message_filters::sync::ApproximateTime

New in ROS CTurtle

1. Overview

This is a policy used by message_filters::sync::Synchronizer to match messages coming on a set of topics. Contrary to message_filters::sync::ExactTime, it can match messages even if they have different time stamps. We call **size** of a **set** of messages the difference between the latest and earliest time stamp in the set.

The algorithm is the product of long discussions with Blaise. It does *not* work like ExactTime (/ExactTime) except with matching allowed up to some epsilon time difference. Instead it finds the *best* match. It satisfies these properties:

a. **The algorithm is parameter free**. No need to specify an *epsilon*. Some parameters can be provided (see below), but they are optional.
b. **Messages are used only once.** Two sets cannot share the same message. Some messages can be dropped.
c. **Sets do not cross.** For two sets S and T, their messages satisfy either $S_i <= T_i$ for all i, or $T_i <= S_i$ for all i, where i runs over topics.
d. **Sets are contiguous.** There is at least one topic where there is no dropped message between the two sets. In other words there is no room to form another set with the dropped messages.
e. **Sets are of minimal size** among the sets contiguous to the previous published set.
f. **The output only depends on the time stamps**, not on the arrival time of messages. It does assume that messages arrive in order on each topic, but not even necessarily across topics (though the queue size must be large enough if there are big differences or messages will be dropped). This means that ApproximateTime can be safely used on messages that have suffered arbitrary networking or processing delays.

Optional parameters:

- **Age penalty**: when comparing the size of sets, later intervals are penalized by a factor (1+AgePenalty). The default is 0. A non zero penalty can help output sets earlier, or output more sets, at some cost in quality.
- **Inter message lower bound**: if messages of a particular topic cannot be closer together than a known interval, providing this lower bound will not change the output but will allow the algorithm to conclude earlier that a given set is optimal, reducing delays. With the default value of 0, for messages spaced on average by a duration T, the algorithm can introduce a delay of about T. With good bounds provided a set can often be published as soon as the last message of the set is received. An incorrect bound will result in suboptimal sets being selected. A typical bound is, say, 1/2 the frame rate of a

camera.

- **Max interval duration**: sets of more than this size will not be considered (disabled by default). The effect is similar to throwing away a posteriori output sets that are too large, but it can be a little better.

2. Algorithm

Let S be the last published set, and T the next one being created. It works as follows:

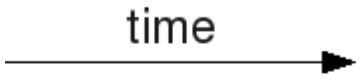
- When set S is published, for each topic, all messages older than the one in the set are discarded.
- Messages are inserted in a topic-specific queue as they arrive.
- Once each topic-specific queue contains at least one message, we find the latest message among the heads of the queues, that's the *pivot*. The pivot belongs to every set contiguous to S, so it must belong to T (property *d*), hence its special status. For each message *m* older than the pivot, call T_m the smallest set that starts at m. It is obtained by taking the earliest message arrived after m on each queue. Since T starts at some message, it must be one of the T_m's.
- Go through messages m in order of time. If the next message cannot yet be determined, wait until the queues fill up enough.
- When m reaches the pivot so that all T_m's have been enumerated, or whenever it can be proved using various bounds that the smallest T_m has been found, publish the smallest T_m.

It works in three nested phases:

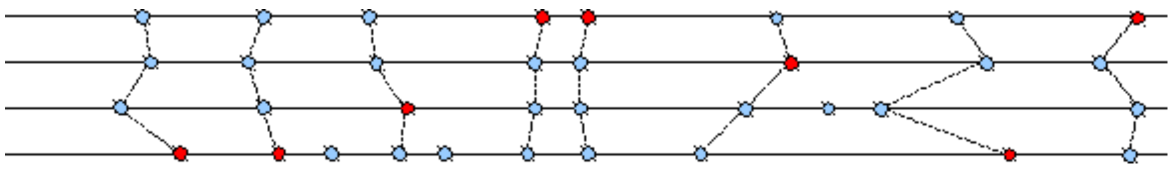
- Find the pivot and a first valid candidate set.
- Advance along the queues until one of the queues is empty, in search of a better candidate.
- Advance past this point assuming that future messages will arrive at the most optimistic time, to try to prove that the current candidate is optimal.

3. Examples

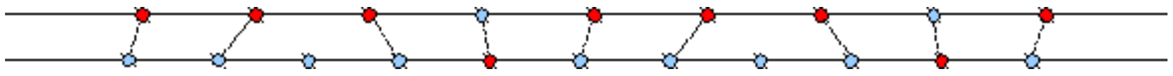
Here are some example results. Time goes from left to right, each line is a topic, and each dot is a message. The red message is the pivot, and the broken line links the messages in a set.



Varying frequency:



Unequal frequency:



Wiki: message_filters/ApproximateTime (2010-07-28 03:10:29由JoshFaust (/JoshFaust)编辑)