Rook

Robotics Orchestration Kit

Getting Started

Project Goals

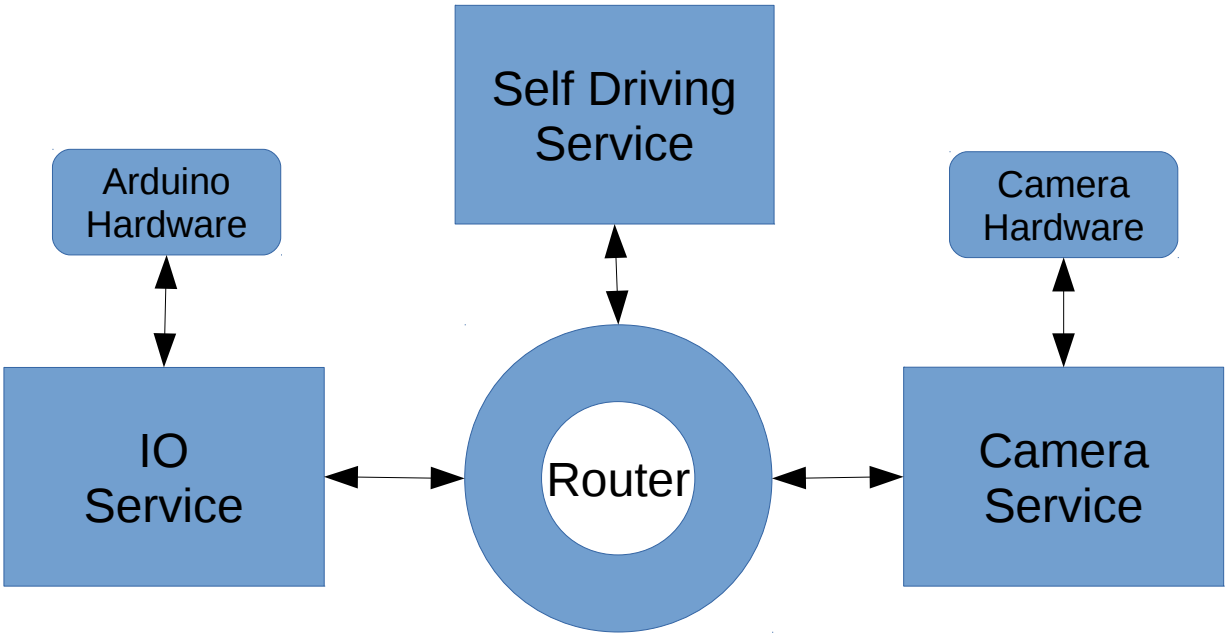
- Provide a feature-rich, flexible robotics platform that is easy to use: Rook's main goal is to provide a strong framework that can be utilized by professionals and amateurs alike.
- Performance and Consistency: Rook strives to provide high-quality, high-performance code. While written in Java, latency-sensitive applications will be happy to know that Rook's code is largely garbage-free which will help keep GC-pauses at bay.
- Distributed Computing made easy: As the Internet-of-Things and Advanced Robotics continue to progress, fleets of commodity hardware will provide a wide range of functionality. Therefore, Rook tries to make it easy to distribute computational power and functionality between many devices.

Services and Routers

At its core, Rook provides a standard service-oriented architecture. A *service* provides a set of functionality, while a *router* is responsible for passing *messages* between services within an application. Messages passed between services are represented in a serialized fashion so services can communicate over a network just as naturally as if they were running in the same process. Specialized services are designed to pass messages between multiple routers. This provides a platform where services can run in a distributed manner just as easily as they can run in your IDE.

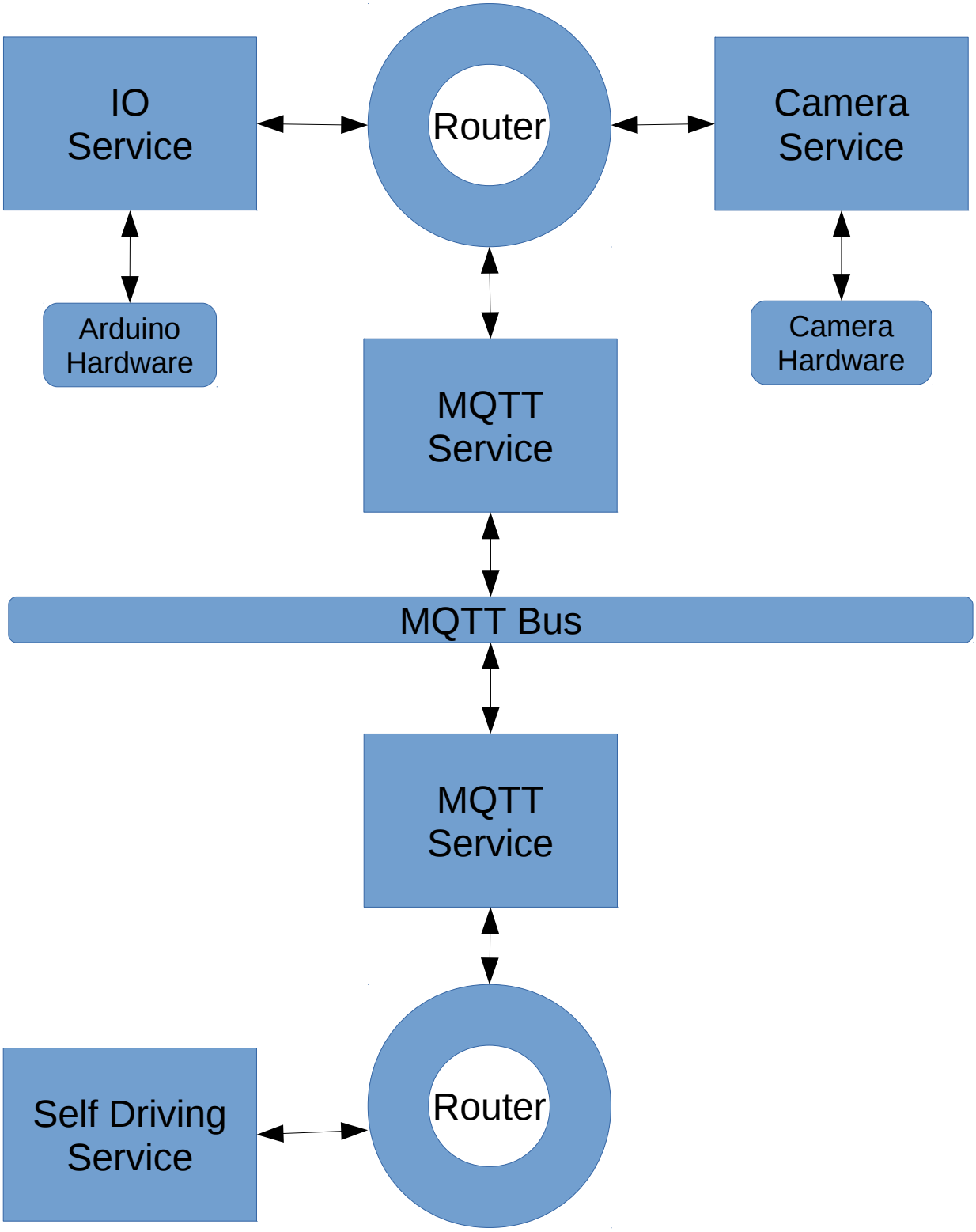
Example Environment

Below is an illustration of an environment where a user-written “Self Driving Service” utilizes an IOService and a Camera Service. The IOService is responsible for communicating with some underlying hardware. In this case, the underlying hardware is an Arduino board. The nature of an IO Service is that it is responsible for abstracting the underlying hardware away from the user so that it can easily be swapped out. In this example, it means that the “Self Driving Service” code wouldn't need to be changed if the Arduino was swapped out with a Raspberry Pi or some other supported hardware.



Distributed Example Environment

Below is an extension of the previous example. In this example, the user has decided that the "Self Driving Service" would be better served with its own hardware. By introducing another machine, and two MQTT services that can communicate via an MQTT bus, the user was able to migrate the "Self Driving Service" to a separate machine without any code changes.



Language Support

In its current state, Rook supports an official Java Library. It has been designed with support for other programming languages in mind, and we aim to officially support more languages in the future. All of our core services use well-documented messaging formats that don't require any special libraries to read. We hope this helps spread Rook across many development platforms and programming languages.

Proxies

Messages passed between services are represented as byte-buffers. So that every developer that uses a service doesn't have to rewrite code to create these byte-buffers, many services have an accompanying proxy library. A proxy is responsible for providing communication with a service through simple method calls. It will encapsulate all messaging logic so the user doesn't have to deal with it.

See Also: Proxy Design Pattern: http://en.wikipedia.org/wiki/Proxy_pattern

Note: To avoid compatibility problems, a proxy library should impose a limited number of extra libraries!

Core Services

The core Rook software platform provides a number of services “out-of-the-box”. These services aim to provide common functionality that many of our users should find useful, while also providing a set of basic functionality for the community to build around in the spirit of collaboration and compatibility. Please see the JavaDocs for more information on how to use these services.

- IOService
- CameraService
- MqttService

Building From Source

Prerequisites

- git
- Java 8
- maven 3.x

Build steps

```
#pull code from GitHub
git pull https://github.com/septapulse/rook.git
cd rook/java

#build using maven
mvn clean install package
cd target/rook*/rook*/
```

Rook Runtime

Rook Environment Structure

cfg
    services
        *.cfg (Service Configurations)
    log4j.properties (Logging Configuration)
    *.list (Service List Configuration)
lib
    *.jar (Java Archive files)
log
    rook.log (Log file)
scripts
    ./start (Script to run on Linux and OSX)

Using the “start” Script

./start <optional:service_list_file_path>

When the service list file path is specified, only the services specified in the given list file will be loaded at startup. If no file is specified, all services in the cfg/services folder will be started.

Service List File Format

The Service list file specifies which cfg files in the cfg/services file to load at startup. Here is an example list file that will load “io-service-serial.cfg” and “my-custom-service.cfg” from cfg/services upon startup:

my-environment.list

```
io-service-serial
my-custom-service
```

To startup the Rook environment with this list file, you would run the followin command:

```
./start cfg/my-environment.list
```