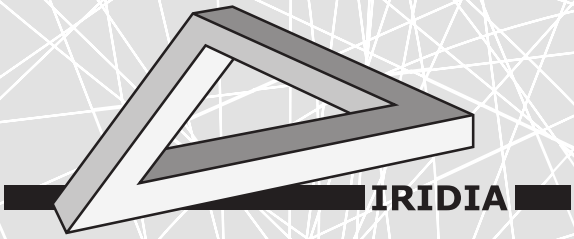

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Supervisor: Software for starting, monitoring, and stopping experiments with swarms of robots

M. Allwright

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2022-004

March 2022
Last revision: October 2022

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2022-004

Revision history:

TR/IRIDIA/2022-004.001	March 2022
TR/IRIDIA/2022-004.002	October 2022
TR/IRIDIA/2022-004.003	October 2022

Table of contents

Introduction

Supervisor is a program for starting, monitoring, and shutting down multi-robot experiments. It is currently compatible with the firmware for the BuilderBot,[1] Pi-Puck,[2] and the IRIDIA drone.[3]

The program provides a web-based GUI in which the user can see which robots are online and upload ARGoS[4] configuration and control software to them for an experiment. During an experiment, the program can record data from the Optitrack tracking system, log messages sent between robots, and capture ARGoS's standard output and standard error from each robot.

This software has been written in Rust[5]. The back-end is built on top of the Tokio asynchronous framework/runtime[6] and the front-end (compiled to WebAssembly for the browser) is built on top of the Yew web framework.[7] The front-end and back-end communicate with each other over a WebSocket.

[1] BuilderBot firmware: https://github.com/iridia-ulb/meta-builderbot
[2] Pi-Puck firmware: https://github.com/iridia-ulb/meta-pipuck
[3] Drone firmware: https://github.com/iridia-ulb/meta-drone
[4] The ARGoS simulator: https://argos-sim.info/
[5] The Rust programming language: https://www.rust-lang.org/
[6] Tokio: https://tokio.rs/
[7] Yew: https://yew.rs/

Installation

The supervisor software has been mostly tested under Linux, however, it should also work under MacOS and Windows.

To compile and use the supervisor, it is necessary to first install the Rust toolchain. To install Rust, follow the instructions on the installation page of the Rust website: https://www.rust-lang.org/tools/install

It is also necessary to install `wasm-pack`, a tool that simplifies the process of compiling Rust to WebAssembly. To install `wasm-pack`, follow these instructions: https://rustwasm.github.io/wasm-pack/installer/

Once Rust and `wasm-pack` has been properly installed, clone this repository and run the command `cargo build` in that directory to compile everything. The compiled binary will be located under `target/debug`.

Configuration

When running the supervisor software, you need to pass a single argument – the path to an XML configuration file. The supervisor software can be built and ran with a single command as follows:

```
cargo run -- --configuration path/to/configuration.xml
```

An example configuration file for the supervisor is shown below:

```
<?xml version="1.0" ?>
<configuration>
  <supervisor>
    <router socket="0.0.0.0:4950" />
    <webui socket="127.0.0.1:3030" />
    <optitrack version="2.9.0"
               bind_port="1511"
               multicast_addr="239.255.42.99" />
  </supervisor>
  <robots network="192.168.1.0/24">
    <drone id="drone1"
           xbee_macaddr="00:04:F3:19:FE:53"
           upcore_macaddr="B0:F1:EC:E9:2F:97"
           optitrack_id="1" />
    <pipuck id="pipuck1"
            rpi_macaddr="B8:27:EB:EF:E1:01"
            optitrack_id="2"
            apriltag_id="10" />
    <builderbot id="builderbot1"
                duovero_macaddr="00:19:88:52:98:0B"
                optitrack_id="3"
                apriltag_id="20" />
  </robots>
</configuration>
```

The `supervisor` node contains global configuration options for the session. * The `router` node specifies the IP address and port on which to run the message router service. The `simple_radios` actuators for the BuilderBot, Pi-Puck, and IRIDIA drone, send TCP messages to this service, which broadcasts those messages to all other connected robots, i.e., robot

controllers configured with the `simple_radios` sensor. The IP address 0.0.0.0 means that robots can connect to this service from any interface (assuming there are no firewall rules preventing this). * The `webui` node specifies the IP address and port on which the web-based user interface can be accessed. The supervisor runs a HTTP-server on this port that will display the user interface when connected to from a browser (e.g., Firefox, Edge, or Chrome). The IP address 127.0.0.1 (local host), means that the user interface is only accessible from the same machine that the supervisor is running on. * The `optitrack` node specifies how to reach the optitrack service and the version of the protocol being used. It is recommended to use version 2.9.0, since other versions are known to either not work correctly or at all.

The `robots` node declares the robots that belong to the swarm and the network to which they are connected. **Important:** the supervisor software uses a very primitive approach to detecting robots, namely it tries to connect to the Fernbedienung service[8] on each network address in the network specified by the `network` attribute. In the case of a class C private network such as 192.168.0.0/24, this includes 253 addresses. It is strongly recommended to not use a network with less than 24 network bits or conversely a network with more than 8 host bits. Using a network with more than 8 host bits will cause an excessive number of connections to be made concurrently and will likely exceed the open file limit of the system.

The nodes underneath the `robot` node list the robots that should be connected to. The `id` tag of each robot should be unique and will be passed to ARGoS automatically when running an experiment. Each robot contains one or more `*_macaddr` attributes which specify the MAC address of the wireless device(s) on the robot. These addresses are used to uniquely identify each robot in the swarm. The attribute `optitrack_id` specifies the rigid-body identifier from the Optitrack data stream. If the optitrack system is running during an experiment, position and orientation data for each robot with a valid identifier will be recorded.

[8] Fernbedienung: https://github.com/iridia-ulb/fernbedienung-python

Design

The design of the supervisor back-end is based on the actor pattern[9]. In this pattern, jobs execute concurrently and communicate with each other by passing messages through channels.

An executor (provided by the Tokio framework) is responsible for running these jobs which are either directly executed as tasks or are multiplexed with other jobs and then executed as a task. The executor efficiently detects when its tasks can perform work and schedules them to be executed. This process can be triggered by a message arriving on a channel, a packet arriving from the network, or the completion of writing or reading a file to disk.

`network`

The network component is responsible for detecting robots in the arena and for managing the connections to those robots. The component is capable of connecting to the Xbee device on a drone or to the Fernbedienung service running on the Linux OS of the BuilderBot, Pi-Puck, or IRIDIA drone. Once a connection has been established, an actor for managing the connection is created and passed to the arena component.

`arena`

The arena component is an actor which maintains the main data structure of the supervisor, that is, it handles routing messages between all other components.

`journal`

The journal component is an actor that records events during an experiment to a Python pickle that can be analyzed after an experiment has been finished. The Python script inside `testing/parse_journal.py` provides an example of how to read the data from that pickle and organize it for further processing.

`webui`

The webui component is an actor that implements a HTTP and WebSocket server. This component subscribes to the messages that it needs to keep the web-based user interface up to date and forwards the messages from that interface back to the other components.

[9]The actor model: https://en.wikipedia.org/wiki/Actor_model

router

The router component is an actor which is connected to by ARGoS when it is started on a robot with the `simple_radios` actuator and sensor. This component broadcasts the messages sent by one robot to all other robots in the swarm.

optitrack

The optitrack actor connects to the Optitrack data stream and forwards position and orientation data to the webui component and to the journal component for display and logging respectively.

Acknowledgements

This work was supported by the Program of Concerted Research Actions (ARC) of the Université libre de Bruxelles and by the Office of Naval Research Global (Award N62909-19-1-2024).