Optimization of Simulated Production Process Performance using Machine Learning

Andreas Leha, Dejan Pangercic, Thomas Rühr, Michael Beetz
Department of Computer Science, Technische Universität München
{leha, pangerci, ruehr, beetz}@in.tum.de

Abstract

This paper investigates integration of the supervised machine learning algorithms (Model Trees, Neural Networks) into a production plan realized in a physics-based realistic simulator. Proposed novelty is in that the learning capability is integrated into the control process which allows for online learning and on the fly control code modification. Running the process in a simulated environment enables hazardless experimenting with the system's setup and integral acquisition of data. Yielded optimization times obtained through learning outperform times of a production process solely based on averaging.

1. Introduction

The work reported in this paper contributes to the creation of a Manufacturing System (MS) [11] that through integration of cognitive capabilities achieves similar levels of flexibility, robustness and improvement as found in human machine shops. The two key areas for meeting that overall goal are knowledge and learning, the latter being the focal point exploited hereafter.

Suchlike MS has to be able, among other cognitive capabilities, to exercise learning of models that reflect the experience gained at execution level. This can be used to reason about the system-state in order to ensure feasibility, reliability, efficiency and flexibility, and also to compensate for non-deterministic behaviour (e.g. due to environmental conditions).

We pursue the above by proposing the following threefold contribution to the field: a)integration of standard Machine Learning algorithms [12, 10] into the b)leading-edge control process module based on Reactive Plan Language (RPL) [6] and Robot Learning Language (RoLL) [3]. Former is a control language that through implementation of *designator* and *fluent* constructs provides utilities for state description and monitoring, while the latter one serves for data acquisition and learning itself.

Production processes are simulated in c) the third component of the above mentioned triple - the Gazebo-based simulation [4] which mimics the main features of the real Flexible Manufacturing System by Festo Corporation as depicted in Fig. 1 - bottom left. The simulator serves as an outstanding testbed allowing us to gather data and perform learning on any arbitrary spatial or process configuration of the MS at zero damage risk. Its realistic physical engine is a core power-horse aiding in a faster development of the control process for a real hardware system.

In this work we investigate a joint-action scenario between the production of a playmaze (see Fig. 4a - left) and a mobile robot involved into two tasks: a)quality scanning of the finished playmaze and b)learning to grasp objects at the remotely located working place.

The overall objective in our demonstration example is to learn a function $f(t)$ minimizing the time the finished playmaze has to wait in order to undergo the quality check and simultaneously minimizing the waiting time of the mobile robot.

The testbed MS presented in this paper consists of a storage unit, a mill and turn station and an assembly station, all interconnected by a conveyor transport system equipped with a changeable number of pallet carriers (see Fig. 1). The industrial robot at the mill and turn station feeds the production process with bolts while the actual assembly of the playmaze is being carried out by another robot at the assembly station. The mobile robot can traverse freely around the MS and is in our case occupied with above mentioned tasks.

The production of the playmazes is showcased in the following video: [1].

The remainder of this paper is structured as follows: section 2 briefly describes the system architecture used. The learning model is laid out in section 3 along with the used learning algorithms. The results are presented in section 4 followed by the concluding remarks on future work.

1.1. Related Work

The review of the Artificial Intelligence algorithms applied to the manufacturing reveals, that so far genetic algorithms stemming from computationalism have been used to programmatically arrange production schedules for the best possible outcome based on a number of constraints, which are pre-defined by the user [1]. These rule-based programs cycle through thousands of possibilities, until the most optimal schedule is arrived at which all criteria best meet. On the connectionism side we noted the work by [9] that applied neural network classifier to automate inspection of manufactured parts. Shen et al in [8], similarly to ours, motivate learning to tackle dynamic and adaptive conditions and prevent failures of machines. They make use of the emerging MetaMorph architecture. Lastly, Ilghami et al in [2] propose a supervised CaMeL algorithm en route to learn how to acquire the domain knowledge the system may need.

2. Architecture

This section gives an overview of the crucial components of our system architecture. Fig. 1 shows their inter-

[1] http://www9.cs.tum.edu/research/cogmash/video/etfa09.avi

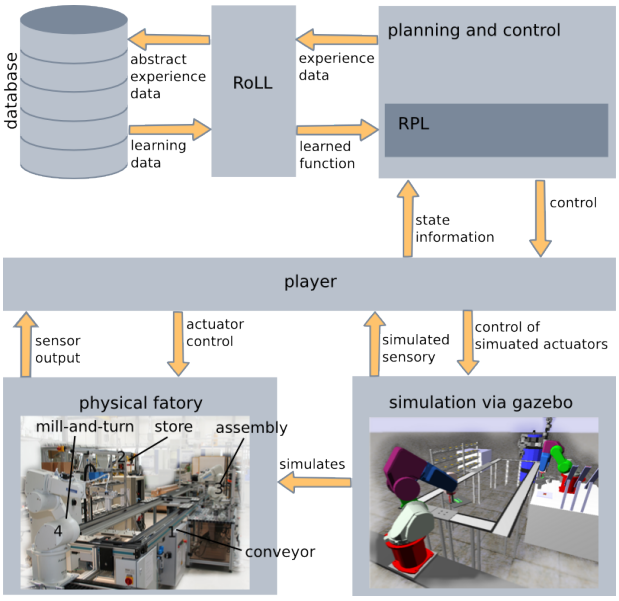


Figure 1: The main components of the system architecture: The middle-ware Player translates between either the physical or the simulated MS and the control level; RoLL provides the learning capabilities

action.

2.1. Simulation

We simulate our real MS using Gazebo [4]. This framework is physics-based and serves as an abstraction for the underlying physics engine Open Dynamics Engine (ODE). Gazebo provides and manages a world model consisting of *entities* with physical properties like shape, size and mass, and sensors that additionally return data from the simulated world. All physical parts of the real world are represented as entities and might, e.g. in case they constitute a robot arm, be connected through *hinge joints*. Since Gazebo is an integral part of the Player/Stage/Gazebo tool it can be controlled either directly, using the *Gazebo client library*, or over the Player middleware using the *Player client library*. In the latter case it is possible to seamlessly replace the Gazebo simulation with the real hardware without the need to write additional code (Fig. 1).

2.2. RPL

The MS controller is written in RPL developed by Drew McDermott on top of LISP and successfully deployed in the work of [6]. Through the in-house extensions, the dynamic typing of LISP is fully supported in RPL. RPL introduces state variables called *fluents* that reflect state changes in the world. It is then for instance possible to wait for the world to enter a specific state (wait−for fluent) or to specify how to act in this world state (whenever (fluent) reaction). Furthermore, RPL provides means for plan transformation. For each execution of a plan a *task net* is constructed that can be accessed and modified by RPL code in advance. Other important language features of RPL include constructs for concurrency control and failure handling.

2.3. RoLL

The Robot Learning Language RoLL [3] is an extension to RPL that augments it with capabilities for data acquisition, learning and seamless integration of learned functions into RPL control programs. The data collection is done with *experience automata* that monitor the execution of a control program and become active when the world resides in the specified state. The actual learning task is denoted in a *learning problem* definition where the programmer states how to transform the collected data. This learning problem is then fed into an instance of a *learning system*, that prepares the data to be conform with the learning algorithm's input specification. The learning system also provides routines to handle the output of the learning algorithm and to integrate it into RPL.

3. Learning

As already indicated in section 1, the production plan consists of four parallel tasks: refilling of bolt carriers (see also Fig. 4a), grasp learning of the mobile robot, assembly of playmazes and laser-based quality scanning. The mobile robot is strongly coupled to the playmaze production in that it has to perform the quality check of the finished playmaze at a provisionary scan station on the conveyor system.

The objective is to have the mobile robot and the finished playmaze arrive to the scan station simultaneously in order to avoid waiting times and thus delay in the whole production. There are two disruptions of the production caused by the quality check: The mobile robot has to interrupt its grasp learning task and the playmaze has to wait on the conveyor until scanned. When both, playmaze and mobile robot, meet at the same time, these two disturbances are minimal.

3.1. Learning Model

The waiting times are minimized by virtue of learning the function call_time (factory_state) that calculates the optimal time for calling the mobile robot based on the state of the MS at the beginning of the playmaze production. The factory_state representation is depicted in Fig. 4.

Given an independent character of the playmaze production and the robot grasp learning task, we split this complex learning problem into two sub-problems: maze_production_time and robot_coming_time. The first is the learning of the function that returns the time needed to finish the production of the playmaze depending on the MS state at the beginning of the playmaze production. Listing 3a shows how to calculate the maze_ready_for_scan_time at which the playmaze shall – preferably in sync with the mobile robot – arrive at the scan station.

The second sub-problem is the learning of robot_coming_time(robot_state) (listing 3b) which deals

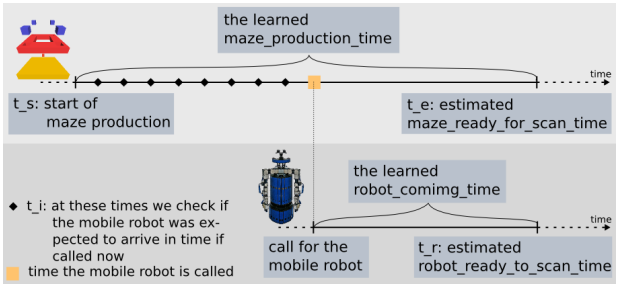


Figure 2: The controller that uses the learned time function runs in two threads: one for the playmaze production and one for the mobile robot.

```
maze_ready_for_scan_time =
    t_s + maze_production_time(factory_state)
```

(a) Estimated time the playmaze becomes ready to be scanned
t_s: start of playmaze production as in Fig. 2

```
robot_ready_to_scan_time =
      t_i + robot_coming_time(robot_state)
```

(b) Estimated time the mobile robot needs docks ready to scan
t_i: time of checking for coming time as in Fig. 2

```
maze_ready_for_scan_time =
    t_s + maze_production_time(factory_state)
do {
  sleep(1s)
  robot_ready_to_scan_time =
    t_i + robot_coming_time(robot_state)
} while (robot_ready_to_scan_time <
         maze_ready_for_scan_time)
call_robot()
```

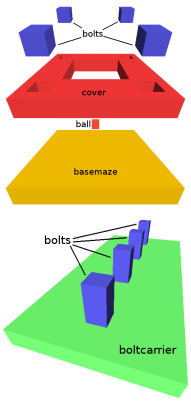
(c) Controller that calls the mobile robot at the learned time
t_s and t_i as in Fig. 2

Figure 3: These listings show the de-coupling of the main learning problem into two sub-learning-problems

with the time it takes for the mobile robot to arrive at the scan station depending on the state of the mobile robot.

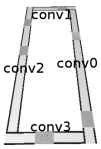
Once these functions have been learned the controller knows during the production of a playmaze (shown in listing 3c), when the playmaze is being expected at the scan station. In parallel to the playmaze production the controller constantly checks how long the mobile robot needs to arrive. See also Fig. 2.

factory_state: Since it is not possible to capture the MS state to the full extent (3d-positions of all entities, action commands of each actuator, state of the task net...), we opt out to use a very high level abstraction taking advantage of acting in the MS which is a highly structured environment. To best represent the MS state we therefore



1	nothing
2	empty-pallet
3	basemaze
4	basemaze-with-ball
5	basemaze-with-cover
6	basemaze-with-one-bolts
7	basemaze-with-two-bolts
8	basemaze-with-three-bolts
9	finished playmaze
10	cover
11	ball
12	bolt
13	empty-bolt-carrier
14	bolt-carrier-with-one-bolt
15	bolt-carrier-with-two-bolts
16	bolt-carrier-with-three-bolts
17	full-bolt-carrier

(a) Possible part variants as values of the location features



18	conv0
19	conv1
20	conv2
21	conv3

(b) The conveyor belts' IDs are the values of the carrier-location features

feature	values
store place 1-40	1-17
assembly station places 1-4	1-17
mill-and-turn station places 1-4	1-17
conveyor carrier playload 1-6	1-17
conveyor carrier location 1-6	18-21

(c) The features of the maze_production_time learning problem

Figure 4: The features of the maze_production_time learning problem along with their values making up the factory_state

propose to monitor the significant locations within the MS and their payloads in the course of playmaze production cycles. We coin them features of our learning problem and the payloads (values) come in form of part variants listed in Fig. 4a. Additionally, the carriers from the respective conveyor belt are added to the feature list with possible values (Fig. 4b) as well. Fig. 4 holds all the features and all possible values, i.e. MS state. We do not however capture the current task (and state of this task) of the stationary robots, since the position and state of the entities in our production implicitly contain this information. For

instance, knowing that there is a base−maze−with−ball at the assembly station, one can infer, that the assembly robot is in the process of assembling a playmaze and will in a next step place the cover on top. Altogether, the herein presented learning problem has 60 features with nominal values and one numerical target value (the production time).

robot_state: The mobile robot is not as predictable as the playmaze production since it can move without restraint around the MS. Furthermore, since the mobile robot is not idle but occupied with learning how to grasp, it might not be desired to interrupt it at any arbitrary time point but only allow for the quality check action to start at certain interruption points. Therefore, the mobile robot's state comprises following features: pose (three numerical values), current task (one nominal variable from Fig. 5) and the time it has spent on this task already (numerical). The target value for this problem is also (numerical) time.

1	Idle
2	Goto scan station
3	Wait for playmaze
4	Scan
5	Come from scan station
6	Goto pick up
7	Pick up
8	Goto place down
9	Place down

Figure 5: The tasks of the mobile robot that we record

3.2. Learning Algorithms

Currently there are two learning systems available to RoLL – one for neural networks provided by the SNNS package [12] and one for the decision, regression and model tree capabilities of the machine learning package WEKA [10]. Since these two require completely different parameters and return their results in different ways, their integration into RoLL is adapted to their respective needs.

We have applied neural networks and model tree learning to both learning sub-problems discussed in this work. A neural network learner needs to know the structure of the network, the initial weights and the routine that updates these weights. All these parameters can be set with RoLL code. To learn the maze_production_time (our first learning problem) we have chosen the network structure to be fully connected feed forward with 60 input neurons to handle our input parameters, one output neuron to present the result and 3 inner layers with 10, 5 and 2 neurons respectively. For the lack of space we do not include an image, please refer to [2] instead. The weights are initialized randomly (drawn from a uniform distribution between -1 and 1). As update routine we chose resilient back propagation which is a variant of back propagation that tries to eliminate strong fluctuations while updating the weights. The model trees are constructed using the *M5 prime* algorithm. For the learned trees, please refer to [3].

4. Empirical Results

We have trained the neural networks and built the model trees for the functions maze_production_time and robot_coming_time with 438 and 466 data sets respectively. During the data collection we called the mobile robot at random times.

In order to test the performance of our learned functions we simulated the production of 10 playmazes by executing the overall production process program in three-mode fashion:

- **mode 1**: using functions learned with neural networks
- **mode 2**: using functions learned with model trees
- **mode 3**: using functions based on the average of the collected data (heuristics)

For evaluation we captured the waiting times of both, the mobile robot as well as the finished playmaze. The results given in Fig. 7 show the average values from three runs of this experiment per each mode.

Correlation coefficient:	0.83
Mean absolute error:	3.68
Relative absolute error:	50 %

(a) Cross validation of the maze_production_time (average value about 81 s)

Correlation coefficient:	0.92
Mean absolute error:	3.86
Relative absolute error:	36 %

(b) Cross validation of the robot_coming_time (average value about 28 s)

Figure 6: Performance statistics of the learned functions

From the sum of the waiting times (Fig. 7c) one can see, that best results are achieved in mode 2 with model trees that outperform the heuristics and neural networks. Thus we in sequel discuss the results (Fig. 6) obtained in mode 2 run only.

We measured the performance on the training data using cross validation with 10 folds to assess their generalization. The resulting values show, that the learned functions represent the training data as the correlation coefficients are high. The Mean Absolute Error in both cases is within an acceptable range. The Relative Absolute Errors reveal that the Mean Absolute Errors are reduced to 50% and 36% resp. compared to the errors produced by predictions based on averaging of the actual values as in mode 3 of the test run.

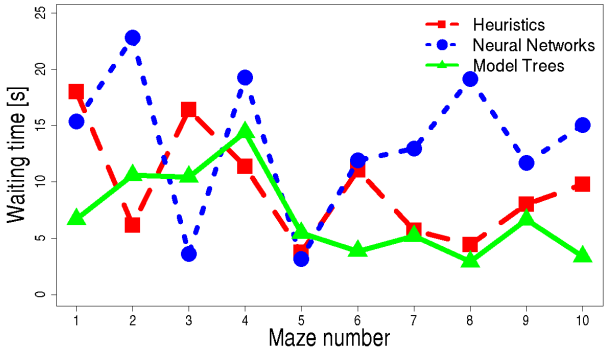
A qualitative comparison of the graphs 7a and 7b also depicts the mutual exclusion of the two tasks to reduce the waiting times for both the playmaze and the mobile robot: if the robot_waiting_time is high, the maze_waiting_time is low and vice versa. This behaviour is expected, because in case the robot arrives early and has to wait for the playmaze to be finished, the playmaze does not have to wait for the robot any more. If both waiting times show high values, this hints at a blocked conveyor e.g. due to rolling out parts from the store.

The results denoted in Fig. 7c display in the proof of concept manner the success and growing potential of our

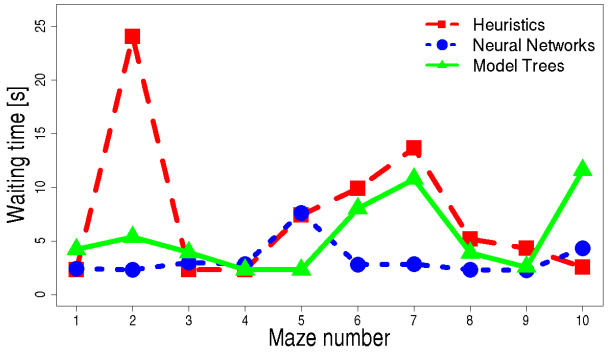
[2] http://www9.cs.tum.edu/research/cogmash/images/etfasnns.png

[3] http://www9.cs.tum.edu/research/cogmash/images/etfatrees.png

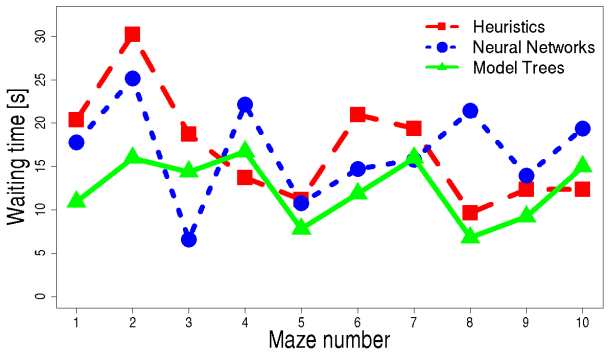
approach. By taking advantage of the ful RPL features we are fairly confident to thanks to learning capabilities be able to surpass state of the art planning algorithms [7].



(a) The waiting times of the mobile robot



(b) The waiting times of the finished playmaze



(c) The sum of the waiting times

Figure 7: Graphs showing the waiting times of the mobile robot and the playmaze respectively during the production of the first 10 playmazes

5. Future Work

These first experiments with learning optimal times in the simulated MS have proven promising but are yet to be improved in several ways:

First we will try to improve the setup that we have been using so far by fixing the erroneous data collection for high-storage unit slots. Mainly due to errors in our physics engine ODE the simulation aborted often prematurely. Thus, data collection for mazes assembled later in the production process was not possible. We foresee that porting to the latest versions of Gazebo and ODE shall overcome this issue.

Secondly, we will enlarge the corpus of learning algorithms by including Support Vector Machines and AdaBoost. Both are available in various free implementations and will be integrated to be used within RoLL.

In a next step we will enhance the production plan with, for instance, more meaningful task on the mobile robot. A hand-over of spare parts scenario with the assembly station has been already implemented and will be used also to further interweave the different tasks currently executed.

Finally we aim towards more complex learning tasks. One first step into this direction will be to add a next learning task that tackles the problem of whether or not to interrupt the mobile robots task at any arbitrary point. This will require to measure the disruption time of the mobile robots task as well. Directly connected to this is the introduction of priorities for different tasks (e.g. various product variants), which will further add implications to the function that is to be learned.

To qualify for the benchmark, the performance of this paper's approach will be compared to the classical planning [7] deployed in changing MS setups.

In the end we also aim to model our machining environment using ontologies [5]. We reckon on this representation, coupled with a reasoner of choice, to come to an alternative inference model helping us to design learning models automatically.

References

[1] M. Ghallab, E. Nationale, C. Aeronautiques, C. K. Isi, S. Penberthy, D. E. Smith, Y. Sun, and D. Weld. Pddl - the planning domain definition language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[2] O. Ilghami, D. S. Nau, and H. Muoz-Avila. Camel: Learning method preconditions for htn planning. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling*, pages 131–141. AAAI Press, 2002.

[3] A. Kirsch. *Integration of Programming and Learning in a Control Language for Autonomous Robots Performing Everyday Activities*. PhD thesis, Technische Universität München, 2008.

[4] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, 3:2149–2154 vol.3, 28 Sept.-2 Oct. 2004.

[5] S. Lemaignan, A. Siadat, J.-Y. Dantan, and A. Semenenko. Mason: A proposal for an ontology of manufacturing domain. pages 195–200, June 2006.

[6] T. Rühr, D. Pangercic, and M. Beetz. Structured Reactive Controllers and Transformational Planning for Manufacturing. In *Proceedings of the 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Hamburg, Germany, September 15-18*, 2008.

The research reported in this paper is partly funded by the German cluster of excellence CoTeSys (Cognition for Technical Systems).

[7] W. Ruml, M. B. Do, and M. P. J. Fromherz. On-line planning and scheduling for high-speed manufacturing. In *ICAPS*, pages 30–39, 2005.

[8] W. Shen, F. Maturana, and D. H. Norrie. Learning in agent-based manufacturing systems. In *Proceedings of the AI and Manufacturing Workshop, AAAI*, 1998.

[9] P. J. Sincebaugh. Applying a neuronal network based classification scheme to inspect manufactured assemblies. In *Proceedings of the AI and Manufacturing Research Planning Workshop, AAAI*, 1996.

[10] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.

[11] M. F. Zäh, M. Beetz, K. Shea, G. Reinhart, O. Stursberg, M. Ostgathe, C. Lau, C. Ertelt, D. Pangercic, T. Rühr, H. Ding, and T. Paschedag. An integrated approach to realize the cognitive machine shop. In *Proceedings of the 1st International Workshop on Cognition for Technical Systems, München, Germany, 6-8 October*, 2008.

[12] A. Zell. *SNNS User Manual*. University of Stuttgart and University of Tübingen, 1998.