Hybrid autonomous controller for bipedal robot balance with deep reinforcement learning and pattern generators

Robotics and Autonomous Systems 146 (2021) 103891

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Hybrid autonomous controller for bipedal robot balance with deep reinforcement learning and pattern generators

Christos Kouppas [a,*], Mohamad Saada [a], Qinggang Meng [a], Mark King [b], Dennis Majoe [c]

[a] Department of Computer Science, Loughborough University, Loughborough, UK
[b] School of Sport, Exercise and Health Sciences, Loughborough University, Loughborough, UK
[c] Motion Robotics LTD, Southampton, UK

ARTICLE INFO

Article history:
Received 4 December 2020
Received in revised form 20 August 2021
Accepted 30 August 2021
Available online 15 September 2021

Keywords:
Bipedal robot
Pattern generator
Reinforcement learning
Hybrid controller

ABSTRACT

Recovering after an abrupt push is essential for bipedal robots in real-world applications within environments where humans must collaborate closely with robots. There are several balancing algorithms for bipedal robots in the literature, however most of them either rely on hard coding or power-hungry algorithms. We propose a hybrid autonomous controller that hierarchically combines two separate, efficient systems, to address this problem. The lower-level system is a reliable, high-speed, full state controller that was hardcoded on a microcontroller to be power efficient. The higher-level system is a low-speed reinforcement learning controller implemented on a low-power onboard computer. While one controller offers speed, the other provides trainability and adaptability. An efficient control is then formed without sacrificing adaptability to new dynamic environments. Additionally, as the higher-level system is trained via deep reinforcement learning, the robot could learn after deployment, which is ideal for real-world applications. The system's performance is validated with a real robot recovering after a random push in less than 5 s, with minimal steps from its initial positions. The training was conducted using simulated data.

1. Introduction

Balancing of a bipedal robot is a necessary ability that has not changed drastically over the last 35 years, using hardcoded Center of Mass (CoM) or Zero Moment Point (ZMP) algorithms [1]. Few new algorithms use Neural Networks (NN) as neural oscillators to mimic muscles, and by combining the signal of those oscillators, a balancing algorithm is achieved [2–4]. Additionally, with Reinforcement Learning (RL), the oscillators are advancing and learning within a realistic environment with a wide variety of different scenarios [5,6]. However, those algorithms need computationally heavy networks to run locally, otherwise, they cannot achieve a real-time response.

The ability to adapt in real-time to changes in the environment, is fundamental for robots to be deployed near humans as their environment can change drastically, from different floor types and slopes to the human's unexpected behaviors. Animals and humans solve this by having a higher-level system (motor cortex) that orchestrates the local motoneurons making it possible to learn patterns and execute them fast, without consuming energy from the motor cortex [7]. Neuroscientists found that humans' postural adjustments are usually handled by local reflexes (sensory neurons, interneurons, motoneurons) while feedforward adjustments are controlled from the spinal cord circuits [8]. The spinal cord circuits tune the reflexes through experiences, and with suitable signals from the spinal cord circuits, the reflexes complete tasks autonomously.

In this paper, a hybrid autonomous onboard, low-power controller is presented for robot balancing. This controller has pre-deployment and post-deployment learning capabilities that are based on humans' and animals gait control. The controller is a hierarchical Central Pattern Generator (CPG) that is divided into two independent systems, in contrast to similar approaches from other researchers where they are combined into a single controller [9–11]. The higher-level system processes all sensors' information to produce parameters for the lower-level system. A lower-level system is a group of pattern generators that utilize local information to adjust their local actuators based on the parameters from the higher-level controller. Those systems can work independently where the top part of the CPG can learn slowly through time-consuming training sessions, whilst

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: C.Kouppas@lboro.ac.uk (C. Kouppas), M.Saada@lboro.ac.uk (M. Saada), Q.Meng@lboro.ac.uk (Q. Meng), M.A.King@lboro.ac.uk (M. King), dennis.majoe@motion-robotics.co.uk (D. Majoe).

https://doi.org/10.1016/j.robot.2021.103891

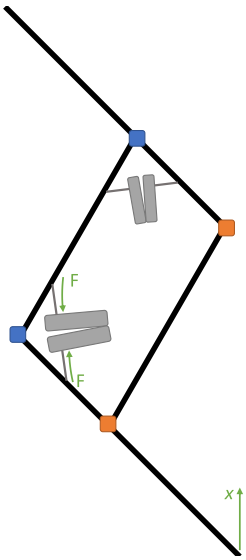


Fig. 1. Linear actuator so that the foot can move upwards without dragging on the floor.

Fig. 2. SARAH as was presented at the New Scientist Live 2017.

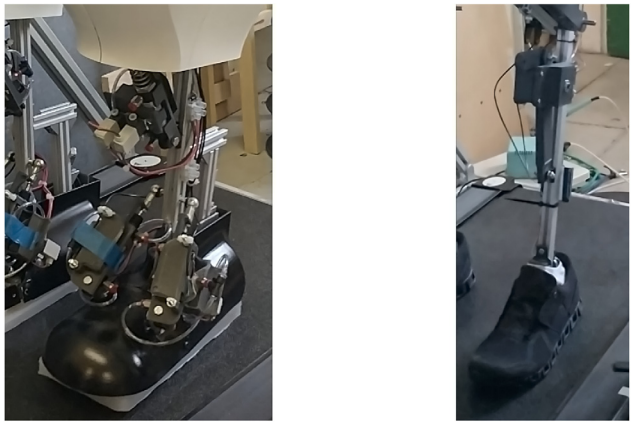


Fig. 3. First generation foot on the left, weighing 2.5 kg. Second generation foot on the right, weighing 1.5 kg.

the lower part can execute closed cyclic local patterns with higher speed. This method combines the speed and the stability of local pattern generators (LPGs) with the learning capabilities of NN's. The controller's learning capabilities were enhanced by using deep RL, which allows the system to learn through experience, similar to humans [12,13].

This paper focuses on the efficiency of bipedal robots. In order to conduct experiments, we have built a robot (**SARAH** - Safe Agile Robust Autono-mous Host) that could maximize efficiency through minimizing its power needs [14]. It utilizes an ostrich-like gait as it is more efficient albeit less dexterous than other animals and humans [15].

2. SARAH, a bipedal robotic host

The contemporary designs of bipedal robots are not much different from the designs of the past, as most of them focus on resembling human anatomy, rather than achieving efficiency [1]. New designs, like *Cassie* and *Digit* from *Agility Robotics*, focus on alternative designs that are potentially more efficient while sacrificing the humanoid look and the dexterity of humans' legs [16]. Similarly, this paper's robot host focuses on an alternative design that can maximize efficiency using (patent-pending) "bang–bang" actuators [17]. These actuators were designed at *Motion Robotics LTD* specifically for this robot. Initially, the actuators had two electromagnets facing one another, and they were fixed together on one edge. The other end was coupled with a moving pin such that when the electromagnets were energized, the pins moved closer to create torque. As shown in Fig. 1, the linear actuator was set up so that it could lift the foot off the ground without dragging. This is important for efficiency, whereas in traditional humanoids the leg has to energize two motors (i.e. one for the knee and one for the hip) to achieve the same result.

The linear actuator is the primary actuator as it was responsible for lifting the leg from the ground, allowing the hip and abductor actuators to act independently. The actuator was a custom-made compact linear actuator capable of lifting 50 kg for 5 s up to 5 cm in impulse mode while the sustained lifting power was 20 kg. Additionally, to take advantage of the maximum lifting power and increase efficiency, the actuator has an internal inverse braking system that can lock the actuator while not in use. This allowed the actuator to perform short movements of 5 cm and climb to the target position. When gravity was in the direction of the target position, the actuator used a small force opposite to the motion to act as a dampener. This increased energy efficiency, especially when the robot was in standby mode, as it can hold its posture mechanically, effectively limiting its energy consumption to its onboard computers. The host included multiple custom controller boards with ATMEL SAM C21 [18] for the LPGs, two Raspberry Pi 3B+ (RPI3) [19] for the higher-level system, communication with the user and the potential future torso.

SARAH went through different revisions to reduce weight, increase dexterity and efficiency. The first prototype can be seen in Fig. 2. The robot weighed 45 kg and had a height of 115 cm, while the second prototype reduced the weight to 27 kg by replacing the hydraulic braking system with an electromagnetic system. Furthermore, to reduce vibration and weight, the flat wood-foot was replaced by a 3D printed foot fitted in athletic shoes. Fig. 3 illustrates the difference between the two feet. With the new lightweight design, the robot was capable of a theoretical maximum walking speed of 5 km/h.

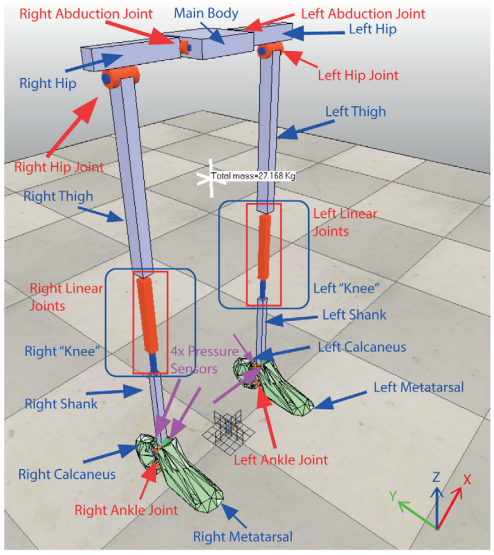


Fig. 4. Detailed representation of the second prototype of SARAH with its center of mass in V-REP Pro Edu.

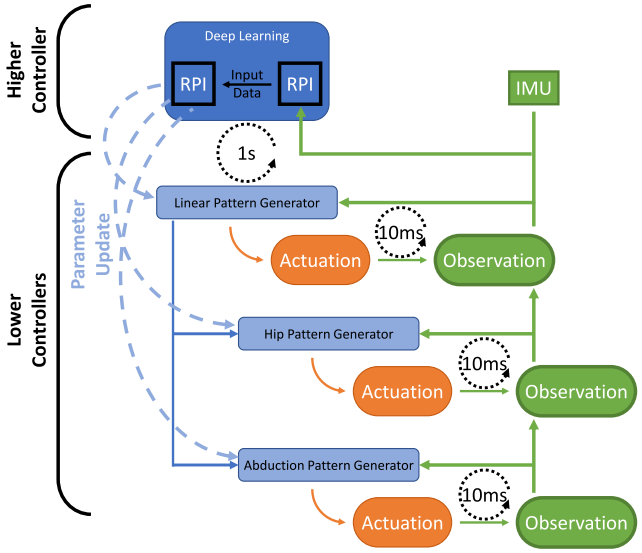


Fig. 5. Control flow of the hybrid controller on SARAH.

To accelerate the experiments, the robot's skeleton was simulated in V-Rep Pro Edu Dynamic simulator [20]. The weight of the actual robot was divided and distributed to each limb of the simulated robot skeleton in order to have the most accurate representation. Additionally, all physical actuators were tested individually to match the simulated actuators. At the same time, the whole model performed a series of tests that were replicated on the physical robot for further tuning. In Fig. 4, the simulated robot model is detailed with its center of mass noted by a white mark.

3. Hybrid autonomous controller

SARAH's mechanical system allowed for a more discrete balancing algorithm as it could make one-shot movements to balance its body while remaining in the resulting position without consuming energy. This uses small hibernation windows to reduce power consumption, hence increasing efficiency. However, the current control algorithms are not based on one-shot actuations, but on a feedback system that has a target, and normally requires energy to maintain it. To benefit from this ability, a new algorithm is designed to use a stability parameter to assess the robot's condition and act only if necessary.

The balancing algorithm is divided in two different systems combined hierarchically, as can be seen in Fig. 5. The higher-level system is responsible for analyzing data from the robot and its environment, in order to produce parameters that can modulate the lower-level system. The lower-level system is responsible for collecting local data and acting based on the parameters that the higher-level system provides. The higher-level system consumes much more energy than the lower-level controller to preform its job, hence its operating speed is set to a 100 times slower (1 Hz) than the lower-level controllers (100 Hz) in order to conserve its energy consumption over time.

3.1. Lower-level controllers

Starting from the lower-level controllers, the system uses three LPGs so that each actuator has its own pattern. The linear actuator is the main LPG that controls when the robot starts moving, as the foot must first lift off to avoid drifting on the ground while acting in the other two axes. The other two LPGs (one for *hip* actuation and one for *abduction* actuation) are activated as soon as the leg starts moving upwards. The higher-level system provides the parameters every second in a memory buffer, while the lower-level controllers read the buffer's values at the beginning of their pattern cycle in order to avoid a mismatch of parameters halfway through a pattern cycle. The linear LPG has three external parameters that control the forces acted by the actuator in different states. The other two LPGs have five external parameters each, four of which create a local activation variable while the last one defines the actuation force.

A pseudocode of the Linear LPG can be seen in Procedure 1. The linear pattern's control flow is divided to 4 stages representing the *waiting*, *pushing to ground*, *lifting to maximum* and *returning to the ground* stages of the foot. At the beginning of each stage, the actuator forces (**P1**, **P2** and **P3**) are set based on the higher-level controller's most recent predictions. The LPG for the other rotational actuators can be seen in Procedure 2.
Both the hip and abduction movement use the same algorithm, only with different external parameters and physical limits of actuation. The pattern initially calculates a variable based on a cubic function with the external parameters **1-4** and information from the Inertial Measurement Unit (IMU). This variable acts as a stability buffer when the robot has small forces that can be absorbed from the design without the need of stepping, in which case the *Rotational Force* is set based on the last external parameter. The leg swing is divided into 6 different motions, and is energized only if the leg is not touching the ground (stage 3 of the linear actuation). If the calculated variable *Right/Left* has absolute value within actuation limits and more than 1, then the leg moves forwards if its positive or backwards if its negative. If the variable's absolute value is less than 1, then the actuator is moving towards 0° and is braking if the position is ±1°. For the abduction actuation, the external parameters **V1-5** are provided by the higher-level controller, while the parameters **H1-5** are provided by the higher-level controller for hip actuation. Abduction is limited to act between −10° and 5°, while the hip is limited between −20° and 15°.

3.2. Higher-level controllers

The external parameters of the lower-level controllers, were set through the higher-level system by learning the system's

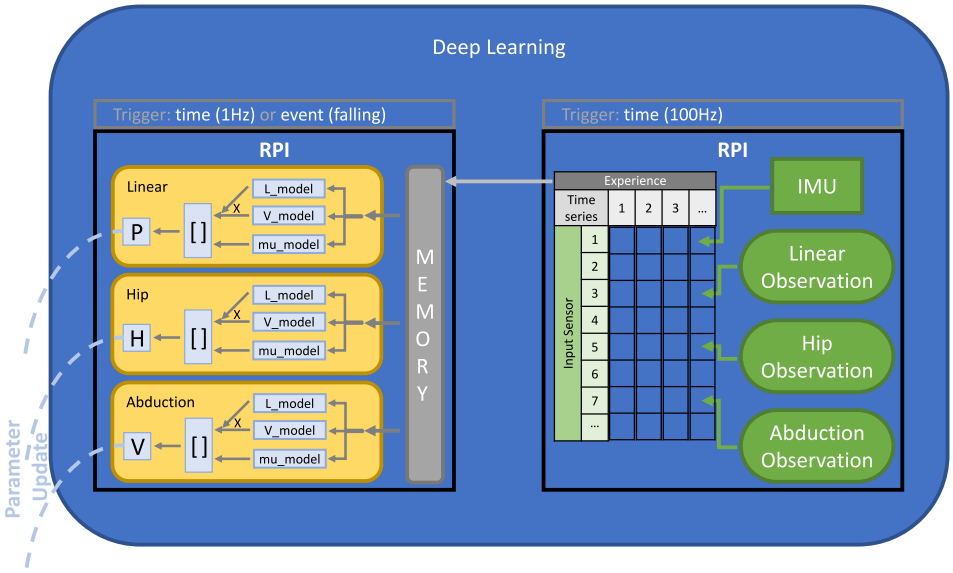


Fig. 6. Detailed decomposition of the higher deep learning controller from Fig. 5 with the two individual Raspberry PIs. The left one collects and forms the experiences, while the right one runs three individual NN, one for each actuator, either every second or based on events that are triggered through experiences.

Procedure 1: LPG(Linear Pseudocode Flow)

Data: IMU, Pressure & Position sensors
while *Robot is ON* **do**
  read current **Data**
  switch do
    case *1*
      if *Pressure Sensors* **then**
        Case 2
    case *2*
      Set Linear Force to **P1**
      if ***not*** *stable* **then**
        Push Ground Down
      if *Position* $<$ *1% of Step* **then**
        Case 3
    case *3*
      Set Linear Force to **P2**
      Lift Leg
      Mid_Air ***True***
      if *Position* $>$ *10% of Step* **then**
        Case 4
    case *4*
      Set Linear Force to **P3**
      Land Leg
      if *Position* $<$ *0* **then**
        Mid_Air ***False***
        Brake Robot
        Case 1

dynamics through experiences. This added learning capabilities to classical and non-adaptive pattern generators to enhance their stability with adaptation to new dynamic requirements. To achieve this, three individual Deep Neural Networks (DNN) were pre-trained through V-REP Pro Edu simulator based on Normalized Advantage Functions (NAF) [21] with a continuous replay memory. The DNN was mapping the sensory system of the robot, with any hidden dynamics, to the lower-level parameters **P1-3**, **V1-5** and **H1-5**. Each network was trained for each actuator in order to utilize smaller recurrent NN for quicker execution times. Fig. 6 illustrates the onboard execution cycle of the higher-level system, with the one RPI3 collecting data and filling an experience file, while the other RPI3 was registering the most recent experience as a memory to be used on the individual LPGs. The experience was extracted and used to train the network externally or onboard if it was required.

Moreover, decoupling the actuators' LPGs meant that the networks can be called on-demand based on external influences instead of every 1 Hz. For example, on softer ground, like sand, training can be limited to the linear actuators instead of spending energy retraining all the actuators. Training and execution on demand is an effective method of reducing power consumption and thus increasing efficiency. However, the complex system dynamics could not be learned without recurrent networks that introduce memory capabilities to the controller.

Recurrent layers can process previous states and examine the differences between them and the current states. This process must take place from the first layer as the previous states can be lost if the first layer is summing them with the simple neuron, or complex convolution layers. The original NAF agent was not able to utilize recurrent networks as first layers, due to the replay memory that was used, as it was mixing the time-series of the dataset with random polling. To overcome this issue, episodic memory was defined, together with random shuffling of episodes and no shuffling of time-series. Then, the network was provided with windowed data that preserved the states of each experience. In comparison, the standard replay memory was creating image-based data by assorting randomly time-series next to each other. The extended NAF agent (eNAF) (Procedure 3) was using the current data as the primary layer, and was filling the memory cells with the windowed data as they were recorded in observations.

Observations of the environment were saved as time-series, and then those time-series were saved in their episode, according to their timestamp. When the experience was called, memory function was randomly picking episodes and not the actual time-series. Afterwards, the random episodes were attached next to each other, offering continuity in sensor data except where the episode was changing. The outcome was a time-based experience and not the usual image-like experience that other memory methods offer. Fig. 7 demonstrates the differences between image-based data and time-based data. In real-world robots, the

Procedure 2: LPG(Rotational Pseudocode flow)

Data: IMU & Position sensors
while *Robot is ON* **do**
  read current **Data**
  Right = $-(V1 \cdot X^3 + V2 \cdot X^2 + V3 \cdot X + V4)$
  Left = $V1 \cdot X^3 + V2 \cdot X^2 + V3 \cdot X + V4$
  Set Rotational Force to V5
  if *Mid_Air* **then**
    if $|Right| > 1$ *or* $|Left| > 1$ **then**
      if *Right* > 0 **then**
        if *R_angle* $< 5^o$ **then**
          Move Right Positive
        else
          Brake Right Actuator
      else
        if *R_angle* $> -10^o$ **then**
          Move Right Negative
        else
          Brake Right Actuator
      if *Left* > 0 **then**
        if *L_angle* $< 5^o$ **then**
          Move Left Positive
        else
          Brake Left Actuator
      else
        if *L_angle* $> -10^o$ **then**
          Move Left Negative
        else
          Brake Left Actuator
    else
      if *R_angle* $> 1^o$ **then**
        Move Right Negative
      else if *R_angle* $< -1^o$ **then**
        Move Right Positive
      else
        Brake Right Actuator
      if *L_angle* $> 1^o$ **then**
        Move Left Negative
      else if *L_angle* $< -1^o$ **then**
        Move Left Positive
      else
        Brake Left Actuator

abrupt change between each time-series can give meaningful information about the dynamic change of states. Abrupt changes usually happen when a robot must act quickly, however, if that point is smoothed without the controlling algorithm noticing it, the situation may worsen.

4. Experiments

In this project, the scenario used for experiments focused on the abrupt change through a random push in intensity and direction on the robot's torso, which unbalanced it. The force lasted 500 ms, had a random direction in the transverse plane and acted on SARAH's main body. The robot was fully stabilized before the force was applied, and the aim was for it to return to a stable state after making a series of small in-place steps (stationary steps), as it could not micro-adjust its position. The robot would keep taking steps until the balance parameter (Eq. (1)) is below a user defined threshold (Fig. 8).

$$Stable = \frac{1}{1 + 2^{IMU_A - 5}} + \frac{1}{1 + 3^{5 - 10 \cdot IMU_G}} \quad (1)$$

Procedure 3: Pseudocode of eNAF

Initialize a Randomized Q Network with target network weights $Q(x, u|\theta^Q)$.
Initialize the Replay R.
while *New Episode* **do**
  Random process N to expand exploration.
  Initial Windowed Observation $\mathbf{x}_{1-10}$.
  Initialize Asynchronous Counter = 0
  Select action $\mathbf{u}_t = \mu(\mathbf{x}_{(t-1)..(t-11)}|\theta^\mu) + N_{(t-1)..(t-11)}$.
  for *New Timestep* **do**
    Record Observation $\mathbf{x}_t$ and calculate reward $\mathbf{r}_t$.
    Store States $(\mathbf{x}_{t-1}, \mathbf{u}_t, \mathbf{r}_t, \mathbf{x}_t)$ in Replay R.
    Increase Asynchronous Counter
    if *Asynchronous Counter* $>=$ *Asynchronous Period* **then**
      Sample a random batch of **I** Episodes from R
      Calculate **L** transitions from the batch
      Restructure and trim **L** to form Episodic Memory (**E**) for continuity between all timesteps of different episodes
      for *E* **do**
        $y_i = r_i + \gamma A'(\mathbf{x}_{i+1}|\theta^{Q'})$
        Update θ^Q to minimize
        $L = \frac{1}{N}\sum_i (y_i - Q(x_i, u_i|\theta^Q))^2$
        Update Q model $\theta^Q \Longleftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
      Reset Asynchronous Counter
      Select action $\mathbf{u}_t = \mu(\mathbf{x}_{(t-1)..(t-11)}|\theta^\mu) + N_{(t-1)..(t-11)}$.

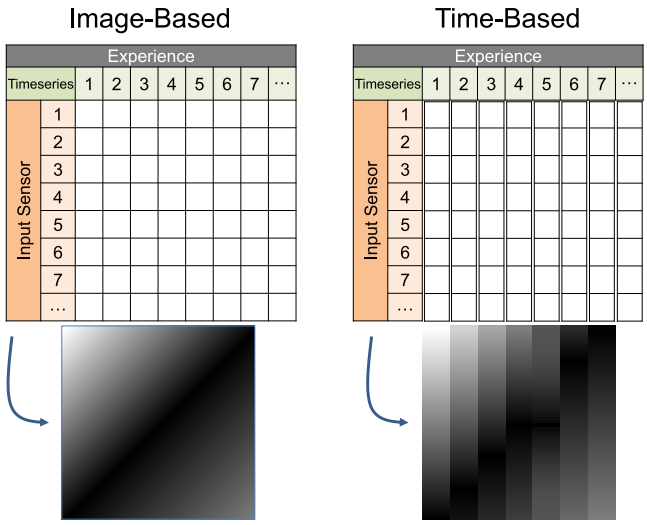


Fig. 7. Differences between the conventional image-based data and the proposed time-based data.

where, *Stable*: Heuristic variable for the robot's stability. IMU_A: Acceleration magnitude of Main Body. IMU_G: Rotational rate magnitude of Main Body.

The equation was constructed heuristically in combination with isolated experiments that were carried in the simulator. During those experiments, the robot was not acting but was pushed around with a random force (both in magnitude and in direction) on its torso. The IMU data were collected with a flag if the robot fell and when. Then, the 90% magnitude of acceleration and rotation data of each experiment were plotted in 3 dimensions, with the third axis being the time for the robot to fall, if it did. The equation was fitted and normalized to 2 with a mean square error of up to 5%. The threshold was chosen

Fig. 8. *Stable* as it was calculated from Eq. (1). The red plane shows the actuation's cut-off area; anything under this plane was not activating the motion as it classified the state of the robot as stable.

when 0.1% of the experiments, where the robot was falling. The experiments were executed for a 1.000.000 times in order to supply a fair amount of points for an accurate equation.

Training took place entirely in V-REP Pro Edu simulator without the use of the physical robot. The lower-level controllers were executed every 10 ms inside the simulator, while receiving new parameters every 1 s from the RL algorithm, running through an external python script. The algorithm trained the linear LPG, then the hip LPG and finally the abductor LPG in consecutive order. In order to keep a balance between the different training runs and not to over-train(overfit) any of the LPGs, the reward was designed so that each LPG will affect certain part of the reward more than others. Additionally, for the linear LPG training, the activation (push) force was limited in the forwards and backwards directions with a magnitude that will destabilize the robot but not tip it over. For the hip LPG, magnitude force limit was removed and for the abduction LPG all limitations on the force were lifted. Each LPG was trained for one million time-steps which resulted in $3500 - 4000$ episodes.

The eNAF agent used a complex reward function to improve its performance based on the robot's performance but not connected to its dynamics. Other rewards may relate the reward with the dynamics of the environment or the model, like acceleration, which guides the reward to a solution. By allowing the agent to use a decoupled reward, it has more freedom to explore possible solutions at the expense of training time and the possibility of no conversing to a solution. The main reward function (Eq. (2)) was calculated on each time-step and was summed at the end of each training episode.

$$r = (1 - C_{m[x]}) \cdot (1 - C_{m[y]}) \cdot C_{m[z]} \cdot 2^{-(S-3)^2} \tag{2}$$

where, $C_{m[x-y-z]}$: Move of CoM in x-y-z axis in absolute metric value.

S: In-place steps per second.

The parameters $C_{m[x-y-z]}$ were creating a concave 4 dimensional reward that was at the highest point only when $C_{m[x]}$ and $C_{m[y]}$ were at the origin, and $C_{m[z]}$ was at the maximum height of the robot (95 cm). The parameter that considered the steps per seconds was designed as a negative square function with its center moved to 3, which was the desired speed. Three steps per second produced an average speed of 1.425 m/s (step length of 47.5 cm, which is the target stepping speed of 5 km/h (1.39 m/s).

In order to fairly reward the robot, the first 2 s were given a zero reward as the agent could not improve the scene because the robot was free-falling from 3 cm and then stabilized soon after. The main reward was activated after the 2 s, as the force was activated between 1.5 s to 2 s and thus was crucial for the next 3 s. During that time, the robot must take steps to stabilize its body and then stop moving after the 5th s. The agent was penalized by reducing the reward through dividing it by 100 after the fifth second as the agent must stabilize the robot in just 3 s period (between $2 - 5$ s of the simulator). If it was achieving stability, the reward function was turning to zero, thus after the 5th s the reward function was omitting the *Steps per second* parameter of the reward if the robot was stable. The omission of the parameter was increasing the reward, similar to having the parameter equal to 1 (Eq. (3)).

$$r_s = (1 - C_{m[x]}) \cdot (1 - C_{m[y]}) \cdot C_{m[z]}/100 \tag{3}$$

If it was unstable, the reward was lower as the parameter will be less than 1 (Eq. (4)).

$$r_u = (1 - C_{m[x]}) \cdot (1 - C_{m[y]}) \cdot C_{m[z]} \cdot 2^{-(S-3)^2}/100 \tag{4}$$

Graphically, Fig. 10 demonstrates how the reward will be evolved if the robot does not move from the origin and achieve ± 3, ± 2,

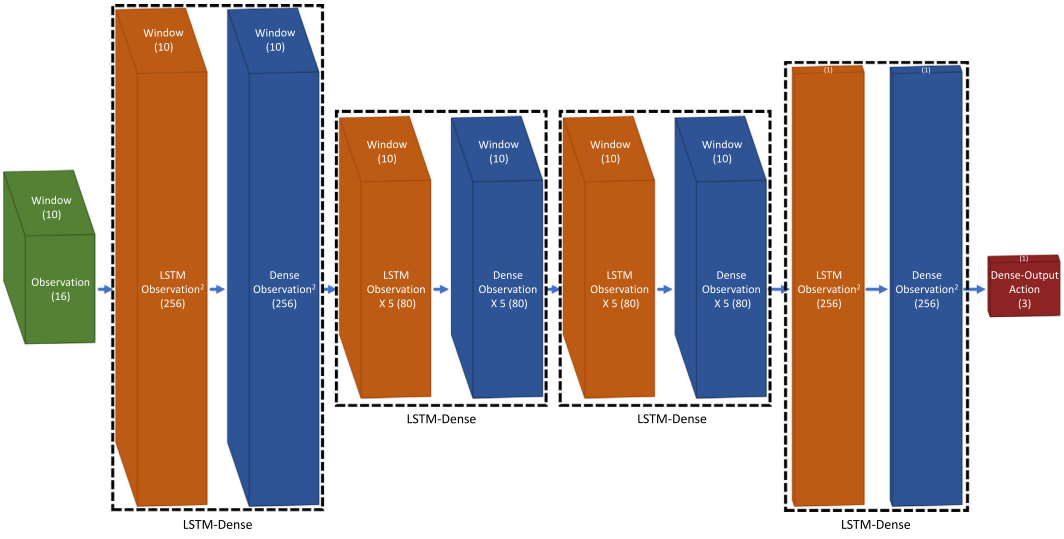


Fig. 9. Fundamental gait for balancing after a random push in the transverse plane on SARAH's torso.

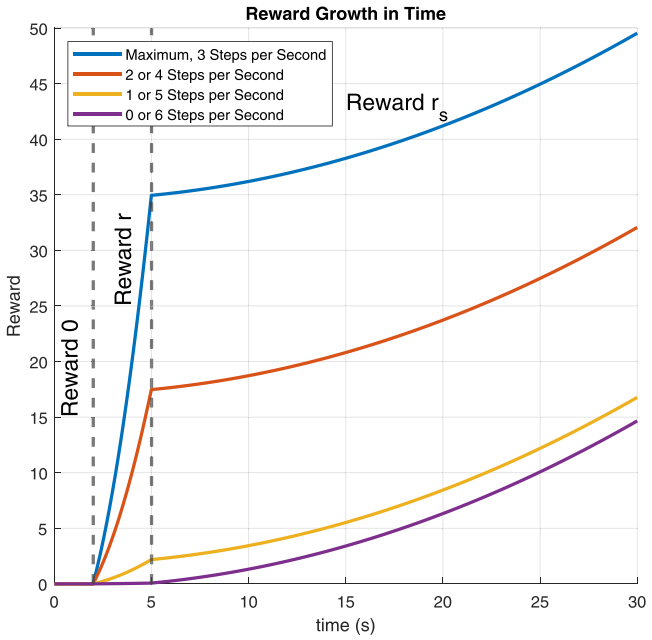


Fig. 10. Reward during an episode with ±3, ±2, ±1, 0 *Steps per second* difference from the target of 3 *Steps per second*.

±1, 0 *Steps per second* from the targeted 3 *Steps per second* during 2 − 5 s while being stable (Eq. (3)) after 5 s.

For the eNAF agent, three NNs were designed to be the actor, Q-Value and Advantage function, respectively, for each LPG. Memory was necessary, particularly from the environment, thus the actor and Q-Value networks included recurrent layers from the input layer. The networks had alternating layers of Long–Short Term Memory (LSTM) and normal (Dense) layers, as is demonstrated in Fig. 9. After systematic experimentation with different network architectures. The final network for the actor had four LSTM-Dense alternations, with 256 − 256 × 80 − 80 × 80−80 × 256−256 neurons per layer. The Q-value network had the same architecture but with 80 − 80 × 32 − 32 × 32 − 32 × 80 − 80 neurons per layer. As for the Advantage function network, the memory was not crucial, as the network was used as a matrix multiplier capable of advanced learning, hence five Dense layers with 16 neurons each were enough to learn the system's dynamics.

Table 1
LPGs final parameters.

Training 1		Training 2		Training 3	
P1	3.1 ± 0.1	H1	-3.7 ± 0.01	V1	0.3 ± 0.01
P2	1.1 ± 0.1	H2	1.9 ± 0.03	V2	-0.4 ± 0.03
P3	2.0 ± 0.1	H3	-0.1 ± 0.03	V3	0.5 ± 0.03
		H4	0.9 ± 0.03	V4	-0.9 ± 0.05
		H5	49 ± 0.1	V5	9.1 ± 0.1

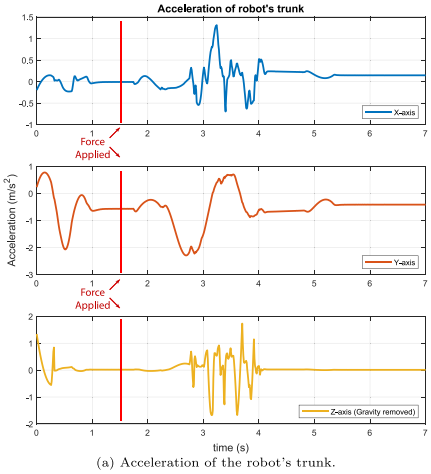
The inputs of each network included the past 10 time-steps resulting in a 100 ms memory. The actor and Q-Value networks were having memory until their last LSTM layer, which forwarded only its last (most current) values to the next Dense layer. For the Advantage function, the windowed data were flattened on the input with the observations as proposed in the original NAF algorithm [21]. The outputs of all networks had the same connections as the original NAF algorithm, hence the learning procedure did not diverge from the original.

The robot was simulated in V-Rep Pro Edu Dynamic simulator [20] with the industrially evaluated Vortex dynamics engine for high precision realistic simulations [22]. The simulated environment was necessary to accelerate the training thorough running continuously while avoiding physical wear and other common problems normally faced in robotics experimentation. The LPGs were coded inside the simulator to represent the close relationship between the robot and its lower-level controllers as they are part of the actuators. The controllers can achieve real-time processing (up to 0.1 ms per pattern cycle), and they were synchronized with the simulator's computational cycles.

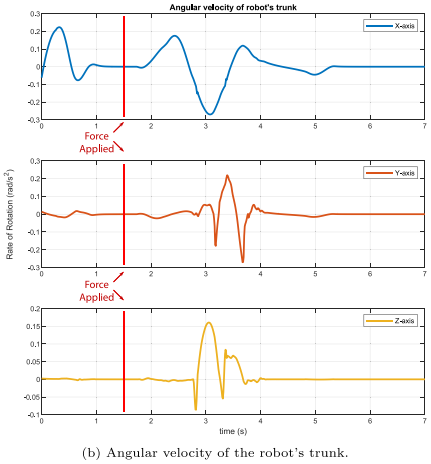
The higher-level system was designed externally, as a back-end program written in Python, and utilizing the TensorFlow library [23]. The use of TensorFlow allows the NN to be extracted and used on the RPI3 for training and execution. Other important packages used on the controller and are available on the RPI3 are keras [24] and keras-rl [25]. The lateral includes various RL algorithms from which *Normalized Advantage Function* was chosen for this project, as it can produce temporal outputs, and it is better than other model-free algorithms [21]. The algorithm was extended to include the use of time-based data as input in order to extract time-domain dynamics that can be lost in image-based data.

5. Results and discussion

The consecutive training was repeated twice to achieve further tuning on the second time, as all the LPGs were pre-trained and



(a) Acceleration of the robot's trunk.



(b) Angular velocity of the robot's trunk.

Fig. 11. IMU data from robot during evaluation.

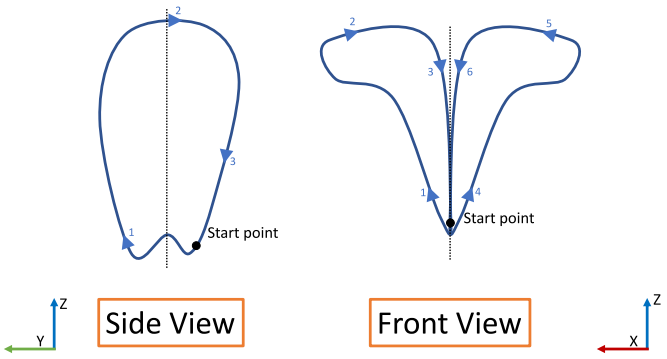


Fig. 12. Fundamental gait for balancing after a random push in the transverse plane on SARAH's torso.

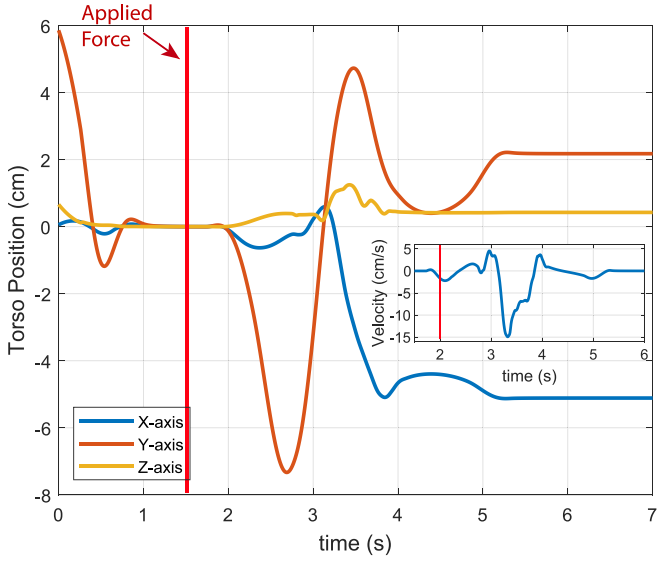


Fig. 13. Trunk Movements in 3D space.

expected to earn a higher initial reward. The high-level controller was trained in the dynamic simulator (deep reinforcement learning) with the random force acting on the middle of the torso as was described in previous section, and was changing/training the parameters on the LPGs. The parameters stabilized to the values that can be seen in Table 1. Those parameters can create a fundamental gait of SARAH, which can withstand forces of up to 100 N similar to the ones faced in the training scenario. A validation experiment was set up similar to the training experiment but without using the higher-level controller. This demonstrated the capabilities of the robot under extreme scenarios where the higher-level system is disconnected or powered off due to power-saving measures or abnormalities.

A median gait, over the validation experiments can be seen in Fig. 12 with the robot's CoM moving in the side and front view. The blue closed loop line shows the resulting gait as a movement of the CoM. As the movement is in 3-dimensional world, a straightforward representation was to present it through the side and frontal view of the robot. Numbers 1, 2 and 3 from the side view match the timing of numbers 1, 2 and 3 or 4, 5 and 6 of the frontal view based on which leg is moving.

For the evaluation experiments, torso movements and data from the IMU (which is placed on the torso) were recorded for analysis. An example of this data can be seen in Figs. 11 and 13. In that example, the robot was pushed with a diagonal force of 30 N forward and 50 N from the left to the right. In Fig. 13, the robot's CoM is presented with the origin of the 3D spaces adjusted to (0, 0, 0) at the 1.5 s when the force was applied to the robot. As can be seen, the robot's final position was just 5 cm on the right and 2 cm backwards.

The *x*-axis position derivative can be used to extract the *steps per second* as when the robot was making one step, its speed in the *x*-axis was changing direction. That results in seven steps (omitting the first direction change as it was due to the force) during 3 s, hence 2.33 *steps per seconds*.

Additionally, by analyzing the IMU data with Fast Fourier Transform (FFT), the vibration magnitude and the dominant frequencies were extracted. This can be useful as the design of the next prototype can incorporate materials that absorb those vibrations and result in less noise, both acoustically and electronically.

To understand the external parameters and how they affect the fundamental gait, the validation experiment was performed with the parameters altered by ±50%, one at a time. Fig. 14 demonstrates how each parameter is modulating the fundamental gait. The yellow lines show the path that a particular point will follow in order to modulate the gait based on the variables of LPGs. Those paths were extracted after manually varying the variables in the LPGs' one by one. Through training in different scenarios, the RL agent would relate different gait types to different values and correlate them with different environments. This would result in a robot that can efficiently adapt to a new environment with its two controllers working autonomously in hybrid mode, while running at different execution speeds.

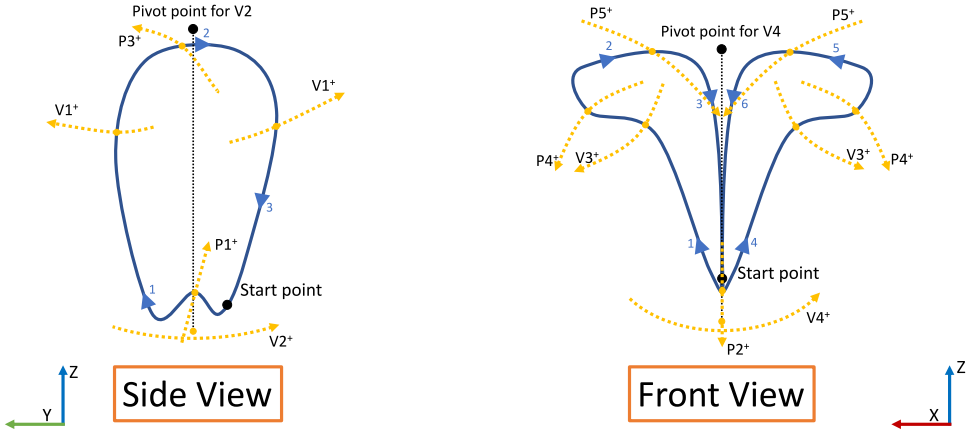


Fig. 14. How higher-level parameters change the final gait.

6. Conclusion

This article proposed a new control algorithm that combines a low-level/high-speed controller with a high-level/low-speed controller to enable learning capabilities without sacrificing high-speed actuation. The proposed controller was designed to take advantage of the power-efficient bipedal robot from Motion Robotics LTD. SARAH does not have a feedback loop on its actuators to eliminate micro-adjustments that draw power in other bipedal robots while they are not in use. The suggested controller offers smoother motions, as the lower-level system acts rapidly with speed up to 10 kHz (tested at 100 Hz as the dynamic simulator was too slow for the maximum speed which could be achieved on the physical robot) based on a predefined fundamental gait. Additionally, it eliminates stuttering as it executes a closed cyclic pattern continuously and blindly.

To control those blind movements, an RL algorithm was added hierarchically on top of the lower-level system to adjust those LPGs. RL can offer learning through experience before deployment, as well as after deployment, while the robot is under normal use. The higher-level system has the possibility to control the fundamental gait and change it based on the environment's requirements. Furthermore, as both controllers (LPG and Deep Neural Networks) are independent, they can be pre-trained offline, and the on-board training of the DNN can happen on demand, reducing the power requirements of the robot for otherwise power-hungry NN. That is a compromise between microcontrollers that are not power-hungry with neural networks that have higher power and computational needs.

This project's future work will include the robot's training under different environments and the controller's deployment on the real robot. After applying the controller on the actual robot, validation experiments will be conducted to compare the simulated robot's performance with the real robot in different scenarios. Finally, an onboard training session will be conducted to evaluate the performance and the time needed to adjust the gait to fit the new environment.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research presented inhere was financially supported by the Engineering and Physical Sciences Research Council (EPSRC) Center for Doctoral Training in Embedded Intelligence (CDT-EI) under grant reference EP/L014998/1. The authors also would like to pay special regards to Motion Robotics LTD employees for their support and input in this research.

References

[1] T. Mita, T. Yamaguchi, T. Kashiwase, T. Kawase, Realization of a high speed biped using modern control theory, Internat. J. Control 40 (1) (1984) 107–119, http://dx.doi.org/10.1080/00207178408933260.

[2] T. Reil, P. Husbands, Evolution of central pattern generators for bipedal walking in a real-time physics environment, IEEE Trans. Evol. Comput. 6 (2) (2002) 159–168, http://dx.doi.org/10.1109/4235.996015.

[3] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, G. Cheng, Learning CPG-based biped locomotion with a policy gradient method: Application to a humanoid robot, Int. J. Robot. Res. 27 (2) (2008) 213–228, http://dx.doi.org/10.1177/0278364907084980.

[4] S.F. Rashidi, M.R.S. Noorani, M. Shoaran, A. Ghanbari, Gait generation and transition for a five-link biped robot by central pattern generator, in: 2014 2nd RSI/ISM Int. Conf. Robot. Mechatronics, ICRoM 2014, 2014, pp. 852–857, http://dx.doi.org/10.1109/ICRoM.2014.6991011.

[5] T. Matsubara, J. Morimoto, J. Nakanishi, M. aki Sato, K. Doya, Learning CPG-based biped locomotion with a policy gradient method, Robot. Auton. Syst. 54 (11) (2006) 911–920, http://dx.doi.org/10.1016/j.robot.2006.05.012.

[6] Y. Nakamura, T. Mori, M.-a. Sato, S. Ishii, Reinforcement learning for a biped robot based on a CPG-actor-critic method, Neural Netw. 20 (6) (2007) 723–735, http://dx.doi.org/10.1016/j.neunet.2007.01.002.

[7] J.B. Nielsen, How we walk: Central control of muscle activity during human walking, Neurosci. 9 (3) (2003) 195–204, http://dx.doi.org/10.1177/1073858403009003012.

[8] J. Massion, Movement, posture and equilibrium : Interaction and coordination, Prog. Neurobiol. 38 (1) (1992) 35–56, http://dx.doi.org/10.1016/0301-0082(92)90034-C.

[9] K. Miyashita, S. Ok, K. Hase, Evolutionary generation of human-like bipedal locomotion, Mechatronics 13 (8-9 SPEC.) (2003) 791–807, http://dx.doi.org/10.1016/S0957-4158(03)00003-5.

[10] A.R. Rao, G.A. Cecchi (Eds.), The Relevance of the Time Domain to Neural Network Models, third ed., Springer US, Boston, MA, 2012, p. 213, http://dx.doi.org/10.1007/978-1-4614-0724-9.

[11] H. Shahbazi, K. Jamshidi, A.H. Monadjemi, H. Eslami, Biologically inspired layered learning in humanoid robots, Knowledge-Based Syst. 57 (2014) 8–27, http://dx.doi.org/10.1016/j.knosys.2013.12.003.

[12] P. Jarvis, An analysis of experience in the processes of human learning, Rech. Form. (70) (2012) 15–30, http://dx.doi.org/10.4000/rechercheformation.1916.

[13] R. Gross, Psychology: The Science of Mind and Behaviour., Seventh, Hodder Education, London, 2015, p. 992.

[14] C. Kouppas, Q. Meng, M. King, D. Majoe, S.A.R.A.H.: The bipedal robot with machine learning step decision making, Int. J. Mech. Eng. Robot. Res. 7 (4) (2018) 379–384, http://dx.doi.org/10.18178/ijmerr.7.4.379-384.

[15] C.R. Taylor, N.C. Heglund, G.M. Maloiy, Energetics and mechanics of terrestrial locomotion. I. Metabolic energy consumption as a function of speed and body size in birds and mammals, J. Exp. Biol. 97 (1) (1982) 1–21, http://dx.doi.org/10.1146/annurev.ph.44.030182.000525.

[16] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, M. van de Panne, Iterative reinforcement learning based design of dynamic locomotion skills for cassie, 2019, arXiv:1903.09537.

[17] D. Majoe, Angular displacement device, 2016.

[18] Microchip Technology Inc., Microchip SAM C, ATSAMC21J18A, 2019, URL https://www.microchip.com/wwwproducts/en/ATSAMC21J18A.

[19] Raspberry P.I. Foundation, Raspberry Pi 3 model B, 2019, URL https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[20] Coppelia Robotics GmbH, V-Rep pro educational, 2018, URL http://www.coppeliarobotics.com/.

[21] S. Gu, T. Lillicrap, I. Sutskever, S. Levine, Continuous deep Q-learning with model-based acceleration, Transplant. Proc. 32 (5) (2016) 932–934, arXiv:1603.00748.

[22] CM Labs Simulations, Vortex studio, 2019, URL https://www.cm-labs.com/vortex-studio/industry/robotics-simulation/.

[23] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous distributed systems, Nat. Neurosci. 16 (4) (2016) 486–492, http://dx.doi.org/10.1038/nn.3331, arXiv:1603.04467.

[24] Chollet François, et al., Keras, 2015, URL https://keras.io.

[25] M. Plappert, Keras-rl, GitHub Repos. (2016) URL https://github.com/keras-rl/keras-rl.

Christos Kouppas is a Ph.D. student in Computer Science at Loughborough University in the United Kingdom. He obtained BEng in Mechanical Engineering from the University of Cyprus, graduating first of his class in 2015. He also graduated from The University of Sheffield first of his class at the M.Sc. Advanced Control & Systems Engineering in 2016. He has been awarded the Laverick Webster Hewitt Prize for his outstanding performance and The Eric Rose Prize for the best project, during his master studies.

Mohamad Saada is a KTP researcher at Loughborough University in the United Kingdom. His main research interest includes but not limited to autonomous and automatic systems, mobile and stationary robotics as well as advance control algorithms with artificial intelligence.

Qinggang Meng is a Professor in robotics and autonomous systems at Loughborough University, UK. His main research interests and expertise include: cognitive robotics, multi-robot/UAV cooperation, AI, machine learning and computer vision, driverless vehicles, human–robot interaction, and ambient assisted living.

Mark King is a Professor in Sports Biomechanics at Loughborough University, UK specializing in using subject-specific computer simulation models to understand optimum performance and injury risk in sport. Integral to this work is the role of muscle and technique on optimum performance and how the force gets transmitted/energy dissipated through the body. He has been at Loughborough since 1990 graduating in Sports Science and Mathematics in 1993 and obtained his Ph.D. in computer simulation of dynamic jumps in 1998.

Dennis Majoe is a senior scientist in a scientific collaboration with the Ecole Polytechnique de Lausanne advising on high fidelity medical wearable devices. He is also CEO for Motion Robotics Ltd, U.K. His main area of research is robotics as applied to assistive living.