Embedding metadata in images at time of capture using physical Quick Response (QR) codes

Gareth Hill

The New Zealand Institute for Plant and Food Research, Private Bag 92169, Auckland 1142, New Zealand

Mark Whitty*

School of Mechanical and Manufacturing Engineering, UNSW Sydney 2052, Australia

Abstract

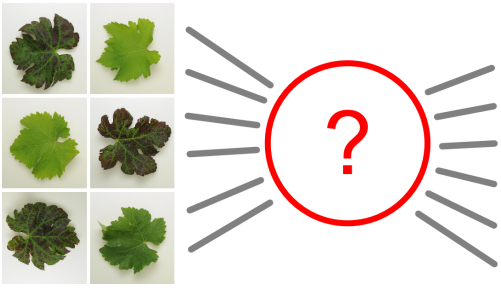
Maintaining metadata records for scientific imaging is challenging where the link between the metadata and the image is labour intensive to create and can easily be broken. We propose a method for using QR codes in images of samples to embed the metadata, so that it can be extracted on demand. By using a novel pipeline for generating QR codes, displaying them in images, reading the QR codes in the images and extracting the metadata for later action such as renaming the image file, a streamlined process for metadata management is introduced. This method was simulated using a range of image types and QR code parameters to identify the limits of various parameter combinations, providing practical insight into code design and usability. The pipeline was also tested with hundreds of images in both laboratory and field situations and proved to be extremely efficient and robust. This method offers potential for anyone taking images of samples who needs to guarantee the existence and correctness of metadata without relying on an external association mechanism.

Keywords: QR code, metadata, scientific imaging, information management, image processing

*Corresponding author
Email address: m.whitty@unsw.edu.au (Mark Whitty)

Introduction

Metadata is vital for putting an image in context in cases where the contents of the image are not immediately obvious, where the image is part of a digital catalogue or where the image is being used to train a machine learning algorithm (Reser & Bauman, 2012; Frisch, 2013; Greenberg et al., 2014; Saleh, 2018). There are some common problems with how metadata is traditionally collected and stored: insufficient or incorrect metadata, vague or non-defined metadata, or metadata that is lost or not readily located when needed (Smith et al., 2014). Many images have little value if the associated metadata is not recorded and stored in a readily accessible manner, particularly in research. While that metadata could be stored in a database or with an associated file of some description and referenced to the filename, there is a risk that the image or metadata file could be copied or moved to a distinct storage location and the link between image and metadata becomes broken or lost (Smith et al., 2014). Even the initial data association step (as in Figure 1) is commonly time consuming and fraught with human error. Because of this, there is an increasing awareness that metadata should be embedded in the file itself, typically through the use of structured data containers (Reser & Bauman, 2012).



	Variety	Row	Block	Diseased	Size [mm]
Sample 1	Merlot	10	B1	Yes	95
Sample 2	Merlot	22	B2	No	87
Sample 3	Pinot Noir	19	B21	Yes	67
...	...	...	...	...	...
...	...	...	...	...	...
Sample N	Chardonnay	5	B22	No	95

Figure 1: Initially associating images with sample metadata is time consuming and fraught with error.

The most prominent standard for using data containers to embed metadata in image files is the Exchangeable image file format (EXIF) (JEITA, 2002),

commonly used in JPEG images. This metadata format has some limitations as it is only available for some file types and the fields are non-extensible and therefore may not be suitable for all applications (Reser & Bauman, 2012). A new metadata format, the eXtensible Metadata Platform (XMP), was developed in response to these issues and intended to be format-agnostic and extensible; it has since become an international standard (ISO, 2019). However, any metadata embedding system that relies on a particular storage format within an image file outside of the pixels that make up the image itself is at risk of metadata loss if the file is copied to a new location or operating system, or saved as a new file or alternative file format without explicitly preserving the metadata. Such situations are commonly encountered when uploading images to cloud storage or social media platforms, but can also occur through manual image manipulations or through automated pipelines. For example, image processing pipelines commonly use the OpenCV software library (Bradski, 2000), which only saves the pixel information in the processed image by default, resulting in a loss of any metadata embedded in a data container, as illustrated in Figure 2.

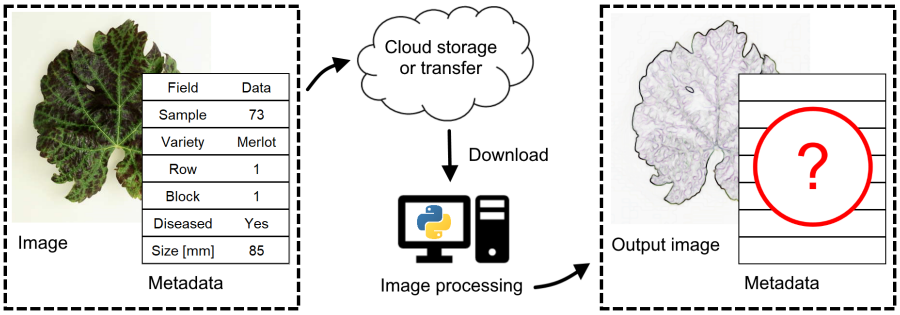


Figure 2: Metadata is commonly stripped out by cloud storage systems and is also lost whenever images are output from an image processing pipeline.

Attempts have been made to standardise not only the metadata, but how it is recorded and stored, such as the International Press Telecommunications Council's (IPTC) 'Embedded Metadata Manifesto' (International Press Telecommu-

nications Council, 2019a) that outlines five guiding principles for embedding metadata, and the Smithsonian's more detailed guidelines for minimal descriptive embedded metadata (Christensen et al., 2010). The IPTC have also issued a metadata standard (International Press Telecommunications Council, 2019b) that has since become the most commonly used schemas in a range of industries and communities (Smith et al., 2014). These guides and standards were primarily developed with journalistic images and museum collections in mind, and indeed a lot of the push for embedded metadata comes from either journalism or heritage collections (Frisch, 2013; Saleh, 2018); however, proponents of embedded metadata emphasise that the practice would have benefits in other areas (Smith et al., 2014).

One such area is maintaining scientific and experimental images for research, in which embedded metadata could benefit machine learning applications, where large volumes of images are required for training. The traditional approach for embedding metadata has been to include a label in the image describing what is in it (e.g. species name, date collected, etc.). This is a robust approach when the labels are self-explanatory and legible but also fraught with risk. There is a limit to the amount of information one can fit in the image using this method and it can be time consuming to generate individual labels for each image, particularly if there are large volumes of very similar, but unique, images. A more modern concern is that the labels must be machine-readable, and current optical character recognition (OCR) tools are not necessarily sufficient for reading these labels (Dietrich et al., 2012), although OCR is advancing and some systems have been developed for digitising such embedded metadata in highly specific instances (Kirchhoff et al., 2018). An alternative to including the text itself is to embed a unique ID, either in text form or embedded in a barcode, that is linked to a separate set of metadata for reference. This method has all the pitfalls of a database mentioned above, as the ID or code is meaningless in itself, and there is potential for total metadata loss if the associated metadata files are lost.

The ideal solution for embedding metadata without the risk of loss is to

embed everything in the pixels of the image itself. Quick response (QR) codes, two-dimensional barcodes with in-built error correction and redundancy (ISO, 2015), offer the ability to do just that, as they can contain a lot of data in a relatively small size. These QR codes are commonly used in marketing and traceability applications, where they often contain a URL link to a website with further information. However, their potential is far greater than this (Pandya & Galiyawala, 2014) and they have been experimented with for storing metadata, primarily for steganographic purposes, such as hiding watermarks and other information in an image without being easily detectable (Gaikwad & Singh, 2015; Gilsang & Hyeoncheol, 2016). There are patents for methods built into imaging hardware that convert metadata into a QR code to be stored as a separate image (Cok, 2012; Tsujimoto, 2016), but these methods do not seem to have been implemented in any existing technology.

QR codes have been used in biological collections for embedding extensive metadata in an image itself (Diazgranados & Funk, 2013). While this is another example of a heritage application with relatively small numbers of images that are only collected once per subject, it demonstrates the power of QR codes in a research context.

Ultimately, the existing tools and techniques for metadata management do not provide a robust solution to associating metadata with images of a scientific nature.

We therefore propose a method for using QR codes in high volume scientific imaging in order to embed the metadata securely in the image, making it transparent and machine readable for computer-based analysis and machine learning applications. The Methods section describes a continuous pipeline for generating large volumes of QR codes, displaying them in images to be permanently embedded from the moment of capture, reading the QR codes in the images and either renaming the files with relevant information or extracting the metadata into a separate but reproducible file. The robustness of this method was investigated using a range of image types and code parameters with millions of randomly simulated combinations of interference, distortion and transforma-

tion; and results are shown in the Results and Implications sections. Other potential uses for QR codes in research imaging are also discussed in the Implications section, such as dynamic encoding of time and location information dynamically and using standard metadata structures directly within the codes for enhanced machine readability.

Methods

This paper presents a method which entails four major components to enhance scientific data management when capturing digital imagery of samples, as shown in Figure 3.

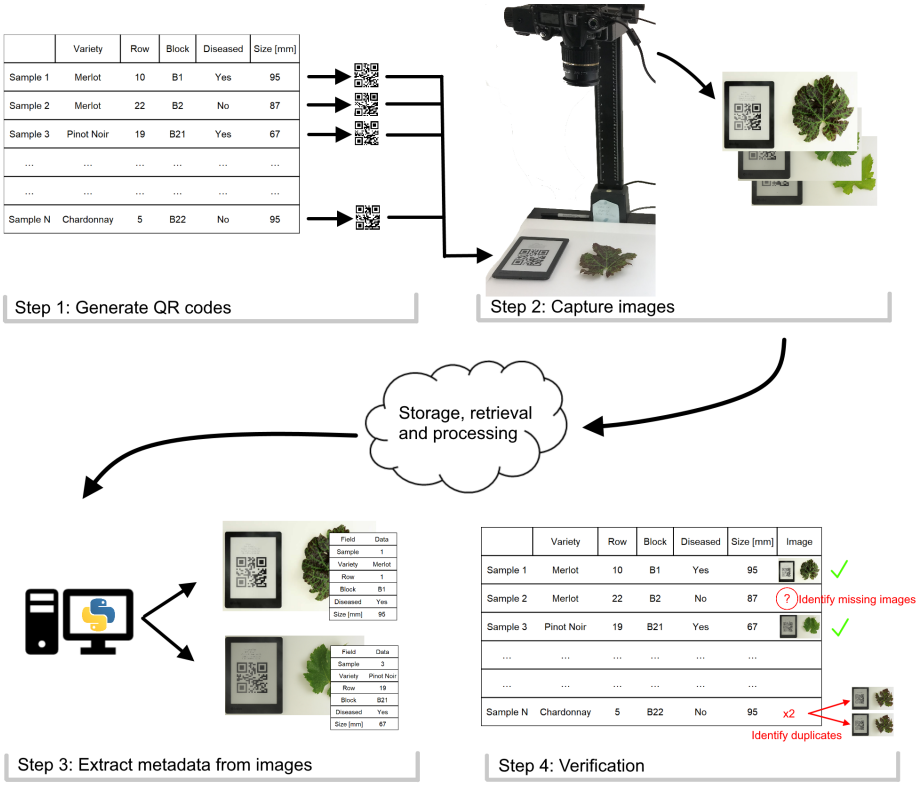


Figure 3: The four steps in the process for embedding and retrieving metadata from images of samples.

1. Generate QR codes: Firstly, the user generates a unique identifier for each expected image. If a unique identifier for each image is already known, then this can be used as the QR code content. This may simply be computed by formulating a spreadsheet with columns representing the metadata associated with that image, saved in Comma Separated Value (CSV) format with a single header row. Alternatively, the CSV file is read and a unique QR code generated for each item in the file, comprised of underscore or hyphen-concatenated key-value pairs (using the header as a key). The code may also be formulated as a JSON format string. These QR codes are generated in the form of either a series of files or pages in a single bookmarked document, along with human-readable text showing their content.
2. Capture images: An image is taken of each sample, with the QR code visible in the image. This may be done in any order and at any time, as long as the corresponding QR code is displayed at the time of capture and is clearly visible.
3. Extract metadata from images: All the images are processed by a decoding and renaming algorithm, which extracts the QR code content and renames the file accordingly, thus linking the file with its content. Regardless of whether metadata has been stripped out (for example by sharing through cloud-based services) or is unreadable on a different operating system, or the image extension has been changed, or a copy of the file has been made or the image has been resized; the metadata can be extracted by rerunning the image through the decoding and renaming algorithm. Additional functionality could be implemented according to the code content, but this is left to the user to action as per their requirements.
4. Verification: Another script is used to compare the original spreadsheet containing expected QR code IDs with those that have actually been captured. This identifies missing samples and duplicate samples which are highlighted to the user for further investigation (using the human-readable text in the image), thus improving the robustness of data capture.

Evaluation with simulated images

Several aspects of the method have been evaluated to identify how robustly it can handle many different practical situations. Background images were chosen to represent a number of use-cases (Figure 4). The first was that of a series of boxes in a warehouse, with a repetitive structure and application in a commercial logistics use-case. The second was that of a fish, with a white background, representing a general specimen laid out in controlled laboratory lighting conditions. The third was a TV calibration image, covering a broad gamut of colour, intensity and many fine details which may make detection of the QR code itself more challenging. The fourth was an image of a vine, to represent field-based imaging in variable and strongly contrasting lighting conditions. Finally, a plain white background image was used to find the best parameters under which the QR code detection and parsing could occur.

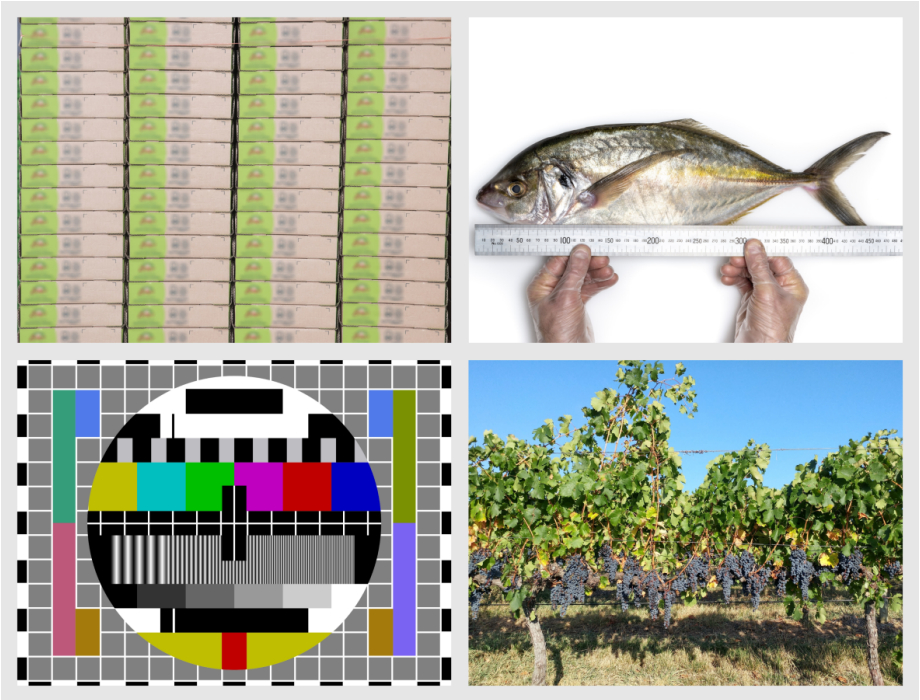


Figure 4: Background images used for overlaying quick response (QR) codes: 'boxes' (top-left), 'fish' (top-right), 'TV' (bottom-left), and 'vine' (bottom-right). A fifth image containing entirely white pixels ('white') was also included. The dimensions of all images were 2000 x 1500 pixels. The boxes images has been blurred here to mask company logos, although the image was not blurred during analysis.

To model real-world use-cases, the code was distorted by several parameters, and the performance of a QR code detection and parsing library (pyqrcode (v1.2.1)) was evaluated. The QR code was generated with a random number of characters, redundancy and size. It was then manipulated according to a random orientation and perspective. The bright value pixels (binary '1') were set to a given 8-bit value and the dark value pixels (binary '0') were set to a given 8-bit value. Onto each of the background images this manipulated QR code was digitally superimposed, with several examples of the manipulated codes shown in Figure 5 and an example of the superimposed image in Figure 6. Finally,

the resulting image was written to a file using OpenCV at a specified JPEG compression ratio. These parameters are listed in Table 1, which also shows the range of values for each parameter, and further described here:

- **Characters** – The length of the string in characters which was encoded in the QR code. Encoding followed the binary method as per the QR code specifications (ISO, 2015), as this allowed more workable solutions than the ASCII encoding which is limited to upper case letters and numerals. An increase in the code content could be achieved by using the ASCII encoding, but results are not shown in this paper.

- **Redundancy** – The level of redundant data contained in the code, as in the QR code specification (ISO, 2015).

- **Size** – The length of each edge of the QR code in the image including the specified 4-pixel white border on each side, noting the background image used was 2000 x 1500 pixels. This meant the width of the QR code was at most 10% of the width of the image, with codes larger than this being more reliable to decode.

- **Rotation** – The QR code was rotated by a certain number of degrees, to replicate misalignment in the image.

- **Perspective** – The QR code was warped in two dimensions to represent a perspective transform. The width of the code was scaled according to the provided ratio, and the height of the right-hand side of the code also scaled by the same amount, giving a perspective effect.

- **Noise** – Salt and pepper noise (0 and 1 values) was added to the code and the 4-pixel wide border.

- **Bright value and dark value** – The existing 0 and 1 values in the code and border (including noise pixels) were correspondingly replaced with a one bright and one dark value in the range of 0 to 255, simulating brightness and contrast changes.

- **Compression** – The final image was compressed to replicate data loss by re-imaging or other means and understand the impact of this on code readability.

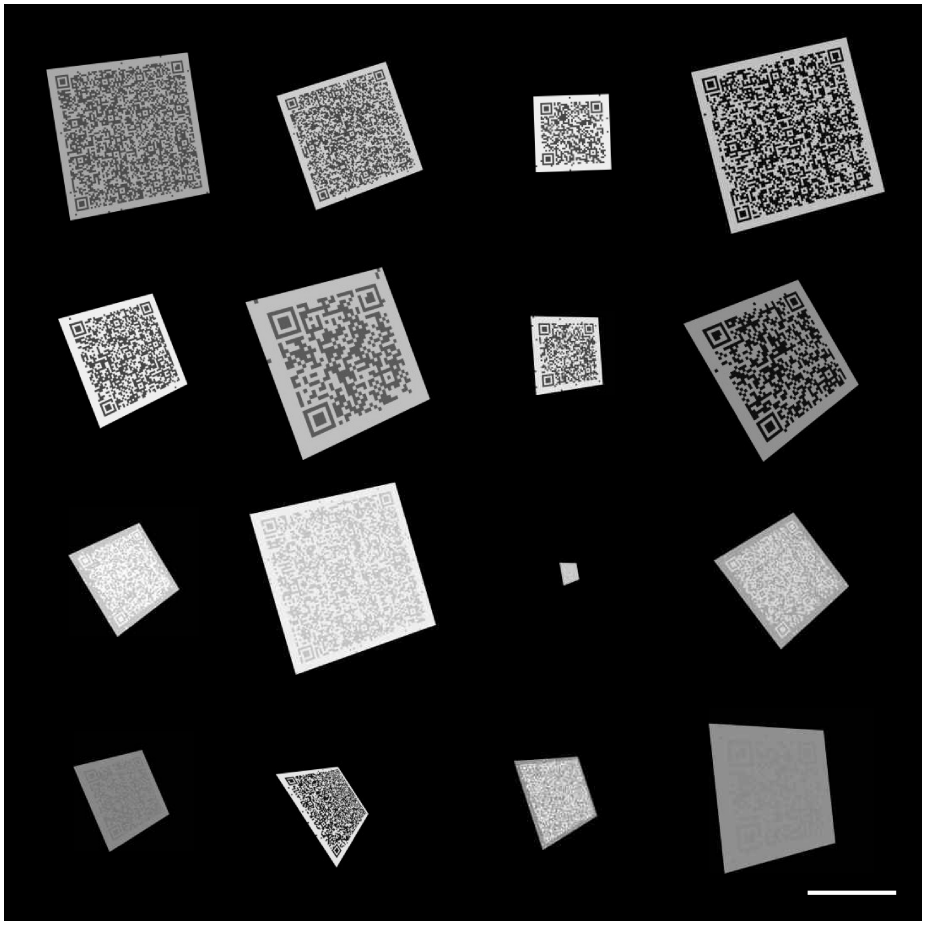


Figure 5: Random sample of quick response (QR) codes generated with a range of transformation parameters. The top eight QR codes were read successfully and the bottom eight were not. White bar indicates 100 pixels at original scale.

Figure 6: Example background image with overlaid random QR code for automated testing.

Parameter	Description	Range / list
Characters	Number of random characters in QR code	32 – 512
Redundancy	Parameter given to QR code generation function	Low, Medium, High
Size	Length of each edge of QR code in pixels	25 – 200
Rotation	Amount QR code was rotated in degrees	0 – 45
Perspective	Ratio of top edge to right edge after distortion	0.5 – 1.0
Noise	Percentage of code covered by salt & pepper noise	0.0 – 0.1
Bright value	Value of white pixels in QR code	128 – 255
Dark value	Value of dark pixels in QR code	0 – 255
Compression	Percentage given to JPG compression function	2 – 100

Table 1: Ranges for random parameter setting prior to QR code generation for each image.

Firstly, a broad analysis of these eight different parameters influencing the accuracy of QR code detection was undertaken. Each of the five images was overlaid with each of 600,000 QR codes generated using a randomly selected set of parameters (Table 1) giving a total of 3 million QR code-containing images to test. Each image was processed by the decoding algorithm and the result compared with the known code value; those which matched exactly were considered successful. One example is shown in Figure 6.

Code was written in Python (v3.6.5) using the following non-standard packages: pyqrcode (v1.2.1), pyzbar (v0.1.8), pillow (v7.0.0), opencv-python (v4.1.2), and scikit-image (v0.16.2). The processing was undertaken with batches of 10,000 images in parallel on a Beowulf computing cluster and the results from each batch concatenated into a single dataset. It was assumed that with the high number of possible parameter combinations no QR codes would be duplicated using this parallelisation method.

Evaluation with field use cases

The use of QR codes at time of image capture was tested in laboratory conditions and in a vineyard as part of separate studies. Laboratory testing used a Python script to generate a PDF containing a unique QR code for each sample, uploading this to an e-reader and placing the e-reader in the frame when capturing an image (Figure 7). The purpose of the QR code in these images was to record the origin and virus status of the grapevine leaf being photographed. Field testing used plastic cards each containing a unique printed QR code, that were fixed within grapevines (Figure 12). The purpose of the QR code in these images was to identify a precise location that could be related to the time stamp in the image in order to test the spatial accuracy of a precise Global Navigation Satellite System (GNSS).

Figure 7: Quick response (QR) code being used in scientific imaging under controlled conditions. An e-reader containing a pre-generated PDF document displays a unique QR code along with human-readable text for each grape leaf being imaged. The QR code contains the character string 'ML-R1B01-CB-L06-G6', which indicates the variety, location and virus status of the grapevine from which the leaf was taken, as well as the position on the vine and number of the leaf itself.

Results

Results from simulated images

The background images used in this study varied in the distribution of pixel brightness values (Figure 8). The Vine images had the most evenly distributed value frequency. The TV image had a range of brightness values, but these were distributed at distinct values, with most pixels having a brightness value of 0, 128, or 255; this is by design as the image is used for testing colour visualisation. Similarly, the Boxes image had a range of brightness values, but the distribution was more skewed towards the high end of the range. Both the Fish and White images had predominantly pixels with a brightness value of 255.

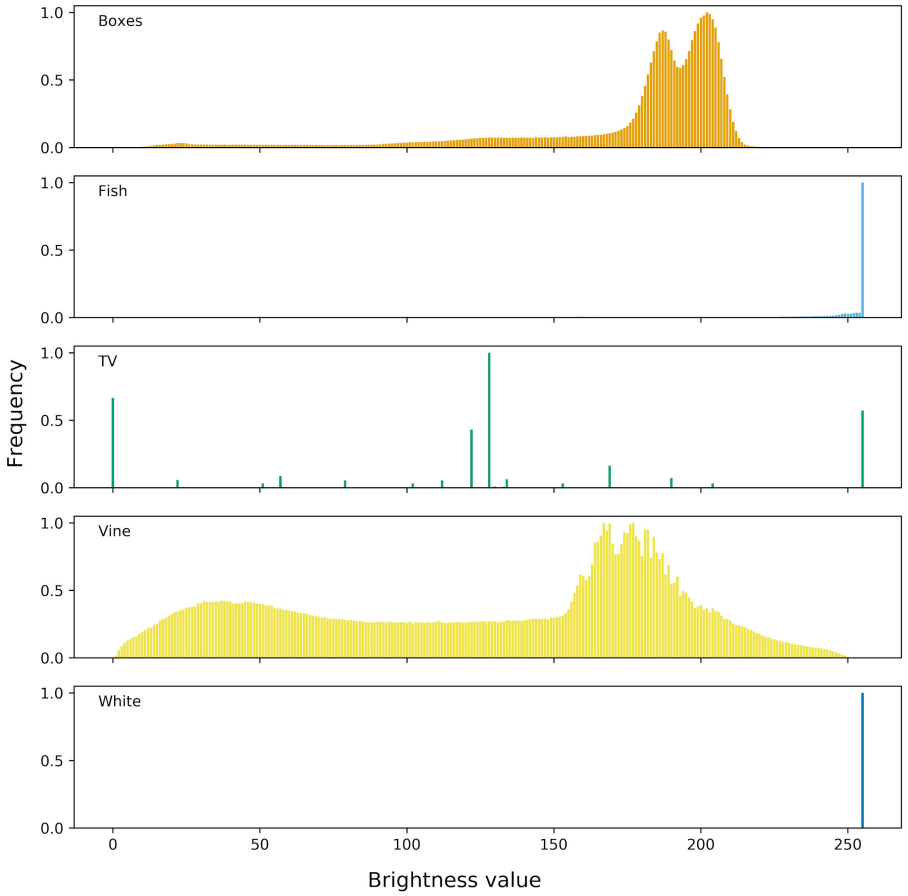


Figure 8: Pixel value frequency distribution for all background images; 0 = black, 255 = white.

Median values for each of the varied parameters where the QR code was read successfully were consistent across many transformations (Figure 9). Number of characters, rotation, noise, and dark value were all relatively low within the specified range, while size, perspective, and bright value were all relatively high. Median bright and dark values varied most between images, with the Boxes, Fish, and White backgrounds requiring higher values of both for QR code reads to be successful.

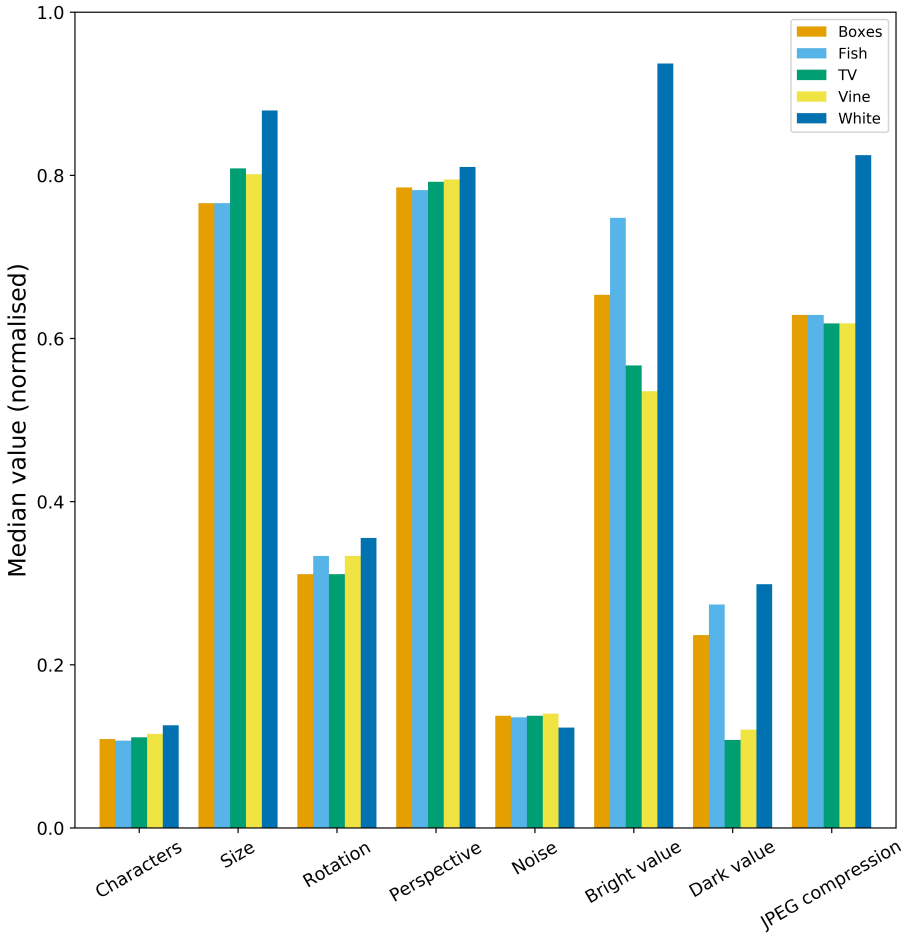


Figure 9: Median values for successful readings of quick response (QR) codes in each background image.

A total of 13,644 of the 3 million images contained QR codes that were successfully read, equating to 0.45% of the images tested (Figure 5). Successful values for each parameter and the interactions between parameters are shown in Figure 10. The values for the parameters not shown in each plot were not standardised; therefore, the values for parameters shown that resulted in a successful read were assumed to be robust against other transformations and dis-

tortions. A lower number of characters in the code content and larger code (i.e. larger element size) tended to overcome most transformations and distortions. Readability was particularly sensitive to perspective changes and the amount of noise. Changes in the bright value of QR code elements did not have a substantial effect on readability, except for the White image, where only values close to 255 resulted in a successful read. The inverse was true for changes to the dark value, with the Vine image also being sensitive to these changes. Rotation did not substantially affect readability, except for high character counts, small codes and extreme transformation and distortion values. The images did not appear to be affected by JPEG compression, with the exception of the White image, for which any JPEG compression reduced the probability of the QR code being read successfully. The redundancy setting affected the readability of the QR code only when the number of characters, size of code, perspective and noise were altered (Figure 11). Higher redundancy settings meant that using fewer characters and increased code sizes were required to achieve success. Higher redundancy settings also made the codes more vulnerable to perspective changes because of the increased number of elements and because the implementation of perspective reduced the size of each element. However, increasing the redundancy made the codes more resistant to noise.

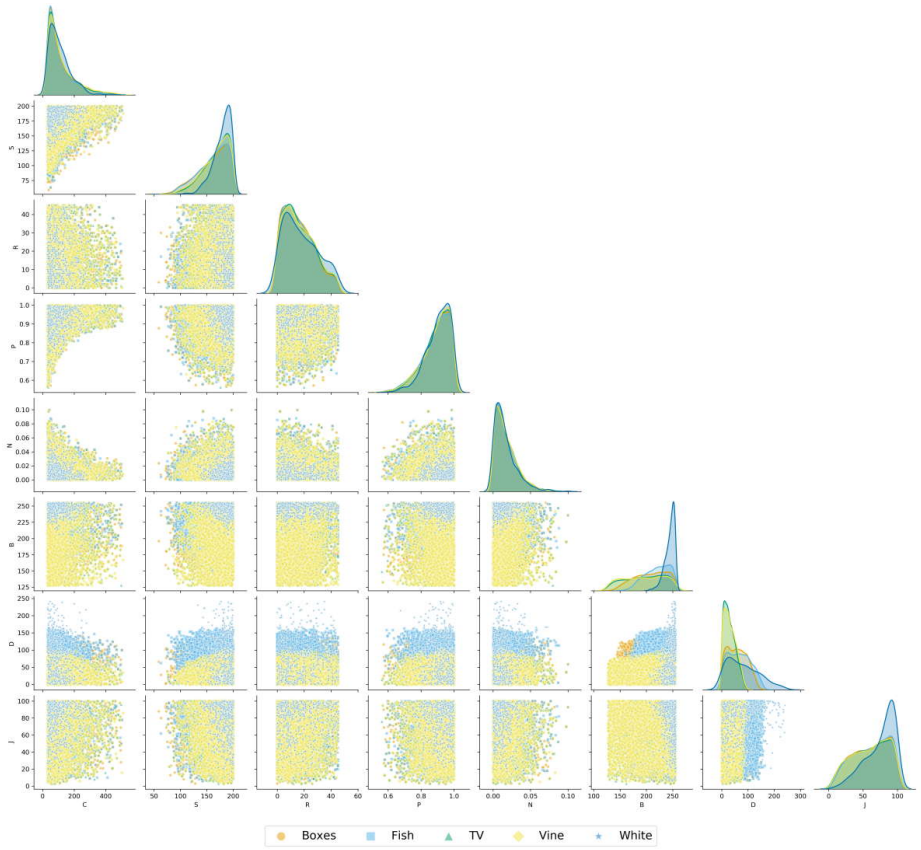


Figure 10: Relationships between all pairs of parameters. Plots along the diagonal show the frequency distributions for successful values of each parameter along the x-axis (y-axis for all these frequency plots is 0 – 1). C = characters, S = size, R = rotation, P = perspective, N = noise, B = bright value, D = dark value, J = JPEG compression ratio.

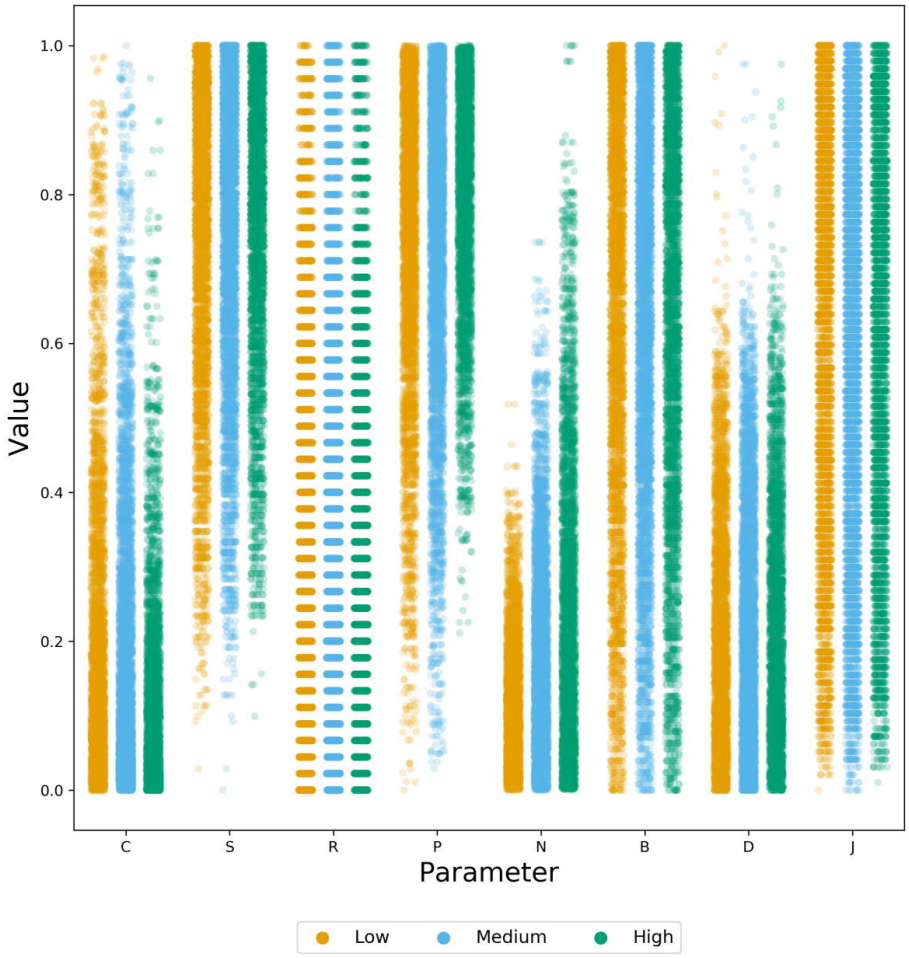


Figure 11: Normalised values for successful readings of quick response (QR) codes with each redundancy level across all images. C = characters, S = size, R = rotation, P = perspective, N = noise, B = bright value, D = dark value, J = JPEG compression ratio.

Results from field use cases

The use of this method with an e-reader placed in images taken under laboratory conditions (Figure 7) was 100% successful, with all 1600 images containing QR codes being successfully read. The use of the method with printed plastic cards hung on grapevines (Figure 12) resulted in successful QR code reads 70.1%

of the time (Table 2), with a range of possible reasons for code read failure.

QR code parsing results	QR code parsing results	Percentage
Correctly parsed	370	70.1%
Insufficient contrast	6	1.1%
Variable lighting across code	42	8.0%
Occlusion of code	31	5.9%
Code bent	28	5.3%
Code angle too acute	28	5.3%
No code in image	0	0.0%
Out of focus	15	2.8%
Unknown	8	1.5%

Table 2: Quick response (QR) code success and failure rates for 528 physical codes used in a vineyard study and the likely reasons for failure.

Figure 12: Examples of misread QR codes in images captured in a trial in grapevines testing the accuracy of a precise global navigation satellite system (GNSS). Possible explanations for read failure are occlusion (top-left), distortion or reflection (top-right), contrast (bottom-left), and perspective warp (bottom-right).

Implications

Discussion of simulated image results

The method of placing QR codes in images to embed metadata has been shown to be not only possible, but also robust against a range of distortions and transformations. The low overall success rate (0.46%) is not an indication

of the reliability of the method in general, but rather a reflection of the extremely large feature space in terms of possible parameter combinations, which resulted in the vast majority of QR codes being unreadable. In reality, these parameters would not be effective to the extent they were in this study, especially with changes to all the other parameters happening simultaneously.

Somewhat surprisingly, JPEG compression did not affect readability for most images until it was set very low (e.g. below 10%). The exception to this was the White image, for which any compression reduced the probability of a successful read. This is possibly due to the entropy within the White image since every pixel was the same value. This meant that any compression applied to the image must have been applied to the pixels belonging to the QR code, whereas the other images had greater variation in pixel values, and thus greater entropy, and so compression could be applied to any part of the image. Therefore, the ability to compress an image containing a QR code and retain readability of that code is dependent on the complexity of the image.

The TV and Vine images had the highest variances in their brightness values (Figure 8), with more pixels at the darker end of the spectrum than other images, hence they were more sensitive to a reduction in the QR code pixel brightness values, which is analogous to shading in the real world. Conversely, the White image had pixel values entirely at the brighter end of the spectrum and was therefore sensitive to increases in the QR code pixel brightness value, which is analogous to glare or over-saturation of an image. The Fish image suffered from this problem, although to a lesser extent. This sensitivity to changes in pixel brightness was likely due to the use of automatic thresholding, such as Otsu's method (Otsu, 1979), in the QR code extraction algorithm, which relies on having easily distinguishable bright and dark values in the QR code. As the QR code only makes up a small portion of the image, the threshold is primarily set on the image itself, so that all the pixels in the QR code can be left on one side of the threshold if the contrast in the code's pixels is not sufficient. As demonstrated here, a darker image will be more susceptible to failure when the QR code pixels are made darker (e.g. shading) and a lighter image will

be more susceptible to failure when the QR code pixels are made brighter (e.g. glare). This means care should be taken when collecting such images that there is sufficient contrast in the QR code pixel values, avoiding shading or glare as much as possible.

Increased redundancy requires a larger number of pixels to store the same number of characters, so results in either an increased code size or reduced pixel size. This predominantly influenced the performance in respect of perspective changes, which resulted in smaller pixels on one side of the code and thereby impacted the resolving power of the decoding algorithm. Thus, holding all other parameters equal, the only parameter which was improved by increasing redundancy was the ability to handle noise in the image. This could be a trade-off that is made in the real world if for some reason the images being taken are particularly prone to noise or physical analogues (e.g. dirt), although it appears that a lower redundancy would be suitable for most applications as it would allow for more information to be stored in the codes.

Despite the complex interactions between these parameters, it is perhaps useful to indicate which values of parameters might be used as a starting point when designing an experiment. From the median normalised values in Figure 9, combined with the ranges of values prior to normalisation in Table 1, and noting the original resolution of the background images was 2000 x 1500 pixels, we make the recommendations in Table 3. A higher value for the compression function here means less compression applied.

Parameter	Description	Recommended value
Characters	Number of random characters in QR code	<80 characters
Redundancy	Parameter given to QR code generation function	Medium
Size	Length of each edge of QR code in pixels	>165 pixels
Rotation	Amount QR code was rotated in degrees	(Arbitrary)
Perspective	Ratio of top edge to right edge after distortion	>0.9
Noise	Percentage of code covered by salt & pepper noise	<1%
Bright value	Value of white pixels in QR code (/255)	>205
Dark value	Value of dark pixels in QR code (/255)	<50
Compression	Percentage given to JPEG compression function	>60%

Table 3: Recommended parameter settings for how the QR code should appear in a captured image of 2000 x 1500 pixels to be successfully read.

Discussion of field use case results

Testing of the method in two real-world scenarios demonstrated that the method is much more reliable than the overall result suggests. Under laboratory conditions, with controlled photographic conditions, there was a 100% success rate. This was using a relatively short code, although it is likely that a much more information-dense code could be used in this scenario, so long as the camera was of adequate quality. The method was less successful in a field setting using QR codes printed on tags and hung in grapevines. The success rate for reading the codes in the images was 60.1%, and was likely due to lighting factors as well as physical factors, such as distortion and occlusion (Figure 12, Table 2). These issues could be resolved through the use of larger QR codes and more care with how the codes are secured and how they are imaged. Other solutions such as artificial lighting to remove shadows could also improve the success rate.

The information format contained in the QR codes in this study was random strings of alphanumeric characters simulating the sort of unique IDs that might be used in other studies. Likewise, the QR codes used in the field studies con-

tained short strings that were either unique IDs or shortened codes referencing metadata stored elsewhere. We propose that a more useful method would be to embed all the metadata using a string stored in the JavaScript Object Notation (JSON) format (Figure 13). Strings formatted in this way, with human-readable keys and values, would remove the reliance on a separate data dictionary to make sense of typical unique IDs that in themselves contain no meaningful information. As an additional advantage to using JSON-formatted strings, when the QR code is read with Python the string is easily converted into the dictionary data structure, from which extracting specific pieces of information is particularly easy. This method would result in smaller element sizes, and therefore require larger QR codes to ensure reliable content reads, but we consider this a worthwhile trade-off for embedding all metadata, not just a unique ID, in an image to create a more robust metadata storage mechanism.

```
{'experiment': 'leaf_imaging',
 'id': 'ML-R1B01-CB-L06-G6',
 'date': '2019-02-21',
 'location': {'latitude': '-36.89', 'longitude': '174.72'},
 'vine': {'variety': 'Merlot', 'row': '01', 'bay': '01'},
 'leaf': {'canopy': 'bottom', 'number': '06'},
 'virus': 'GroupVI'}
```

Figure 13: An example of a string stored in JavaScript Object Notation (JSON) data format embedded in a quick response (QR) code and the resulting dictionary object when read and interpreted by Python.

Conclusion

In this study we have presented and tested a useful method for embedding metadata in images using physical QR codes at the time of capture. These codes are subject to failure under certain conditions, but with careful QR code placement and image capture, the embedded metadata will be retained, as long

as the image is not cropped. We have included recommended parameter values in Table 3 as a guide to practitioners applying this method.

Metadata is crucial to contextualising images, and without it the image itself may have little value, particularly in a research context. Previous publication have discussed the problems with storing metadata separate from images (Smith et al., 2014), and the importance of embedding metadata in the image itself (Reser & Bauman, 2012). Data container standards have been established to ensure embedded metadata is in a consistent format and as interoperable as possible (JEITA, 2002; ISO, 2019), although these are vulnerable to data loss through image duplication, or system or format changes. Including metadata information in the image itself is a common approach, although text labels can be difficult for computers to read (Dietrich et al., 2012), and barcoded IDs rely on a separate reference file to retrieve the metadata from an otherwise meaningless ID.

We believe that containing visible QR codes within images is the next logical step forward in embedded metadata, and offers two novel components over common approaches to metadata management: robustness and openness.

The use of QR codes provide the ability to retain image metadata within the image itself in a machine-readable format, robust to metadata stripping by many cloud computing services and image processing pipelines, and tolerant of image manipulation and file format changes. Like many image processing pipelines, the method we describe used OpenCV to duplicate, transform and save images (Bradski, 2000). If data containers such as EXIF (JEITA, 2002) were relied on to store metadata in these case, that metadata would have be lost after the first image duplication. If cloud services were being used as part of the pipeline, close attention would need to be paid to how the images and the metadata within them are treated to ensure the metadata is not being lost. By embedding the information within the pixels themselves as a visible QR code, the metadata cannot be lost without substantial alterations to the pixel values or by being cropped out of the image. As the analysis above shows, these alterations need to be relatively severe to affect readability and subsequent loss

of metadata.

While the QR codes themselves are governed by a set of standards (ISO, 2015), the information contained within them is almost completely without restriction. This is in contrast to standard data containers, which have a set list of fields that may not be appropriate for the use case (Reser & Bauman, 2012). Storing as a JSON-formatted string (Figure 13) would provide the triple benefits of structure, flexibility, and clarity. Naturally, this could contain a URI which specifies further instructions for parsing the data, as suggested in the Linked Data method (Berners-Lee, 2009), however for simple applications this is unnecessary, and adds a dependency on an external resource which is not guaranteed to exist in the future.

Embedding visible QR codes may present a problem in a machine learning context as there is a very real risk of training an algorithm that does nothing more than read QR codes. However, this problem is easily solved by extracting the metadata from the QR code in the image then using the detection component of the reading software to identify the boundaries of the QR code and replace it with a black rectangle before feeding the image into the training step. The requirement for a 4-cell wide blank border around the QR code as stipulated in the standards makes this relatively straightforward.

An additional benefit of the specific pipeline described here is that users are provided with a log of images which have been captured to verify that everything necessary has been captured correctly. The key limitation is that the user taking the images needs to ensure the correct QR code is visible at the time of capture. This is challenging in large datasets, but the use of human-readable text along with the code reduces this risk, as does the use of bookmarks to find the appropriate code and the ability to double-check the existence of the image once the images have been decoded.

Furthermore, ensuring the metadata is correct at time of capture is essential. There is always scope for a hybrid approach, where essential metadata is embedded in the image, as well as a unique code for linking to a database. This would enable editing of the metadata where errors have occurred as well

as the addition of data such as experimental results pertaining to this sample. The tradeoff between maintaining the database and the simplicity of having the metadata embedded in the image is a decision which is best made by the user.

We believe the power of QR codes for the purpose of storing metadata in an open and robust way has not yet been fully utilised. By combining the methods described here with more advanced data formatting and display technology, the issue of missing metadata and contextually meaningless images could become a thing of the past.

Code Availability

A program for generating the QR codes from a spreadsheet of sample data is available here: `https://github.com/markwhittyunsw/QR_image_renamer`. *A program to extract the metadata, rename the images and verify which samples were images is available here:* `https://github.com/markwhittyunsw/GenerateQRCode`

1. Acknowledgements

We wish to thank Karmun Chooi, Kai Lewis and Mark Wohlers (Plant & Food Research New Zealand) for their help in testing the QR code method. We would also like to thank Plant & Food Research New Zealand, the Strategic Science Investment Fund (SSIF), and the Collaborative Research Council – Spatial Information (CRC-SI) for funding the projects in which the method was tested.

References

Berners-Lee, T. (2009). Linked data - design issues. URL: `https://www.w3.org/DesignIssues/LinkedData.html`.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, .

Christensen, S. O., Dunlop, D., Pilsk, S., Snyder, R., Nguyen, D., Stauderman, S., Smith, S., Ferrante, R., Rahaim, K., & Roby, M. (2010). *Basic Guidelines for Minimal Descriptive Embedded Metadata in Digital Images*. Report Embedded Metadata Working Group. URL: https://repository.si.edu/handle/10088/9719.

Cok, R. S. (2012). Digital image file including optical code. URL: https://patents.google.com/patent/US20120273579.

Diazgranados, M., & Funk, V. A. (2013). Utility of qr codes in biological collections. *PhytoKeys*, (pp. 21–34). URL: https://www.ncbi.nlm.nih.gov/pubmed/24198709https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3819127/. doi:10.3897/phytokeys.25.5175.24198709[pmid] PMC3819127[pmcid].

Dietrich, C., Hart, J., Raila, D., Ravaioli, U., Sobh, N., Sobh, O., & Taylor, C. (2012). Invertnet: a new paradigm for digital access to invertebrate collections. *Zookeys*, (pp. 165–181). URL: <GotoISI>://WOS:000308267500013. doi:10.3897/zookeys.209.3571. Times Cited: 17 Si Dietrich, Christopher/0000-0003-4005-4305 1 17 1313-2970.

Frisch, S. (2013). Embedded metadata in cultural image collections and beyond: Embedding metadata in image files at calpoly, san luis obispo. *VRA Bulletin*, *39*, 3. URL: https://online.vraweb.org/cgi/viewcontent.cgi?article=1031&context=vrab.

Gaikwad, A., & Singh, K. (2015). Information hiding using image embedding in qr codes for color images: A review. *International Journal of Computer Science and Information Technologies*, *6*, 278–283. URL: http://ijcsit.com/docs/Volume%206/vol6issue01/ijcsit2015060163.pdf.

Gilsang, Y., & Hyeoncheol, K. (2016). Embedding qr code in the wavelet domain of image for metadata hiding. *Indian Journal of Science and Technology*, *9*. URL: https://koreauniv.pure.elsevier.com/en/publications/embedding-qr-code-in-the-wavelet-domain-of-image-for-metadata-hid.

Greenberg, J., Murillo, A., Ogletree, A., Boyles, R., Martin, N., & Romeo, C. (2014). Metadata capital: Automating metadata workflows in the niehs viral vector core laboratory. In S. Closs, R. Studer, E. Garoufallou, & M. A. Sicilia (Eds.), *Metadata and Semantics Research, Mtsr 2014* (pp. 1–13). volume 478 of *Communications in Computer and Information Science*. URL: <GotoISI>://WOS:000363269900001.

International Press Telecommunications Council (2019a). Embedded metadata manifesto. URL: http://www.embeddedmetadata.org/embedded-metatdata-manifesto.php.

International Press Telecommunications Council (2019b). IPTC photo metadata standard. URL: https://iptc.org/standards/photo-metadata/iptc-standard/.

ISO (2015). Information technology — automatic identification and data capture techniques — qr code bar code symbology specification, iso/iec 18004:2015 [iso/iec 18004:2015], international organization for standardization, geneva, switzerland., . URL: https://www.iso.org/standard/62021.html.

ISO (2019). Graphic technology — extensible metadata platform (xmp) — part 1: Data model, serialization and core properties, iso 16684-1:2019, international organization for standardization, geneva, switzerland.

JEITA (2002). Exchangeable image file format for digital still cameras: Exif version 2.2. jeita cp-3451. URL: https://www.exif.org/Exif2-2.PDF.

Kirchhoff, A., Buegel, U., Santamaria, E., Reimeier, F., Roepert, D., Tebbje, A., Guentsch, A., Chaves, F., Steinke, K.-H., & Berendsohn, W. (2018). Toward a service-based workflow for automated information extraction from herbarium specimens. *Database-the Journal of Biological Databases and Curation*, . URL: <GotoISI>://WOS:000447034100002. doi:10.1093/database/bay103. Times Cited: 0 0.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, *9*, 62–66. doi:10.1109/TSMC.1979.4310076.

Pandya, K. H., & Galiyawala, H. J. (2014). A survey on qr codes: in context of research and application. *International Journal of Emerging Technology and Advanced Engineering*, *4*.

Reser, G., & Bauman, J. (2012). The past, present, and future of embedded metadata for the long-term maintenance of and access to digital image files. *International Journal of Digital Library Systems (IJDLS)*, *3*, 53–64. URL: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jdls.2012010104. doi:10.4018/jdls.2012010104.

Saleh, E. I. (2018). Image embedded metadata in cultural heritage digital collections on the web: An analytical study. *Library Hi Tech*, *36*, 339–357. URL: <GotoISI>://WOS:000432185900010. doi:10.1108/lht-03-2017-0053.

Smith, K. R., Saunders, S., & Kejser, U. B. (2014). Making the case for embedded metadata in digital images. In *Archiving Conference* (pp. 52–57). Society for Imaging Science and Technology volume 2014.

Tsujimoto, T. (2016). Apparatus and method for automatically generating an optically machine readable code for a captured image. URL: https://patents.google.com/patent/US20160269675.