TriggerSync: A Time Synchronisation Tool

Andrew English, Patrick Ross, David Ball, Ben Upcroft and Peter Corke

***Abstract*— This paper presents a framework for synchronising multiple triggered sensors with respect to a local clock using standard computing hardware. Providing sensor measurements with accurate and meaningful timestamps is important for many sensor fusion, state estimation and control applications. Accurately synchronising sensor timestamps can be performed with specialised hardware, however, performing sensor synchronisation using standard computing hardware and non-real-time operating systems is difficult due to inaccurate and temperature sensitive clocks, variable communication delays and operating system scheduling delays.**

Results show the ability of our framework to estimate time offsets to sub-millisecond accuracy. We also demonstrate how synchronising timestamps with our framework results in a ten-fold reduction in image stabilisation error for a vehicle driving on rough terrain. The source code will be released as an open source tool for time synchronisation in ROS.

I. Introduction

Many robotic systems fuse multiple sensors and require measurement timestamps that are accurate with respect to a common clock – ideally the computer's internal clock. However, obtaining accurate local timestamps for sensor data is challenging due to varying clock frequencies, unknown and variable communication delays, and operating system scheduling delays.

Many sensors support simultaneous capture via a hardware trigger line. However, obtaining timestamps for these synchronised sensors with respect to a local clock often requires specialised computing hardware to measure the timing of trigger pulses, and matching triggered measurements together is difficult when communication delays are highly variable.

Without hardware support, synchronisation software can learn a mapping between a sensor's clock and the computer's local clock, however methods that do not require special software support from sensors produce a clock offset estimate that is biased by the minimum communication delay.

This paper presents TriggerSync, a framework for synchronising multiple sensors with hardware trigger lines to local computer time using standard computing hardware. Our framework makes use of one-way synchronisation algorithms, combined with sharing of triggered timestamps between clock estimators to gain many of the benefits of two-way synchronisation algorithms without requiring software support from sensors. We also provide a general purpose tool for time synchronisation in Robot Operating System (ROS).

Results demonstrate the effectiveness of this framework by synchronising offsets between a camera and an IMU to within approximately 100μs. We further demonstrate that synchronising timestamps improves image stabilisation performance (Figure 1) by ten-fold on a vehicle driving across rough terrain in an agricultural field.

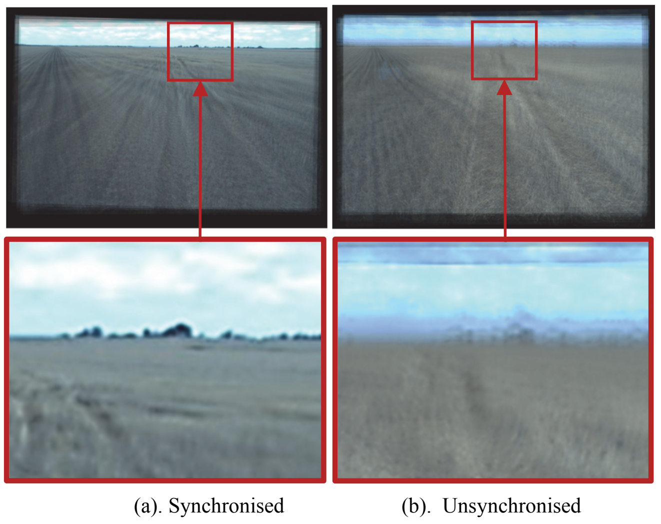


Figure 1: This figure shows the importance of accurate time synchronisation between sensors. Images show the average of 10 consecutive frames from an IMU stabilised camera with (a) synchronised timestamps and (b) unaltered timestamps. Synchronisation significantly improves stabilisation with features on the horizon accurately aligned in (a) and misaligned in (b).

Our framework allows seamless sharing of trigger information between clock estimators, as well as providing tools to assist with time synchronisation in ROS. The framework makes use of the open source TICSync timing library [14]. We will shortly release our source code as a time synchronisation library for ROS to provide a flexible and easy to use tool for time synchronisation of triggered and untriggered sensors in ROS.

II. Related Work

The goal of time synchronisation is to accurately know when measurements occurred with respect to a common clock, usually the computer's internal clock. This section has a summary of various methods of synchronising clocks, such as using specialised hardware, Global Positioning System (GPS), sensor data optimisation, and one and two-way software synchronisation methods.

Specialised hardware to achieve time synchronisation include interfacing all sensors with an FPGA [1][2], and measuring trigger line transitions with low-latency hardware interrupts on a dedicated processor[3]. The IEEE588 Precision Time Protocol Loop [4] standard provides accurate timestamps in hardware. Custom hardware solutions are not always practical, can be time consuming to develop and application specific.

GPS can be used for accurately synchronising clocks to UTC time [5]. For accurate results a Pulse Per Second (PPS) signal from a GPS receiver can be connected to a computer via a hardware interrupt line with a kernel mode driver [6].

*Authors are with the School of Electrical Engineering and Computer Science at Queensland University of Technology, Brisbane, Australia (email: a.english@qut.edu.au).

Without hardware support, software methods can perform time synchronisation by observing sensor measurements between sensors that have the same motion. Approaches include optimisation based methods such as [7], [8] and online filtering methods such as [9]–[11] where the time offset is explicitly included in a filtering framework. These motion-based methods are generally sensor specific, assume a static offset, and cannot account for variable transport delays.

Another software based method is to examine the send and receive timestamps of data packets. It is common to synchronise computers across a local network or the internet using methods such as NTP [12] the Berkley algorithm [13], IEEE 1588 Precision Time Protocol [4] or TICSync [14]. These algorithms use timestamps from two-way packet traces between computers to estimate the offset between their clocks. These two-way algorithms require software support from both devices, something which few sensors provide.

When sensors do not support producing two-way timestamps, one-way clock synchronisation algorithm such as [15] and [16] can be used to learn a one-way clock mapping. These methods improve timestamp estimates by removing stochastic delays, however, they estimate an offset that is biased by the unknown minimum transport delay. This leaves sensors with different minimum transport delays unsynchronised relative to each other. An advantage of these methods is that only the occasional low latency packet is required to obtain a good estimate of clock offset. This is helpful when synchronising over congested network links as well as when using non-real-time operating systems where scheduling delays can be highly variable.

Another common method of synchronising sensors is to initiate measurements via a hardware trigger line [17][18]. Hardware trigger inputs and/or outputs are often available on many common sensors such as cameras, IMUs, laser scanners and GPSs. Even standard PC hardware can detect trigger pulses, for example using the Linux-PPS API kernel driver.

While sensors that share a trigger line are able to capture measurements simultaneously, to assign unified timestamps to the triggered measurements they need to be matched together based on their receive timestamps. This matching is challenging when the variability in transport delays is large. Additionally, timestamps for triggered measurements are usually in the sensor clock timeframe, not the computer's clock. We argue that it is much more useful for timestamps to be both consistent between sensors, as well as closely synchronised to local computer time since it allows synchronisation with other sensors and computers that have been synchronised using other methods (such as NTP or TICSync). Sensor timestamps in local computer time are also important for closed loop robotic systems where the age of sensor measurements should be known in order to predict the necessary control input.

III. Background

In this section we summarise terminology for clock synchronisation, and briefly outline one-way clock estimators.

A. Clock Terminology

Synchronisation between a pair of clocks is quantified by clock *offset*, *skew*, and *drift.*

Consider a sensor that timestamps its measurements with its own internal clock time T_s whose time is obtained by the function $T_s = C_s(t)$, where t is true time. Similarly the computer's internal time is denoted as $T_c = C_c(t)$.

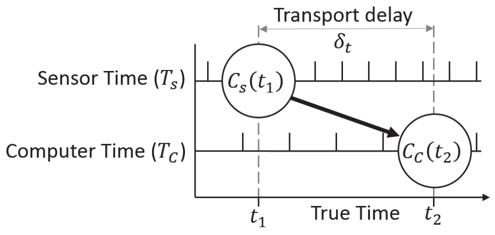


Figure 2: One-way timestamping. The sensor and computer each have their own internal clock running at different frequencies. The sensor timestamps measurement according to its internal clock and the computer later receives the measurement and notes the time according to its own clock. Measurements are delayed by some unknown and varying transport delay.

As shown in Figure 2, at some time t_1 the sensor takes a measurement and timestamps it with its internal clock time $C_s(t_1)$. The computer receives this packet at some later time t_2 at which time its local clock reads $C_c(t_2)$. The transport delay $t_2 - t_1$ includes communication and software delays. To synchronise between these clocks we need to discover the clock offset defined as

$$\tau(t) = C_c(t) - C_s(t).$$

Once the offset $\tau(t)$ is known it can be added to sensor message timestamp $C_s(t_1)$ to calculate the measurements time according the computer's clock $C_c(t_1)$, despite variable transport delays.

The clock *skew* is defined as the difference between the clock frequencies and is the first derivative of $\tau(t)$. Clock *drift* measures the relative frequency stability of clocks and is defined as the second derivative of $\tau(t)$.

B. Learning a One-Way Clock Mapping

In Figure 2 we plot the *measured delay* – the difference between the send timestamp $C_s(t_1)$ and receive timestamps $C_c(t_2)$. The data points are supported by a line whose slope is equal to the clock skew. Since clock frequencies are generally quite stable over a period of minutes is reasonable to assume a first order (constant skew) model for clock skew:

$$\tau(t) = C_c(t) - C_s(t) = \alpha t + \beta$$

where α is the clock skew and β is an initial offset.

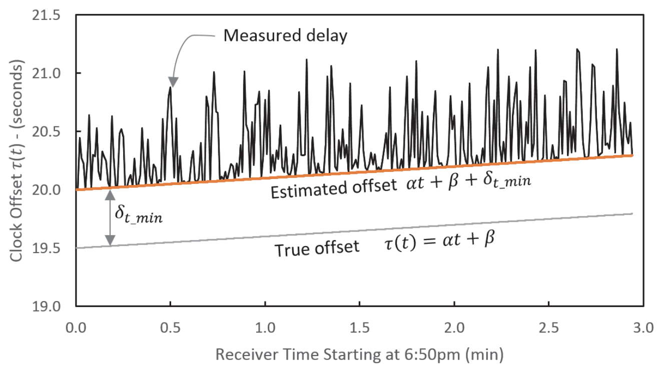


Figure 3: Measured delay trend is supported by a straight line. One-way clock estimators estimate the clock mapping represented by this straight line, however the true clock mapping is represented by a parallel line offset by the minimum transport delay δ_{t_min}. One-way estimators therefore produce a biased clock offset estimate

The true clock offset is equal to the measured delay minus the unknown and variable transport delay. By fitting a line hard up against the measured delay data points, one-way estimators such as [15][16] are able to estimate an offset defined by the line

$$\alpha t + \beta + \delta_{t\_min} = \tau(t) + \delta_{t\_min}$$

That is, these one-way methods produce a clock mapping that underestimates the offset by the minimum transport delay δ_{t_min}. This bias is expected since with only one-way timestamps it is impossible to measure the underlying network delay. From here on in the paper we will refer to this bias in one-way clock estimators as the *offset bias*.

IV. Design

Our approach to clock synchronisation is to apply a one-way clock estimator to the timestamps for each sensor and to share receive timestamps between these estimators for events that we know occurred simultaneously due to triggering. Clock estimators use the receive timestamps from the lowest latency sensor in a group of triggered sensors so as to minimise offset bias. This achieves timestamp synchronisation between sensors and also ensures timestamps are synchronised with respect to our local clock with the minimum offset bias possible. In this way we can achieve much of the function of two-way synchronisation algorithms without requiring software support from sensors.

A. Synchronised Clock Mappings for Triggered Sensors

Consider an application where we have two sensors with independent clocks, and both sensors are triggered simultaneously by a physical trigger line. We may correct each sensor's timestamps with a one-way clock estimator such as in Moon et al. [16], however the two sensors may have very different transport delays. This difference in minimum transport delay results in a different offset biases for each clock estimator, which in turn means the corrected timestamps for the two sensors will be unsynchronised with respect to each other. This difference in transport delay can be large, for example cameras often take tens of milliseconds to transmit images that can be several megabytes in size, whereas an IMU may take less than a millisecond to transmit a sensor measurement.

However, in the case of triggered sensors, we know that sensor measurements occur simultaneously and so their send times (in true time) are identical. We can treat the moment these measurements were produced as a single event, making it valid to choose either receive timestamp as input to either sensor's clock estimator. It is desirable to use the *earliest* of the two receive timestamps for both sensor's clock estimators since we would like to minimise offset bias due to transport delays. By matching simultaneous sensor measurements together and always using the earliest receive time, both clock estimators enjoy the offset bias of the sensor with the smallest transport delay. Both sensors will be as closely synchronised to local computer time as possible with the information available. Additionally, since the offset bias is the same for both sensors, the sensors will be synchronised relative to each other.

This concept can be extended to groups of sensors. If we have *n* sensors triggered from the same trigger line, then each trigger event will produce *n* sensor measurements, each with a

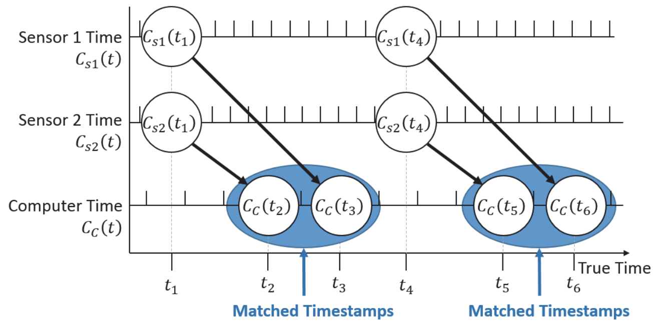


Figure 4: If we have prior knowledge that sensor 1 and 2 are triggered simultaneously then we can match messages together base on their arrival time and can use the earliest of the timestamps as input to the clock estimator for both sensors.

receive time according to our local clock. For each trigger event we use the *earliest* receive timestamp of these *n* measurements as the receive time input to all *n* estimators. With this scheme, only one sensor with the occasional low latency message is required in a group of triggered sensors for all sensors in that group to be closely synchronised to the local computer time.

In order to determine the earliest receive timestamp for a group of simultaneously triggered sensor measurements, the measurements must first be grouped according to their associated trigger pulse. This can be done by finding the events with the closest receive timestamps as shown in Figure 4. One drawback of our proposed system is that it is not robust to incorrect matches. To reduce the possibility of incorrect matches we match events to each other based on their *corrected* receive time (i.e. the send time converted to local time based on the estimated clock mapping) rather than their uncorrected receive time. This removes the variable communication delay from the matching process and results in reliable matching performance in most practical situations. Incorrect matches may still occur if the difference minimum transport between two triggered sensors approaches or exceeds half the triggering frequency. Detecting and resolving these incorrect matches remains future work for our system.

B. Software Design

The TriggerSync library has a distributed design with no central controller. An instance of TriggerSync is added to the source code for each sensor to be synchronised, and the library transparently shares timestamp information between the sensor drivers. The modification necessary to the source code of each sensor driver is usually very minimal, only requiring a few additional lines of code. Figure 5 shows a code snippet demonstrating how to include the TriggerSync library into a ROS driver node.

The library has been designed to be used in a ROS environment, and while the sensor driver doesn't necessarily need to be a ROS driver, the library uses ROS as a message passing framework to share timestamps between the processes running TriggerSync, as well as using multiple ROS libraries in its implementation.

A block diagram of TriggerSync is shown in Figure 6 and each of the components in this diagram will be described in more detail in the following sections.

```
#include "ros/ros.h"
#include “trigger_sync.h”
int main(int argc, char **argv) {
  TriggerSync ts(“my_sensor_clock”, “local_clock”, “trigger_line_1”);
  ros::NodeHandle nh;
  ros::Publisher pub=nh.advertise<sensor::sensor_msg>("/sensor ",1);
  sensor my_sensor;                    // Class for reading our sensor
  my_sensor::sensor_msg msg;   // ROS message for my  sensor
  while(ros::ok(){
    msg = my_sensor.read();  // Blocking read of triggered sensor
   // Use TriggerSync to correct receive time and publish trigger event
   msg.recv_time=ts.update(msg.send_time, ros::Time::now() );
    pub.publish(msg);
  }
}
```

Figure 5: Code snippet demonstrating how the TriggerSync library can be included in a ROS driver node. Only the lines in red need to be added.

1) *Sharing Trigger Events*

Each TriggerSync instance sends and receives trigger events to/from other TriggerSync instances via a common ROS topic named */event*. Using the library is usually as simple as calling the “update” method with send and receive timestamps as arguments. This non-blocking call returns a corrected receive time that is based on the current best estimate of the clock mapping. It also publishes the timestamps for use by other TriggerSync instances and internally buffers the event for

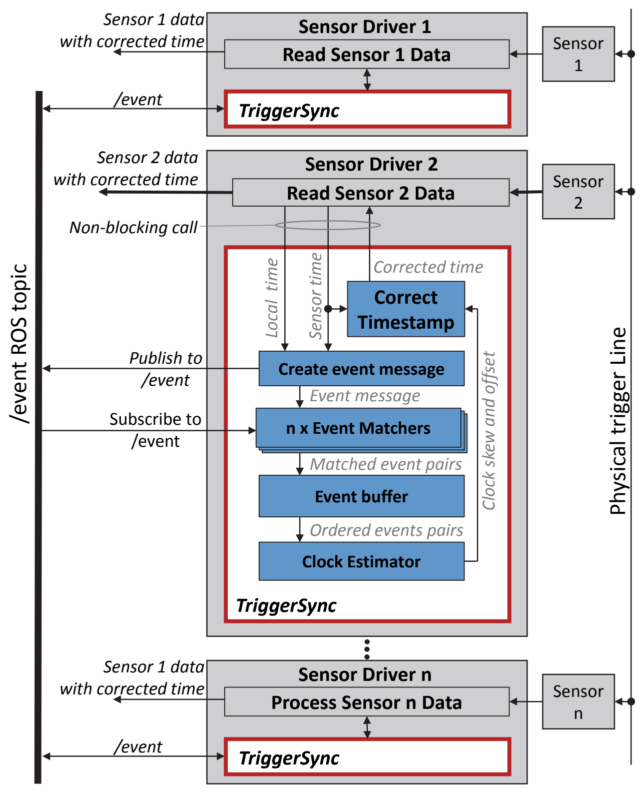


Figure 6: Block diagram of three sensors synchronised with TriggerSync. The middle instance of TriggerSyanc shows additional information with the internal structure.

```
string event_name         //ID to allow multiple trigger lines. Default is “trigger”
string device_clock_id    //Unique ID  for each sensor clock i.e. “IMU_clock”
string local_clock_id     //Unique ID for receive clock. Default is “local_time”
time device_time          // Trigger time according the the sensor clock
time local_recieve_time // Time that sensor measurement was recieved
```

Figure 7: ROS message definition for /event message.

matching with events from other sensors. The library maintains a separate thread for receiving, buffering and matching messages on the /event topic. All interactions with the /event topic occur automatically and users of the library are not required to publish or subscribe to this topic (though they may choose to).

Figure 7 shows the message definition for the /event topic. Trigger sources and clocks can all be identified by their own unique string to cater for complex systems with multiple clock reference frames and trigger lines. TriggerSync will synchronise timestamps with any sensor that publish /event messages that match its pre-configured *event_name* and *local_clock_id* stings.

2) *Matching Trigger Events*

Trigger events from the local sensor are paired with events received on the */event* ROS topic based on their corrected receive timestamps. To perform this matching, each TriggerSync instance has a bank of *event matchers* – one for matching local sensors trigger events against each of the other triggered sensors on the same trigger line. The number of event matchers in this bank adjusts automatically online.

The timestamp matchers are implemented using ROS’s Approximate Time Synchroniser[1], which implements a robust and parameter free method of matching messages based on their timestamps. Matched pairs of events are merged into a single event using the send timestamp from the local sensor, and the earlier of the two receive times.

3) *Event Buffer*

Since events are matched across multiple sensors running in separate processes, events are often matched out of order. Our clock estimator requires that timestamps are added in chronological order, so a simple fixed size buffer is used to re-order matched events in chronological order.

4) *Clock Estimator*

The clock estimator we use is an efficient implementation of the one way estimator by Moon et al. [16] from the TICSync timing library [14]. The TICSync library was publicly released for academic use with some geographic restrictions. To accommodate slow clock drift we use a switching estimator that maintains two overlapping filters that are periodically swapped to estimate a series of short overlapping piecewise linear sections.

5) *Correcting Timestamps*

The final step is to estimate a corrected receive timestamp from the current sensor send timestamp using the most recently estimated clock mapping (skew and offset). This estimation predicts slightly into the future from the most recent timestamp added to the estimator since events are delayed by both the event matchers and event buffer.

[1] http://wiki.ros.org/message_filters/ApproximateTime

C. *Additional Functionality*

In addition to implementing the timestamp sharing and matching scheme described in this paper, our library provides a ROS wrapper around most of the functionality provided by the TICSync library on which TriggerSync is based, allowing the TriggerSync to also function as a more general purpose timing library for ROS, providing one-way synchronisation for un-triggered sensors, and two-way synchronisation between two or more computers on a ROS network. Additionally, tools have been provided for visualising clock mappings (such as those in Figure 8) and measuring trigger pulses with low latency using a standard serial port.

V. Experiments

In this section we perform experiments to verify the accuracy of our time synchronisation method using the open source Kalibr calibration library. We also demonstrate that accurate timestamp synchronisation significantly improves IMU image stabilisation on a robot driving over rough terrain.

A. *Time Offset Measurement*

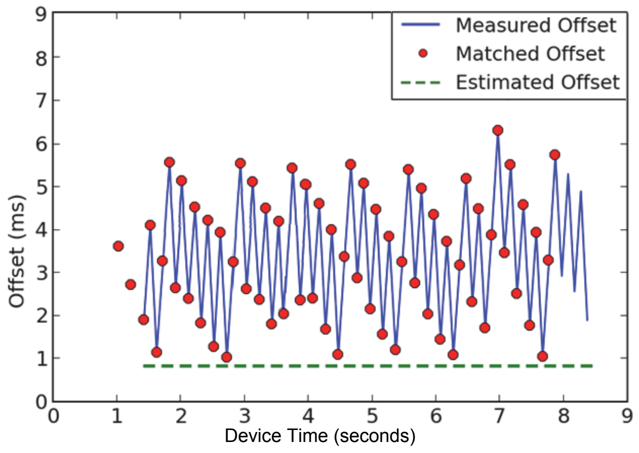
In this experiment we examine the accuracy of our proposed method using the open source Kalibr calibration library released by Furgale et al. [8], [19]. This library performs offline spatio-temporal calibration between an IMU and camera by analysing data captured while moving the sensors in front of a calibration target. In [8] the method demonstrated accuracies of around 50us RMS.

We use a global shutter camera (Point Grey GS3-U3-23S6C-C) and an industrial grade IMU (Vectornav VN100), each connected via USB to a PC running Ubuntu 12.04 and ROS. A 4Hz trigger line initiates image capture. The IMU produces measurements asynchronously at 40Hz while also reporting the timing of trigger pulses with respect to its internal clock. The TriggerSync library was added to both the IMU and camera ROS drivers and configured to share trigger events between each the two drivers.

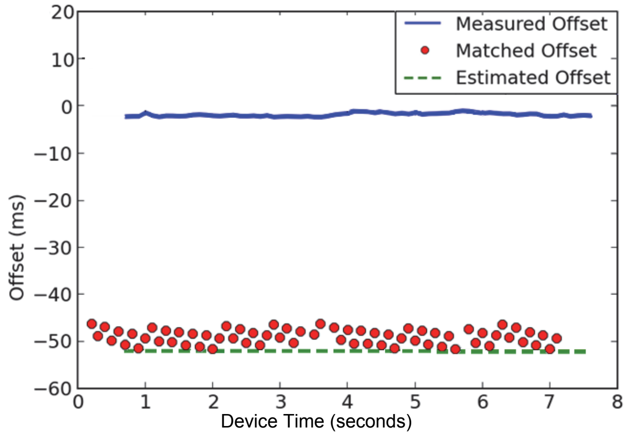
Additionally, to measure the "true" trigger time according to the local computer's clock we connect the trigger line to the Data Carrier Detect line of the computer's serial port and measure trigger pulses with the Linux Pulse Per Second (PPS) kernel API[2]. While we are not able to measure the delay caused by this API, we believe typical delays are likely to be less than a few tens of microseconds[20].

We carried out four sets of experiments, collecting ten separate 60 second datasets for each experiment which were then processed with the Kalibr library. In the first experiment we left receive timestamps unmodified. In the second experiment we correct receive timestamps using two one-one-way clock estimators *without* sharing trigger information between them. In the third experiment we correct timestamps of both sensors with TriggerSync (i.e. sharing trigger events between the clock estimators). In the fourth experiment we correct both sensors with TriggerSync and add include a third source of trigger events from the serial port measurements, effectively having three sensors in the TriggerSync network.

Table 1 summarises the measured time offset between the camera and IMU from the Kalibr library to indicate how well the camera and IMU are synchronised with respect to each other. Table 2 shows the delay between the "true" trigger time measured by the serial port and the corrected trigger timestamp measured by each of the sensors. This measurements indicates how the sensors are synchronised with respect to the computer's local clock.



(a). Clock mapping for IMU



(b). Clock mapping for camera

Figure 8: Clock offsets for experiment 3. Matched offsets in red are added to the clock estimators. Notice the lower latency IMU receive times have been used by the camera's clock estimator. This allows both sensor to be synchronised with respect to each other and also keeps the minimum offset bias to the local computer time.

In experiment 1 where timestamps were uncorrected we observed a large 72ms time offset between the sensors with variability (a standard deviation of 9ms) due to the highly variable communication delays.

In experiment 2 where receive timestamps were corrected without sharing trigger information (simple one-way estimators correcting each sensor), the offset between sensor is still large at around 50ms but is now much more consistent with a standard deviation of just 138us. This 50ms offset represents the difference between the minimum transport delays for the two sensors. In some systems this offset may be removed by subtracting a static delay from the corrected timestamps of the slower sensor, however we found this delay varies based on sensor configuration (baud rate, image size, data fields retrieved) as well as the image intensity (i.e darker

[2] http://linuxpps.org

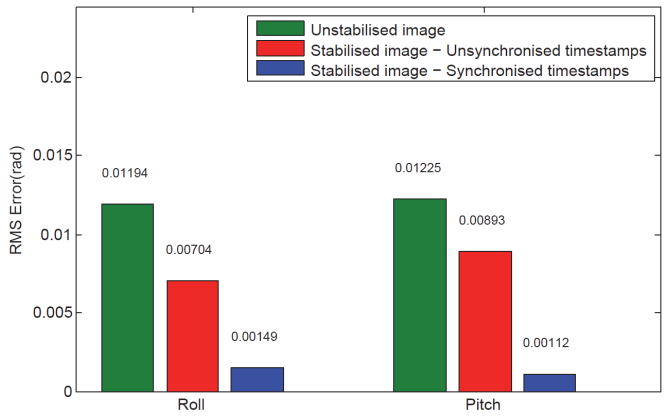


Figure 9: Results for IMU stabilised camera experiment. Roll and pitch errors in the stabilised image are made by fitting a straight line to the horizon.

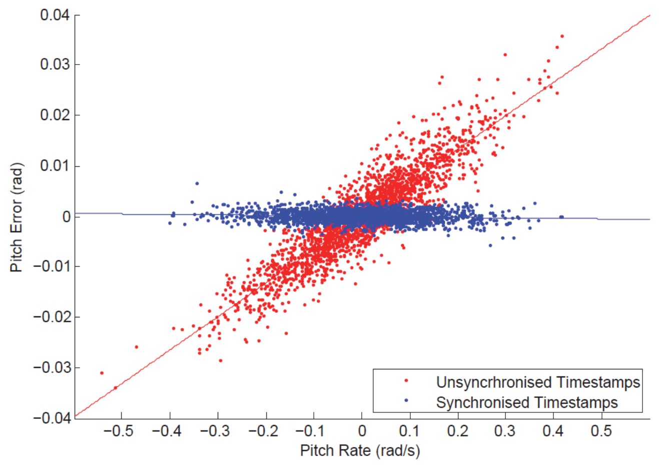


Figure 10: Stabilised image pitch error plotted against IMU pitch rate with lines of best fit. Synchronised sensors (blue) show only small movements in the stabilised image whereas unsynchronised sensors (red) show larger movement that is correlated with the IMU rotation rate which is consistent with a time delay.

camera frames were processed more quickly with constant exposure time).

In experiment 3 where TriggerSync was used to exchange receive timestamps, the camera and sensors achieved a mean synchronisation error of around 3.6us with a standard deviation of 195us indicating synchronisation between the two sensors was achieved to well below millisecond level. While the sensors were synchronised with each other, Table 2 shows

TABLE II. TIME OFFSET ERROR ($T_{IMU} = T_{CAM}$ + OFFSET) (ms)

Experiment	*Mean (ms)*	*Std. Dev. (ms)*
(1). Uncorrected Timestamps	*- 71.8*	*8.79*
(2). One-way Estimators	*-49.9*	*0.138*
(3). TriggerSync	*0.0355*	*0.195*
(4)TriggerSync with PPS input	*0.1828*	*0.2230*

TABLE I. SENSOR TIMESTAMP DELAY FROM COMPUTER TIME(ms)

Experiment	*Mean Delay(ms)*		*Std. Dev. Delay(ms)*	
	Camera	*IMU*	*Camera*	*IMU*
(1). Uncorrected Timestamps	*53.5*	*5.34*	*0.79*	*1.52*
(2). One-way Estimators	*51.7*	*2.56*	*0.14*	*0.14*
(3). TriggerSync	*1.14*	*1.18*	*0.22*	*0.25*
(4)TriggerSync with PPS input	*-0.02*	*-0.007*	*0.013*	*0.008*

their timestamps lag behind the computer's local clock by the minimum transport delay of the IMU, in this case about 1.1ms.

Figure 8 shows a graph of measured, matched and estimated offsets for a few seconds of experiment 3. Since the IMU has a lower latency communication channel, the matched events added to its estimator (red dots in Figure 8a) are the same as the measured offsets, however the matched events for the camera (red dots in Figure 8b) use the receive timestamps from the IMU to bring both sensors into synchronisation with the minimum possible offset bias.

In the fourth experiment where the additional low latency measure of trigger timing from the serial port is added as a third trigger source, the sensors are still well synchronised with each other (with a mean offset of 0.18ms and standard deviation of 0.22ms) and the sensors are additionally extremely well synchronised with the computers clock with a measured offset bias of around 20us with a standard deviation of 7us. Note that it is unsurprising that this offset measurement is so small since in this case we are using our ground truth measurement as one of the inputs to the system. However this does demonstrates that if the minimum transport delay of the fastest triggered sensor is not small enough for the desired level of synchronisation with the local computers clock, this delay can be reduced further by adding an addition lower latency measure of trigger transitions such as via a PC serial port.

B. *Application – Image Stabilisation*

In this experiment we demonstrates the importance of time synchronisation by presenting a real-world application involving an IMU stabilised camera mounted on a vehicle driving on rough terrain.

We mount the IMU and camera from the previous experiment to an agricultural robot. Images are triggered at 10Hz while the IMU measurements are captured asynchronously at 200Hz and again the IMU reports the timing of the trigger pulses with respect to its internal clock. The vehicle is driven on uneven ground twice along the same route, once with sensor timestamps synchronised using the method outlined in this paper and again with receive timestamps left uncorrected.

Each image is matched to the nearest IMU message based on their timestamps. The IMU attitude is then estimated at the image timestamp by linear extrapolation using the IMU rotational velocity. Roll and pitch in the image is then corrected by rotating and translating the image according the estimated IMU attitude. Correct stabilisation allows distant features (i.e. the horizon) to remain stationary in the stabilised image stream, while inaccurate timestamps will cause unwanted movement in the stabilised images.

Figure 1 qualitatively demonstrates the resulting improvement in stabilisation with synchronised timestamps by showing the average value of 10 consecutive stabilised images overlaid on one another. In Figure 1a where timestamps are corrected, the consecutive frames align well and the horizon is sharp, indicating good image stabilisation. This contrasts Figure 1b. where timestamps are uncorrected and consecutive images do not align well, making the horizon appear blurry.

To quantitatively evaluate image stabilisation performance, the vehicle path was chosen to give an unobstructed view of a flat horizon. In each stabilised image the roll and pitch error is measured by fitting a straight line to the horizon using the method outlined in our previous work [21]. Note that in these results the offset errors have been high-pass filtered to remove slowly varying errors in IMU attitude measurements to more clearly demonstrate the higher frequency errors caused by time synchronisation inaccuracies.

Figure 9 shows the measured stabilised image roll and pitch errors for these two experiments. These results show that in this case stabilising the imagery with uncorrected timestamps reduces the RMS roll and pitch in the stabilised image by around 30-40%, where synchronised timestamps give much better results with an approximately a ten-fold decrease RMS error.

Figure 10 plots the measured pitch errors against the IMU rotation rate along with lines of best fit. The errors for the synchronised experiment show little deviation and are centred near zero. The errors for the unsynchronised experiment are larger and show a strong correlation with the IMU rotation rate, which is consistent with a time delay. The slope of this line of 0.067±0.001s seconds indicates the time delay between sensors and is consistent with the unsynchronised time delay measured in the previous section of 72±9 milliseconds.

VI. Conclusion

This paper demonstrated a method of synchronising timestamps for triggered sensors to each other as well as to our computer's local time. Importantly the method requires no special computing hardware or custom kernel drivers and is applicable to a wide range of off the shelf sensors. Additionally we will release our source code as an open source time synchronisation library for ROS called TriggerSync to provide a flexible, easy to use tool for time synchronisation of both triggered and un-triggered sensors.

Acknowledgment

This work was supported in part by the Australian Research Council Linkage Project LP110200375 "Robotics for Zero Tillage Agriculture" awarded to the Queensland University of Technology, SwarmFarm Robotics and the Australian Centre for Field Robotics.

References

[1] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Y. Siegwart, "A Synchronized Visual-Inertial Sensor System with FPGA Pre-Processing for Accurate Real-Time SLAM," 2014.

[2] J. Ma, S. Susca, M. Bajracharya, L. Matthies, M. Malchano, and D. Wooden, "Robust multi-sensor, day/night 6-DOF pose estimation for a dynamic legged vehicle in GPS-denied environments," *2012 IEEE Int. Conf. Robot. Autom.*, pp. 619–626, May 2012.

[3] 1939- Maune David F. (David Francis), A. S. for Photogrammetry, and R. Sensing, *Digital elevation model technologies and applications : the DEM users manual*, 2nd ed. Bethesda, Md. : American Society for Photogrammetry and Remote Sensing, 2007.

[4] *IEEE 1588-2002, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems."*2002.

[5] J. Mannermaa and K. Kalliomaki, "Timing performance of various GPS receivers," *Freq. Time …*, no. l, pp. 8–11, 1999.

[6] M. Behn, V. Hohreiter, and A. Muschinski, "A Scalable Datalogging System with Serial Interfaces and Integrated GPS Time Stamping," *J. Atmos. Ocean. Technol.*, vol. 25, no. 9, pp. 1568–1578, Sep. 2008.

[7] J. Kelly and G. Sukhatme, "A general framework for temporal calibration of multiple proprioceptive and exteroceptive sensors," *Exp. Robot.*, pp. 1–15, 2014.

[8] P. Furgale, J. Rehder, and R. Siegwart, "Unified temporal and spatial calibration for multi-sensor systems," *2013 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pp. 1280–1286, Nov. 2013.

[9] M. Bosse, R. Zlot, and P. Flick, "Zebedee: Design of a spring-mounted 3-d range sensor with application to mobile mapping," *Robot. IEEE Trans.*, vol. 28, no. 5, pp. 1–15, 2012.

[10] M. Li and A. Mourikis, "3-D motion estimation and online temporal calibration for camera-IMU systems," *Robot. Autom. (ICRA), 2013 IEEE …*, pp. 5709–5716, May 2013.

[11] F. M. Mirzaei and S. I. Roumeliotis, "A Kalman Filter-Based Algorithm for IMU-Camera Calibration: Observability Analysis and Performance Evaluation," *IEEE Trans. Robot.*, vol. 24, no. 5, pp. 1143–1156, Oct. 2008.

[12] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network time protocol version 4: Protocol and algorithms specification," *IETF RFC5905, June*, 2010.

[13] R. Gusella, S. Zatti, T. A. of the C. S. Achieved, and by T. in B. U. 4.3BS, "The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD," *Softw. Eng. IEEE Trans.*, vol. 15, no. 7, pp. 847–853, 1989.

[14] A. Harrison and P. Newman, "TICSync: Knowing when things happened," *Robot. Autom. (ICRA), …*, pp. 356–363, May 2011.

[15] L. Zhang, Z. Liu, and C. H. Xia, "Clock synchronization algorithms for network measurements," *INFOCOM 2002. Twenty-First …*, vol. 00, no. c, pp. 1–10, 2002.

[16] S. Moon, P. Skelly, and D. Towsley, "Estimation and removal of clock skew from network delay measurements," *INFOCOM'99. Eighteenth …*, vol. 00, no. c, 1999.

[17] K. Schmid, P. Lutz, and T. Tomić, "Autonomous Vision-based Micro Air Vehicle for Indoor and Outdoor Navigation," *J. F. …*, 2014.

[18] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "PIXHAWK: A system for autonomous flight using onboard computer vision," *2011 IEEE Int. Conf. Robot. Autom.*, pp. 2992–2997, May 2011.

[19] P. Furgale, T. D. Barfoot, and G. Sibley, "Continuous-time batch estimation using temporal basis functions," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012, pp. 2088–2095.

[20] J. Quesada and J. U. Llano, "Evaluation of clock synchronization methods for measurement and control using embedded Linux SBCs," *… (REV), 2012 9th …*, 2012.

[21] A. English, P. Ross, D. Ball, and P. Corke, "Vision based guidance for robot navigation in agriculture," *2014 IEEE Int. Conf. Robot. Autom.*, pp. 1693–1698, May 2014.