The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

The Design of LEO: a 2D Bipedal Walking Robot for Online Autonomous Reinforcement Learning

Erik Schuitema, Martijn Wisse, Thijs Ramakers and Pieter Jonker

Abstract— **Real robots demonstrating online Reinforcement Learning (RL) to learn new tasks are hard to find. The specific properties and limitations of real robots have a large impact on their suitability for RL experiments. In this work, we derive the main hardware and software requirements that a RL robot should fulfill, and present our biped robot LEO that was specifically designed to meet these requirements. We verify its aptitude in autonomous walking experiments using a pre-programmed controller. Although there is room for improvement in the design, the robot was able to walk, fall and stand up without human intervention for 8 hours, during which it made over $43,000$ footsteps.**

I. Introduction

The unpredictability of the home environment requires that future domestic robots will be able to learn new motions for new tasks. Their users will generally not be trained roboticists, so the robots will have to learn from sparse and simple feedback. The success criteria may be different for each task, and the system or environment may change over time. This is a learning problem with a difficult set of requirements. One of the increasingly popular solutions is the concept of Reinforcement Learning (RL), a generic framework in which systems can autonomously learn complex behaviors from coarse feedback in the form of rewards for good and bad behavior. Most publications on this topic, however, are limited to simulation studies only.

Successful applications of RL to the problem of learning sensori-motor skills on *real* robots are rare. In [1], a three link robot learned to stand up using actor-critic and Temporal Difference (TD) learning methods inside a hierarchical framework. In [2], a 3D bipedal robot, which is intrinsically stable, increased its walking performance using actor-critic TD-learning. In [3], bipedal locomotion was learned from demonstration using a central pattern generator. In [4], a complex humanoid robot learned various movement skills from demonstration. The successful learning behaviors of these studies demonstrate the potential of Reinforcement Learning. However, it is striking that there are only so few studies with real robots, and even these were always restricted in one way or another. The allowed control policies were restricted to be predefined, parameterized and sometimes partially pre-programmed policies. The learning process often had to be initialized by learning in simulation first, or by priming the solution with a demonstration of the task by a human. The hardware is often the cause of such restrictions. Most hardware lacks the robustness to withstand a large number of learning trials with random exploration. Although these limitations are known by experienced researchers, we found no publication that explicitly maps the essential requirements of the RL framework onto hardware (and software) requirements, making it difficult to develop robots suitable for RL. Therefore, this paper reports the design of a robot (named ‘LEO’) that we explicitly developed for online, autonomous RL research.

E. Schuitema, M. Wisse, M.J.G. Ramakers and P.P. Jonker are with the Delft Bio-Robotics Lab, Faculty 3ME, Delft University of Technology, Mekelweg 2, 2628CD, Delft, The Netherlands. Tel: +31152782578, Fax: +31152784717. {e.schuitema, m.wisse, p.p.jonker}@tudelft.nl

The robot LEO (Fig. 1) is a 2D biped (two-legged) robot. We selected the bipedal walking motion as our example task for two reasons. It is a complex, challenging task, and, we have extensive experience with it from previous work [5]. In this paper, we derive system requirements from the fundamentals of RL in Sec. II, provide a system overview of LEO in Sec. III, followed by detailed hardware and software requirements and their implementation in Sec. IV to VIII. The paper ends with conclusions in Sec. IX.

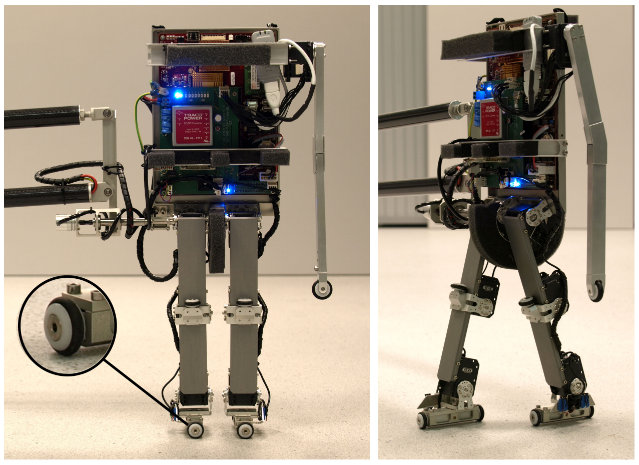

Fig. 1. LEO: a 2D walking robot designed for Reinforcement Learning. LEO is equipped with seven servo motors (hips, knees, ankles and shoulder) and is connected to a boom construction that provides power and lets it walk in circles. It is allowed to fall, after which it stands up autonomously.

II. Reinforcement Learning requirements

The concept of RL (see for example [6] for an introduction) has important implications for the hardware and software design of a robot that is to be controlled online by RL. Here we summarize the key properties of RL and the robot design requirements that follow from it. The resulting requirements will be numbered for easy reference and will be further explained in the subsequent chapters.

A. Robustness

A key aspect of on-line RL is that the learning agent tries something new every now and then, i.e. it *explores*. Although exploration can be guided, it can in principle lead to system damage. For a humanoid robot, the cause is usually a fall or a self-collision. From simulation of a biped with a complexity comparable to LEO [7], [8], we know that learning a walking behavior from scratch by controlling two hip motors takes approximately 20 hours or more (without counting the time required to stand up) and hundreds of falls. This leads to the following requirement:

1) The robot can walk over a period of days and is robust against falls and self-collisions.

In Sec. IV, we discuss how this requirement has influenced the design of our robot.

B. State transitions

The common framework for RL is that of the Markov Decision Process (MDP). The learning system is modeled as an MDP with discrete time steps labeled $k = 0, 1, .. \in \mathbb{Z}$ and is defined as the 4-tuple $\langle S, A, T, R\rangle$. Here, S is a set of states and A is a set of actions. The state transition probability function T defines the probability that the system reaches state $s_{k+1} \in S$ after executing action $a_k \in A$ in state $s_k \in S$. The reward function R defines the scalar reward of a state transition as $r_{k+1} = R(s_k, a_k, s_{k+1})$. The goal of the learner is to find a control policy that maps states to actions and that maximizes the expected cumulative sum of rewards from R.

An MDP has the *Markov property*, which means that T and R only depend on the current state-action pair and not on past state-action pairs, nor on information excluded from s. The on-line learning robot will experience a stream of events of the form $s_0, a_0, r_1, s_1, a_1, r_2, s_2, ...$ For the Markov condition to be true, every observation $[s_k, a_k, r_{k+1}, s_{k+1}]$ has to be in accordance with T at any moment during a learning experiment, which might take days. In this period, T must be static. This leads to the following robot design requirements:

2) The robot can observe state s, which holds all information relevant to the learning problem.
3) The effect of action a in every state s is predictable.
4) The sampling time is constant.
5) T must be static within a time frame of tens of hours.

In addition, within the MDP framework, a control action a_k that is based on state observation s_k is assumed to take place immediately after the observation itself, which poses an additional requirement:

6) The time between measurement s_k and action a_k is zero.

In a real robot, these conditions can only be met approximately. The system will suffer from sensor and actuator noise, finite accuracy in the periodic timing, and computational delays due to, e.g., running the learning algorithm. However, we made specific design decisions that should keep the violation of these requirements to a minimum. These are discussed in Sec. V, VI, VII and VIII.

C. System complexity

For the learning system to discover the long term value of all actions $a \in A$ in all states $s \in S$, in principle, it should experience all actions in all states at least a number of times. This search space can be limited in several ways by changing the learning algorithm. Examples are restricting the allowed policies by parameterizing them (policy search and policy iteration methods), extracting recurring tasks by using hierarchies of learning tasks (Hierarchical Reinforcement Learning), and assuming that neighboring states and actions have related values (function approximation and generalization). However, applying RL to complex robots still remains one of the greatest challenges in the field. Therefore, our robot design should meet the following additional requirement:

7) The number of degrees of freedom should allow for currently viable learning tasks, as well as more complex ones.

This requirement is further discussed in Sec. III.

III. System overview

The robot was designed by checking all 7 requirements as detailed in the subsequent chapters. Although the complete system design is the final result (i.e., the conclusion) of this paper, we provide the overview here at the start of the paper for ease of understanding.

LEO (Fig. 1) is small, approx. 50 cm in height, and light, approx. 1.7 kg. It has foam bumpers on both sides of the top of the torso and between the hip motors, thereby capable of taking numerous falls in a wide range of configurations. From simulation results on bipedal walking robot 'META' [7], [8], we know that learning to walk can take place in an acceptable time frame of days for a robot with 7 degrees of freedom. Therefore, to comply with requirement 7, we designed our robot to have a number of degrees of freedom comparable to robot META. LEO has 7 servo motors (Dynamixel RX-28; max. torque 3 Nm): two in the ankles, knees and hips and one in its shoulder. The servo motors communicate with an embedded computer (VIA Eden 1.2GHz CPU and 1GB RAM) over RS-485 serial ports. They are capable of position control and voltage control, and can communicate their current position and temperature. While previous research has showed the added advantage of accurate torque control [9], which these servo motors cannot accomplish, they are commercially available, all-in-one, easily replaceable packages. The feet have pressure sensors that measure foot contact. LEO has an arm that enables it to stand up after a fall.

LEO is connected to a boom (length 1.6m) with parallelogram construction. This keeps the hip axis always horizontal, which makes it effectively a 2D robot and thus eliminates the sideways stability problem. The boom also supplies power to the robot and makes it walk in circles, which together guarantee long-term continuous operation. An encoder in the hip joint measures the absolute angle between the torso and the boom. The foot contact points can roll sideways, see Fig. 1, which is needed to counter the effects of running in circles.

A wide variety of learning tasks can be conducted, ranging from learning a walking motion by actuating the two hip motors (keeping the ankles stiff; virtual constraints on the knees), to learning optimal ankle push-off, to learning a complete stand up behavior using all 7 motors.

The video accompanying this paper shows LEO being controlled by a pre-programmed (i.e., non-learning) limit cycle walking controller [10]. Typical walking strides from this process are shown in Fig. 2 by means of the evolution of the angles of both hip motors and the torso.

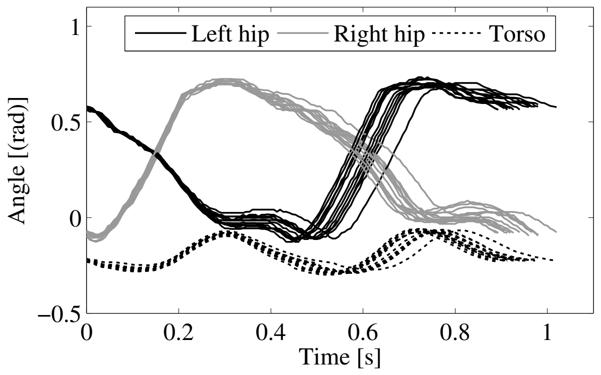

Fig. 2. The hip angles and torso angle for typical strides of LEO when using a limit cycle walking controller. The strides are aligned at left heel strike.

IV. Robustness

In order to comply with requirement 1, the robot should be able to walk over a period of days and be able to withstand falls and self-collisions. This has led to the following design choices.

By keeping the robot small, approximately 50 cm in height, the impacts during a fall are kept small as they scale more than cubically with the robot's height. By choosing 'smart' actuators with internal protection against overloading and overheating, the actuators are less likely to fail. In case they do fail, they are easy to replace. To reduce the impact on the torso during a fall, foam was placed on both sides of the top of the torso. A strong leaf spring with added foam between the two hip motors protects them at all possible hip joint angles. In the construction of previous robots, we used micro switches in the feet to detect foot contact, which were quite sensitive to failure. To increase the robustness of this design, we used pressure sensors in the feet, which reduced the number of moving parts.

A. Empirical verification

The first tests with our robot exposed a number of weak mechanical links. The weakest links were the aluminium brackets bought with the servo motors, which broke several times. In the hips, we replaced these with custom designed brackets that had increased flange thickness (2.5mm) and were made of higher quality aluminium; in the knees, we improved the mounting of the bracket to the lower leg to decrease the chance of fatigue.

The single board computer formed a weak link as well. At each impact with the floor, the inertia of the heavy CPU cooler caused significant bending of the motherboard, which ultimately led to loose contacts in the motherboard area around the CPU. To solve this, we improved the mounting of the CPU cooler and the mounting of the motherboard to the robot.

After implementing the above mentioned changes, we tested the robustness of the robot by letting it walk using a pre-programmed (i.e., non-learning) limit cycle walking [10] controller until it failed. The hip angle was controlled to a fixed reference angle using voltage control until the detection of the next heel strike. Upon foot contact, the back leg performed knee flexing while swinging forward. The torso was controlled to a reference angle as well. The robot made over $43,000$ footsteps and walked more than $6,000$ meters in a period of about 8 hours, of which 2.3 hours were spent to periodically let the motors cool down by resting on the floor. The robot fell 30 times (mostly automatically, to cool down) and stood up by itself afterward. The experiment stopped due to a rare software bug (that has now been fixed). Inspection of the machine revealed the failure of the potentiometer of an ankle motor. One foot sensor also stopped working. Although the robot did not meet our goal of days of autonomous operation, the achievement is approaching the same order of magnitude. We are currently looking into replacing the potentiometers inside the servo motors by optical or magnetic encoders to increase the robustness.

V. Reliable state information

To produce reliable and reproducible state information (requirement 2), the robot should be able to accurately measure its complete state and that of the environment. This section describes the robot's sensors.

A. Sensors

The Dynamixel RX-28 servo motors use a potentiometer to record the angular position. According to the specifications, positions can be measured in the range $[0, 300]^\circ$ with 10-bits accuracy, or 0.3°. To verify this, we compared the position readout from 30 servo motors with the readout from a 13-bit absolute magnetic encoder by means of a custom made calibration device. This revealed a large spread in linearity and range of the position information. Local, nonlinear deviations of up to 4° and (linear) range deviations up to $[10, 280]^\circ$ have been observed. A few typical error plots are shown in Fig. 3. In addition, typically 33% of the 1024 possible positions are never observed. Therefore, the local accuracy varies between 0.3° up to 1.0° (when two adjacent positions are never observed).

While any (static) non-linearities in the position readout will not matter for the learning process - the learning agent simply maps states to actions and does not have a notion of absolute angles - it does pose a problem when a servo motor needs to be replaced. In that case, the learning agent would have to adapt to the new mapping from true angles to measured angles, which is typical for each motor. We addressed this by creating a lookup table for each servo motor, which maps the measured angles to the calibrated reference angles.

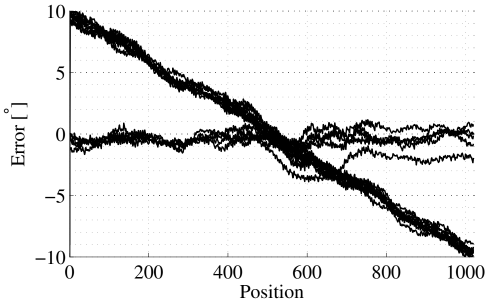

Fig. 3. Typical angle measurement errors from Dynamixel RX-28 servo motors from calibration with a 13-bit absolute magnetic encoder.

The servo motors also provide a velocity signal. However, we found the update rate of that signal to be 7.5Hz, which is too low for our purposes. Therefore, we derive the velocity signal ourselves from the position signal.

A 13-bit absolute magnetic encoder in the hip joint measures the angle between the torso and the parallelogram construction of the boom, which serves as our vertical reference.

The robot lacks a sensor that measures its elevation. Its state is only fully determined from the other sensors if it makes contact with the floor. Therefore, we have to verify during learning experiments that the robot touches the floor at all times.

The foot impact with the floor creates vibrations throughout the combined construction of robot and boom. Such vibrations are visible in the form of large state transitions, especially in the velocity state variables. The vibrations form a disturbance on top of the average dynamics of the robot and can make the learning problem harder. The initial boom construction with two $\varnothing 25 \times \varnothing 23.5$ mm carbon composite tubes was not stiff enough and caused large vibrations, which especially disturbed the torso angle measurements. The lower tube was replaced with a $\varnothing 55 \times \varnothing 53$ mm tube, which increased the stiffness of the construction. This reduced the vibrations and increased their frequency. When calculating joint velocities, we filter the differentiated position signal with a third order Butterworth filter with a low cutoff frequency (10Hz). This further reduces the effect of the vibrations on the velocity signals.

B. Environment

When the environment can be considered static, its properties are not part of the state signal. This requires the floor to be constant over the whole walking circle. We encountered problems with height irregularities in the floor; although they are small - several millimeters - they are significant in relation to the leg length of the robot. This is a disadvantage of our small robot design. Our tests have shown that the floor height variations have a significant negative impact on the walking behavior. Since the position of the robot within the circle is not measured (we do not want the robot to 'memorize' this particular floor), the floor height irregularities have to be treated as noise.

VI. Reliable actuation

The RX-28 servo motors provide position control, velocity control and an open loop actuation mode. Although the manufacturer describes the latter as torque control (literally denoted as "endless turn mode"), this mode is actually voltage control.

The motor torque at a given voltage V_{motor} depends on the temperature of the motor, which is measured inside the RX-28. While temperature effects are probably not noticeable when the motor uses its internal control loop for position or velocity control, this effect is important in voltage control mode. In our robustness test, the pre-programmed controller was based on voltage control.

To comply to requirement 3, we compensated for the temperature dependency. We used the following model (omitting gear box friction) for the output torque M

$$M = kG\frac{U - kG\omega}{R} \tag{1}$$

with k the torque constant, R the winding resistance, G the gearbox ratio, U the voltage and ω the joint angle velocity. The winding resistance R will change with temperature θ according to

$$R(\theta) = R_{\text{ref}}(1 + (\theta - \theta_{\text{ref}})\alpha_{\text{Cu}}) \tag{2}$$

with α_{Cu} the thermal resistance coefficient of copper. The torque constant k will change according to

$$k(\theta) = k_{\text{ref}}(1 + (\theta - \theta_{\text{ref}})TK_{\text{Br}}) \tag{3}$$

with TK_{Br} the temperature dependency of the magnetic flux density of the permanent magnets.

For voltage control, we can keep the generated torque at a given voltage U_{ref} constant with temperature by calculating $U(\theta)$ from (1), (2) and (3) as

$$U(\theta) = U_{\text{ref}}\frac{k_{\text{ref}}}{k(\theta)}\frac{R(\theta)}{R_{\text{ref}}} + \omega k_{\text{ref}} G\left(\frac{k(\theta)}{k_{\text{ref}}} - \frac{k_{\text{ref}}}{k(\theta)}\frac{R(\theta)}{R_{\text{ref}}}\right) \tag{4}$$

The DC motor inside the RX-28 was identified as the Maxon RE-max 17, type 214897. We used catalog values for the motor parameters: $k = 9.92e-3$Nm and $G = 193$. We used $\alpha_{\text{Cu}} = 3.93e-3\text{K}^{-1}$. We chose $TK_{\text{Br}} = 0$, because its true value is unknown and relatively small compared to α_{Cu}. Detailed experiments on individual motors can further improve the model and its parameters.

We tested the temperature compensation by letting LEO walk using the pre-programmed controller. With compensation according to (4), we observed qualitatively similar walking behavior in the temperature range of 45°C to 70°C. Above 70°C, the robot would start to fall more frequently. Without compensation, the robot noticeably fell more frequently at much lower temperatures. We can conclude that temperature compensation is important when using voltage control, and that there is room for improvement in our model (parameters).

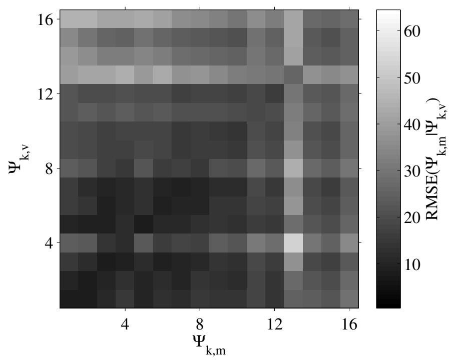

Fig. 4. Root mean squared error comparison between data sets recorded every 20 minutes during the robustness test. The RMSE generally grows when the time between the model data set and validation data set becomes larger, which indicates that the system is not time invariant. Data sets recorded at a later time generally produce larger RMSE values.

VII. System invariability

The MDP framework is defined for a static state transition function T. For this to be true for a robotic learning task, the robot and its environment should be invariable during the length of a learning experiment (requirement 5), or at least change slowly enough so that the learning system is able to adapt in time to the changing situation.

To verify the robot's invariability over time, we performed the following test during the robustness experiment described in Sec. IV. After every 20 minutes of walking, we recorded a two minute data set Ψ_t of state measurements and actuation patterns: $\Psi_1, \Psi_2, .., \Psi_{16}$. We used half of each data set, $\Psi_{t,m}$, to build a state transition model using Local Linear Regression (LLR) (see, e.g., [11]), and the other half, $\Psi_{t,v}$, for validation. If the system behavior does not vary over time, or changes slowly enough, the model built from any $\Psi_{t_1,m}$ should be able to predict state transitions from any other data set $\Psi_{t_2,v}$.

To calculate a measure for the predictive power of data set $\Psi_{t_1,m}$ with relation to data set $\Psi_{t_2,v}$, we did the following. For every state transition $(s_{k+1}|s_k, a_k)$ from $\Psi_{t_2,v}$, the 40 nearest neighbors around (s_k, a_k) in $\Psi_{t_1,m}$ were used to build a local linear model. This model was then used to produce prediction $(\hat{s}_{k+1}|s_k, a_k,)$. The root mean squared prediction error $\mathrm{RMSE}(\Psi_{t_2,v}|\Psi_{t_1,m})$ over the total validation set (of N samples) then served as our measure:

$$\mathrm{RMSE}(\Psi_{t_2,v}|\Psi_{t_1,m}) = \sqrt{\frac{\sum_{k=0}^{N-1} \sum_{ij} (\hat{s}_{k+1}^{(ij)} - s_{k+1}^{(ij)})^2}{N}} \tag{5}$$

This error can be compared with the cumulative prediction error from simultaneously recorded data by calculating $\mathrm{RMSE}(\Psi_{t_k,v}|\Psi_{t_k,m})$. In Fig. 4, $\mathrm{RMSE}(\Psi_{t_2,v}|\Psi_{t_1,m})$ is plotted for all combinations of the 16 data sets that were recorded during the 8 hour test described in Sec. IV.

From the results, two things become apparent. We can see that in general, $\mathrm{RMSE}(\Psi_{t_2,v}|\Psi_{t_1,m})$ increases when t_1 and t_2 are further apart. We can also see that data sets recorded near the end of the testing period generally produce larger cumulative error sums than data sets from the beginning of the experiment. In both cases, the difference is approximately a factor three. We must conclude that the system is not time invariant. Detailed inspection of the recorded data revealed that the position signal from the right ankle's servo motor was heavily deteriorating during the experiment (the signal showed more and more random spikes). This is the most likely cause of the observed time variance of the system.

The noise in the graph can be (partially) explained by the difference in motor temperatures (the robot cooled down after every 30 minutes of walking), floor height variations, and by the fact that the robot would sometimes switch to a slower walking gait during a significant part of the data recording time.

VIII. Real-time control

In the MDP framework, it is assumed that state transitions occur periodically, i.e., that the sampling time is constant (requirement 4). Furthermore, there should be no *control delay*, by which we mean the delay between the moment of measuring the state and the moment of acting upon that state (requirement 6). In reality, these requirements can only be approximately satisfied.

A. Periodic sampling

When the length of a sampling period is not constant, this means that control actions have varying length and therefore varying effect. This makes the learning problem harder, because the more variation in sampling period, the more learning experience is needed to learn the average effect of taking a control action in a certain state. Good periodic behavior requires hard real-time behavior. For LEO, we used Linux with the Xenomai real-time extension. On a sampling period of 6666.6μs, we measured an average standard deviation of 230μs (3.4%). Although this is generally a good result, the magnitude and frequency of allowable sampling time irregularities on RL are as of yet unknown.

B. Control delay

Control delay reduces the stability of a system and the possibility to accurately control it. Control delay is also not modeled in the MDP framework. The consequence of control delay for a RL system is that control actions take effect in future states, instead of immediately. However, the learning update is by default performed as if the control action was performed in the measured state. This can cause convergence problems [12].

To minimize the control delay in our robot, all processing is done on-board, which avoids transporting state and actuation signals over a network (transporting all sensor and actuator lines individually over the central boom joint is impractical). The RX-28 motors are controlled by a serial link, through which they are addressed one after another. Although this causes extra delay compared to parallel connection, the serial bus needs only two data cables to address all motors. Despite the relatively high serial bus speed of 0.5 Mbaud, the process of requesting position data and sending actuation commands still takes several milliseconds. By operating two RS-485 serial ports in parallel, each port controlling the motors of a single leg, this time was roughly cut in half while avoiding extra cabling. The foot sensors and hip sensor are read out over a much faster Serial Peripheral Interface (SPI) bus, causing a small delay in the order of microseconds. In total, measuring the complete state of the system takes on average 1850μs with a maximum of 2350μs.

Delay in the context of RL has been studied, see e.g. [13], but not specifically in the context of robotics. In [12], it is shown that a delay of less than a time step can already have a negative influence on the learning performance of a robotic system. Both studies show the benefit of extending the MDP framework to incorporate the delay. In [12], online temporal difference learning algorithms are proposed that improve the learning performance of a robotic system suffering from delay, while maintaining low computational complexity.

C. Software

LEO's software architecture is event based. The controller software is separated from the rest of the software. The robot produces state information at regular intervals, to which a controller (and also a logger, visualization program, etc.) can subscribe. The state information producing loop is the only periodic loop in the system; controller calculations and logging are purely reactive because the system is event based. The periodic state measurement loop does not wait for actuation signals, which guarantees real-time periodic state information for all its subscribers. The robot provides an actuation interface to the controller, containing functions that can change the actuation signals and return key properties of the actuators.

Because of the abstraction of the state information interface and actuation interface, these interfaces can be replaced by simulated versions. As a result, the controller code can be used both on the robot and in simulation *without modification*, which facilitates development. In addition, any controller, whether pre-programmed or learning, can be connected to the state information and actuation interfaces and can be replaced on-line. For example, during the process of learning to walk, a pre-programmed stand up behavior can seamlessly take over control after a fall.

IX. Conclusions

With LEO, we designed and built a bipedal walking robot specifically for online, autonomous Reinforcement Learning experiments. Due to its boom construction which makes it run in circles and supplies power, its fall protection and its ability to stand up by itself, LEO can perform RL experiments without human assistance.

We checked the robustness of the system by letting it walk for 8 hours using a pre-programmed controller, during which it made $43,000$ footsteps and fell 30 times before failing. Although this is an amount of effort comparable to that needed for a learning experiment, we believe it should be, and can be improved, mainly by finding an alternative for the position recording potentiometers in the actuators.

We made the actuation more predictable by compensating for the temperature of each motor. We verified the invariability of the system over a period of 8 hours by periodically building a model from measured state transitions, and validating this model with measurements recorded later. This showed that the system is not completely time invariant, which was mostly caused by a deteriorating position recording potentiometer in one of the actuators.

We discussed the timing characteristics of our real-time control loop. The control delay between measuring the state and actuating the motors is significant and negatively influences learning performance in simulation. Therefore, the framework of constant delay MDPs is more adequate for online RL experiments on this robot, which requires different learning algorithms.

X. Acknowledgments

The authors would like to thank the Dutch Technology Foundation STW for financially supporting this research, and Guillaume Feliksdal, Jan van Frankenhuyzen, Guus Liqui Lung and Gijs van der Hoorn for their contributions to the development of the robot.

References

[1] J. Morimoto and K. Doya, “Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning,” *Robotics and Autonomous Systems*, vol. 36, no. 1, pp. 37–51, 2001.

[2] R. Tedrake, “Applied optimal control for dynamically stable legged locomotion,” PhD thesis, Massachusetts Institute of Technology, 2004.

[3] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato, “Learning from demonstration and adaptation of biped locomotion,” *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 79–91, 2004.

[4] A. J. IJspeert, J. Nakanishi, and S. Schaal, “Learning attractor landscapes for learning motor primitives,” in *Advances in neural information processing systems*, S. T. S. Becker and K. Obermayer, Eds. Cambridge, MA: MIT Press, 2003, vol. 15, pp. 1547–1554.

[5] S. Collins, A. Ruina, R. Tedrake, and M. Wisse, “Efficient bipedal robots based on passive-dynamic walkers,” *Science*, vol. 307, no. 5712, p. 1082, 2005.

[6] R. S. Sutton and A. G. Barto, “Reinforcement learning: An introduction,” 1998.

[7] E. Schuitema, D. G. E. Hobbelen, P. P. Jonker, M. Wisse, and J. G. D. Karssen, “Using a controller based on reinforcement learning for a passive dynamic walking robot,” *5th IEEE-RAS International Conference on Humanoid Robots*, pp. 232–237, 2005.

[8] S. Troost, E. Schuitema, and P. Jonker, “Using cooperative multi-agent Q-learning to achieve action space decomposition within single robots,” in *18th European Conference on Artificial Intelligence, workshop ERLARS*, Patras, Greece, 2008.

[9] D. Hobbelen, T. de Boer, and M. Wisse, “System overview of bipedal robots flame and tulip: Tailor-made for limit cycle walking,” in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS2008, Nice, France*, 2008, p. 24862491.

[10] D. Hobbelen, “Limit cycle walking,” PhD thesis, Delft University of Technology, 2008.

[11] A. C. Rencher and G. B. Schaalje, *Linear models in statistics*. Wiley New York, 2000.

[12] E. Schuitema, L. Buşoniu, R. Babuška, and P. Jonker, “Control delay in reinforcement learning for real-time dynamic systems: a memoryless approach,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, October 2010.

[13] T. J. Walsh, A. Nouri, L. Li, and M. L. Littman, “Learning and planning in environments with delayed feedback,” *Autonomous Agents and Multi-Agent Systems*, vol. 18, no. 1, pp. 83–105, 2009.