Real-Time High-Accuracy 2D Localization with Structured Patterns

Lukas Hostettler[1], Ayberk Özgür[1], Séverin Lemaignan[1,2], Pierre Dillenbourg[1] and Francesco Mondada[2]

Abstract— **Building over algorithms previously developed for digital pens, this article introduces a novel 2D localization technique for mobile robots, based on simple printed patterns. This method combines high absolute accuracy (below 0.3mm), unlimited scalability, low computational requirements (the presented open-source implementation runs at above 45Hz on a low-cost microcontroller) and low cost (below €30 per device at prototype stage). The article first presents the underlying algorithms and localization pipeline. It then describes our reference hardware and software implementations, and finally evaluates the performance of this technique for mobile robots.**

I. Introduction

Localization, namely to know the pose of a mobile robot with respect to the environment, is a fundamental problem in robotics. Indoor localization in particular has to deal with specific constraints: besides the absence of a ubiquitous technology such as GPS, indoor localization typically requires relatively high precision (often at a centimeter-scale or below) and needs to deal with the presence of numerous occlusion sources (including dynamic ones such as humans). Other common concerns when deciding for a localization technique include the severity of deployment requirements in the environment (*e.g.* beacons, ceiling cameras), computational efficiency (*e.g.* real-time operation), power usage, cost, end-user friendliness for long-term deployment and scalability with respect to the number of robots.

Following these lines, we frame our work to the context of an indoor environment where one or many mobile robots move on their own on flat surfaces. External agents (*e.g.* humans) are expected to closely interact with the robots, including moving them around (thus defeating localization techniques relying only on dead-reckoning). This application context covers several typical scenarios encountered in indoors robotic research, from swarm robotics to human-robot interaction, while excluding situations involving non-grounded robots such as drones.

Mautz gives a comprehensive survey of absolute indoor localization methods in [5]; Table I summarizes the characteristics of the main ones, in regard to the application context introduced above. It appears that none of these techniques achieve affordable yet accurate localization of many devices on a surface, especially where occlusions due to robot handling are significant. [2] comes close to meeting these criteria, but the nature of this method makes real-time localization and device miniaturization difficult due to the capacitive sensor array requirement. [3, 4] describe structured optical patterns designed for 3D localization in open spaces; however, miniaturizing these patterns for our application would either significantly limit the size of the localization space or increase the minimum image resolution requirement and processing power need.

A promising approach to this problem involves the structured pattern described in the Anoto positioning technology ([6] and other related patents). It has been originally researched for localizing "intelligent pens" on paper. This article proposes to investigate the applicability of this method to robotics: it describes the underlying algorithms and presents our reference hardware and software (open-source) implementations. We also provide a detailed quantitative performance analysis (we reach an absolute positioning accuracy below 0.3mm for a hardware design below €30).

While this technique has intrinsic limitations (discussed at the end of the article), we believe that structured pattern localization offers **a unique combination of low cost, computational efficiency and high accuracy, holding a strong potential for indoor mobile robotics**.

II. Theory & Localization Pipeline

A. Encoding Principle

Structured patterns such as the Anoto pattern (found in [6]) are visual micro-dot patterns organized in a grid (Figure 10b). When decoded, they uniquely identify absolute positions while leaving only a small visual imprint on the printed surface. They are made of four symbols (*up*, *down*, *left* and *right*) corresponding to the relative position of each of the dots to the closest grid intersection. Each of the four symbols encodes two bits, one for x and one for y.

At the core of the encoding lie quasi De Bruijn sequences. Given an alphabet (in our case, $\{-1, 1\}$), such a sequence of order n contains every possible string of length n from this alphabet at most once. Every column of the pattern contains the same quasi De Bruijn sequence of order 6, called the *main number sequence*. This sequence has length 63 (the string 111111 is unused) and repeats itself in a cyclic manner, with different offsets for each column. Given two adjacent strings of length 6 from consecutive columns, the offset differences of these columns can be uniquely determined regardless of the row index.

The sequence of these offset differences, called the *primary difference sequence*, can be composed of numbers $d \in \{0, \ldots, 62\}$. However, only $d \in \{5, \ldots, 58\}$ are used in order to be able to decompose each d into its unique "digits" a_i under the "basis" equation $d = 5 + a_1 + 3 \cdot a_2 + 3^2 \cdot a_3 + 2 \cdot 3^2 \cdot a_4$, where $a_1 \in \{0, 1, 2\}$, $a_2 \in \{0, 1, 2\}$, $a_3 \in \{0, 1\}$ and $a_4 \in \{0, 1, 2\}$.

[1]Computer-Human Interaction in Learning and Instruction Laboratory, EPFL, 1015 Lausanne, Switzerland
[2]Robotic Systems Laboratory, EPFL, 1015 Lausanne, Switzerland
Email: firstname.lastname@epfl.ch

Method	**Cost** (per device)	**Accuracy**	**Deployment?**	**Occlusion Robustness**	**Scalability** (# of devices)	**CPU load** (on device)
Infrared light beacon	**Low**	**Sub-mm**	Beacons	None	Low	**Very low**
Laser scanner	High	**Few µm**	**None**	Moderate	Low	High
RF (Wi-Fi, Bluetooth, RFID, ...)	**Low**	Sub-m	Beacons	Moderate	Moderate	**Very low**
Ultra-wideband (UWB)	Potentially low	Few cm	Beacons/scanners	**High**	Moderate	**Very low**
Structured light (e.g Kinect)	Mid	Few cm	**None**	Moderate	Very low	High
Fiducial tag on device/motion capture	**Very low**	**Sub-mm** [1]	Camera(s)	Moderate	**High**	**None**
Deployed fiducial tags	**Low**	**Sub-mm**	Optical tags	Moderate	**High**	High
Deployed capacitive patterns [2]	**Low**	**Few mm**	Capacitive sheet	**Full**	**High**	**None**
Deployed optical patterns [3, 4]	**Low**	Few cm	Optical patterns	**High**	**High**	**Low**/High
This study	**Low**	**Sub-mm**	Paper sheet	**Full**	**High**	**Low**

TABLE I: Prominent absolute indoor localization methods found in the literature ([5] unless cited otherwise), compared to our method. Where used, *device* describes the object whose pose is recovered.

These a_i, sequenced in consecutive columns, also build quasi De Bruijn sequences – the *secondary number sequences* – from their respective alphabets, of order 5 and of lengths 236, 233, 31 and 241 respectively. Their lengths are chosen to be relatively prime so that the start of these cyclic sequences only line up after $236 \cdot 233 \cdot 31 \cdot 241 = 410\,815\,348$ positions. Conversely, given any number in $\{0, \ldots, 410\,815\,347\}$, there is a unique 4-tuple of positions in the secondary number sequences, corresponding to 4 unique a_i strings of length 5 thanks to the De Bruijn property. Every such number encodes an actual x position which we can read by observing 6 elements of 6 consecutive columns of the pattern (in practice, we use an 8×8 matrix that aids in finding the correct orientation and correcting errors, as described in the following sections).

The encoding for y occurs independently and in parallel over the rows instead of columns, using the second bit encoded by the positions of the dots.

Finally, all columns can be offset by $\{0, \ldots, 62\}$ without affecting the differences d (therefore not affecting the x position); this offset can only be detected by decoding the y position separately and by using this information to deduce how much the columns are shifted down. This number, called the *sector* of x, defines 63 different ways of laying out the columns. Identically, laying out rows defines 63 independent y sectors. The only limitation is that if a sector is juxtaposed to another one, the readings containing the boundary will not be decoded correctly due to the sector offset being interpreted as a difference d.

In total, $(410\,815\,348 \cdot 63)^2 \simeq 6.7 \cdot 10^{20}$ unique 2D positions can be therefore encoded. With the dot density used in this study (0.508mm on average between two dots), this allows absolute, sub-millimetric localization over an area approximately equivalent to $1/3$ of the surface area of the Earth. The interested reader can refer to [7] for a further account of the mathematics behind this encoding.

B. Localization Pipeline

Before performing the actual position decoding based on the principle explained above, captured video frames need to be processed to turn them from images to sequences of bits. This involves a number of steps organized into a pipeline illustrated in Figure 1. We hereafter present each of these steps, along with the main algorithms they are built on.

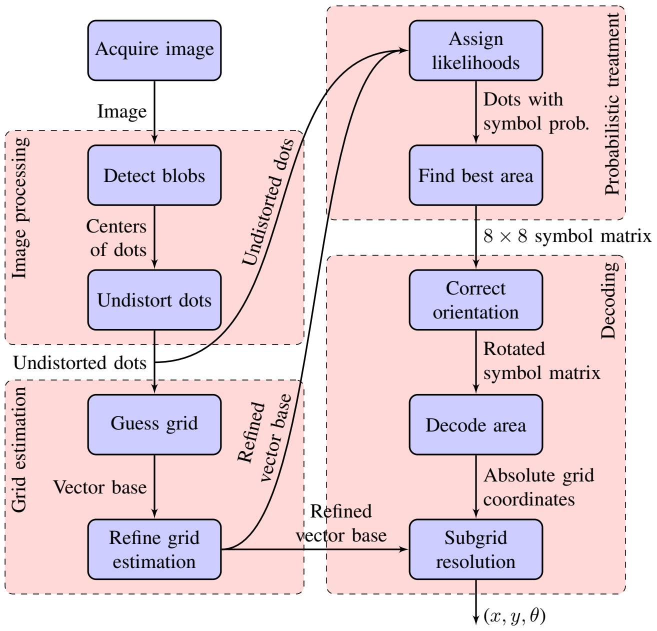
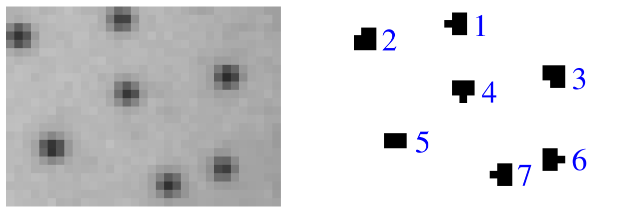


Fig. 1: The pipeline for decoding an image



(a) Dots on a captured frame (b) Thresholding & blob detection

Fig. 2: Dot detection

1) Image Processing: The positions of the dots are obtained through a global thresholding and a standard binary blob-detection algorithm (Figure 2) where the centers of mass of detected blobs correspond to dot positions. Only blobs with sufficient pixels are retained in order to overcome salt and pepper noise.

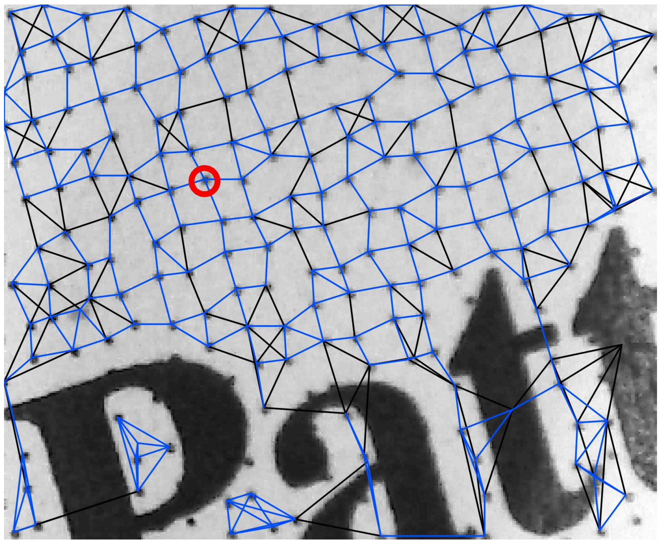


Fig. 3: Edges detected by the 4-nearest neighbor search. Blue lines represent symmetric connections (both dots agree to be neighbors) and are used in estimating the grid. Red circle marks the intact neighborhood center used as the grid origin.

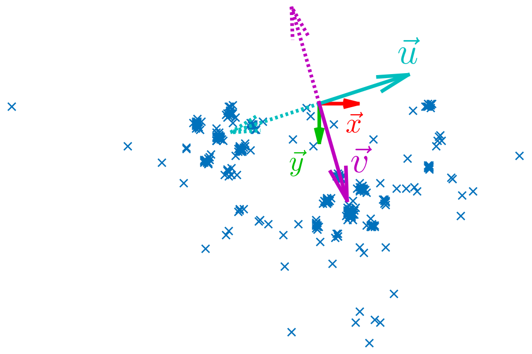


Fig. 4: Symmetric edges from Figure 3, plotted at the same origin in image coordinates $\{\vec{x}, \vec{y}\}$. Due to blob detection's ordering, all edges point downwards. The constrained k-means algorithm assumes 4 clusters which are grouped into two pairs. After each step, each pair is forced to be symmetric around the origin. Final pairs are represented with one dotted and one solid vector each. These provide an acceptable initial estimate for $\vec{u}$ and $\vec{v}$.

We assume that in practice, an optical system in close-up focus is likely to utilize a wide-angle lens in order to decrease the optical system height while imaging a sufficiently large area. These lenses typically suffer from barrel distortion, which can lead to significant positioning errors on the dots away from the center. To prevent this, we use the Brown-Conrady model (only radial distortion, 2nd degree):

$$\vec{r_d} = \vec{p_d} - \vec{m} \tag{1}$$
$$\vec{p_u} = \vec{m} + \vec{r_d}/(1 + k\|\vec{r_d}\|^2) \tag{2}$$

where $\vec{m}$ is the image center, $\vec{p_d}$ is a given dot's position in the original image and $\vec{p_u}$ is the same dot's undistorted position. The radial distortion parameter k is calculated offline from calibration images and is static during runtime.

2) Initial Grid Estimation: The grid is modeled as a vector space spanned by $\vec{u}$ and $\vec{v}$ at an origin $\vec{o}$. The vectors are aligned to the grid directions and their lengths are defined to be the average grid spacing. In order to find an initial estimate, the neighborhood connections of the dots are used: for each dot, the four nearest neighbors are first determined (Figure 3). $\vec{u}$ and $\vec{v}$ are then estimated by clustering the set of *symmetric* edges (where both dots agree to be neighbors) through a constrained k-means algorithm (Figure 5) on a data set similar to the one in Figure 4.

The origin of the new coordinate system is found by searching for an *intact* neighborhood of 3×3 dots close to the center of the image (example in Figure 3) in order to minimize cumulative errors due to the estimated length of the base vectors $\vec{u}$ and $\vec{v}$. There are two conditions to be an *intact* neighborhood: *1)* all of the four neighbors of a starting dot (*side dots*) must be symmetrically connected to the starting dot and *2)* there must be exactly four other dots (*corner dots*) in the 3×3 dot grid that are connected to two distinct side dots. If such a neighborhood is found, a weighted average of the starting dot's and the side dots' positions is used as the origin. If not, the search continues with the next closest candidate.

We denote the dot positions in the new coordinate system with $\vec{c}$ (in units of $\|\vec{u}\|, \|\vec{v}\|$). By rounding them to the nearest integer, we obtain the coordinates $\vec{g}$, hereafter called *grid coordinates*. Using these, the dots' offsets with respect to the grid coordinates (called $\vec{\delta}$) are calculated:

$$\vec{c} = \begin{bmatrix} \vec{u} \cdot (\vec{p} - \vec{o})/\|\vec{u}\|^2 \\ \vec{v} \cdot (\vec{p} - \vec{o})/\|\vec{v}\|^2 \end{bmatrix} \tag{3}$$
$$\vec{g} = \lfloor \vec{c} \rceil \tag{4}$$
$$\vec{\delta} = \vec{c} - \vec{g} \tag{5}$$

The grid coordinates, if estimated correctly, reveal the true neighborhood of dots: A pair of dots are now considered neighbors only if their grid coordinates are adjacent in either the $\vec{u}$ or the $\vec{v}$ axis.

3) Grid Refinement: The grid origin estimate $\vec{o}$ is refined using the median value of the offsets (calculated separately in each of the two axes):

$$\vec{o} \leftarrow \vec{o} - \text{median}(\vec{\delta}) \tag{6}$$

The length of vectors $\vec{u}$ and $\vec{v}$ are refined as well by setting them to the median distances between all neighbor dots (in their respective axes), according to the new true neighborhood. The offsets to the grid positions $\vec{\delta}$ are finally recomputed based on the better grid estimation.

4) Likelihood Assignment: This step consists in assigning likelihoods for each dot to be one of the four symbols (*up*, *down*, *left*, *right*). As Figure 6 shows, the probabilities for each of the two directions (x and y) can be calculated independently by projecting the offsets $\vec{\delta}$ onto the diagonals.

For a given dot, we assume that the projected offset r follows a Gaussian distribution, centered around a nominal offset r_0 (*i.e.* the offset of a perfectly detected dot). Since

```
Input: List of symmetric edges $E$
Output: Main grid directions $\{\vec{u}, \vec{v}\}$
 1: procedure CONSTRAINED $k$-MEANS
 2:     Initialize cluster means $\vec{m}_{1-4}$
 3:     repeat
 4:         Number of dots belonging to cluster $k$: $n_k \leftarrow 0$
 5:         Accumulators: $\vec{a}_{1-4} \leftarrow \vec{0}$
 6:         for all edges $\vec{e}$ from $E$ do
 7:             $k \leftarrow$ index of smallest distance $\vec{m}_k - \vec{e}$
 8:             $\vec{a}_k \leftarrow \vec{a}_k + \vec{e}$
 9:             $n_k \leftarrow n_k + 1$
10:         end for
11:         $\vec{m}_1 \leftarrow \frac{\vec{a}_1 - \vec{a}_3}{n_1 + n_3}$
12:         $\vec{m}_2 \leftarrow \frac{\vec{a}_2 - \vec{a}_4}{n_2 + n_4}$
13:         $\vec{m}_3 \leftarrow \frac{\vec{a}_3 - \vec{a}_1}{n_1 + n_3}$
14:         $\vec{m}_4 \leftarrow \frac{\vec{a}_4 - \vec{a}_2}{n_2 + n_4}$
15:     until $\vec{m}_k$ did not change or iterations $> 10$
16:     Dots belonging to cluster $k$: $p_k \leftarrow \varnothing$
17:     for all edges $\vec{e}$ from $E$ do
18:         $k \leftarrow$ index of smallest distance $\vec{m}_k - \vec{e}$
19:         if $k = 1$ or $k = 2$ then
20:             $p_k \leftarrow p_k \cup \{e\}$
21:         else
22:             $p_{k-2} \leftarrow p_{k-2} \cup \{-e\}$
23:         end if
24:     end for
25:     $\vec{u} \leftarrow \text{median}(p_1)$
26:     $\vec{v} \leftarrow \text{median}(p_2)$
27: end procedure
```

Fig. 5: Constrained k-means algorithm to find grid directions. Medians are calculated separately in each of the two axes.

the calculation of exponentials is costly, we use a custom function to build a quasi-probability distribution (Figure 7):

$$r_x = \delta_v + \delta_u \quad (7)$$

$$r_y = \delta_v - \delta_u \quad (8)$$

$$f(r) = \begin{cases} \frac{3r}{2r_0} - \frac{r^3}{2r_0^3} & \text{if } r^2 < 3r_0^2 \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

This distribution is used to create two quasi-probability matrices $\mathbf{P}^{(x)}$ and $\mathbf{P}^{(y)}$ that cover all detected dots, constructed according to the algorithm in Figure 8. A positive entry in $\mathbf{P}^{(x)}$ denotes that the associated dot's x bit is likely a 1 while how much likely is given by the entry's magnitude, between 0 and 1. A negative entry denotes the same for -1. $\mathbf{P}^{(y)}$ works similarly for the y bit. For a given dot, the magnitude of the product of its entries in these matrices, *i.e.* $\mathbf{P}_{i,j} = |\mathbf{P}^{(x)}_{i,j} \cdot \mathbf{P}^{(y)}_{i,j}|$, represents the likelihood of that dot to represent any symbol accurately.

5) Best 8×8 *Decoding Region Selection:* In a typical implementation, more dots are visible on a given frame than required for the decoding. For improved robustness, we choose the best 8×8 area by finding: $\max_{i,j} \sum_{i,j}^{i+8,j+8} |\mathbf{P}^{(x)}_{i,j} \cdot \mathbf{P}^{(y)}_{i,j}|$. Maximization is achieved through convolution of the product matrix $\mathbf{P}$ with an 8×8 window (example result in Figure 9).

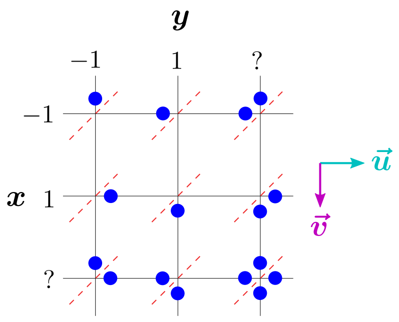


Fig. 6: Each dot encodes two bits (x, y). In order to find out whether the x bit is 1 or -1, it is sufficient to know on which side of the diagonal (red dashes on the figure) the dot lies. Thus, in order to assign probabilities for x, we project the dot onto the diagonal by adding the components of the offset from the grid intersection: $\delta_v + \delta_u$ (and same for y with the other diagonal, *i.e.* by $\delta_v - \delta_u$).

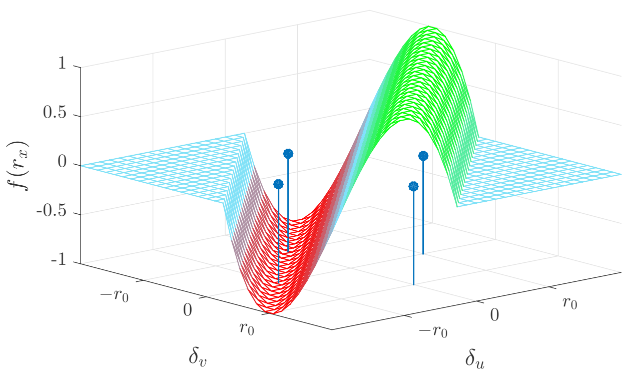


Fig. 7: The quasi-probability distribution for a dot's x bit, $f(r_x) = f(\delta_v + \delta_u)$ approximates a Gaussian mixture distribution with the nominal offset positions r_0 as means. The four possible dot positions are shown in blue. $f(r_y)$ is similar but is symmetric with respect to the other diagonal.

6) Finding the Correct Orientation: Decoding depends on the orientation of the grid, which is not yet accounted for at this stage. The grid directions found in earlier steps ensure the right orientation of decoding only if the device has not rotated by more than 45° with respect to the dot pattern. If this is not the case, dots will be read as if rotated by 90°, 180° or 270°, causing at least one axis to be read upside-down. Both x and y bit sequences in such an axis will have all their bits flipped and their order reversed. At this point, we rely on another property of the main number sequence: none of its 8-long substrings are found in the sequence when bit flipped and reversed (hence a reason for using an 8×8 matrix instead of 6×6). Therefore, the correct orientation is determined by attempting to find the 8-long rows/columns both normally and after bit flipping and reversing in the main number sequence and voting on the correct direction for each axis. Knowing the correct directions, the axes can be labeled as x and y since the coordinate system is right-handed. Finally, symbols, probability matrices and $\{\vec{u}, \vec{v}\}$ are

```
Input: Grid coordinates $\vec{g}_i$, offsets $\vec{\delta}_i$, distribution $f(r)$
Output: Quasi-probability matrices $\mathbf{P}^{(x)}, \mathbf{P}^{(y)}$
1: procedure CONSTRUCT $\mathbf{P}^{(x)}, \mathbf{P}^{(y)}$
2:     $s \leftarrow (\max_i(g_i^u) - \min_i(g_i^u), \max_i(g_i^v) - \min_i(g_i^v))$
3:     Create $\mathbf{P}^{(x)}, \mathbf{P}^{(y)}$ with size $s$
4:     Initialize $\mathbf{P}^{(x)}, \mathbf{P}^{(y)}$ with zeros
5:     for all detected dots $i$ do
6:         $p_x \leftarrow f(\delta_i^v + \delta_i^u)$
7:         $p_y \leftarrow f(\delta_i^v - \delta_i^u)$
8:         if $|p_x \cdot p_y| > |\mathbf{P}^{(x)}_{g_i^u, g_i^v} \cdot \mathbf{P}^{(y)}_{g_i^u, g_i^v}|$ then
9:             $\mathbf{P}^{(x)}_{g_i^u, g_i^v} \leftarrow p_x$
10:            $\mathbf{P}^{(y)}_{g_i^u, g_i^v} \leftarrow p_y$
11:        end if
12:    end for
13: end procedure
```

Fig. 8: Algorithm to construct quasi-probability matrices (one for x bits and one for y bits) that cover all detected dots. Dots associated with incorrect grid coordinates (due to incorrect detection in image processing) are overwritten by the correct dots (with higher associated likelihoods).

rotated by 90°, 180° or 270° accordingly, if required.

The information at this point is enough to determine the device orientation θ, which is simply the orientation of the image in $\{\vec{u}, \vec{v}\}$ coordinates.

7) Decoding: Decoding works as described in Section II-A with the addition of one final property of the main number sequence: any 8-long substring of the main sequence is no longer found in the sequence if *any one* of its bits is flipped. Therefore, instead of attempting to directly find 8-long strings in the main number sequence, we first convolve the strings with the main number sequence in order to find the index with the highest correlation. If it is unknown which bit is flipped, this index will be correct 36% of the time on average (considering all possible bit flips of all 8-bit substrings of the main sequence). This accuracy can be improved by weighting the correlation by the corresponding rows or columns of $\mathbf{P}$ (since it provides information on bit detection quality).

8) Sub-Grid Accuracy: Rewriting the middle of the image $\vec{m}$ (assuming it is on the middle of the device) in absolute grid coordinates enables us to determine the absolute position with a higher resolution than the grid spacing. We know the decoded position $\vec{x}_{\text{decoded}}$ as well as the position $\vec{p}$ of the top left corner of the decoded 8×8 section in the image (Figure 9). We first express the transformation of the middle of the image in $\{\vec{u}, \vec{v}\}$ coordinates centered on $\vec{p}$:

$$\vec{t} = \vec{m} - \vec{p} \quad (10)$$

$$\vec{t}_{\text{proj}} = \begin{bmatrix} \vec{u} \cdot \vec{t} / \|\vec{u}\|^2 \\ \vec{v} \cdot \vec{t} / \|\vec{v}\|^2 \end{bmatrix} \quad (11)$$

Then, knowing that the base is in grid units, the accurate position becomes:

$$\vec{x} = \vec{x}_{\text{decoded}} + \vec{t}_{\text{proj}} \quad (12)$$

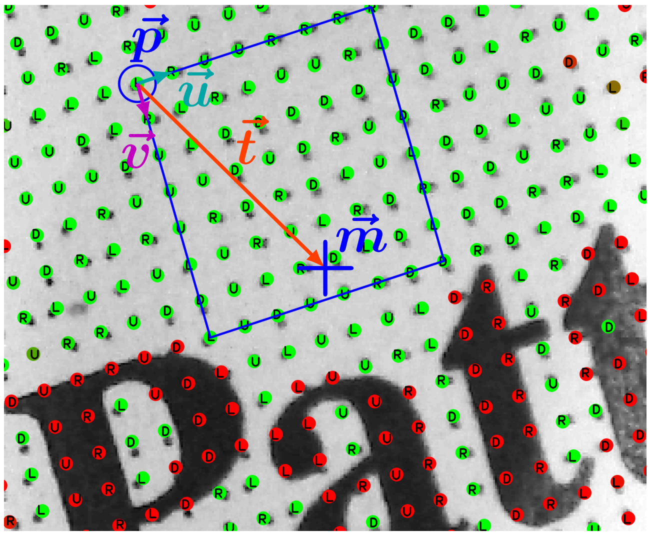


Fig. 9: Grid positions, marked with the colored dots (green: good quality; red: bad quality). The most likely symbol is marked on top of the dots (R, L, D, U). Blue square corresponds to the best area. Blue circle ($\vec{p}$) represents the top left corner which is the position to decode. Device center ($\vec{m}$) is displaced from the decoded position as much as $\vec{t}$.

Component (off-the-shelf)	**Cost** (€)
Lens (5.5mm focal length, S-mount)	1.90
Lens mount	0.17
Microcontroller (*PIC32MZ1024ECG064*)	8.51
Image sensor (*MT9V034C12STM*)	12.86
NIR LEDs (*VSMY3850-GS08*)	3 × 0.46
MOSFET for LED switching (*BSS138BK*)	0.06
0.1% precision resistors for LED voltage drop	3 × 0.29
Total	**25.75**

TABLE II: List of localization components and their costs. Bypass capacitors and various non-precision resistors are not included due to negligible cost.

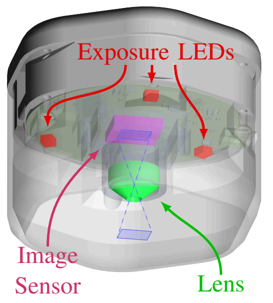
III. Implementation

A. Reference Hardware & Software

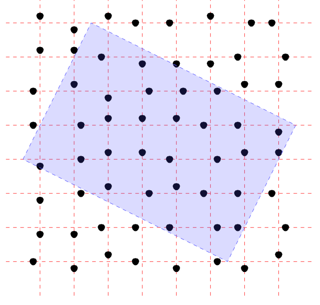
Our reference localization device is implemented on a single double-sided Printed Circuit Board (PCB) housing all of the relevant electronics and optics in order to reduce production cost (Figure 10a). All of the physical components are off-the-shelf, low-cost components; they can be seen in Table II along with their typical cost.

The optical pipeline consists of the following:

1. Generation of the **dot pattern**, and placement of the device directly on top of it;
2. **Exposure** of the scene with Near-Infrared (NIR) LEDs;
3. **Focusing** of light using a board lens; **framing** of the image;
4. **Capturing of the image** onto the image sensor; transmission of the image to the microcontroller;
5. **Image processing and pattern decoding** on the microcontroller; broadcasting of the pose.



(a) Device's optical hardware. Three near-infrared LEDs expose the scene that is focused by a lens onto the image sensor. The field of view geometry (in blue) is exaggerated.



(b) Example pattern on the ground below the device (scaled up about 10 times). Dashed red lines are for reference only, they do not physically exist. The dot size and spacing versus the area viewed by the camera (in blue) is exaggerated.

Fig. 10: Optical hardware setup

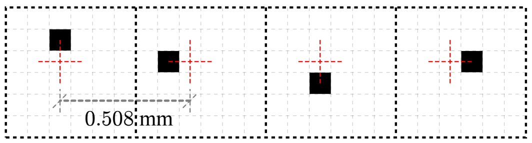


Fig. 11: Choice of symbol glyphs. When the grid spacing is chosen as 0.508mm, the dots align with the printer dots under 300dpi density. Red cross indicates the origin of symbols.

1) Dot Pattern: The dot pattern is generated from a text file containing the adequate sequences of symbols as letters (u, d, l, r). It is rendered using a custom font made of four glyphs representing the four possible positioned dots (Figure 11). We choose a grid spacing of 0.508mm: considering the dots offsets ($0.508/6 = 0.084667$mm from the grid intersections), it allows the dots to be aligned with the *physical* dots printed by the printer at 300dpi. Rendered sequences of dots are easily overlaid onto pre-existing Portable Document Format (PDF) documents, allowing for easy distribution, viewing, compression and printing.

2) Exposure: The scene, *i.e.* the printed dot pattern on the surface directly under the device, is exposed using three identical NIR LEDs (850nm wavelength) evenly placed at equal distance from the optical center. Their voltage drops are constrained to be as similar as possible by precision resistors. These ensure that the scene is illuminated as uniformly as possible. To have controlled exposure, the scene is isolated from external light sources, such as ambient daylight, by the device housing itself (3D printed, 1.2mm thick).

With our specific clock speed, the exposure time can be chosen to be as low as approximately 1/50000s thanks to the image sensor's global shutter. From this, we gradually increased the exposure time to 1/7684s at which point the image was sufficiently exposed so that the thresholding was satisfactory. Given the physical size of one pixel on the ground (0.046mm) and the physical image size (furthest pixel is at 4.29mm orthogonal distance away from center), it would take approximately 353mm/s linear speed in the x or y axes or 785rpm angular speed to cause motion blur of one pixel magnitude.

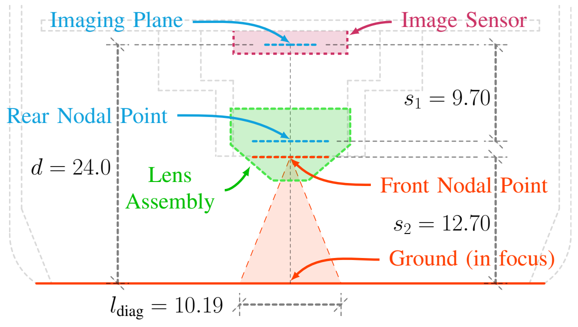


Fig. 12: Cross-section of the optical system, side view, all units in mm.

3) Focusing & Framing of the Image: In order to focus the exposed image onto the image sensor, an off-the-shelf S-mount CCTV lens is used. As a compromise between Field of View (FOV) and typically increased distortion, a lens with 5.5mm nominal focal length (f) and 54° nominal diagonal FOV was chosen. It is mounted on a manual focus housing, as seen in Figure 12.

The suitable distance between the image plane and the point of focus on the ground (d) was found by successively increasing the device housing height (starting with the theoretical limit, $4f = 22$mm) and attempting to manually focus the lens. With this, after distortion correction, the physical shape of the image becomes a 5.48×8.59mm rectangle with a diagonal length of 10.19mm. In this shape, an 8×8 dot matrix with 0.508mm dot spacing must fit. The largest (diagonal) length of this matrix is calculated to be $\sqrt{2} \times 0.508 \times (7 + 1/6 + 1/6 + 1/6) = 5.39$mm in the worst case (the physical diameter of one dot and the possible offsets of the furthermost two dots are each included as 1/6 of the grid spacing), which ensures that at least 8×8 dots fit inside the image in any given orientation.

4) Image Capture: Digital capturing of the image is done by the image sensor (global shutter, grayscale), which runs in master mode and generates all necessary timing and data signals. Our microcontroller then uses these timing signals to capture the image data (188×120 pixels, 8-bits per pixel) via Direct Memory Access (DMA). This ensures that the least amount of processing cycles possible are spent for this task.

5) Image Processing & Pattern Decoding: All of the image processing and decoding pipeline (as described in Section II-B) runs locally on the microcontroller (Microchip PIC32MZ, 200MHz core clock, 512Kb SRAM). In order to allow real-time operation, a number of measures are taken. Where possible, the lack of Floating Point Unit (FPU) is compensated by manually introducing a rational number representation with fixed divisor, while taking care that no overflow occurs. Memory allocation is made statically where possible to avoid dynamic memory allocations. Lookup tables are used where feasible. Finally, a polynomial (Equation 9) is used instead of Gaussian distribution functions (requires exponentials) to increase performance.

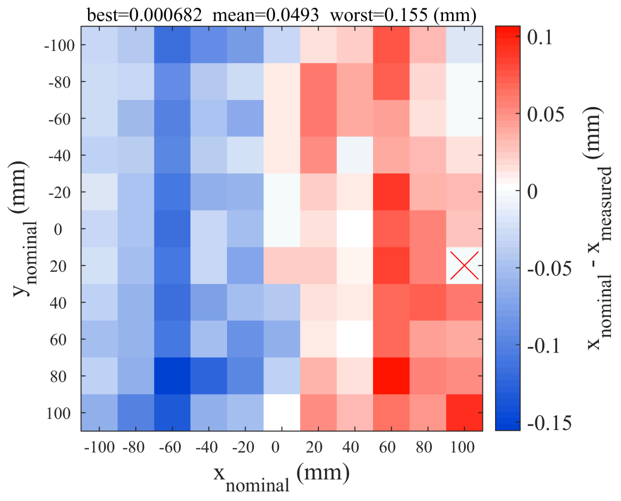


Fig. 13: Accuracy of x coordinate measurements when device is stationary, 20 samples each. Position marked with the cross was consistently misdecoded. Best, mean and worst accuracies are calculated with absolute values.

Coordinate	Accuracy	Precision (± one σ)
x	0.155 mm	±0.010 mm
y	0.273 mm	±0.014 mm
θ	1.581°	±0.407°

TABLE III: Performance when device is stationary; worst absolute values.

B. Open-Source Software Release

Our reference software implementation is available under an open-source license from `chili.epfl.ch/libdots`. It can be built as a standalone library and has been successfully cross-compiled for low-end targets such as the PIC32MZ microcontroller. The repository also provides a sample test application that works with a standard desktop webcam (as long as it permits to focus on close objects, so that printed dots are visible). Tools to generate dot patterns and overlay them on any PDF file are provided as well.

IV. Validation

A. Methodology

Performance of individual localization coordinates (x, y, θ) were each measured separately. For x and y, the device was mounted (without modifications) on the toolhead of a Computerized Numerical Control (CNC) platform with 17µm nominal step size. For θ measurements, the device was mounted (without modifications) on a servomotor with 0.29° nominal accuracy. Commands given to this platform and servomotor were recorded as ground truth values, referred to as *nominal* values from here on.

Measurements were done on an A3 sheet carrying only the pattern and no other graphics, printed in black and white by a Xerox Workcentre 7665 laser printer. y was chosen as the paper rolling axis while x was chosen as the laser scanning axis (corresponds to head motion axis in inkjet printers). In this setup, the sources of significant systematic noise include:

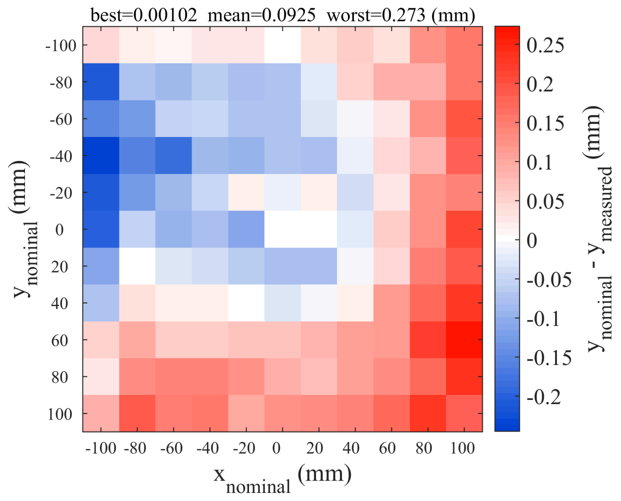


Fig. 14: Accuracy of y coordinate measurements when device is stationary, 20 samples each. Best, mean and worst accuracies are calculated with absolute values.

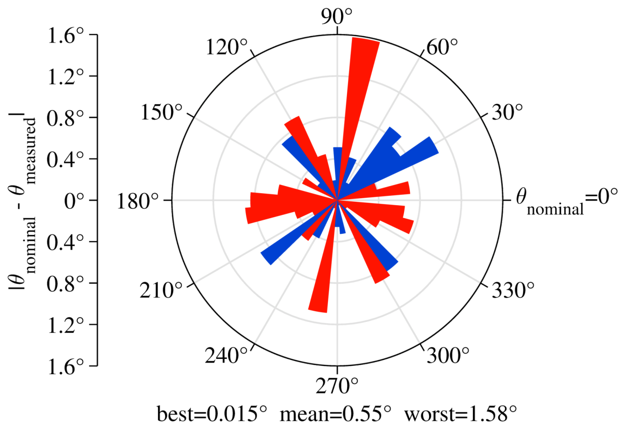


Fig. 15: Accuracy of orientation measurements when device is stationary, 20 samples each. Positive and negative biases are coded with red and blue respectively. Best, mean and worst accuracies are calculated over absolute values.

- Pattern printing process inaccuracies
- Plastic device housing deformation and manufacturing tolerances
- Image sensor and lens assembly mounting inaccuracy
- Paper placement inaccuracy below the device

To measure performance of x and y coordinates, the device was moved to 11×11 distinct positions on a 200×200mm grid in spiral from the center towards the periphery. 20 real-time samples were collected for each position; these can be viewed in Figures 13 and 14 for x and y respectively. 99.17% of x and 100% of y coordinates were correctly decoded. 1 out of 121 x positions was consistently measured to be in an unrelated location due to misreading of dot offsets.

To measure performance of angular position, the device was rotated to 36 distinct angles over 360°. 20 real-time samples were collected for each position; these can be viewed in Figure 15. 100% of these angular positions were correctly decoded. Finally, an overview of accuracy and precision can be seen in Table III.

The average framerate of localization was measured to be 46.6Hz. The system was measured to consume 352mW when stationary (sleeps, wakes up every second to process one frame to decide whether moved, exits stationary mode if moved) and 873mW when moving (continuously processes frames, enters stationary mode if not moved for 5 seconds).

B. Results

Figures 13 and 14 evidence that distinct regions on the paper induce biases on x and y coordinate measurements; we attribute this systematic error mainly to the pattern printing process. The y axis is seen to be significantly less accurate than the x axis (all 121×20 samples used, unpaired t-test, $p < 0.0001$). This observation leads us to consider that the uncertainties in the paper rolling process were more significant than the laser neutralizing process in our case. This may be generally true for similar axes in different printing techniques, such as the paper rolling axis *vs.* inkjet head motion axis in inkjet printers. To generalize however, tests should be done with other laser and inkjet printers.

In any case, certain x and y biases should be expected by the user of this localization method. Moreover, there is no guarantee that these accuracies are bounded across the whole localization space (unlike θ whose space and therefore accuracy is bounded). In reality, given a paper size and a specific printer, bounds for x and y accuracy can be measured; but the biases are likely to be worse across larger distances due to cumulative systematic printing errors (*e.g.* slipping and deformation of paper).

Considering the power consumption, a typical single cell 600mAh Lithium-Polymer battery, such as the one we used for our experiments, lasts for more than 2.5 hours in the worst case. In reality, it lasts longer since the device is not always moving, and will allow hours-long experiment sessions.

V. Conclusions

In this article, we introduced to the robotics community a real-time, pattern-based 2D localization method well suited for mobile robots. Furthermore, our contribution includes an open-source, high-performance software implementation, alongside a reference hardware setup whose accuracy has been validated. The main benefits of our method include:

- Absolute localization;
- Runs entirely on device, no need for a central server, thus no need for external communications;
- Works in real-time using off-the-shelf components with near-constant processing time per frame;
- Unlimited scalability, each device localizes itself;
- Fully robust against occlusions and lighting conditions as long as the device rests on the surface;
- Designed to work while in motion, can be used *e.g.* for real-time trajectory tracking on mobile devices;
- No calibration required;
- Simple deployment and disposal, as it only requires regular printable support (*e.g.* paper) that is to be placed on a surface and can later be removed and stored away;
- Working area is only limited by printing capacity, printed patterns can be stitched together to cover larger areas if the stitching can be calibrated;
- Patterns are unobtrusive and can be overlaid on top of existing documents, *augmenting* them with localization;
- Affordable, below €30 per device; printable support with dot pattern can be reproduced at very low cost if damaged or if replication is needed.

We also identify certain limitations to this method:

- Provides localization in 2D space only (not in 3D);
- Contrary to SLAM techniques, the environment needs to be altered by deploying the dotted pattern;
- Close proximity with the pattern surface is typically required, which makes it less suitable for certain robots (such as legged robots and drones);
- Printing the pattern may prove non-trivial (exactly 1:1 scale, at least 300dpi resolution) for large surfaces;
- Accuracy and precision is dependent on the quality of the printer being used.

However, within the frame of this work (2D localization requiring the deployment of passive markers in the environment), we believe that this method may prove to be particularly relevant and valuable to a range of sub-domains, including swarm robotics (where high-accuracy, scalability and low cost are especially desirable), educative playful mobile robots and tangible interfaces (where low processing load, low cost and real-time operation are often sought after).

Acknowledgements

This research was supported by the Swiss National Science Foundation through the National Centre of Competence in Research Robotics. The authors would also like to thank Roger Fong for his work on the software release.

References

[1] M. Windolf et al. “Systematic accuracy and precision analysis of video motion capturing systems - exemplified on the Vicon-460 system”. *J. of Biomechanics*, 41(12): 2776–2780, 2008.

[2] K. Nakatsuma and H. Shinoda. “High Accuracy Position and Orientation Detection in Two-Dimensional Communication Network”. In *Conf. on Human Factors in Computing Systems*, 2010.

[3] S. Saito et al. “Indoor Marker-based Localization Using Coded Seamless Pattern for Interior Decoration”. In *Virtual Reality Conf.*, 2007.

[4] L. Jorissen et al. “Robust Global Tracking Using a Seamless Structured Pattern of Dots”. In *Augmented and Virtual Reality*, pages 210–231. Springer, 2014.

[5] R. Mautz. “Indoor Positioning Technologies”. Habilitation Thesis, ETH Zürich, Switzerland, 2012.

[6] M. P. Pettersson. “Method and Device for Decoding a Position-Coding Pattern”, December 5 2006. US Patent 7,145,556.

[7] E. Aboufadel et al. “Position Coding”. *arXiv preprint arXiv:0706.0869*, 2007.