Documentation

Filip Nilsson

August 30, 2021

1 Description of current solution

The current solution is a website where you interact with different parts of the backend through a web interafce. The software is diveded into two parts. One which runs on the client and one which runs on the server. This is a description of how the different parts of the software works and how to setup everything is described in the Github readme.

1.1 Client-side

On the client there are two different files which are supposed to be run. The **client_setup.py** is only supposed to be run initially to create the network setup and setup the necessary tools for multimaster to be run. The **client_setup.py** needs to be run with root priviliges and hence can not be run as part of the main client program.

The second part of the client-side is run by the **client.py** file. This program should be started after the **client_setup.py** is run and this is the part of the client-side that runs to recieve commands from the server. The client-side recieves the command via http and runs a python subprocess to run the command on the client and then returns the output. If a process that does not terminate is started in the subprocess the output will retrieve nothing because continuous returns are not currently possible.

1.2 Server-side

The server has multiple parts to make everything work. The essential files are **server/server.py, server/network.py and webserver/app.py**.

The network file needs to be run with root access and is a process that recieves clients that wants to connect to the server and sets up the multimaster configuration.

The **server.py** file is a class where most of the backend is collected with different methods. This class object is initiated by **app.py** and therefore does not need to be started by itself. The app file calls different functions in the server file to send commands to the clients and to run backend commands like retrieving the connected machines ips.

The web interface is created in flask and is a simple website that allows for interaction with the backend. After the flask app is started the commands are sent to the backend through **app.py**. It is simple to add new commands by just adding them to the **app.py** file. The commands that need to be updated live on the website also runs through the **webserver/static/scrips.js** file.

2 Multimaster

The major problem with multimaster is the network setup when the network setup is done everything works as it should. You can send messages over multiple masters and the messages can be recieved on all nodes. This automatic network setup is done as part of the current solution.
When it comes to synchronizing the sent messages that could be done simply by sending ros time-stamped messages. Over a 5GHz wifi network this could be done with about 100Hz. As seen in figure 1.

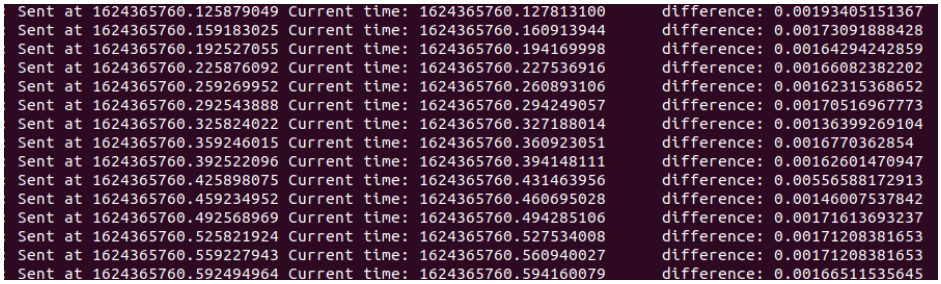

Figure 1: Time stamped ros messages sent over wifi with 100Hz.

3 Future improvement

A list of some important features and fixes that needs to be added for a great improvement of usability.

- Adding support for ros commands to the website could be added from a library like roslibpy. If this was added the ROS page could be updated to include a monitor for multiple different things on the network like bandwidth and a stream of recieved data.
- The automatic network setup need to be changed to include the hostname of the connection for multimaster to work.
- The terminals on the index page needs to be able to start processes through commands. And recieve updates which would mean to change the subprocesses to start but also be able to get the latest update from that specific subprocess.

If these changes were added the future website could potentially be a very good way to control multiple different machines through a simple web interface.

References

[1] Github repository, https://github.com/flippe3/ros_multimaster.

[2] Ros multimaster paper, https://digital.csic.es/bitstream/10261/133333/1/ROS-systems.pdf.

[3] Roslibpy, https://roslibpy.readthedocs.io/en/latest/.