INTEGRATION OF A PHYSICAL SYSTEM, MACHINE LEARNING, SIMULATION, VALIDATION AND CONTROL SYSTEMS TOWARDS SYMBIOTIC MODEL ENGINEERING

Sebastian Bohlmann
Institute of Applied Mathematics
Gottfried Wilhelm Leibniz University Hannover
Welfengarten 1
D-30167 Hannover, Germany
bohlmann@ifam.uni-hannover.de

Volkhard Klinger
Department of Embedded Systems and Information Engineering
FHDW Hannover
Freundallee 15
D-30173 Hannover, Germany
volkhard.klinger@fhdw.de

Helena Szczerbicka
Department of Simulation and Modelling
Gottfried Wilhelm Leibniz University Hannover
Welfengarten 1
D-30167 Hannover, Germany
hsz@sim.uni-hannover.de

ABSTRACT

System simulation without detailed prior knowledge or data of the system is a complex challenge. In this paper we present an approach to automatically generate a model on the fly in a symbiotic way. Basically the data based model generation system introduced is an agent based evolutionary optimization system creating continuous differential equations from simple predefined operators. The well known paradigm of symbiotic simulation is then enhanced with this agent based machine learning system. Here we focus on the emergent behavior of the model generation system resulting from the interaction of multiple agents optimizing a common model and the effects arising from the direct coupling and steering of the connected physical system. Different emergent mechanisms and effects can be observed speeding up the model generation process. To measure and evaluate these effects multiple experiments with a robotic system are discussed.

Keywords: symbiotic simulation, symbiotic circle, system identification, agent-based evolutionary computation, memetic optimization algorithms.
1 INTRODUCTION

To understand or control any type of complex processes a process model is essential, an empirical process
description does not provide a detailed functional and time related process specification. The process model,
based on the combination of physical equations and a graph structure, allows the reconstruction of process
behavior, the optimization of the entire process and a forecast of process behavior. This paper describes
a method to generate a process model from scratch, without using any type of initial model description.
The method is based on machine learning and symbiotic simulation (Aydt, Turner, Cai, and Low 2009) and
describes a type of symbiotic circle. The model generation is a continuous process and provides therefore
an adaptation to changing process parameters, like friction or bearing clearance, boundary conditions and
constraints.

Based on the classification of symbiotic simulation in (Aydt, Turner, Cai, and Low 2008), the symbiotic
circle combines a SSCS (Symbiotic simulation control system), SSFS (Symbiotic simulation forecasting
system), SSMVS (Symbiotic simulation model validation system) and a data driven agent based online
machine learning system with a real world process (Trianni 2014).

The central challenge in this paper is on the one hand to create a system which can produce a human
understand model without the existence of prior data, model or knowledge (as in (Schmidt and Lipson
2009)). And on the other hand to be able to simulate and/or predict the behavior of the system without
a known model. Both challenges presuppose each other (Yang, Koziel, and Leifsson 2013). First it is
basically a kind of chicken or the egg dilemma. The simulation system needs a model, this is produced by
the machine learning system, which needs some sort of input data, but this data is produced by the motion
of the robot, finally controlled by the simulator. The basic method we propose and demonstrate in this paper
is the symbiotic solution. If all steps are running in parallel the central dilemma disappears. This is what
we call the symbiotic cycle. All components of the self learning system are connected by using a streaming
event-driven approach. If an event which could cause an action in a different component is happening, it is
immediately streamed to the corresponding component. In fact there is no macroscopic sequence or stepping
between components. Everything is processed simultaneously in parallel. The machine learning module for
example continuously outputs model candidates at a relatively unknown rate. The simulator then reacts by
profiling the solution proposal. Clearly this could lead to an short-time overload of modules, if some burst
input is generated. To minimize this effect buffering and load balancing technology are used.

2 SYSTEM ARCHITECTURE

The system architecture follows the paradigm of the symbiotic circle, shown in figure 1. Five modules form
this symbiotic cycle which is application independent, described in the following subsections.

2.1 Process

The process block covers all physical relations of the considered process. Analog inputs or outputs have to
be transformed using analog-to-digital- or digital-to-analog-converters. The interface to the digital inout-
/outout-signals is handled using the Process Data Streaming Protocol (PDSP) managing distributed process
data flows (Bohmann, Klinger, and et al 2010). This protocol is designed to be used in mixed continuous
and discrete environments (Zeigler, Praehofer, and Kim 2000) referred to as hybrid. Focusing on symbiotic
simulation (Fujimoto, Lunceford, Page, and Uhrmacher 2002), PDSP is primary designed to satisfy four
modes of operation (analytic, transparent, online, prediction); here we focus on the online mode. In this
mode PDSP is used to simulate a process and transmit the results back to the process. The data is directly
transmitted between the physical process and the simulator. Therefore latency is minimized although proxy
servers may be necessary for large scale simulations.
PDSP should be classified as an application protocol (layer 7) in the Open System Interconnection (OSI) reference model. It encapsulates three inner layers primarily used to guarantee causality in data processing.

### 2.2 Simulator

The simulator block is a Java based simulation system specifically designed for online and symbiotic simulations. The simulator especially has online compiling capabilities, e.g. models can be compiled during runtime in memory and then dynamically injected to the simulator. It is capable to dynamically load or receive models (basically any kind of java program) and simulate multiple isolated instances in the same memory/thread context. The simulator combines Java classloading mechanism and byte code enhancing to calculate user defined metrics while processing prior structural unknown models on the fly. In combination with an OS GI framework PDSP can be directly embedded to running simulations.

### 2.3 Verification

The verification block confirms the quality of the model and forwards this information to the machine learning. The verification strategy is based on a set of process input and output sequences and of the simulation output sequences:

- **Input sequences**
  \[ (x_1)_t, \ldots, (x_m)_t, t \in \mathbb{N} \]
- **Simulator output sequences (simulated process model)**
  \[ (y_1)_t, \ldots, (y_j)_t, t \in \mathbb{N} \]
- **Process output sequences**
  \[ (z_1)_t, \ldots, (z_j)_t, t \in \mathbb{N} \]
The output sequences of the simulator are related to the input sequences by functional relationships \( f : \mathbb{R}^m \rightarrow \mathbb{R}^j \), illustrated in figure 2:

\[
\begin{align*}
 f_1 ((x_1)_t, \ldots, (x_m)_t) &= (y_1)_t, \ t \in \mathbb{N} \\
 \vdots & \ \\
 f_j ((x_1)_t, \ldots, (x_m)_t) &= (y_j)_t, \ t \in \mathbb{N}
\end{align*}
\]

2.4 Machine Learning

The machine learning is based on evolutionary algorithms embedded in a multi-stage and multi-agent implementation (Bohlmann, Klinger, and Szczerbicka 2010). Figure 3 shows the overall structure.

The planet structure represents the environment for the populations inside the evolutionary algorithm. Every planet provides a data field, the software agents can operate on. The number of planets is scalable, the current predetermined size is \( n = 9^4 = 6561 \). Using a multiprocessor system, the number of planets have to be multiplied by the number of cores. Data acquired from the process connection are preprocessed, equal to the filling of the planet structure. The preprocessing consists of several steps to guarantee a high average information content of the data, so called data entropy.

- **Resampling**
  The data produced in the Data Factory has two main weaknesses: The samples are asynchronous and aperiodic. In order to get a time series of data samples we perform the following steps:
  - Interpolation and FIR Filter (finite impulse response)
    For each sequence we interpolate the given values and smooth the result with a convolution.
  - Error Correction
    The interpolated data is equalized with the original samples gained from the Data Factory.
  - Downsampling
    We pick euclidean equidistant samples from each sequence and combine them to data samples with a timestamp.
During the machine process the data samples will not stay in their chronological ordering. To be able to perform time derivation, it is necessary to save the chronological neighbors for each sample. The resulting time series of equidistant data samples $p$ consists of a timestamp $p_{\text{time}}$, a vector $p_{\text{data}} = [p_{\text{out}}, p_{\text{in}}]$, with $p_{\text{out}} \in \mathbb{R}$ and $p_{\text{in}} \in \mathbb{R}^m$, containing the output and input data and its chronological neighbors $p_{\text{pre}}$ and $p_{\text{post}}$. With $P$ we denote the set of all such data samples. Furthermore we define $p_{\text{data}}^A \in \mathbb{R}^m$ with

$$
(p_{\text{in}}^A)_j := \frac{1}{2} \left( \frac{(p_{\text{in}})_j - (p_{\text{pre}}^\text{in})_j}{p_{\text{time}} - p_{\text{pre}}^\text{time}} + \frac{(p_{\text{in}})_j - (p_{\text{post}}^\text{in})_j}{p_{\text{time}} - p_{\text{post}}^\text{time}} \right),
$$

$$
(p_{\text{out}}^A)_j := \frac{1}{2} \left( \frac{p_{\text{in}} - p_{\text{pre}}^\text{in}}{p_{\text{time}} - p_{\text{pre}}^\text{time}} + \frac{p_{\text{in}} - p_{\text{post}}^\text{in}}{p_{\text{time}} - p_{\text{post}}^\text{time}} \right)
$$

$p_{\text{data}}$ as the result is then used for the machine learning module of the composed symbiotic system.

- **Data Prefilter**
  In general the amount of data delivered by the process is too large for our framework, that is providing planets of a predetermined size. To choose the samples, which should be passed to the planets. We are using a cluster algorithm to chose the samples. We have chosen k-means (Kanungo, Mount, Netanyahu, Piatko, Silverman, and Wu 2002) for two reasons. With k-means we are able to determine the number of clusters to be build in a set of data, i.e. the data rate. Moreover the clusters generated by this algorithm are formed spherical, what is more suitable for our purpose, compared to e.g. density based clusters. We subdivided the data from the data factory in blocks. In each of these blocks we build a fixed number of clusters. Only the centers of these clusters were passed on.

- **Data Filling**
  In the last step of the data preprocessing the data samples are arranged on a 2D surface of a so-called planet. The surface of the planets is built in a recursive pattern of squares containing nine elements, filled meander like. This method leads to the planet size $9^4$. This arrangement has the advantage, that the data set used for the local optimization consists of data samples, which may be spread more widely across the input sequences.

In the lower part of figure 3 the agent factory, the model library and the 2-step optimization, dedicated to every planet, is shown. The agents have 4 essential features: an age, an energy level, an area and their...
Bohlmann, Klinger, and Szczerebicka

model function, approximating the corresponding process function. Moreover a replication mechanism is implemented, meaning the agents are able to produce a child and put it on an area. The age and the energy level are increased after each iteration. All operations an agent can perform, have an energy effort, by which the energy level is lowered, if the operation is executed. Furthermore the agents have the ability to learn from their local data and improve their model function by executing different evolutionary operations to change the structure of the model function and a local optimization algorithm to calibrate the parameters. In each iteration the software agents perform the following operations:

- **Calculate Fitness**
  The individual evaluates the error of his model function with respect to a chosen metric. According to this error the energy level is recalibrated. If it is negative, the agent is removed from the planet and his child, if present, is put on his position.

- **Move**
  The agent moves to another area, meaning the local test data he uses is modified, so that he can use new data in the next iteration. If the agent carries a child it is set to the former area. Agents in a multi planet system can travel with a small probability to different planets.

- **Local Optimization**
  The model functions parameters are improved by trying to reduce the error of the current local data with respect to the chosen search metric.

- **Evolutionary Operation**
  One of the evolutionary operations, explained below, is performed.

- **Nomination**
  The agents elect a few individuals with the highest fitness values and age on each planet. Next the global fitness value of these agents is calculated. The 25 best agents form the so-called *Elite Population*, containing the best dissimilar agents. The model function of the elite agents are evaluated on the whole data set. If any of these functions has an error below a certain error bound, the algorithm terminates and this function is returned. Copies of these agents are then spread across all planets to distribute their information to other agents.

The agents can perform four different evolutionary operations to produce a child:

- **Mutation**
  The agents model function gets changed randomly: Either a subtree of the model function is exchanged or new operations are inserted.

- **Crossover**
  When an agent moves it may happen that the chosen area is already occupied with another individual. In this case, a subtree of the individuals model function is replaced by a randomly chosen, suitable subtree of the other agents model function.

- **Replication**
  The agent duplicates himself.

- **Global Optimization**
  The agents, which own enough energy or are not adult yet, optimize the parameters of their model function in the Memetic Coprocessor, explained below.

The global optimization is realized on memetic coprocessors, running on an extra processor core, executing more sophisticated algorithms for a global optimization. In the current configuration we use a downhill-simplex algorithm (Nelder and Mead 1965). The algorithm chosen for this local parameter optimization is resource-saving, because it is executed for all agents in every iteration.
The machine learning provides the capability of running the evolutionary algorithm described above on several planets at the same time. If this is the case, some of the areas on each planet get marked as so called beam areas. After each iteration copies of all individuals placed on such an area are send to a randomly chosen area on a randomly chosen planet, provided the chosen area is not yet occupied by an agent. In the experiments 100 of the 6561 areas on every planet were marked as beam areas. Our implementation associates each planet to one processor core, on an additional processor core a universe supervisor is executed. This supervisor manages the elite population using the data from all planets and controls the termination condition. The information exchange between the cores is implemented via a non-blocking Message Passing Interface.

2.5 Process Model

The area provides data samples to learn from and calculate the error of the model function \((\mathbb{R}^m \rightarrow \mathbb{R})\) is stored in a tree representation (figure 4). This function is composed of elementary operations (Schmidt and Lipson 2007), like \(+,-,*,/\), sin, sqrt, the variables \(x_1, \ldots, x_m\) and a set of parameters within their model function.

According the multi-agent capability, there are several models existing. The number of currently active agents and their inner candidate function is regulated by an software PID control. It is programmed to use the available processing power in an nearly optimal manner. The best model is picked according their ability to survive multiple times longer than the mean agent population. This metric corresponds to the selection of the fittest agents. The complexity of the process models is either dynamic. It depends on the number of input variables, constants and operations. The complexity is defined by counting the number of nodes in the tree representation.

3 APPLICATIONS

The symbiotic circle approach can be used for all applications with bandwidth-limited input signals. For example a process including boolean equations can not be identified using this method. Many technical or biological systems fulfill these criteria, for example electroneurogram (ENG)-based nerve signal identification or robotics.

In this paper we focus on the robotics application, evaluated in the results section.

4 EXPERIMENTAL DEMONSTRATOR SYSTEM

The demonstrator system is a parallel delta Robot (see figure 5). It basically has 6 vertical linear rails grouped into 3 pairs. These are placed around the working area in 120° spacings. Each linear axis is connected to a joint, which is connected to a arm and again to a joint. This joints are all linked to a central
platform. As common to all parallel kinematic chains a move in one actuator influences the position of the central platform in all possible degrees of freedom (DOF). This 6 linear actuators are the output stage of the simulation. To be able to detect the concrete position of the system a depth camera combined with a standard 2D camera is mounted above the working area inside the robot. The depth camera used is a infrared structured light sensor. In the experiment the camera follows a round target placed in the center of the platform. Camera output (after some processing) is the position of the platform. This is a 6 channel output: 3 spatial directions and 3 rotation angles. These 6 channels are the sensor output of the robot and the input for the simulation system. All 6 sensors and 6 actuators combined add up to the input for the machine learning part. This has been referenced in the previous chapter as $p_{data}$. The position of all actuators influence the position of all axis so the complete information set is required to be able to calculate a accurate prediction model. To create a symbiotic online learning environment the delta robots internal real time network is connected to a external computer. All data produced by the robotic system is streamed via PDSP (Process Data Streaming Protocol) to and from a central routing software. Machine learning and the massive parallel simulation are running outside the roots real time domain. The control simulation, which is using the currently best known model, is running inside an embedded system in real time. Communication between the two domains basically consists of a model transmission and a movement direction proposal directed into the real time domain. Data transmission directed to the machine learning, verification and simulation is a set of streamed time series data. The compute part of the system is located about 50km away from the robotic system an connected over the internet. All calculations outside the real time domain are processed on two quad Intel Xeon E7-8860v3 compute systems (128 logical cores) with 1TB of DDR4 RAM each. The real time control System located in the head box of the physical system consists of 14 Arm Cortex M4 Microcontrollers and three 8 Core Arm Cortex-A7 Processors. Local compute power is therefor relatively small compared with the cloud connected machine learning modules.
4.1 Feedback Loop

To realize the symbiotic machine learning system described in this paper one central question is: Where should the symbiotic robot acquire new data? The basic idea is to collect new experimental data at a location where the possible solutions generated by the machine learning system differ the most.

We define \( x_i \in \mathbb{R} \) as the input variables, \( p_j \in \mathbb{R} \) as the constant parameter and \( y_j \in \mathbb{R} \) as the output variables of a suggested model. The machine learning system produces a list of candidate functions \( f_{e,j}(x_0,...,x_n,p_0,...,p_m) = y_j \). \( e \) is defined as the solutions index (typically \( e < 30 = e_{\text{max}} \)).

\[
D_j = \{ f_{g,i} - f_{g,j} | i \neq j \text{ and } g,h = 0,...,e_{\text{max}} \} \tag{1}
\]

Now we can define the movement direction \( s \) for data acquisition as:

\[
s = \arg\max_{d \in D_j, j=0..6} \|\nabla d\|_1 \tag{2}
\]

This direction \( s \) is calculated by the simulator on the fly while processing new input data and transmitted to the robotic system. The simple idea behind this is to acquire new input data where model candidates tend to have different results. This leads to a continuous hypothesis filtering in the elite generation.

The basic challenge of the machine learning system can also be interpreted as a global non-linear optimization problem. The concept behind the above formulated feedback metric is to continuously adapt the target function, while machine learning is running. The adaption process described is on the other hand based on the output of the candidate functions \( f_{e,j}(x_0,...,x_n,p_0,...,p_m) = y_j \) evaluated by the online simulation system. This is again what we depict as the symbiotic cycle. If the global target is the deepest valley on the multidimensional target function, then the symbiotic circle will force secondary minima to increase their value by sampling new contradictory data from the physical system. The only valleys not affected are the global optima formed by different possible formulations of a globally valid and correct process model.

5 RESULTS

To demonstrate the emergent effects arising from the combination of all modules of the symbiotic circle we used the same machine learning system used in (Bohlmann, Klinger, and et al 2010). The basic complexity of the model detected in this former paper is around 15 nodes in the tree representation of the process model (see again figure 4). By utilizing better hardware and more compute power we could nowadays increase the detection of the described system to about 19 nodes with a success rate limit of 50%.

The first simple result is that the system needs a blank time at the beginning. Otherwise the first and only sample (start position) is solved immediately by using a simple constant. We set this blanking to 10s where no error estimation is calculated. In figure 6 and figure 7 error values and the complexity of the solutions are shown. We picked this two axis because the solution to be found is quite different. Actuator A (in the back of the robotic system) has a much simpler equation than the actuators in the front. Although this is a bit counterintuitive for a symmetric robot, this is correct. It arises form the fact that this actuators are aligned to the Y-Axis of the camera. Therefore the resulting equation does not include any coordinate transformations. The solution for this actuator is found in about 150s and has a complexity (sum of nodes in function tree) of 21 including 4 parameter. For Actuator B (similar to the non depicted Axis C) the task is a bit more challenging. Here the system requires about 800s to get to a 33 Node solution including 5 parameters. First thing that can be seen in all cases: It takes some time at the beginning to acquire enough input data.
to construct a complex solution. Secondly the error does not progressively decrease as often found in offline machine learning. This is caused by the feedback metric sampling new data at positions with low performance of some model candidate. This is a direct result from new online data injected to the machine learning progress. These spikes for the three actuators are depicted in figure 8 on the same time scale. The interesting aspect here is, that the peaks sometimes but in most cases not correspond to others. We found that this is a direct result from the $\text{argmax}$ operator in formula 2. It is basically caused by the selection of one pair out of all actuators to determine the robotic movement direction until a different pair has a higher rating.

By far the most amazing aspect is the comparison of the offline performance (Bohlmann, Klauke, Klinger, and Szczerbicka 2011) (Bohlmann, Klauke, Klinger, and Szczerbicka 2012) of the system and the online system interacting with the real world robot. While the offline machine learning system is only capable to identify models with a tree complexity of around 19 nodes with non good success rate, the online system identifies a model with complexity of 33 nodes. Furthermore as described in (Bohlmann, Klinger, and et al 2010) the machine learning modules offline performance decreases dramatically if additional noise is present in the input data. Obviously experiments described in this paper additionally and inherently contain
measuring noise through the use of a real physical system. The direct and automated online interaction of online simulation, verification, machine learning and physical process produce far better performance than each component alone could produce.

6 SUMMARY

In this paper we demonstrate a symbiotic combination of a robotic system, a machine learning and a simulation system. The symbiotic combination of these components has benefits for all three parts: a) The simulation is started without a model; b) the data driven machine learning does not need any samples before the system is started and c) the robotic system and its model predictive control system does not need a predefined model. An additional advantage is the adaptability of the symbiotic circle system: As it is continuously present, the robotic controller can react to changes in the system. The most valuable result in fact is that model complexity which could be learned form a real world process is clearly higher using a continuously interacting system.

REFERENCES


Figure 8: Error Band for actuators A,B,C.


AUTHOR BIOGRAPHIES

SEBASTIAN BOHLMANN is a Ph.D. candidate at Department of Simulation and Modelling - Institute of Systems Engineering at the Leibniz Universität Hannover. (currently working at the Institute of Applied Mathematics). He received a Dipl.-Ing. (FH) degree in mechatronics engineering from FHDW university of applied sciences. His research interests are machine learning and heuristic optimization algorithms, complex dynamic systems, control system synthesis and grid computing. His email address is bohlmann@ifam.uni-hannover.de.

VOLKWARD KLINGER has been a full time professor for embedded systems and computer science at the university of applied sciences FHDW in Hannover and Celle since 2002. After his academic studies at the RWTH Aachen he received his Ph.D. in Electrical Engineering from Technische Universität Hamburg-Harburg. He teaches courses in computer science, embedded systems, electrical engineering and ASIC/system design. His email address is Volkhard.Klinger@fhdw.de.

HELENA SZCZERBICKA is head of the Department of Simulation and Modelling - Institute of Systems Engineering at the Leibniz Universität Hannover. She received her Ph.D. in Engineering and her M.S in Applied Mathematics from the Warsaw University of Technology, Poland. She teaches courses in discrete-event simulation, modeling methodology, queuing theory, stochastic Petri Nets, distributed simulation, computer organization and computer architecture. Her email address is hsz@sim.uni-hannover.de.