RtBot: a signal processing framework focused on early anomaly detection

Robert Carcassés Quevedo, Eduardo Pérez Verdecia, Yuriel Núñez Jiménez

Abstract

RtBot framework is presented as a general purpose digital signal processing framework focused on anomaly detection.

Keywords: digital signal processing, open source, anomaly detection

1. Introduction

RtBot is a general purpose digital signal processing framework focused on early anomaly detection. It's written in c++ and has been built with performance in mind.

Wrapper libraries exist for python, javascript (using wasm) and rust, and in the future we plan to support java and swift languages as well. In this sense RtBot can be easily adapted in any existing codebase, as it already provides interfaces to the most popular existing computer languages or new ones can be built for those missing languages using their native interface with c++. RtBot programs are very fast and have a low footprint, being able to run even in microcontrollers with very limited resources, such a raspberry pi pico or the esp32.

As a digital processing framework, it contains the common pieces found in such frameworks such as high and low pass filters, finite response filters, signal re-samplers and so on.

As an anomaly detector framework it is designed to give developers the tools to minimize the time difference between the moment when the anomaly is detected and the moment when the anomaly actually happened.

2. Architecture

2.1. Modules

RtBot framework is divided into a set of modules: core, api and std ones. This list is likely to grow in the future and users can also extend it with their own private business model specific ones.

The core module contains the general definitions and it does not depend on any other. You can write a program using only the core module, but you will have to write it in c++ and explicitly construct the operators and their connections.

The api module allows users to easily interact with the core module. For instance it can read RtBot programs saved in json format and construct the correspondent internal pipeline of operators at runtime. In this module is also where the external bindings are defined, giving a simplified api to run programs. If you are developing a web or desktop app, or even a mobile app, you will likely use this module. On the other hand, if you are building an embedded application

for a microcontroller, including this module in your build will cause a large binary output that probably won't fit on the device. In this case you will have to rely only on the core and std modules to make sure you produce binaries small enough.

The std module represents a standard library of commonly used operators that can be used to build programs. Operators defined here are general purpose ones, like the inputs, resampler or finite response operators, which are ubiquitous used in any kinds of input signals. In practice this module and the core one will be the minimal components you will need in order to create some non trivial program.

2.2. Inside the core

An RtBot program expects a signal as input, which is a time-ordered stream of messages to be fed to it, one at the time, in what we call a program iteration. This iteration may or may not produce an output, which will be defined later.

A message is a tuple of timestamp and value. It is important to remark, even if obvious, that the timestamp in the message is the one used for any internal analysis, and not the time that the message arrives. RtBot has no clock inside or anything similar, so it has no way of counting real life time.

RtBot assumes the time inside a message to be an integer. The main motivation for this is that comparing two timestamps is more efficient and meaningful in this case than if we allow the timestamps to be of float type. This allows, for instance, to synchronize signals according to its timestamp in a much simplified and efficient way. This shall not, however, be a hurdle to adopt RtBot to process signals with fractionary timestamps, as a rescaling in the time dimension of the signal prior to sending it to the RtBot program will solve any type mismatch. This time scale, i.e. seconds or milliseconds, depends on the nature of the signal being processed and it is up to the programmer to decide which makes sense for the given type of signal.

Multidimensional signals can be processed by sending their components to specified input operators.

2.3. Operators

The main concept in the RtBot architecture is that of an operator. An operator it's simply a computational unit that takes an input, transforms it and produces an output. Operators take messages as inputs and produce other messages as outputs. Remarkably, operators assume that they will receive messages with increasing timestamps, this is, ordered in time.

Operators can be connected with other operators, in a directed way, such that one operator can have children operators. If an operator has children then whenever it produces an output it will be sent to its children which will consider those messages as its own input, and so on. Operators may change the throughput of the stream they receive, producing any number of messages, or none, per each message it receives. In this sense they are also flow controllers, deciding when and what to forward to its children operators.

2.3.1. Input

An RtBot program is then this set of operators connected in a graph structure, a pipeline, where the external signal entry points are marked by the input operators. An input operator is just an operator that is designed to be a good intake of the external signal. Input operators are not mandatory but recommended entry points, as they perform sanity checks on the input signal before forwarding the message to others. For example, an input operator will discard new

time	value
.	.
2010	2.34
2020	2.53
2020	0.82
2030	5.67
2040	7.04
.	.

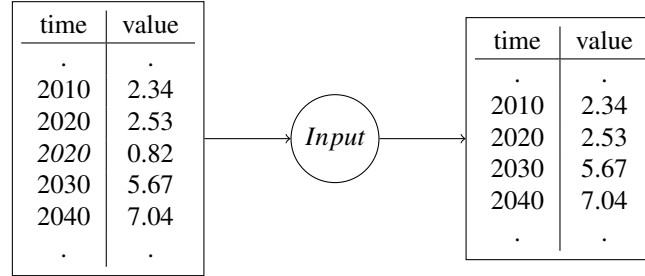| time	value
.	.
2010	2.34
2020	2.53
2030	5.67
2040	7.04
.	.

Figure 1: Action of the *Input* operator in the incoming signal. Notice how the second message with the same timestamp $t = 2020$ is ignored. The same would happen if the message timestamp would have been less than the previous one.

messages with timestamps prior or equal to the last message received, guaranteeing the time order in the messages it forwards.

2.3.2. Resamplers

All operators that implement some math over their input assume that the messages it will receive are not only time ordered but also equidistant in time, or equally spaced in time. We will call this type of signal a regular one, while those that don't fulfill this will be named irregular. This is a strong assumption and it will likely be the case that input signals are irregular. This is why we have a special set of operators called resamplers. A resampler operator will take an irregular input signal and will produce a regular one. Internally, it will use the irregular signal as its source of truth, to find a realistic interpolation over the time grid points. The interpolation algorithm depends on the resampler used. In practice this means that, while not strictly necessary, programs will start with input operators followed by resampler ones.

3. Use cases

3.1. Network traffic monitoring and statistics

3.2. Real time biofeedback

3.3. Trading bots

References

[1]