The Looking-Glass System: A Unidirectional Network for Secure Data Transfer Using an Optic Medium

Gal Oren[1,2](✉), Lior Amar[3], David Levy-Hevroni[2], and Guy Malamud[2]

[1] Department of Computer Science, Ben-Gurion University of the Negev, P.O.B. 653, Beersheba, Israel
orenw@post.bgu.ac.il

[2] Department of Physics, Nuclear Research Center-Negev, P.O.B. 9001, Beersheba, Israel
dlhevroni@gmail.com, guy.malamud@gmail.com

[3] Parallel Machines - Information Technology and Services Ltd., Tel-Aviv, Israel
liororama@gmail.com

Abstract. The Looking-Glass system is a unidirectional network for data transfer using an optic medium, base on the principle of transferring information digitally between two stations without an electric connection. The implementation of this idea includes one side encoding and projecting the information to a screen in high speed, and a receiving side, which decodes the information image back to its original form. The decoding is done using a unique algorithm. Also, in order to synchronize between the transmitter and the receiver sides a separate synchronization system base on video pattern recognition is used. This technique can be useful whenever there is a need to transfer information from a closed network – especially sensitive one – to an open network, such as the Internet network, while keeping the information in its original form, and without any fear of an uncontrolled bidirectional flow of information – either by a leakage or a cyber attack.

Keywords: Confidential networks · Information security · Unidirectional systems · Encrypted data transmission

1 Introduction

"Oh, what fun it'll be, when they see me through the glass in here, and can't get at me!"

Lewis Carroll, *Through the Looking-Glass* (1871)

There are two major risks in the process of data transference from a secure confidential network into an open network [1]. The first risk is based on the reasonable assumption that there are possible threats which would like to break the networks' security fence in order to steal confidential information (in the best-case scenario), or to inject a malware in order to harm the network (in the worst-case). In the latter, one can

R. Doss et al. (Eds.): FNSS 2016, CCIS 670, pp. 81–97, 2016.
DOI: 10.1007/978-3-319-48021-3_6

assume that a leak of secure information is possible due to metadata which was embedded into the transmitted data without the knowledge of the transmitter.

In many cases, the way those risks are handled is by an act of avoidance from any digital transmission of data from a secure network to an open network, when the alternative is by exporting a physical copy of the file by printing it on a plain paper. If it is necessary to get a digital copy of the data, the printed paper is scanned as an image, and by a usage of the Optical Character Recognition (OCR) technique it is been transformed back to a digital form in the open network [1]. This kind of solution holds many technical and immanent problems, which make the whole process ineffective at the best-case scenario because of its mechanical fashion, or even unreliable in the worst case because of the OCR algorithm inherent imperfect capabilities.

The Looking-Glass system supplies a technological solution to the current mode of work while supplying a secure solution to the two major risks of digital data transfer discussed above. The implementation of the idea includes a high speed transmitting side (i.e. transmitter), encoding and projecting the information to a screen, and a receiving side (i.e. receiver), which receive the information by remotely filming the screen and decoding the image information back to its original form, using unique image-processing algorithm. In order to synchronize between the transmitter and the receiver, we built a separate synchronization system based on video pattern recognition. Figure 1 shows a flow graph of the systems' activity, from the start of transmission until its end.

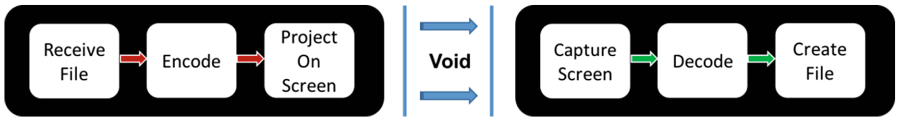


Fig. 1. The principles of information transport in the Looking-Glass system.

The system allows only a flow of ASCII characters from one network to the other without any physical channel that connects the two networks. This transference is done solely in a transfer only mode – meaning a 100 % secure system from any malware injection, simply because there is no bridge into it (in differ of the current solutions such as a unidirectional optical-fiber cable [2] or the Pump invention by the American Naval Forces [3]).

In addition, the fact that the system transfer ASCII characters explicitly means that there is no risk of a secure metadata hidden in the data file, simply because there are only characters and not a file. Also, in order to maintain a high reliability, several quality control features and optimizations were embedded in the system algorithm in order to prevent the inherent problems of unidirectional data transmit.

In this paper we discuss the Looking-Glass system and its algorithms. Section 2 will introduce the physical system and the principles of the development of the code written for the transmitting and receiving sides; Sect. 3 will elaborate and describe the encoding and decoding methods that were implemented at the transmitting and receiving sides, respectively; and at last, Sect. 4 will describe the system's benchmarks.

2 The Physical System

The physical infrastructure of the Looking-Glass system includes usage of two physically separate computers, one as a transmitting unit and one as a receiving unit. On the transmitting side, a computer has been installed with two screens (24 inch, 1080p) – a screen for monitoring at the front and an internal screen inside the box to project the encoded data to the cameras at the receiving side. At the receiving side, a computer has been installed with one screen for monitoring at the front, and two cameras inside the box to film the transmission screen – an SLR (Single-Lens Reflex) camera and a valid Video camera (30 frames per second) to track and monitor the stream of transmitting frames. The units were separated 1-meter away from each other – more than the sufficient distance in order to capture all of the transmitting screen.

In order to prevent transmission of data using the magnetic field between the computer of the transmitting unit (which connects to the secured confidential network) and the computer of the receiving unit (which connects to the open network) [4–6], the two computers have been placed in two separated and sealed Faraday cages on a mobile facility, designed for the Looking-Glass system in a modular fashion base on ITEM profiles, 40 $\times$ 40 cm^2. The sealed box was chosen to maintain a constant quality

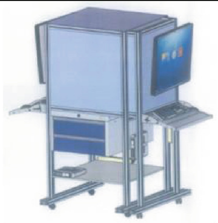
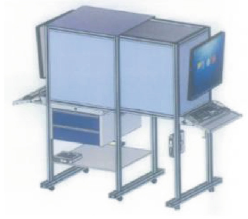
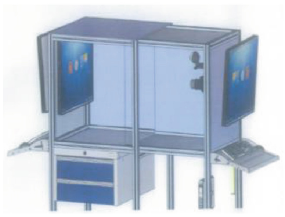
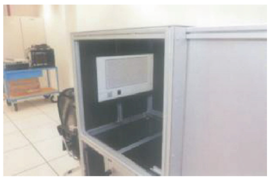
Description	System Sketch	System Pictures
The Looking-Glass system in closed mode. The system is not active at this mode.		
The Looking-Glass system in open mode. The system is active at this mode.		
An internal look into the Looking-Glass system.		

Fig. 2. Sketches and pictures of the Looking-Glass System.

of frames filming, without dependence on external lighting. This box can be opened and closed using tracks, and is installed on a mobile stand as described in Fig. 2.

The system code was developed in Python 2.7, using the OpenCV library [7], on Ubuntu operating system, and which all were installed on the computers.

2.1 The Transmitting Side

The implementation of the file transmission process is described in the algorithm below. In the first step, the data is divided into n equal segments of m characters in length, when the last segment will be equal to or less than the set size. The algorithm is performed $n + 1$ times, including one calibration segment. In every such action, one segment of the data is encoded, signed and presented to the screen.

Transmitting side – the algorithm for transmitting.

```
1. Divide the data to $n$ segments of $m$ characters.
2. Transmit the calibrated matrix of colors and
   shapes.
3. $i$ = 1.
4. While $i$ <= $n$:
   4.1   Encode segment $i$ according to the scheme.
   4.2   Sign the segment.
   4.3   Export the following to the transmission
         monitor:
         4.3.1 The matrix of the encoded $m$ characters.
         4.3.2 The signature embedded into a QR code.
         4.3.3 The mark of screen substitution or end of
               screen transmission.
         4.3.4 The textual information about the encoded
               segment.
   4.4   $i$ = $i$+1.
```

Figure 3 shows a scheme of the transmitting monitor placed inside the box. In the top part, it is possible to see the m characters encoded into a matrix of squares in different colors. At the bottom of the figure, it is possible to see the control area, where the QR code is shown (which represents the signature of the encoded segment) and the geometric figure that is used as a feedback for frame transfer, or the starting and ending of a transmission (all steps will be explained next).

2.2 The Receiving Side

The receiving side includes two processes: filming and decoding. The filming process was implemented using a technique that is initialized automatically after initializing the transmission process. During the process execution, progress is reported to the monitoring screen.

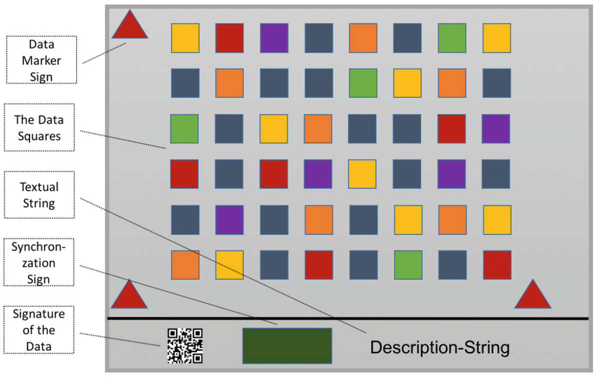


Fig. 3. The scheme of the transmitting side.

The algorithm developed for the receiving side is base on a usage of two cameras: an SLR camera, which performs the filming of the screen, and a Video camera, which continuously tracks the control area of the transmitting screen and provides a feedback to the filming process. The difference between the cameras is base on their quality of resolution: The video camera, which works in video mode, gives a low quality image, sufficient for real time decoding, while the SLR camera captures a high quality image, which is needed for the capture of large volumes of data, and for the quality assurance of the decoding algorithm. The frames filmed using the SLR camera are saved and decoded, while the images of the video camera are decoded but not saved; they are sampled through the entire duration of transmission until the end of the entire process, and are decoded solely in order to determine the status of the transmitted frame (substitution or end of transmission).

The identification mark chosen is a transfiguration of two geometric shapes, from a triangle to a square, which can be captured by the video camera. Using the change of shapes on the screen, a mark is given for filming and decoding of another image. Using another mark, a quadrilateral, it is possible to indicate that the process has reached its end (signaling the termination of the processes at the receiving side). The filming process is executed continuously as long as no last-image mark has been recognized by the Video camera, as shown in the algorithm below.

Receiving side – the algorithm for shooting pictures.

```
1. While the video camera hasn’t recognized end-of-
   transmission configuration:
   1.1  Check the image transfer mark:
      1.1.1 If there was a shape transfiguration:
         1.1.1.1   Capture the transmission screen,
                   decode it, and save the results.
2. End process.
```

3 Encoding and Decoding of Information

3.1 The Process of Information Encoding

The Looking-Glass system encodes each byte of information using color squares, such that each byte of information is converted to a separate collection of squares. The number of squares that can be displayed on a single screen is a function of the squares total size and the size of the area where the squares can be displayed. Given the number of squares that can be displayed on a single screen, the number of displayed bytes per frame is the number of those squares divided by the number of squares per byte. Given the number of bytes per frame, the data is divided into segments of this length. After dividing the file into segments, the system encodes each segment into a sequence of squares. Each byte is encoded to a number of squares (base on the system configuration), the squares are presented as a matrix – from left to right and from top to bottom – and the color of each square is selected from a palette of defined colors whose size is the number of colors that the system is requires to differentiate between. There are two colors that are not part of the information encoding: black (0, 0, 0) and white (255, 255, 255). The black color is used for mark of boundaries, and the white color is used for background. An example of encoding the words “The Dog” can be shown in Fig. 4.

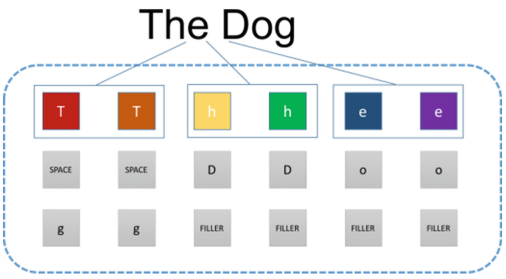


Fig. 4. Encoding the words “The Dog” into color squares (The letters are for demonstration).

The figure shows 3 lines of color squares, and every line has 6 squares. Each character is encoded using 2 squares, and the string “The Dog”, which is composed of 7 characters (including one whitespace), is encoded to 14 information squares, and 4 additional squares are added as placeholders.

During the system development, we chose to code each byte of information using two color squares, such that each byte of information is converted to a separate collection of squares. These two squares provide a more than the sufficient amount of alphanumeric characters necessary for possible ASCII text file encoding. The squares color panel contains 16 colors that provide a sufficient distance in the color spectrum to successfully differentiate between the different colors in the decoding process. Therefore, in order to encode each character of the ASCII code, we use two squares of 16 different colors (16×16 combinations), where each square’s color represents a number between 1 and 16. This encoding method was chosen to make the conversion process from image back to digital information as simple as possible. This format allowed representing an encoded data in size of 3–4 kB on the transmitting monitor, with 80 lines and 90 columns of squares. By that, one obtains a decoding ability at a very high level of reliability. Obviously, these encoding parameters can be vastly larger than in this prototype, and they are depends, as previously mentioned, on the SLR camera and the transmitting screen resolution.

The reason for choosing this encoding method rather than transmitting an image of the plain text is due to the following: if one had transmitted the information as raw data into the screen, and then try to decode it after filming from a significant distance (which creates the effects of filmography such as curvature and changing lighting), one wouldn't have obtained the high reliability we currently achieved using the squares encoding method, to which we made many image processing optimizations in order to adjust it to a remote screen filming (as will be explained in the hereby section). It is possible that a high reliability could have been obtained by filming the information in a raw form and decoding it using OCR algorithms, but only at the cost of enlarging the fonts sizes, which would have made the amount of information per transfer critically smaller – a state which eventually would cause to a significant slowdown of the whole system functionality.

3.2 Decoding the Information Area

After filming the transmitting screen, the receiving side decomposes the image structure into two parts and decodes each part separately. The parts are the information area and the control area, while the main phase is the decoding process of the former. This area consists of color squares, aligned line by line, meaning that by the end of the decoding process, we should obtain a list of all of the square's color indexes, when the order in this list is according to the order of appearance of the squares in the screen from left to right and from top to bottom. The decoding of the information area includes the following steps:

1. Converting the image to black-and-white scale and eliminating noise.
2. Accurately marking the squares matrix boundaries.
3. Identifying the squares correct order of appearance in every line.
4. Using the calculated squares position, average the real color for each square.

In the following sub-chapters we will review each of those steps and its algorithms.

3.2.1 Converting to Black-and-White Scale and Eliminating Noises

The image-processing algorithm that recognizes the squares locations needs an image where the squares are marked in black and the matrix spaces boundaries are marked in white. In order to obtain such an input image, we needed to convert the original image of the information area to solely shades of black-and-white scale, and to clean out all noises. The sub-steps in this process include converting the image to shades of gray; using a *Threshold* algorithm to convert the image to black and white where information squares are marked black and spaces are marked white; and cleaning noises, which are black spots that do not represent information squares. The first sub-step conversion process can be seen in the following comparison (Fig. 5).

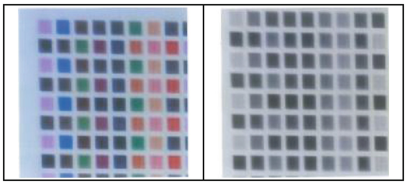


Fig. 5. The square matrix in color (left) vs. in shades of gray (right).

The next step is converting the gray-scale image into black-and-white scale image. In order to achieve that goal there is a need to use a *Threshold* algorithm [8], which can provide separation into black and white using a threshold value, where anything higher than this value is mapped to white, and anything lower than this value is mapped to black. However, when the information area is represented by a small image (a few hundred pixels in every direction), this technique works well, but when the image is large (as in our case), using a single threshold value to perform separation is not sufficient due to changes in the values that represent the spaces between the squares in the various areas of the image. For instance, in the shades-of-gray image in Fig. 5, the values of the pixels of the spaces is varying from values of 165–175 in one area of the image, and values of 170–180 in another area. In addition, we found that in occasion the values of color squares in a specific area are matching to the range of the space values in another area. The three following images (Fig. 6, from left to right) show the results while using the *Threshold* algorithm statically with values of 160, 170 and 180. It is possible to see that using a threshold value of 160 results in squares that completely deleted, when few squares attached to each other. It is also possible to see that although using a threshold value of 170 is not causing the information squares to disappear, it is causing them to become inseparable. Lastly, a usage of a threshold value of 180 shows that many squares turned into one black chunk. Therefore, we can conclude that it is not simple to find a sufficient threshold value which results a well defined black-and-white squares matrix.

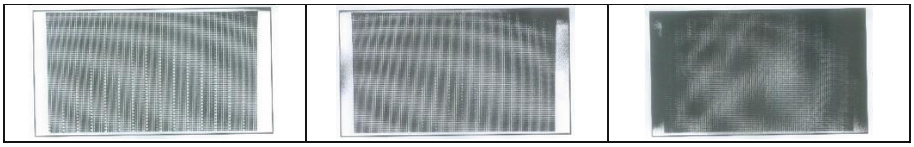


Fig. 6. Performing *Threshold* algorithm with values of 160 (left), 170 (middle) and 180 (right).

The solution for this problem is given by using the adaptive version of the *Threshold* algorithm. This version does not use a single value, but maps the area around a pixel in order to determine the threshold value. An example to this algorithm usage can be seen in the left side of Fig. 7, which shows an excellent separation of the information squares, without any disappearing squares and without inseparable squares zones. In order to calibrate the *Adaptive Threshold* function [8] we used empirical checks. Nevertheless, the image still contains black pixels originating from noise or environmental causes. The right side of Fig. 7 shows a zoom-in of the results after the *Adaptive Threshold*, and shows black pixels above the squares, which are not unified in large groups, and are not around the squares size.

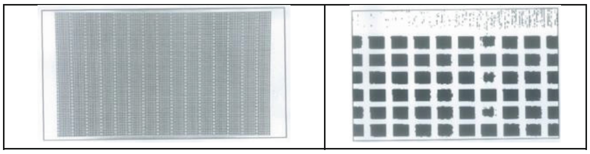


Fig. 7. The frame after using the *Adaptive Threshold* algorithm (left) and a zoom-in (right).

However, the next decoding algorithms still require a clean image. In order to clean the image, we will use two known filters – *Erode* and *Dilate* [8]. The *Dilate* function schematically calculates the maximal value for the area around the pixel, and changes the pixel value to this value. This operation lessens the black area, and hence makes single pixels, or a small amount of pixels combined together, disappear. The *Erode* function calculates the minimal value in the area around a pixel and changes the pixel's value to this value. This operation makes the black areas thicker, and hence thickens the square's boundaries that were earlier diminished by the *Dilate* operation. The three following images (Fig. 8, from left to right) respectively show the obtained noise after applying *Adaptive Threshold*; the noise elimination by *Dilate*; and thickening the squares back using *Erode.* At the end of the *Erode* operation, we obtain a clean image, on which the decoding process can be applied.

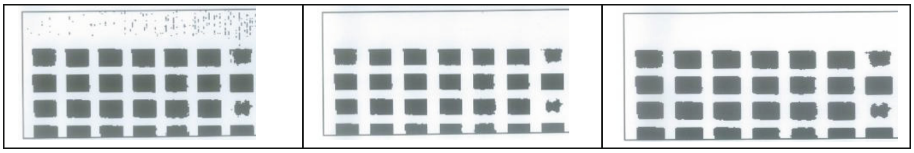


Fig. 8. Information squares with noise (left); after *Dilate* (middle); and after *Dilate* and *Erode* (right).

3.2.2 Marking the Matrix Boundaries

After converting the image using *Adaptive Threshold* and eliminating background noises, the obtained image contains the exact locations of most of the squares in a clear form. However, some of the squares, especially those containing light colors, are not completely hermetic and occasionally marked in more than one black area. In order to fix this situation – which does not enable correct recognition of the squares – we perform a process of identifying the separation lines (the matrix boundaries) between the different rows and columns of the squares matrix. The premise of this process is that even though there are squares that are not marked accurately, the most of the squares are intact and in a hermetic form, and they can be some good indicators to a total reconstruction of the original matrix boundaries. Using this knowledge, we can generate a new image where the squares are accurately defined. This algorithm is applied in the following form:

Marking the matrix boundaries algorithm.

```
1. Let SQ_ZONE, a matrix representing the image after its
   conversion to black-and-white-scale and after
   performing an eliminating noises process.
2. Let GRID, a matrix set with zeroes (representing
   black) with the same dimensions of SQ_ZONE.
3. Let N, a parameter that determines the number of
   continuous white pixels required to decide if a
   scanned segment is a matrix boundary.
4. For every line in SQ_ZONE, do:
   4.1. Measure the longest segment of continuous
        white pixels (CWP).
   4.2. If running into a black pixel (square/noise):
        4.2.1. Check whether CWP is approximate to N.
               4.2.1.1. If yes, paint the CWP segment in the
                        GRID matrix in white, as a boundary.
               4.2.1.2. Otherwise, call 4.1.
```

The three following images (Fig. 9, from left to right) show the state of the image after the previous cleaning process; the result after scanning the rows; and the result after scanning the columns (by a 90 degrees' rotation for a scan as for rows) with overlap with the previous results. It is possible to see that the squares positions are marked in a hermetic form. It is clearly seen that some damaged squares boundaries, such as square [3, 7], were reconstructed hermetically.

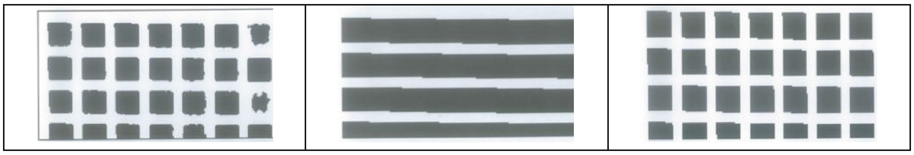


Fig. 9. The state of squares after the cleaning process (left); after scanning the rows (middle); and after scanning the columns with an overlap with the scanned rows (right).

Another example of this boundaries re-mark process can be seen in Fig. 10 which shows another area of the original image where the squares were marked inaccurately, and occasionally even discontinuously (meaning, with a few disconnected black spots on the same area of a square). It can be seen that the re-marking process had fixed the bisected squares, and that after the process they were marked accurately.

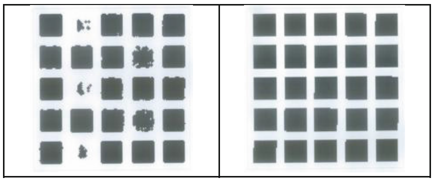


Fig. 10. Segment of information with faulted, split information squares (left) vs. the information squares after redraw (right).

3.2.3 Discovering Squares Correct Sequence by Walking-on-Line Algorithm

After resulting a black-and-white scale image with the squares appearing without noises and after a full marking process, the next step is to recognize the squares sequence and to store the squares in a data structure in the order they appear in the image. In order to do so, our Walking-On-Line algorithm is base on a simple fundamental which state that given a black pixel, it is possible to find the square shape boundaries that containing the pixel. This fundamental exists in the OpenCV library [5] and is implemented using the *FloodFill* algorithm. The basic idea underlying this square recognition algorithm is to start the scanning process from the top left corner, and scan from left to right and top to bottom. Each time the algorithm runs into a black pixel, the square containing the pixel and its connected pixels are recognized. Additionally, all of the pixels in the discovered square are set with a unique index, base on their location, in order to differentiate between the discovered squares and the squares that already been discovered. Using this method, when the next line of pixels is scanned and the algorithm runs into a pixel that belongs to an already-recognized square, the algorithm would skip it and not recognize the same square twice. In this recognition fashion, squares are found according to their order of appearance.

Seemingly, this algorithm does solve the problem; however, we note that the algorithm is based on the assumption that a line of squares is a straight line. This was found to be incorrect when filming long lines of squares. The camera lens creates a distortion, which makes the received image have iris curvature [9]. Additionally, the camera positioning might be un-aligned to the filmed screen perfectly, which makes the lines appear tilted. An example for this curvature can be seen in Fig. 11. The figure shows a green line starting at the left part of the second line of the squares. It can be seen that this line (a straight, balanced line) reaches the upper pixels of the first lines in the image shown, meaning that some squares of the first line will be recognized only after some information squares of the second line have already been recognized – a situation which cannot be tolerable.

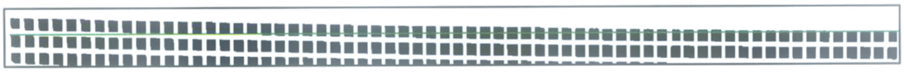


Fig. 11. The curvature in the information squares is highlighted by the balanced green line. (Color figure online)

The difficulty introduced by lines curvature can be dealt by assuming that the height difference between two squares within the same line is negligible, e.g. two or three pixels between each square at most. For the current system design, this assumption enables recognition of the information squares adaptively at every image decoding (since the curvature is not always identical). By using this method, the problem can be overcome. The recognition of squares sequence is performed using the algorithm below.

Walking-on-Line Algorithm.

```
1. Scanning the pixels in the image from left to right,
   top to bottom.
   1.1.    When running into a black pixel:
      1.1.1.     Calculate the boundaries of the square
           that contains the pixel, compute its center,
           and save the results.
      1.1.2.     Walk right from the calculated center
           of the square until running into a black pixel
           which not belongs to the current identified
           square, or until reaching the end of the line.
         1.1.2.1. If a black pixel found:
            1.1.2.1.1.  Call 1.1.1.
      1.1.3.     If reaching the right end of the line:
         1.1.3.1. Continue to the next line.
         1.1.3.2. Call 1.1.
2. When there are no more pixels to scan:
   2.1.    End process.
```

Figure 12 shows an example of the algorithm actions in order to recognize the squares correct sequence. First, the top left pixel of the first square in the line is recognized (marked in red). The arrows show the process of recognizing the sequence, starting from the recognized pixel, towards the center of the square, and finally to the right, until a new square is recognized. By using this method, the heights of the scanned squares are continuously fixed, and the squares are captured based on their order of appearance. At the end of the recognition algorithm's run, all of the squares are obtained and ordered in the order of their appearance, and they are finally ready to be decoded back to the ASCII characters they represent.

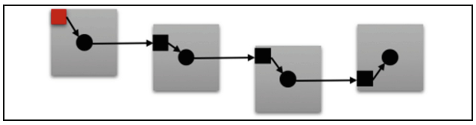


Fig. 12. Recognizing the squares correct sequence in an adaptive scanning. (Color figure online)

3.2.4 The Color Decoder and Its Calibration

After achieving to get the correct boundaries of the squares and its correct sequence, the remaining step is to convert each original color of each square into the number the color represents, and then back to its ASCII code. As previously explained, every character – with 8-bit limit in this prototype – is encoded into two squares, meaning every square represents four bits (which represent 16 values). Therefore, 16 different colors are in need, when every color of every square represents an index between 1 and 16. The task for the color decoder is to convert a color sample in RGB format from the correctly marked square to an index between 1 and 16.

However, after filming the screen, the obtained color for each square was found to be not identical to the color the square originally painted in at the encoding side. Furthermore, it has been found that some colors located in different areas of the transmitting screen were decoded with different coloring, even though the squares were originally painted in the exact same color. This phenomenon is caused by lighting conditions, which can be different in each part of the screen, as well as optic distortions that affect the color sample. For example, a color square that was originally painted at the encoding side with the color [240, 163, 255] was sampled at the decoding side with three different values of [140, 106, 205], [144, 109, 210], and [145, 111, 211] at three different zones of the frame. Nevertheless, based on the measurements it seems that there is no great spatial proximity between the sampled values; however, these values – as well as the rest of the samples – are relatively close to each other on the color spectrum, meaning that the distortion in the conversion of the original color does not provide a great scattering of the values.

Either way, the color decoder needs to take this phenomenon into account and overcome it. This is done by calibrating the decoder by color 'zones', and not by a 'one-to-one' value, meaning that although a color can be decoded with a deviation from the original RGB representation, the algorithm still will be able to recognize that a distorted RGB representation represents a specific color which have an ASCII index translation. In order to achieve this goal, the transmitting side encodes a calibration image as the first frame, containing squares which represent all of the 16 different colors in all of the different parts of the frame, and which the decoding algorithm at the receiving side knows all of its precise original RGB representations and indexes, base on its locations. Then, in order to represent all of the original indexes in all of the possible zones, a 3-dimentional matrix of $256 \times 256 \times 256$ values – which represents all of the RGB spectrum – is set with the original indexes at each of the RGB representations, including all of its nearest neighbors. For example, if it is known that the green color index was set to be 7, and if it was found that the decoding of the color square which represented this color at a specific zone in the frame was a RGB representation of (76, 153, 0), then the value of those indexes and of all of its nearest neighbors will be set to be 7 in the 'translation' matrix for further decoding of the colors of the squares (exemplification in Fig. 13).

Note that since the chosen 16 colors are relatively far from each other on the spectrum, it is reasonable to assume that there will be no overlap between the squares' RGB representation in the three-dimensional matrix.

The Color Detector and its Calibration Algorithm.

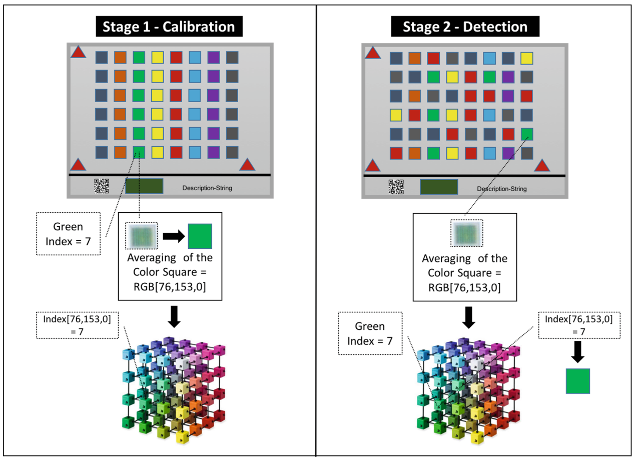


Fig. 13. Example for square decoding: Calibrating the green color (on the left) and using the calibration matrix to decode it (on the right). (Color figure online)

```
1. Set the matrix ORIG_CALIBRATION_MAT with the original
   color indexes of the calibration matrix.
2. For every square in the calibration decoded frame
   (DECODE_CALIBRATION_MAT):
   2.1.    Calculate the square average color using RGB
       fashion, such that the R represents a X coordinate
       index, the G represents a Y coordinate index, and
       the B represents a Z coordinate index of a 3-
       dimentional matrix (CALC_CALIBRATION_MAT) of
       256X256X256 (the RGB spectrum).
   2.2.    Set the index of the original color from
       ORIG_CALIBRATION_MAT in CALC_CALIBRATION_MAT base
       on the R, G, and B indexes which calculated in 2.1
       from DECODE_CALIBRATION_MAT, as well as in all of
       its nearest neighbors in the 3-dimentional matrix
       (a total of 27 squares that will have the same
       index).
3. For every square in the next decoded frames:
   3.1.    Calculate the square average color using RGB
       fashion.
   3.2.    Access CALC_CALIBRATION_MAT using the RGB
       indexes and check the color index which was
       previously set.
      3.2.1.    If index found:
         3.2.1.1. Return index.
      3.2.2.    Else, if no color index has been found:
         3.2.2.1. Start a rotational scan using the
                  Nearest Neighbor technique until finding
                  the color's matching index.
```

3.2.5 Decoding the Control Area and Quality Control

The decoding process of the control area (Fig. 14) consist of three different, independent parts:

Fig. 14. The information area. From right to left: the frame's title, the recognition square for frame switch, and the QR encoding square.

1. The first part focuses on the square and rectangle, which appears alternately. These shapes alternation indicate the receiving side on a change of the encoded information, meaning a new information transmission has been done, and there is a need to capture the new frame. A termination of the process is identified when a quadrilateral is being detected.
2. The second part focuses on the quality control aspect of the process, and includes decoding a QR code. As known [10], the QR code represents an accurate collection of characters (in this case, 256 or 512 characters, depending on the size of the square and the type of encoding). Using this knowledge, the encoding side takes the textual information which needs to be transmitted as a matrix of squares and uses a cryptographic hash function to obtain a checksum of this information, base on *sha256-512*. This checksum is encoded by the program into the QR square. Using OpenCV functions [7], this part of the control area is identified and decoded at the receiving side. When the receiving side has both the Checksum of the encoded information and the information itself as decoded by the decoding algorithm, the quality control algorithm encodes the information using the same cryptographic hash function and compares it to the checksum extracted from the QR square. If the comparison of the two checksums is found to be identical, it means that the information transfer has been completed successfully. Otherwise, it means that at least one character was missed or transferred incorrectly. This method ensures the completeness of the information transfer in an 100 % certainty (assuming checksum reliability).
3. The third part is an accurate documentation of the frame's name. This information is derived and saved in the folder the frame is saved in for logging purposes.

4 Performance Tests

Performance tests for the Looking-Glass system have been executed for three different stages of the system operation: the encoding stage, the transmission and receiving stage, and for the decoding stage. The distribution of time for a standard 3 kB frame in the system is shown in the following three steps. The speed of encoding, transmitting and receiving, and decoding of a single frame is currently stand on $\sim$3 s.

1. **The encoding step:** the data is processed into squares which instantly displayed on the screen. This step is relatively faster than the other steps, and currently stand on less than a second.
2. **The transmission step:** the data is transmitted from the transmitting side to the receiving one. This step completely depends on the rate of the filming process (i.e. cameras properties) and the rate of the saving process (i.e. receiving computer properties), and it is currently stand on less than a second.
3. **The decoding step:** the data is extracted from the received frame, and a verification process confirms that all of the data that have been transformed back to the original digital form successfully. This whole process currently stands on less than two seconds (and this time factor can be significantly improved in the future using parallel processing at the decode process).

The three steps have been tested separately in 20 experiments. In order to verify the system's stability and reproducibility factors, a script was written to generate 20 simple, textual data files with random content, with sizes that varied from 10 kB to 100 kB, and from 100 kB to 1 MB – sizes which represents the common range of data sizes the system will need to transfer. The files were transferred continuously, and benchmarks was measured for each data transfer. It was found that the Looking-Glass system performances was linear to the amount of submitted data, as expected.

5 Conclusions and Future Work

The work presented in this paper lead to the conclusion that it would be beneficial to use the Looking-Glass system in order to securely and reliably transfer information digitally using an optic medium. Also, this paper opens a number of prospective directions for future research, which one immediate direction is to explore how to optimize the system infrastructure using new hardware, and how to increase the performances of the suggested algorithms, mainly using parallel processing.

Acknowledgments. This work was supported by the Lynn and William Frankel Center for Computer Science.

References

1. Shabtai, A., Elovici, Y., Rokach, L.: A Survey of Data Leakage Detection and Prevention Solutions. Springer Science & Business Media, New York (2012)
2. Okhravi, H., Sheldon, F.T.: Data diodes in support of trustworthy cyber infrastructure. In: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research, p. 23. ACM, April 2010
3. Kang, M.H., Moskowitz, I.S., Chincheck, S.: The pump: A decade of covert fun. In: Computer Security Applications Conference, 21st Annual, p. 7. IEEE, December 2005

4. Kuhn, M.G., Anderson, R.J.: Hidden data transmission using electromagnetic emanations. In: Kuhn, M.G., Anderson, R.J. (eds.) Information Hiding. LNCS, vol. 1525, pp. 124–142. Springer, Heidelberg (1998)
5. Kramer, F.D., Starr, S.H.: Cyberpower and National Security. Potomac Books Inc, Lincoln (2009)
6. Zhao, N., et al.: EMI Spy: harnessing electromagnetic interference for low-cost, rapid prototyping of proxemic interaction. In: 2015 IEEE 12th International Conference on Wearable and Implantable Body Sensor Networks (BSN), IEEE (2015)
7. Suarez, O.D., Carrobles, M.D.M.F., Enano, N.V., García, G.B., Gracia, I.S., Incertis, J.A.P., Tercero, J.S.: OpenCV Essentials. Packt Publishing Ltd., Mumbai (2014)
8. Petrou, M., Petrou, C.: Image Processing: The Fundamentals. Wiley, New York (2010)
9. Goldberg, N.: Camera Technology: the Dark Side of the Lens. Academic Press, Boston (1992)
10. Furht, B. (ed.): Handbook of Augmented Reality. Springer Science & Business Media, New York (2011)