The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

A Passive Solution to the Sensor Synchronization Problem

Edwin Olson
University of Michigan
Ann Arbor, MI 48105
ebolson@umich.edu
http://april.eecs.umich.edu

***Abstract*— Knowing the time at which sensors acquired data is critical to the proper processing and interpretation of that data, particularly for mobile robots attempting to project sensor data into a consistent coordinate frame. Unfortunately, many popular commercial sensors provide no support for synchronization, rendering conventional synchronization algorithms useless.**

In this paper, we describe a *passive* synchronization algorithm that can significantly reduce timing error versus naively time-stamping sensor data when it arrives at the host. It is passive in the sense that the algorithm requires no special cooperation from the sensor. Our method estimates the timing jitter induced by hosts, and thus does not require a real-time operating system. We rigorously derive and characterize the method, proving that it can only improve upon the synchronization accuracy of the standard approach.

I. Introduction

Many standard commercial sensors provide no special support for synchronizing the data with the robot's computer or with the data provided by other sensors. Poor synchronization directly impacts the ability of the robot to perform sensor fusion: projecting sensor data into a common frame of reference requires precise knowledge of the robot's position at the time when the data was collected. If the robot is moving, synchronization error results in projection error.

The magnitude of this error can be surprising, particularly in the case of rotating robots. Consider, for example, a robot rotating at a modest 90 deg/s that is observing an object 10 m away: a synchronization error of just 10 ms results in a projection error of 15.7 cm. For vehicles moving at high speeds, such as the vehicles of the DARPA Urban Challenge [1], [2], [3], even modest synchronization errors can become serious safety issues.

Unfortunately, most common sensors provide no explicit means for sensor synchronization. Standard synchronization algorithms, such as the familiar Network Time Protocol (NTP) [4], require the cooperation of each node, and thus are not applicable to the sensor synchronization problem.

The problem is compounded by the way in which many robotic systems are constructed. Many devices (including ubiquitous Hokuyo and SICK LIDAR scanners, Xsens IMUs, and others) commonly interface to computers through a USB to serial converter. The deep buffers and buffer-flushing logic[1] of these devices adds variable-length delay (jitter) beyond that of the devices themselves. In the case of more primitive sensors like infrared range finders, ultrasound sensors, and odometry sensors, a data acquisition board is typically involved that introduces additional delays. The heterogeneous nature of these sources complicates a single common solution.

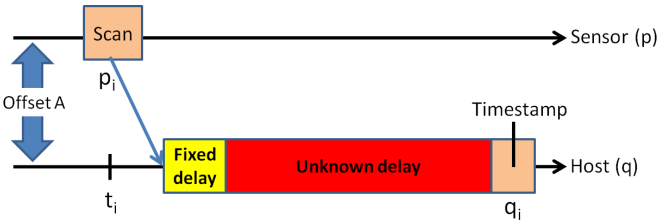

Fig. 1. Problem statement. We wish to estimate the time t_i on a host that some external event p_i occured on a sensor. We observe the time p_i that corresponds to t_i, but is subject to an unknown offset A, and we observe q_i which is subject to a delay of unknown duration. This paper describes a passive sensor synchronization algorithm that provides principled estimates of t_i, without the additional active cooperation of the sensor.

In addition, most robotic systems are built on top of non real-time operating systems that place no upper bound on the length of time that data will sit in a buffer before the application is given a chance to process it. Building a robot system on a non real-time operating system (including standard flavors of Linux and Windows) is extremely convenient to the developer, but comes at the price of considerable timing jitter. This jitter can amount to hundreds of milliseconds on a loaded system.

Despite the challenges, there is room for improving synchronization. First, even very problematic systems will *occasionally* exhibit low-latency for a single sensor message. Second, most sensors have some notion of *sensor time*; they might have their own clock, or produce messages at a predictable rate that effectively serves as the "tick" of a clock. If the low-latency messages can be identified, they can be used to improve the synchronization of other messages. One challenge is to identify which messages are the "low-

[1]The ubiquitous FTDI USB-to-serial converter is well-known for exhibiting high latency. In its default configuration, it artificially waits 16ms before transmitting data in an attempt to minimize large numbers of small transactions. On Linux, at least, this "latency timer" can be reduced to 1 ms via the `proc` file system.

latency" ones; a second is to correctly account for the accuracy (or lack thereof) of the sensors' clocks. The clocks on many sensors are not nearly as accurate as those on computers, which hampers synchronization.

In this paper, we provide an algorithm that passively synchronizes sensor data. We exploit the property that some observations will be low-latency, account for errors between the sensor and host clocks, and exploit causality information (i.e., that the the latency must be positive) in order to estimate the offsets between the host and sensor clocks.

The central contributions of this paper are:

- We describe a passive synchronization algorithm for recovering the local time at which a sensor observation was obtained. We develop two variants: one which operates causally (and is thus suitable for online use), and another which operates anti-causally yielding even better performance.
- We prove that our method produces timing estimates that are at least as good as naive time-stamping (and usually much better).
- We propose a simple clock drift model and use it to derive a principled method for determining which observations were made with low latency.
- We validate our algorithm using synthetic data, characterizing the relationship between the sensor's timing accuracy relative to the host, the observed timing jitter, and the resulting quality of the output produced by our method.

Our algorithm was originally developed for use on MIT's DARPA Urban Challenge vehicle [3], where it was used to synchronize virtually all of the sensors to a single time base, including 12 SICK LIDARs, a Velodyne HDL-64E sensor, 15 Delphi ACC Radars, an Applanix IMU/GPS, and several small embedded microcontrollers tasked with low-level control. With this number and variety of input sources, careful synchronization was critical.

In the following section, we review other time synchronization methods. In Section III, we describe a simplified version of our algorithm that ignores drift between the two clocks. We then describe our clock drift model and re-derive our method in terms of it. We illustrate the performance capabilities of our algorithm using synthetic experiments in Section IV. Finally, we summarize our method and results in Section V.

II. Related Work

Despite the central importance of sensor synchronization, there is little discussion of it in the literature that is applicable to black-box sensors. In our literature search, almost all of the methods we found all involved active collaboration between the participating nodes. This is impossible in many practical robotics scenarios, since the sensors cannot be modified. Still, these other approaches offer relevant mathematical tools and insights that are applicable to our case.

Data from odometry and camera sensors can be synchronized by identifying events which are detectable in both data streams [5]. For example, when the robot begins moving from a stop, the first odometry measurement and camera measurement exhibiting motion can be reasonably associated. Since odometry and camera data are sampled only sparsely in time, some timing uncertainty remains. However, as additional events are recorded, this uncertainty can be decreased by carefully constructing and intersecting the confidence intervals. Unfortunately, if these "canary" events cannot be identified, the method is not applicable.

Most clock synchronization schemes are variants on Cristian's Algorithm [6], which relies on round-trip measurements. Node a sends a packet to node b, which replies back to a with the time on node b. Node a knows that the time reported by b corresponds to some point after it sent the message and before it received the response. The smaller the round-trip time, the tighter the bound.

The Berkeley algorithm [7] employs Cristian's algorithm as a building block, but proposes a scheme for a network of computers to converge more rapidly to a single common time. The well-known Network Time Protocol uses Marzullo's algorithm [8], which involves intersecting the time bounds obtained from multiple offset measurements.

The IEEE 1588 protocol specifies an algorithm for two devices to synchronize their clocks. It is most typically layered on top of 802.3 ethernet or 802.11 wireless ethernet [9] and can provide sub-microsecond synchronization. The hardware requirements of IEEE 1588 are modest enough that the protocol is supported even on low-end microcontrollers like the Stellaris LM3S6965. Of course, the protocol does not help synchronization for devices that do not support the standard.

Coordinating the trajectory of a welding robot along a seam requires tight synchronization of the control and sensing subsystems. In the case of [10], the authors describe a synchronization scheme that exploits the sensor's ability to be triggered on demand. This allows a conventional round-trip time method to be used. As in this case, machine vision grade cameras (as opposed to consumer grade cameras) can often be triggered remotely.

Mesh networks pose unique challenges to time synchronization, due to the undesirability of centralizing computation and the presence of significant bandwidth and power limitations. Many systems employ a flooding-like mechanism. See [11], [12] for representative examples.

In summary, there is a wealth of literature on clock synchronization between hosts that can actively collaborate, but our proposed method for passive clock synchronization appears to be the first of its type.

III. Method

Our proposed method is designed around a typical sensor data acquisition scenario. The sensor (which we denote p) generates messages that either explicitly or implicitly contain a time-stamp. Some sensors provide an actual time, while others produce data at a sufficiently regular rate that the arrival of a message constitutes a "tick" of p's clock. An example of such an implicit clock is the SICK LMS 291

which generates messages at an approximately constant 75 Hz rate[2].

This data is then transmitted to the host (which we denote as q), and the host's clock is sampled. We know that some time e will elapse between the p's time stamp and q's time-stamp, but this time period is generally unobservable. We also assume that the jitter e varies erratically from one measurement to another.

A. Simplified drift-less version

We begin with a simplified version of the algorithm that does not allow the host or device clock to drift: there is a fixed offset between the two clocks. While this simplified method is not of practical usefulness, it illustrates the basic ideas of our approach without the complications introduced by drifting clocks.

Let A denote the offset between clocks p and q. This offset can not be directly observed, however, due to the fact that some unknown latency e elapses before clock q can be observed (see Fig. 1).

Specifically, suppose an event occurs at t_i on the host's clock. Because the sensor is offset by A, it measures the time as $(A+t_i)$. After some unknown delay e_i, the host receives the sensor's message; the time on the host's clock is now $(t_i + e_i)$. To summarize:

$$\begin{aligned} p_i &= A + t_i \\ q_i &= t_i + e_i \end{aligned} \tag{1}$$

By subtracting the two equations, we can write an expression without t_i:

$$p_i - q_i = A - e_i \tag{2}$$

Recall that we are unable to observe either A or e_i directly— we can only observe p_i and q_i. When the latency e_i is large, the value $p_i - q_i$ will be correspondingly smaller. In contrast, when e_i is small (near zero), corresponding to the case where the host was quickly able to process the sensor's data, $p_i - q_i$ will be large *and* a good estimate of A. Because $e_i \geq 0$, it follows that:

$$A \geq p_i - q_i \tag{3}$$

Let us now suppose that we have multiple measurements of p and q's clock. We can robustly estimate A as:

$$A \approx \max_i (p_i - q_i) \tag{4}$$

The error in the approximation above is equal to the *smallest* e_i. In other words, if we ever obtain a low-latency pair of clock samples (p_i,q_i) for which e_i is small, we will be able to recover a good estimate of A.

Our goal is to recover t_i, the time (according to the host's clock) at which the sensor observed the data. Once A is estimated, we have:

$$t_i \approx p_i - A \tag{5}$$

[2]In fact, SICK sensors can be configured to emit a scan counter. This scan counter is invaluable in maintaining proper synchronization since it allows the system to recover from message loss resulting from overflowing buffers or invalid checksums.

If we have an estimate of the minimum (best-case) latency of the system, our estimate can acount for it by subtracting that delay from t_i.

B. Drift Model

We now consider the case where the sensor and device clocks drift. Since the clocks drift, the offset between them is now a function of time. Let us write the offset between the two clocks as $A(p_i)$, where the offset is characterized in terms of the time on the sensor.

We will characterize the drift of the clocks in terms of the maximum amount by which the clock offset can change in any given interval of time. Specifically, given two points of time p_i and p_j as measured by the sensor, we wish to specify some function f such that:

$$|A(p_i) - A(p_j)| \leq f(p_i - p_j) \tag{6}$$

While our synchronization method applies to monotonic functions $f(\Delta p)$ of any form, we will focus on the common case where some there is maximum rate error between the two clocks. I.e., consider a single interval of time measured by the sensor and host to be of duration Δp and Δt respectively. Since $A(p_i) - A(p_j) = (p_i - t_i) - (p_j - t_j)$, we can rewrite Eqn. 6 as:

$$|\Delta p - \Delta t| \leq f(p_i - p_j) \tag{7}$$

We can now model the drift of these clocks in terms of rate error as:

$$(1 - \alpha_1)\Delta t \leq \Delta p \leq (1 + \alpha_2)\Delta t \tag{8}$$

In this model, we assume that the sensor's clock might be either fast or slow in comparison to the host's clock. This model is parametrized by α_1 (the extent to which p's clock could be fast) and α_2 (the extent to which p's clock could be slow). Note that α_1 and α_2 are positive, and for a typical sensor, we would expect the magnitude of α_1 and α_2 to be quite small: a few percent.

It is useful to rewrite this in terms of the drift of Δt in comparison to Δp, which we can do with some algebra:

$$\frac{\Delta p}{1 + \alpha_2} \leq \Delta t \leq \frac{\Delta p}{1 - \alpha_1} \tag{9}$$

Our goal is to bound the change in the offsets of the two clocks during this interval, i.e., $|\Delta p - \Delta t|$. There are two extreme cases to consider: where $\Delta p - \Delta t$ is maximized, and where $\Delta t - \Delta p$ is maximized. By substituting in the appropriate bounds for Δt in each case, we obtain the following bound:

$$|\Delta p - \Delta t| \leq \max\left(\Delta p - \frac{\Delta p}{1 + \alpha_2}, \frac{\Delta p}{1 - \alpha_1} - \Delta p\right) \tag{10}$$

Which simplifies to:

$$|\Delta p - \Delta t| \leq \max\left(\frac{\alpha_2 \Delta p}{1 + \alpha_2}, \frac{\alpha_1 \Delta p}{1 - \alpha_1}\right) \tag{11}$$

This is the model that we will use in our experiments. It makes intuitive sense, in that the change in the offset of the

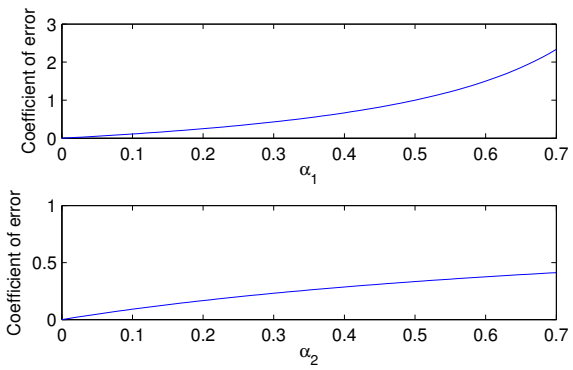

Fig. 2. Effect of α parameters on offset error. While the effect of α_1 and α_2 on offset error are well-approximated as linear for small values, their non-linear effects can become significant for systems with poor clocks.

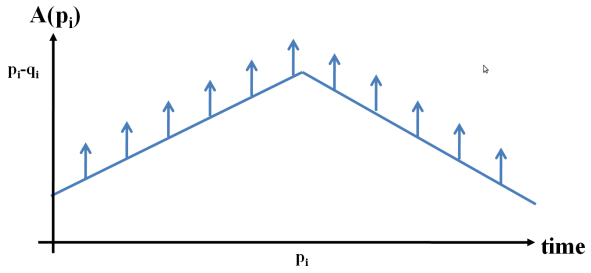

Fig. 3. Clock offset for a single observation. The minimum offset between the host and sensor clock is plotted. The tightness of the bound is the greatest at the point the measurement was made, since the sensor clock has not had time to drift away from the host clock. The bound gets looser on either side according to the worst-case rate predicted by $f(\Delta t)$. In this figure, we assume a rate drift noise model.

clocks increases in proportion to the length of the interval Δp. For the small positive values of α_1 and α_2 typical in real systems, their contribution is also approximately linear.

Let us also consider the behavior of these functions in extreme situations. If p runs extremely fast ($\alpha_2 >> 1$), the error in the offset approaches the elapsed time as reported by p. If p runs very slowly (consider $\alpha_1 = 1$), time actually appears to "stop" according to p's clock, with the result that offset error is unbounded. These effects are evident at smaller values as well, as shown in Fig. 2: at $\alpha_1 = 0.2$, the error bound is 25% higher than a purely linear model would predict. A value of $\alpha_1 = 0.2$ represents a large drift rate, but large values can arise in real-world systems that, for example, use inexpensive microcontrollers with RC oscillators.

C. Complete Method

The offset between clocks p and q, now that we consider drift, is a function of time. We will index this offset relative to the clock on p, writing $A(p)$. For example, in analogy to Eqn. 1, we write:

$$\begin{aligned} p_i &= A(p_i) + t_i \\ q_i &= t_i + e_i \end{aligned} \tag{12}$$

We can again subtract the two equations to eliminate t_i:

$$p_i - q_i = A(p_i) - e_i \tag{13}$$

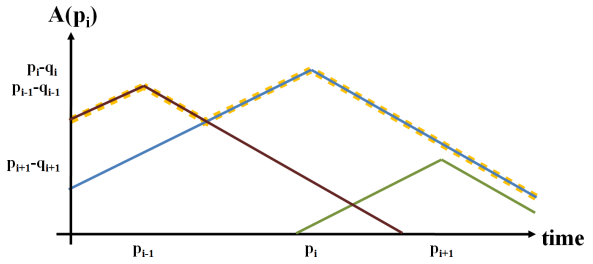

Fig. 4. Clock offset for multiple observations. When multiple observations are taken into account, the lower bound for $A(p)$ (shown as a thick dashed line) can be improved. In this example, the observations at $i-1$ and i are both locally useful. The observation at $i+1$ had higher latency and so does not contribute towards the offset computation.

Algorithm 1 Passive Synchronization Algorithm

```
{ Forward Pass }
for i = 1 to N do
    if p_i − q_i − f(0) ≥ p − q − f(p_i − p) then
        p = p_i
        q = q_i
        A_i = p_i − q_i − f(0)
    else
        A_i = p − q − f(p_i − p)
    end if
    t_i = p_i − A_i
end for
{ Backward Pass }
for i = N to 1 do
    if p_i − q_i − f(0) ≥ p − q − f(p_i − p) then
        p = p_i
        q = q_i
        A_i = max(A_i, p_i − q_i − f(0))
    else
        A_i = max(A_i, p − q − f(p_i − p))
    end if
    t_i = p_i − A_i
end for
```

We can conclude that:

$$A(p_i) \geq p_i - q_i \tag{14}$$

However, we wish to be able to infer information about the offsets at other points in time (i.e., $A(p_j)$) given a measurement at time i. Because of our noise model, we know that:

$$A(p_i) - A(p_j) \leq f(p_i - p_j) \tag{15}$$

We can substitute Eqn. 15 into Eqn. 14. With some algebra, we obtain:

$$A(p_j) \geq p_i - q_i - f(p_i - p_j) \tag{16}$$

This result makes intuitive sense: locally, observation i predicts that $A(p_i) = p_i - q_i$. However, due to the unknown clock drift, the farther away some query point p_j is, the less information we will have about $A(p_j)$. This behavior is shown graphically in Fig. 3 for the case of a rate-error noise model.

Finally, in order to take advantage of all available data, we arrive at a new "max" rule for estimating $A(p_j)$:

$$A(p_j) \approx \max_i \left(p_i - q_i - f(p_i - p_j)\right) \tag{17}$$

An example of this procedure is shown in Fig. 4, which shows three observation pairs, the $A(p)$ bound computed by each pair, and the resulting posterior estimate based on the max rule.

We can now finally recover the host's time t_j corresponding to p_j by combining Eqn.12 and Eqn.17:

$$t_j \approx p_j - \max_i \left(p_i - q_i - f(p_i - p_j)\right) \tag{18}$$

Our method has several useful properties.

Claim 1: Our method never over-estimates the clock offset, or equivalently, it will never claim an observation occurred before it actually did. In other words:

$$p_j - \max_i (p_i - q_i - f(p_i - p_j)) \geq t_j \tag{19}$$

Proof: From Eqn. 12, we know $A(p_j) = p_j - t_j$. We substitute into Eqn. 19:

$$A(p_j) \geq \max_i (p_i - q_i - f(p_i - p_j)) \tag{20}$$

This is true because $A(p_j)$ is greater than every possible value within the max expression, by Eqn. 16.

Claim 2: Our method will never produce a result worse than the naive synchronization algorithm that sets $A(p_i) = p_i - q_i$.

Proof: This follows immediately from Eqn. 17, since the max expression includes $p_i - q_i - f(p_i - p_i)$. Provided $f(0) = 0$, our estimate of $A(p_i)$ will be at least as large as $p_i - q_i$.

D. Computational Complexity

A literal implementation of our method using Eqn. 18 would yield an $O(N^2)$ algorithm for N observation pairs: there are N values of t_j to estimate, and each one requires a maximization over all N observation pairs.

However, it is immediately obvious that if $f(\Delta t)$ increases monotonically, the interval over which a single observation pair dominates the maximization in Eqn. 18 is compact. In other words, as time progresses in one direction or another, once an observation stops dominating, it will never dominate again.

This suggests a simple two-pass algorithm over the data (see Alg. 1). During each pass, the best known pair of observations is compared to the current observation pair; the pair with the larger value of $p_i - q_i - f(p_i - p_j)$ becomes the new best pair. This process is applied once in the forward direction and once in the reverse direction, and yields the same results as the $O(N^2)$ algorithm in $O(N)$ time.

The method can be trivially modified to work in online (causal) applications by omitting the second pass. This results in a slight decrease in synchronization accuracy due to the loss of synchronization information from future observations. The computational complexity for each observation is $O(1)$.

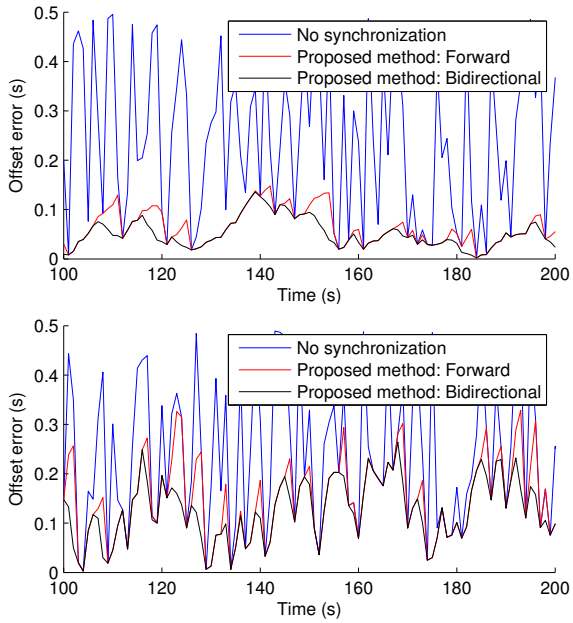

Fig. 5. Simulation results. Figures show the offset error resulting from three synchronization methods. Observations are obtained every 1 s and contaminated with uniformly-distributed random latency of up to 0.5 s. We show results for sensors with good clocks (top, $\alpha_1 = \alpha_2 = 0.01$) and poor clocks (bottom, $\alpha_1 = \alpha_2 = 0.05$). The algorithm's ability to exploit low latency measurements is evident, though a more accurate clock increases the usefulness of those observations. The bidirectional method out-performs the causal method, since it can exploit future information.

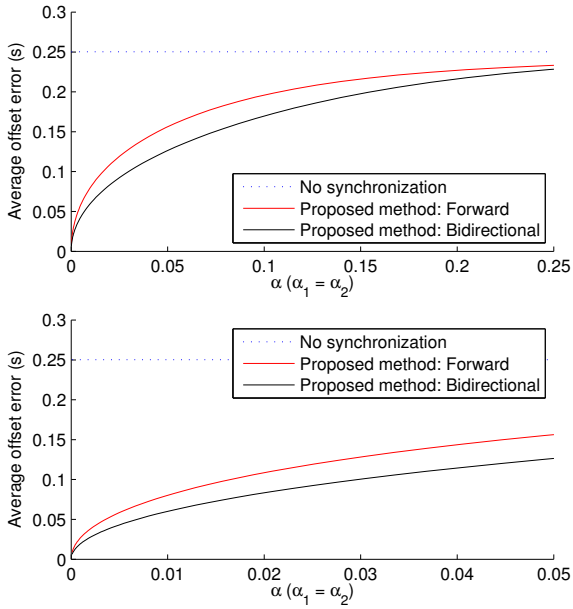

Fig. 6. Synchronization performance versus α. In this synthetic experiment, observations with uniformly distributed random latency of up to 0.5 s were obtained at 1 s intervals. The same data has been plotted twice, with a zoomed image on the bottom. Without our method, average synchronization error was 0.25 s independent of α. Our methods substantially improve on this average error, with performance better when the sensor's clock is accurate. The bi-directional method performs better, but requires non-causal processing. As the sensor's clock degrades (large values of α), performance of our methods asymptotically approach the "no synchronization" case.

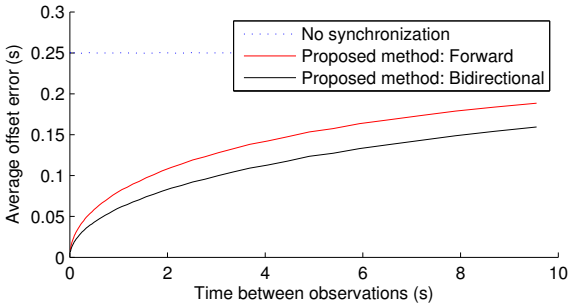

Fig. 7. Synchronization performance versus observation rate. As the time between observations is increased, the offset error increases. This experiment uses uniformly distributed random latency of up to 0.5 s, with $\alpha_1 = \alpha_2 = 0.1$.

IV. Results

While the behavior of our algorithm is illustrated in a toy example in Fig. 4, Fig. 5 shows a more complex example. This figure shows the behavior of the algorithm for two different clock accuracies: $\alpha = 0.01$ and $\alpha = 0.05$. (Note that Fig. 4 plots the estimate of $A(p)$, whereas Fig. 5 shows the offset error, which varies according to $-A(p)$. Thus, the downward "decay" seen in Fig. 4 appears upside-down.)

We also characterize the performance of our algorithm in terms of the synchronization error of its estimates. The amount of this error depends on two parameters: the accuracy of the clocks (as measured by α_1 and α_2), and the frequency with which observations are made. More accurate clocks increase the effect of low-latency measurements, while greater observation frequencies increase the density of low-latency measurements.

Fig. 6 shows the effect of the α parameters on synchronization error. Note that as α grows large, performance gracefully degrades to the same performance as the naive data association algorithm. We also see that the bidirectional variant of our algorithm has lower error than the causal version, which is logical considering that it makes use of future observations.

Increasing the time between observations has a similar negative effect on synchronization error, as shown in Fig. 6. Fewer observations decreases the frequency with which low latency observations are made.

V. Conclusion

We have presented a novel algorithm for passive synchronization of sensor data that significantly improves timing accuracy for common robot sensors. While the problem of time synchronization is well studied in the case of collaborating hosts, we believe this is the first solution to the synchronization problem that does not require the explicit coordination of one party. Our algorithm can significantly reduce the synchronization error, even in the presence of unknown latencies and for sensors with significant clock errors. Our algorithm also provides provable performance guarantees.

Our algorithm can be implemented in only a few lines of code, and a reference implementation is available at http://april.eecs.umich.edu.

Acknowledgements

This work was supported by U.S. DoD grant W56HZV-04-2-0001. We also thank the members of MIT's DARPA Urban Challenge for discussions and for helping with early implementations, particularly David Moore and Albert Huang.

References

[1] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. R. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson, "Autonomous driving in urban environments: Boss and the urban challenge," *J. Field Robot.*, vol. 25, no. 8, pp. 425–466, 2008.

[2] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun, "Junior: The Stanford entry in the urban challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 569–597, 2008.

[3] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams., "A perception driven autonomous urban vehicle," *Journal of Field Robotics*, September 2008, to appear.

[4] D. L. Mills, "Internet time synchronization: The network time protocol," United States, 1989.

[5] M. Zaman and J. Illingworth, "Interval-based time synchronization of sensor data in a mobile robot," in *Proceedings of the 1st International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP-2004), Melbourne, Australia, 5-8 December 2004*, December 2004, pp. 463–468.

[6] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, vol. 3, no. 3, pp. 146–158, 1989. [Online]. Available: http://dx.doi.org/10.1007/BF01784024

[7] R. Gusella and S. Zatti, "The accuracy of the clock synchronization achieved by tempo in berkeley unix 4.3bsd," *IEEE Trans. Softw. Eng.*, vol. 15, no. 7, pp. 847–853, 1989.

[8] K. A. Marzullo, "Maintaining the time in a distributed system: an example of a loosely-coupled distributed service (synchronization, fault-tolerance, debugging)," Ph.D. dissertation, Stanford, CA, USA, 1984.

[9] J. Kannisto, J. Kannisto, T. Vanhatupa, T. Vanhatupa, M. Hannikainen, M. Hannikainen, and T. D. Hamalainen, "Software and hardware prototypes of the ieee 1588 precision time protocol on wireless lan," in *Local and Metropolitan Area Networks, 2005. LANMAN 2005. The 14th IEEE Workshop on*, 2005, pp. 6 pp.+.

[10] M. G. de, R. Aarts, J. Meijer, and J. Jonker, "Robot-sensor synchronization for real-time seamtracking in robotic laser welding," in *Lasers in Manufacturing 2005*, E. Beyer, F. Dausinger, A. Ostendorf, and A. Otto, Eds. München, Germany: AT-Fachverlag GmbH Stuttgart, 2005, pp. 419–424. [Online]. Available: http://doc.utwente.nl/52662/

[11] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2004, pp. 39–49.

[12] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*. New York, NY, USA: ACM, 2002, pp. 147–163.