Dothraki: Tracking Tangibles Atop Tabletops Through De-Bruijn Tori

Dennis Schüsselbauer
University of Regensburg
Regensburg, Germany
dennisschuesselbauer@web.de

Andreas Schmid
University of Regensburg
Regensburg, Germany
andreas.schmid@ur.de

Raphael Wimmer
University of Regensburg
Regensburg, Germany
raphael.wimmer@ur.de

ABSTRACT

Tangibles are small, graspable objects that act as input devices or physical representations of digital data. Oftentimes, it is desirable to track the position of tangibles on a surface and their relation to each other. However, outside-in tracking techniques - such as capacitive touchscreens or cameras - require setting up elaborate infrastructure and are prone to occlusion or interference. We propose Dothraki, an inside-out tracking technique for tangibles on flat surfaces. An optical mouse sensor embedded in the tangible captures a small (36×36 pixel / 1×1 mm), unique section of a black-and-white De-Bruijn dot pattern printed on the surface. Our system efficiently searches the pattern space in order to determine the precise location of the tangible with sub-millimeter accuracy. Our proof-of-concept implementation offers a recognition rate of up to 95%, robust error detection, an update rate of 14 Hz, and a low-latency relative tracking mode.

CCS CONCEPTS

• **Human-centered computing** → **Interaction devices**; Mixed / augmented reality.

KEYWORDS

tracking, tangibles, de-bruijn torus, hardware

ACM Reference Format:
Dennis Schüsselbauer, Andreas Schmid, and Raphael Wimmer. 2021. Dothraki: Tracking Tangibles Atop Tabletops Through De-Bruijn Tori. In *Fifteenth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '21), February 14-17, 2021, Salzburg*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3430524.3440656

1 INTRODUCTION: SPATIALLY AWARE TANGIBLES

Tangibles, also called *tangible blocks*, or *bricks* [9] are small, graspable objects that can be used as input devices or as physical representations of digital data and relations [13]. They are tools that support interaction with intangible information by leveraging our innate and trained ability to manipulate the physical environment. Through physical manipulation of these tangibles, actions can be executed; information can be edited, transferred or combined. Tangibles inherently support bi-manual [9], eyes-free [32], and collaborative [27] manipulation. Thus, they extend and augment the input and output capabilities of the traditional keyboard/mouse/display troika. Due to these properties, tangibles have been used in many different application areas so far - e.g., for playing games [27], for programming [23], to support learning [4, 28, 34], or to manipulate digital media [10, 15, 21, 29, 36]. Shaer and Hornecker give a good overview of applications and properties of tangibles [25].

Figure 1: Dothraki is a reference implementation for tracking tangibles on tabletop surfaces. Via an optical mouse sensor, a tangible detects its position within a printed De-Bruijn-torus pattern and tracks its own movement with low latency.

For many applications, it is desirable to determine and track the location or orientation of a tangible. Fitzmaurice [9] calls this capability "spatial awareness". When the position of a tangible is known, it can be used for selecting objects in 2D or 3D space, changing scalar parameters, or as a proxy for arranging and combining multiple pieces of information [36]. However, robust tracking of small, handheld objects is non-trivial.

In this paper, we present *Dothraki* (Figure 1), an approach for tracking the positions of multiple active tangibles on a flat surface that is covered with a custom dot pattern. An optical sensor from a computer mouse is embedded into the tangible and continuously captures a section of this pattern right under the tangible. The captured image is then used to determine the tangible's location within the pattern space, and therefore within the surface area. Dothraki is low-cost and can be integrated with common microcontroller platforms. It requires no external infrastructure besides a sheet of paper printed with the pattern and a computer that runs

TEI '21, February 14-17, 2021, Salzburg

the position-tracking code. Relative motion can be tracked with very low latency by the mouse sensor itself. This makes Dothraki suitable for ad-hoc, collaborative applications, such as educational games.
In the following, we describe how Dothraki works, document its properties and limitations, and discuss applications that can be implemented using Dothraki.

2 BACKGROUND AND RELATED WORK

Dothraki is an inside-out, occlusion-free tracking technique for tangibles on flat surfaces. It uses a mouse sensor for image acquisition. We present basic background information and previous research on each of these topics.

2.1 Outside-in / Inside-out Tracking

In general, the position of objects in two or three dimensions can be tracked in two ways:
With *outside-in* tracking, sensors in the environment are used to determine the location of the object. Oftentimes, the tangible is modified slightly in order to enable or simplify tracking. For example, camera-based tracking approaches may require the object to be of a certain color, to have a unique pattern on it (e.g., ARuCo markers), or to be augmented with retroreflective markers (e.g., OptiTrack).
With *inside-out* tracking, the object itself is equipped with sensors that capture properties of the environment in order to determine its position. The environment may also be augmented in order to facilitate tracking. For example, a camera in the tangible might capture a unique pattern in the environment [18], or a magnetic field sensor might measure strength and shape of a magnetic field that is generated by coils or permanent magnets in its proximity [19].
Outside-in tracking has the primary advantage that there are few constraints regarding size, shape and material of the tangible. The tangibles don't need a battery or complex electronics which makes it possible to use many of them in an application.
Inside-out tracking has the primary advantage that little external infrastructure is needed. Tracking data is available directly in the device and it is easier to add additional tracked objects to the environment without degrading overall tracking performance.
Mutual awareness - i.e., the ability of tangibles to know about the relative position of other tangibles - can be implemented via inside-out or outside-in tracking. In this case, the tangibles themselves can set up the tracking environment so that no external infrastructure is needed. For example, Sifteo Cubes[1] are not spatially aware in general but contain electromagnetic transceivers on four sides that allow them to detect and identify other Cubes directly next to them. Krohn et al. demonstrate a prototype that employs multiple infrared transmitters and receivers per device which allow the devices to track each other's relative position [16].

2.2 Occlusion and Line of Sight

Both inside-out and outside-in tracking techniques may suffer from occlusion. Optical 6DOF tracking systems which are suitable for tangible interaction (e.g., Valve's Lighthouse [35], OptiTrack) require a line of sight between the tracked object and the sender/receiver hardware that is installed in the environment. This poses a problem for collaborative use cases and small objects, as users' hands and bodies will constantly occlude the objects. The occlusion problem also affects systems using cameras mounted above a table [33].
Furthermore, these systems are not well suited for ad-hoc use because they require infrastructure to be set up and calibrated before use. In addition, camera-based systems may also pose privacy concerns. This may hinder their use in sensitive environments such as schools.
Overall, such general-purpose tracking systems may be less suited for ad-hoc, collaborative tangible interaction.

2.3 Surface-based Tracking Approaches

A surface - e.g., a tabletop - may include sensors or markers for tracking. This avoids most occlusion issues as long as the tangibles are in contact with the surface. While there are novel approaches using e.g. magnetic field sensing [19] or near-field communication [30], the three most commonly employed tracking approaches on tabletops are outside-in optical tracking, outside-in capacitive tracking and inside-out optical tracking.
Outside-in optical tracking uses a camera mounted beneath the surface - or light sensors embedded into a display - to detect and track objects. The surface needs to be transparent or translucent. This appproach is used widely [15, 17] because it requires only off-the-shelf cameras and a simple lighting setup. Many different objects can be tracked reliably and with reasonable speed. Objects can be identified via unique optical codes (*fiducials*). However, a custom tabletop is required and the approach is susceptible to changes in lighting conditions. Therefore, outside-in optical tracking is not well suited for ad-hoc interaction or for applications where each user should have their own private surface/table - e.g., in classrooms.
Outside-in capacitive tracking uses raw data from a capacitive touch screen to detect and distinguish tangibles placed on it. These need to be equipped with conductive traces that capacitively couple the touch screen's antennas to each other or to electrical ground. Voelker et al. [31] give a good overview of the principle and related work. An advantage of this approach is that it also works with tablets or smartphones as tracking surfaces. This makes it suitable for ad-hoc use. However, the interaction space is limited by the size of the capacitive touch screen. Furthermore, only active tangibles with embedded electronics can be reliably detected.
Inside-out optical tracking uses an optical sensor - usually a camera - to capture an image of the surface below the tangible. Based on features in this image, the tangible may determine its absolute location or relative movement. For example, the sensor of an optical mouse contains a tiny camera and calculates relative movement based on movement observed in the sensor image. There are two approaches to determine the absolute location of a tangible via inside-out optical tracking. The system may generate a fingerprint of the surface seen by the camera and compare it to a database of known fingerprints or classify it using a set of heuristics [24]. While this approach works in certain cases, where the fingerprints

[1] https://web.archive.org/web/20131109052319/https://www.sifteo.com/cubes

are distinct and robust enough, it does not allow for precise or reliable tracking. The more common approach is to embed a digital code into the surface that can be captured and extracted by the sensor. This approach is discussed in more detail in the following paragraphs.

2.4 Optical Pattern Tracking

Augmenting a surface with a printed pattern that encodes absolute positions allows objects with embedded cameras to precisely and robustly detect and track their own position. This approach has been employed by manufacturers of digital pens. *Anoto* pens contain a small camera that captures a dot pattern printed on the surface (usually a sheet of paper) [22]. The dot patterns encodes a position by slightly shifting the dots relative to a common grid. Movement trajectories are stored on the pen or streamed to a host computer via a wireless connection. Smart pens by other manufacturers, such as the *Neo Smartpen* or the *Ravensburger TipToi*, employ their own, similar, encodings[2]. While some of these encodings have been reverse engineered, all of them are under-documented, protected by patents, and require custom sensor/lens combinations.
Hostettler et al. [12] developed an optical position tracking approach for robots/tangibles based on the Anoto pattern. It requires a custom camera/lens combinaton in order to read the pattern. The software, *libdots* has been published under an open-source license[3]. However, the authors mention that the Anoto technology is still protected by patents in multiple countries. This might make it difficult to use *libdots* in commercial products.
The *Sony toio* micro-robots[4] also use an embedded camera to track their position on a mat on which a special pattern has been printed. No further information on the implementation is available, however.

2.5 Using Mouse Sensors for Tracking

Mouse sensors have been used for precisely tracking a variety of objects. Jackson et al. [14] suggest using them to track a vehicle's movement along the road. Domburg [7] implemented a simple handheld scanner by reading out the raw image from the sensor and stitching multiple images together. Borsato and Morimoto [3] explored whether a mouse sensor could be used for eye-tracking. Schüsselbauer et al. [24] built a tangible with an embedded mouse sensor that allowed it to distinguish different textures printed on a piece of paper and activate a corresponding mode. They also used the mouse sensor to track X/Y movement of the tangible.
In summary: while there are several methods for tracking tangibles on surfaces, all of them either require complex or expensive setups, are susceptible to occlusion, or are protected by patents.

3 HARDWARE

As the basis for our tangible blocks, we use M5Stack blocks[5] (Figure 2). While M5Stack is a commercial platform, the danger of it becoming unavailable or unusable is rather small for several reasons. Hardware is based on the wide-spread ESP32 microcontroller and commonly available components. For mechanical parts, CAD files are available. Software is open-source and built upon the Arduino ecosystem. Furthermore, the company has been expanding its portfolio for several years, and devices are available through multiple retailers. While the M5Stack blocks contain different sensors, such as an IMU, they are not able to sense their own position or their relation to other devices.
For determining the location of the tangible on the surface, we use a PixArt PMW3360 mouse sensor[6] which contains an optical sensor and a laser diode for illumination of the surface. The sensor

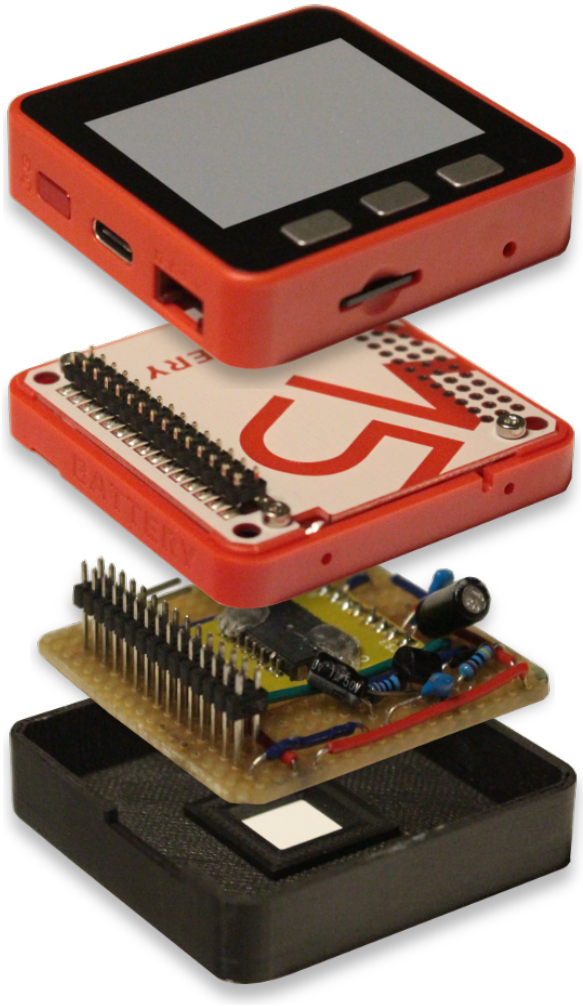


Figure 2: Dothraki hardware consisting of an M5Stack Fire module, a battery module, the PMW3360 mouse sensor, and a 3D-printed case part.

[2]https://github.com/entropia/tip-toi-reveng/wiki/PEN-Optical-ID-and-codes
[3]https://www.epfl.ch/labs/chili/dissemination/software/libdots/
[4]http://www.sony.net/SonyInfo/design/stories/toio/
[5]https://m5stack.com/

[6]Datasheet: http://www.pixart.com/products-detail/10/PMW3360DM-T2QU. Arduino code for this sensor can be found at https://github.com/SunjunKim/PMW3360_Arduino. Our code is based on the manufacturer's datasheet, however.

is attached to the ESP32 via SPI. Only a few passive components need to be added as shown in the datasheet.
Using a mouse sensor instead of a small camera results in three major advantages and two major limitations.
Major advantages are:

- The lens system is optimized for capturing sharp images of the surface with a high update rate.
- Mouse sensors are fast. X-Y-movement is decoded in hardware within the sensor. This results in an update rate of more than 1000 Hz and low resource consumption on the microcontroller. Accessing the raw image captured by the sensor is much slower, however, taking about 30 ms.
- Mouse sensors offer a good combination of low price and high availability. The PMW3360 has been on the market for several years and is still the best sensor available. It is available in small quantities for $10 and requires little external hardware.

Major limitations are:

- Due to consolidations on the market, only few manufacturers of mouse sensors exist anymore. Therefore, it could be difficult to find a replacement if PixArt ceased production of this sensor series. However, the large market for computer mice might ensure availability of such sensors.
- The sensor has a rather low resolution of 36×36 grayscale pixels and captures an area of only 1.0 × 1.0 mm (Figure 1). This means that traditional location patterns - such as the Anoto pattern - cannot be used. We address this limitation with a novel image processing pipeline based on De-Bruijn tori described in the following section.

We decided to offload the image processing pipeline to a computer connected to the M5Stack via wifi. While the M5Stack has just sufficient hardware ressources to run the complete image processing pipeline, doing so would require highly optimized code that would be difficult to modify. Moving image processing to a host computer makes the whole setup more flexible. A similar approach was chosen for the Sifteo Cubes[7].

4 POSITIONING AND TRACKING

Dothraki is able to determine the location of the tangible by instructing the mouse sensor to capture an image of the surface beneath the object, extract the printed dot pattern from the sensor image, and then find the location of the captured section within the whole dot pattern. In the following, we first describe the core principle and then add further implementation details.

4.1 Positioning Based on De-Bruijn Patterns

Our approach is based on finding a unique binary dot pattern within a large pattern that covers the whole surface. De-Bruijn sequences [6] are sequences of k different symbols where each sub-sequence of length n only occurs once within the whole sequence. A special property of a De-Bruijn sequence is that it wraps around, i.e. that a sub-sequence constructed by appending x symbols from the start to n-x symbols from the end is also unique. In the following we focus on binary De-Bruijn sequences, i.e. sequences where a symbol is either `0` or `1`. For example, the De-Bruijn sequence `0011` contains each of the two-bit patterns `00, 01, 11, 10` (wrap-around) exactly once.

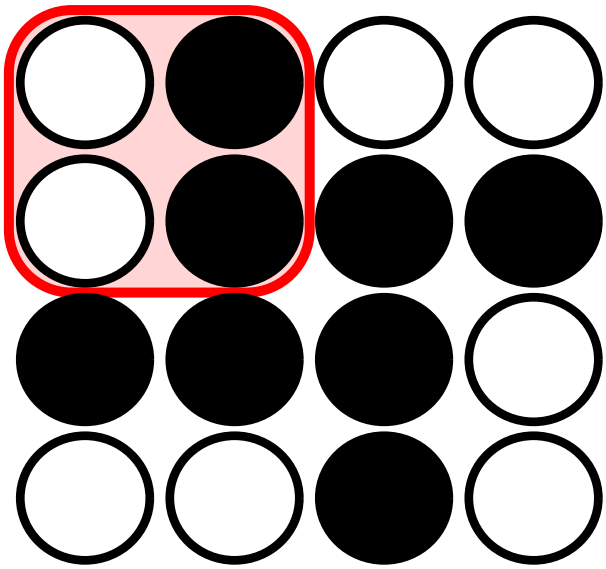


Figure 3: A binary $\{4, 4, 2, 2\}$ De-Bruijn torus. Each unique 2×2 window occurs exactly once.

Extending this concept to two dimensions results in a De-Bruijn torus (DBT). A $\{r, s, m, n\}$ DBT is an array of r×s binary values where each possible m×n *window* occurs exactly once. As indicated by its name, a De-Bruijn torus also wraps around. Figure 3 shows an example of a binary $\{4, 4, 2, 2\}$ DBT.
An obvious implication is that each m×n window encodes a unique position within the DBT. If a DBT is printed onto a surface, a tangible could determine its location on the surface by capturing an image of the m×n window below it and finding its position within the DBT. In principle, this works for arbitrary sizes of the DBT, as long as the tangible can reliably capture at least one whole m×n window.
For the sake of clarity, in the following we use the term *DBT pixel* for a single binary value within the DBT which is represented as a black or white region on the surface. A DBT pixel consists of multiple *printer dots*. A *printer dot* is the smallest printable unit produced by the printer. A *sensor pixel* is a pixel captured by a single photosensitive cell in the mouse sensor.
Figure 4 shows an excerpt of a DBT printed on paper and the area seen by the mouse sensor.
To support as many different applications as possible, the DBT pattern should cover a large area. To achieve high resolution, rather small DBT pixels are desirable. This requires generating a large DBT and efficiently finding the position of a distinct window within the DBT.

[7] https://blog.adafruit.com/2012/12/05/how-we-built-a-super-nintendo-out-of-a-wireless-keyboard-sifteo-sifteo/

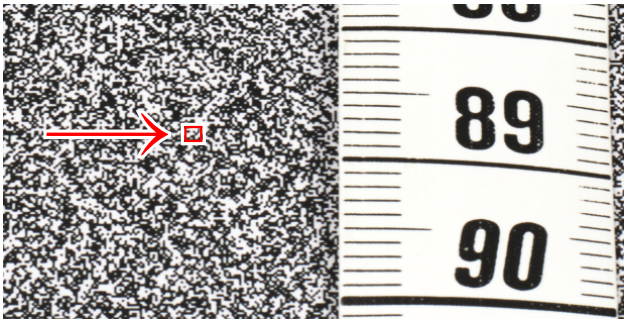


Figure 4: Section of an $\{8192, 4096, 5, 5\}$ De-Bruijn torus (DBT) printed at 150 DBT pixels per inch. A red square indicates the region captured by the mouse sensor. The measuring tape has a millimeter scale.

DBT resolution	100 ppi	150 ppi	200 ppi	300 ppi	400 ppi
# DBT pixels	4×4	6×6	8×8	13×13	17×17
size of DBT pixel	8.1 px	5.4 px	4.0 px	2.7 px	2.0 px

Figure 5: DBT printed at different resolutions as viewed by the PMW3360 sensor. At more than 200 ppi, dust and toner defects make it difficult to reconstruct the DBT pixels. At 100 ppi, the limited number of DBT pixels visible limits the number of positions that can be encoded to 32768. DBT resolutions of 150 and 200 ppi offer a good trade-off between robustness and positioning area. All patterns were printed on a Lexmark MS510dn laser printer at 1200 dpi.

Fan et al. [8] describe an algorithm for iteratively generating larger DBTs from smaller ones[8]. Starting with an $\{r, s, m, n\}$ DBT, the algorithm produces an $\{r, 2^n \times s, m + 1, n\}$ DBT. As a DBT can be transposed (i.e. mirrored along its diagonal) and still maintain its De-Bruijn properties, the algorithm by Fan et al. allows for extending the DBT in both directions.

For our prototypical implementation, we used this algorithm, as implemented by An[9], to generate an $\{8192, 4096, 5, 5\}$ DBT, i.e. a binary pattern of 8192×4096 pixels where each of the distinct 2^{25} 5×5-patterns occurs exactly once. The reasons for choosing exactly this DBT are explained in the next section.

Re-finding a distinct DBT window within the DBT can naively be done via a linear brute-force search. As this would not be practical for large DBTs, we implemented Shiu's algorithm [26]. This algorithm specifically works for the DBTs generated iteratively by the Fan et al. algorithm. It requires that the source DBT and all intermediate transformations are known. Therefore, we store this information when generating our DBT. Shiu's algorithm works by generating a linear mapping $D(M)$ of the DBT window that is to be found, brute-force locating it in the source DBT, and then transforming these coordinates based on the transformations used when generating the large DBT. While alternatives to Shiu's algorithm exist [11, 20], we were satisfied with its practical performance (see section *Properties*).

4.2 Rationale for an {8192,4096,5,5} De-Bruijn Torus

While we explored different DBT types and sizes, we selected an $\{8192, 4096, 5, 5\}$ DBT for Dothraki as it is the largest and most robust one that can be used in practice with the PMW3360 sensor. A first constraint is that the DBT needs to be printable using common laser or inkjet printers. These typically have a print resolution of 600 or 1200 dpi. In order to avoid aliasing artifacts, the size of a DBT pixel should be an integer multiple of the size of a printer dot. Therefore, the effective DBT resolution should be 400, 300, 200, 150, or 100 ppi.

The optical sensor in the PMW3360 captures a surface area of 1.0 × 1.0 mm. As shown in Figure 5, a DBT resolution of 100 ppi would result in only 4×4 full DBT pixels to be visible in the camera image. That would mean that the DBT may only have a window size of 4×4, resulting in $2^{16} = 32768$ unique positions that could be encoded - or 256 in each axis. While this may be sufficient for some applications, the {256, 265, 4, 4} DBT would only cover a size of 65×65 mm. On the other hand, a DBT resolution of 300 ppi would mean that each DBT pixel only covers 2.7×2.7 sensor pixels. With so few sensor pixels per DBT pixel, a dirty lens, dust, or defects in the print can easily result in one or more DBT pixels to be "flipped", i.e. interpreted as black although they are white or vice versa. Therefore, in practice only 150 ppi and 200 ppi are viable resolutions for the DBT. Experiments with different printers showed that a pattern printed at a resolution of 150 ppi could be more robustly reconstructed than at 200 ppi. At 150 ppi, the PMW3360 is able to see a section of about 6.5×6.5 DBT pixels. One *might* therefore choose a DBT with a window size of 6×6 - allowing for 2^{36} positions to be encoded. However, this solution would not be robust enough for most use cases because DBTs have no built-in error detection. The Hamming distance between distinct windows is 1, i.e. if any DBT pixel flips, we get another valid DBT window that is located somewhere else in the DBT. This means that the tangible's reported position would jump all over the place whenever the pattern on the surface was captured or interpreted incorrectly.

We therefore opted for a more robust approach using a DBT with a 5×5 window size. This allows for detecting and partially correcting recognition errors. As the mouse sensor captures a section of 6×6 DBT pixels, we can extract four overlapping 5×5 DBT windows and calculate their individual locations within the DBT. If at least three of these four patterns are located next to each other in the DBT, we can assume that we decoded them correctly. If however, the decoded locations are distributed across the DBT, we can assume that a pixel flip occured and that the actual location cannot be reliably determined. In this case, the system does not output any position and waits for a new raw image from the mouse sensor.

[8] Fan et al. actually describe two ways for constructing larger DBTs: Type 1 and Type 2. We implemented generation and decoding of Type 1 DBTs.
[9] https://github.com/man4/debruijn-torus

Printing the $\{8192, 4096, 5, 5\}$ DBT at a resolution of 150 ppi (and 1200 dpi) covers a surface area of 1370×685 mm - approximately the size of a school desk. Alternatively, the DBT can also be sliced into multiple smaller surfaces. By placing each of the DBT slices on a different table, it is possible to find out on which table the tangible is. We consider a position resolution of 0.16 mm sufficient for most HCI applications.

4.3 Image Preprocessing

In order to extract the DBT pixels from the raw image captured by the mouse sensor, we apply a rather straightforward computer-vision pipeline implemented using OpenCV[10]. We provide default parameters that we found to work well for plain paper. For different types of paper, parameters need to be tuned to get optimal results. The pipeline encompasses the following steps, as illustrated in Figure 6:

- **Sharpen image** via an edge-enhancement convolution matrix (3×3, center: +9, all other values: -1) and **remove noise** via a bilateral filter (`BilateralFilter(d=13, sigmaColor=50, sigmaSpace=50)`)
- **Remove speckles and close holes** via a combination of `erode(kernel=3×3, iterations=1)` and `dilate(kernel=5×5, iterations=1)`. The larger dilate kernel also 'sharpens' the corners of DBT pixels.
- **Binarize brightness values** (`adaptiveThreshold(maxValue=255, adaptiveMethod=ADAPTIVE_MEAN_THRESH_C, blockSize=19, C=2)`)
- **Find grid offset**: put a virtual 6×6 grid over the image where each cell has the size of a DBT pixel. As the DBT pixels are not aligned with the origin of the sensor image, the correct offset of the virtual grid needs to be found first. To this end, for each possible X/Y offset, the area covered by the grid is extracted from the raw image and scaled to a size of 6×6 pixels. We select the offset where the contrast of this scaled-down image is highest - i.e., where the image contains the highest amount of completely white or black pixels.
- **Extract DBT pixels**: As the 6×6 array resulting from the previous step might still contain grey pixels, the values of those grid cells are determined with a two-stage threshold procedure. First, each grid cell with an average brightness of less than 60 respectively more than 195 is considered black/white. For the remaining cells, we compare their average brightness to the average of the average brightness of all adjacent cells. If it is higher, the cell is considered white - otherwise it is considered black. This approach mitigates uneven lighting and the slight differences in width between black and white lines that is common in laser printers.
- **Find position**: the extracted DBT pattern of size 6×6 is sliced into four 5×5 DBT windows. For each of the windows, its position within the DBT is determined by Shiu's algorithm and error correction is performed as described above.
- **Determine orientation** (see next section)
- **Convert DBT position into physical location** via a linear equation

[10] https://opencv.org/

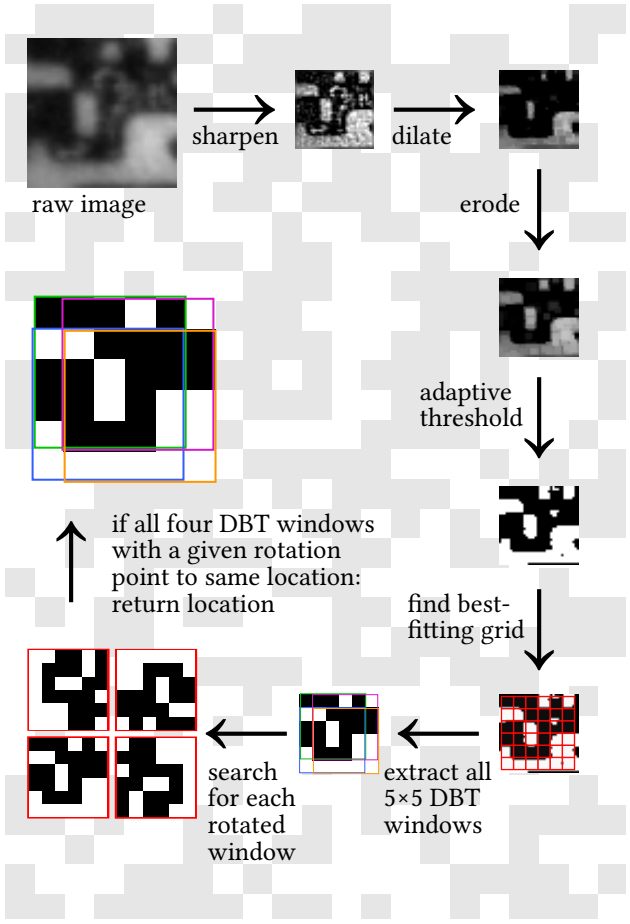


Figure 6: Image processing pipeline: the raw image captured by the PMW3360 is sharpened first. Afterwards, a dilate/erode filter combination removes noise and speckles. An adaptive threshold filter converts the image into a binary black/white image. The most probable offset of the DBT pixels within the sensor image is determined iteratively and all four distinct 5×5 DBT windows are extracted. As the tangible can be oriented in one of four directions, each DBT window is searched for in all four 90-degree rotations. For each rotation: if all four DBT windows point to the same location within the DBT, this location is converted into a physical location and returned to the tangible.

Extract DBT pixels: As the 6x6 array resulting from the previous step might still contain grey pixels, the values of those grid cells are determined with a two-stage threshold procedure. First, each grid cell with an average brightness of less than 60 respectively more than 195 is considered black/white. For the remaining cells [...]
Minor implementation details were left out and can be found in the source code.

4.4 Determining Orientation

In general, tangibles can not only be moved but also be rotated implicitly or explicitly. The user might want to orient the tangible

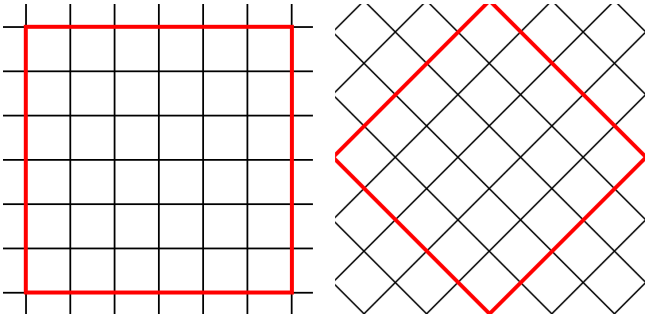


Figure 7: When rotating the tangible, fewer DBT windows are captured by the sensor. In our implementation, four 5×5 DBT windows can be extracted from the 6×6 DBT pixel raw wide image. When rotated by 45 degrees, only one 5×5 DBT window can be extracted at best. Therefore, error detection would no longer be possible.

in a more ergonomic way while interacting with it. Rotating the tangible may also be used as an input technique for changing an associated digital value.

Dothraki currently only works for the four orientations of the tangible where the axes of sensor grid and DBT grid align (0, 90, 180, 270 degree). This is caused by two limitations: First, extracting the DBT windows from the sensor image is only possible if the orientation of the sensor relative to the pattern is known. However, due to the low sensor resolution, we could not find an efficient way to reliably determine arbitrary rotations of the tangible. Applying a Hough transform does not work because continuous lines are rare in the DBT. A brute-force approach - rotating the image by multiples of one or five degree before trying to extract DBT windows was deemed too computationally expensive. Second, a non-axis-aligned raw image contains fewer DBT windows (Figure 7). Even if one could reliably determine orientation, not enough information remains for reliably determining the tangible's position.

Therefore, our implementation expects that the tangible is oriented so that the axes of the mouse sensor are aligned relatively well with the axes of the DBT pattern. Rotating the tangible by exactly 90, 180 or 270 degree keeps the sensor aligned with the pattern and allows for extracting the DBT windows. Dothraki detects these rotations by rotating the captured image by 90/180/270 degrees and checking for which of the orientations the extracted DBT windows agree on a location. We discuss options for implementing true orientation-independent position dectection in the final section.

4.5 Absolute and Relative Tracking

To improve responsiveness, Dothraki combines this abolute positioning approach with relative tracking and dead reckoning. After sending the captured sensor image to the host computer, Dothraki switches the sensor into relative tracking mode and reads movement data directly from the PMW3360 until an answer from the host computer arrives. In this mode, the sensor outputs position updates at 1000+ Hz, resulting in very low tracking latency. Once the tangible receives a new (certain) absolute position from the host computer, Dothraki uses this position anew as the starting point for further relative tracking. This behavior may be tuned to specific needs of an application. For example, Dothraki could also just determine an absolute position once and then switch to relative tracking until the device is lifted off the surface. However, a small sensor drift is inherent, so that large movements would result in large positional errors over time. A Kalman filter might also be used to combine absolute and relative position information.

5 PERFORMANCE AND LIMITATIONS

We conducted both automated and manual measurements on recognition performance and speed of our current implementation.

For the automated tests, a robotic arm (AxiDraw 3) moved the tangible between known positions on an A4 sheet of paper printed with a DBT pattern. For each position, an image was captured, rotated in four different orientations, and analysed using the processing pipeline. Due to occasional mechanical failures, datasets have different numbers of samples. However, each automatically generated dataset contains at least 1000 samples. Our image processing pipeline was then applied to each image from the dataset. Processing of single raw image took 70 ms on average on a current workstation with an Intel i7 processor. Due to our conservative algorithm, there were no false positives. However, we found major differences between recognition rates for different types of paper.

5.1 Paper Types

We tested our approach with the same DBT pattern printed on four different materials: common printer paper, coated glossy paper (ColorCopy 135g Coated Glossy), photo paper for laser printers (Avery Zweckform 120g Glossy), and clear acrylic (Figure 9). All sheets were printed by a Lexmark MS510dn laser printer at 1200 dpi at default settings.

The clear acrylic can be used as an overlay on other surfaces. As long as the lower surface reflects the IR light of the mouse sensor, Dothraki can reliably distinguish the IR-absorbing black dots printed by the laser printer. This allows for reversibly adding the tracking pattern to existing printed documents, such as maps. However, as shown in Figure 8, the black pattern makes it hard to see fine details of the image below, which must be considered when implementing concrete use cases. We tested recognition rate both for clear acrylic on an empty sheet of paper and on a sheet of paper containing an inkjet-printed image.

As shown in Figure 9, plain printer paper only offers a relatively low recognition rate of 70.9% whereas glossy paper (93.6%) and photo paper (95.6%) offer recognition rates sufficient for many interactive applications. Clear acrylic lies in the middle with 85.5% (overlay on paper) respectively 86.6% (overlay on inkjet-printed paper).

In summary, smoother paper surfaces result in higher recognition rates. For most practical purposes, photo paper seems to be the best choice.

5.2 Manual Placement and Movement

In order to determine whether manually placing the tangible on the surface or moving it around would result in worse recognition, we conducted two further small tests.

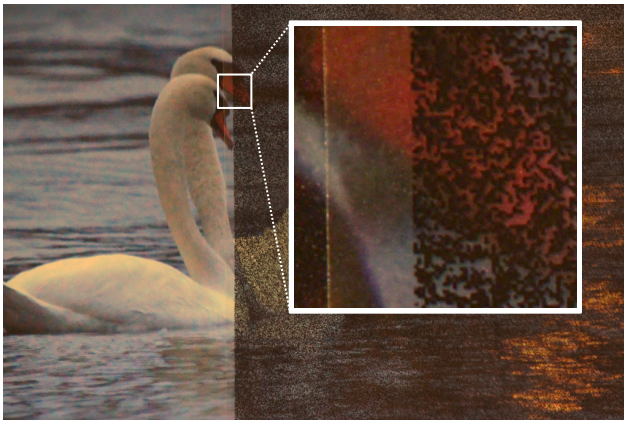


Figure 8: A sheet of clear acrylic printed with the DBT pattern can be layed over other surfaces, such as printed maps and photos. This allows for tracking tangibles atop arbitrary documents. While the image is darkened and partially obscured by the DBT pattern, salient details can still be distinguished.

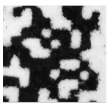
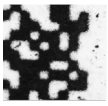
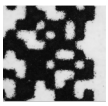
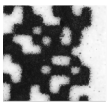
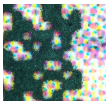
micro-scope	sensor	material	# samples	recognition rate (%)
		plain printer paper	5296	70.9
		glossy coated paper	1372	93.6
		inkjet photo paper	1598	95.6
		clear acrylic over paper	2521	85.5
		clear acrylic over print	2200	86.6

Figure 9: The type of paper on which the DBT pattern is printed greatly affects recognition rates. Common printer paper performs worst at about 70% recognition rate. Glossy and photo paper offer a high recognition rate of 93-95%. Clear acrylic offers a recognition rate of only 85%. However, the acrylic can be layed over other sheets of paper that contain inkjet-printed images. This allows for adding a removable tracking pattern to arbitrary documents on the fly.

In the first test, a user was asked to repeatedly pick up and place the tangible on different locations within a sheet of photo paper printed with a DBT pattern. They were asked to keep the tangible in parallel to the edges of the sheet without trying too hard. We captured and processed raw images from 100 placements. In 92 cases, position and orientation were identified correctly.
In the second test, a user dragged the tangible across the sheet in different directions for a short time. 204 raw images were captured. In 180 cases (88%), the location was determined correctly.
These results indicate that Dothraki is generally suitable for interactive applications where instantaneous localization is not of major importance. Given that Dothraki continuously determines its position and does not suffer from false positives, latency and recognition errors are even less perceptible during active use.

5.3 General Properties

In summary, Dothraki offers the following features and limitations:

- Low price (< $20 including assembly)
- Long-term availability of sensor
- Robust and simple hardware design
- Dot pattern can be printed with a standard laser printer
- No external hardware setup required
- Trackable area of 1370×685 mm that can be sliced into multiple smaller areas
- Can be combined with arbitrary microcontroller platforms as computation is offloaded to a host computer
- Sub-millimeter resolution
- Latency of 30 ms for reading sensor image and 70 ms for image processing results in a delay of 100 ms until a tangible's position is determined
- While the device is dragged around, absolute position is updated at 14 Hz, relative movement is updated at 1000+ Hz, depending on the microcontroller
- Recognition rate only between 70% and 95% depending on the type of paper used
- Error detection eliminates false positives
- Not covered by patents to the best of our knowledge[11]

This positions Dothraki as a scalable, low-cost alternative to capacitive or optical interactive tabletops. Dothraki can be deployed within minutes and takes up little space when not in use. Therefore, it is uniquely suited for use in classrooms, boardgames, or as an addition to traditional physical desktops. Unlike capacitive and optical tabletops, Dothraki cannot track arbitrary rotation, however. This limits the interaction space of gestures.

6 APPLICATIONS

While we see Dothraki as a generic sensing technique that can be embedded in a multitude of applications, we envision a few concrete use cases for it. In the following we describe two generic interaction techniques supported by Dothraki and present example applications that demonstrate these interaction techniques. Examples are partially based on demo applications we developed for an earlier prototype that did not yet have DBT tracking integrated [24].

[11]a recently submitted patent application [1] touches on some of the ideas behind Dothraki. However, we see no direct impact.

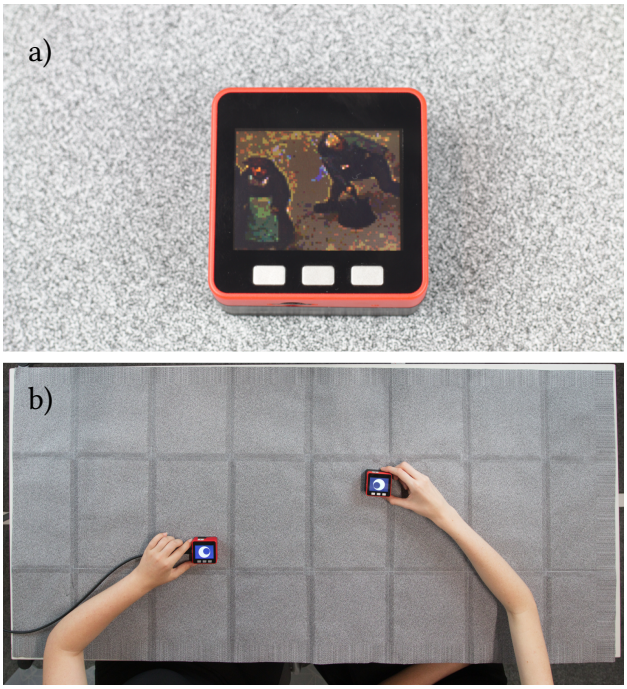


Figure 10: Example applications: a) *Magic Lens* that allows for scrolling through a virtual image by moving the tangible, b) *Googly Eyes* which always look at each other even if the tangibles are picked up and placed elsewhere.

Magic Lens (Figure 10a) allows the user to place the tangible anywhere on the surface and see a section of a larger image on the tangible's screen. By moving the tangible around the surface, the user can move the section of the image that is shown on the screen. As these relative movements are directly tracked by the mouse sensor, latency is low. Such a magic lens could be used for adding digital content to a printed map or image, hiding digital hints on the pages of a physical books, or using the tangible as a cursor for selecting objects projected onto a tabletop.
Mutual Awareness allows multiple tangibles to interact with each other (Figure 10b). In this application, the host computer not only calculates each tangible's position but also transmits the position to all connected tangibles, allowing them to react to each other. In our example, each of the two tangibles displays a stylized eyeball on its screen. The eyeballs always look at each other even if the tangibles are moved around the surface, lifted and put somewhere else, or rotated by multiples of 90 degrees. Mutually aware tangibles can be used as avatars in board games, bimanual input devices, or as building blocks in educational applications where students e.g., have to arrange tangibles in a certain order.

7 CONCLUSION AND FUTURE WORK

With Dothraki we demonstrate that it is feasible for tangibles to determine and track their position on a surface using a mouse sensor. We also highlight limitations of our approach and show examples of applications that can be supported by Dothraki. While Dothraki is ready to be used in new applications, we also see it as a baseline implementation for further improvements, such as:

- more robust position detection by implementing other approaches from the literature [2, 5], by improving image processing, or by using alternative patterns.
- orientation detection by improving image processing, by adding additional information to the printed pattern, or by choosing a DBT-like encoding which contains more continuous lines.
- invisible DBT patterns printed with IR-absorbing ink.
- local implementation on the microcontroller by porting the processing pipeline to C and optimizing code performance and storage of the DBT.
- high-speed rotation tracking by modifying the mouse sensor's firmware or by adding a second mouse sensor to the device. Modifying the firmware would require major reverse-engineering efforts or collaboration with the manufacturer, however.
- conducting a study with user groups in order to evaluate and improve practical performance for concrete use cases

In order to facilitate further development, source code and schematics are available under open-source licenses at https://github.com/PDA-UR/DotTrack and https://hci.ur.de/projects/dottrack.

ACKNOWLEDGMENTS

We thank Thomas Fischer for help with performance measurements, Florin Schwappach for ideas on image processing, and all anonymous reviewers for their constructive feedback which caused us to revisit and significantly improve our image processing pipeline. This project is funded by the Bavarian State Ministry of Science and the Arts and coordinated by the Bavarian Research Institute for Digital Transformation (bidt).

REFERENCES

[1] Aras, R. 2020. Multi-axis position sensing system. WO2020031041A1. Feb. 2020.
[2] Berkowitz, R. and Kopparty, S. 2016. Robust positioning patterns. (Jan. 2016), 1937–1951. DOI:https://doi.org/10.1137/1.9781611974331.ch136.
[3] Borsato, F.H. and Morimoto, C.H. 2016. Episcleral surface tracking: Challenges and possibilities for using mice sensors for wearable eye tracking. *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications* (Charleston, South Carolina, Mar. 2016), 39–46. DOI:https://doi.org/10.1145/2857491.2857496.
[4] Bouck-Standen, D., Schwandt, M., Winkler, T. and Herczeg, M. 2016. ELBlocks - An Interactive Semantic Learning Platform for Tangibles. Gesellschaft für Informatik e.V.
[5] Bruckstein, A.M., Etzion, T., Giryes, R., Gordon, N., Holt, R.J. and Shuldiner, D. 2012. Simple and Robust Binary Self-Location Patterns. *IEEE Transactions on Information Theory*. 58, 7 (Jul. 2012), 4884–4889. DOI:https://doi.org/10.1109/TIT.2012.2191699.
[6] de Bruijn, N.G. 1946. A combinatorial problem. *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam*. 49, 7 (1946), 758–764.

[7] Domburg, J. Sprites mods - Optical mouse-cam. https://spritesmods.com/?art=mouseeye.
[8] Fan, C.T., Fan, S.M., Ma, S.L. and Siu, M.K. 1985. On De Bruijn arrays. *Ars Combinatoria*. 19, MAY (Jan. 1985), 205–213.
[9] Fitzmaurice, G.W., Ishii, H. and Buxton, W.A.S. 1995. Bricks: Laying the Foundations for Graspable User Interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1995), 442–449. DOI:https://doi.org/10.1145/223904.223964.
[10] Fujinami, K., Kosaka, M. and Indurkhya, B. 2018. Painting an Apple with an Apple: A Tangible Tabletop Interface for Painting with Physical Objects. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 4 (Dec. 2018), 162:1–162:22. DOI:https://doi.org/10.1145/3287040.
[11] Horan, V. and Stevens, B. 2015. Locating Patterns in the De Bruijn Torus. *arXiv:1505.04065 [math]*. (Nov. 2015).
[12] Hostettler, L.O., Ozgur, A., Lemaignan, S., Dillenbourg, P. and Mondada, F. 2016. Real-Time High-Accuracy 2D Localization with Structured Patterns. *Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA)* (2016), 4536–4543. DOI:https://doi.org/10.1109/ICRA.2016.7487653.
[13] Ishii, H. and Ullmer, B. 1997. Tangible Bits: Towards Seamless Interfaces Between People, Bits and Atoms. *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, United States, 1997), 234–241. DOI:https://doi.org/10.1145/258549.258715.
[14] Jackson, J.D., Callahan, D.W. and Marstrander, J. 2007. A Rationale for the use of Optical Mice Chips for Economic and Accurate Vehicle Tracking. *2007 IEEE International Conference on Automation Science and Engineering* (Sep. 2007), 939–944. DOI:https://doi.org/10.1109/COASE.2007.4341837.
[15] Jordà, S., Geiger, G., Alonso, M. and Kaltenbrunner, M. 2007. The reacTable: Exploring the synergy between live music performance and tabletop tangible interfaces. *Proceedings of the 1st international conference on Tangible and embedded interaction* (New York, NY, USA, Feb. 2007), 139–146. DOI:https://doi.org/10.1145/1226969.1226998.
[16] Krohn, A., Beigl, M., Hazas, M. and Gellersen, H.-W. 2005. Using Fine-Grained Infrared Positioning to Support the Surface-Based Activities of Mobile Users. (2005), 463–468. DOI:https://doi.org/10.1109/ICDCSW.2005.139.
[17] Ladha, C., Ladha, K., Hook, J., Jackson, D., Wood, G. and Olivier, P. 2010. TouchBridge: Augmenting active tangibles for camera-based multi-touch surfaces. *ACM International Conference on Interactive Tabletops and Surfaces* (New York, NY, USA, Nov. 2010), 271–272. DOI:https://doi.org/10.1145/1936652.1936711.
[18] Li, M. and Kobbelt, L. 2012. Dynamic tiling display: Building an interactive display surface using multiple mobile devices. *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia* (New York, NY, USA, Dec. 2012), 1–4. DOI:https://doi.org/10.1145/2406367.2406397.
[19] Liang, R.-H., Cheng, K.-Y., Chan, L., Peng, C.-X., Chen, M.Y., Liang, R.-H., Yang, D.-N. and Chen, B.-Y. 2013. GaussBits: Magnetic tangible bits for portable and occlusion-free near-surface interactions. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, Apr. 2013), 1391–1400. DOI:https://doi.org/10.1145/2470654.2466185.
[20] Makarov, D.A. and Yashunsky, A.D. 2019. On a Construction of Easily Decodable Sub-de Bruijn Arrays. *Journal of Applied and Industrial Mathematics*. 13, 2 (Apr. 2019), 280–289. DOI:https://doi.org/10.1134/S1990478919020091.
[21] Merrill, D., Kalanithi, J. and Maes, P. 2007. Siftables: Towards Sensor Network User Interfaces. *Proceedings of the 1st International Conference on Tangible and Embedded Interaction* (New York, NY, USA, 2007), 75–78. DOI:https://doi.org/10.1145/1226969.1226984.
[22] Pettersson, M.P. 2006. Method and device for decoding a position-coding pattern. US7145556B2. Dec. 2006.
[23] Scharf, F., Winkler, T. and Herczeg, M. 2008. Tangicons: Algorithmic reasoning in a collaborative game for children in kindergarten and first class. *Proceedings of the 7th international conference on Interaction design and children - IDC '08* (Chicago, Illinois, 2008), 242. DOI:https://doi.org/10.1145/1463689.1463762.
[24] Schüsselbauer, D., Schmid, A., Wimmer, R. and Muth, L. 2018. Spatially-Aware Tangibles Using Mouse Sensors. *Proceedings of the Symposium on Spatial User Interaction* (New York, NY, USA, 2018), 173–173. DOI:https://doi.org/10.1145/3267782.3274690.
[25] Shaer, O. and Hornecker, E. 2010. Tangible User Interfaces: Past, Present, and Future Directions. *Foundations and Trends in Human-Computer Interaction*. 3, 12 (Mar. 2010), 4–137. DOI:https://doi.org/10.1561/1100000026.
[26] Shiu, W.-C. 1993. Decoding de Bruijn arrays as constructed by Fan et al. (1993).
[27] Speelpenning, T., Antle, A.N., Doering, T. and van den Hoven, E. 2011. Exploring How Tangible Tools Enable Collaboration in a Multi-touch Tabletop Game. *Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction - Volume Part II* (Berlin, Heidelberg, Sep. 2011), 605–621.
[28] Terrenghi, L., Kranz, M., Holleis, P. and Schmidt, A. 2006. A cube to learn: A tangible user interface for the design of a learning appliance. *Personal and Ubiquitous Computing*. 10, 2-3 (Apr. 2006), 153–158. DOI:https://doi.org/10.1007/s00779-005-0025-8.
[29] Ullmer, B., Ishii, H. and Glas, D. 1998. mediaBlocks: Physical Containers, Transports, and Controls for Online Media. *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), 379–386. DOI:https://doi.org/10.1145/280814.280940.
[30] Villar, N., Cletheroe, D., Saul, G., Holz, C., Regan, T., Salandin, O., Sra, M., Yeo, H.-S., Field, W. and Zhang, H. 2018. Project Zanzibar: A Portable and Flexible Tangible Interaction Platform. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, Apr. 2018), 1–13. DOI:https://doi.org/10.1145/3173574.3174089.
[31] Voelker, S., Cherek, C., Thar, J., Karrer, T., Thoresen, C., Øvergård, K.I. and Borchers, J. 2015. PERCs: Persistently Trackable Tangibles on Capacitive Multi-Touch Displays. *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (New York, NY, USA, Nov. 2015), 351–356. DOI:https://doi.org/10.1145/2807442.2807466.
[32] Voelker, S., Øvergård, K.I., Wacharamanotham, C. and Borchers, J. 2015. Knobology Revisited: A Comparison of User Performance between Tangible and Virtual Rotary Knobs. *Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces - ITS '15* (Madeira, Portugal, 2015), 35–38. DOI:https://doi.org/10.1145/2817721.2817725.

[33] Waldner, M., Hauber, J., Zauner, J., Haller, M. and Billinghurst, M. 2006. Tangible tiles: Design and evaluation of a tangible user interface in a collaborative tabletop setup. *Proceedings of the 20th conference of the computer-human interaction special interest group (CHISIG) of Australia on Computer-human interaction: Design: Activities, artefacts and environments - OZCHI '06* (Sydney, Australia, 2006), 151. DOI:https://doi.org/10.1145/1228175.1228203.

[34] Winkler, T., Scharf, F. and Herczeg, M. 2013. SpelLit - Tangible Cross-Device-Interaction beim Erlernen von Lesen und Schreiben. *Workshopband Mensch & Computer 2013* (München, 2013), 179–184.

[35] Yates, A. and Selan, J. 2016. Positional tracking systems and methods. US20160131761A1. May 2016.

[36] Zigelbaum, J., Horn, M.S., Shaer, O. and Jacob, R.J.K. 2007. The Tangible Video Editor: Collaborative Video Editing with Active Tokens. *Proceedings of the 1st International Conference on Tangible and Embedded Interaction* (New York, NY, USA, 2007), 43–46. DOI:https://doi.org/10.1145/1226969.1226978.