2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)
Beijing, China, November 6-9, 2018

Learning Efficient Omni-directional Capture Stepping for Humanoid Robots from Human Motion and Simulation Data

Johannes Pankert, Lukas Kaul and Tamim Asfour

***Abstract*— Two key questions in the context of stepping for push recovery are where to step and how to step there. In this paper we present a fast and computationally light-weight approach for capture stepping of full-sized humanoid robots. To this end, we developed an efficient parametric step motion generator based on dynamic movement primitives (DMPs) learnt from human demonstrations. Simulation-based reinforcement learning (RL) is used to find a mapping from estimated push parameters (push direction and intensity) to step parameters (step location and step execution time) that are fed to the motion generator. Successful omni-directional capture stepping for 89% of the test cases with pushes from various directions and intensities is achieved with minimal computational effort after 500 training iterations. We evaluate our method in a dynamic simulation of the ARMAR-4 humanoid robot.**

I. Introduction

One of the fundamental requirements for successful operation of humanoid robots in the real world is robustness against external disturbances such as pushes. Several balancing strategies are available, but in case of a strong enough push, taking a step is the most promising way to maintain balance. The stepping trajectory in this case has to be generated fast and needs to be adapted to the intensity and direction of the applied push to guarantee effective push recovery. In this paper we address both, fast motion generation and adaptation, and we evaluate it in a dynamic simulation of the ARMAR-4 robot. An overview of the presented work is shown in Fig. 1. Our approach stands out among other works in the literature due to its computational efficiency and the intrinsic human-likeness of the generated stepping motions.

We start with human motion recordings of recovery steps, adapt them to the kinematics of the ARMAR-4 robot, (see [1]) and learn joint-level DMPs [2] from those demonstrations. The DMPs are encapsulated in a parametric step-trajectory generator. If the robot needs to step the trajectory generator is called with a set of push parameters that the robot can sense. The low-dimensional mapping from push to step parameters can be efficiently learned using reinforcement learning.

Converting human motions to dynamically stable robot trajectories is generally difficult due to the complex dynamics of bipedal robots. Some more recent works have addressed this problem with machine learning techniques, requiring a very large number of training iterations (e. g. [3]). In contrast to these methods, our approach reduces the problem of motion generation to finding a small set of high-level parameters such as the step goal and the execution speed of the step, making the problem much more tractable.

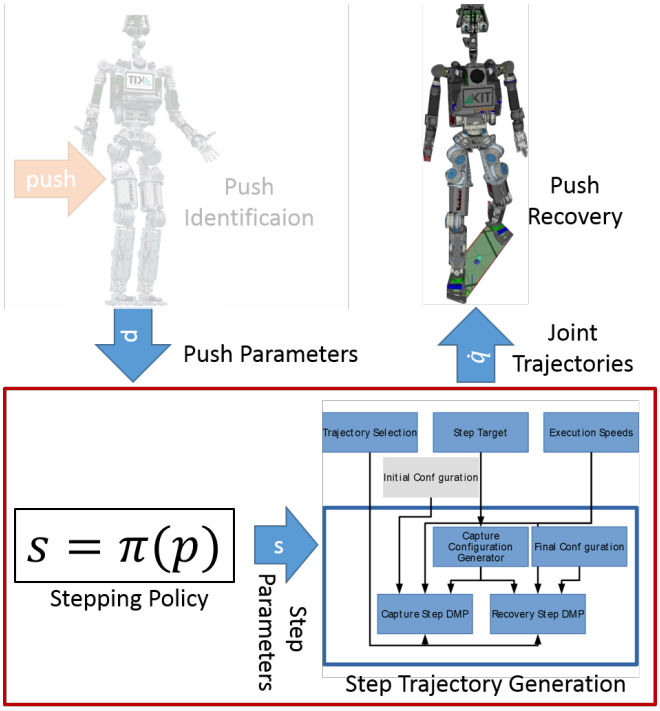


Fig. 1: The Recovery Stepping Pipeline. This paper focuses on the policy that maps estimated push parameters p to step parameters s and on a DMP-based step trajectory generator that turns the step parameters into stepping motions (red box). Evaluation is carried out in simulation. An enlarged version of the step trajectory generator is depicted in Fig. 2.

We show that our approach enables omni-directional human-like capture stepping on a humanoid robot coping with pushes of varying intensities. A successful parameter mapping can be learned in simulation with RL starting from very few initial training examples. The proposed approach is shown to be computationally light-weight and thus suited to be implemented on real robotic hardware.

II. Related Work

Balancing and push recovery has been a topic of ongoing research for decades. Within this broad field, different approaches can be classified into three categories or combinations thereof, namely *ankle-*, *hip-* and *stepping-* strategies.

The authors are with the High Performance Humanoid Technologies Lab, Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology (KIT), Germany, {lukas.kaul, asfour}@kit.edu, johannes@pankert.eu

This work has been supported by the German Federal Ministry of Education and Research (BMBF) under the project INOPRO (16SV7665).

This classification was introduced in the biomechanics literature in [4] and adopted by the robotics community (e. g. in [5]). Our work presented here exclusively focuses on stepping strategies.

In a widely adopted way of formalizing robotic step strategies for push recovery, the term *Capture Point* was coined by Pratt et al. in [6]. The capture point denotes the point on the ground on which the robot needs to step to come to rest. Its location can be analytically derived for the linear inverted pendulum model (LIPM). Extensions and similar formalizations were presented under the names XcoM [7], ICP [8] and 3D DCM [9]. Despite their reliance on drastically simplified dynamic models, Capture Point methods have successfully been applied to push recovery and walking [10], [11]. Recently, a method for changing the locomotion from walking or running to hopping was introduced that demonstrated push-recovery by single-legged hopping [12]. As one way of bridging the gap between planning and control methods derived from simplified abstractions and more complex models, learning techniques have been proposed for finding control policies for walking [13], postural balancing [14] or suitable capture points for a 3-dimensional robot model [15]. This particular work is closely related to ours, and we adopt their notion of the *Stopping Energy* as a quality measure of push recovery attempts in the reward function of our RL-routine.

Dynamic Movement Primitives (DMPs) are a way to encode trajectories as dynamical systems [2]. DMPs have been widely used in robotic applications due to their virtue of easy parameterization and fast motion adaptation [16], [17]. In the presented work we use the DMP formulation established in [18] with a *transformation system* for each robot joint and a shared, time-independent *canonical system*.

Ankle, hip and step strategies for push recovery based on DMPs have been investigated in theory and application to a small PKU-HR5 humanoid robot in [19]. In contrast to our work, the authors used a task-space DMP to model end-effector trajectories. With this approach, the joint angles have to be calculated by solving the inverse kinematics at every time-step, which is computationally expensive for more complex humanoids. Another differentiation to our work is that the authors only investigate pushes from the back, whereas our method generalizes to pushes from different directions.

III. Approach

We propose a capture step generator based on step trajectories from human motion-capture recordings that takes as input a set of *step parameters* (see Fig. 2). The demonstration motions are encoded as joint-level DMPs. A reinforcement learning procedure is used to learn appropriate step parameters from *push parameters* that can be estimated by the robot. The learning was executed in a dynamic physics simulation of the 63 DoF ARMAR-4 robot in which arbitrary pushes from every direction can be applied.

A. Step Trajectory Generator

Joint-angle DMPs are created based on trajectories acquired in human motion-capture demonstrations. The acquisition of these demonstration trajectories is described in section IV. We use the DMPLib C++ library[1] which, among others, provides an implementation of the DMP formulation presented in [18]. The step is divided into two segments conceptually: The *lunge step* is the placement of the first foot after the push has been applied. During the *recovery step* the other foot follows such that the robot recovers to its original posture. Separate DMPs are used to represent both step segments.

Using the spatial scaling properties of the DMPs, the origin of the lunge step is set to the robot's current joint angle configuration, making it independent of the robot's initial state. The capture DMP's goal is the *capture configuration*. The capture configuration depends on the desired foot location at the end of the capture step and is computed by solving the inverse kinematics using a constrained IK solver. The origin of the recovery step DMP is the capture configuration. The goal configuration is the initial configuration of the lunge step, driving the robot back to its stable initial pose at a new location.

Both DMPs, the IK to generate the capture configuration, and the final configuration are encapsulated within a *step trajectory generator* implemented in C++ as a component of the ArmarX [20] robot development environment. Fig. 2 depicts the structure of this generator.

1) Step Parameters: Trajectories for steps into all directions are generated according to a small set of high-level parameters: The desired foot location (the step target) is parametrized by a *step angle* and a *step length* parameter. With the *capture step duration* and the *recovery step duration* the execution speeds of both parts are controlled. The underlying DMPs are temporally rescaled accordingly. Based on the trajectory selection parameters *selection length* and *selection angle*, the DMP with parameters closest to the selection parameters is chosen as a basis for the generated step. Choosing DMP selection parameters independently of the step parameters allows mitigation of otherwise problematic discontinuities at the decision boarders by the RL algorithm. We call these six parameters the *step parameters*.

2) Push Parameters: The applied pushes were characterized with the two parameters (normalized) *push impulse* and direction. The normalized push impulse is computed as the integral over the applied push force $F(t)$ normalized by the subject's mass m to achieve better comparability between different robots:

$$p = \frac{1}{m} \int_{t_{start}}^{t_{stop}} F(t)dt \quad [\mathrm{Ns/kg}] \tag{1}$$

The second parameter, the *push angle*, describes the direction of the push with respect to the pushed subject.

[1] https://gitlab.com/h2t/DynamicMovementPrimitive

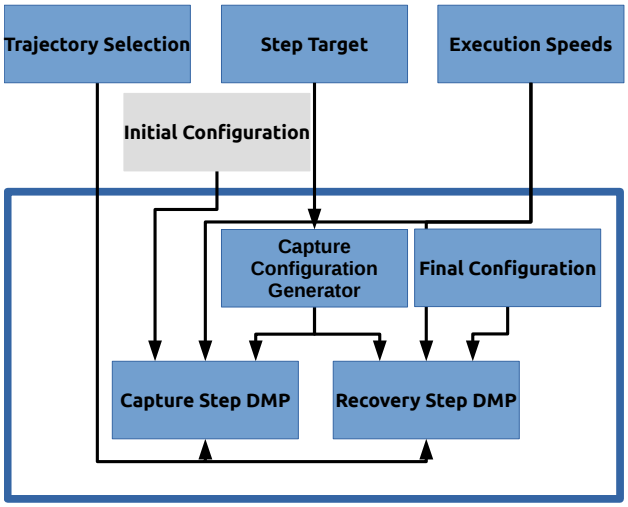


Fig. 2: The blue frame contains the parameterizable step trajectory generator. It consists of the capture and recovery step DMP, a stable final configuration and the capture configuration generator that converts step targets to joint angle configurations. The parameters are the step target, the execution speeds and the trajectory selection parameters. The initial configuration is neither part of the trajectory generator nor a parameter to be chosen.

3) Ankle Control: The motion capture trajectories recorded for the ankle joints were not directly suitable to train the DMPs on. Retrieving high quality ankle trajectories from motion capture recordings is known to be difficult for rigid skeleton models with relative small feet [21]. We therefore discarded the recorded ankle trajectories and actively control the ankle joint using the following strategy: For the swing foot, the ankle joints are driven such that the foot is aligned in parallel with the ground. The stance foot's ankle joints are driven along linear trajectories from the origin to the target configuration (both during the lunge and the recovery step).

B. Policy Learning

Taking the right step with the right speed is achieved by means of a learned policy that maps push parameters to step parameters, $\pi(p) = s$. $p \in P \subset \mathbb{R}^2$ is a push parameter vector and $s \in S \subset \mathbb{R}^6$ a step parameter vector. The problem of finding an optimal mapping can be formulated as:

$$\text{Find the optimal policy} \quad \pi^* : P \to S \tag{2}$$

$$\text{that minimizes} \quad J(\pi) = \iint_P E(p, \pi(p))dp \tag{3}$$

$$\text{with} \quad P \mathrel{\widehat{=}} \text{space of push parameters}$$
$$S \mathrel{\widehat{=}} \text{space of step parameters}$$
$$E \mathrel{\widehat{=}} \text{error weight function}$$

In our implementation we use Alg. 1 to learn the policy function π.

For learning we use an initial list of *training examples* L_0 as an input. Training examples are pairs of push and step parameters (p, s) for which push-recovery experiments have successfully been performed. These examples were derived from human motion capture experiments[2]. For these examples, push experiments are performed.

Algorithm 1 Algorithm to find a policy that maps push parameters to step parameters

function LEARNPOLICY(L_0)
  $R_0 \leftarrow$ PUSHRECOVERYTESTING(L_0)
  $successRate \leftarrow 1$
  for $i \leftarrow 1 \dots N$ **do**
    $\pi_i \leftarrow$ TRAIN(L_{i-1}, R_{i-1})
    $\{\tilde{p}_1, \tilde{p}_2, \dots, \tilde{p}_{10}\} \leftarrow$ EXPLORE($L_{i-1}, successRate$)
    $\tilde{L} \leftarrow \{(\tilde{p}_1, \pi_i(\tilde{p}_1)), (\tilde{p}_2, \pi_i(\tilde{p}_2)), \dots, (\tilde{p}_{10}, \pi_i(\tilde{p}_{10}))\}$
    $\tilde{R} \leftarrow$ PUSHRECOVERYTESTING($\tilde{L}$)
    $successRate \leftarrow$ MEAN($\tilde{R}.success$)
    $L^+ \leftarrow \tilde{L}$ *where* $(\tilde{R}.success = true)$
    $R^+ \leftarrow \tilde{R}$ *where* $(\tilde{R}.success = true)$
    $L_i \leftarrow \{L_i, L^+\}$
    $R_i \leftarrow \{R_i, R^+\}$
  end for
  $\pi_{final} \leftarrow$ TRAIN(L_N, R_N)
  return π_{final}
end function

Two *performance criteria* describe the outcome of a push experiment. The first simple standing/not standing criterion is evaluated 3 s after the push. A second (non-binary) indicator is the *Stopping Energy* as proposed in [15]:

$$E_{stopping} := \frac{1}{2} v_{COM}^2 + \frac{1}{2} \frac{g}{l} |\vec{x}|^2 \tag{4}$$

The first term represents the massless kinetic energy. v_{COM} is the velocity of the robot's center of mass (COM). The second term can be interpreted as a form of potential energy. $|\vec{x}|$ is the distance between the centroid of the robot's support polygon and the floor projection of the COM. $E_{stopping} = 0$ means that the robot is statically stable. The performance indicators r are saved in the list R_0.

L_0 and R_0 are the initial inputs for the following learning iterations: In each iteration we train a function approximator as a regressor to the correct parameter mapping based on all available training examples L_i, considering the performance indicators R_i. We use a feed-forward neural network with 4 hidden-layers as an approximator. Each layer has 10 neurons with sigmoid activation functions. The input- and output-layers match the dimensionality of P and S. Linear activation functions are used in the output-layer. The neural network trained in iteration i of the learning algorithm is the current policy π_i.

Our network architecture is relatively simple compared to those used in the deep learning literature such as [22] and

[2]Since a human recovering from a push usually combines different balancing techniques such as hip, ankle and step strategies but our approach solely relies on capture steps, we had to reduce the applied push impulses to obtain valid training examples. The reduction factor varied for different directions and was in the range of 25 % to 80 %.

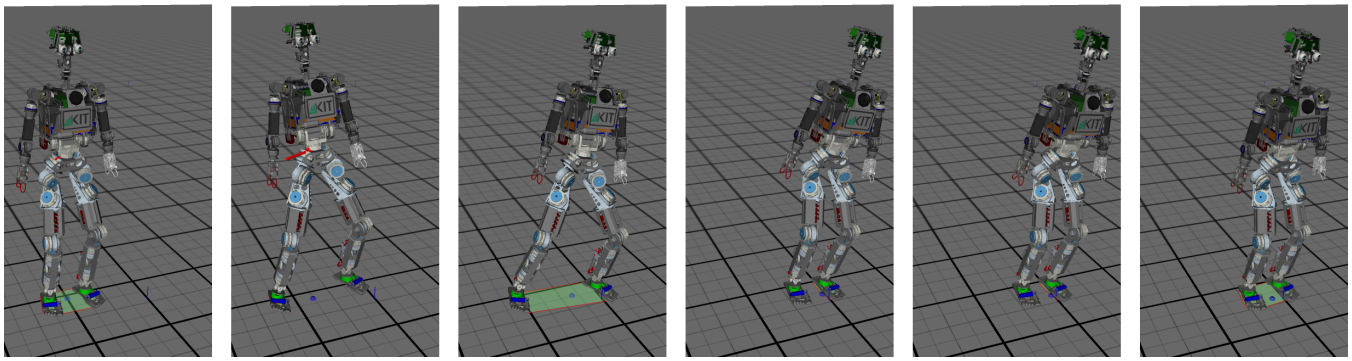


Fig. 3: ARMAR-4 recovering from a push applied from the side (red arrow in the second image). Temporal order of the images from left to right 0.32 s spacing. The green area is the robot's current support polygon. The turquoise and blue dots are the Zero Moment Point and the floor projection of the robot's Center of Mass.

[23]. This is a virtue of the fact that we learn a step parameter vector instead of the entire step trajectory, which greatly reduces the dimensionality of the problem to an extent that such a simple neural network is sufficient. More complex network designs would increase the risk of overfitting or require more data to train.

The error function $E(p, s)$ is used to weight the training samples depending on the performance criterion:

$$E(p, s) := E_s(p, s) + E_{distance}(p) \tag{5}$$

with

$$E_s(p, s) := \text{MINMAX}(-\log(E_{stopping}(p, s))) \tag{6}$$

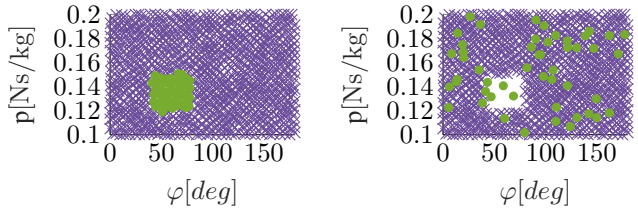
$$E_{distance}(p) := \text{MINMAX}(d_{nearest\ neighbor}(p))) \tag{7}$$

$E_s(p, s)$ promotes training examples with a low Stopping Energy. $E_{distance}(p)$ endorses samples in areas of the P space with a low density of training examples to ensure solitary positive samples in the P-space are not disregarded in the regression.

At each iteration, 10 new push parameters are selected depending on the last iteration's success rate. If the rate is high, push parameters in areas of the P space that are sparsely populated with training examples will be selected to drive exploration. This is done by randomly generating large sets of parameters and selecting those with the highest nearest neighbor distances to the existing samples. If the success rate is low, the push parameters are distributed randomly in the P space. The proximity of the new parameters to the existing samples makes it likely for the selected pairs to be successful, the 'knowledge' of the current policy is exploited. Figure 4 visualizes the selection process for the *exploitation* and *exploration* case.

The current policy π_i is applied to the newly generated push parameters. The ten pairs $(p, s = \pi_i(p))_{1...10}$ are evaluated in the simulation environment. Successful pairs and their performance indicators are added to the list of training examples. After N iterations, the algorithm terminates and returns the final policy, trained on the complete list of training examples.

During the first iterations of the algorithm, the function approximator has to be trained on relatively few examples. To increase the number of training examples in the first 50 iterations, we also include push experiments that failed (i. e. the robot was not standing after 3 s) but in which both steps were executed before the robot fell. After these 50 iterations of 'warm-starting' the learning, the failed experiments are removed from the list of positive examples. An additional bias is set in the evaluation function for the truly successful experiments to favor them over the failed experiments:



(a) Exploration (b) Exploitation

Fig. 4: Generation of new push parameters for exploration and exploitation. Purple crosses represent the existing samples L. Orange dots are the newly created samples.

$$\tilde{E}(p, s) := E(p, s) + E_{success}(p, s) \quad \text{with} \tag{8}$$

$$E_{success}(p, s) := \begin{cases} 5 & \text{if } \tilde{R}(\tilde{p}, \pi_i(\tilde{p})).success = \text{true} \\ 0 & \text{else} \end{cases} \tag{9}$$

IV. Experimental Setup

The proposed step trajectory generator uses DMPs based on human sample motions. In subsection IV-A we describe the experimental setup needed to record those demonstration. The reinforcement learning algorithm interacts with a simulated world to learn appropriate capture steps. Subsection IV-B illustrates the environment we set up to evaluate the performance of the generated step trajectories and the learning procedure.

A. Human Motion Data Collection

Human capture steps were recorded with a marker-based VICON motion capture system. Fig. 5 shows an example

Fig. 5: Push-Recovery experiment in the motion capture lab. Temporal order of the images from left to right $0.2\,\mathrm{s}$ spacing.

of the data collection procedure. Subject B (on the right) pushes subject A (on the left) with an apparatus recording the push force. Subject A always takes a capture step consisting of a lunge and a recovery step, and comes to a still stand in the initial posture. Pushes were applied with varying intensities and from different angles, forcing subject A to take steps of different lengths into different directions. White tape markings on the floor guide subject B in choosing the planned push angles. Pushes were applied from the angles $\phi \in \{0°, 30°, 45°, 60°, 90°, 120°, 135°, 150°, 180°\}$ and with two different intensities (*soft* and *hard*), aimed at provoking steps with lengths of approximately $30\,\mathrm{cm}$ and $50\,\mathrm{cm}$, resulting in 18 different experiments. The sector $(180°, 360°)$ was omitted because of the human body's symmetry with respect to the sagittal plane. The motion capture recordings are publicly available as part of the KIT Whole-Body Human Motion Database [24] with the subject ID #1721 [3].

The VICON system records marker-positions in Cartesian coordinates. They are converted to joint-angle trajectories of the Master Motor Map (MMM) human reference model with tools available from the MMM framework [25]. These reference model trajectories are then converted to joint-angle trajectories for the ARMAR-4 robot [1]. Of the ARMAR-4's 63 actuated degrees of freedom we consider the 14 joints in our conversion that comprise the two legs and the torso since we exclusively focus on stepping motions.

B. Dynamic Simulation

Push experiments were performed in the ArmarX Simulation environment. ArmarX Simulation encapsulates the Bullet physics library [26]. The trajectory generator step size was set to $10\,\mathrm{ms}$. The inner simulation loop operates with a step size of $0.5\,\mathrm{ms}$.

The majority of the force profiles measured during the motion capture push experiments roughly revealed an isosceles triangular shaped time-force profile. Fig. 6 shows examples of time-force profiles for different recorded pushes of the type 'soft'. The duration of the pushes shows very little variation, whereas the peak force depends on the intensity of the applied push. Based on these findings we chose to implement isosceles triangular shaped force profiles with a duration of $0.5\,\mathrm{s}$ for our simulation experiments, mimicking the real-world conditions.

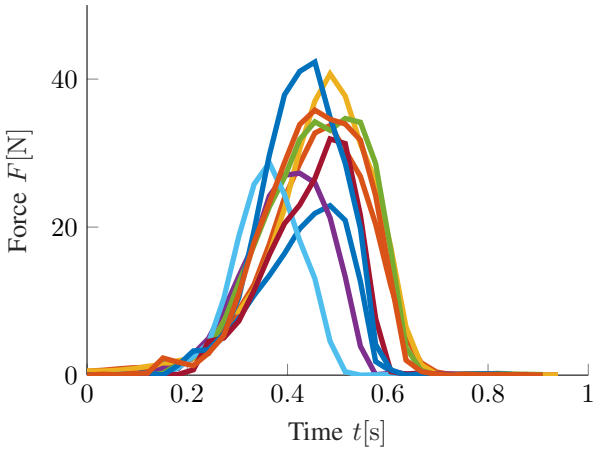


Fig. 6: Multiple force recordings during push trials. The recorded force profiles resemble isosceles triangles and all have similar durations. The effective push impulse depends on the maximum of the push force.

The step trajectory generator and the dynamic simulation engine can be interfaced externally. They take a set of push and step parameters as input, simulate a push and the robot's motion and return a set of performance indicators r for the push experiment.

V. Evaluation

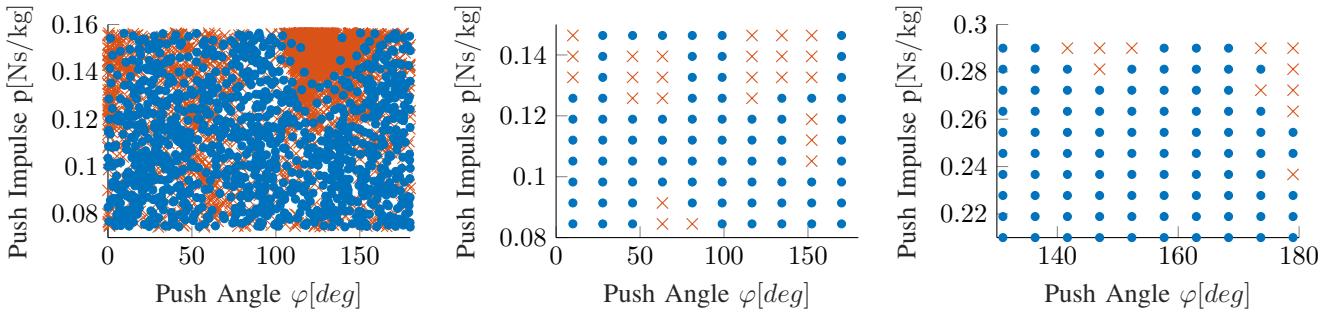
After 500 learning iterations, ARMAR-4 was able to perform successful capture steps in all directions to recover from pushes with varying intensities. The attached video shows several simulated push-recovery experiments[4], all of which are consistent with the velocity capabilities of the real robot. Fig. 3 shows an exemplary capture step sequence. In subsection V-A we will discuss the evaluation of the step-parameter learning routine and in subsection V-B the computation speed of the approach.

A. Learned Step Parameters

For nine pairs of push and step parameters (p, s) derived from the motion-capture recordings, initial successful push recovery experiments could be performed in simulation (see

[3] https://motion-database.humanoids.kit.edu/list/motions/?subjects=1721

[4] https://youtu.be/eLVUd1DdTOo



(a) P space after 500 training iterations

(b) Validation, motion capture derived initial training examples.

(c) Validation, hand picked initial training examples (harder pushes).

Fig. 7: Successful (blue) and unsuccessful (red) push experiments in the push parameter space

subsection III-B). In nine other cases, the robot executed both steps but fell afterwards. Those 18 examples were the initial training data. The impulses of these 18 pushes were in the range $[0.75\,\mathrm{Ns/kg}, 0.151\,\mathrm{Ns/kg}]$ and the angles spread across $[0°, 180°]$. Alg. 1 was executed for 500 iterations. Fig. 7a shows successful and failed experiments performed during training in the 2D space of push parameters (the P space).

To evaluate the learned policy, a 10×10 grid of push parameters was spanned in the P space. The final policy was applied to all grid points and the resulting pairs were evaluated in the dynamic simulation environment. Fig. 7b shows the results in the P domain. A success rate of $\mathbf{75\,\%}$ could be achieved.

The developed algorithm operates in a rectangular subset of the P domain that encloses the initial training examples. Since the motion-capture-derived push impulses are relatively low, the learned policy does not perform well for stronger pushes. To evaluate the capabilities of learning push recovery for stronger pushes, we ran the algorithm on a set of five manually selected initial parameter-vector pairs $(p, s)_{init}$. The impulses of these training examples were in the range $[0.2\,\mathrm{Ns/kg}, 0.3\,\mathrm{Ns/kg}]$ in good accordance with the impulses measured in the motion capture experiments (the mean recorded push impulse was $0.277\,\mathrm{Ns/kg}$). To reduce the required training time, we chose the initial training parameters from the exemplary push angle range $[130°, 180°]$.

After 500 training iterations the performance of the final policy was evaluated on a 10×10 grid. Fig. 7c shows the evaluation result. The push recovery attempt was successful in $\mathbf{89\,\%}$ of the test cases. In an intermediate step we evaluated the policy π_{200} based on the first 200 training iterations on the same grid, resulting in a success rate of $69\,\%$. As expected the developed algorithm improves the stepping policy over the course of the learning process.

B. Computational Performance

To evaluate the computation speed of the DMP method, we measured the time it takes to compute the joint angles for one DMP trajectory time step. The differential equations of the transformation systems and the canonical system were solved for one iteration with an Euler 1-step integrator. The right column of Tab. I lists the mean execution times over 10 steps for 12 different capture step DMPs. The mean execution time is $195.11\,\mu s$ and the standard deviation is $26.56\,\mu s$. The measurements were performed on a Desktop PC with an Intel® Core™ i7-4790 CPU running a non-realtime Linux OS. The fast execution times suggest that a realtime implementation with high sampling rates is feasible.

We compare the DMP computation times to the time it takes to solve a constrained IK problem. A fixed-pose constraint was enforced on the stance foot while the swing foot was made to follow a predefined trajectory. The left column of Tab. I displays the measured times for IK solution steps in 12 different stepping trajectories. The mean execution time is $2507.25\,\mu s$ and the standard deviation is $2590.05\,\mu s$. The mean execution time is about 10 times greater than that of the DMP computation. The high standard deviation is unfavorable for realtime implementations.

TABLE I: Execution time for solving a constrained inverse kinematics (IK) problem compared to stepping the transformation systems of a DMP once.

$\mathbf{T_{ConstrainedIK}}$ [µs]	$\mathbf{T_{Euler\ 1-Step}}$ [µs]
1198	184.7
6244	184.7
314	185.7
5300	189
532	183.5
738	189.2
1259	190.6
6376	279.1
329	191.3
5960	188.6
577	186.7
1260	188.2

VI. Conclusion

We presented a system that can produce dynamically consistent stepping motions for push recovery of the ARMAR-4 humanoid robot from estimated push parameters. The system consists of two components, a learned policy that maps push parameters (direction and intensity) to step parameters

(step location and step execution time), and a step trajectory generator that produces joint-level trajectories from the step parameters. The parameter mapping is represented as feed-forward neural network and trained in a dynamic simulation using reinforcement learning, initialized on very few initial examples. The trajectory generator is based on Dynamic Movement Primitives derived from human demonstrations of push recovery actions and allows generating stepping motions in arbitrary directions.

Our results show that the learned parameter mapping is successful in 89% of our trials for harder pushes after as little as 500 training iterations. This efficiency is achieved by reducing the dimensionality to a small number of parameters and and delegating the rest to the trajectory generator. Since the trajectory generator is based on joint-level DMPs (rather than on end-effector DMPs) it requires very little computation time. Our evaluation suggests a more than 10-times speed-up over deriving the joint trajectories with a constrained IK.

A. Future Work

Our method assumes that push intensity and direction are provided. We are working on the online estimation of these parameters with ARMAR-4's integrated 6 DOF force-torque sensors in the ankles, extending our earlier IMU-based work [27] and completing the pipeline shown in Fig. 1.

During the learning process we could observe phases in which samples from an area of the P space were selected repeatedly, without noticeably improving the policy (see Fig. 7a, top right corner). For those cases, adding noise to the predicted step-parameter vector could be helpful to find a suitable parametrization and could be explored in future work. In addition, we plan to apply ankle or hip balancing after the recovery step to mitigate the inevitable gap between simulation and robot experiments.

REFERENCES

[1] T. Asfour, J. Schill, H. Peters, C. Klas, J. Bücker, C. Sander, S. Schulz, A. Kargov, T. Werner, and V. Bartenbach, "ARMAR-4: A 63 DOF torque controlled humanoid robot," in *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Oct. 2013, pp. 390–396.

[2] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 2, 2002, pp. 1398–1403.

[3] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills," *arXiv:1804.02717 [cs]*, Apr. 2018.

[4] F. B. Horak and L. M. Nashner, "Central programming of postural movements: adaptation to altered support-surface configurations," *Journal of neurophysiology*, vol. 55, no. 6, pp. 1369–1381, 1986.

[5] Z. Aftab, T. Robert, and P.-B. Wieber, "Ankle, hip and stepping strategies for humanoid balance recovery with a single model predictive control scheme," in *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*. IEEE, 2012, pp. 159–164.

[6] J. Pratt, J. Carff, S. Drakunov, and A. Goswami, "Capture Point: A Step toward Humanoid Push Recovery," in *2006 6th IEEE-RAS International Conference on Humanoid Robots*, Dec. 2006, pp. 200–207.

[7] A. L. Hof, "The extrapolated center of mass concept suggests a simple control of balance in walking," *Human movement science*, vol. 27, no. 1, pp. 112–125, 2008.

[8] T. Koolen, T. De Boer, J. Rebula, A. Goswami, and J. Pratt, "Capturability-based analysis and control of legged locomotion, part 1: Theory and application to three simple gait models," *The International Journal of Robotics Research*, vol. 31, no. 9, pp. 1094–1113, 2012.

[9] J. Englsberger, C. Ott, and A. Albu-Schäffer, "Three-dimensional bipedal walking control based on divergent component of motion," *IEEE Transactions on Robotics*, vol. 31, no. 2, pp. 355–368, 2015.

[10] B. J. Stephens and C. G. Atkeson, "Push recovery by stepping for humanoid robots with force controlled joints," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference On*. IEEE, 2010, pp. 52–59.

[11] J. Englsberger, C. Ott, M. A. Roa, A. Albu-Schäffer, and G. Hirzinger, "Bipedal walking control based on capture point dynamics," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 4420–4427.

[12] T. Kamioka, H. Kaneko, M. Kuroda, C. Tanaka, S. Shirokura, M. Takeda, and T. Yoshiike, "Dynamic gait transition between walking, running and hopping for push recovery," in *Humanoid Robotics (Humanoids), 2017 IEEE-RAS 17th International Conference on*. IEEE, 2017, pp. 1–8.

[13] R. Tedrake, T. W. Zhang, and H. S. Seung, "Learning to walk in 20 minutes," in *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*, vol. 95585. Yale University New Haven (CT), 2005, pp. 1939–1412.

[14] S.-J. Yi, B.-T. Zhang, D. Hong, and D. D. Lee, "Learning full body push recovery control for small humanoid robots," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2047–2052.

[15] J. Rebula, F. Canas, J. Pratt, and A. Goswami, "Learning Capture Points for humanoid push recovery," in *2007 7th IEEE-RAS International Conference on Humanoid Robots*, Nov. 2007, pp. 65–72.

[16] D. H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," in *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*, Dec. 2008, pp. 91–98.

[17] Y. Zhou and T. Asfour, "Task-Oriented Generalization of Dynamic Movement Primitive," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[18] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[19] D. Luo, X. Han, Y. Ding, Y. Ma, Z. Liu, and X. Wu, "Learning push recovery for a bipedal humanoid robot with Dynamical Movement Primitives," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, Nov. 2015, pp. 1013–1019.

[20] N. Vahrenkamp, M. Wächter, M. Kröhnert, K. Welke, and T. Asfour, "The robot software framework ArmarX," *it - Information Technology*, vol. 57, no. 2, Jan. 2015.

[21] L. Kovar, J. Schreiner, and M. Gleicher, "Footskate Cleanup for Motion Capture Editing," in *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '02. New York, NY, USA: ACM, 2002, pp. 97–104.

[22] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills," *arXiv:1804.02717 [cs]*, Apr. 2018, arXiv: 1804.02717. [Online]. Available: http://arxiv.org/abs/1804.02717

[23] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 41, 2017.

[24] C. Mandery, Ö. Terlemez, M. Do, N. Vahrenkamp, and T. Asfour, "The KIT whole-body human motion database," in *2015 International Conference on Advanced Robotics (ICAR)*, Jul. 2015, pp. 329–336.

[25] C. Mandery, O. Terlemez, M. Do, N. Vahrenkamp, and T. Asfour, "Unifying representations and large-scale whole-body motion databases for studying human motion," *IEEE Transactions on Robotics*, vol. 32, no. 4, pp. 796–809, 2016.

[26] E. Coumans, "Bullet physics library," *Open source: bulletphysics. org*, vol. 15, p. 49, 2013.

[27] L. Kaul and T. Asfour, "Human push-recovery: Strategy selection based on push intensity estimation," in *International Symposium on Robotics (ISR)*, 2016, pp. 547–554.