Loskyll, M., Ojea, J. A., Solowjow, E., and Levine, S. (20.05.2019 - 24.05.2019). Residual reinforcement learning for robot control. In *2019 International Con-*

ference on Robotics and Automation (ICRA), pages 6023–6029. IEEE.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection.

Kool, W., van Hoof, H., and Welling, M. (2018). Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475 In Citavi anzeigen*.

Li, K. and Malik, J. (2016). Learning to optimize.

Li, K. and Malik, J. (2017). Learning to optimize neural nets.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2.

Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9839–9849.

Pardo, F., Tavakoli, A., Levdik, V., and Kormushev, P. (2018). Time limits in reinforcement learning. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4045–4054, Stockholmsmässan, Stockholm Sweden. PMLR.

Sutton, R. S. and Barto, A. (2018). *Reinforcement learning: An introduction*. Adaptive computation and machine learning. The MIT Press, Cambridge, MA and London, second edition edition.

van Laarhoven, P. J. M. and Aarts, E. H. L. (1987). Simulated annealing. In van Laarhoven, P. J. M. and Aarts, E. H. L., editors, *Simulated Annealing: Theory and Applications*, pages 7–15. Springer Netherlands, Dordrecht.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.