Real-Time Online Adaptive Feedforward Velocity Control for Unmanned Ground Vehicles

Nicolai Ommer, Alexander Stumpf, and Oskar von Stryk

Department of Computer Science, TU Darmstadt, Germany

Abstract. Online adaptation of motion models enables autonomous robots to move more accurate in case of unknown disturbances. This paper proposes a new adaptive compensation feedforward controller capable of online learning a compensation motion model without any prior knowledge to counteract non-modeled disturbance such as slippage or hardware malfunctions. The controller is able to prevent motion errors a priori and is well suited for real hardware due to high adaptation rate. It can be used in conjunction with any motion model as only motion errors are compensated. A simple interface enables quick deployment of other robot systems as demonstrated in Small Size and Rescue Robot RoboCup leagues.

1 Introduction

Precise control in an unknown, unstructured or dynamic environment is a crucial ability for autonomous robots to move safely among obstacles. Here, navigation, path planning, and controllers rely on a reliable motion model that cannot consider detailed knowledge about unknown external disturbances such as friction, slippage or even hardware malfunctions. These unmodeled disturbances may be counteracted by appropriate feedback controllers but will always decrease locomotion performance of the robot system as any deviation from a planned path requires continuous correction movements or in case of nonholonomic systems such as tracked robots even trajectory replanning.

The used motion model as well as the controllers are often derived directly from robot structure but can neither automatically adapt to changes in structure nor in the environment. In order to overcome such problems, model based agents have become popular in the fields of robotics which can infer characteristics of its structure and the surrounding world.

On the other side, learning approaches often require an unreasonable amount of training data in order to generalize well. In this work we have investigated a way to benefit from both worlds: instead of learning the entire motion model we figure out how to fix the performance errors of the existing model.

This motivates our contribution of an adaptive compensation feedforward controller which is capable of adapting online to motion errors caused by unknown disturbances. For this purpose, multiple learning approaches have been evaluated in terms of applicability, real-time performance, motion precision and motion error adaptation time. While learning approaches have been widely investigated in motor and multi-joint trajectory control, the application to locomotion trajectory control has not received much attention yet to the best of the author's knowledge.

The proposed controller has been evaluated with two robot platforms with distinct motion capabilities used in different RoboCup leagues: Small Size and

Robot Rescue League. The adaptation is performed without any prior knowledge of any underlying motion model which demonstrates that the method can be applied to a wide range of systems.

2 Related Work

The RoboCup Small Size League (SSL) has to tackle with large robot control delays with at least 100ms due to off-board processing. This problem can be bypassed by capable feedforward controllers, reducing all feedback error terms significantly. An early attempt was done by Gloye et al. using neuronal networks to predict the behavior of the robots based on the current state and motion command to overcome the system delay [1]. Based on the first trained neural network a second network has been trained to predict appropriate motion commands leading into the desired robot behavior. They demonstrated how the neural network is able to learn a model compensating for a disconnected motor. However, these networks can only be trained offline and require uniformly distributed data sets to cover the whole function space.

In order to use neural networks in adaptive control tasks, the concept of Feedback Error Learning has been applied in [2]. The online trained artificial neural network using an expanded version of the traditional back-propagation algorithm outperformed a standalone PD controller applied to a SCARA robot arm but the learning rate has to be adapted manually during training. Although the neural network compensated general deviations, the feedback controller was still active to capture irregularities.

The application of neural networks needs a lot of manual structural tuning such as determining the optimal number of layers and neurons. A more convenient class of learning approaches are function approximators which can directly infer from data. Locally Weighted Learning (LWL) categorizes a class of non-linear function learners based on locally linear approximations. This type of function approximation can be even used to learn forward or inverse robot models [3]. In general, LWL methods can be split into two categories: Memory-based and incremental LWL.

A memory-based LWL method is Locally Weighted Regression (LWR) that stores all training data in memory in order to calculate the prediction at a query point [4]. For this purpose, all data is accumulated into a matrix from which the locally weighted regression coefficient must be determined via matrix inversion for each prediction. Obviously, this method is not suitable for online learning and real-time control as LWR suffers from intensive computations increasing quadratically by the number of data samples.

Locally Weighted Projection Regression (LWPR) instead is keeping and updating multiple locally weighted linear models to approximate non-linear functions [5][6]. This method enables incremental updates as the existing local model is just updated or a new one is created if there is no trustworthy model available. This is achieved by keeping a set of Receptive Fields (RF) with activation terms. Training data is then only added to a model if the RF provides sufficient activation. A forgetting rate weights training points gathered earlier down which is useful for systems that change over time. Furthermore, LWPR takes use of the online dimensionality reduction method Partial Least Squares (PLS) to handle redundant and irrelevant input data. As the method works incremental by design, it is well suited for our

proposed approach. In this work the LWPR implementation by Klanke et al. [7] has been adopted.

Another interesting class of non-linear function approximators is Gaussian Process Regression (GPR) which is a non-parametric regression method that can approximate (multivariate) non-linear functions [8]. However, the complexity of prediction is cubic in the number of training samples [9] as all training data is put into a single kernel. For this reason Csato et al. proposed Sparse Online Gaussian Process (SOGP) [10] building a sparse kernel by using a Bayesian online algorithm and subsampling relevant data from sequentially arriving training data. In this way, the processing time has been reduced significantly and therefore makes it suitable for online processing. An implementation of SOGP is available in the Online Temporal Learning (OTL) library[1] which has been used in this work.

Recursive Least Squares (RLS) is an iterative approach of Linear Least Squares Regression (LLSR) [11]. The RLS algorithm minimizes the squared error of incrementally arriving samples by updating the internal weight vector instead of computing a full matrix inversion as LLRS does. This results in a constant prediction and update time but the model can only cover linear dependencies between inputs and output variables. Again the OTL library provides an implementation of RLS which is used in this work as well.

3 Adaptive Compensation Feedforward Controller

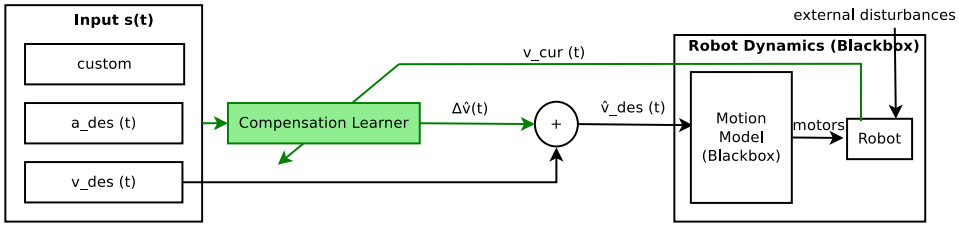


Fig. 1. Basic schema of the Adaptive Compensation Feedforward Controller

The proposed adaptive compensation feedforward control approach in Figure 1 can be categorized as Direct Modeling [12] which learns a motion error compensation model based on all previously observed errors during robot movements. The basic idea is not to learn the entire motion model determining a transferring action between two consecutive states but the required compensating action to nullify all motion errors produced by feedforward commands. This approach reduces model complexity significantly and enables to focus on training of motion error compensations rather than the complete motion model of the robot. As the presented adaptive compensation feedforward controller does only predict compensating actions, it can be used in combination with any existing controller such as feedback controllers. Additionally, the used compensation model and learning method can be initialized to return neutral (e.g. zero-valued) compensating actions when no prior

[1] https://bitbucket.org/haroldsoh/otl

training data is available. This reduces undesirable behaviors due to insufficient training or bad generalization capability of the used learner to a minimum.

The input state vector $\boldsymbol{s}(t)$ used by the proposed controller is composed of the desired velocity $\boldsymbol{v}_{des}(t)$, the desired acceleration $\boldsymbol{a}_{des}(t)$ and optional user selectable input values. The optional input values, for example IMU angles, can be selected by an expert in order to improve the inference from observed data.

In this work, the controller tracks the velocity of the robot base rather than single motor commands. Thus, the learning part tries to estimate a compensating velocity $\Delta\hat{\boldsymbol{v}}$ to be added up to the target robot base velocity $\boldsymbol{v}_{des}$ in order to achieve the target velocity in real world while using feedforward control only.

The adaptive compensation controller works in two steps: Compensation model learning and error compensation prediction which are both explained next.

3.1 Compensation Model Learning

An ideal feedforward controller should be able to replace any feedback controllers. Indeed, we can learn from feedback errors in a way the adaptive compensation feedforward controller can prevent motion errors a priori instead of performing corrections in the feedback control loop afterwards.

This problem can be stated in a mathematical way. The robot will move in the next time step with a certain speed $\boldsymbol{v}_{cur}(t+1)$ resulted from a previously commanded speed $\boldsymbol{v}_{des}(t)$ and an execution error $\boldsymbol{\epsilon}(t)$ caused by unmodeled disturbances:

$$\boldsymbol{v}_{cur}(t+1) = \boldsymbol{v}_{des}(t) + \boldsymbol{\epsilon}(t). \tag{1}$$

The motion error $\boldsymbol{\epsilon}(t)$ is often compensated by feedback controllers in robotic systems and can be determined by computing the difference between the commanded and resulting velocities:

$$\boldsymbol{\epsilon}(t) = \boldsymbol{v}_{des}(t) - \boldsymbol{v}_{cur}(t+1). \tag{2}$$

On the other side, $\boldsymbol{\epsilon}(t)$ can be minimized by adapting $\boldsymbol{v}_{des}(t)$ properly which results in the velocity compensation rule:

$$\hat{\boldsymbol{v}}_{des}(t) = \boldsymbol{v}_{des}(t) - \boldsymbol{\epsilon}(t) \tag{3}$$

$$\Rightarrow \qquad \boldsymbol{0} = \hat{\boldsymbol{v}}_{des}(t) - \boldsymbol{v}_{cur}(t+1) \tag{4}$$

with $\hat{\boldsymbol{v}}_{des}(t)$ as adapted control output. As a result, we require a compensation policy $\pi(\boldsymbol{s}(t))$ that must be able to predict a compensating action $\Delta\hat{\boldsymbol{v}}$ to overcome the expected motion error $\boldsymbol{\epsilon}$ based on the current state $\boldsymbol{s}(t)$:

$$\pi(\boldsymbol{s}(t)) \mapsto \Delta\hat{\boldsymbol{v}}. \tag{5}$$

In this work, the labels $\Delta\hat{\boldsymbol{v}}$ for the input state $\boldsymbol{s}(t)$ are updated iteratively based on the current prediction and observed motion error:

$$\Delta\hat{\boldsymbol{v}}^{(k+1)} = \underbrace{\Delta\hat{\boldsymbol{v}}^{(k)}}_{\hat{=}\pi(\boldsymbol{s}(t))^{(k)}} - k\boldsymbol{\epsilon} \tag{6}$$

where k controls the adaption rate and the initial value $\Delta\hat{\boldsymbol{v}}^{(0)}$ is usually zero. The resulting updated mapping $\boldsymbol{s}(t) \mapsto \Delta\hat{\boldsymbol{v}}^{(k+1)}$ is then given to the learner to update the compensation policy:

$$\pi(\boldsymbol{s}(t))^{(k+1)} \mapsto \Delta\hat{\boldsymbol{v}}^{(k+1)} \ (= \pi(\boldsymbol{s}(t))^{(k)} - k\boldsymbol{\epsilon}). \tag{7}$$

Non-modeled disturbance such as terrain (slippage) will affect execution errors as well. Therefore, it is highly desirable to add all available informative data to the state vector $\boldsymbol{s}(t)$. Modern learning approaches are well suited for such kind of mapping problems as demonstrated later.

3.2 Error Compensation Prediction

In each control loop cycle the controller estimates a compensating action $\Delta\hat{\boldsymbol{v}}$ based on the previously learned model. The compensation step can be expressed as:

$$\hat{\boldsymbol{v}}_{des}(t) = \boldsymbol{v}_{des}(t) + \Delta\hat{\boldsymbol{v}} = \boldsymbol{v}_{des}(t) + \pi(\boldsymbol{s}(t)). \tag{8}$$

The resulting compensated velocity command $\hat{\boldsymbol{v}}_{des}(t)$ can be directly forwarded to the robot controllers using a motion model internally that maps the velocity commands to motor commands. The measured velocity in the next time frame $\boldsymbol{v}_{cur}(t+1)$ is back-propagated to the proposed adaptive compensation feedforward controller in order to update the compensation policy as described above.

4 Experiments

Multiple evaluation criteria and scenarios have been evaluated during this work. Especially runtime performance is crucial for robot controllers as computational time must comply with the control loop timing. Furthermore, the presented framework has been tested in simulation first and applied on real robots afterwards. During this experiments, error correction capabilities and adaptation speed have been investigated which is summarized next.

In the following sections, an iteration refers to a reproducible movement based on predefined trajectories. These iterations are repeated in order to compare the performance of the used learners over time.

4.1 Runtime Performance

The processing time can be split into two parts: Compensation model update and prediction. While the update time is negligible for the control loop as it can be performed asynchronously, the prediction time determines the maximum possible control loop frequency. During all experiments, the update rate for the rescue robot was 40Hz, for the soccer robot 100Hz and the prediction rate 100Hz.

The experiments have been performed on the real robots and with a simulated SSL Robot within the software framework from TIGERs Mannheim [13] running on Intel i7 CPU with 2.6 GHz. Three different trajectories were specified while keeping the head orientation fixed (see Figure 4): A circle, a curved rectangle, and a star, each about 2m in diameter and between 8s and 60s long.

The robot receives velocity commands from our feedforward controller and uses an internal kinematics model to calculate wheel velocities. As the simulated robot would obviously behave perfectly, the low-level wheels commands have been artificially disturbed by a constant error while turning off all feedback controllers to simulate a non-calibrated system.

Table 1 shows that LWPR and SOGP take more processing time when performing star trajectory due to input diversity provided by this movement pattern. This behavior can be explained as LWPR uses local models whose number is increasing over time when additional input space has been covered while in case of SOGP the number of basis vectors increases over time.

		LWPR		SOGP		RLS	
	#	Prediction	Update	Prediction	Update	Prediction	Update
Rectangle	1	0.0310	0.0278	0.1700	1.1209	0.0490	0.2183
	2	0.0488	0.0423	0.2741	1.7781	0.0485	0.2111
	5	0.0479	0.0432	0.2704	1.6948	0.0479	0.2184
Circle	1	0.0271	0.0237	0.1477	0.9327	0.0524	0.2132
	2	0.0349	0.0317	0.1804	1.1189	0.0528	0.2198
	5	0.0360	0.0355	0.1899	1.2149	0.0519	0.2156
Star	1	0.0948	0.0845	0.8102	3.9831	0.0496	0.2293
	2	0.1317	0.1221	0.8146	4.1630	0.0490	0.2191
	5	0.1395	0.1315	0.8191	4.1901	0.0474	0.2202

Table 1. Average processing timings of 1st, 2nd and 5th iteration for all evaluated learning approaches given in milliseconds.

The noticeable update time of SOGP is caused by large matrix inversions. Although the prediction time of SOGP requires only a simple matrix multiplication, it suffers from the high matrix dimension and therefore underperforms LWPR and RLS. As expected, RLS runs with constant processing timings in every scenario.

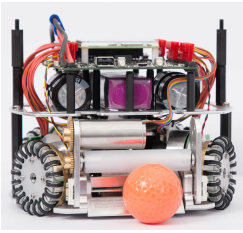
All methods can be considered sufficiently fast for real-time control. The prediction time allows theoretical control rates beyond 1kHz for all approaches, but SOGP can only be updated in a much slower rate. As prediction and update is done asynchronously, SOGP is still feasible for real-time control.

4.2 RoboCup Small Size Soccer Robot

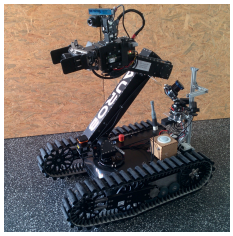
The real soccer robot (see Figure 2(a)) was driven with a kinematics model that maps velocity commands to motor commands [14]. The evaluation has been carried out with the trajectories introduced previously.

Running these experiments on the real soccer robot reveals another motivation for using adaptive feedforward controls. SSL robots are performing off-board processing due to limited space on the robot. This system setup introduces a delay time of about 100ms to the control loop from which feedback controllers are suffering.

Comparison of Learning Methods The next experiment has been performed with previously used learning approaches, compared with a PD feedback controller in addition. Figure 3 illustrates the evolution of all approaches after each iteration.



(a) SSL Robot by TIGERs Mannheim



(b) Rescue Robot by Team Hector

Fig. 2. Robots from different RoboCup Leagues have been used for evaluation.

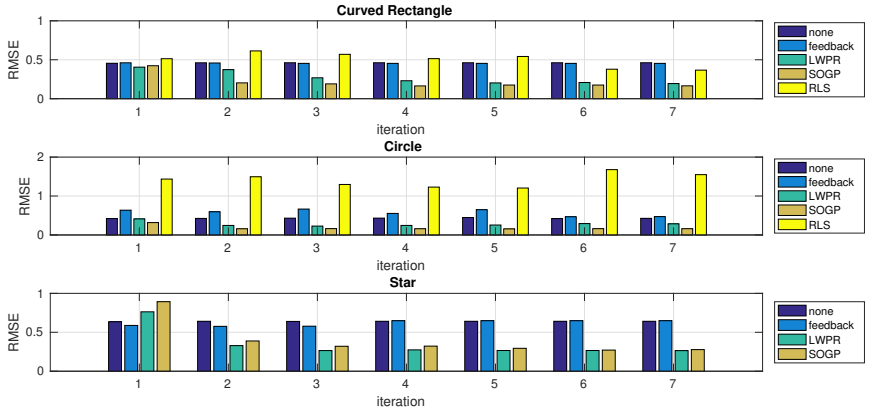


Fig. 3. Evolution of velocity RMSE for different control approaches applied at the real soccer robot. The 6th and 7th iteration have been performed without model updates to evaluate stability of the controller.

The most exciting result is the bad performance of the PD feedback controller caused by high control loop time leading to late and overcompensating reactions. This emphasizes the use of adaptive feedforward control proposed in this work.

RLS has performed very bad as well due to used linear compensation model. This linear model cannot capture non-linearities as required for controlling the robot orientation. This resulted in non-converging model updates and continuous overcompensation leading to oscillatory motions in the rotational axis. For this reason, RLS could not be applied to the star shape due to the lack of controller equilibrium. From this point on, RLS has been rejected from all future experiments due to stability issues.

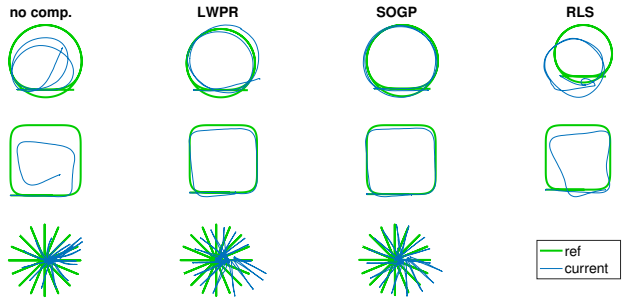
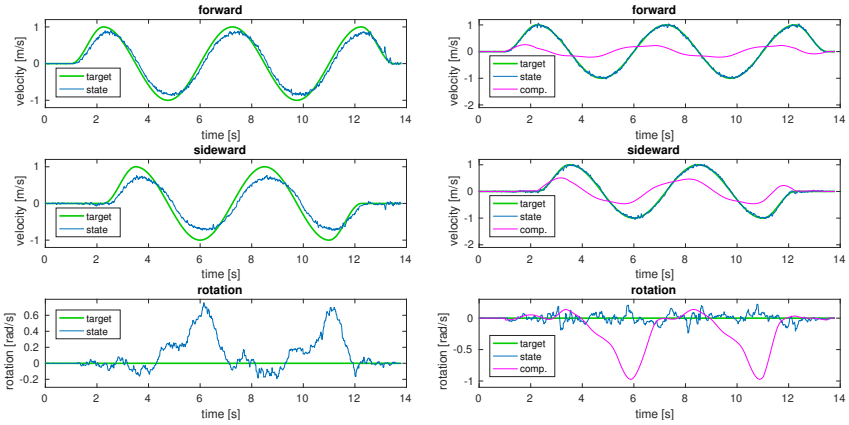


Fig. 4. Plot from the resulted movement of the fifth iteration for each learning approach. The green line shows the reference trajectory and blue the feedforward executed trajectory.

Throughout all experiments, LWPR and SOGP performed similar well. After the first three iterations, all approaches were able to reduce the overall pose error significantly by feedforward motion error compensation which is visualized in Figure 4. It shows the pose tracking improvement when using LWPR or SOGP.

Evaluation of Compensation Commands using LWPR Each deviation from the commanded velocity accumulates to a pose error which is typically compensated by feedback pose controllers that have been disabled throughout all experiments. As stated before the goal of our approach is to reduce the required pose adjustments. Hence, the velocity deviation is analyzed next using LWPR as example as it has performed best.



(a) Uncompensated (b) LWPR compensation

Fig. 5. Comparison of circle trajectory execution without (left) and with LWPR compensation after five iterations (right) using the real soccer robot.

The uncompensated robot system shows a phase shift in transverse plane leading to reduced amplitudes (Figure 5(a)) and is affected by an undesired rotational velocity accumulating to a rotational drift up to 90° after each iteration. This can be explained by the use of omni wheels causing a lot of non-modeled slippage.

Figure 5(b) shows how the LWPR feedforward controller is able to compensate successfully the phase shift and reduces even significantly the rotational drift of the system. The magenta line shows the compensating actions that the controller predicts at the corresponding states given the execution errors that have been observed in the five previous iterations. A full video of this experiment is available online at `https://youtu.be/3__shNa66oA`.

Evaluation of Performance on a Major Malfunction The ability to adapt towards hardware wear and malfunctions is a major motivation for adaptive feedforward control. In the next experiment, the right rear wheel has been disabled causing the robot to drive a curve, although a forward movement is commanded.

This experiment should evaluate how well LWPR and SOGP can adapt to such situations. The first step was to drive in a straight line back and forth, starting with an empty model.

Figure 6 shows that SOGP converges much slower than LWPR during this experiment. In case of LWPR, the quick convergence has been achieved by tuning the integrated forgetting factor. But in contrast SOGP has no notion of forgetting old data why already sampled data will stay in the model until it is replaced by

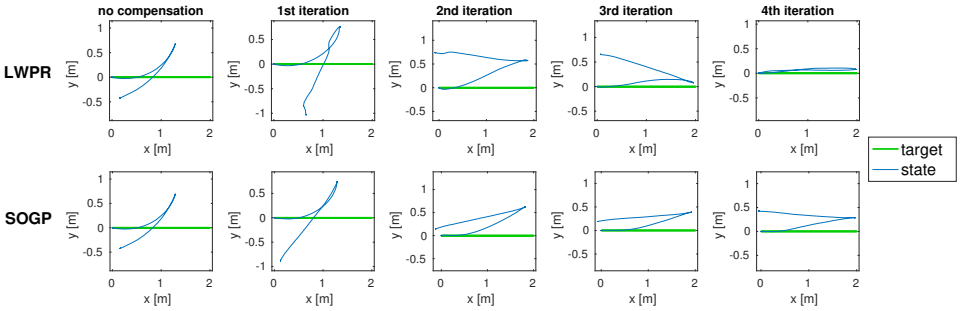


Fig. 6. Evolution of controllers when right rear wheel is disconnected.

new observations. Thus, the model will indeed slowly adapt, but the adaptation speed cannot be easily influenced.

Next, the previously learned models were reused, but the motor has been enabled again allowing for straight forward movements without any compensation.

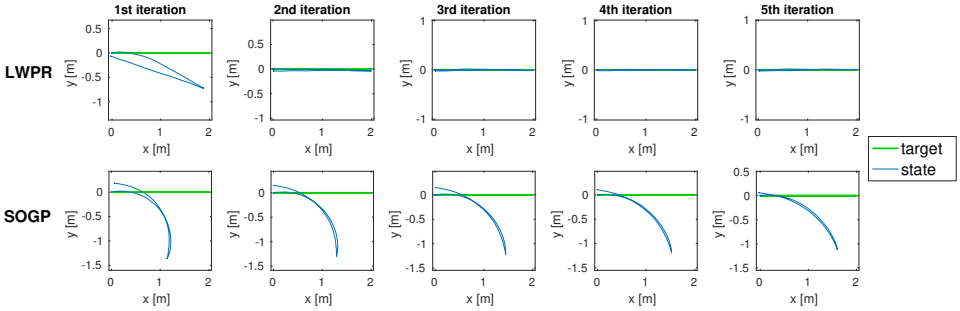


Fig. 7. Robot trajectory when right rear wheel is reconnected after a compensation model has been learned with disconnected wheel.

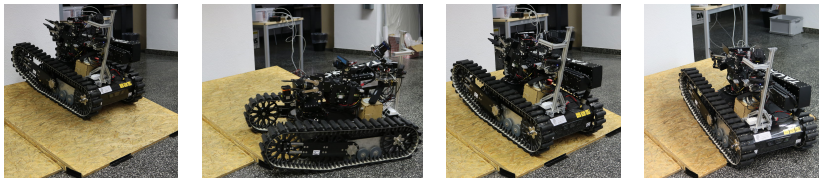
In the first iteration all controllers performing an expected overcompensation (Figure 7). While LWPR can adapt to this new situation after the first iteration quickly, SOGP converges only slowly back to a zero compensation policy. As result LWPR shows clearly a better adaption capability.

4.3 RoboCup Rescue Tracked Robot

The second used robot platform for experiments is a tracked robot from the RoboCup Robot Rescue League by Team Hector [15]. This nonholonomic platform is based on a tracked differential drive and cannot move sidewards, which is why lateral drifts have devastating influence on locomotion performance. Furthermore, the rubber chains cause downhill sliding on inclined terrain during movements, no matter which orientation the robot has.

The robot world pose is estimated using Hector Slam [16] and the data from the integrated IMU is added to the input state of our adaptive compensation feedforward controller as discussed in Section 3. All compensating actions become dependent on the robot's three-axis-orientation, especially roll and pitch.

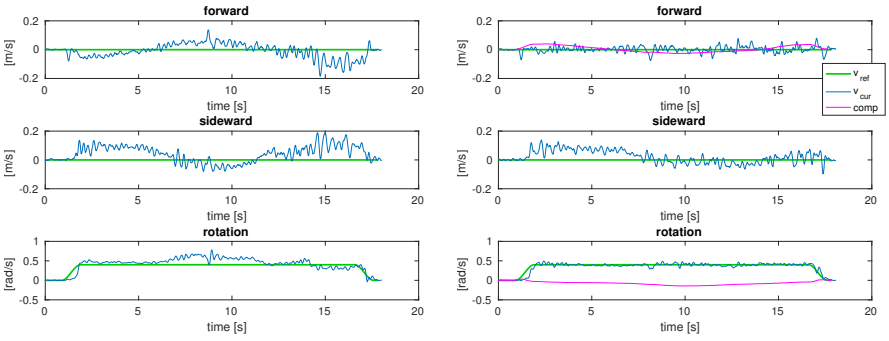
Rotation on a Ramp Movements on even terrain can be executed precisely by the existing controllers, but the discussed slippage on inclined terrain is an interesting application for the presented controller. In the following experiment the robot should perform a full 360° rotation on a ramp. An ideal system would just rotate in place and stop exactly on its initial position (see Figure 8(a)).



(a) Initial Pose (b) No Comp. (c) LWPR (d) SOGP

Fig. 8. 360° Rotation on a ramp using LWPR and SOGP after three iterations.

But the real robot system slides downhill ending off the ramp (see Figure 8(b)). Both approaches, LWPR and SOGP, cannot prevent sliding downwards at all but reduce it as illustrated in Figure 8. Only LWPR was even able to stabilize the rotatory velocity resulting in nearly zero orientation offset after each iteration.



(a) Uncompensated velocity (b) LWPR compensated velocity

Fig. 9. Velocity plot during the 360° rotation on a ramp with LWPR compensation.

The plots in Figure 9(a) illustrates how the robot is sliding downhill, observable as non-zero velocities in transverse plane depending on current robot orientation. The LWPR controller is able to reduce successfully deviations by adapting the speed of each single track indirectly. In Figure 9(b) the compensating actions are plotted in magenta demonstrating how the LWPR predictor does influence the track speeds by predicting appropriate compensating actions. This adaptive behavior can be watched in the video at `https://youtu.be/cW20_JsZCzg`.

All compensating actions are predicted reasonably based on motion errors observed previously. However, velocity deviations in lateral direction cannot be com-

pensated directly due to nonholonomic constraints. Thus, the LWPR and SOGP feedforward controllers are able to reduce but not nullify sliding downhill the ramp.

5 Results

We have demonstrated that our proposed adaptive feedforward controller is able to compensate non-modeled disturbance online. All used learning approaches can run using state of the art consumer computer hardware in control rates beyond 1kHz.

The versatility of the proposed controller is achieved by the commonly used velocity interfaces. Although it has been used as robot base velocity controller throughout this work, it can be considered as generic velocity controller which would even work on motor level. But it should be noted that velocity estimation accuracy is crucial for training the learner effectively.

While the SSL robot showed undesirable motion errors, the proposed controller is able to improve the movement accuracy significantly. Especially the robot orientation is suffering from major deviations depending on multidimensional input but could be stabilized as well by the implemented control policy.

While all three learning methods showed improvements in simulation, experiments with the real robot revealed that RLS uses too simplistic models for real world application. A simulated malfunction showed that SOGP suffers from its sample based model and was not able to adapt as quickly as LWPR does.

We applied the proposed framework to a tracked robot with completely different locomotion capabilities in order to demonstrate the versatile usability. Although the tracked robot performs less smooth movements, has a less accurate state estimation than the SSL robot, and can only compensate sagittal and rotational motions, the proposed controller could be applied here as well.

In order to compensate slippage on a ramp, the roll and pitch angles from IMU have been used as additional input values. The experiments showed that sagittal and rotational velocity tracking has been improved significantly and even sliding downhill on the ramp could be compensated by 50%. The remaining downhill movement cannot be compensated due to nonholonomic constraints.

All experiments have been performed under controlled conditions. As soon as there are non-consistent external influences that cannot be captured as input state for the learner such as changing traction, the adaptive feedforward controller may infer wrong, but the model will be adapted continuously. In such scenarios, the forgetting factor of LWPR is a crucial mechanism to control adaptation speed towards structural and environmental changes. However, the experiments demonstrated the high adaptation rate of the LWPR controller making it suitable for real hardware.

6 Conclusion

In this paper, an adaptive compensation feedforward controller framework based on a compensation model has been presented that can be used in conjunction with any existing motion model. While commonly used motion learning approaches are dependent on training the whole robot motion model, the presented compensation controller can be deployed without any pre-training.

A selection of different learning approaches has been analyzed for their capability as adaptive compensation feedforward controller. Only LWPR proved to comply

with required computational speed, prediction accuracy, and model adaptability during the experiments. The proposed controller is able to compensate unmodeled disturbances such as robot decalibration or slippage. It can even improve the system's fault tolerance against hardware malfunctions due to online adaptability.

The framework has been applied to two very different robot systems across the RoboCup Leagues. Although those robots are underlying distinct mobility challenges, the controller was able to improve locomotion performance for each system.

References

1. Gloye, A., Wiesel, F., Tenchio, O., Simon, M., Rojas, R.: Robot Heal Thyself - Precise and Fault-Tolerant Control of Imprecise or Malfunctioning Robots. RoboCup 2005. International Symposium, Osaka, Japan (2005)
2. Passold, F., Stemmer, M.R.: Feedback Error Learning Neural Network Applied to a Scara Robot. 4th Int. Workshop on Robot Motion and Control (2004) 197–202
3. Atkeson, C.G., Moore, A.W., Schaal, S.: Locally Weighted Learning for Control. Artificial Intelligence Review **11**(1) (1997) 75–113
4. Christopher, A., Andrew, M., Stefan, S.: Locally Weighted Learning. Artificial Intelligence Review **11**(1-5) (1997) 11–73
5. Vijayakumar, S., Schaal, S.: Locally Weighted Projection Regression: An O(n) Algorithm for Incremental Real Time Learning in High Fimensional Space. In: Proc. 17th Intl. Conf. on Machine Learning (ICML). Volume 1. (2000) 288–293
6. Vijayakumar, S., D'Souza, A., Schaal, S.: Incremental Online Learning in High Dimensions. Neural Computation **17**(12) (2005) 2602–2634
7. Klanke, S., Vijayakumar, S., Schaal, S.: A Library for Locally Weighted Projection Regression. Journal of Machine Learning Research **9**(1) (2008) 623–626
8. Rasmussen, C.E.: Gaussian Processes for Machine Learning. (2006) Chapter 2
9. Barber, D.: Bayesian Reasoning and Machine Learning. Cambridge U.P. (2012)
10. Csató, L., Opper, M.: Sparse On-line Gaussian Processes. Neural computation **14** (2002) 641–668
11. Benesty, J., Paleologu, C., Gänsler, T., Ciochina, S. In: Recursive Least-Squares Algorithms. Springer Berlin Heidelberg (2011) 63–69
12. Nguyen-Tuong, D., Peters, J.: Model Learning for Robot Control: a Survey. Cognitive processing **12**(4) (2011) 319–340
13. Ryll, A., Geiger, M., Ommer, N., Sachtler, A., Magel, L.: TIGERs Mannheim - Extended Team Description for RoboCup 2016 (2016)
14. Rojas, R., Förster, A.: Holonomic Control of a Robot with an Omnidirectional Drive. KI-Künstliche Intelligenz (2006) 7
15. Kohlbrecher, S., Rose, C., Koert, D., Manns, P., Kunz, F., Wartusch, B., Daun, K., Stumpf, A., von Stryk, O.: RoboCup Rescue 2016 Team Description Paper Hector Darmstadt. Technical report, Technische Universitaet Darmstadt (2016)
16. Kohlbrecher, S., Meyer, J., von Stryk, O., Klingauf, U.: A Flexible and Scalable SLAM System with Full 3D Motion Estimation. In: Proc. IEEE Intl. Symp. on Safety, Security and Rescue Robotics (SSRR), Kyoto, J, IEEE (2011) 155–160