Unleashing the Power of LED-to-Camera Communications for IoT Devices

Alexis Duque, Razvan Stanica, Hervé Rivano, Adrien Desportes

Unleashing the power of LED-to-camera communications for IoT devices

Alexis Duque
Rtone, Lyon, France
Univ Lyon, INSA Lyon, Inria, CITI
F-69621, Villeurbanne, France
alexis.duque@insa-lyon.fr

Razvan Stanica
Univ Lyon, INSA Lyon, Inria, CITI
F-69621, Villeurbanne, France

Herve Rivano
Univ Lyon, Inria, INSA Lyon, CITI
F-69621, Villeurbanne, France

Adrien Desportes
Rtone, Lyon, France

ABSTRACT

In this paper, we propose a line of sight LED-to-camera communication system based on a small color LED and a smartphone. We design a cheap prototype as proof of concept of a near communication framework for the Internet of Things. We evaluate the system performance, its reliability and the environment influence on the LED-to-camera communication, highlighting that a throughput of a few kilobits per second is reachable. Finally, we design a real time, efficient LED detection and image processing algorithm to leverage the specific issues encountered in the system.

1. INTRODUCTION

With the rise of the Internet of Things, consumer electronics products, which yesterday were single function, tend to be smarter and connected to the user, through his smartphone. However, providing wireless connectivity with Bluetooth Low Energy (BLE) or WiFi means adding an extra radio chip, increasing the object size and price.

This kind of hardware modification is not without impact for the manufacturers: even if the radio chip cost in negligible for a single unit, it may become huge when millions of products are sold.

Besides, many of these products already have a microcontroller and several light emitting diodes (LED), which are the only requirement to enable visible light communication [1]. To further reduce the cost of the system, previous works demonstrate the possibility of receiving information through visible light using an unmodified smartphone thanks to its camera [2]. In fact, for a few years, researchers addressed light-to-camera communication issues in different scenarios. Lee et al. [3] proposal targets line-of-sight (LOS) LED-to-camera communication by exploiting the rolling shutter effect to transmit information between a lighting LED and a smartphone. On the other hand, in [4], the authors investigate data transmission in non-LOS mode applied to indoor localization. Kuo et al. [5] use a different approach to indoor positioning using the coordinates of several ceiling LEDs embedded in a picture. The LEDs are transmitting their identifier, and the receiver position is computed by applying trigonometric and optic laws.

While these previous studies demonstrated the feasibility of the approach, they use in their performance evaluation commercial lighting LED bulbs. Instead, our focus is on small LEDs, already integrated in many objects we use daily. Even if the size of these LEDs highly reduces the throughput and the range of this kind of communication, potential applications exist and VLC can be a cheap alternative to traditional radio communication. This is the case when we want to read a small payload coming from a sensor, such as a battery level or a temperature, or occasionally configure an equipment. Infrared (IR) communication is already used and is often the most obvious choice when we consider wireless, non-radio, near communication. However, a dedicated LED, suited for IR emission is required on the products, and most of the smartphones do not embed an IR receiver, whereas they all have a camera.

Therefore, in this paper, we first design and evaluate a VLC system between a small color LED, that can be found in most embedded system or consumer electronic, and an unmodified smartphone. A similar evaluation is proposed in [6] in the case of connected toys, but using a different, lower throughput communication technique (the aliasing method in [6], the rolling shutter effect in our study).

We propose an experimental evaluation of the environment impact on this system, which targets near communication scenarios. The obtained results shows that a throughput of nearly 2 kb/s in ordinary illumination condition is achievable. As the main contribution of this study, we develop an original decoding algorithm suitable for LOS LED-to-camera communication which demonstrates for the first time real time LED-to-camera communication using the rolling shutter effect.

We begin by describing the proposed communication system in Sec. 2 and evaluating its performance in Sec. 3. The impact of different environmental factors is discussed in Sec. 4. Finally, the proposed decoding algorithm and its performance are presented in Sec. 5, before concluding the paper in Sec. 6.

2. SYSTEM DESCRIPTION

As proof of concept, we design a cheap and small printed circuit board (PCB) embedding only a surface mounted RGB LED, a Micro-Controller Unit (MCU), and a temperature sensor, to broadcast the ambient temperature through the light. The signal is received by the smartphone camera and decoded by an Android application.

To evaluate the system, we use a development board to

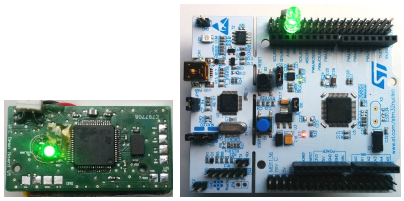

Figure 1: On the left, our proof of concept prototype of VLC temperature sensor. On the right, the Nucleo STM32L0 development board

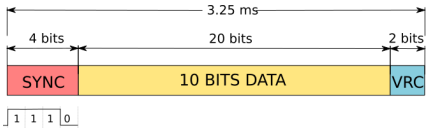

Figure 2: Packet format and its duration assuming a 8KHz clock rate

interface the MCU with more convenience, and so, change without hardware modification the LED type, the General Purpose Input/Output (GPIO) mapping or the firmware implementation. This is depicted in Fig. 1.

2.1 Emitter

The chosen micro-controller is a low cost and low power STM32L051, from ST Microelectronics, and its Nucleo development board. The core is a Cortex M0+, running up to 32MHz, with 32Ko Flash and 8Ko RAM. To get a better clock accuracy and avoid clock bias due to the temperature, we use an 8MHz high speed external crystal oscillator as the clock source and make the core run at this speed.

As proposed in [2], [3], the LED signal is modulated using the simple On-Off-Keying (OOK) modulation scheme. We consider a clock-rate varying from 2KHz to 10KHz, which is a suitable bandwidth for Optical Camera Communication (OCC) using the rolling shutter effect [7]. To ensure a balanced duty cycle signal and avoid any flickering effect, we use the Manchester coding proposed in [2], [7], [8]. The transmission scenario chosen is simple, and summarized in Fig. 2: we transmit a 10 bits sensor reading, encoded using the Manchester coding and resulting in 20 chips. This Run Length Limited (RLL) code ensures the signal is DC balanced and will not cause flickering. In Fig. 2, we also show that four synchronization bits (0111) are prepended to solve the synchronization issue on the receiver part. Finally, two integrity control bits are append, resulting in a 26 chips packet.

2.2 Receiver

On the receiver side, we use a LG Nexus 5 smartphone running Android Marshmallow version number 6.0.1. It has a Qualcomm Snapdragon 800 quad-core CPU 2,26 GHz CPU and 2Go RAM. Its 8 megapixels 1080p 1/3.2" CMOS sensor with 1.4 μm pixel size can capture up to to 30 frames per second and supports advanced imaging application provided by the Camera2 API.

We have developed an Android application that sets up the camera parameters to observe the rolling shutter effect produced by the modulated LED. For that, based on [7], we set a very short exposure time and an increased sensor sensitivity, respectively to 100 μs and ISO 10000. As soon as a new frame is available, the application creates and starts a new thread to process and decode the picture on the background. This processing consists of a LED detection algorithm that extracts the region of interest (ROI) before performing signal processing methods to retrieve the transmitted information. This step in described more precisely in Sec. 5.

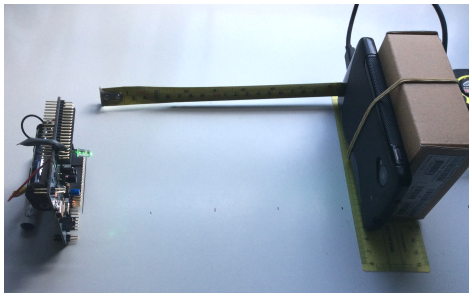

Figure 3: On the left, the LED driven by the Nucleo development board. On the right the Nexus 5 smartphone connected to ADB on a laptop.

3. EVALUATION

We first evaluate the system performance as a function of the distance between the LED and the smartphone. We set the emitter clock rate to 8KHz and place it in standard indoor illumination conditions, that is about 650 lux. We make the distance vary from 0 to the farthest one where we can detect a signal, without using a zoom or an additional lens. To avoid any external noise and ensure constant experimental conditions, we control the scene illumination with a light meter, and fix the smartphone position (Fig. 3). We repeat each measure on different days, at the same place and same conditions, to record enough data to consider them sterling.

The emitter broadcasts continuously 50 different packets built according to Sec. 2.1. We compute the throughput as the number of received information bits per second, error free, including duplicated packets. The goodput is calculated removing duplicates until the smartphone receives all the 50 packets.

The lower curve in Fig. 4 shows the goodput achieved by the system. This is 1550 bit/s at 5 cm, but is divided by two at 15 cm and smoothly decreases with the distance up to 40cm. On the other hand, the throughput, obtained before removing duplicate packets, is 1983 bit/s at 5cm, corresponding to a duplicate packets ratio of about 25%. This lets us the possibility of further improvements by implementing a more advanced loss mitigation method.

In OCC, and especially in the LOS mode, the size of the ROI produced by the rolling shutter effect is the primary limiting factor of performance, compared to the camera frame rate or the inter-frame delay. In fact, each pixel row of the CMOS sensor not exposed to LED light induces a signal loss. Since we use small color LED, the size of the ROI, and so, the number of symbols per frame, decrease quickly over distance (Fig. 5). This is the reason why we chose to send tiny pieces of data. Even if this is less efficient than bigger packets when the receiver is close to the

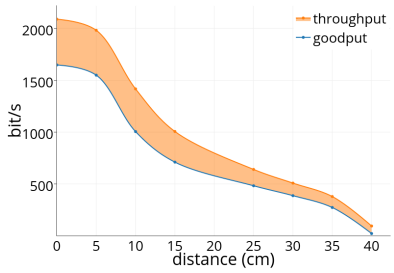

Figure 4: Throughput and goodput

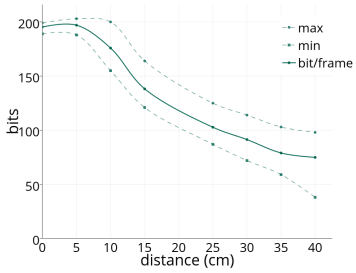

Figure 5: Number of bit per image

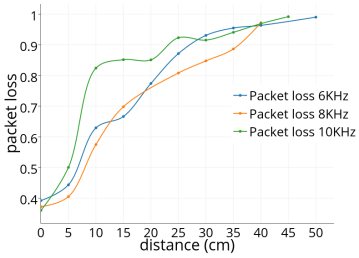

Figure 6: Packet loss

emitter, as it introduces overhead, it becomes an advantage when the ROI is small.

We also compute the packet loss ratio (Fig. 6) dividing the number of transmitted packets per the received ones, error free. The results show that 70% of packets were lost at 15cm, outlining the importance of adding a redundancy mechanism.

4. ENVIRONMENT IMPACT

In this section, we propose an evaluation of the environment impact on the performances of LED-to-camera communication using small color LED. We study the effect of the ambient light and of the angle between the camera and the transmitting LED.

4.1 Noise

The interference of the ambient light, coming from the indoor ceiling lights or the sun, is a major issue in VLC, and even more using low power LEDs that are not used for lighting purposes. In rolling shutter camera communication, this interference reflects in a reduced difference between dark and bright stripes on the picture, which correspond to the 0 and 1 logic states, and a reduced ROI. Moreover, the resulting problem only roughly depends on the sensor sensitivity and the exposure time; therefore, during this evaluation, we keep them constant using parameters defined in Sec. 2.2.

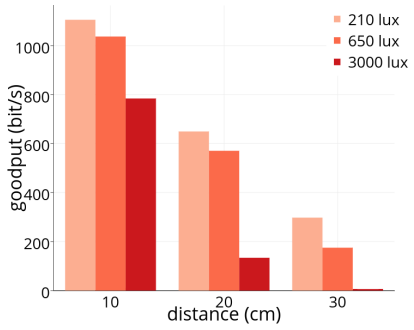

Figure 7: Goodput in different illumination condition.

We evaluate the performances of our system in different conventional illumination condition, indoor and outdoor: *i)* in a room, normally lighted but not exposed to the sun through a window (210 lux); *ii)* on a laboratory work table, near a window and illuminated with neon lights (650 lux); *iii)* outdoor, during a very cloudy day (3000 lux), and *iv)* outdoor, during a sunny day (40000 lux).

In 40000 lux conditions, the signal is totally lost, even when the receiver and emitter are very close. Results for the other cases are depicted in Fig. 7, showing that the system is robust against indoor lighting. Sunlight on the other hand has a stronger impact, but the results show that the throughput stays correct, only 25% less than in a room at 10 centimeters, if we avoid the direct sunlight. The robustness might be further improved by adding an automatic sensor sensitivity adjustment mechanism.

4.2 User and angle impact

During our experiments, we noticed the user can have a significant impact on the system performance. This can be observed by comparing the results obtained when the smartphone was held by the user and those obtained when using a fixed support. As Fig. 8 shows, the average throughput is the same in both cases, but a wider distribution can be noticed for a handheld device. To understand the factor that causes this phenomenon, we make the LED direction and its angle with the camera change from -10° to +10°, when at 0° they are perfectly aligned with both plans.

Figure 8 brings out the sharp dropping of the throughput with the variation of the angle. In fact, due to the directivity of the LED used (C503B-GAN), which is 30°, the angle has a huge influence on the ROI size on the picture. This can be avoided by choosing a LED with a large angle of view, but the side effect is that the light reflection on the camera lens will be weaker.

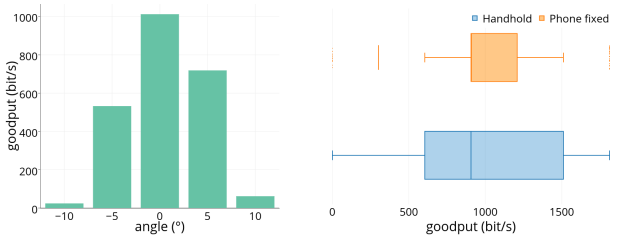

Figure 8: On the left, the impact of the angle between the LED and the camera at 10 cm. On the right, the throughput when he user is holding the smartphone compared to a mechanical support

5. REAL TIME COMPUTATION

Computation time is an important metric in OCC, and even more when a real time transmission is required. For

real time operations, all the image processing operations must be completed in less than 30ms, which is the common camera frame rate. A higher processing duration would reduce the system throughput or introduce delay. Besides, loading the receiver with intense processing tasks will alter the capture rate regularity, and flood the Random Access Memory (RAM) of the smartphone.

Figure 9 shows the duration distribution of the whole decoding chain on the Nexus 5 smartphone. On the left part of the figure, we show that a naive implementation, respecting the state of the art approaches [7], does not cope with real time constraints. However, as the right part of the figure highlights, after modifying the receiver Android application, we can process each frame in real time, by detecting the LED and decoding the signal, on average, in 18ms. The rest of this section describes the steps we followed to achieve this major gain in computation time.

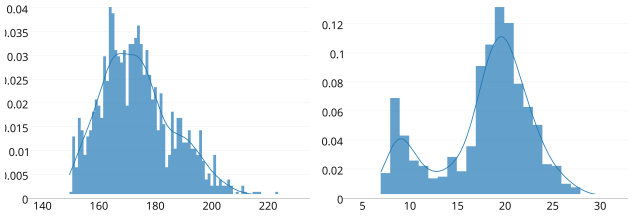

Figure 9: Algorithm duration. State of the art implementation (left) and optimized (right)

5.1 Algorithm presentation

We developed an image processing algorithm that can be decomposed into five steps:
Step 1. Image acquisition: handling incoming captures and preparing the buffer for next steps.
Step 2. ROI detection: finding the LED position on the picture and reducing the buffer length.
Step 3. Signal improvement and threshold computing: enhancing the signal to deal with inter symbol interferences and background noise.
Step 4. Thresholding and binarization: recovering the digital signal.
Step 5. Decoding and error checking: decapsulating the packets, decoding the Manchester chip and checking data integrity with vertical redundancy check (VRC).

As already explained, we first produced an implementation following the state of the art guidelines. This first implementation took up to 165ms processing time per picture, which is not an acceptable value, as it constrains the frame-rate to avoid the system overflow, forcing the Android scheduler to randomly kill threads or drop camera captures. Thus, we investigate possible improvements at different layers: pure algorithmic, software implementation on the Android Execution Runtime (ART), and the Android Camera2 framework.

5.2 Image format

Since the release of Android Lollipop (API 21) and the new Camera2 API, developers are now able to control the smartphone CMOS sensor and the camera subsystem without any modification in the lower layer of the operating system. Numerous parameters can be chosen that can affect the efficiency and the amount of processing performed by the Android system.

The most relevant of these parameters are the image size, format and its compression ratio. We tried different image sizes from 800x600 up to 3264x2448 pixels. Augmenting the resolution increases the rolling shutter stripes length, and so, reduce the inter-symbol interference error [9]. However, this also increases the matrix size to process. To deal with that, and unlike in [7] which prefers a larger image size, we choose an intermediary value of 1920 pixels width per 1080 height: this is about 2.5 smaller than the larger one, and provides sufficient precision regarding the symbol length.

Also, choosing the right color space makes the processing easier and affects the smartphone computing load. If a JPEG encoded RGB image representation has a lower memory footprint, it also requires more post-processing, due to color conversion and compression, which affects frame rate and the global system CPU and GPU load.

In contrast, RAW format contains the sensor value without any post-processing but is not exploitable as is, due to its size. To take advantage of this representation, we choose the YUV_420_888 uncompressed format as suggested in [7]. YUV representation has three channels, the luma (Y) and the chroma (U and Y plane contains the luminosity level while U and V represent the color information). As we modulate the LED light, only the Y plane is relevant for us, reducing the size of the required data.

5.3 Capture workflow

As soon as a new frame is available, the Android application UI thread invokes the `onCaptureCompleted` callback. However, if the UI Thread is overloaded or busy, the newer frame, and as consequence data, can be lost. That means we should take care to keep the CPU and memory offloaded. To face frame loss, we can allocate an Image buffer to queue a limited number of frames upstream the callback. We follow the API recommendation setting this buffer to five.

An `Image` object in Android contains a `ByteBuffer` for each plane, and metadata, such as the capture timestamp. Accessing and manipulating big sized Java objects can be memory costly and not efficient, particularly when they contain huge buffers, most of them useless. This is typically our cases, as U and V planes will be ignored.

As soon as we acquire a new image, we allocate a new bytes array with the same size as the Y plane `ByteBuffer`, and fill it using the `Java NIO ByteBuffer.get` method which generates optimized native code after Ahead-Of-Time (AOT) code compilation. We prefer moving to primitive type bytes array because read operations are notably faster.

Once this processing is done, we call the `Image.close` method to quickly release and free the memory.

Finally, we create a new Java `Thread`, passing the buffer and frame metadata to its constructor, and push it into the `ThreadPoolExecutor` set up at the application initialization.

Background processing on Android can be achieved in several ways: using Java `Thread` class implementing the `Runnable` interface, or the Android way using `AsyncTask`. Even if `AsyncTask` is generally preferred, we chose the Java Thread way. In fact, by implementing both solutions, our experiments showed that the processing is 8% slower, and with a larger distribution, when using AsyncTask (Fig. 10).

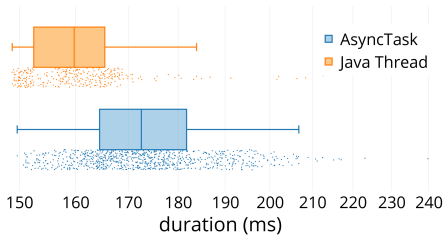

Figure 10: Total process duration distribution comparing AsyncTask and Thread

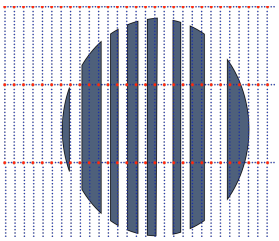

Figure 11: A small ROI and image pixels

5.4 ROI Detection

In LOS LED-to-camera communication, only small pieces of the whole image hold the signal, meaning that we need to search the coordinates of the LED in the picture. Despite being the most time consuming, this step is many times ignored in the literature when measuring the time required to recover the data [7]. Among the studies that considered this problem, [5] uses a computer vision-based technique, where an RGB image is converted to gray-scale, blurred, and filtered. This allows extracting contours and finding the minimum enclosing circle, but the method is computationally intensive and it can be run in real time on a smartphone. The problem is partially solved by [3], where the LED is localized on the picture only on the first frame, at the beginning of the transmission, implying that the receiver must stay motionless for the following frames. However, as discussed in Sec. 4.2, a user holding a smartphone performs involuntary moves and this will change the LED position on the picture, especially as we use small LEDs.

To leverage these issues, we implement an alternative method by avoiding costly computer vision algorithm. Two version of this ROI detection algorithm were tested. The first one is very basic and just looks for the brighter and larger region. We run through the whole bytes buffer of pixels luma intensity, starting by column and then by row. If the value is not 10 % less than the previously highest pixel, we keep it for further processing. To improve the execution time of this step, which loops over the 2073600 bytes allocated globally at the thread construction, we create a local bytes buffer variable `byte[] data` inside the looping function, and assign it to the global one. Pulling everything into local variables avoids the look-ups during memory access, loading them in the RAM Heap, and makes our function 10.75 percent faster.

Nonetheless, working on the whole 1920x1080 picture takes up to 160ms, which is not suitable to process 30 images per second in real-time. To further accelerate the process, we test only 1/3 rows and 1/20 rows (the red points in Fig.11) and skip the other pixels (the blue points). These values are chosen such as even the smallest symbol can be recovered. This allows us to cut the detection time to a few ms, while keeping a low LED misdetection probability.

5.5 Signal processing and thresholding

5.5.1 *Signal enhancement*

Blooming effect and interference mitigation in VLC have been studied in [10]. The proposed filtering methods proved to reduce the error rate in low SNR condition. However these solutions are not computationally efficient and their implementation on a smartphone has not been studied.

Our method to face the irregularity of the bright and dark stripes is straightforward: starting from the region of interest determined in Sec. 5.4, which fundamentally is a two-dimensional array, we average the luma value for each column, reducing it to a one-dimensional array. For performance consideration, we put these values into an integer array. In a first time, we considered using a new byte array, to reduce the memory impact and keep 8 bits representation. However, using 32 bits integers proved to be faster. The explanation is that the Nexus 5 processor has a 32-bit architecture, so the Android Runtime (ART) and its AOT does the operations using 32-bit instructions in each case. Hence, in the byte case, it executes extra instructions to convert the intermediate 32-bit value to a byte in each loop iteration. This makes performances worse in our arithmetic and compute intensive algorithm.

The cost of this operation depends on the ROI width (the column length), and averaging a lot of values can be useless or inefficient. So, after some experiments, we decide to limit it to 50 points, which is enough to obtain a proper signal without impacting the system performance. The result of this step is shown in Fig. 12.

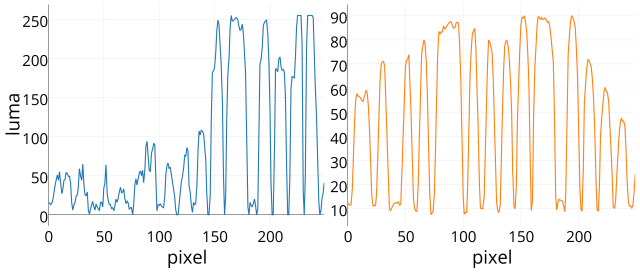

Figure 12: Comparison of the raw signal (left) after ROI averaging step (right)

5.5.2 *Thresholding*

We next determine the threshold level by computing the mean value of the array defined in Sec. 5.5.1, to binarize it. While we run through the array during the pixel binarization, we count how many successive 0 or 1 pixel will follow, in order to convert them to bit symbols. This is trivially done by dividing this counter by the number of consecutive pixels per symbol we expect, which depends on the transmitter clock rate, the image size and intrinsic characteristic of the CMOS sensor [9].

To avoid potential decoding errors and face the inconstant symbol size, we add an error correction mechanism based on the Manchester RLL code and packet format expected. In practice, this proves to be robust against a 1KHz clock bias on the emitter side.

5.6 Manchester decoding and VRC check

Since we now have a short character string of bit symbols, we can take advantage of Java String API, to easily decode symbols to data information.

Then, we compute the VRC and compare it with the received one. Finally, we convert Manchester symbols to data symbols and return the result to the Android activity.

5.7 Results

With these optimizations, the application can recover the data information in 18.4 ms on average, in various illumination conditions. Fig. 9 compares the duration distribution of the different algorithms versions, showing that this has been radically reduced. Also, even if the execution time may vary, the optimized version stays below 27 ms.

Fig. 13 summarizes the average execution time of the processing thread functions on the Nexus 5 smartphone. This shows the ROI detection step lasts the longest and takes on average 7.95 ms meaning that processing a single LED signal takes less than 9 ms. On average, signal enhancement takes 6.87 ms and thresholding 1.31 ms. Initialization, which essentially performs a buffer copy, and threshold computation both have a negligible duration. The final step, which decodes the binary data and check their integrity, lasts 2.18 ms on average.

These results outperform previous works, especially when considering that, unlike the state of the art solutions, we compute the LED position in each frame (about 30 times per second), adding robustness against user motion.

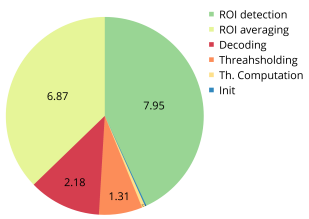

Figure 13: Steps duration repartition of the optimized algorithm

6. CONCLUSIONS

In this paper, we introduce an original LOS LED-to-camera communication system between a small colored LED and a smartphone. By taking advantage of the rolling shutter effect, the smartphone can receive 1550 bit/s of useful information at 5 cm in common indoor condition, and up to 40 cm with a decreasing throughput, which makes it suitable for several IoT use cases.

We also propose an efficient decoding algorithm, which can detect the LED position, process and decode the signal on average in 18.4 ms, for each frame, on a Nexus 5 non-rooted smartphone. Thus, this implementation is convenient for low latency indoor localization or real-time transmission with a moving receiver. Also, as the ROI detection is the longer step of the algorithm, scenarios with several transmitters can be envisaged, enabling MIMO transmission.

7. REFERENCES

[1] S. Schmid, G. Corbellini, S. Mangold, and T. Gross, "An LED-to-LED Visible Light Communication system with software-based synchronization", *Proc. IEEE Globecom Workshops*, 2012.

[2] C. Danakis, M. Afgani, G. Povey, I. Underwood, and H. Haas, "Using a CMOS camera sensor for visible light communication", *Proc. IEEE Globecom Workshops*, 2012.

[3] H.Y. Lee, H.M. Lin, Y.L. Wei, H.I. Wu, H.M. Tsai, and C.J. Lin, "RollingLight : Enabling Line-of-Sight Light-to-Camera Communications", *Proc. ACM Mobisys*, 2015.

[4] N. Rajagopal, P. Lazik, A. Rowe, X. Zhang, J. Duan, Y. Fu, and A. Shi, "Visual light landmarks for mobile devices", *Journal of Lightwave Technology*, 32(21):249–260, 2014.

[5] Y.S. Kuo, P. Pannuto, K.J. Hsiao, and P. Dutta, "Luxapose", *Proc. ACM MobiCom*, 2014.

[6] G. Corbellini, K. Aksit, S. Schmid, S. Mangold, and T. Gross, "Connecting networks of toys and smartphones with visible light communication", *IEEE Communications Magazine*, 52(7):72–78, 2014.

[7] D. Camps-Mur, J. Paradells-Aspas, and J. Ferrandiz-Lahuerta, "A reliable asynchronous protocol for VLC communications based on the rolling shutter effect", *Proc. IEEE Globecom*, 2015.

[8] T. Nguyen and Y.M. Jang, "High-speed asynchronous Optical Camera Communication using LED and rolling shutter camera", *Proc. IEEE ICUFN*, 2015.

[9] T.D. Do, "Analysis on Visible Light Communication using Rolling Shutter CMOS Sensor", *Proc. IEEE ICTC*, 2015.

[10] C.W. Chow, C.Y. Chen, and S.H. Chen, "Enhancement of Signal Performance in LED Visible Light Communications Using Mobile Phone Camera", *IEEE Photonics Journal*, 7(5):1–7, 2015.