CANopen Robot Controller (CORC): An open software stack for human robot interaction development

Justin Fong*, Emek Barış Küçüktabak$^{\$\dagger}$, Vincent Crocher*, Ying Tan*, Kevin M. Lynch$^{\$}$, Jose L. Pons$^{\$\dagger}$, Denny Oetomo*

Abstract— **Interest in the investigation of novel software and control algorithms for wearable robotics is growing. However, entry into this field requires a significant investment in a testing platform. This work introduces CANopen Robot Controller (CORC) — an open source software stack designed to accelerate the development of robot software and control algorithms — justifying its choice of platform, describing its overall structure, and demonstrating its viability on two distinct platforms.**

I. INTRODUCTION

The rapid progression of actuation, power and processing technologies has significantly improved the capabilities and possible applications of exoskeleton and wearable robotic devices. With these developments, interest in developing controllers and control algorithms for gait trajectory planning and human-robot interactions strategies has risen, with this field being recently identified as a key developmental opportunity for improving exoskeleton performance [1]. However, a key challenge in the pursuit of this field is the development of tools for testing said algorithms. This is particularly true for wearable devices, which see large variations in actuator and sensor configurations and designs.

The present work introduces the CANopen Robot Controller (CORC) — an open source software stack designed to accelerate the development of robotic devices based on CANopen hardware. CORC allows modular hardware use by leveraging the CANopen standardisation while providing a real-time system meeting the safety requirements of wearable devices. Within this work, the design and architecture of CORC is introduced, and the viability of the the stack is demonstrated through measurement of control loop period jitter on two CORC implementations running respectively on the Fourier Intelligence X2 exoskeleton [2] and the EMU upper-limb manipulandum [3].

II. MATERIALS AND METHODS

A. Goals and Objectives

CORC was developed as a common development platform for robots which primarily use CANOpen devices. CANOpen has a 25+ year history as a communications protocol in industrial automation, and is commonly used in robotic devices. Therefore, the objectives of CORC are to provide a flexible, modular architecture for different applications and devices, with a loop rate of at least 200Hz with low jitter.

*University of Melbourne and Fourier Intelligence Joint Laboratory, the University of Melbourne. $^{\$}$McCormick School of Engineering, Northwestern University †Legs + Walking Lab, Shirley Ryan Ability Lab
Corresponding author: `fong.j@unimelb.edu.au`

B. Platform

Two major decision points were made for the platform of the CORC software stack — the operating system and the programming language.

1) Operating System - Linux-based: Linux-based operating systems presented themselves as the obvious choice, primarily due to their open and stable nature, as well as their ability to be run with a real time kernel — which can be critical in human-robot interactions. Furthermore, Linux is well-supported on the larger range of hardware platforms.

2) Programming Language - C/C++: For efficiency reasons, C is used for the critical sections of the software and C++ at the higher level to take advantage of the Object Oriented Programming design without sacrificing performance.

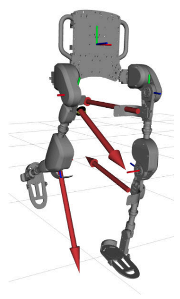
C. Software Architecture

CORC software stack is divided into three distinct layers to enable flexible implementations on different platforms.

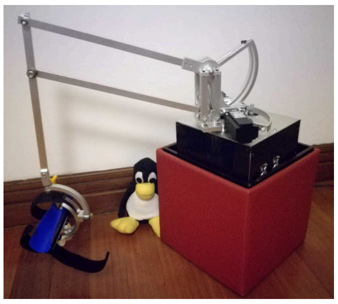
1) CANopen Layer: The CANopen layer handles the application layer transfer of CAN messages, including poll-response (Service Data Object, SDO) and streamed (Process Data Object, PDO) protocols. This layer is based on CANopenSocket [4], and licensed under the Apache License, Version 2.0. Limited changes are required to this layer to develop using CORC.

2) Robot Layer: The robot layer describes the robotic structure, and links it with the input (*e.g.* sensors) and output devices (*e.g.* motor drives). It acts as an abstraction between the control algorithms and the hardware, allowing implementation of kinematic and/or dynamic models. CORC also contains base classes according to CAN in Automation (CiA) standards. This layer is designed to be modified only when new hardware is added to the robotic device.

3) Application Layer: The application layer implements overall program logic and control algorithms. In CORC, each application is a dedicated state machine, derived from a common class. This architecture encourages safe execution while leaving complete freedom for logic implementation. Specific ad hoc libraries can be used at this level to extend capabilities such as providing Robot Operating System (ROS) node capabilities to the application. Given an existing robot implementation, only the application layer modification is required to perform research into novel control strategies. With this approach, a CORC application can also be used as a robot firmware for a specific application with routine execution, or for communication with a third-party software.



(a) RViz Visualisation of Fourier Intelligence X2 Exoskeleton showing forces at force sensors



(b) EMU Upper Limb Manipulandum

Fig. 1: Hardware platforms tested

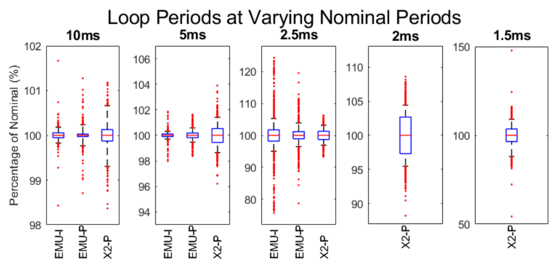


Fig. 2: Loop Periods (normalised to nominal). The extents of the box capture exactly 80% of datapoints, and the extents of the whiskers capture exactly 99%.

D. *Viability test*

The viability test sought to demonstrate two key goals of the toolbox: flexibility for different hardware platforms, and consistency in control loop execution rate. Therefore, simple programs were implemented on two hardware platforms (Fig. 1), and executed at different control frequencies. The hardware platforms were:

1) A Fourier Intelligence X2 Exoskeleton including 4 Copley Accelnet ACK-055-06 motor drives and 4 custom force sensors, driven by a laptop computer (Intel Core i7-9750H CPU, 16.0GB RAM, Ubuntu 18.04, ROS Melodic) with a PCAN-USB adapter [5].
2) The EMU Upper Limb Manipulandum including 3 Kinco FD123-CA motor drives and driven by a Beaglebone AI singleboard computer (Dual Arm® Cortex®-A15, 1.0GB RAM, Debian, Linux Kernel v4.14 with a PREEMPT-RT patch).

Three tests cases were run: *X2-P:* The X2 moving continually between sitting and standing with calculation of joint angles through inverse kinematics based on a hip position trajectory, and with the application set up as a ROS node broadcasting joint state and force sensor readings to be visualised using Rviz (Fig. 1a); *EMU-I:* the EMU operated in impedance control with a feedforward position-dependent gravity compensation term; and *EMU-P:* in position control following a minimum jerk task space trajectory.

Each test case was run with different nominal control loop periods for a minimum of 60 seconds, and the actual loop period recorded whilst the control was operational — with 'system configuration' periods removed. Loop periods were normalised against nominal for comparison.

III. RESULTS

Fig. 2 illustrates the control loop times. Note that it was not possible to run the EMU platform with a period of less than 2.5ms. However, all cases produced a mean loop period within 0.001% of the nominal period.

It is to note that no changes were required at the CANopen layer between the applications on the two platforms, and only at the application layer for the two EMU cases.

IV. DISCUSSION

The results demonstrate the capability of the CORC software stack to be used on two entirely different platforms which share no common hardware. As such, CORC achieved its primary goal of portability and flexibility.

Increasing variation was observed with decreasing nominal period. It is noted that the effect of this variation is extremely dependent on the sensitivity of the implemented control, and as such, these results are presented only as an indication of performance. Interestingly, at lower frequencies and despite having less processing power, the EMU platform demonstrated less variation — likely due to the real time kernel running on the Beaglebone AI.

CORC's primary limitation is that it is designed primarily to run robots which have CAN devices. Whilst the choice of CANopen provides a number of advantages, its serial nature results in bandwidth limitations. Thus, as additional CAN devices are added to the bus, less bandwidth is available for each, reducing the possible motor update speeds.

V. CONCLUSIONS

The present work introduces the CORC toolbox — a software stack designed for robotic devices — to accelerate the development of control algorithms and trajectories. This first evaluation demonstrates its implementation flexibility and performance stability across different hardware platforms. This open software stack is available[1] to the wearable robotics community for use and development. Future work includes the development of a robust logging module and further integration with ROS, beyond the currently-implemented broadcast functionality.

REFERENCES

[1] Sawicki, GS., et al. "The exoskeleton expansion: improving walking and running economy." Journal of NeuroEngineering and Rehabilitation 17.1 (2020): 1-9.
[2] Fourier Intelligence, X2 http://www.fftai-global.com/lower-extremity/
[3] Fong, J, et al. "EMU: A transparent 3D robotic manipulandum for upper-limb rehabilitation." 2017 IEEE International Conference on Rehabilitation Robotics (ICORR). pp. 771-776
[4] CANopenSocket https://github.com/CANopenNode/CANopenSocket
[5] PEAK-System, PCAN-USB https://www.peak-system.com

[1]https://github.com/UniMelbHumanRoboticsLab/CANOpenRobotController