**Workshop on Robot Motion Planning:**
**Online, Reactive, and in Real-time**
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

# Collective Motion Pattern Scaling for Improved Open-Loop Off-Road Navigation

Frank Hoeller, Timo Röhling, Markus Ducke, and Dirk Schulz

***Abstract*— This paper presents an adaptive navigation system which is able to steer an electronically controlled ground vehicle to given destinations while it adjusts to changing surface conditions. The approach is designed for vehicles without a velocity controlled drive-train, making it especially useful for typical remote-controlled vehicles without upgraded motor controllers. The vehicle is controlled by sets of commands, each set representing a specific maneuver. These sets are combined to form trajectories towards a given destination. While one of these sets of commands is executed the vehicle's movement is measured to refine the geometry of all maneuvers. A scaling vector is derived from the changes in dimensions of the bounding boxes of the assumed and the actual path, which is then used to collectively update all known maneuvers. This enables the approach to quickly adapt to surface alterations. We tested our approach using a 300 kg Explosive Ordnance Disposal (EOD) robot in an outdoor environment. The experiments confirmed that the Collective Motion Pattern Scaling significantly increases the adaptation performance compared to an approach without collective scaling.**

## I. INTRODUCTION

In the design of robot systems operating in unstructured outdoor environments special care has to be taken that the robots do not accidentally collide with obstacles in their vicinity. Compared to indoor situations the robot can suffer drastically more damage from the more hazardous surroundings. The risk is increased by different ground surfaces, which have a distinct effect on the wheel grip. The resulting deviation has to be anticipated to ensure the reproducibility of planned motions, and thus making collision avoidance possible.

Additional complications arise for robots which were designed for remote-control. Such robots are usually only equipped with relatively simple motor controllers missing an appropriate servo loop for interpreting velocity commands. Unfortunately, this is a requirement for most classic navigation algorithms. Moreover, the impact of adhesion changes is further intensified by such open-loop controllers because no matter how much the wheel grip, and thus the behavior of the robot changes, there is no feedback of the actual movement. Of course, these problems could easily be solved by using a wheel encoder or similar, but adding such to existing, especially commercial robots is rarely possible. Mostly a time-consuming redesign or an expensive new acquisition are the only options.

In this article we present an approach, which allows a mobile robot with any kind of electronic motor controller to operate in outdoor environments while adjusting to changing surface conditions to provide safety and effectiveness. All components of our system follow a local navigation paradigm and do not need global information on the environment, neither of surface characteristics nor on obstacles. Instead, the system decides solely based on the robot's sensory input.

Frank Hoeller, Timo Röhling, Markus Ducke, and Dirk Schulz are with the Fraunhofer Institute for Communication, Information Processing and Ergonomics FKIE, Germany

Fig. 1. A Telerob Teodor robot equipped with sensors for basic autonomous navigation.

The motion planning developed for our robot composes paths by combining predefined Motion Patterns. Each Motion Pattern consists of a set of robot commands and a series of poses that represent the robot's movement when the command set is executed by the controllers. With these Motion Patterns, the local navigation module repeatedly computes trees of collision-free command sequences. From each tree a path is extracted which brings the robot close to the destination coordinate as fast as possible.

If one is able to measure the robot's motion on the fly, e.g using SLAM (simultaneous localization and mapping) techniques or an INS (inertial navigation system), one can also monitor movement trajectories. Compared to drive-trains with servo loop that only regard the motor speeds, the results of command sequences can be observed on a larger scale, which allows to tackle the surface traction problem in a novel way: The collected trajectory data is used to update the previously measured movement trajectory of the corresponding Motion Patterns. Furthermore, the detected changes are propagated to all other Motion Patterns by calculating a scaling vector from the alteration in dimensions of a trajectory. The upgraded Motion Patterns are handed over to the planning process and used for the tree generation from then on. Note, that it is not possible to adapt the command sequence to match the desired trajectory because the mapping from trajectories to commands is unknown.

The remainder of this article is organized as follows: After discussing related work in Section II, we introduce our Motion Pattern based local navigation approach in Section III, followed by a description of the learning and collective

scaling procedures in Section IV. Before we conclude, we describe some experiments to illustrate the capabilities of our approach. We implemented our approach on a Telerob Teodor EOD robot (Fig. 1) and verified its feasibility in outdoor settings.

## II. RELATED WORK

In the field of outdoor robotics the terrain always is of special interest. The analysis and classification of different surfaces regarding their traversability has been addressed by many different authors. The classification of the different surface types using vibration sensors is very popular because this sensing mode is not vulnerable to lighting or perspective issues. Brooks et al. [1] attached this kind of sensor to axle arms and classified different terrains by traversing them. They used offline learning in combination with a voting mechanism to enable the system to identify a set of different surfaces. Unfortunately, classification approaches of this type can only identify known terrains without deriving information concerning the behavior of a vehicle on the respective surface. The identified terrain type would have to be associated with a parameter set for the trajectory generator in an intermediate step. Furthermore, this would make the navigation dependent on the a priori learned identification data, which would violate the intended local navigation paradigm.

The DARPA Grand Challenge winning robot Stanley [12] also uses a vibration sensor to regulate its maximum speed. Unwilling to catalog every possible terrain type, Stavens et al. evaluate the occurring vibrations to limit Stanley's speed according to observed human driving behaviors [10]. A similar approach is presented by Castelnovi et al. [2], but instead of sensing vibrations the authors used a 2D laser range finder aimed downwards to calculate a ruggedness grade. Based on this result the robot's top speed is reduced, resulting in a decreased number of terrain-related incidents. A combined approach was later proposed by Stavens et al. as an upgrade for Stanley's system [11]. Here a learning component associates vibration intensities with surface profiles measured with a forward-facing 3D laser distance scanner. This way the system can automatically learn terrain-speed-associations. Thus the maximum speed can be adjusted before the vibration sensor detects a surface transition and the vehicle is exposed to less shock. These approaches show several similarities to the technique presented in this paper as the analysis of the ground directly affects the local navigation. Nevertheless, only the maximum translation velocity is altered. Rotation velocities are not addressed at all, which is not necessary for a robot like Stanley.

Another interesting approach by Martinelli et al. [5] also uses laser range finders, but in combination with SLAM and Kalman filter techniques. The system is able to determine the systematic component of the odometry error by using the wheel encoder readings and the estimated SLAM position. The non-systematic error, which is more interesting in outdoor applications, can also be determined by a Kalman filter applied on a history of robot states. These are provided by the former Kalman filter, making the estimation indirectly dependent on wheel encoders and closed-loop control.

Crusher, a six-wheeled robot for extreme outdoor environments also suffered from trajectories differing from planned paths. Seegmiller at al. proposed an approach to automatically calibrate a dynamic model [9]. It linearizes the nominal vehicle model and then calibrates the dynamics to explain the observed prediction residuals using a Kalman filter. Their system, just as our system, takes advantage of the precise short-term localization techniques or sensors available. Although results are impressive, their approach is again tailored to a velocity driven model which unfortunately is not applicable to our robot.

The combination of motion templates and learning has been used widely in the area of walking robots. In this field, learning techniques are mostly applied to improve walking policies that were derived from simulations [6] or observed from human walking [7]. Furthermore, due to the complexity of biped locomotion, every walking robot is equipped with many sensors to determine its stance and to allow closed-loop control.

Similarly, the concept of motion template based learning has also been employed to simplify the learning of complex motions [8]. In contrast to our approach the templates are parameterized, so they can be adjusted to fit the desired trajectory. This implies a feasible correlation between parameter input and drive-train behavior.

## III. LOCAL NAVIGATION WITH MOTION PATTERNS

The core of the overall approach is a local navigation planning component that directly controls the robot and steers it on a collision-free path from its current position to a given destination in configuration space. For this purpose special precautions for the open-loop motor controllers have to be taken. Since remote-controlled robots lack a velocity regulator circuit, the control commands influence the motor power directly. This induces that their outcome depends on many factors and is far too complex to compute in an online approach. To make motion planning still possible, we introduce Motion Patterns. The first component of a Motion Pattern *MP* is a series of robot control commands $U = (u_1, \ldots, u_T)$. A command $u_t$ can be of any type and dimension: when used with a Teodor robot they are motor power commands, when controlling a car they probably are throttle position and steering angle. A command sequence $U$ is immutable, which implies that Motion Patterns cannot be parameterized e.g. regarding their velocity. The second component of a Motion Pattern is an array of oriented relative positions $R = (\Delta r_1, \ldots, \Delta r_T)$. It represents the trajectory on which the robot would probably move when the command series is sent to the robot, so each pose $\Delta r_t$ describes the relative position of the robot after it executed the command sequence from $u_1$ to $u_t$. The Motion Patterns can now be combined to form motion paths $P = (R_1, \ldots, R_k)$. Of course it has to be checked if concatenated Motion Patterns fit together so that no harsh velocity change creates

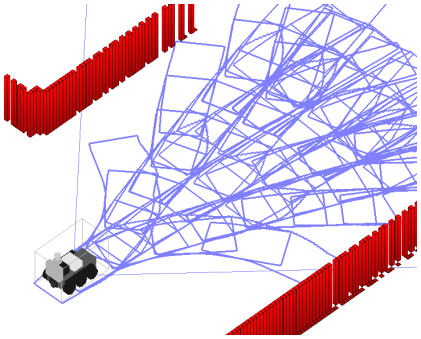

Fig. 2. A planning tree of collision free paths that has been build using Motion Patterns in an 2-dimensional simulated environment.

unexpected movement trajectories. The created paths can then be checked for collisions e.g. by using an occupancy grid [4]. The corresponding sequence of series of commands $C_P = (U_1, \ldots, U_k)$ can be merged to a large array of robot commands which can be executed by the robot sequentially. Notice that the number, shape, and complexity of Motion Patterns are not restricted, but definitely have an impact on the planning process. In general, with fewer patterns a larger range can be covered, while more patterns increase the quality of the resulting paths. Since basic navigation capabilities were sufficient for the following learning algorithm, a set of five different Motion Patterns was used.

Based on the model above, we can now also build a collision-free tree of Motion Patterns and extract the best path towards the destination. The path planning process is described in detail in [4] and an example tree generated by this technique can be seen in Fig. 2. The theoretical principles of a very similar navigation approach are examined in [3] extensively.

## IV. MOTION LEARNING USING COLLECTIVE MOTION PATTERN SCALING

A problem arising from our kind of local navigation is its sensitivity to surface and traction changes. Motion Patterns are created for specific surfaces only. And it is unlikely that the surface or the surface's condition always remains constant, especially in outdoor scenarios.

To compensate for this, the local navigation has been extended by a learning mechanism. While the command set of a selected Motion Pattern *MP** is executed, the robot's reactions are measured. For this purpose the robot's movement trajectory *M*, consisting of the $x$, $y$ and $\varphi$ deltas, is recorded. The saved trajectory $R_{MP^*}$ inside the Motion Pattern *MP** can directly be updated with this updated measurement. This technique was combined with an exponential smoothing (see below) to form the first version of the learning Motion Pattern based local navigation. The potential of this basic system has been shown in [4]. Both learning approaches, the basic version from [4] and the improved version discussed here, as well as the planning mechanism assume, that applied Motion Patterns yield in similar trajectories repeatedly. The exponential smoothing can cope with singular discontinuities, but it is not able to handle continuously changing results, i.e. occuring on slopes or rough terrain. Both methods are computationally simple and the calculation effort is negligible compared to the time needed for path planning.

The trajectory update inside the Motion Pattern can be regarded as information gain. We would like to propagate this additional knowledge to all other Motion Patterns as well to improve the adaption speed considerably. For this purpose we calculate a vector of scaling factors $V = (x_s, y_s, \varphi_s)^T$ by comparing the recently measured trajectory and the prediction $R_{MP^*}$ saved inside the Motion Pattern *MP**. $x_s$ and $y_s$ regard the positions inside the trajectory, and $\varphi_s$ the trajectory's yaw information. The first approach would be to compare the final pose of the measured trajectory *M* and the predicted trajectory from *MP** to compute these factors. Unfortunately, this would decrease the robustness for trajectories which have a final lateral position close to its initial value (e.g. double-lane change maneuvers). In these cases, it is likely that the measurement noise exceeds the position change, which would generate enormous scaling factors. A similar problem occurs for patterns which only include a movement in only one dimesion, e.g. forward motions or in-place turns. Although these patterns can be scaled, they cannot be used to calculate a reasonable scaling vector $V$ because their movement in the unaddressed axes is only caused by noise and would again generate exaggerated scaling factors. For this reason, patterns like these are taken out of the learning process, because it is not possible to gain information from something, that did not change. To adresse the former case of this problem, the length and the width of the bounding boxes (*BB*) around the two considered trajectories are used to calculate the scaling factors $x_s$ and $y_s$. The yaw scaling factor $\varphi_s$ is calculated similarly to the position factor: here the interval between the minimum and maximum of all recorded yaw angles is used. Now we can compute a quotient for every dimension and compose the scaling vector $V$:

$$V(M, MP^*) = \begin{pmatrix} x_s \\ y_s \\ \varphi_s \end{pmatrix} = \begin{pmatrix} \frac{\text{Length}_{BB}(M)}{\text{Length}_{BB}(MP^*)} \\ \frac{\text{Width}_{BB}(M)}{\text{Width}_{BB}(MP^*)} \\ \frac{|\text{AngleInterval}(M)|}{|\text{AngleInterval}(MP^*)|} \end{pmatrix} \quad (1)$$

This vector can now be used to collectively scale all Motion Patterns making it possible to update even yet unregarded Motion Patterns. An example application is illustrated in Fig. 3. In this manner the knowledge of change of movement behavior is propagated without the need to wait for every Motion Pattern to be chosen and executed, reducing the adoption time notably.

In both cases, without and with Collective Motion Pattern Scaling, the new predictions are integrated in the Motions Patterns' existing trajectory using a component-by-component exponential smoothing function. This allows continuous learning and at the same time smooths minor surface variations to prevent an oscillating learning behavior:

$$R_{p,t} = (1 - w)M_{p,t} + (w)R_{p,t-1} \quad (2)$$

with $0 \leq w < 1$. Here $M_{p,t}$ is a new trajectory for a Motion Pattern $p$ at time $t$, measured or derived by collective scaling.

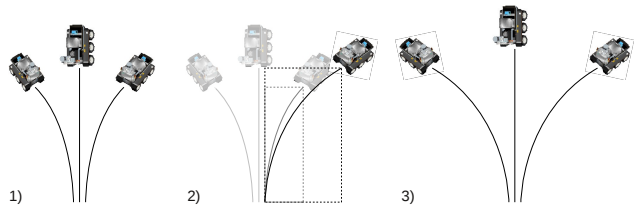


Fig. 3. Evolution of a Motion Pattern set with three patterns: 1) Initial set before a Motion Pattern is chosen and executed. 2) After measuring a Motion Pattern's (MP*) result the bounding box changes are analyzed. 3) All Motion Patterns are scaled yielding to a new set.

$R_{p,t-1}$ represents the existing prediction of the pattern and $R_{p,t}$ the updated prediction. To limit the impact of new measurements $w$ is introduced. Setting it to $0.5$ turned out to work fairly well. The whole learning approach is outlined in Algorithm 1.

**Algorithm 1** Collective Motion Pattern Scaling

**while** $inMotion$ **do**
  get next Motion Pattern *MP* from planner
  send *MP* to motor controllers and record motion
  let *M* be the recorded robot motion
  calculate scaling vector *V* ▷ see eqn (1)
  UPDATEMOTIONPATTERN(*MP*, *M*)
  **for all other** MotionPatterns **do**
    UPDATEMOTIONPATTERN(*MP*, $MP \cdot V$)
  **end for**
**end while**

**procedure** UPDATEMOTIONPATTERN($mp$, *update*)
  let $w$ be the impact reduction
  $mp \leftarrow (1-w) \cdot mp + (w) \cdot update$ ▷ see eqn (2)
**end procedure**

The idea behind this mechanism is that changes in road grip in general affect the forward and lateral movement achieved when executing a command sequence, and that these changes can be approximately captured by the bounding box around the resulting trajectory. Although we disregard it in our learning approach, there certainly is a relation between the length and the width of a trajectory: If the ground surface changes from a sticky to a slippery nature, an executed turn would be wider than before. The resulting bounding box of the trajectory would be longer and correspondingly narrower. But to model this relation more information about the shape of the trajectory would be needed. By the nature of our approach, these are not available, so further detailed analysis of the trajectory would require more computational time, which at the moment is beyond the time frame of our online application. Surface transitions are likely to change also the orientation of the robot while executing a Motion Pattern. To be able to predict complete poses, the position scaling technique is reduced to one dimension and applied again to the orientation values. In a one-dimensional space a bonding box diminishes to an interval enclosing all occurring yaw angles.

Although not required for the learning technique presented here, it is reasonable that most sets of Motion Patterns contain a number of symmetric patterns, e.g. a left and a similar right turn. If one of these patterns is executed and measured, a very precise prediction for the other is generated, which lessens the error margin of the approximation and thus increases the effectiveness of the collective scaling.

## V. EXPERIMENTS

The system described in the previous chapters has been tested in simulations but using data from a real robot. The Collective Motion Pattern Scaling algorithm proposed here is compared with its predecessor, whose performance is shown in [4]. To demonstrate the performance of the learning and prediction techniques a controlled environment is essential. Although this might be realizable in simulations, the authors have chosen a different approach.

### A. Testing Approach

The following experiment aims at investigating the results of the two mentioned algorithms on a transition from a surface $A$ to another surface $B$. To maximize the quantitative outcome of the experiments the data acquisition was separated from the algorithm tests. For this purpose, the different surfaces were traversed separately to collect data sets for each surface. Later, these sets are used in combination to simulate surface transitions.

In the data collecting stage the robot executed a large number of Motion Patterns on different ground surfaces, one surface at a time. The driven trajectory of each Motion Pattern was collected, resulting in a sample set $RP_S$ of 300 recordings per Motion Pattern for every considered surface $S$. As preparation for the second stage a set of averaged Motion Patterns is calculated from the 300 recordings for each $RP_S$. The result is a set of Motion Patterns $IP_S$ with very precise trajectory predictions for the regarded surface $S$. When reproducing a terrain change from a surface $A$ to a surface $B$, the set $IP_A$ is used as initial Motion Pattern database representing the already learned surface type $A$.

For the next step a test sequence *TS* of Motion Patterns, which is executed after the virtual terrain change, has to be determined. To show the propagation capabilities, two sequences were chosen and are used alternately. Each sequence consists of three turning patterns resulting in a left-left-right and accordingly a right-right-left motion. Note that the left and the right turn Motion Patterns roughly are mirrored equivalents and have a total length of about a second. The test sequence has to be compiled of turning patterns because straight movements only allow one-dimensional corrections (see Section IV). For each run of the test sequence the initial Motion Pattern database $IP_A$ is used to create identical starting conditions.

In the next step the test sequence *TS* is processed one pattern after another. Before a Motion Pattern's command sequence is sent to a robot, the two algorithms to be tested already have a prediction of the prospective trajectory. The accuracy of these trajectories will be inspected. Instead of

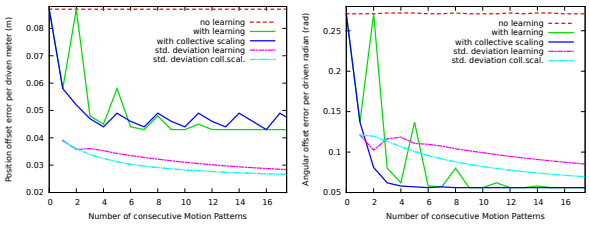


Fig. 4. Error developing of a Teodor robot changing from cinder to grassland during an experiment with extended run-time.

sending the command sequence of each Motion Pattern to a real robot and recording its motion anew, the pattern is processed offline: In exchange for the online motion recording, we use a random trajectory recording from the initially measured collection of trajectories $RP_B$ of surface $B$. The trajectory is given to the two algorithms to initiate the learning process to update their predictions for the next request. We assume that the trajectories recorded for a pattern on one particular surface are exchangeable i.e. that the trajectories are randomly distributed given a surface. This allows us to do quantitative evaluations using sampling techniques. Further, this enables us to easily process larger numbers of test repetitions.

The processing of the test sequence is repeated 30,000 times. The random trajectories from $RP_B$ are chosen anew every time, but in a fashion that both learning algorithms receive the same data in every run. At the end of every pattern of *TS*, the final position of the recording is compared with the two position predictions provided by the two learning algorithms.

A disadvantage of this simulation method is that surface changes can only be simulated in between Motion Patterns, which is unlikely to happen in the real world. But even in the likely cases the learning mechanisms would adjust the trajectory predictions in the right direction, still reducing planning error margins. The learning process would only be slowed down, and both algorithms would be affected in the same way.

### B. Testing Results

The input data for this simulation study was recorded using a tracked Telerob Teodor EOD robot which was upgraded with the required sensors to enable autonomous behavior. In addition, an INS was installed to enable easy and precise movement recordings. For the first phase of the experiment the robot was operated on the following four surfaces to collect the data for the sample sets $RP_S$: tarmac, grassland, chunky gravel, and volcanic cinder (commonly used for soccer fields). During the experiment the robot always moved at its maximum speed of about 1.0 m/s. The generated sample sets were used to simulate all possible surface transitions.

Table I shows a comparison of four processing methods including a naive and an omniscient policy: a) no learning while keeping the database, b) single pattern learning from [4], c) learning with collective scaling, d) a priori correctly chosen pattern database with learning deactivated showing the minimal possible errors. The error reduction percentages for the normal learning and the learning with collective scaling are presented in Table II. As in the simulated experiment, the reduction of the orientation error is noticeably higher then the position error reduction. The angular error reduction ranges from 7.3 % up to 27.9 % with one exception that probably resulted from similar surface properties. The position error reduction on the other hand only reaches 17.8 % at best, but is normally below 10 %, sometimes even negative. The measured orientation error values are quite large, but it has to be kept in mind that these are normalized. Just as the position error are scaled with respect to the number of driven meters, the rotation errors are scaled with respect to the covered angle. For the patterns examined here the scaling factor is around 3.

Note that the position error reduction can artificially be improved by increasing the pattern length, taking advantage of the large orientation error reduction. When the pattern length is doubled most of the negative position error reductions raise well above zero. To keep the results realistic, these oversized patterns were not used.

Fig. 5 depicts the error distributions of the two tested approaches when changing from cinder to grassland as an example. The distribution of the position error is not as well formed as the distribution of the angular errors, but still a shift towards small errors can be seen. Most of the error distributions of the other transitions with a crucial reduction look similar.

Since both learning algorithms use the same exponential smoothing function, both errors will correlate with the minimal error if the experiment is continued long enough. Fig. 4 shows the error development of the two algorithms when the chosen Motion Pattern sequence is processed repeatedly without resetting the initial pattern database. Both algorithms reach the minimal error during the extended time span, but especially in the beginning the differences are very large. This confirms that the beginning phase is crucial for a fast adaption, and encourages the concentration on this phase. The position error graph also reveals the reason for the poorer performance of the position estimation: After reaching a value of about 0.045 m, the error begins to oscillate in a range of approx. 0.005 m. These 5 mm are the error caused by the propagation of the Collective Motion Pattern Scaling. At this point the algorithm has reached its highest absolute accuracy; even when extending the Motion Pattern length, the amplitude does not increase.

The long-term experiment also enables us to evaluate the variance of the occurring errors. After the first three patterns the standard deviation with collective scaling is higher than the standard deviation of the simple learning method. This is caused by the exponential smoothing: When it is used, it intentionally prevents instant adaption and generates intermediate trajectory predictions while converging towards the measured behavior. When adding collective scaling, these intermediate predictions appear more often due to the greater number of adjusted Motion Patterns, especially immediately after a surface transition. As soon as the adaption to a new surface is completed, the deviation decreases and stays below

TABLE I

RESULTS OF THE SIMULATIONS: POSITION (FIRST ROW) AND ORIENTATION (SECOND ROW) ERRORS OF ALL POSSIBLE TRANSITIONS. VALUES DENOTE THE ERROR PER DRIVEN METER/RADIAN. A) NO LEARNING, B) SIMPLE LEARNING, C) COLLECTIVE SCALING, D) OMNISCIENT.

| to | cinder | | | | tarmac | | | | grassland | | | | gravel | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| from | a) | b) | c) | d) | a) | b) | c) | d) | a) | b) | c) | d) | a) | b) | c) | d) |
| cinder | - | - | - | - | 0.043 | 0.043 | 0.042 | 0.035 | 0.087 | 0.077 | 0.063 | 0.039 | 0.063 | 0.061 | 0.058 | 0.046 |
| | - | - | - | - | 0.041 | 0.043 | 0.042 | 0.037 | 0.271 | 0.226 | 0.162 | 0.067 | 0.164 | 0.147 | 0.126 | 0.099 |
| tarmac | 0.059 | 0.055 | 0.056 | 0.041 | - | - | - | - | 0.060 | 0.057 | 0.051 | 0.039 | 0.054 | 0.054 | 0.057 | 0.046 |
| | 0.107 | 0.100 | 0.091 | 0.030 | - | - | - | - | 0.204 | 0.172 | 0.129 | 0.067 | 0.113 | 0.111 | 0.109 | 0.099 |
| grass | 0.108 | 0.093 | 0.085 | 0.041 | 0.071 | 0.064 | 0.060 | 0.035 | - | - | - | - | 0.063 | 0.060 | 0.059 | 0.046 |
| | 0.372 | 0.313 | 0.231 | 0.030 | 0.256 | 0.214 | 0.154 | 0.037 | - | - | - | - | 0.165 | 0.151 | 0.133 | 0.099 |
| gravel | 0.060 | 0.056 | 0.055 | 0.041 | 0.045 | 0.044 | 0.046 | 0.035 | 0.064 | 0.060 | 0.057 | 0.039 | - | - | - | - |
| | 0.191 | 0.166 | 0.134 | 0.030 | 0.090 | 0.079 | 0.066 | 0.037 | 0.136 | 0.119 | 0.098 | 0.067 | - | - | - | - |

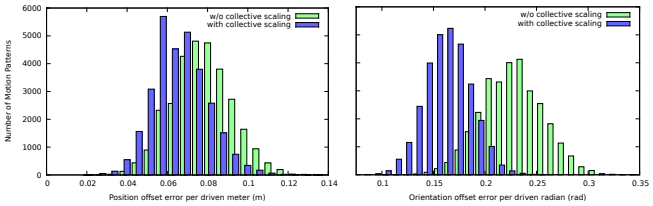


Fig. 5. Result of a simulation: The position (left) and orientation (right) error distribution of a Teodor robot changing from cinder to grassland without and with collective scaling.

TABLE II

THE POSITION (FIRST ROW) AND ORIENTATION (SECOND ROW) ERROR REDUCTIONS OF THE TRANSITION EXPERIMENTS (99% CONFIDENCE).

| transition from / to | cinder | tarmac | grassland | gravel |
|---|---|---|---|---|
| cinder | - | 1.5 % | 17.8 % | 3.4 % |
| | - | 18.5 % | 27.9 % | 13.5 % |
| tarmac | -0.5 % | - | 10.3 % | -4.9 % |
| | 8.5 % | - | 24.9 % | 1.2 % |
| grassland | 9.1 % | 5.0 % | - | -1.1 % |
| | 26.0 % | 27.8 % | - | 7.3 % |
| gravel | 0.3 % | -5.4 % | 4.5 % | - |
| | 18.4 % | 15.6 % | 17.2 % | - |

the deviation of the simple method.

## VI. SUMMARY AND CONCLUSION

In this paper we presented an adaptive navigation system based on predefined motion templates called Motion Patterns. The system has the ability to incorporate the actual robot movement into the Motion Patterns. The updating process is not limited to the actually driven Motion Pattern. In most cases the system is also able to derive adjustment information for all other patterns as well. The soundness of our approach has been shown in a simulation study using real-word data. The system proved it can efficiently learn the robot's behavior after transitions between different surface types while outperforming the previous approach without collective scaling. Future work will focus on further improving the performance of the motion learning and adapting mechanisms. Decomposing a Motion Pattern's recorded trajectory in a series of straight lines and connection angles could lead to an improved geometrical understanding.

**Acknowledgements:** The presented system is part of the project RoboGasInspector, funded by the German Federal Ministry of Economics and Technology based on a resolution of the German Bundestag.

## REFERENCES

[1] C.A. Brooks and K. Iagnemma. Vibration-based terrain classification for planetary exploration rovers. *Robotics, IEEE Transactions on*, 21(6):1185 – 1191, December 2005.

[2] M. Castelnovi, R. Arkin, and T.R. Collins. Reactive speed control system based on terrain roughness detection. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 891 – 896, April 2005.

[3] E. Frazzoli, M.A. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *Robotics, IEEE Transactions on*, 21(6):1077 – 1091, Dec. 2005.

[4] F. Hoeller, T. Röhling, and D. Schulz. Offroad navigation using adaptable motion patterns. In *International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, June 2009.

[5] A. Martinelli, N. Tomatis, A. Tapus, and R. Siegwart. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1499 – 1504 vol.2, Oct. 2003.

[6] J. Morimoto and C.G. Atkeson. Nonparametric representation of an approximated poincaré map for learning biped locomotion. *Auton. Robots*, 27(2):131–144, 2009.

[7] J. Morimoto, J. Nakanishi, G. Endo, G. Cheng, and C. G. Atkeson. Poincare-map-based reinforcement learning for biped walking. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 2381–2386, 2005.

[8] G. Neumann, W. Maass, and J. Peters. Learning complex motions by sequencing simpler motion templates. In *Proc. of the International Conference on Machine Learning (ICML)*, 2009.

[9] Neal Seegmiller, Forrest Rogers-marcovitz, Greg Miller, and Alonzo Kelly. A unified perturbative dynamics approach to online vehicle model identification.

[10] D. Stavens, G. Hoffmann, and S. Thrun. Online speed adaptation using supervised learning for high-speed, off-road autonomous driving. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[11] D. Stavens and S. Thrun. A self-supervised terrain roughness estimator for off-road autonomous driving. In *Proc. Conference on Uncertainty in AI (UAI)*, pages 13–16, 2006.

[12] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23:661–692, 2006.