A Low-Cost System for High-Rate, High-Accuracy Temporal Calibration for LIDARs and Cameras

Hannes Sommer, Raghav Khanna, Igor Gilitschenski, Zachary Taylor, Roland Siegwart, and Juan Nieto*

***Abstract*— Deployment of camera and laser based motion estimation systems for controlling platforms operating at high speeds, such as cars or trains, is posing increasingly challenging precision requirements on the temporal calibration of these sensors. In this work, we demonstrate a simple, low-cost system for calibrating any combination of cameras and time of flight LIDARs with respect to the CPU clock (and therefore, also to each other). The newly proposed device is based on widely available off-the-shelf components, such as the Raspberry Pi 3, which is synchronized using the Precision Time Protocol (PTP) with respect to the CPU of the sensor carrying system. The obtained accuracy can be shown to be below 0.1 ms per measurement for LIDARs and below minimal exposure time per image for cameras. It outperforms state-of-the-art approaches also not relying on hardware synchronization by more than a factor of 10 in precision. Moreover, the entire process can be carried out at a high rate allowing the study of how offsets evolve over time. In our analysis, we demonstrate how each building block of the system contributes to this accuracy and validate the obtained results using real-world data.**

I. Introduction

Multi-sensor systems that involve cameras and LIDARs, e.g. for motion estimation, are ubiquitous in robotics research. An easy to use, fast and reliable method to acquire accurate and high frequency estimates about when sensors actually take measurements and the reliability of their hardware timestamps, while being independent of any other spatial or intrinsic calibration, has the potential to save a lot of tedious calibration work and errors. This is particularity true for research systems that contain large numbers of different sensors. Furthermore, as the underlying algorithms and concepts for such multi-sensor systems grow more mature, these systems are about to hit the market in diverse end-user products. An example of this is in the automotive field, where a combination of multiple cameras and LIDARs are typically used for autonomous driving and advanced driver assistance systems. Furthermore, recent developments in LIDAR technology and strong interest from industry, promise wider availability of this sensor type and a further reduction in prices. This means that the availability of multi-sensor setups is likely to significantly increase in the future. When these setups are utilized on high speed platforms (such as cars, trains or multi-copters), or used to jointly perceive highly dynamic environments, precise time synchronization becomes a critical aspect. Experience with multi-sensor fusion shows that un-modeled delays can have a strong impact on the performance of motion estimation systems [1]. Unfortunately, most multi-sensor systems are not always hardware synchronized, particularly when low-cost sensors and commodity personal computers are used. And even if hardware synchronization is available, there may still be unknown offsets with respect to the synchronization signal.

*The authors are with the Autonomous Systems Lab, ETH, Zürich {hannes.sommer, raghav.khanna}@mavt.ethz.ch, {igilitschenski, ztaylor, rsiegwart, jnieto}@ethz.ch

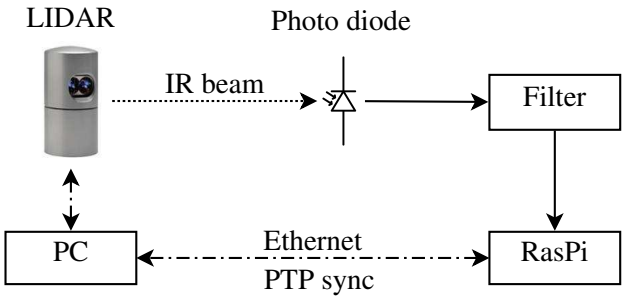

Fig. 1. The setup for LIDAR-to-CPU temporal calibration. When a LIDAR beam hits the photo diode a hardware interrupt is triggered on the Raspberry Pi and a low latency timestamp is assigned and sent to the PC.

The current state-of-the-art in multi-sensor timing calibration usually addresses the problem either by exploiting co-observed motion [2], [3] within self-calibration approaches, or using hardware synchronization signals [4], [1]. Motion based methods require a large amount of data to achieve millisecond precision (down to about 0.1 ms with targets such as a checker board, in laboratory environments [1]) and their accuracy is highly dependent on the motion. They also have no means to observe the offset between a sensor and the computer's system time. Hardware signal based methods demand suitable sensors that support this functionality (excluding many low cost camera systems, among others) and additional wiring to every sensor. Our work targets cases where hardware synchronization is not possible or too expensive, or where additional means for validation or inspection are required. This work is motivated by the observation that the use of simple external devices involving LEDs (for cameras) or photo diodes (for LIDARs) can result in significantly improved performance in comparison to motion based approaches. While this comes at the cost of an extra device, the gain in high rate and high accuracy estimates of the timing offsets can be particularly useful to optimize the exploitation of hardware or receive timestamps. Our approach can also serve as an accurate benchmark (ground truth) for better / easier evaluation of self-calibration algorithms. The proposed device is based on a Raspberry Pi 3 which is connected either to several LEDs for temporal camera to CPU calibration or a photo diode

for LIDAR to CPU calibration. PTP is used for precise time synchronization between the calibration device and the CPU of the system to which the sensors are attached. Overall, this work makes the following primary contributions

- A system for calibrating the time offset between the actual camera exposure taking place and the corresponding image timestamp (hardware clock and arrival).
- A system for calibrating the time offset between LIDAR timestamps and actual measurements.
- An evaluation of the accuracy and precision obtained using the two systems and 5 popular sensors, and validation for both based on hardware signals.
- A free and open source, ROS compatible C++ library to simplify hardware time related tasks for authors of device drivers, such as translating timestamps, and providing the user with tuning and inspection options. Inspection tools and several modified device drivers are also provided.[1]

II. Related work

To the best of our knowledge very little has been published about temporal sensor calibration using external devices. The only example we are aware of is [5] in which the authors use a single LED flashed at known system time (through a microcontroller) together with a high speed camera recording at 100 Hz This could be considered a simpler version of our LED-signaler device. The fundamental problems with a single LED are first, the achievable accuracy is limited by the maximum of exposure time and LED flash duration - as for most simple approaches with LEDs, and second the smaller this maximum is, the less probable it is that an LED is flashed during the camera exposure.

Two-way synchronization methods like TICSync [6], PTP [7], or NTP perform well but require support by the sensor's firmware, which is rare and expensive.

TriggerSync [4] operates by exploiting co-observation of simultaneous trigger signals. This is a great practical solution for online sensor synchronization if the sensors have trigger or synchronization inputs and very accurate when one of the sensors has very low latency or the main CPU has low latency direct inputs such as a RS-232 port [8]. Unfortunately many setups do not match these requirements. This is especially true in the case of LIDARs. Furthermore, the estimation rate is limited for similar reasons to the single LED approach above, due to the ambiguity that occurs when the trigger rate is high. This problem makes it a) even more restrictive regarding the sensors that can be used, when they are supposed to take measurements at a higher rate than the trigger rate, and b) less useful as a tool to analyze the statistical properties of the individual delays. A further difference is that TriggerSync must rely on the sensors handling hardware synchronization correctly, whereas our approach is directly based on the physical act of taking a measurement.

[1] `https://github.com/ethz-asl/cuckoo_time_translator/` (BSD-3-Clause)

Target-less self-calibration methods such as described in [2] or [3] rely on the same motion being observed in different sensors to find their relative offsets. This typically require many seconds of sensor data to reach a precision of milliseconds. This limitation means that these approaches cannot analyze the short term stability of the transport and processing delays or accuracy / stability of sensor's hardware timestamps. Furthermore, they cannot provide insight into the offset between CPU and sensor time.

The approach presented in [1] can achieve very high accuracy for temporal calibration LIDAR to camera by supporting the motion estimates with a checkerboard in the camera's field of view. However it still requires much more data (thousands of LIDAR measurements and hundreds of images) to reach the precision for the *average* delays that our approach can yield for each single image or LIDAR pulse / revolution. Given that much data, the variance of the average offset estimate per measurement can be much smaller, bounded from below by the variance of the physical delays within that data. For example, if we assume 20 s of data from a LIDAR spinning at 50 Hz our method accumulates 1000 delay estimates, each of which has a random error with a standard deviation (SD) of about $\sigma := 1\,\mu$s. Assuming independence[2] we get a theoretic SD of the estimated average delay of about $\sigma/\sqrt{1000} \simeq 32$ ns. Typically, the true average delays over 20 s are far less stable. However, this instability comes from variations in the actual delays, something our method allows a user to inspect. Please note that this is not about the method's accuracy — only its precision. Nonetheless, is the right quantity to compare with competing methods, for which typically only their precision is published due to fundamental problem of obtaining precise ground-truth data for calibration. Another benefit of the simplicity and immediacy of our method is that we can provide small upper bounds for the actual inaccuracy.

Another typical problem most motion based approaches suffer from is that they do not have the ability to analyze the delays and hardware clocks independently of other calibration parameters such as spatial extrinsics or even sensor intrinsics. Solving the calibration problems jointly does not prevent high precision, in fact it might even improve it [1]. But it typically makes the problem more complex and fragile as well as making it more sensitive to the quality of the model and of the prior calibration.

III. Design

The core idea of our sensor-to-CPU temporal calibration approach is to create a small, low cost device that allows the CPU to a) detect the event of taking a measurement for active sensors, such as LIDARs, or b) trigger physical events that can be detected and identified in the measurements of a passive sensor, such as a camera. In order for the calibration to be accurate it is necessary that the design of the device yields low unknown latency between an event and

[2] Assuming independence is probably to coarse considering the Raspberry Pi we employ for our experiments. But slightly more expensive hardware can almost diminish this source of inaccuracy — if needed.

its detection. In this paper we present two such devices: a *LIDAR beam detector* (III-A.1), and a *LED signaler* (III-A.2) for laser-CPU and camera-CPU temporal calibration respectively. For both devices it is essential to have accurate timestamps for edges of a digital electrical signal. We assume here that the computer receiving the sensor measurements (PC) does not have an appropriate input. We solve this by connecting a Raspberry Pi 3 Model B with 1200MHz ARM CPU (RasPi) via Ethernet, and synchronize it's clock with the PC using the *Precision time protocol* (PTP, see III-B). We use its general purpose input output (GPIO) pins to directly trigger an interrupt on its ARM CPU. In order to handle these interrupts at low latency, we use a specifically developed kernel module on the RasPi. In our setup Linux is utilized on both machines. In case the PC does have such an input (e.g. the Data Carrier Detect line of a RS-232 port as used in [4]) the approach is even easier to realize because no RasPi or PTP synchronization is required.

A. Calibration-devices

In the following sections we outline the two devices developed and used to perform the calibration.

1) LIDAR beam detector: The *LIDAR beam detector*, is a photo diode based detector of LIDAR pulses. This detector is used as depicted in Figure 1 to allow highly accurate and high rate *time of flight* (TOF) LIDAR-to-CPU temporal calibration.

a) Working principle: Whenever a laser pulse of the TOF-LIDAR hits the photo diode the impact gets detected through the filter circuit in the RasPi and a timestamp of the Linux system time, or for this paper *CPU-time* (CT), is recorded and sent via TCP to the PC. As the RaspPi's CT is synchronized via PTP to the PC's CT it is possible to align these timestamps directly with the receive timestamps of the LIDAR for any particular measurement. For this to work it is necessary to place the photo diode at a place where a beam hits it and the corresponding beam's bearing needs to be retrieved. To retrieve the bearing we manually selected a point in the LIDAR's point cloud using the live data visualization capability of RVIZ, a visualization tool provided by the Robot Operating System (ROS)[9]. As the bearing stays constant over time when both the LIDAR and the photo diode are stationary this step only has to be done once per LIDAR for a series of delay estimations.

b) Circuit: The entire detector circuit we used is an IR-photo diode, 900 nm, SFH 203 FA, Osram Semiconductors connected with its anode to the base of a BC547 transistor, whose emitter is grounded, while its cathode is pulled up to 5V through a 10 kΩ resistor. The transistor's collector is pulled up with a 2 kΩ resistor to 5 V. The signal at the collector is sent through a Schmitt trigger (74HCT14) and a 10 kΩ resistor to the GPIO of the RasPi to trigger the interrupt.

c) Data assignment: The periodic nature of the measurements (e.g. per revolution of the spinning LIDAR) yields an assignment problem, as the beginning of the two measurement series (LIDAR and photo diode) may not coincide. This yields an ambiguous latency by multiples of the time for one LIDAR revolution (typically $\gtrsim 20$ ms). In order to resolve this ambiguity we manually block the path from the LIDAR to the photo diode for a few periods. This blocking event is easy to automatically detect, as it appears as gaps in the photo-diode time-series data along with unusually short ranges measured by the LIDAR. Having detected this in both the photo-diode and laser time series we compute a relative shift for the laser time series which aligns its shorter range data to the gap in the photo diode time series. A shift is enough for the entire data assignment after gaps in both series have been filled with placeholders for missed packages from the LIDAR or undetected events on the photo diode. This can be done for both the laser and photo diode time-series based on the approximate LIDAR revolution period. After applying the correct shift to the dataset containing the blocking event a good first estimate for the transport delay can be retrieved. This first estimate may then be used to resolve the ambiguity for future datasets since it only needs to be accurate to within one revolution of the sensor. For the same reason this step is *unnecessary* in case of sufficiently accurate prior knowledge.

d) Accuracy: The detector itself is not without delay. However, is possible to retrieve bounds for the delay. First an upper bound for the delay from when the diode the diode detects the laser until the RasPi takes a timestamp can be found by employing an IR-LED to periodically trigger the photo diode. This can then be compared to the trigger signal via the use of a GPIO output of the RasPi. This is done by changing the output pins value whenever the RasPi receives an interrupt and after taking the timestamp using a suitable oscilloscope. The oscilloscope is only used here for validating our method, and is not necessary for application scenarios. With our setup the overall delay from the positive edge of the IR-LED current to the RasPi feedback was randomly distributed between 5 and $10\,\mu$s. Comparing the induced photo current in the photo diode between when it was triggered with the IR-LED and when a LIDAR beam hits it (Figure 2) using an oscilloscope clearly shows that the first signal edge is even steeper for the laser beam. Therefore, we assume an upper bound for the delayed LIDAR pulse arrival to RasPi timestamp of $10\,\mu$s. The synchronization error between RasPi and the PC is within approximately $\pm 20\,\mu$s according to the internal assessment of the ptpd[3]. In total we expect the absolute error for our setup per **single** laser pulse to be within $[-20, 30]\,\mu$s. The concept has the potential for nanosecond accuracy when used with more expensive hardware.

2) LED signaler: The *LED signaler*, is comprised of a LED display with 10 large LEDs that are sufficiently bright to be visible in the camera images, even with very short exposure time (e.g. 0.1 ms). These LEDs constitute a digital "clock" employing a special redundant encoding to prevent false readings through the camera images. The LED Signaler is employed as depicted in Figure 3 to allow high rate camera-to-CPU temporal calibration with an expected

[3] `https://sourceforge.net/projects/ptpd/`

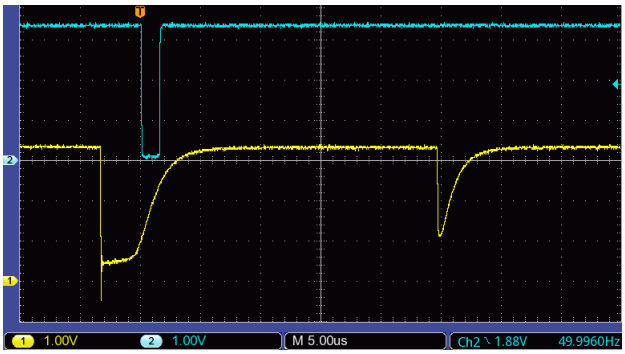

Fig. 2. The photo current of the photo diode (yellow, measured as voltage at a 10 kΩ resistor) caused by the impact of two consecutive laser pulses of a LMS151 spinning at nominal 50 Hz and the feedback signal of the RasPi after taking the timestamp (cyan). It shows the typical delay about 3.2 μs of the feedback. The second valley of the yellow curve corresponds to the subsequent laser pulse from the LIDAR. In the RasPi we prevent in software (based on the temporal distance) to take another timestamp at the second (no feedback).

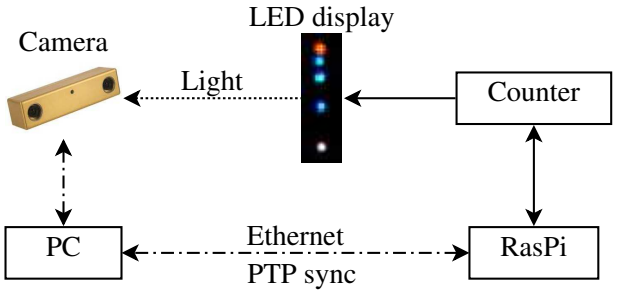

Fig. 3. The setup for Camera-to-CPU temporal calibration. Every image taken by the camera can be located in time based on what LED state is displayed. The LED status is controlled by the Raspberry PI with little unknown latency.

accuracy per measurement of approximately the minimal exposure time of the camera.

a) Working principle: The principle here is to observe the state of the LED "clock" in the camera images and use this state to associate the image timestamps with the RasPi timestamps of a corresponding state. Our setup consists of an LED display comprised of 10 LEDs in a row. The two outermost LEDs are always kept on in order to predict the locations of the inner LEDs. The 8 inner LEDs (0..7) are driven by a digital 6-bit counter at a user defined frequency, and define the state of the display. The 4 most significant bits directly control LEDs 4..7, while the two least significant bits control the other 4 LEDs, such that exactly only one is on at a time and its index corresponds to the binary number encoded with the two bits. Hence, the $(4 + 4)$ LED setup can display $2^6 = 64$ states. The digital counter that controls the LEDs also triggers an interrupt on the RasPi through one of its GPIO ports, whenever the LED starts showing a "0". Similar to the LIDAR beam detector setup, the RasPi assigns a timestamp to such events using the Linux CT, which is recorded and sent via TCP to the PC. These "0" timestamps are then used to compute timestamps for every state of the LED display, assuming a constant frequency between two "0"s of the LED display. To generate the input signal for the clock-counter we used our kernel module running on the RasPi, but it could be any source.

b) Data Assignment: To determine the timestamp of an image to a precision of the camera exposure time (τ_e), the LED display must be driven at a frequency of ($\frac{1}{\tau_e}$). However, this leads to images where a transition in the state of the display takes place during an exposure. This leads to a data assignment problem as the state transition with one LED turning OFF and another turning on during an exposure may be incorrectly detected as a state with both LEDs ON. In order to remove such frames from the data association we represent the two least significant bits of our LED clock by 4 LEDs (see above). When 2 of them appear ON in a camera frame there is only one possible transition that can cause this, and the time in the middle between the two states is estimated for the image. In addition, frames showing the transitions between LEDs 0 and 3 are skipped since these imply a change in state of one of the more significant bits, whose observed state might then not be distinguishable from state transitions. The so encoded number on the LED display in any given camera frame is calculated in an automated manner using a simple computer vision blob detector algorithm. This allows us to align the camera timestamps with the clock timestamps with an ambiguity of the period of the entire LED-clock (the duration after which it repeats itself). In order to resolve this ambiguity, a box prior for the average total processing and transmission delay per image is required with an uncertainty smaller than the period of the LED clock. This poses no problem because the period is adjustable through the clock frequency and one can start with a low frequency e.g. 500 Hz which requires the uncertainty to only be below $64/500$ s $= 128$ ms. One pass with this frequency reduces the uncertainty to 2 ms. Such a prior is good enough for our goal frequency of 10 kHz, which can be used in a next pass.

c) Circuit: The schematic of the circuit driving the LED display is too much detail for the paper. But it is very simple to design once the necessary LED pattern is known. For our prototype we used two 74HCT191, 4-bit counters chained together to an 8-bit counter, a 74LS42 to demultiplex the two least significant bits, and a HCF4069 to invert the output signals of the 74LS42. Additionally we use one BC547 transistor per LED to amplify the output current of the digital ICs and resistors to adjust the current through the transistors and LEDs, and some capacitors to stabilize the signals. One input of the RasPi is connected to the 7-th bit of the counter to notify about newly reached "0" display states.

d) Accuracy: Our approach limits the accuracy of the estimated image timestamps to within the exposure time of the camera. Hence, to obtain accurate measurements, the exposure time should be set to a small value and the LEDs chosen should be bright and close enough to be clearly visible in the image. The LED clock's base frequency (rate of a single clock increment) should be about the inverse of the camera's exposure time. A higher clock frequency would observe the average effect of multiple LED transitions during one exposure and a lower frequency limits the accuracy achievable with a single image. For our setup we set the

camera exposure time to 0.1 ms and the LED clock frequency to 10 kHz leading to an absolute error for each image timestamp to be within $[-0.05, 0.05]$ ms.

B. Precision time protocol (PTP)

The Precision time protocol (PTP, [7]) is a protocol for two-way synchronization between computers or other network nodes and from the user perspective similar to NTP. In contrast to NTP it is designed for local area networks and aims at much higher accuracy. Up to few nanoseconds when used with hardware support in the network devices. Even without any hardware support it typically maintains synchronization within a few microseconds. To synchronize the main PC with the RasPi we used ptpd[3] without using any hardware timestamping because the RasPi's Ethernet adapter does not support hardware timestamping.

C. One way clock translation

In order to exploit the *device time* (DT) measured by clocks within the sensors for sensor fusion or control it is necessary to translate it into CPU-time (CT). The same is necessary in order to use the calibration devices presented in this paper because they measure the sensor activity effectively in the CT. This is typically done by a suitable filter / batch algorithm fusing the receive CT-stamps with the DT-stamps in the received packages. Therefore we are referring to the resulting time with *translated device time* (TDT). To filter the DT-stamps we used our free software implementation[4] of the convex hull algorithm presented in [10]. This is identical to what was used in [1] and equivalent to the TICSync implementation used in [4] with the difference that the latter's authors are using an estimator switching periodically between two of the convex hull filter, which get reseted whenever their turn is over to prevent relying on too old data and therefore be more resilient against long term drifts. For our experiments we also use a switching version of the convex hull algorithm. To retrieve the offset to this TDT with respect to the CT (it is always lagging behind, roughly by the minimal unknown transport delay) is typically the goal of temporal calibration.

With the devices presented here it is possible to go much further due to their high rate high accuracy nature. They enable the user to monitor the performance of the hardware clock filter itself on data generated from the real target hardware clock. This is useful for tuning, developing and debugging hardware clock filters directly for a given sensor possibly taking relevant influences such as sensor temperature into account. These tasks are typically hard problems unless one is using hardware synchronization, which we assume to be no option.

In Section IV we are going to show two versions of TDT for the same raw data for the reader to better appreciate the problem.

[4]Available as part of `https://github.com/ethz-asl/cuckoo_time_translator/`

IV. Experiments and Results

A. Setup

We evaluate our method using the following sensors:

- two different Sick LMS151 LIDARS spinning at 50 Hz using a 100 MBit Ethernet connection,
- two Hokuyo LIDARs, UTM-30LX (USB2; 2 hubs) and UTM-30LX-EW (100 MBit Ethernet),
- a Point Grey BB2-08S2C-25 (Firewire; 1 hub),
- a Point Grey Chameleon3 CM3-U3-13Y3C (USB3; 1 hub).

We recorded data from these sensors while estimating the time until the measurements are received on the main PC. All sensors connected via Ethernet communicate through two gigabit Ethernet switches with the PC. All sensors are connected and running in parallel, together with a Velodyne 32E and one more LMS151 within a complex robotic system. However, only one sensor is analyzed at a time. Each LIDAR sensor is recorded for 60 s and each camera for 20 s and the estimated arrival delays and offsets to their TDT are presented. The Chameleon3's strobe signal and the UTM-30LX's "synchronous output" are additionally connected to GPIO ports of the RasPi and the timestamps of their rising edges are recorded in parallel to provide ground truth to validate our methods.

We use ROS [9] as the middleware to interface with the sensors. To our surprise all four required open source ROS-drivers lacked the support for DT-stamps. In order to be able to present the hardware clock offsets as part of our analysis, these drivers were extended to allow the hardware clocks to be used. We made this work also publicly available[1].

B. Evaluation

All the figures in the following section follow a common layout:

- The x-axis reflects the CT of the plotted events shifted such that the first measurement is always at zero.
- The y-axis is a time offset between two CT-stamps associated with a common event, of which the reference time is always a time determined with one of the calibration devices by the synchronized RasPi.
- Red points correspond to the overall transport and processing delay of a measurement package (camera image / LIDAR ranges), i.e. the CT of arrival at the main PC (*receive time*) of a sensor measurement minus the corresponding CT of the *first* measurement in the data package.
- Green points correspond to offsets of TDT-stamps, i.e. the one way TDT minus the CT of the corresponding detection.
- Optional blue points correspond to the offset of hardware signals coming directly from the sensor (e.g. strobe) to the RasPi for validation of our method.

C. LIDAR-to-CPU

For the LIDAR-to-CPU temporal calibration experiments we use the *LIDAR beam detector* device to generate timestamps in CT (by the means of PTP synchronization between

the main PC and RasPi) for the detection of LIDAR pulses on the photo diode. For all the experiments one event is measured per revolution of the spinning sensor as one photo diode is used and only the first detection per revolution is recorded (based on the roughly known sensor period)

Figure 4 shows results for the UTM-30LX and the UTM-30LX-EW, each spinning at 40 Hz. The specific pattern of the TDT-stamps (green) is caused by the remainder of the sensor's revolution time modulo the milliseconds counted by their internal clocks. The UTM-30LX data indicates a very stable revolution period. It yields almost a fixed fraction of a millisecond that it looses every revolution until it has lost an $\epsilon > 0$ more than a millisecond, which corresponds to only ϵ — modulo the one millisecond resolution. The high stability of the revolution period can be exploited to get timestamps of lower variance than a naive translation of the low precision DT, as suggested by [1]. A very simple way to do this is to use the revolution counter as a "sensor clock" instead of its timestamps. This clock's "time" can be translated with the same type of algorithms into CT. We show in Figure 4 (black) the offsets of the revolution counter after translating it with the same convex Hull algorithm we use for the green plots. This method is not reliable enough for real world application but it yields a good way to asses the stability of the revolution time. The UTM-30LX-EM shows similar effects, with the differences coming from a less stable revolution period.

To validate the performance of the LIDAR beam detector we retrieve additionally CT-stamps for the rising edge of the UTM-30LX's "synchronous output" (also in Figure 4, blue). This demonstrates high precision. However, the mean (2.607 ms) only roughly corresponds to the 2.75 ms delay specified by the manufacturer (as given in [1]). Because of this significant discrepancy, discrepancy, we must rely upon the oscilloscope (Figure 2) and the corresponding camera test in Figure 6 for the accuracy. Furthermore, we found that this delay changes over time between 2.56 and 2.62 ms, qualitatively corresponding to the drift in rotation frequency from 40.2 to 39.9 Hz. This indicates that the specification of 2.75 ms is indeed inaccurate.

Figure 5 shows the result for 20 s of data from the LMS151(b), spinning at 50 Hz [5]. These sensors emit two timestamps from the same clock with each package. One timestamp for the measurement start and one for the transmission start. The latter can be used to improve the clock translation as it happens very close to the physical receive time. Using this improved translation for the start timestamp yields better accuracy and a CT that is in deed close to the actual measurement start, without relying on assumptions about the measurement and processing duration as can be seen in Figure 5 (green). The remaining negative offset comes from the fact that according to the specification, the start timestamp is taken internally at 14 degree before the actual measurement start. This corresponds to -0.77 ms, which is very close to the mean of the translated start time (-0.71). The remaining difference is probably due to the random 1 ms jumping of the green signal. To our great surprise we had to assume here that the start-timestamp delivered along with the measurements from one measurement phase actually marks the beginning of the next measurement phase[6].

Statistical data for all these datasets is presented in Table IV-C. Each is based on 60 s of data per LIDAR, of which only the beginning may be plotted in the corresponding figures for the sake of readability. The two numbers for the receive latency's mean correspond to first and last measurement of one package. Both are inferred based on the measurement for one beam, its index, and the duration of a full measurement phase. Please note that all these numbers will not translate well to other setups and systems because the sensors are only a part of the communication. This holds in particular for the means. Furthermore, they depend on the revolution period, which typically drifts significantly over time and possibly depends on the motion / alignment of the sensor with respect to gravity. Additionally, for the Hokuyo sensors it depends on how the internal bandwidth limit[7] is overcome: skip revolutions or restrict the measurement angles range. We use the latter, which also leads to smaller packages that typically have less random transport delays than larger packages. Nevertheless, comparing the different LIDARs, it becomes apparent that the two LMS151 are much quicker at delivering the data through Ethernet than the UTM-30LX-EW (about 7 ms vs. 14 ms minimal latency). But at the same time their delay is less predictable (0.14 ms vs. 0.04 ms). The UTM-30LX connected with USB2 is even quicker (minimal latency mean 3.6 ms). However on the USB2 bus we observed quite frequent and rather severe outliers in the latencies, up to ~ 10 ms. Without specialized DT translators all presented LIDARs have more accurate receive time. However, a) more recent / expensive models are likely to have better DT-stamps (e.g. the Velodyne HDL-32E, whose analysis is beyond the scope of this paper), and b) in applications where larger outliers must be avoided it might still be better to utilize the DT-stamps as they can be more reliable, especially in the case of a dedicated data connection.

Our results clearly indicate that to analyze the performance of a LIDAR's hardware clock and to tune, develop and evaluate new filters for it, the LIDAR beam detector is a very useful device because it provides the user with the exact feedback needed: a reference timestamp to compare with for a real device.

D. Camera to CPU

For Camera-to-CPU temporal calibration experiments we use the LED Signaler device to generate timestamps in CT (by means of RasPi-CPU PTP synchronization) of each state of the LED display. We obtain a delay or offset for each image coming from the free running camera (Chameleon3

[5]The corresponding plot for the LMS151(a) is not included because it is very similar including the very surprising hardware clock behavior (see caption).

[6]Assumed it to mark the beginning of the same data the transmission of a package containing 15 ms measurements would happen < 3 ms after the first measurement.

[7]The sensors are not able to deliver at full rate over the full angular range.

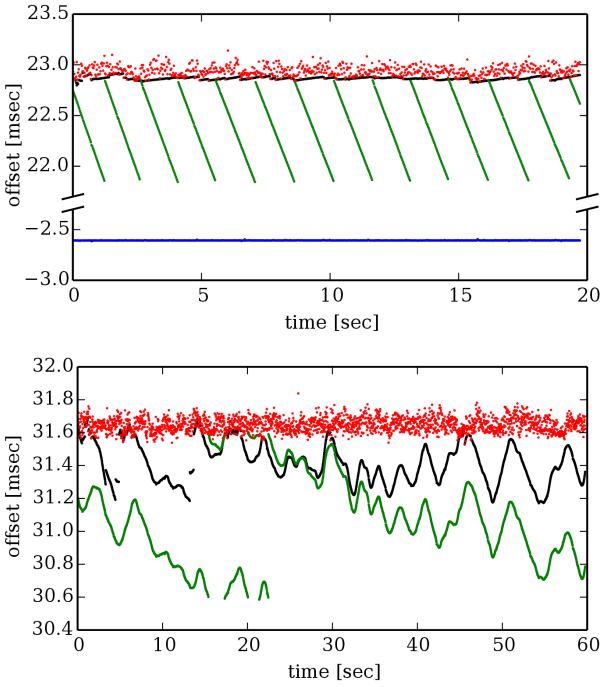

Fig. 4. Time offsets for the UTM-30LX (top) and UTM-30LX-EW (bottom). The latency (red) shows the typical pattern of random delays caused on the USB bus and in the OS kernel. The translated hardware clock offset (green) nicely shows the effect of DT taken with too low precision (1 ms; less precise than the revolution period). This lack of precision makes it less accurate than the receive time (SD $\simeq 50\,\mu$s) when used with a simple hardware clock translator (green, SD $\simeq 290\,\mu$). However, it is possible to improve on this by exploiting the precision: black (SD $\simeq 17\,\mu$). The UTM-30LX-EW's corresponding plot (bottom, black) shows the limitations of this "trick". It's higher variance (SD $\simeq 120\,\mu$s) indicates that this sensor's revolution period is less stable. The delay from the rising edge of the "synchronous output" of the UTM-30LX until the first measurement as detected by the photo diode (blue) shows very low variance (SD $\simeq 1\,\mu$s). This demonstrates the high precision of our method. It is not close to 0 because it is not synchronized with the first measurement but with a specific angle before it. Please note the break in the y-axis of the plot.

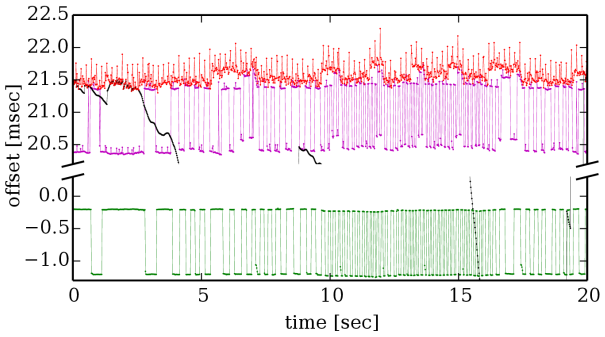

Fig. 5. The LMS151(b) receive latency's (red) show the expected random delays (mean $\simeq 21.6$ ms and SD $\simeq 0.14$ ms). The TDT (green) shows very surprising behaviour. It seems to jump between two hardware clocks that are almost precisely 1 ms apart. Each of these apparent clocks is very precise ($< 10\,\mu$s SD between the jumps; the DT resolution is $1\,\mu$s). Because of these unpredictable jumps the overall randomness of the DT-stamps has a standard deviation of approximately 0.5 ms, which is significantly larger than the SD. of the random transport delays. Using the revolution counter as a clock (black) is no alternative. Instead it reveals significant short term instability of the rotation period. The magenta plot shows the translated transmission timestamp. Its existence allows the start timestamp (green) translated with the same model to be close to the actual measurement start. Please note the break in the y-axis of the plot.

Sensor	receive latency		trans. hw. clock off.	
	mean [ms]	SD [ms]	mean [ms]	SD [ms]
UTM-30LX	$21.9 - 3.9$	0.085	21.3	0.289
...-EW	$31.6 - 13.6$	0.042	31.1	0.257
LMS151(a)	$21.6 - 6.6$	0.174	-0.7	0.498
LMS151(b)	$21.6 - 6.6$	0.142	-0.7	0.491
Chameleon3	13.8	0.712	11.6	0.020
Bumblebee2	51.7	0.116	51.3	0.023

TABLE I

LATENCY AND OFFSET STATISTICS FOR THE TESTED SENSORS

or Bumblebee2) by detecting the state of the LED display in the image and associating it to a corresponding state of the display as described in section III-A.2. The camera exposure times are set to 0.1 ms, which should limit absolute errors to within $[-0.05, 0.05]$ ms for each measurement. Thanks to the short exposure time normal office lighting does not pose any problem for the procedure. Typically only the LEDs are bright enough to be visible in the images[8].

Figure 6 shows the delay and offset plots for a 20 s dataset collected using the Chameleon3. In addition to recording the image arrival and TDT-stamps, we also recorded the timestamp for the strobe signal for each image directly in RasPi time. As seen from the figure, the **difference** between estimated image timestamps using our method and the strobe signals (blue) has a mean of only $10\,\mu$s (SD $19\,\mu$s), clearly validating our method. Furthermore, we can observe that the variance of the image arrival timestamps (red) over the dataset is much greater than the translated image DT-stamps (green), reflecting the variable delay in image transmission. Table IV-C shows statistics providing a comparison between image arrival timestamps and TDT-stamps for both cameras studied during this work. Figure 7 shows similar plots for the delays and offsets of the Bumblebee2 camera but without strobe signal. The offsets plotted in purple show the typical bad performance of a freshly started convex hull filter. Only after some time it obtains steady values for the delays (green), hence the particular switching concept described above, which allows a filter to mature before being used. Long term clock drifts (acceleration / deceleration) are not captured by the employed convex hull filter, and hence a gradual drift in the offset values can sometimes be observed in longer datasets, such as the one shown in Figure 8. Against this problem the reset part of the switching concept is a good remedy. Switching was disabled for this last experiment.

Since our method provides an offset almost for each image in contrast to batch processing based approaches [1], [2], it is a useful tool to study such effects for a variety of sensors.

V. CONCLUSIONS

We presented a novel approach for temporal calibration of LIDARs and cameras with respect to a connected computer's system time that offers high accuracies at a high rate. And we showed with 5 different sensors that it can easily provide

[8]The LED display image in Figure 3 was taken in a bright office and behind the LEDs there was white plastic.

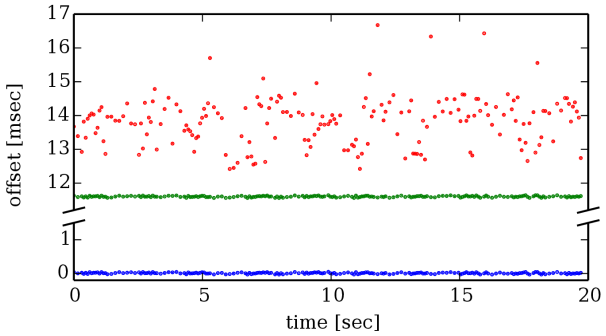

Fig. 6. Chameleon3 time offsets estimated using the LED signaler. As hoped, the strobe signal time offsets (blue) have almost zero mean (10 μs; SD 19 μs; both below exposure time, 0.1 ms). Hence they validate the method. The standard deviation of the receive latencies (red; SD 0.7 ms) is observed to be much higher than the offsets of the translated image timestamps from the camera hardware clock (green; SD 20 μs). Data collected over 20 s, 192 images (+66 skipped). Please note the break in the y-axis of the plot.

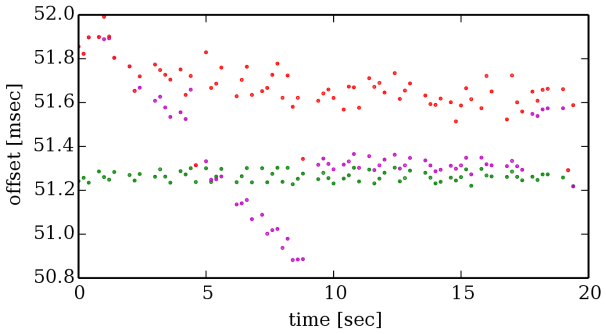

Fig. 7. Bumblebee2 time offsets estimated using our method for received (red), and hardware (green) timestamps. The statistics for this 20 sec dataset are tabulated in Table IV-C. The offsets shown in purple are also obtained using DT-stamps. However, they are translated to CT using a filter (Section III-C) initiated at the beginning of the dataset as opposed to the green offsets, which are translated using a filter which has been running for a while. This highlights the effect of using one way clock translation, to learn the higher order terms of clock skew and offset for the sensor hardware clock with respect to the CPU clock.

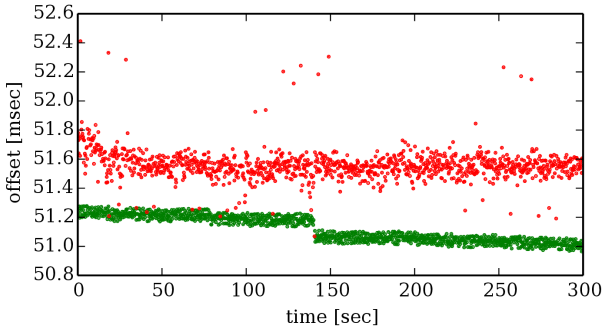

Fig. 8. Bumblebee2 time offsets over a longer, 300 s dataset (1124 images, +374 skipped due to LED transitions). The red points show the offsets of the image arrival timestamps. One can also see the slow, drift of the camera hardware clock with respect to the CPU clock over time (green). This is due to imperfection of the one way clock translation applied here (green; convex hull without switching). Our setup is precise enough to highlight these effects, and provides a stepping stone towards addressing them in application scenarios.

insights into the nature of the sensor delays that are difficult to retrieve otherwise. The presented solution promises to be relevant to all applications that require low timing uncertainty when utilizing these sensors, such as high speed state estimation and control. The proposed methodology is particularly useful in the detailed analysis of the temporal offsets to leverage optimal hardware time filters or investigate causes for random transfer delays. We expect the contribution of this work to be most useful for robotic researchers and industrial users relying on these sensor types without having the resources for custom hardware development or hardware synchronization. To validate our methods we compared our results with the hardware synchronization signals of a camera and a LIDAR and measured the delays of the LIDAR beam detector using an artificial IR source to simulate laser pulses from LIDARs.

ACKNOWLEDGMENTS

This work was supported by the European Union's Seventh Framework Programme for research, technological development and demonstration under the EUROPA2 project No. FP7-610603, the European Union's Horizon 2020 research and innovation programme under grant agreement No 644227 (FLOURISH) and from the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0029. The authors would also like to thank Jörn Rehder for the very helpful discussions and Mark Pfeiffer for the code contributions!

REFERENCES

[1] J. Rehder, R. Siegwart, and P. Furgale, "A general approach to spatiotemporal calibration in multisensor systems," *IEEE Transactions on Robotics*, vol. 32, no. 2, pp. 383–398, 2016.

[2] Z. Taylor and J. Nieto, "Motion-based calibration of multimodal sensor extrinsics and timing offset estimation," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1215–1229, 2016.

[3] J. Kelly and G. S. Sukhatme, "A general framework for temporal calibration of multiple proprioceptive and exteroceptive sensors," in *Experimental Robotics*, pp. 195–209, Springer, 2014.

[4] A. English, P. Ross, D. Ball, B. Upcroft, and P. Corke, "Triggersync: A time synchronisation tool," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 6220–6226, IEEE, 2015.

[5] I. Sa, S. Hrabar, and P. Corke, "Outdoor flight testing of a pole inspection uav incorporating high-speed vision," in *Field and Service Robotics*, pp. 107–121, Springer, 2015.

[6] A. Harrison and P. Newman, "Ticsync: Knowing when things happened," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 356–363, IEEE, 2011.

[7] J. Eidson and K. Lee, "Ieee 1588 standard for a precision clock synchronization protocol for networked measurement and control systems," in *Sensors for Industry Conference, 2002. 2nd ISA/IEEE*, pp. 98–105, Ieee, 2002.

[8] J. Quesada, J. U. Llano, R. Sebastian, M. Castro, and E. Jacob, "Evaluation of clock synchronization methods for measurement and control using embedded linux sbcs," in *Remote Engineering and Virtual Instrumentation (REV), 2012 9th International Conference on*, pp. 1–7, IEEE, 2012.

[9] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, 2009.

[10] L. Zhang, Z. Liu, and C. H. Xia, "Clock synchronization algorithms for network measurements," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, pp. 160–169, IEEE, 2002.