2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)
Munich, Germany, August 20-24, 2018

An EtherCAT-Based Real-Time Control System Architecture for Humanoid Robots

Felix Sygulla, Robert Wittmann, Philipp Seiwald,
Tobias Berninger, Arne-Christoph Hildebrandt, Daniel Wahrmann and Daniel Rixen*

***Abstract*— The design of humanoid robots naturally requires the simultaneous control of a high number of joints. Moreover, the performance of the overall robot is strongly determined by the low-level control system as all high-level software e.g. for locomotion planning and control is built on top of it. In order to achieve high update rates and high bandwidth for the joint control, an advanced real-time control system architecture is required. However, outdated communication protocols with associated limits in the achievable update rates are still used in nowadays humanoid robots. Moreover, the performance of the low-level control systems is not analyzed in detail or the systems rely on specialized hardware, which lacks reliability and persistence. We present a reliable and high-performance control system architecture for humanoid robots based on the ETHERCAT technology. To the authors' knowledge this is the only system, which operates at control rates beyond 2 khz and input/output latencies below 1 ms. Our control architecture includes a learning-based feedforward control strategy to improve joint tracking performance. The improved joint control method and the communication system are evaluated on our humanoid robot LOLA. Our software framework is available online to allow other researchers to benefit from our experiences.**

I. INTRODUCTION

Legged robots have the potential to navigate through very unstructured and uneven terrain, where conventional wheeled robots may fail to find a feasible path. The ability to overcome obstacles by stepping or using the arms as an additional support comes with the drawback of a very high number of actuated joints. The simultaneous control of a high number of degrees of freedoms requires fast and reliable low-level control systems to achieve high control rates and overall reliability of the robot. This is particularly important for the operation in uneven and/or unknown terrain, as the system must detect and react to disturbances quickly, [1].

For robot control, often cascaded control structures are used. With torque-controlled joints, an inner position control loop allows to track positional trajectories, whereas for position-controlled joints, a force-control scheme is used as an inner loop to stabilize the robot. For both concepts the bandwidth of the inner loop is limited by the bandwidth of the outer loop (hardware layer). In general, all high-level software is restricted by the update rates of the hardware layer, which is the connection to the physical machine. With a lot of ongoing research in the field of robotics, the high-level control methods for humanoid robots become increasingly sophisticated. Consequently, also the hardware layer becomes more and more important for the performance of the overall robot.

Despite significant progress in modern communication systems and joint controllers, many robots still include hardware layers with relatively low update rates, often constrained by the used fieldbus technology. Furthermore, the performance of these hardware layers is seldom analyzed in detail, but specified only by the corresponding update rate. The latencies, transmission delays and the synchronicity of simultaneous control commands, however, are equally important to the performance of the whole system. In this paper, we present a control system architecture based on the ETHERCAT[1] fieldbus as well as the corresponding real-time software framework. We focus on the ability for hard real-time constraints, a reliable architecture and backwards-compatibility e.g. for devices with CAN interface. The hardware-layer software framework is available online to allow other researchers to easily equip their robots with this high-performance control system. The high achievable update rates enable the use of an improved joint control concept, which is part of our architecture. This includes an online-learning based feedforward control method as well as target data interpolation. Although we evaluate this control system on our humanoid robot LOLA with position-controlled joints, the general methods are applicable to other humanoid robot platforms and control structures, e.g. with torque-controlled joints.

In the following section, we describe the state of the art for control systems in humanoid robots. The used hardware components are then presented in Section III. In Section IV, we describe the real-time software framework and improved joint control concept of our approach. The system performance of the communication system, joint controller and real-time software is analyzed in Section VI. Lastly, Section VII includes a discussion on the results, conclusion and comments on future work.

II. RELATED WORK

Although modern real-time bus technologies have been available for quite some time, only few humanoids are equipped with these high-performance communication systems. In contrast to distributed control systems with a digital bus communication system, central control concepts were used e.g. in the Honda humanoid robot [2], HRP-2 [3], or Wabian-2 [4]. In these architectures, all sensors and actuators

*Technical University of Munich, Chair of Applied Mechanics, 85748 Garching, Germany. E-mail: felix.sygulla@tum.de

[1]https://ethercat.org

978-1-5386-3593-3/18/$31.00 ©2018 IEEE

are directly attached to I/O interface boards in the central control computer. As an advantage, high update rates are possible as no communication bus is needed. However, the complexity of the system is high, as all peripheral sensors and actuators must be connected directly to the central control unit. Due to their high complexity, centralized systems are in general more error-prone than distributed systems, where some error checking and handling is already executed on the intelligent distributed slaves. With decentralized control concepts, parts of the computational effort can be offloaded to the intelligent actuator controls. This also allows for very high update rates of the local control cycles (e.g. 20 khz current control).

Because of its easiness and reliability, the Controller-Area-Network (CAN) is a prevalent technique for the communication in distributed control systems of humanoid robots. It is used in popular robots such as the HRP robots version 3 and 4 [5], [6], Hubo-2 [7] and the iCub [8] to send and receive data to and from the distributed joint controllers. However, the maximum bandwidth of CAN is relatively low (1 Mbit/s) and communication is only partially deterministic (for high priority messages). Therefore, multiple CAN networks are used in parallel for robots with a high number of degree of freedoms (DoFs). Still, the maximum achievable update rates for joint-controller set-points are considerably low. The DRC-Hubo uses CAN for communication and is limited to a control rate of 200 Hz [9]. With the use of four parallel CAN-Buses CHIMP reaches an update rate of 500 Hz [10]. In addition, the CAN protocol does not allow to compensate for the transmission delays, i.e. allow synchronous execution of commands on the distributed joint controllers.

Several different approaches were used in literature to overcome the drawbacks of CAN. [11] proposes the use of multiple RS422 connections with an effective data rate of 6 Mbits per connection and a central control system with ART Linux operating system. However, it is unclear what kind of protocol and media access control is used in the daisy-chain setup of the RS422 interface. In [12], a real-time communication system based on the Ethernet protocol is developed for the HRP-3P (prototype). It uses a custom protocol to link several bus nodes, which operate an ART-Linux real-time operating system. While originally designed to replace the unreliable central control system of the HRP-2, the AIST group later switched to CAN for the final HRP-3 humanoid "... to improve reliability and maintenance of the system.", [5, p.2476]. PETMAN [13] uses a modified CAN bus to reach an update frequency of 1 khz, details have however not been published so far. Unfortunately, there is only little information on the hardware of ATLAS (1 khz update rate) [14] and no information on the inner structure of Honda's ASIMO. For the robot TORO [15], a Sercos-II bus with a bandwidth of up to 16 Mbit/s is used. Although this enables a 1 khz control rate, the bitrate would probably not allow for much higher update rates. The former hardware design of our humanoid robot LOLA used a Sercos-III bus based on 100 Mbit/s-Ethernet [16]. However, the distributed I/O boards and interfaces to the actual joint drives were in-house made and complex. As this introduces another source of errors, the reliability of the whole solution was limited. For the design of newer robots, the ETHERCAT Bus became increasingly popular, as it is fast, reliable and a widely-used technology. In-house made electronics are often used for the actual joint control and fault handling in such systems [17], [18]. In our experience, this greatly reduces the reliability and persistence of such solutions. Team RoboSimian from the DRC used a hardware structure quite similar to ours, with an ETHERCAT bus and the same commercial joint controllers to control their robot at 1 khz, [19].

In addition to the communication concept, the joint control method is important for the performance of the overall system. Advanced feedforward control strategies provide a high potential for improving joint tracking and can be found in other mechatronic disciplines. For example, [21] uses a self-tuning feedforward strategy for the control of a milling machine. [20] gives a good overview on the application to robots. They present an offline learning based strategy, which works on position-level and uses a previously identified linear error model. In [14], experimentally identified velocity feedforward gains are used to improve positional tracking for torque-controlled joints. To the authors' knowledge, this is the only other application of velocity feedforward gains to bipedal robots.

Compared to the systems found in literature, our control architecture allows to operate at an update rate of 2 khz and above. Furthermore, we use reliable and available hardware modules, and focus on low latency of the overall control loop. The control concept also includes an online-learning based feedforward control method, which operates on velocity-level. As it is based on reinforcement learning, no prior knowledge on the joint design is necessary.

III. HARDWARE OVERVIEW

This chapter gives an overview of the overall mechatronic system of our humanoid robot LOLA, which we used for evaluation of our low-level control system architecture. Special effort is put on sensors, actuators and the real-time system. All joints are electrically actuated using high power brushless DC motors operating at 80 VDC except for the head (24 VDC). Most of the robot's joints are equipped with stiff high ratio Harmonic Drive transmission gears except for the ankle and knee. The ankle joints are actuated over a parallel kinematics using two spatial slider crank mechanisms. The knee joints are based on a roller-screw based linear drive with a four-bar linkage mechanism. Each joint is equipped with an incremental encoder on motor side and an absolute encoder on link side. An additional limit switch is used for safety issues. Following a decentralized concept, local servo controllers from ELMO MOTION CONTROL[2] are used for each joint. They allow to apply feedback control at high sampling rates while the overall wiring effort is reduced. The servo controllers provide an ETHERCAT-interface. An overview of one joint is shown in Fig. 1.

[2]Gold Servo Drives, `http://elmomc.com`

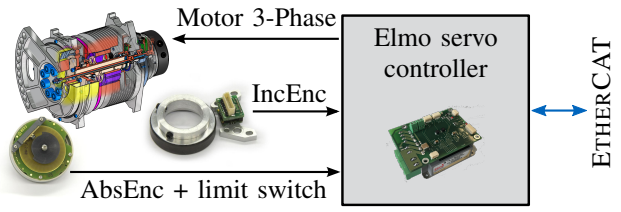

Fig. 1: Cabling overview of one joint: motor, incremental encoder (IncEnc), absolut encoder (AbsEnc), and ELMO servo controller.

The robot has an Inertial Measurement Unit (IMU), which is the commercial high precision system iVRU-FC-C167 from iMAR Navigation. The sensor is rigidly fixed to the upper body of the robot and consists of three fiber-optic gyroscopes and three MEMS accelerometers. The system provides data at a frequency of 200 Hz and runs internal sensor fusion algorithms as well as error compensation models. The generated data can be accessed via CAN. LOLA's feet are equipped with in-house made 6 axis force/torque sensors (FTS) with an optimized shear-beam geometry and strain gauges to measure deformations. They are mounted between ankle joint and foot in order to measure the reaction forces acting on the robot. This data and the contact state are post-processed with two in-house developed microcontroller boards (Cortex-M4, one for each leg), which provide a CAN interface. A commercial CAN gateway with a throughput of 1 Mbit/s is used to integrate CAN data from IMU and FTS into the ETHERCAT bus. It provides a CAN message queue for receiving and sending data from the master. A GPIO ETHERCAT-slave with A/D inputs allows to include additional measurements or triggers of external sensors and devices. It is based on a Beckhoff EK1100[3] with several I/O modules.

All ETHERCAT slaves are connected to a central control unit (ETHERCAT master) via a line topology. The central control unit runs the real-time operating system QNX NEUTRINO 6.6 and is mounted on the back of the robot. It consists of a mini-ITX embedded board with Intel Core i7-4770S@3.1GHz (4x) processor and 8GB RAM. An additional watchdog circuit to enable or disable the motor power is integrated, and directly controlled by digital outputs of the embedded motherboard. The vision system consists of an RGB-D sensor (Asus Xtion PRO LIVE [4]) located at the head as well as an onboard computer identical to the one of the central control unit. The vision computer runs a Linux OS and both computers are mounted on a common frame (cf. Fig. 2). The onboard computers and an external monitoring computer use TCP/UDP to communicate via standard Ethernet. An overview of the hardware is shown in Fig. 2.

[3]https://www.beckhoff.com
[4]https://www.asus.com/Multimedia/Xtion_PRO_LIVE

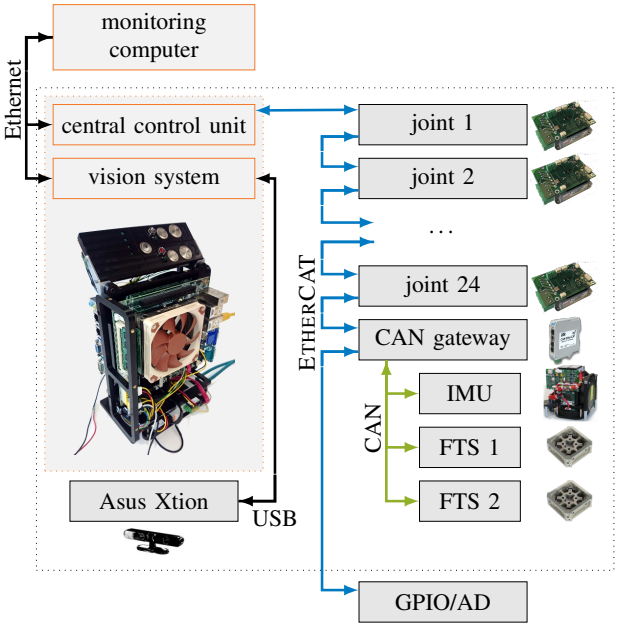

Fig. 2: Mechatronic network overview. The elements in the black dotted frame are physically located on the robot.

IV. COMMUNICATION AND CONTROL SOFTWARE

The walking control system of LOLA is based on a hierarchical approach, see Fig. 3. Based on user-inputs, a high-level planning module first generates ideal trajectories $\boldsymbol{w}_{id}$ in task-space, which are then modified by predictive and local stabilization methods to reject external disturbances. The desired joint trajectories $\boldsymbol{q}_d$ are finally calculated by inverse kinematics and sent to the decentralized position-controllers of each joint. Simultaneously, sensor data is read and stored for the next planning/control cycle. While the planning of trajectories is triggered every time a new walking step is executed, the local stabilization strategies run with a cycle time of $\Delta t_{\text{cont}} = 1\,\text{ms}$. Therefore, we aim at an update

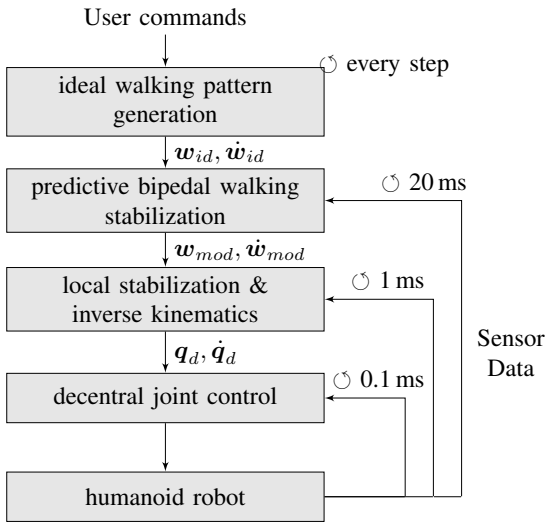

Fig. 3: Overview of the walking control system.

rate of at least 1 kHz for sensor and target data.

A. Overview on Lola's Control Software

The hierarchical walking control system basically consists of three main parts, which are realized via parallel processes on our central control unit. This general software-structure has already been introduced in [22] and can be summarized as follows:

- *The Planning Process:* Contains all software components to generate the ideal trajectories based on current user input. Predictive Stabilization also takes place here.
- *The Stabilization Process:* Receives the trajectories from the planning process via a shared memory interface with a FIFO-based ring-buffer. Based on the desired and actual state of the robot (including all sensor data), a local stabilization is executed.
- *The Hardware-Driver Process:* Gets desired values for the current time-step from the Stabilization Process. Handles the communication with the actual hardware and runs other low-level tasks. Sensor data received via the communication bus is sent to both the Planning- and Stabilization Process via a shared memory interface.

In the following, a new hardware-layer for this software concept is described. We use the commercial EtherCAT master stack from Acontis[5] on the QNX real-time operating system. The code for our hardware layer framework (with interfaces to the commercial EtherCAT master stack as well as the commercial motion controllers) is available online[6]. Although the code is optimized for QNX Neutrino, our control concept will work with any real-time operating system.

B. Real-Time Bus Middleware

To allow an easy integration of future real-time bus systems and/or unproblematic change of the EtherCAT master-stack, we implemented an additional layer to separate the hardware-near software from the application code. It may be used to wrap any PDO/SDO[7] based communication system. This middleware provides an easy and powerful interface on the application side to write and read data from the EtherCAT bus via "Bus Variables". On the other side, it contains interfaces to an EtherCAT master stack. Basically, a Bus Variable is an instance of a special class used in the application side of the hardware driver. This class represents a variable of a certain predefined primitive data type (such as int, float, char,...) and may be used as any standard variable in the application code. However, the application can tell the middleware software to link the data of this variable to a certain EtherCAT slave variable (PDOs). If the value of an output variable is changed, the middleware automatically sends the new data to the corresponding slave. Equivalently, the data in input variables is updated every time a new EtherCAT frame is received. The connection between the

[5]http://www.acontis.com/eng/products/ethercat/ec-master/

[6]https://github.com/am-lola/lolaCAT

[7]Process- (PDO) and Service Data Objects (SDO)

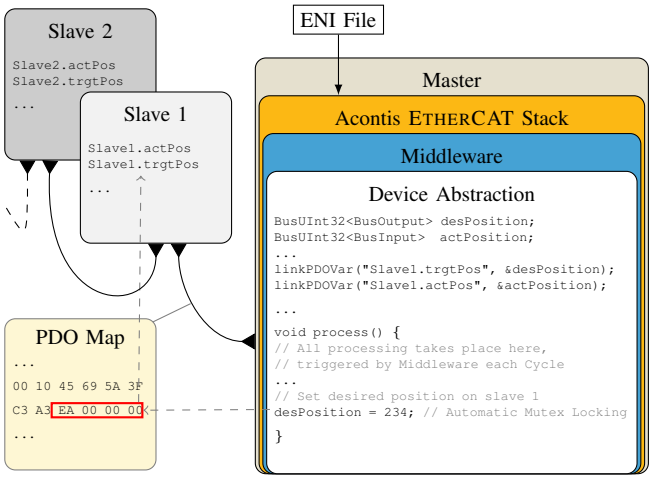

Fig. 4: The concept of Bus Variables, which automatically links application code to data on the EtherCAT slaves.

Bus Variables in the application code and the variables on the slaves is made through the respective slave and variable names defined in the EtherCAT Network Information File (ENI). Bus Variables are thread-safe and implement an automated data-type checking during runtime. The concept is visualized in Fig. 4. For asynchronous communication, the middleware implements an interface to send and receive Service Data Objects (SDOs). Furthermore, communication errors on the bus are handled and delegated to the safety code in the hardware driver.

C. Device Abstraction Layer

On top of the middleware layer, all devices on the bus are represented by Device Abstraction Classes. These map internal logic and physical behavior of the slaves to the software. The Device Abstraction Classes are derived from a general "BusSlave" class provided by the middleware, which enables the use of Bus Variables. In our setup, we use a device class for the Elmo Motion Controllers, as well as for the CAN gateway, our IMU, and the FTS sensors. Every device class implements a state-machine, specialized error handling and provides a high-level interface to control the corresponding device in the hardware driver. On the motion controller device class for example, *get* and *set* methods for the actual and desired position are provided.

D. Low-Level Hardware Driver

The Hardware-Driver Process contains both the device abstraction layer and the middleware. It spawns several child threads for error handling and safety measures, the bus communication, logging, and the inter-process communication with the other two control framework processes. The main thread basically operates as the central data hub, i.e. it processes raw data sent from and to the EtherCAT slaves, and sends interpreted actual values to - as well as accepts new target values from - the stabilization and planning process. Furthermore, a state machine represents the global state of

ID	Name	Description
14 ↷	waitForNextCycle	Block the thread until the EtherCAT Cycle Time has elapsed
4	updateDes	Get new desired values from the stabilization process (if available)
6	waitForRXData	Block the thread until new input data has been processed in EC-thread
10	procData	Process raw input data in Device Abstractions
11	updateAct	Push processed input data to shared memory (this triggers a new calculation of target data)
13 ↶	updateSTM	Execute hardware driver state-machine logic and error handling

TABLE I: Methods called in the main thread of the Hardware Driver.

the hardware system and reacts to changed user commands or hardware states. In case of an error, a software watchdog in a separate child process automatically triggers an emergency stop with motor voltage shutdown. The main thread executes these tasks in a loop and is synchronized to a higher-priority "EtherCAT thread" (EC) to ensure minimum latency of input and output data. Within the EC-thread, methods of the bus middleware and the EtherCAT master stack are executed. This includes copying data between the Bus Variables and the actual EtherCAT PDO Map as well as sending the cyclic and acyclic EtherCAT frames. To keep the timing between consecutive EtherCAT cycles precise, a separate timing task with highest priority is used to trigger execution of the EC-thread. All important methods executed in the main and EC-thread are described in Tab. I and Tab. II. Note that the communication with the stabilization and planning processes is asynchronous, i.e. the main thread is not blocked if no new data is available. This allows to set the bus cycle time Δt_{bus} to values equal or lower than the stabilization update interval Δt_{cont}.

V. Improved Decentralized Joint Control

Each decentralized joint controller receives target position, target velocity and electric current feedforward values from the central control unit. The electric current feedforward values are calculated from inverse dynamics of the robot and the motor specifications. In case the bus system runs faster than the central control loop, $\Delta t_{\text{bus}} < \Delta t_{\text{cont}}$ with $\Delta t_{\text{cont}} = \Delta t_{\text{bus}} \cdot n$, new trajectory data is only available every n_{th} bus cycle. For the intermediate bus cycles, the last target position for each joint is linearly extrapolated using the last target velocity. This enables interpolated set-points for the joint controllers with higher update rates than the central control unit's stabilization process.

ID	Name	Description
1 ↷	waitForTimingEvent	Wait for an event from the timing thread
2	signalNewCycle	Unblocks the main thread
3	RX	Process the frames received in the last cycle (EtherCAT Stack)
5	copyRXData	Copy incoming data from the EtherCAT Stack to the Bus Variables
7	copyTXData	Copy outgoing data from the Bus Variables to the EtherCAT Stack
8	signalRXData	Signal main thread new incoming data is available
9	sendFrames	Queue cyclic EtherCAT frames to be sent on the bus
12 ↶	adminStuff	Send acyclic frames and execute administrative stuff of the EtherCAT Stack

TABLE II: Methods called in the EtherCAT thread of the Hardware Driver.

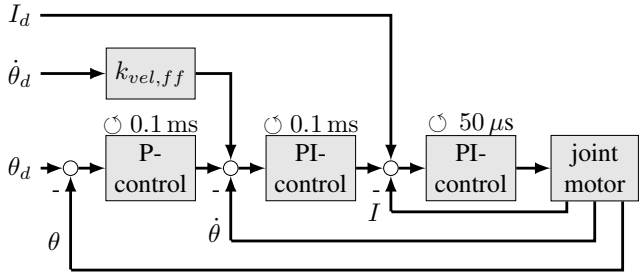

Fig. 5: Cascaded decentral motor control.

The structure of one decentralized control unit is shown in Fig. 5. It consists of a P-control for the motor position, a PI-control for the velocity and PI-control for current. Sampling times are 0.1 ms for position and velocity and 50 µs for current feedback. Feedback of the motor motion $(\theta, \dot{\theta})$ is obtained from the incremental encoders and current (I) is measured by an integrated sensor. Additional feedforward values for velocity $(\dot{\theta}_d)$ and current (I_d) are commanded to the controllers. We identified that a modification of the feedforward velocity $\dot{\theta}_d$ improves the overall tracking performance. Experiments revealed that a constant gain k_{ff} multiplied by $\dot{\theta}_d$ can be used to estimate the tracking error of the position $\Delta\theta = \theta_d - \theta$ for the closed loop system

$$\Delta\theta \approx k_{ff}\dot{\theta}_d. \tag{1}$$

Considering the gain of the position controller K_p this can be used to add a corresponding factor to the overall commanded velocity

$$k_{vel,ff} = 1 + K_p k_{ff} = 1 + k_e. \tag{2}$$

To enable an automatic computation of the feedforward gain k_e a reinforcement learning [23] based strategy is used. This has the advantage that the optimal gains can be identified online when the joint control system or motor are changed. Initially, all gains are set to $k_e = 0$, which equals a standard velocity feedforward scheme. The robot is stepping in place and its joint tracking errors are recorded for the learning process. A cost function is defined with the root mean squared error (RMSE) over a time period of two steps. The simple policy of increasing k_e by 0.1 increments as long as the cost function decreases is used. This is performed for all joints simultaneously. As example, the learning progress for the hip joint is depicted in Fig. 6. The algorithm finds the best gain $k_e = 1.0$ and can reduce the RMSE of the tracking error by approximately 90 %. Similar results are obtained for all other joints.

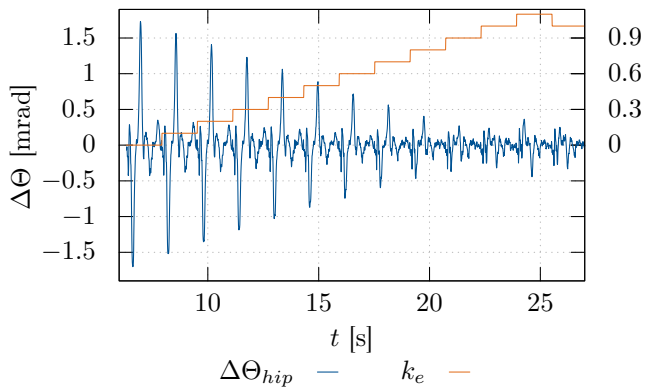

Fig. 6: Tracking error and gain k_e during the learning process of the hip joint.

VI. System Performance

We evaluated the performance of our real-time system in three different ways. First, by measuring timing and statistics on the software processes. Second, by evaluating the performance of the EtherCAT Bus communication. And third, by analyzing the tracking behavior of the joint controllers.

A. Software Performance

The performance and quality of the software system is evaluated on our QNX real-time system by using an event tracelogger. This enables to get precise information on the timing of individual program sections. For the test we chose an update rate of 2 khz for the communication bus. The exact execution times for all operations in one EtherCAT bus cycle (500 μs) are shown in Fig. 7. For a description of the shown program sections, refer to Subsection IV-D. The total cpu time per cycle is only 86 μs, which is well below the actual bus cycle time (500 μs). New desired values are copied to the PDO map right before the frames are sent on the bus, which allows for minimal latency in desired values. The actual sensor values however are taken from the received data from the Ethernet frame of the last cycle. This ensures all frames have been received by the time the update is done and can lead to an input-data latency of up to $\Delta t_{bus} + 20\,\mu$s. Note that we do not know the latency between the execution of *sendFrames* and the actual beginning of the communication on the bus. Furthermore, the bus latency, which is analyzed in the following section, must be added. Note that the pairs *copyRXData*, *procData* and *updateDes*, *copyTXData* access the same memory areas (protected via mutual exclusion). However, simultaneous requests to the same memory are also excluded by the order of the function calls to reduce jitter and latency[8]. By measuring the absolute time (with a high-precision timer) between consecutive bus cycle loops, we observe a high precision of the timing in the software concept. The average software cycle time is 500.34 μs, with a standard deviation of 1.02 μs.

[8]Because of the QNX scheduling timeslice, mutual exclusion with try-lock commands would still lead to large delays

B. Bus Performance

We use a total of 294 Bus Variables for the communication with the 25 slaves, which equals 1026 bytes of input/output data. As input and output variables can share the same space in an EtherCAT frame (data is written on-the-fly), one Ethernet frame with 846 bytes total size is sufficient in our case. With a link speed of 100 Mbit/s, the theoretical transmission delay is

$$\tau = \frac{8\,\text{bits/byte} \cdot 846\,\text{bytes}}{100\,\text{Mbit/s}} = 67.7\,\mu\text{s}. \quad (3)$$

However, propagation delays (copper wires) and latencies within the slaves must be added to get the minimum bus cycle time Δt_{min}. As it is difficult to estimate these values, we measured the total delay by attaching an Ethernet switch to the bus and analyzing the packages with the open-source software Wireshark[9]. This is a conservative measurement, as additional latencies from the switch and measurement computer increase the total packet delay. The mean value of the minimum bus cycle time (or total delay) over a period of 10 seconds is $\Delta\bar{t}_{\text{min}} = 124.1\,\mu$s with a standard deviation of 10.2 μs. Consequently, much higher bus update rates (> 4 khz) are possible with the EtherCAT bus and our framework. The results indicate that - based on a conservative scaling estimation - around 50 joints can be controlled at an update rate of 2 khz.

To compensate for the transmission delays, which are different for the output data sent to each slave, we use the Distributed Clocks technology supported by the EtherCAT bus and our motion controller slaves. We operate the system in BusShift mode, with the first motion controller as a reference clock. Fig. 8 shows the difference of the master and the slave clocks to the global system time during bus initialization. Once the bus is in operational state, the mean absolute error for the slave clocks is as low as 0.12 μs (standard-deviation of 0.11 μs). The mean absolute error for the master clock is 3.25 μs (standard-devation of 2.45 μs), and is limited by the drift of the clock in the consumer-type master-computer. Still, the synchronization of the system time ensures target set points for the motion controllers are executed at the same point in time, independently of the bus delay.

C. Joint Controller Performance

We tested the performance of the joint-controllers in walking scenarios with our humanoid robot LOLA. For this experiments, the robot is commanded to walk 5 m straight on even terrain with a speed of 2.7 km/h. For comparison of the joint tracking performance, the root mean squared position tracking error of the hip joint (flexion) is computed from joint encoder measurements. The results are shown in Tab. III for settings without learned feedforward gains ($k_e = 0$, ref), with learned feedforward gains (1khz w/ff), and with feedforward gains as well as a higher bus update rate and extrapolation of target positions (2khz w/ff). Relative to the

[9]http://www.wireshark.org

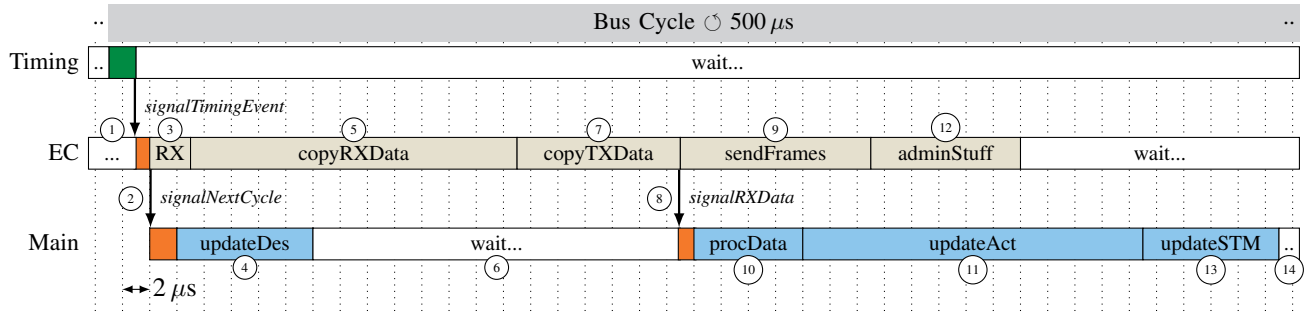

Fig. 7: Timing of the three main threads in the hardware layer process. The timing thread unblocks every 500 μs. Orange sections describe CPU-time used by the QNX scheduler. The sections are numbered according to the IDs in Tab. I and II.

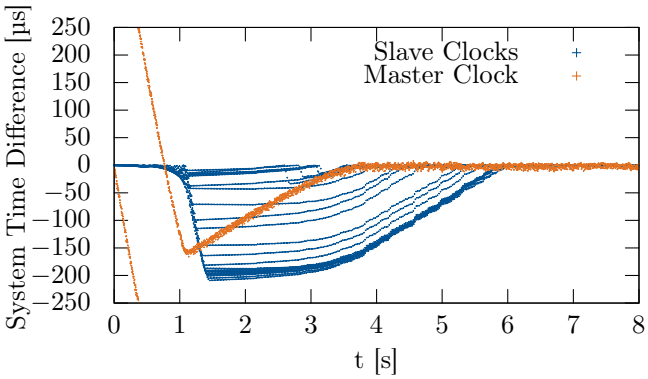

Fig. 8: Distributed Clocks system time difference

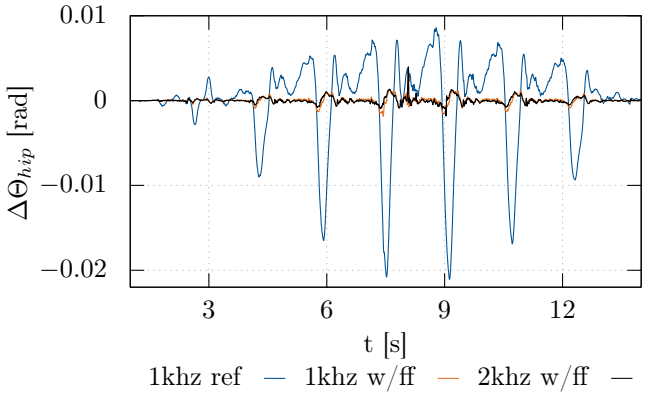

Fig. 9: Closed loop tracking error of the hip-joint (flexion) without learned feedforward gains (1khz ref), with learned feedforward gains (1khz w/ff), and with 2khz bus update rate including target data extrapolation and learned feedforward gains (2khz w/ff).

reference, our methods reduce the tracking error by $\approx 94\,\%$. The maximum tracking error is also reduced significantly, which is observable in the time-domain data in Fig. 9. Note that the standard velocity feedforward control scheme is used throughout all three variants.

Additionally, we analyzed the bandwidth for the positional tracking of the closed loop. Hereby, we excited the hip-joint (flexion) with target trajectories based on a chirp signal, which contains frequencies from 10 Hz to 500 Hz at an amplitude of 0.0002 rad. By measuring the response of the closed loop system via the incremental encoders, we generated an estimation for the transfer function, which is shown in Fig. 10. The raw data is additionally filtered in the frequency domain to reduce the effect of noise. We identified the bandwidth of the system at 0 dB to 241.28 Hz. Given this includes the whole mechanical components from hip to toe, the attained bandwidth is beyond our expectations.

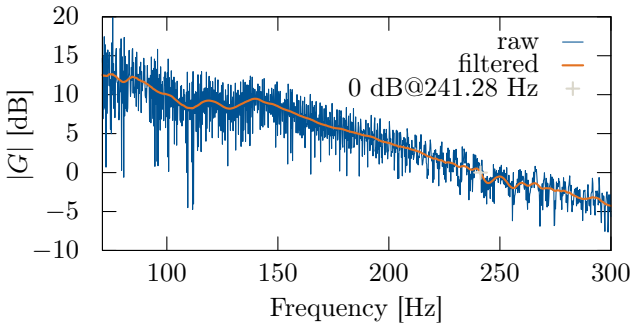

Fig. 10: Bandwidth for positional tracking of the closed loop system (hip flexion). The data is obtained by estimating the transfer function from measurement data.

TABLE III: Root mean squared (RMS) error on the hip joint in walking experiments for reference, with learned feedforward gains and for a 2khz update rate with extrapolated target positions and learned feedforward gains.

$\Delta t_{\text{cont}} = 1$ khz	$\Delta t_{\text{bus}} = 1$ khz ref	$\Delta t_{\text{bus}} = 1$ khz w/ff	$\Delta t_{\text{bus}} = 2$ khz w/ff
RMS tracking error [rad]	0.0047	0.0004	0.0003

VII. Conclusion

In this paper we presented a control system architecture for humanoid robots. Hereby, we used commercial motion control modules at the joint, and the EtherCAT technology for communication. In order to integrate a commercial EtherCAT master stack to our robot's control software, we implemented a middleware layer. This software is available online and provides an easy and powerful interface for PDO/SDO communication with EtherCAT devices. Tests on our humanoid robot LOLA showed a total control loop latency of only $\Delta t = 644\,\mu$s at a 2 khz bus update rate. By utilizing the distributed clocks functionality of the EtherCAT bus, we attained high synchronicity ($\Delta t_{\text{sync}} < 1\,\mu$s) for the set-point commands of each joint controller. Comparing with the only other example we could find in literature, the control loop of the humanoid robot TORO has a $2\,\text{ms}$ latency at a 1 khz bus update rate [15].

In addition we presented a feedforward control method based on reinforcement learning as part of our low-level control system architecture. Furthermore, we used our high bus update rates to extrapolate the target data for each joint controller. Using these techniques, the tracking error for a normal walking sequence is reduced by $\approx 94\,\%$. We experimentally identified the bandwidth of the closed-loop joint controllers to $\approx 240\,\text{Hz}$. For future work, we will concentrate on further methods to increase the overall system's tracking performance.

Acknowledgment

This work is supported by the Deutsche Forschungsgemeinschaft (projects BU 2736/1-1, RI 2451/7-1).

References

[1] C. G. Atkeson, B. P. Babu, N. Banerjee, D. Berenson, C. P. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, P. Franklin, M. Gennert, J. P. Graff, P. He, A. Jaeger, J. Kim, K. Knoedler, L. Li, C. Liu, X. Long, T. Padir, F. Polido, G. G. Tighe, and X. Xinjilefu, "No Falls, no Resets: Reliable Humanoid Behavior in the DARPA Robotics Challenge," in *IEEE/RAS International Conference on Humanoid Robots*, 2015.

[2] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, "The Development of Honda Humanoid Robot," in *IEEE International Conference on Robotics and Automation*, 1998.

[3] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi, "Humanoid robot HRP-2," *IEEE International Conference on Robotics and Automation*, 2004.

[4] Y. Ogura, H. Aikawa, K. Shimomura, A. Morishima, H.-o. Lim, and A. Takanishi, "Development of a New Humanoid Robot WABIAN-2," in *IEEE International Conference on Robotics and Automation*, 2006.

[5] K. Kaneko, K. Harada, F. Kanehiro, G. Miyamori, and K. Akachi, "Humanoid Robot HRP-3," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

[6] K. Kaneko, F. Kanehiro, M. Morisawa, K. Akachi, G. Miyamori, A. Hayashi, and N. Kanehira, "Humanoid Robot HRP-4 - Humanoid Robotics Platform with Lightweight and Slim body," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[7] B. K. Cho, S. S. Park, and J. H. Oh, "Controllers for Running in the Humanoid Robot, HUBO," in *IEEE/RAS International Conference on Humanoid Robots*, 2009.

[8] G. Metta, L. Natale, F. Nori, G. Sandini, D. Vernon, L. Fadiga, C. von Hofsten, K. Rosander, M. Lopes, J. Santos-Victor, A. Bernardino, and L. Montesano, "The iCub humanoid robot: An open-systems platform for research in cognitive development," *Neural Networks*, vol. 23, no. 8-9, pp. 1125–1134, 2010.

[9] M. Zucker, S. Joo, M. X. Grey, C. Rasmussen, E. Huang, M. Stilman, and A. Bobick, "A General-purpose System for Teleoperation of the DRC-HUBO Humanoid Robot," *Journal of Field Robotics*, vol. 32, no. 3, pp. 336–351, 2015.

[10] A. Stentz, H. Herman, A. Kelly, E. Meyhofer, G. C. Haynes, D. Stager, B. Zajac, J. A. Bagnell, J. Brindza, C. Delli, M. George, J. Gonzalez-Mora, S. Hyde, M. Jones, M. Laverne, M. Likhachev, L. Lister, M. Powers, O. Ramos, J. Ray, D. Rice, J. Scheifflee, R. Sidki, S. Srinivasa, K. Strabala, J.-P. Tardif, J.-S. Valois, J. M. V. Weghe, M. Wagner, and C. Wellington, "CHIMP, the CMU Highly Intelligent Mobile Platform," *Journal of Field Robotics*, vol. 32, no. 2, pp. 209–228, 2015.

[11] J. Urata, Y. Nakanishi, K. Okada, and M. Inaba, "Design of High Torque and High Speed Leg Module for High Power Humanoid," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[12] K. Akachi, K. Kaneko, N. Kanehira, S. Ota, G. Miyamori, M. Hirata, S. Kajita, and F. Kanehiro, "Development of Humanoid Robot HRP-3P," in *IEEE/RAS International Conference on Humanoid Robots*, 2005.

[13] G. Nelson, A. Saunders, N. Neville, B. Swilling, J. Bondaryk, D. Billings, C. Lee, R. Playter, and M. Raibert, "PETMAN: A Humanoid Robot for Testing Chemical Protective Clothing," *Journal of the Robotics Society of Japan*, vol. 30, no. 4, pp. 372–377, 2012.

[14] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous Robots*, vol. 40, no. 3, pp. 1–27, 2015.

[15] J. Englsberger, A. Werner, C. Ott, B. Henze, M. A. Roa, G. Garofalo, R. Burger, A. Beyer, O. Eiberger, K. Schmid, and A. Albu-Schäffer, "Overview of the torque-controlled humanoid robot TORO," in *IEEE-RAS International Conference on Humanoid Robots*, 2014.

[16] V. Favot, T. Buschmann, M. Schwienbacher, A. Ewald, and H. Ulbrich, "The Sensor-Controller Network of the Humanoid Robot LOLA," in *IEEE/RAS International Conference on Humanoid Robots*, 2012.

[17] H. Kaminaga, K. Tianyi, R. Masumura, M. Komagata, S. Sato, S. Yorita, and Y. Nakamura, "Mechanism and Control of Whole-Body Electro-Hydrostatic Actuator Driven Humanoid Robot Hydra," in *Springer Proceedings in Advanced Robotics 1*, vol. 1. Springer International Publishing AG, 2017, pp. 656–665.

[18] M. Ferrati, A. Settimi, L. Muratore, N. G. Tsagarakis, L. Natale, and L. Pallottino, "The Walk-Man Robot Software Architecture," *Frontiers in Robotics and AI*, vol. 3, no. May, p. 25, 2016.

[19] S. Karumanchi, K. Edelberg, I. Baldwin, J. Nash, J. Reid, C. Bergh, J. Leichty, K. Carpenter, M. Shekels, M. Gildner, D. Newill-Smith, J. Carlton, J. Koehler, T. Dobreva, M. Frost, P. Hebert, J. Borders, J. Ma, B. Douillard, P. Backes, B. Kennedy, B. Satzinger, C. Lau, K. Byl, K. Shankar, and J. Burdick, "Team RoboSimian: Semi-autonomous Mobile Manipulation at the 2015 DARPA Robotics Challenge Finals," *Journal of Field Robotics*, vol. 34, no. 2, pp. 305–332, 2016.

[20] M. Grotjahn and B. Heimann, "Model-based Feedforward Control in Industrial Robotics," *The International Journal of Robotics Research*, vol. 21, no. 1, pp. 45–60, 2002.

[21] S.-J. Huang and C.-C. Chen, "Application of self-tuning feed-forward and cross-coupling control in a retrofitted milling machine," *International Journal of Machine Tools and Manufacture*, vol. 35, no. 4, pp. 577–591, 1995.

[22] T. Buschmann, V. Favot, S. Lohmeier, M. Schwienbacher, and H. Ulbrich, "Experiments in Fast Biped Walking," in *IEEE International Conference on Mechatronics*, 2011.

[23] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge: MIT press, 1998.