The Journal of Open Source Software

The o80 C++ templated toolbox: Designing customized Python APIs for synchronizing realtime processes

Vincent Berenz*[1], Maximilien Naveau[1], Felix Widmaier[1], Manuel Wüthrich[1], Jean-Claude Passy[1], Simon Guist[1], and Dieter Büchler[1]

1 Max Planck Institute for Intelligent Systems, Tübingen, Germany

DOI: 10.21105/joss.02752

Software

- Review
- Repository
- Archive

Editor: George K. Thiruvathukal

Reviewers:

- @traversaro
- @vissarion

Submitted: 07 July 2020
Published: 02 June 2021

Statement of need

o80 (pronounced "oh-eighty") is a software for synchronizing and organizing message exchange between (realtime) processes via simple customized Python APIs. Its target domain is robotics and machine learning. Our motivation for developing o80 is to ease the setup of robotics experiments (i.e. integration of various hardware and software) by machine learning scientists. Such setup typically requires time and technical effort, especially when realtime processes are involved. Ideally, scientists should have access to a simple Python API that hides the lower level communication details and simply allows to send actions and receive observations. o80 is a tool box for creating such API.

o80 is in some aspects similar to ROS's actionlib (Carroll et al., 2009), as both allow a process to create and monitor execution of commands executed by another process, with python and C++ interoperability. A particularity of o80 is its support for queues of command and the possibilities to request the server to automatically perform linear interpolation between them. o80 also introduces new synchronization methods (see "bursting mode" in the next section). Contrary to actionlib, o80 does not support network communication as it expects the processes it orchestrates to run on the same computer.

Overview

For implementing synchronization, o80 organizes two types of processes:

- A server process encapsulates an instance of o80 back-end and an instance of a driver. At each iteration, the back-end instance computes for each actuator the desired state to be transmitted to the hardware via the driver. The back-end also reads sensory information from the driver. Typically, the server process is programmed in C++ with consideration for realtime.
- A client process encapsulates instances of o80 front-end, which provides: 1) an interface to send commands to the back-end (i.e. requests to compute desired states), 2) methods for querying sensory information, and 3) methods for synchronizing the client with the server process.

In the background, back-end and front-end(s) communicate by exchanging serialized object instances via a interprocess shared memory. Serialization is based on the cereal library (W. & Voorhies, 2017), and the shared memory is based on the Boost interprocess library (Schling, 2011).

*corresponding author

o80 is templated over actuator states and driver; and may therefore support a wide range of systems.

o80's core API supports:

- methods for specifying via commands either full desired state trajectories or partial trajectories relying on interpolation.
- interpolation methods based on specified duration, speed, or number of server iterations.
- methods for either queuing or interrupting trajectories.
- frontend methods for setting commands that are either blocking or non blocking.
- frontend methods for retrieving sensory data that request the latest available information, sensor information corresponding to a specific past server iteration, or the full history of sensory information.
- client processes and the server processes that run asynchronously or synchronously.

Synchronization methods may be based on a fixed desired frequency set for the server, or may be set up by the client process ("bursting mode").

o80 library may be considered complex as it is versatile and can be used in the context of a wide range of control strategies. Yet, the objective of o80 is to provide robot users with a simple Python API. For this purpose, o80 provides convenience functions for generating application tailored Python bindings. The expected usage is that an expert developer uses o80 to design the Python API that will hide to end users the complexities related to interprocess communication and synchronization. Scientists may then use this simplified API to design new robotic experiments. In this sense, o80 aims to be a toolbox for creating customized Python APIs. Generation of Python bindings via the o80 API is based on pybind11 (Wenzel et al., 2016).

Modular Implementation

o80 is based on open source packages that are maintained by the Max Planck Institute for Intelligent Systems, and which may be reused in other contexts. These packages are available on the GitHub Organizations "intelligent-soft-robots," "machines-in-motion," "mpi-is" and "open dynamic robot initiative." Examples of such packages are:

- synchronizer: a library for synchronizing processes
- shared memory: a wrapper over the boost interprocess library that makes exchange of serialized data over an interprocess shared memory trivial
- time series: a templated circular buffer with time stamps, supporting multiprocess access and synchronization

The complete list, the sources, the binaries as well as the documentation of theses packages can be found online (Naveau et al., 2020).

Examples of usage

Integration with SL

An instance of o80 backend has been integrated into the SL realtime library (Schaal, 2009) used for the control of the Apollo manipulator (Kappler et al., 2018). This allows scientists to program robot behavior using a simple Python interface running at a low non-realtime

frequency that synchronizes with the realtime higher frequency C++ control loop of the robot. This interface was used for example for the experiments described in (Baumann et al., 2020).

HYSR training

o80 has been used in the context of reinforcement learning applied to real robotic hardware. (Büchler et al., 2020) describes a setup in which a robot arm driven by pneumatic artificial muscles learns autonomously to play table tennis using an hybrid sim and real training approach (HYSR), i.e, performing real hardware motions to interact with simulated balls. o80 was used in this context to:

- provide a Python API that has been integrated into a Gym environment to provide an interface to reinforcement algorithms (Brockman et al., 2016)
- setting up the synchronization between the real robot control and the MuJoCo simulator (Todorov et al., 2012) used for HYSR
- setting up asynchronous processes for live plotting of the robot state

The code and documentation of this project is available open source online (Berenz et al., 2020).

References

Baumann, D., Solowjow, F., Johansson, K. H., & Trimpe, S. (2020). *Identifying causal structure in dynamical systems*. http://arxiv.org/abs/2006.03906

Berenz, V., Guist, S., & Büchler, D. (2020). http://people.tuebingen.mpg.de/mpi-is-software/pam/docs/pam_documentation/index.html

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI gym*. http://arxiv.org/abs/1606.01540

Büchler, D., Guist, S., Calandra, R., Vincent, B., Schölkopf, B., & Peters, J. (2020). *Learning to play table tennis from scratch using muscular robots*. http://arxiv.org/abs/2006.05935

Carroll, M., Perron, J., Marder-Eppstein, E., Pradeep, V., & Arguedas, M. (2009). http://wiki.ros.org/actionlib

Kappler, D., Meier, F., Issac, J., Mainprice, J., Garcia Cifuentes, C., Wüthrich, M., Berenz, V., Schaal, S., Ratliff, N., & Bohg, J. (2018). Real-time perception meets reactive motion generation. *IEEE Robotics and Automation Letters*, *3*(3), 1864–1871. https://doi.org/10.1109/LRA.2018.2795645

Naveau, M., Berenz, V., Widemaier, F., Viereck, J., & Wüthrich, M. (2020). http://people.tuebingen.mpg.de/mpi-is-software/corerobotics/docs

Schaal, S. (2009). *The SL simulation and real-time control software package*. http://www-clmc.usc.edu/publications/S/schaal-TRSL.pdf

Schling, B. (2011). *The boost c++ libraries*. XML Press. ISBN: 0982219199

Todorov, E., Erez, T., & Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. *IROS*, 5026–5033. https://doi.org/10.1109/iros.2012.6386109

W., S. G., & Voorhies, R. (2017). *Cereal - a c++11 library for serialization*. http://uscilab.github.io/cereal/

Wenzel, J., Rhinelander, J., & Moldovan, D. (2016). *pybind11 — seamless operability between c++11 and python*. https://github.com/pybind/pybind11