Integration of a spine surgery robot controller using EtherCAT and ROS2

Veysi ADIN [1,2], Chunwoo Kim [1,2]

Abstract— **With the increasing complexity of the robot systems, system integration is becoming more challenging. Additionally, there is a gap between the robots developed in the academia and industry. This is important especially in the medical robotics, as the robot needs to comply with several industry standards for translation of the technology from academia to clinic. To address integration and industry compatibility issues, in this paper, we propose industry compatible hardware and software integration method to deal with increasing complexity in medical robotics. We present system integration of medical robot system using EtherCAT fieldbus and ROS2 middleware, both of which are developed for industrial level applications. As an example, EtherCAT master for controlling spine surgery robot is implemented and real-time performance is evaluated.**

I. INTRODUCTION

Nowadays, robots are not just an actuated mechanism performing predefined tasks in a structured environment, but robots are interacting with the unstructured environment. This change in robotics led to more sensors, actuators and powerful controllers usage in the system, and more time and engineering efforts are required to integrate all these components into a single system. With the increasing complexity of a robot system, several system integration methods have been proposed to facilitate the system integration process. [1]–[4].

System integration can be separated into two parts; physical integration and software integration. Physical integration is process of establishing a physical connection between the components and providing a communication method through this connection. Various fieldbus and protocols such as Controller Area Network (CAN), RS232, RS485, Universal Serial Bus (USB), IEEE 1394 (Firewire), and EtherCAT, are used for physical integration. With the increasing complexity and dynamic requirement of the robot system, fieldbus with real-time capability, that is, high bandwidth for reliable and fast communication, is necessary. CAN bus is one of the most widely adopted fieldbus, but it is limited due to its bandwidth (1Mbps). RS232 and RS485 are limited by their bandwidths as well (1 Mbps/10 Mbps). USB and Firewire satisfy the requirement in terms of bandwidth and have been successfully used in control of experimental surgical robots project [5], [6]. However, both project required custom USB/Firewire IO boards, which limits the integration of additional sensors. Also, Firewire is becoming less available in market. EtherCAT is relatively new compared to previous fieldbuses, it provides fast (up to 10Gbps) and flexible protocol, uses standard Ethernet card for IO and many commercial actuators and IO boards are now available with EtherCAT interfaces.

Software integration is process of building a robot control software that collects information from the physically integrated components, execute control algorithm using the collected information, and sends outputs generated from the algorithm to the components. This requires sharing of the digitized information among the multiple threads and processes running in one or multiple computers in the robot. Various robotics middleware such as ROS [7], Orocos [8], cisst [9], ROS2 [10] has been developed to facilitate the software integration. Middleware provides hardware abstraction, component management and message passing between processes and threads, allowing developers to apply component based software engineering methods [11] to robot control software. With the help of middleware, robot control software can be split into loosely coupled components, resulting reduced complexity, separation of concerns for easier debugging, and improved re-usability. Among the middlewares, ROS is most widely used in robotics community. Since it's initial release in 2007 researchers all around the world built hundreds of robots and drivers and shared it with community.

In field of medical robotics, system integration is particularly challenging due to safety critical nature of the medical robots. Although there are no specific standards regarding physical and software integration of medical robots, general standards for medical devices and software such as IEC-60601 [12] (safety and essential performance of medical electrical equipment standard), IEC-62304 [13] (medical device software life cycle standard) and IEC80601-2-77 [14] (Medical electrical equipment — Part 2-77: Particular requirements for the basic safety and essential performance of robotically assisted surgical equipment) should be considered. To comply with these standards, in terms of physical integration, high speed, reliable, deterministic communication protocol is required. In terms of software integration, system should have well-defined states and transition between states. System integration efforts in medical robotics domain led to implementation of domain-specific libraries such as cisst [9] library and several other integration methods such as Saras project [2], and ROS based operation room integration methods [1].

In this paper, as a part of a larger effort for developing a software framework that can be used for various medical robot and device developments, we present a system integration effort for spine surgery robot controller. Our integration

[1]Veysi ADIN and Chunwoo Kim are with the Center for Healthcare Robotics, Korea Institute of Science and Technology, Seoul, Korea

[2]Veysi ADIN and Chunwoo Kim are with the Division of NT-IT, University of Science and Technology, Daejeon, 34113, Korea

uses EtherCAT as a fieldbus for physical integration ROS2 as a middleware for software integration. Implementation details and realtime performance evaluation by jitter measurement is presented in subsequent sections.

II. Implementation Details

A. EtherCAT protocol

EtherCAT is an Ethernet based fieldbus standard focusing on real-time requirements for automation, and it is designed to achieve very low cycle times typically less than 100 µs with low jitter less than 1 µs [15]. EtherCAT implements physical and data link layer of Open Systems Interconnection (OSI) model and its specification is freely available through EtherCAT Technology Group [16]. As a result many different higher layer protocols such as CANopen over EtherCAT (CoE), File Access over EtherCAT (FoE) etc. [15] are available. Additionally, many robot hardware including I/O boards, motor drivers, sensors, and actuators with EtherCAT interface readily are available in the market. In short, EtherCAT provides fast (100Mbps/1Gbps/10Gbps), reliable, and scalable communication, and it's an open protocol with several open-source libraries, which makes it the best option for research and development.

EtherCAT protocol implements master-slave topology. EtherCAT uses standard IEEE 802.3 Ethernet frames and physical layers, and it uses on-the-fly processing of data for passing data to the next slave in the bus. A control PC must implement a EtherCAT master which synchronize this communication of data frames. Fortunately, there are open-source libraries for master implementation such as SOEM [17] and Etherlab [18]. Among these open-source implementations SOEM is not suitable for applications requiring highly deterministic and low cycle timing, because implementation of the library itself is not real-time [19]. SOEM runs on user-space, means that it communicates with network stack to transfer process data to network driver, and network driver will send process data to EtherCAT network. Therefore,SOEM has no control over latency caused by operating system.In Etherlab master library, EtherCAT master is implemented as a kernel module, it can communicate directly with network drivers without involving network stack, it has control over latency caused by operating system. Etherlab library provides EtherCAT capable native drivers for commonly used network interface controllers, including generic network driver. It supports any real-time extension for Linux, since its architecture is independent. Additionally, it provides set of tools to monitor bus configuration, configure slaves and debugging [20]. Therefore, we chose to use Etherlab master library developed by IgH. As for the EtherCAT slaves, may motor controllers provide interfaces for CANopen over EherCAT (CoE). For additional IOs such as limit switch, several custom slaves based on most common microcontrollers such as Arduino and Raspberry Pi [21] are available to process any sensor data into packet in the EtherCAT bus.

B. ROS2

Robot operating system (ROS) provides a flexible framework for robotic software development.It provides; hardware abstraction, publish/subscribe mechanism allowing multiple processes to communicate with each other, several libraries and tools to develop robotic software rapidly.

ROS has been used by researchers for a long time, to integrate their physical implementation into ROS as a part of software integration.However, ROS is not suitable for real-time applications, researchers used several methods to provide real-time support such as integrating OROCOS with ROS and using Xenomai real time framework. [22] [23]. To address real-time incompatibility of ROS, Open Source Robotics Foundation (OSRF) started development of ROS2 in 2015 and released first non-beta release in 2017 [24]. ROS2 changes transport system of ROS1 (TCPROS) with Data Distribution Service (DDS), which is an end-to-end middleware providing high-performance, inter-operable, real-time, scalable data exchanges using a publish–subscribe pattern. Additionally, DDS have been used in several mission critical applications such as space systems, flight systems, battleships and medical devices [25].

Design idea of ROS2 was to create open-source, industry compatible software framework for robotics developers, we can see realization of design ideas of ROS2 from implementation of several features :

- Life-cycle node: Life cycle is a managed node, which includes state machine and defined transition between states. A state machine is an abstract representation of all possible states that a machine or software can be in. When states and transitions between states are well-defined, system become safer.Therefore, life-cycle node with well-defined state machine is particularly important for medical device software.
- Quality of service settings (QoS): QoS is a technology which manages data traffic to reduce packet loss, latency and jitter on a network, it controls and manages resources by adjusting priority for specific data types. Therefore, developers can have control over network by adjusting QoS settings for each topic.
- Data-Distribution Service: One of the major changes between ROS and ROS2 is DDS usage, DDS is a networking middleware providing publish-subscribe pattern.DDS has a default discovery system, therefore DDS programs do not require ROS master to communicate, which makes system, more fault tolerant and flexible. [25]

ROS2 has well-documentation, great tutorials available and programming examples on real-time implementations [26]. Compared to cisst and Orocos, ROS2 has better documentation, bigger community, more examples and drivers for several common sensors. This is important, because well documentation and having more user, shortens the development and debugging time.

C. Implementation Overview and Real-time performance

System integration presented in this paper is for con-

troller of a spine robot currently under development in our institution. Fig 1 shows the physical and software integration of the controller. In the physical integration part, the system consists of three motor driver slaves, three motor connected to motor drivers and one Arduino based custom slave (EasyCAT) for additional sensor integration. Currently, custom slave has two limit switch sensors, and emergency-stop button added to EtherCAT bus. Seven more digital I/O and six analog I/O can be added to the bus by using the custom slave. Slaves are connected to the master in daisy-chain topology. Since implementation of EtherCAT master does not require any special hardware, any PC with network interface card can be EtherCAT master. Our master is Generic Linux computer and all necessary specification regarding master hardware is shown in Table I.

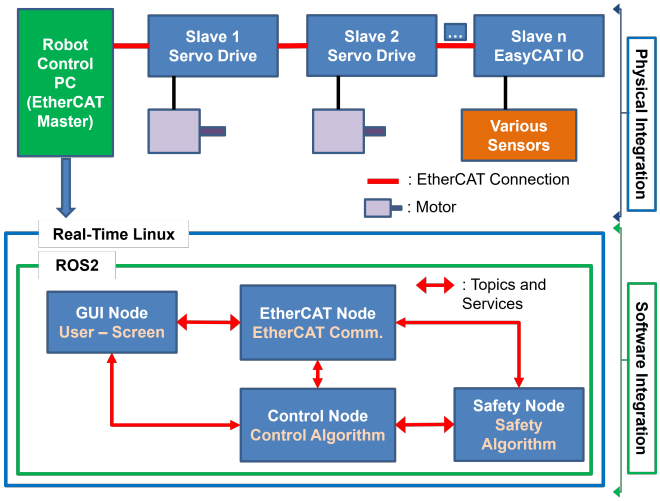

Fig. 1. System Integration Diagram

Software integration part consists of real-time patched (RT_PREEMPT) Linux distribution Xubuntu 18.04, ROS2 and several nodes in ROS2 depicted in Fig 1. Linux itself is not real-time, however there are several methods to bring real-time features to Linux such as dual-kernel approach with Xenomai and RTAI, or single-kernel approach with RT_PREEMPT maintained by Linux Foundation.Even though dual kernel implementations has better real-time performance [27], considering implementation time, portability, scalability, adaptation to programming environment of each patch, and community support, RT_PREEMPT is better real-time solution for our application.

Control software is consists of four components (ROS2 nodes) as shown in Fig 1. Currently we implemented three nodes, and planning to implement one more node as safety node in which system related safety information will be checked every cycle.

- EtherCAT node : Responsible for EtherCAT communication between master and slaves and publishes acquired feedback from slaves under /slave_feedback topic name in 1 kHz frequency. Additionally, subscribes to /master_commands topic published from control node and sends control commands to slaves via EtherCAT communication.
- Control node : Kinematic calculations will be done in this node.This node subscribes /slave_feedback topic published from EtherCAT node and publishes control commands under /master_commands topic.
- GUI node : Consist of camera viewer and slave feedback visualizers such as motor state, communication state, emergency button state. Publishes under /gui_data consists of soft emergency button events, subscribes to /master_commands and /slave_feedback topics to give visual feedback to user.

TABLE I
SOFTWARE & HARDWARE IMPLEMENTATION DETAILS

Master (Generic Linux Computer)					
CPU		**RAM**	**Network Card**		
Intel Core i7-3520M CPU @ 2.90GHz		8GB	Intel 82579V Gigabit Network (driver: e1000e)		
OS	**Kernel + RT_Patch**	**ROS2 Version**	**Control Frequency**	**Task Priority**	**Ethercat Version**
Xubuntu 18.04	4.19.182-rt74	Dashing Didemata	1 kHz	98	v1.5.2+ patch
Slaves					
Servo Drives			**Custom Slave**		
Maxon Epos4 50/5 Compact -3 pcs.-			EasyCAT Custom Slave - 1 pcs.-		
PDO Mappings Servo Drives			**PDO Mappings Custom Slave**		
28 bytes each, TxPDO 14 bytes RxPDO 14 bytes			8 bytes, TxPDO 4 bytes RxPDO 4 bytes		
MODE			Cyclic Synchronous Position Mode		
DC PSU			40V		
Controller					
Microsoft Xbox Controller USB Communication					

III. EXPERIMENTAL RESULTS

We measured performance of our implementation by measuring jitter, periodicity and execution time of our real-time loop. In our EtherCAT node, we implemented real-time communication loop, which runs with 1 kHz frequency, and publishes acquired feedback data from slaves in same frequency. Experiment took three hours and timing samples acquired every 10ms which created 1.8 million timing sample. It was discovered that the performance changes significantly with the Quality of Service (QoS) settings of the ROS2 DDS.With default QoS settings (”reliability : reliable”,”history : keep last”, ”queue size:10”, ”durability : volatile ”, ”liveliness : system default”), it was not performing sufficiently for real-time implementation (up to 8ms jitter) because default reliablity settings may retry multiple times to guarentee that messages are delivered.By changing reliability settings to best effort which will attempt to deliver messages, but may lose some messages if the network

is not robust, and decreasing history queue size to one, made significant difference. With new QoS settings, acquired measurements for our implementation shown in Table II and jitter measurements depicted in distribution box plot shown in Fig 2. According to the table, maximum jitter of our implementation is 107.339 µs and maximum execution time is 772.578 µs.Timing measurements showed that, our ROS2 and EtherCAT based integration method is suitable for real-time applications.

TABLE II
TIMING MEASUREMENTS TABLE

T_{period} (µs)	Avg. ± St.D	999.999±9.733
	Min / Max	892.661 / 1101.423
T_{jitter} (µs)	Avg. ± St.D	4.604±8.575
	Min / Max	0 / 107.339
T_{exec} (µs)	Avg. ± St.D	68.593±15.892
	Min / Max	26.628 / 772.578

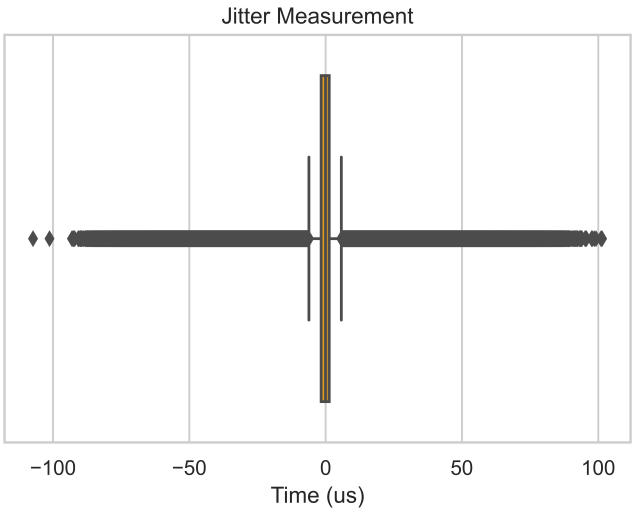

Fig. 2. Jitter Measurements

IV. CONCLUSION

In this paper, we proposed ROS2 and EtherCAT based physical and software integration method for medical robotics to deal with increasing complexity in the medical robotics domain.To the best of our knowledge, this integration method with mentioned framework and fieldbus has not been proposed before. Although our application domain is medical robotics, this integration method can be used in any robotic system. Using ROS2 and EtherCAT in the system closes the gap between research and industry, because both integration methods have been developed to solve problems in industrial fields, and the performance of these integration methods are sufficient for robotic applications. We measured real-time performance of our EtherCAT master implementation and, showed that it is suitable for real-time applications.

As a future work, ROS2 implementation of the EtherCAT master needs to be optimized (publisher memory allocations and QoS settings) for more reliable inter-node communication. Also, to expand current development into a general software framework for medical robot and device development, while keeping ROS2 as an underlying middleware, we will need to establish standard processes for software development cycle, risk analysis, safety feature design based on the result of risk analysis, and verification and validation.

REFERENCES

[1] A. Bihlmaier, T. Beyl, P. Nicolai, M. Kunze, J. Mintenbeck, L. Schreiter, T. Brennecke, J. Hutzl, J. Raczkowsky, and H. Wörn, *ROS-Based Cognitive Surgical Robotics*. Cham: Springer International Publishing, 2016, pp. 317–342. [Online]. Available: https://doi.org/10.1007/978-3-319-26054-9_12

[2] "Saras project: Multirobots-surgery platform conceptual ROS architecture," Apr 2020, (Accessed on 08/13/2021). [Online]. Available: https://saras-project.eu/?p=1442

[3] J. L. Fernández-Pérez, A. C. Domínguez-Brito, D. Hernández-Sosa, and J. Cabrera-Gámez, "Programming by integration in robotics," in *Computer Aided Systems Theory – EUROCAST 2005*, R. Moreno Díaz, F. Pichler, and A. Quesada Arencibia, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 590–595.

[4] D. Brugali and P. Scandurra, "Component-based robotic engineering (part i) [tutorial]," *IEEE Robotics Automation Magazine*, vol. 16, no. 4, pp. 84–96, 2009.

[5] M. J. H. Lum, D. C. W. Friedman, G. Sankaranarayanan, H. King, K. Fodero, R. Leuschke, B. Hannaford, J. Rosen, and M. N. Sinanan, "The RAVEN: Design and Validation of a Telesurgery System," *The International Journal of Robotics Research*, vol. 28, no. 9, pp. 1183–1197, 2009. [Online]. Available: https://doi.org/10.1177/0278364909101795

[6] Z. Chen, A. Deguet, R. H. Taylor, and P. Kazanzides, "Software architecture of the da vinci research kit," in *2017 First IEEE International Conference on Robotic Computing (IRC)*, 2017, pp. 180–187.

[7] M. Quigley, B. Gerkey, K. Conley, J. F. andTully Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[8] H. Bruyninckx, "Open robot control software: the orocos project," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 3, 2001, pp. 2523–2528 vol.3.

[9] A. Deguet, R. Kumar, R. Taylor, and P. Kazanzides, "The cisst libraries for computer assisted intervention systems," 07 2008.

[10] D. Thomas, W. Woodall, and E. Fernandez, "Next-generation ROS: Building on DDS," in *ROSCon Chicago 2014*. Mountain View, CA: Open Robotics, sep 2014. [Online]. Available: https://vimeo.com/106992622

[11] I. Crnkovic, "Component-based software engineering — new challenges in software development," *Software Focus*, vol. 2, no. 4, pp. 127–133, 2001. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/swf.45

[12] "ISO - IEC 60601-1-11:201 - medical electrical equipment — part 1-11: General requirements for basic safety and essential performance."

[13] "ISO - IEC 62304:2006 - Medical device software — Software life cycle processes," https://www.iso.org/standard/38421.html, (Accessed on 08/13/2021).

[14] "IEC 80601-2-77:2019 - medical electrical equipment — part 2-77: Particular requirements for the basic safety and essential performance of robotically assisted surgical equipment."

[15] "EtherCAT Technology Group — EtherCAT," https://www.ethercat.org/en/technology.html, (Accessed on 08/13/2021).

[16] "EtherCAT Technology Group — Organisation," https://www.ethercat.org/en/tech_group.html, (Accessed on 08/13/2021).

[17] "Open EtherCAT Society: Home of SOEM and SOES," https://openethercatsociety.github.io/, (Accessed on 08/13/2021).

[18] "IgH EtherCAT Master for Linux," https://www.etherlab.org/en/ethercat/index.php, (Accessed on 08/13/2021).
[19] R. Zurawski, "Industrial Communication Technology Handbook, Second Edition," 2014, pp. 622–624.
[20] F. Pose, "IgH EtherCAT Master Documentation," https://www.etherlab.org/download/ethercat/ethercat-1.5.2.pdf, October 2017, (Accessed on 08/15/2021).
[21] "EtherCAT® and Arduino," https://www.bausano.net/en/hardware/ethercat-e-arduino/easycat.html, (Accessed on 08/17/2021).
[22] D. Renjewski, A. Peekema, and J. Hurst, "Open-source real-time robot operation and control system for highly dynamic, modular machines," vol. 7, 08 2013.
[23] R. Delgado, B.-J. You, and B. W. Choi, "Real-time control architecture based on Xenomai using ROS packages for a service robot," *Journal of Systems and Software*, vol. 151, pp. 8–19, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219300160
[24] OSRF, "Distributions — ROS 2 Documentation: Foxy documentation," https://docs.ros.org/en/foxy/Releases.html, (Accessed on 08/16/2021).
[25] W. Woodall, "ROS on DDS," https://design.ros2.org/articles/ros_on_dds.html, (Accessed on 08/16/2021).
[26] "Real-time programming in ROS 2 — ROS 2 Documentation: Foxy documentation," https://docs.ros.org/en/foxy/Tutorials/Real-Time-Programming.html, (Accessed on 08/17/2021).
[27] F. Reghenzani, G. Massari, and W. Fornaciari, "The real-time linux kernel: A survey on preempt_rt," *ACM Computing Surveys*, vol. 52, pp. 1–36, 02 2019.