How to Use The Passive Time Schronization Package

David Hanley

December 16, 2018

Abstract

This document describes how to use the time synchronization package (1)in a batch process, (2)online, and (3)when a sensor does not have its own clock. Installation instructions are also provided. Finally, a full example using this package is included at the end. Details on the algorithm can be found by reading Edwin Olson's paper (which is cited in the references).

1 Introduction: System Overview

This passive time synchronization package attempts to sychronize sensor measurements to some master clock. The solution is "passive" because the synchronization can be performed without any cooperation from the sensor. Therefore, this approach is different from approaches like Precision Time Protocol [1] and Network Time Protocol [2]. This can be run online or on data offline. The offline approach produces better results (though based on a given application, it may not be possible). The principle of operation here is that major time latencies between the sensor and the master clock vary with time. Therefore, if we look for the smallest latency, we can better synchronize the clock. Note that if there is some fixed latency (i.e. if the master clock recieves sensor measurements at some latency that is always greater than 0.1 seconds), then the best this time synchronization method can do is synchronize sensor measurements to near that fixed latency. Clock drift is also accounted for. The user is expected to provide upper and lower bounds on the clock drift, the smaller those bounds can be, the better the synchronization. Note that a sensor does not need a clock in order to use this synchronization procedure. This method is an implementation of Edwin Olson's time synchronization approach [3]. More details on this method are described in that paper.

2 Install and Run Instructions

Note that this package has been tested successfully on:

1. Ubuntu 16.04

This package requires CMake for installation. In Ubuntu, this can be installed by using the terminal command:

```
sudo apt-get install cmake
```

To install the package, download and (if necessary) unzip the package from github. In the terminal, enter the time_sync directory and run the following commands:

```
chmod +x install_script.sh
./install_script.sh
```

This script installs the package and runs a test module.

3 Synchronizing a Batch of Sensor Measurements

To use the package in any C++ code, the package should be installed and the appropriate header file should be included.

```
#include "passive_time_sync/passive_time_sync.h"
```

Then we need to create a time synchronization object

```
alpha_1 = 0.05;
alpha_2 = 0.05;
f0 = 0.0;
passive_time_sync  test1(alpha_1,alpha_2,f0);
```

Here we assume that the drift in the sensor clock is bounded as

$$(1-\alpha_1)\Delta t - f(0) \le \Delta p \le (1+\alpha_2)\Delta t + f(0) \tag{1}$$

where Δt is the true time between sensor samples and Δp is the sensor clock's estimate of the time between sensor samples. The constant $f(0)$ is the range of clock drift when Δt goes to zero. This is included to ensure that these are conservative bounds on Δp.

Given a C++ vector of sensor time stamps (in datatype double and in seconds) and a vector of master clock time stamps when the sensor measurements are recieved, we can compute a sychronized set of sensor time stamps as

```
vector<double> output;
output = test1.get_batch_sync(sensor_clock,  master_clock);
```

4 Synchronizing Sensor Measurements Online

We don't need to synchronize sensor time stamps in a batch process. We can (and often do) synchronize online. To do this we still need to include the package and create a time synchronization object. To use the package in any C++ code, the package should be installed and the appropriate header file should be included.

```
#include "passive_time_sync/passive_time_sync.h"
```

To create the time synchronization object

```
alpha_1 = 0.05;
alpha_2 = 0.05;
f0 = 0.0;
passive_time_sync  test1(alpha_1,alpha_2,f0);
```

Given a sensor time stamps (in datatype double and in seconds) and a master clock time stamp when the sensor measurement is recieved, we can compute a sychronized set of sensor time stamp as

```
double online_output;
online_output = test1.online_sync(sensor_clock,  master_clock);
```

5 Synchronizing Sensor Measurements without a Sensor Clock

The sensor we use does not have to have a clock or its own time stamps in order for us to perform synchronization. All we really need to know is the sample rate of the sensor. The inconsistency in a sensor's sample rate can be seen as clock drift. Assigning a sensor time stamp to sensor measurements can be done by implementing the following procedure:

Sensor Time Stamp

```
Require: Sensor Sample Rate
  if first sensor measurement then
      Time Stamp = 0
  else
      Time Stamp = Prior Time Stamp + Sensor Sample Rate
  end if
  return Time Stamp
```

These resulting time stamps can then be synchronized to the master clock's time stamps online or as a batch process using the processes described in the prior sections.

6 Example of Use

```
//
// Passive time synchronization attempts to synchronize sensor
// measurements to a master clock using the fact that latency tends to
// vary and sensor clocks tend to drift relative to the master clock.
//
// David Hanley
//
// test_time_sync.cpp
//
// Test time synchronization implementation file for the April Lab's
// passive time synchronization approach. This approach is described in:
// E. Olson. "A Passive Solution to the Sensor Synchronization Problem."
// IEEE/RSJ International Conference on Intelligent Robots and Systems
// (IROS). Taipei, Taiwan, Oct, 2010.
//
// Usage:
//
//

/*-----------------------------------------------------------------------*/
/*------------------------------ Preamble -------------------------------*/
/*-----------------------------------------------------------------------*/
/*----------------- Defines --------------------*/
/*--------------- End Defines ------------------*/

/*------------------ Includes ------------------*/
#include <iostream>
#include <vector>
#include <boost/random/normal_distribution.hpp>
#include <boost/random/mersenne_twister.hpp>
#include <boost/random/variate_generator.hpp>
#include "passive_time_sync/passive_time_sync.h"
/*--------------- End Includes -----------------*/

/*------------------ Globals -------------------*/
/*--------------- End Globals ------------------*/

/*------------------ Classes -------------------*/
```

```
/*------------------------- End Classes -------------------------*/

/*------------------------- Namespaces -------------------------*/
/*----------------------- End Namespaces -----------------------*/

/*-------------------------- Pragmas --------------------------*/
/*------------------------- End Pragmas -------------------------*/

/*-------------------- Function Prototypes --------------------*/
/*------------------ End Function Prototypes ------------------*/
/*-------------------------------------------------------------------------------*/
/*--------------------------------- End Preamble ---------------------------------*/
/*-------------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------------*/
/*---------------------------------- Functions ----------------------------------*/
/*-------------------------------------------------------------------------------*/
int main()
{
    // Initialize variables
    int N = 1000;
    vector<double> master_clock(N);
    vector<double> sensor_clock(N);
    vector<double> true_time(N);
    double A = 10.0;
    double dt = 0.01;
    double average = 0.0;
    double online_average = 0.0;
    double online_output;

    // This is the underlying integer random number generator
    boost::mt19937 igen;
    // The second template parameter is the actual floating point
    // distribution that the user wants
    boost::variate_generator<boost::mt19937, boost::normal_distribution<> >
        gen(igen,boost::normal_distribution<>(0.0,0.001));
    boost::variate_generator<boost::mt19937, boost::normal_distribution<> >
        gen2(igen,boost::normal_distribution<>(0.0,1.0));

    // Iterate through time to generate all time stamps
    for (int i = 0; i < N; i++)
    {
        if (i == 0)
        {
            true_time[i] = 0.0;
        }
        else
        {
            true_time[i] = true_time[i-1] + dt;
        }
        // Add latency
        master_clock[i] = true_time[i] + abs(gen2());
        // Add offset
        sensor_clock[i] = true_time[i] + A;
        // Add clock drift
        A = A + gen();
    }
    // Create a clock synchronization object
    passive_time_sync test1(0.05,0.05,0.0);

    // Batch synchronization
    vector<double> output;
    output = test1.get_batch_sync(sensor_clock, master_clock);

    // perform online synchronization, compute average error, and compute
    // batch error average
    for (int i = 0; i < N; i++)
    {
        online_output = test1.online_sync(sensor_clock[i], master_clock[i]);
        online_average = (true_time[i] - online_output) + online_average;
        average = (true_time[i] - output[i]) + average;
    }
    average = average/((double)true_time.size());
    online_average = online_average/((double)true_time.size());

    // Output Results
    cout << "--------------------------------------------" << endl;
    cout << "␣␣␣␣␣␣␣␣Batch␣Average␣Sync␣Error␣␣␣␣␣␣␣␣" << endl;
    cout << "--------------------------------------------" << endl;
    cout << abs(average) << "␣seconds" << endl;
    cout << "--------------------------------------------" << endl;
    cout << "␣␣␣␣␣␣End␣Batch␣Average␣Sync␣Error␣␣␣␣␣␣" << endl;
    cout << "--------------------------------------------" << endl;
    cout << "--------------------------------------------" << endl;
    cout << "␣␣␣␣␣␣␣␣Online␣Average␣Sync␣Error␣␣␣␣␣␣␣" << endl;
    cout << "--------------------------------------------" << endl;
    cout << abs(online_average) << "␣seconds" << endl;
    cout << "--------------------------------------------" << endl;
    cout << "␣␣␣␣␣␣End␣Online␣Average␣Sync␣Error␣␣␣␣␣" << endl;
    cout << "--------------------------------------------" << endl;

    return 0;
}
/*-----------------------------------------------------------------------------*/
/*------------------------------- End Functions -------------------------------*/
/*-----------------------------------------------------------------------------*/
```

References

[1] “IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems,” *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pp. 1–300, July 2008.

[2] D. L. Mills, “Internet time synchronization: the network time protocol,” *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, Oct 1991.

[3] E. Olson, “A passive solution to the sensor synchronization problem,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2010, pp. 1059–1064.