Towards Lifelong Learning of Optimal Control for Kinematically Complex Robots*

Alexander Dettmann[1], Malte Langosz[2], Kai von Szadkowski[1], and Sebastian Bartsch[2]

***Abstract*— Robots intended to perform mobile manipulation in complex environments are commonly equipped with an extensive set of sensors and motors, creating a wide range of perception and interaction capabilities. However, to exploit all theoretically possible abilities of such systems, a control strategy is required that allows to determine and apply the best solution for a given task within an appropriate time frame. In this paper, a lifelong self-improving control scheme for kinematically complex robots is presented, which uses simulation-based behavior generation and optimization procedures to create a library of well-performing solutions for varying tasks and conditions, and combines it with case-based selection, evaluation, and online adaptation methods.**

I. INTRODUCTION

Behavior-based systems are best suited for changing environments, where fast response and adaptivity are crucial [1]. Their distributed nature increases fault tolerance and promotes component reuse as well as distributed development [2], [3]. However, coordination effort of behaviors increases with system complexity, making the control of behavior-based systems a challenging task. This is especially true for kinematically complex robots such as walking machines, whose different locomotion patterns, postures, or reflexes will be more or less efficient depending on the context, i.e., the external environment, the internal state, and the actual task. Current systems react to context changes by tuning parameters of their existing behaviors [4], [5], [6], but work on online integration of new behaviors for completely new contexts is rather sparse.

As future robots will be required to act more and more autonomously in well-known as well as in novel environments, it is important to equip robotic systems with tools to efficiently adapt to unknown contexts. For mobile platforms it is thus desired to use machine learning algorithms and/or simulation methods to create new sets of behaviors for situations in which none of the predefined locomotion behaviors is well suited. Such newly-derived behaviors will then have to be directly included in the running robot control to be utilized on the spot.

*The presented work was carried out in the project LIMES, a collaboration between the DFKI Robotics Innovation Center and the University of Bremen, funded by the German Space Agency (DLR, Grant numbers: 50RA1218, 50RA1219) with federal funds of the Federal Ministry of Economics and Technology (BMWi) in accordance with the parliamentary resolution of the German Parliament.

[1]Alexander Dettmann and Kai von Szadkowski are with the Faculty of Mathematics and Computer Science, University of Bremen, 28359 Bremen, Germany firstname.lastname@uni-bremen.de

[2]Sebastian Bartsch and Malte Langosz are with the German Research Center for Artificial Intelligence - Robotics Innovation Center (DFKI RIC), 28359 Bremen, Germany firstname.lastname@dfki.de

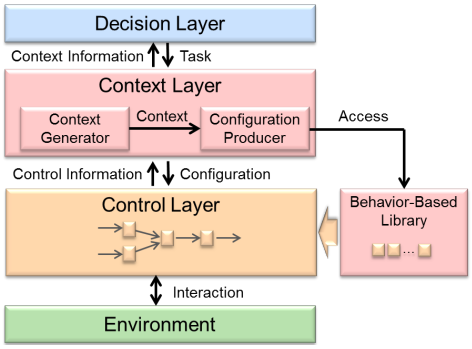


Fig. 1. Context-based control approach

Commonly a *decision layer* is used on top of a *control layer*, with the former generating tasks derived from deliberative planning processes and the latter executing these by producing appropriate target values for the actuators. This structure leaves a gap, as higher deliberative planning algorithms cannot take all configuration possibilities of the control layer into consideration. Consequently, the full potential of a robot is not used or an operator has to be included in the loop to tune the system according to the changing contexts in which the robot finds itself in.

As a solution for these problems, an intermediate layer is proposed which configures the control layer autonomously according to the inputs from higher and lower layers (Fig. 1). This *context layer* receives commands specifying the desired task from the decision layer, e.g., “move forward at a certain speed and as stable as possible”, as well as data defining the current state from the control layer, e.g., “hard ground with small scattered obstacles”, and builds up a *context* representation. Based on this *context*, the best-known configuration of the control layer is retrieved from a *behavior-based library* (BBL) and applied on the robot. This encapsulation of the control layer provides a robot-independent interface to higher levels, allowing a far more abstracted design of the deliberative control layer.

Section II describes the BBL and its components followed by a description of how optimization in simulation can generate additional competence (Section III). The selection principle to choose the best configuration of the control layer according to the current context and the data available in the BBL is described in Section IV. SectionV details a simple example for how the proposed approach can be used to improve the performance of the robot by integrating externally generated knowledge. In the last section, a brief conclusion and an outlook are provided.

II. BEHAVIOR-BASED LIBRARY

The BBL basically represents the memory of a robot. On the one hand, it consists of solutions in form of algorithms and parameterizations, which both together result in *configurations* of the control layer. On the other hand, it holds all information required for proper selection or even creation of new solutions, i.e. performance evaluations of configurations in diverse contexts.

The BBL can easily be used in conjunction with robotic frameworks such as ROS[1] and Rock[2], and thus its functionality can be integrated in a large number of diverse robotic systems.

A. Behavior Representation

In reactive control approaches, the overall robot behavior emerges from the interaction of several behavior producing modules, simply called *behaviors*. In the proposed approach two behavior types are available, *graph behaviors* and *parameter behaviors*.

Graph behaviors are defined as nodes, as described in [7], mapping a defined number of input ports to a defined number of output ports (Fig. 2). This mapping is realized via a graph of behavior producing modules, which are either atomic transfer functions or themselves graph behaviors. Since both are using the same interfaces, there is no need for special handling in the different layers, thus allowing a hierarchical decomposition of behaviors. Possible atomic transfer functions include direct mapping of input to output, various mathematical such as trigonometric functions or conditional branching.

The signals used in these graphs are tuples of values and accompanying weights, allowing flexible interactions of the modules. Both outputs and inputs may be connected to multiple other ports, however in the latter case, one of various available merge functions is used to calculate a resulting input value for each input port. If an input is not connected a default value is used instead.

Behavior modules and their individual input and output ports can be named and annotated with additional meta information in the form of textual descriptions. This allows to attach a description or information on a behavior's suitability for a certain scenario, providing higher control layers or a human operator with necessary information to select an appropriate behavior for a given context. Annotations for ports can take the form of units and type information as well as expected input and output ranges. The latter allow the usage of interval analysis [8] for consistency checking.

The flexibility of this design and the common interface of graph behaviors, no matter how complex internally, opens many possibilities. For instance, any algorithm for which numerical inputs and outputs can be specified can be wrapped in a graph behavior module and thus be integrated in the overall behavior of a robot on any level of the graph hierarchy. Tools for designing new and altering existing behaviors

[1] http://www.ros.org
[2] http://rock-robotics.org

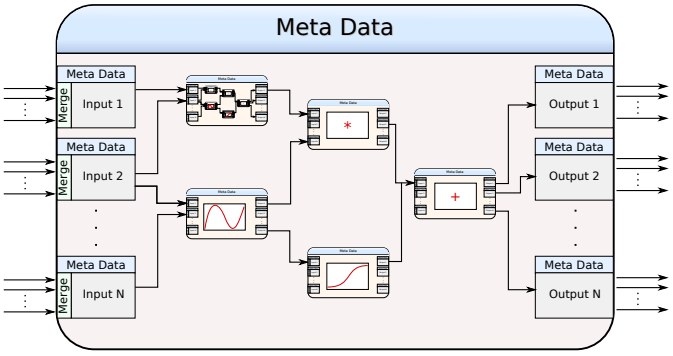


Fig. 2. Graph behavior consisting of a network of atomic transfer functions and another graph behavior

```
nodes:
  - id: 1
    type: "[PIPE]"
    inputs:
      idx: 0
      type: "[PRODUCT]"
      default: 1.0
    outputs:
      idx: 0
  - id: 6
    type: "[OUTPUT]"
    inputs:
      idx: 0
      type: "[SUM]"
      default: 1.0
    outputs:
      idx: 0

networkInputIds:
  - 1

edges:
  - fromNodeId: 1
    fromNodeOutputIdx: 0
    toNodeId: 6
    toNodeInputIdx: 0
    weight: 2.5
```

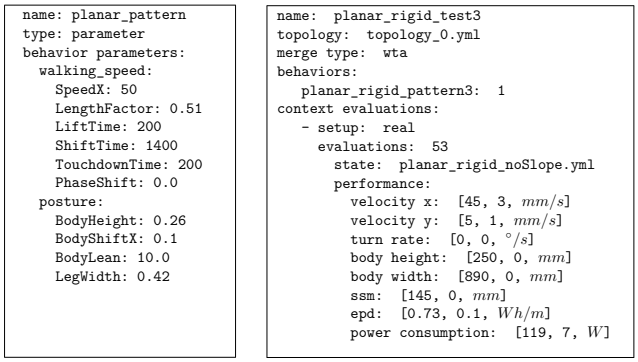
Fig. 3. An example of a graph behavior in YAML (www.yaml.org) format is shown. First, two nodes, one pipe and one output node are defined, followed by the declaration of the network's input nodes and the definition of an edge from the input node to the output node.

can be built with little effort. One resulting advantage is the possible application of machine learning algorithms to change or optimize an algorithm by modifying the structure and parameters of the corresponding behavior graph; similar work is successfully done in the field of neuroevolution or genetic algorithms [9], [10], [11]. The modular structure simplifies testing, as a generic test suite can be used to validate that a behavior operates in a given output range and does not contain singularities such as divisions by zero. Finally, due to the annotation with meta information, even complex graph behavior can be saved in human readable files such as shown in Fig. 3.

The second concept of behaviors used in the BBL is that of parameter behaviors which parameterize the algorithmic graph behaviors, significantly influencing the emergent robot behavior; e.g., the same walking pattern generator can be switched from generating patterns for obstacle covered slopes to patterns suitable for plain soft soil simply by assigning different parameters. A parameter behavior can be imagined as a behavior module having no inputs but outputs with default values or as a parameter list as depicted in Fig. 4(a).

B. Context-Based Behavior Evaluation

To be able to use the best-suited behaviors in the right situation, their performance in diverse contexts needs to be



(a) Parameter behavior (b) Configuration evaluation

Fig. 4. Example descriptions of components of the BBL

known. Therefore, this information is stored in the BBL as well (Fig. 4(b)). *Configuration evaluations* can be generated from experiments with the real robot or derived from optimization results in simulation. They hold information about:

- the overall behavior graph (topology of behaviors)
- applied parameter behaviors and their merge method
- evaluation information for each encountered context
  - setup information (useful to filter out undesired experiences)
  - number of evaluations
  - all state context features
  - mean and standard deviation of all performance features

III. SIMULATION-BASED BEHAVIOR GENERATION AND OPTIMIZATION

The efficacy of choosing context-based behaviors strongly depends on the quality of the underlying behavior database. This refers to both its extent, i.e. the number of contexts for which a suitable behavior is available, and its elaborateness, i.e. how well-tested stored behaviors are and how reliably they perform. As it is virtually impossible to optimize behaviors over a large number of experimental cycles on the actual robotic system, the BBL is built not only by the robot itself, but is also fed with simulation results. For this, the open-source physical simulation environment MARS[3] is utilized which is based on the Open Dynamics Engine (ODE)[4]. Using MARS together with an integrated framework for behavior learning, testing of vast sets of possible behavioral solutions to problems posed by various contexts and refinement of the resulting behaviors with sophisticated optimization algorithms is made possible within reasonable constraints of time and resources. This is further assisted by the use of the meta information provided by the behaviors being optimized, which allows to restrict the search space by pre-defining dependencies, parameter ranges and other properties of the respective modules.

[3]MARS (Machina Arte Robotum Simulans) is available on *https://gitorious.org/rock-simulation/mars*

[4]*http://www.ode.org/*

In order to yield useful results, the simulation model of the robot model as well as the characteristics of the simulated environments have to be sufficiently accurate representations of their real-world counterparts. This necessitates measurements of the robot's single components' physical behavior as well as including environmental factors such as soil dynamics in the simulation. Still in most cases, behaviors evolved in simulation will have to be adapted to be used on real robots due to the remaining simulation-reality gap. Even given these difficulties, developing behaviors in simulation still constitutes a dramatically reduced effort when new strategies are required in novel or changed contexts as compared to developing said strategies on the real system from scratch. Moreover, simulation allows to discard erroneous or unfit parameter sets that might result in malfunction or damage of the robot, providing a mechanism of safety-relevant quality control. This is true for both offline-development of novel behaviors in simulation as well as for online-testing of newly-derived behaviours in the current context of a robot, an approach becoming more and more feasible with the constant increase in processing power.

IV. ONLINE BEHAVIOR SELECTION AND ADAPTATION

Kinematically complex robots in real world scenarios have a tremendous amount of possibilities to solve certain tasks. Consequently, especially at the beginning of the lifetime of a robot, only sparse domain knowledge and anecdotal experience are available. Thus, case-based reasoning (CBR) is used to infer the best-suited configuration of the control layer for a certain context. This artificial intelligence paradigm solves problems by reusing experiences from similar, previously solved problems [12]. A *case* consists of two parts. The first part represents a *solution*, which is in the proposed application a configuration of the control layer. The second part holds a list of E performed configuration evaluations, where each entry describes the number of evaluations, the evaluation setup, the state context, and the evaluated performance.

The BBL will contain several case bases which are used in different scenarios, e.g., a six-legged locomotion scenario and a manipulation scenario have different case bases. This refines the case retrieval step described in the following section. In addition, it can be beneficial to introduce new case bases when something unpredictable occurs (as proposed in [13]), e.g., a malfunction of a leg. Then, a corresponding five-legged locomotion case base could be created with some initial cases from the six-legged case base as a starting point. These cases could then be adapted and optimized to fit the current conditions. The BBL has the opportunity to load, use, and store cases or switch entire case-bases according to the current task.

In the proposed application of adaptive robot control, the input problem is described by two feature vectors describing a robot's current state $\mathbf{S}^{cur}$ with M state features $(s_1^{cur}, ..., s_M^{cur})$ and task $\mathbf{T}^{cur}$ with N performance features $(p_1^{cur}, ..., p_N^{cur})$, where each feature is a tuple consisting

of value and weight. This format matches the description of configuration evaluations in the BBL, with the notable difference that the library contains mean and standard deviation values for the performance features p_n^{ref}. To decide which behavior configuration to deploy, the CBR algorithm processes this input in multiple steps (Fig. 5) which are very common for many CBR systems and are motivated from [12]. First, the similarity of the provided context to previously tested cases is evaluated, resulting in a number of candidate behaviors. Then, a *ballpark solution* for the given problem is derived from this set and adapted if necessary according to the reigning conditions. The resulting behavior configuration is checked one last time before execution to avoid mistakes from the past. During application, the performance is evaluated. Finally, the gained experiences are stored in the corresponding case base. The algorithmic details are described in the following subsections.

A. Case Retrieval

In the first step of CBR, all configuration evaluations in the BBL are rated according to the input query. First, the state and performance features are normalized according to robot-specific limits, before a similarity measure is used to determine how well a case from the case base matches the input query. Since the input query consists of two feature vectors of variable length, a multi-stage approach is proposed: First, the state similarity Sim_e^{State} between the current state features s_m^{cur} and stored reference state features s_m^{ref} is calculated for each evaluation of each case, then the task similarity Sim^{Task} is computed for the library entry with the highest state similarity, and finally the overall case similarity Sim is calculated.

In the first step, the weighted mean square error is used, as it is more sensitive to large differences of one single feature than to small differences of several features compared to the weighted mean absolute error. The error is subtracted from one (since normalized values are used) to get the state similarity for each evaluation $e \in E$ (1).

$$Sim_e^{State} = 1 - \frac{\sum_{m=1}^{M}(s_m^{cur} - s_m^{ref})^2 \cdot w_m^S}{\sum_{m=1}^{M} w_m^S} \tag{1}$$

The weights for each feature variable w_m^S are used to include the confidence of the corresponding context feature estimation. Alternatively, they could be used to model the features' importance, which can be learned to improve the case retrieval [14]. If a state similarity has to be calculated for a current state feature not listed in the evaluation, two solutions are possible. The safest way is to set the corresponding feature state similarity to zero. A more curious strategy would be to set it to one. Finally, the state similarity for the entire case Sim^{State} is represented by the maximum state similarity of the E evaluations (2).

$$Sim^{State} = max(Sim_1^{State}, ..., Sim_E^{State}) \tag{2}$$

The task similarity is calculated for the evaluation with the highest state similarity, also using the weighted mean square error (3),

$$Sim^{Task} = 1 - \frac{\sum_{n=1}^{N}(p_n^{cur} - p_n^{ref})^2 \cdot w_n^P}{\sum_{n=1}^{N} w_n^P} \tag{3}$$

where w_n^P are the weights of a performance ratio, which the operator or higher layers can define to influence the robot behavior. Therefore, the vector of performance features p_n^{cur} contains the task-depending features (desired motion, ...) and the evaluation features (stability, energy efficiency, ...). The latter stay constant at their best value, e.g., the best stability value would be one whereas the power consumption would be zero. Finally, the overall similarity Sim of a case is the product of both single similarities (4).

$$Sim = Sim^{State} \cdot Sim^{Task} \tag{4}$$

After determining the similarity of each case, the k nearest neighbors (K-NN) are chosen, yielding a limited set of candidates for the next processing stage. The K-NN are limited by number and also have to reach a pre-defined relative similarity threshold.

B. Ballpark Solution Proposal

In this step, a case solution or parts of several solutions are extracted to form a temporary solution to be used in subsequent processing stages. A number of methods are available for this, including choosing the most similar case [15], drawing randomly from K-NN [16], or drawing from K-NN according to the degree of similarity [17]. While the first method simply selects the best known solution, the second and third method avoid overusing one particular solution, which might be beneficial in situations where the solution with the highest similarity does not necessarily result in the best performance. Deriving the average of K-NN solutions is also common in case-based regression. Beyond these methods, other merging techniques such as the application of machine learning to explore new solutions can be imagined as well.

C. Case Adaptation

Once a ballpark solution has been obtained, it is adapted to fit the current needs. Here, two adaptation variants exist: constant adaptation and continuous adaptation. Constant adaptation can be applied to vary the ballpark solution by taking the dissimilarity between input query and ballpark solution into consideration to enhance the expected performance. Some rule-based approaches exist which use deep domain knowledge to infer adaptation rules. However, this leads to losing the generality of the overall approach. Instead, so-called "knowledge-light" approaches use the implicit knowledge of the case base to infer adaptations. For instance, case difference heuristic approaches build adaptation rules by comparing pairs of cases and identifying their context and solution differences. The resulting mappings between incoming context difference and resulting solution adaptation are then scored according to some gradient [18], [19] or covariance metrics [20].

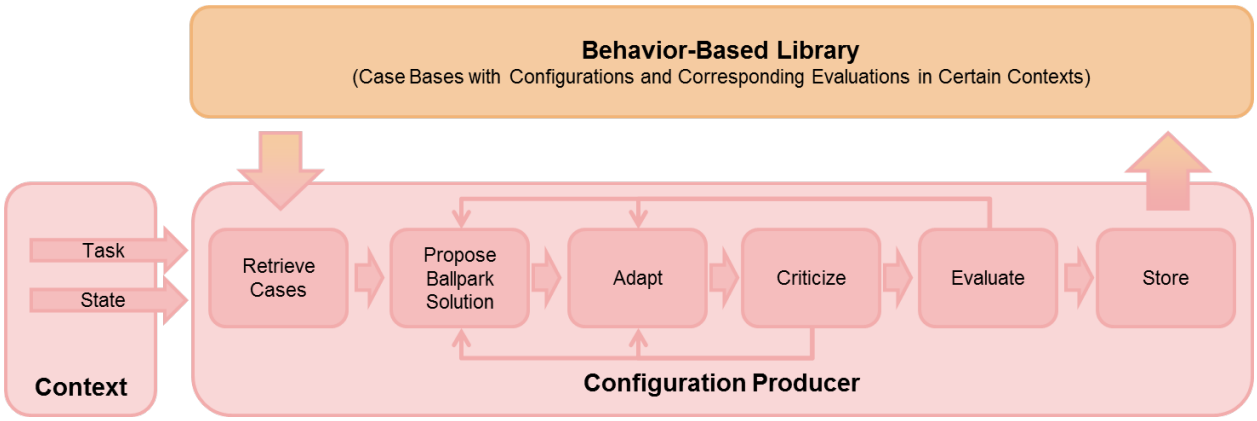


Fig. 5. Processing steps of the case-based reasoning system

Continuously adapting algorithms on the other hand can be applied in environments with low fluctuations to find local performance maxima. Simply adding noise [15], [16] or crouching and inverting a randomly initialized adaptation vector [17] can lead to increasing system performance. Some more sophisticated machine learning approaches such as CMA-ES [21], REPS [22] or PSO [23] could certainly improve the results. In addition, they could be used to handle the simulation-reality gap. Imagine a behavior configuration evolved through optimization in simulation is chosen as a ballpark solution and finally applied. Because of the simulation-reality gap, the performance will most certainly differ, but probably still be close to the actual (local) optimum. Thus, a lazy, fast-converging learning algorithm could most likely be used to adapt the simulation solution to reality. The required feedback would in this case be generated in the evaluation step.

D. Case Criticism

Before the adapted solution is applied on the system, it is *criticized*. In this processing step, solutions can be rejected which have already been chosen and tested before but did not perform well, thus avoiding known failures. If this happens, a new solution has to be provided, repeating the algorithm's previous subroutines. This step is sparsely used in literature. In [17] a case switching tree is used to recover from overused cases which do not improve the situation. Here, it is also advantageous to predict the performance of the chosen solution. This information can help in later steps to analyze the outcome of applying the generated solution.

E. Evaluation

The evaluation of a solution itself is separated from the CBR algorithm since computation of the performance features is robot-specific. The resulting performance vector only has to match the case description. As mentioned before, the results of the evaluation are needed in the other processing steps. Useful performance metrics are energy efficiency, stability, precision of task realization, processing time, or other control-specific information.

F. Memory Storage

The last stage of a CBR system is the memory update, which incorporates updating performance values of known cases (configuration evaluations) or creating new cases (configurations and their evaluations) if a new solution was applied. Through gathering of new experiences, the robot gets the opportunity to learn, i.e. increasing its performance and competence. In addition, constantly updating the case base incorporates wear out of the system.

V. PROOF OF CONCEPT

In order to show the possibility to include externally generated knowledge in a robot control, the following experiment was conducted in simulation and reality. The SpaceClimber [24] robot was set up to walk on a plane for 30 s as energy-efficient as possible at a given speed of 50 mm/s. In a first step, SpaceClimber's BBL consisted of one configuration evaluation holding performance information of one graph behavior in combination with one parameter behavior. The latter (Table I) was created with expert knowledge and has shown good results in previous experiments [24]. During context-dependent configuration of the control layer, this solely available configuration evaluation was of course most similar to the experimental context, which led to the application of the corresponding graph and parameter behavior.

In the second step of the experiment, a new parameter behavior was created in simulation, using a CMA-ES optimization aiming for energy efficiency of the previously-tested graph behavior in the same context of walking on plane ground. The resulting parameter behavior (Table I) was stored in the BBL along with the corresponding configuration evaluation (performance based on optimization fitness). When repeating the 30 s walk, the optimized walking pattern was selected by the configuration producer and applied on the robot control because of its better performance in the same context.

The resulting power consumption in both scenarios, simulation and reality, are depicted in Fig. 6. It is visible that the power consumption in simulation is higher than on the real system indicating a gap between simulation and reality.

TABLE I

PARAMETERS OF USED WALKING PATTERNS (ALL OTHER PARAMETERS ARE KEPT AT THEIR DEFAULT VALUE AND ARE OMITTED FOR CLARITY)

pattern	plane, handcrafted	plane, optimized
body shift x in mm	0	100
body height in mm	250	275
speed x in mm/s	50	50
length factor [0...1]	0.5	0.5
lift time in ms	200	240
shift time in ms	1400	1560
touchdown time in ms	200	1600
phase shift [0...1]	0.0	0.0
swing amplitude in mm	100	100

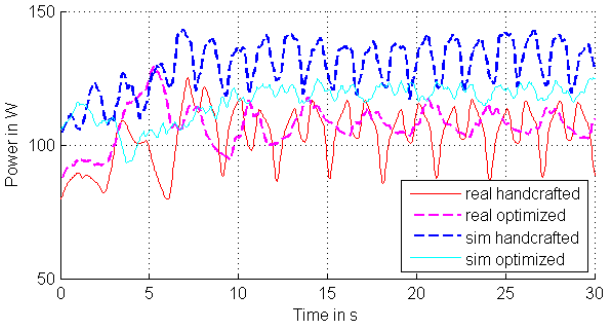


Fig. 6. Power consumption while walking

While in simulation, the power consumption was lower with the optimized locomotion pattern, this was not the case with the real system. However, the optimized walking pattern traversed 1,38 m during the 30 s while the handcrafted traversed 1,23 m. Consequently, the resulting energy per distance was lower for the optimized locomotion pattern. The main reason is that with the optimized pattern the feet are placed more smoothly due to higher touchdown time resulting in less slippage. Although the movement was improved, the walking behavior on the real system was not optimal since the power consumption was not less as indicated from the simulation comparison.

VI. CONCLUSIONS AND OUTLOOK

In this paper, a control scheme for kinematically complex robots is presented, which uses a BBL to store possible control configurations and their performances in varying contexts. Real world experiences and optimization results build the knowledge base which is continuously growing during life-time increasing the robot's performance and competence. A case-based reasoner is used to find the best-known control configuration for a given context. The given example of increasing the energy efficiency of a walking pattern is a rather simple problem. In future, the scalability of this approach for complex problems have to be discussed. In addition, the generation of new solutions through intelligent case merging and adaptation as well as the storage of experiences need to be analyzed. Though, in simulation optimized behaviors can improve the performance of the real system, an online optimization on the real system will be needed to handle the simulation reality gap.

ACKNOWLEDGMENT

Special thanks are due to all team members of the project LIMES.

REFERENCES

[1] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2008.
[2] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986.
[3] R. Arkin, *Behavior-based robotics*. MIT press, 1998.
[4] J. Albiez, *Verhaltensnetzwerke zur adaptiven Steuerung biologisch motivierter Laufmaschinen*. GCA-Verlag, 2007.
[5] B. Gassmann, "Modellbasierte, sensorgestützte navigation von laufmaschinen im gelände," Ph.D. dissertation, University Karlsruhe (TH), 2007.
[6] F. Michaud, "Selecting behaviors using fuzzy logic," in *Proceedings of the Sixth IEEE International Conference on Fuzzy Systems*. IEEE, 1997, pp. 585–592.
[7] M. Langosz, L. Quack, A. Dettmann, S. Bartsch, and F. Kirchner, "A behavior-based library for locomotion control of kinematically complex robots," *Proceedings of the 16th International Conference on Climbing and Walking Robots, (CLAWAR-2013)*, pp. 495–502, Aug. 2013.
[8] R. E. Moore, *Interval Analysis*. Prentice Hall, 1966.
[9] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Learning to drive in the open racing car simulator using online neuroevolution." *IEEE Trans. Comput. Intellig. and AI in Games*, pp. 176–190, 2010.
[10] Y. Kassahun, J. de Gea Fernández, M. Römmermann, and F. Kirchner, "On applying neuroevolutionary methods to complex robotic tasks," in *IEEE IROS Workshops on Exploring new horizons in Evolutionary Design of robots*, 2009, pp. 26–30.
[11] M. Römmermann, M. Ahmed, L. Quack, and Y. Kassahun, "Modeling of leg soil interaction using genetic algorithms," in *Proceedings of International Conference of the International Society for Terrain-Vehicle Systems*, 2011.
[12] J. L. Kolodner, "An introduction to case-based reasoning," *Artificial Intelligence Review*, vol. 6, no. 1, pp. 3–34, 1992.
[13] D. Leake and R. Sooriamurthi, "When two case bases are better than one: Exploiting multiple case bases," *Case-Based Reasoning Research and Development*, pp. 321–335, 2001.
[14] S. Gunawardena, R. Weber, and J. Stoyanovich, "Learning feature weights from positive cases," *Case-Based Reasoning Research and Development*, pp. 134–148, 2013.
[15] A. Ram and R. Arkin, "Case-based reactive navigation: a method for on-line selection and adaptation of reactive robotic control parameters," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 27, no. 3, pp. 376–394, 1997.
[16] M. Likhachev and R. Arkin, "Spatio-temporal case-based reasoning for behavioral selection," *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1627–1634, 2001.
[17] M. Likhachev, M. Kaess, and R. Arkin, "Learning behavioral parameterization using spatio-temporal case-based reasoning," *IEEE International Conference on Robotics and Automation*, 2002.
[18] N. McDonnell and P. Cunningham, *A knowledge-light approach to regression using case-based reasoning*. Springer Berlin Heidelberg, 2006, vol. 4106.
[19] V. Jalali and D. Leake, "A context-aware approach to selecting adaptations for case-based reasoning," *Modeling and Using Context*, pp. 101–114, 2013.
[20] ——, "Extending case adaptation with automatically-generated ensembles of adaptation rules," in *Case-Based Reasoning Research and Development*. Springer, 2013, pp. 188–202.
[21] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, pp. 159–195, 2001.
[22] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, 2013.
[23] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proceedings of ICNN'95 - International Conference on Neural Networks*, 1995.
[24] S. Bartsch, "Development , control , and empirical evaluation of the six-legged robot spaceclimber designed for extraterrestrial crater exploration," Ph.D. dissertation, University of Bremen, 2013.