Mathematical Institute
University of Oxford

Investigating Image Colour Recovery Using Reproducing Kernel Hilbert Spaces

Contents

1 Introduction

We consider the function extension approach to the full colour recovery of grey-scale images using prescribed colour information at a small number of pixels. By first introducing key image and colour properties, we build up to a rigorous formulation of the problem at hand, and show how it can be solved using the abstract theory of reproducing kernel Hilbert spaces (RKHS). After forming these necessary foundations, we consider the implementation of the problem into Matlab, and show how it can be solved computationally. We then consider the effects of the different parameters involved, and show how some can be optimised to reduce a particular form of error which we shall define. Finally we offer an alternate approach to solving the problem, by treating one of the resulting equations as a constrained minimisation problem. For this we compare results with the standard approach and offer a possibly improved method.

2 Image Storage

A prerequisite to image recolourisation is the basic understanding of image storage and colour representation. As such we define some trivial yet fundamental notions that will be helpful throughout this report. We can start by considering a grey-scale image. On a computer this is stored as a $L \times W$ matrix with 8-bit entries, where L is the number of pixels that constitute the length of the image, and W the width. This simply means that each element in the matrix takes integer values between 0 and $2^8 - 1 = 255$ inclusive. We call such matrices 'channels' [1]. Here, each element then represents that pixels 'intensity', in that a higher number indicates a darker pixel. Just as a grey scale image is stored as a single 8-bit channel, we can represent coloured images as three 8-bit channels (also known as 24-bit colour[1]), one for each of the primary colours red, green and blue. Each channel represents the intensities of pixels in that respective colour, and so a single pixel has a colour defined by a 3-tuple with 8-bit entries. The matrix representation of images allows for very convenient manipulation both mathematically and digitally. We note that other colour models as opposed to the RGB framework that we employ do exist, however these are often

[1] 24-bit colour is also referred to as 'true colour' since the number of possible colours that can be represented in this model (2^{24}) is greater than the number of colours perceivable by the human eye [1]. As such colour models with higher colour depth (more bits per channel) are rarely needed, so we limit our discussion to true colour.

derived from this model and as such we do not consider them in this report.

3 Conversion to Grey-scale

We first need to consider methods of converting a true colour image to grey-scale - this is not necessarily a trivial process and ultimately the final recoloured image will depend on this initial grey-scale conversion. The first, and perhaps the naive, method we consider is the average method. As the name suggests, to obtain the 8-bit grey-scale value of an RGB pixel, we simply take the average of the intensities of that pixel from each channel. If r_i, g_i and b_i represent the intensities in each channel of the pixel i, then the grey-scale value is simply given by $\gamma_i^{\text{avg}} = (r_i + g_i + b_i)/3$. Though not a completely unreasonable approach, this overshadows the perception of colours by the human eye. A more physically sound option would be to also factor in that the human eye is more sensitive to specific wavelengths. In particular it is the most sensitive to green, and the least to blue. The specific weightings of this are encompassed by the mapping $\gamma_i^{\text{real}} = 0.299r_i + 0.587g_i + 0.114b_i$. In general we can consider weighted mappings of the form $G_i = w_r r_i + w_g g_i + w_b b_i$ where $w_r + w_g + w_b = 1$. It is perhaps not easily seen which choice of grey-scale mapping will minimise the error in recolourisation (if any) *a priori*, as such we shall later consider the relative errors of these.

4 Mathematical Formulation and Method

Up to now we have considered ideas informally so as to motivate the problem. In this next section we formulate the mathematics behind the problem, and describe how it will be solved. We also consider some of the more rigorous elements of the governing functional analysis, namely we introduce the concept of reproducing kernal hilbert spaces, for which we shall largely use results from [7].

4.1 Formulating the Problem

We begin with an image of $m = L \times W$ pixels. We want to recover the full colour information of each pixel of an image given grey-scale values at most pixels and full colour information at a small amount of pixels. We consider the set

$$\Omega = \{z_1, \ldots, z_m\} \tag{1}$$

of all m pixels of an image , and a subset $D \subseteq \Omega$ of these points

$$D = \{x_1, \ldots, x_n\} \tag{2}$$

for $n \ll m$ (we typically take $n \approx 0.05m$), where elements represent the n pixels whose full colour information is known. Since we assume to have grey-scale information at all points (if we have full colour information at a pixel we also have grey-scale colour information there) z_i, $0 \leq i \leq m$, we can say there exists a map

$$\gamma : \Omega \mapsto I \tag{3}$$

from pixels to the set $I = \{i \in \mathbb{Z} : 0 \leq i \leq 255\}$ of values representing the grey-scale intensity at each pixel. Similarly we can define the map from pixels whose colour is known to their respective intensities in each channel. Namely we can define three maps $f^r, f^g, f^g : D \mapsto I$. More succinctly we have

$$f : D \mapsto I^3 \tag{4}$$

where $f = (f^r, f^g, f^b)$. In our problem we would like to find a map from each pixel to a sensible RGB 3-tuple. By sensible here we simply mean that the resulting image is recoloured somewhat accurately, in particular we add the additional constraint that our resultant map retains colour information for pixels whose information is known prior. Formally we seek the map $F = (F^r, F^g, F^b)$ where $F : \Omega \mapsto I^3$ under the constraint $F|_D \approx f|_D$, this is known as an extension problem [7]. A general approach to solving such extension problems is the variational approach. In this we compute F by minimising the functional

$$\inf_{F \in X_2(\Omega)} \{\mathcal{F}(F) = \delta \mathcal{F}_2(F) + \mathcal{F}_1(F - f)\}, \tag{5}$$

where $\mathcal{F}_1$ and $\mathcal{F}_2$ are functionals defined on the function spaces for which f and F are defined on, $X_1(D)$ and $X_2(\Omega)$ respectively [7]. Here $\delta > 0$ is some parameter whose value may dictate how 'well behaved' the extension is on the entire domain Ω. Essentially we find the function F defined on the required function space such that the expression (5) is minimised given some currently arbitrary functionals $\mathcal{F}_1$ and $\mathcal{F}_2$. We shall next see more specific forms of these and exploit that this approach is equivalent to a regularised least squares problem in a vector-valued reproducing kernel Hilbert space.

4.2 Reproducing Kernel Hilbert Spaces

We briefly introduce the abstract concept of reproducing kernel Hilbert spaces (RKHS) in the context of this problem. We then consider vector valued reproducing kernal Hilbert spaces which have applications in machine learning and optimisation in general. The results reviewed here follow the presentation of the subject in [7].
We first consider the notion of a positive definite kernel. Let X be a nonempty set, then the kernel $K : D \times D \mapsto \mathbb{R}$ is a positive definite kernel if $K(x_i, x_j) = K(x_j, x_i)$ (symmetric) and

$$\sum_{i,j}^{n} K(x_i, x_j) a_i a_j \geq 0$$

for points x_i in any finite subset of D and real coefficients a_i. If K is such a positive definite kernel then it permits an evaluation operator $K_x : X \mapsto \mathbb{R}$ defined by $K_x(t) = K(x, t)$. We can now define the (unique) Hilbert space $\mathcal{H}_K$ induced by this kernel, we have

$$\mathcal{H}_k = cl\left(\left\{\sum_{i=1}^{n} K_{x_i} a_i : n \in \mathbb{N}\right\}\right) \tag{6}$$

where $cl(V)$ denotes the closure of V. This is a Hilbert space of functions $f : D \mapsto \mathbb{R}$ and it satisfies the following properties for any $f \in \mathcal{H}_K$ and $x \in D$

1. $K_x \in \mathcal{H}_K$;
2. $\text{span}\{K_x\}_{x \in D}$ is dense in $\mathcal{H}_K$;
3. the reproducing property is satisfied. That is the inner product induced by $\mathcal{H}_K$ satisfies $f(x) = \langle f, K_x \rangle_{\mathcal{H}_K}$.

In this case we call $\mathcal{H}_K$ the RKHS with reproducing kernel K [7]. The inner product induced by this RKHS is given by

$$\langle f_a, f_b \rangle_{\mathcal{H}_K} = \left\langle \sum_{i=1}^{n} K_{x_i} a_i, \sum_{j=1}^{n} K_{y_j} b_j \right\rangle_{\mathcal{H}_K} = \sum_{i,j=1}^{n} K(x_i, y_j) a_i b_j$$

where $f_a, f_b \in \mathcal{H}_K$. The norm is defined with respect to this, i.e. $||f||_{\mathcal{H}_K} = \langle f, f \rangle_{\mathcal{H}_K}$. The canonical example of an RKHS is the space induced by the Gaussian kernel defined by

$$K(x, y) = \exp\left(-\frac{||x - y||^2}{\sigma^2}\right), \qquad x, y \in \mathbb{R}^n. \tag{7}$$

Though not fully relevant for our discussion, for completeness we state the RKHS induced by this kernel, we have

$$\mathcal{H}_K = \left\{ ||f||^2_{\mathcal{H}_K} = \frac{1}{(2\pi)^n(\sigma\sqrt{\pi})^n} \int_{\mathbb{R}^n} \exp\left(\frac{\sigma^2|\xi|^2}{4}\right) |\hat{f}(\xi)|^2 \mathrm{d}\xi \right\}$$

where $\hat{f}$ denotes the Fourier transform of f.

4.2.1 Vector-valued RKHS

We next consider reproducing kernel Hilbert spaces of vector valued functions. In particular we make use of functions that map onto Hilbert spaces; we let $\mathcal{W}$ denote a real Hilbert space with associated inner product $\langle \cdot, \cdot \rangle_{\mathcal{W}}$ and let $\mathcal{L}(\mathcal{W})$ denote the Banach space of bounded linear operators on this Hilbert space. Just as in the standard RKHS case where we considered a kernel that maps $D \times D$ to the vector space $\mathbb{R}^n$, in the vector valued case we have the operator-valued kernel $K : D \times D \mapsto \mathcal{L}(\mathcal{W})$. We say K is an operator-valued positive definite kernel if $K(x_i, x_j) \in \mathcal{L}(\mathcal{W})$ is self-adjoint and

$$\sum_{i,j=1}^{n} \langle w_i, K(x_i, x_j) w_j \rangle_{\mathcal{W}} \geq 0$$

for points x_i in any finite subset of D vectors $w_i \in \mathcal{W}$ [7]. We can denote by $\mathcal{W}^D$ the vector space formed by all function $f : D \mapsto \mathcal{W}$. Then for all $x \in D$ and $w \in \mathcal{W}$ we construct the function $K_x w \in \mathcal{W}^D$ defined by

$$K_x w(y) = K(y, x) w$$

for all $y \in D$. Clearly this is the vector-valued analogue of the evaluation operator. We can now define the (unique) Hilbert space $\mathcal{H}_K$ induced by this kernel in a similar fashion to the scalar case, we have

$$\mathcal{H}_K = \mathrm{cl}(\mathrm{span}\{K_x w : x \in D, w \in \mathcal{W}\}). \tag{8}$$

We note the clear generalisation from the scalar case (6) to the vector case (8). The induced inner product also generalises naturally from scalars to vectors, we have

$$\langle f_a, f_b \rangle = \left\langle \sum_{i=1}^{n} K_{x_i}(w_a)_i, \sum_{j=1}^{n} K_{y_j}(w_b)_j \right\rangle = \sum_{i,j=1}^{n} \langle (w_a)_i, K(x_i, y_j)(w_b)_j \rangle_{\mathcal{W}}. \tag{9}$$

Finally, we note the generalisation of the reproducing property, we have

$$\langle f(x), w \rangle_{\mathcal{W}} = \langle f, K_x w \rangle_{\mathcal{H}_K}$$

for every $f \in \mathcal{H}_K$.

4.3 Solving the Variational Problem

Recall the extension problem that we are considering. For a closed set Ω and a subset $D \subset \Omega$ on which we define $f^s : D \mapsto \mathcal{W}$, we would like to extend the function f^s to the entire domain Ω such that the extended function approximates the original function on D and behaves reasonably on the entire domain. We note here that we currently assume $\mathcal{W} = \mathbb{R}^3$. If we consider some operator-valued positive definite kernel K and induced vector-valued RKHS $\mathcal{H}_k$ given by (8), then we seek $F^s \in \mathcal{H}_K$. As such F can be expressed as

$$F^s(x) = \sum_{j=1}^{n} K(x, x_j) a_j^s \tag{10}$$

where $a^s \in \mathbb{R}^n$ is the vector of coefficients to be determined for each colour $s = r, g, b$. It can be shown that when prescribing that the extension function belongs to the vector-valued RKHS, the variational problem (5) is equivalent to the following regularised least square problem:

$$F^s = \arg \min_{F^s \in \mathcal{H}_K(\Omega)} \frac{1}{n} \sum_{i=1}^{n} ||F^s(x_i) - f^s(x_i)||_{\mathcal{W}}^2 + \delta ||F^s||_{\mathcal{H}_K(\Omega)}^2. \tag{11}$$

Here $|| \cdot ||_{\mathcal{W}}$ is nothing but the euclidean norm since $\mathcal{W} = \mathbb{R}^3$, and the RKHS norm is given by $\langle F^s, F^s \rangle_{\mathcal{H}_K}$ where the inner product is given by (9). Since K is positive definite, it follows that we can minimise over $a^s \in \mathbb{R}^n$. We have

$$F^s = \arg \min_{a^s \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} (F^s(x_i) - f^s(x_i))^2 + \delta \sum_{i,j=1}^{n} K(x_i, x_j) a_i^s a_j^s. \tag{12}$$

Upon finding the coefficients we can simply extend the domain of F and prescribe values at all points $z \in \Omega$, namely we have that the extension is

$$F^s(z) = \sum_{j=1}^{n} K(z_i, x_j) a_j^s \tag{13}$$

for all $1 \leq i \leq n$ and each colour $s = r, g, b$. After substituting the expression (10) into (12), we can define $J = J(a^s)$ as the function to be minimised with resepect to a^s. We have

$$J(a^s) = \frac{1}{n} \sum_{i=1}^{n} \left[\sum_{j=1}^{n} K(x_i, x_j) a_j^s - f^s(x_i) \right]^2 + \delta \sum_{i,j=1}^{n} K(x_i, x_j) a_i^s a_j^s. \tag{14}$$

We assume that J is minimised at a turning point, differentiating with respect to a_k^s gives

$$\begin{aligned}\frac{\mathrm{d}J}{\mathrm{d}a_k^s} &= \frac{2}{n}\sum_{i=1}^{n} K_{ik}\left[\sum_{j=1}^{n} K_{ij}a_j^s - f_i^s\right] + 2\delta\sum_{i=1}^{n} K_{ik}a_i^s \\ &= 2\sum_{i=1}^{n} K_{ik}\left[\frac{1}{n}\left(\sum_{j=1}^{n} K_{ij}a_j^s - f_i^s\right) + \delta a_i^s\right]\end{aligned}$$

for every $1 \leq k \leq n$, where $K_{ij} = K(x_i, x_j)$. So at turning points we have

$$\begin{aligned} & 2\sum_{i=1}^{n} K_{ik}\left[\frac{1}{n}\left(\sum_{j=1}^{n} K_{ij}a_j^s - f_i^s\right) + \delta a_i^s\right] = 0 \\ \iff \quad & \sum_{i=1}^{n} K_{ik}\left[\sum_{j=1}^{n} K_{ij}a_j^s - f_i^s + \delta n a_i^s\right] = 0\end{aligned}$$

for every k. In particular we require

$$K_D(F^s - f^s + \delta n a^s) = 0 \tag{15}$$

where $K_D \in \mathbb{R}^{n\times n}$ has entries $(K_D)_{ij} = K(x_i, x_j)$, $(F^s)_i = F^s(x_i)$ and $(f^s)_i = f^s(x_i)$ for all $1 \leq i, j \leq n$. We assume K_D is non-singular and so require $F^s - f^s + \delta n a^s = 0$. Noting that by the definition (10) of F we have $(F^s)_i = (K_D a^s)_i$, it follows that this is equivalent to the system

$$(K_D + \delta n I)a^s = f^s \tag{16}$$

where we can conveniently solve for coefficients in each colour channel. Then the extension is given by (13). It is important to note that the choice of $\mathcal{W} = \mathbb{R}^3$ means that the extension may not take 8-bit values. When implementing this into Matlab, we simply account for this by converting any resulting real numbers to 8-bit.

4.4 Choosing the Kernel

We finally consider perhaps the most important element of this formulation - the kernel. We choose the general kernel defined by

$$K(x, y) = \phi\left(\frac{||x - y||_2}{\sigma_1}\right)\phi\left(\frac{|\gamma(x) - \gamma(y)|^p}{\sigma_2}\right) \tag{17}$$

where $\sigma_1, \sigma_2 > 0$ are constants, $p \in [0, 1]$ is a constant and $\gamma(x)$ gives the grey-scale intensity of pixel $x \in \Omega$. Here $\phi = \phi(r)$ depicts a general radial basis function (RBF) whose maximum is attained at $r = 0$. The choice of a radial basis function allows us

to govern contributions to the kernel by functions of distance from the current pixel. We mean distance both in the sense of euclidean distance in the space of pixels and the 'grey-scale' distance in terms of differences in grey-scale intensities. This seems like a reasonable approach since we would expect pixels of unknown colour that are close to a pixel whose colour is known to have a similar colour (meaning a larger contribution to the kernel). Similarly if a pixel in the entire domain has a grey-scale value close to the grey-scale value that a coloured pixel would have (determined by the weighted grey-scale map, for example), we would also expect that pixel to have a similar colour and so it would also provide a large contribution to the kernel. These concepts are clearly not exhaustive - a nearby pixel could be a completely different colour, and grey-scale intensities can be close even if the colour-intensities in each channel are very different. As such we can use the parameters σ_1 and σ_2 to control the relative contributions of pixel 'closeness' in each case respectively. We shall later consider how varying these parameters effects the error in the recoloured image. in the following sections, we investigate both the Gaussian radial basis function

$$\phi(r) = e^{-r^2} \in C^\infty, \tag{18}$$

and the compactly supported radial basis function

$$\phi(r) = (1-r)_+^4(4r+1) \in C^2, \tag{19}$$

where $x_+ = \max(x, 0)$. We shall consider the errors in the recoloured images for both of these functions with varying parameters.

4.5 Method

We consider the computational method used to test the recolouring method. We import images of size $m = L \times W$ into Matlab using the *imread()* function. This returns the $L \times W \times 3$ (uint8) tensor where each face represents a colour channel whose elements give the respective intensities at that pixel. We denote this tensor by O. For convenience in calculations, we store red, blue and green channels as individual matrices and convert them to type double, however figure 1 shows their representations as uint8-types. The individual matrices only cover a single channel so they appear grey-scale. It is not difficult to notice that the resulting grey-scale images are darker in locations where the respective colour is not prominent.

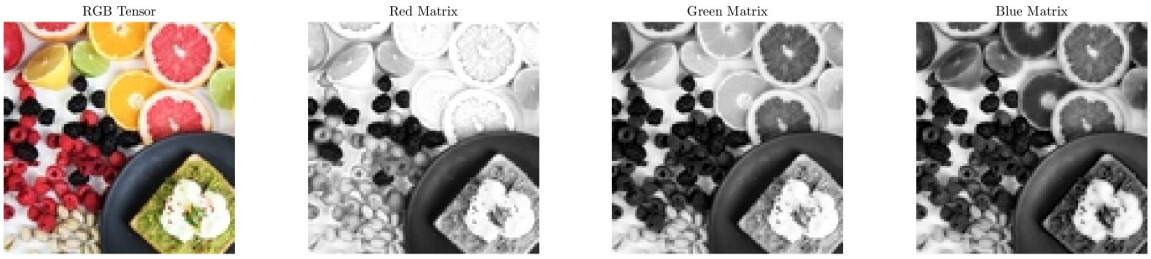


Figure 1: Visualising the stored uint8-type tensor and matrices using the *imshow()* function. Source image taken from [6].

Denoting the colour matrices as R, G and B respectively, we compute the single uint8 matrix representing grey-scale conversion of the colour image using one of the methods outlined in section 3. Namely, to find the grey-scale matrix Γ we compute $\Gamma_{ij} = 0.299R_{ij} + 0.587G_{ij} + 0.114B_{ij}$ for all $0 \leq i \leq L$ and $0 \leq j \leq W$, using the 'realistic' grey-scale mapping as an example. We note that at this point we retain the grey-scale matrix as type double for later calculations.

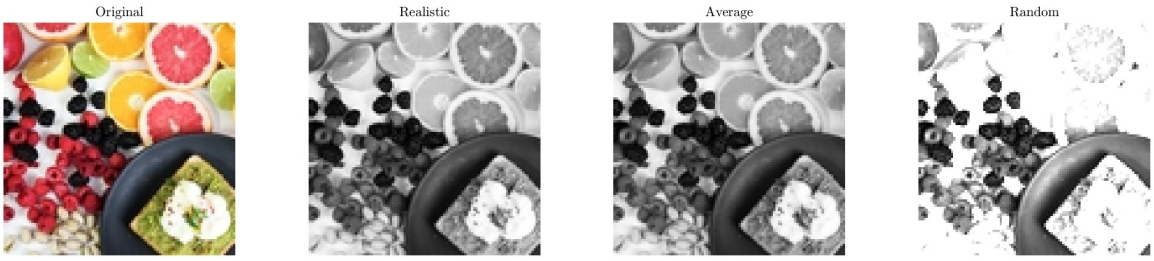


Figure 2: Comparisons of different grey-scale maps. The random weightings were found to be 0.3479, 0.0082 and 0.6440 for red, green and blue respectively. Source image taken from [6].

Figure 12 shows three different grey-scale maps including one with random coefficients that sum to one. We note that the difference between the average and realistic maps are very subtle in this case. Although we do compare results of these different maps in the next section, for the most part we work with the realistic map. We next partially recolour this full grey-scale matrix with the exact colour information stored in the R, G and B matrices. This can be done in several methods. The main method we make use of randomly selects 5% of pixels to prescribe colour by looping through the indices and, with probability 0.05, appending the current index to an array of 'colour indices', C_I. For viewing purposes we can create a new $L \times W \times 3$ tensor $\tilde{F}$. At the indices stored in C_I we let the entries in each face of the tensor equal that of the original image tensor, and for other indices we simply let the entry in each face

equal the grey-scale value prescribed by Γ. Thus we have

$$\tilde{F}_{ijk} = \begin{cases} O_{ijk} & (i,j) \in C_I \\ \Gamma_{ij} & \text{otherwise.} \end{cases}$$

Though this allows us to view the recoloured image, and is thus needed for implementation in a GUI, for calculations it is sufficient to store only the indices in C_I to retrieve colour information.

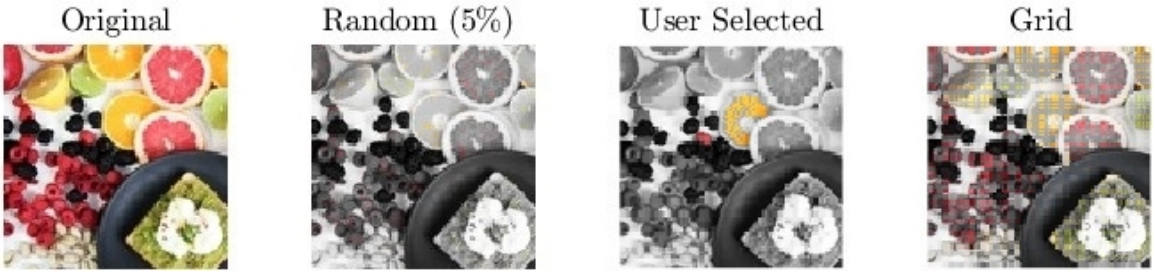


Figure 3: Comparisons of different partial-recolour maps. Source image taken from [6].

Now we have all the information needed to employ the RKHS solution to the variational problem. We initially prescribe values for the parameters σ_1, σ_2 , p and δ. We consider results for both kernels (18) and (18). With m already defined as the total number of pixels, we also have that n is simply the number of coloured pixels which is calculated by taking the length of the colour index array C_I. Then f^r, f^g and f^b are given by the first, second and third faces of $\tilde{F}$ at indices in C_I, where we can simply convert subscript indices (i,j) to linear indices to be consistent with the definitions of these functions. We can easily construct the $n \times n$ kernel matrix K_D by evaluating (17) with either (18) or (19) for elements in C_I, where we can extract the index corresponding to the coloured pixels and use Γ evaluated at these indices to get the grey-scale value $\gamma(x)$. The euclidean distance follows from using the subscript indices of two coloured points, (x_i, x_j) and (y_i, y_j). We note that we only evaluate half of the points in the kernel matrix since it is symmetric. We can simply use the backslash command to solve the linear systems $(K_D + \delta n I_n)a^s = f^s$ for each colour and obtain the necessary coefficients. Finally, we compute the colour s at each z indexed by (i,j) for $0 \leq i \leq L$ and $0 \leq j \leq W$ by computing (13) for computed coefficients for $s = r, g, b$. We end up with three $L \times W$ matrices which we can concatenate into the required tensorial representation using the *cat()* function.

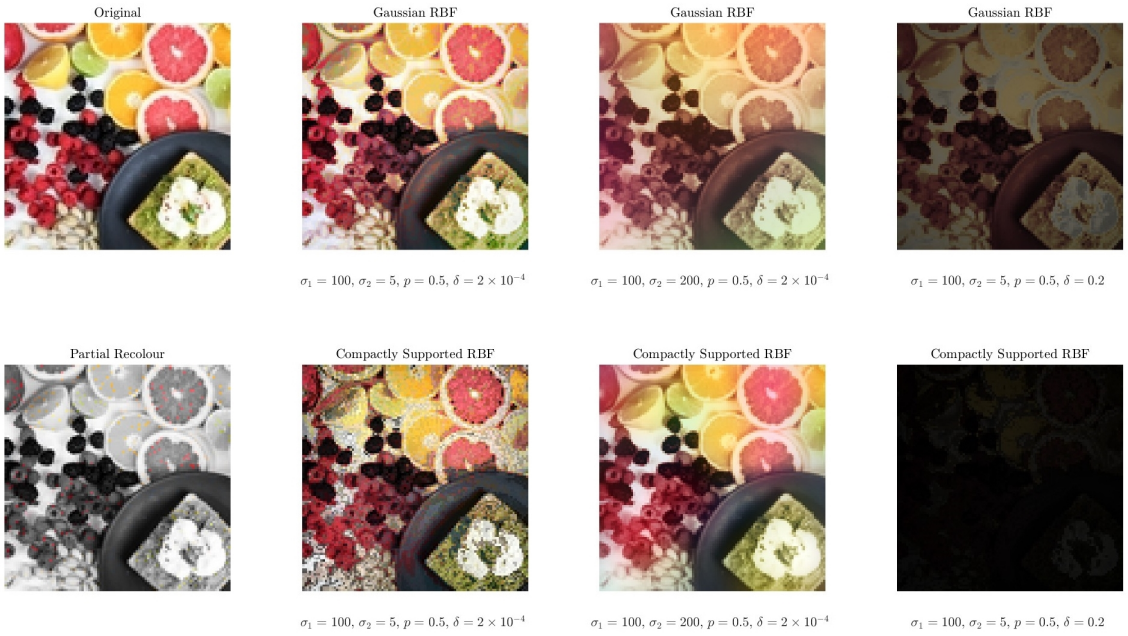


Figure 4: Several forms of recolour of the original image given the shown random subset of partially coloured pixels (5%). We see results for both the Gaussian radial basis function (18) and the compactly supported radial basis function (19) with varying parameters σ_1, σ_2 , p and δ. Source image taken from [6].

Figure 4 shows the results of the algorithm for a grey-scale image with a random selection of 5% of pixels retaining exact colour information. We see that given $\sigma_1 = 100$, $p = 0.5$, and $\delta = 2 \times 10^{-4}$, a lower value of $\sigma_2 = 5$ seems to provide a more clear recolour with the Gaussian RBF, with the compactly supported RBF recolour appearing to cluster colours significantly which forms a less continuous image. When we increase to a higher value of σ_2, the compactly supported RBF seems to provide a better recolour, with the Gaussian recolour appearing continuous however lacking distinction between edges in the image. These results seem akin to the relative smoothness of each function (C^∞ and C^2) and one could postulate that this is not a coincidence. Finally we also notice that a significant increase in δ appears to simply darken the final recolour, with a slight favouring in recovery towards the Gaussian RBF given these parameters. These are perhaps not the most salubrious ways of defining how 'good' a final recolour is, we thus investigate the effects of varying parameters more rigorously in the next section by defining a measure of error.

5 Investigating Parameters

We next investigate the optimal parameters for this least squares approach to the colourisation problem. To be able to quantify how 'good' a resulting coloured image is in comparison to the original, we define an error that we shall use throughout this section, this is based on the Frobenius norm of the differences between the original coloured channels and the final coloured channels. In particular, we have that the squared error is defined by

$$e_F^2 = \frac{1}{3m} \sum_{s=r,g,b} \sum_{j=1}^{m} (F^s(z_i) - f^s(z_i))^2, \tag{20}$$

where we recall that m is just the total number of pixels. By parameters here, we also mean the type of grey-scale map used initially on the original image, since this will naturally provide different grey-scale values to each pixel based on the method used. Since the results of using different grey-scale maps are not particularly interesting, we leave the discussion to appendix A. We note the results of this are that the two considered grey-scale maps do not significantly differ in error. We instead consider optimising the parameters σ_1, σ_2, p and δ for both the Gaussian RBF (18) and the compactly supported RBF (19). Throughout this section we obtain results using three sample images. These are relatively different images with key features that may help signify the effect of changing different parameters or functions involved in the problem. These images are shown in figure 5.

(a) *'field'* (b) *'cats'* (c) *'rainbow'*

Figure 5: Sample images. 5a: Taken from [2]. 5b Taken from [5]. 5c Taken from [8]

5.1 Optimising Parameters

We first consider a brute force method to optimising parameters σ_1 and σ_2 by varying them each between 10 and 1000, just to get a feel for the relative importance of each parameter for each image. We keep $\delta = 2 \times 10^{-4}$ and $p = 0.5$ fixed for now. We see from figure 6 the brute force approach to optimising σ_1 and σ_2. It appears that

the effects of changing parameters are certainly image dependent. Despite this, given our sensitivity in the variation of each parameter, the optimal results are not too dissimilar.

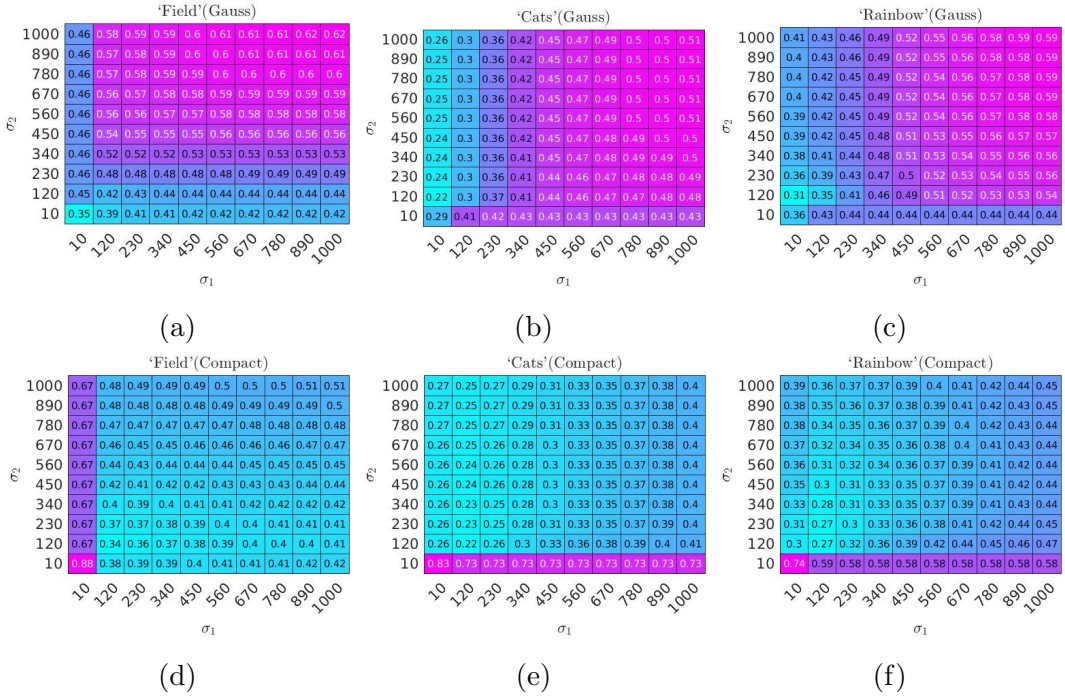


Figure 6: Error heat-maps for each image and each RBF. We choose to vary σ_1 and σ_2 by a linear separation of 10 points ranging between 10 and 1000. The numbers on each square represent the error found given the RBF, image and parameters σ_1 and σ_2. (6a - 6c): Gaussian RBF. (6d - 6f): Compactly supported RBF.

We have the 'optimal' values for σ_1 and σ_2 for the Gaussian RBF as follows. For the *'field'* image: $\sigma_1 = 10$, $\sigma_2 = 10$; for the *'cats'* image: $\sigma_1 = 10$, $\sigma_2 = 120$ and for the *'rainbow'* image: $\sigma_1 = 10$, $\sigma_2 = 120$. And for the compactly supported RBF we have as follows. For the *'field'* image: $\sigma_1 = 120$, $\sigma_2 = 120$; for the *'cats'* image: $\sigma_1 = 120$, $\sigma_2 = 120$ and for the *'rainbow'* image: $\sigma_1 = 120$, $\sigma_2 = 120$ or $\sigma_2 = 230$. We stress that the steps between parameter values are fairly large since this method is just used to get an idea of what the values should be. In what follows we try to determine these results more rigorously. We consider optimising the parameters $\sigma_1, \sigma_2, \delta$ and p for different images and for each radial basis function. We start by formally presenting the problem to be solved. We have the constrained minimisation problem

$$\min_{(\sigma_1,\sigma_2,\delta,p)} e_F(\sigma_1, \sigma_2, \delta, p) \quad \text{subject to} \quad p \in [0,1], \sigma_1, \sigma_2, \delta > 0$$

for a given image. The constraints can be written in matrix form[2] as

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \delta \\ p \end{bmatrix} \leq \begin{bmatrix} -\epsilon \\ -\epsilon \\ -\epsilon \\ 1 \\ 0 \end{bmatrix} \tag{21}$$

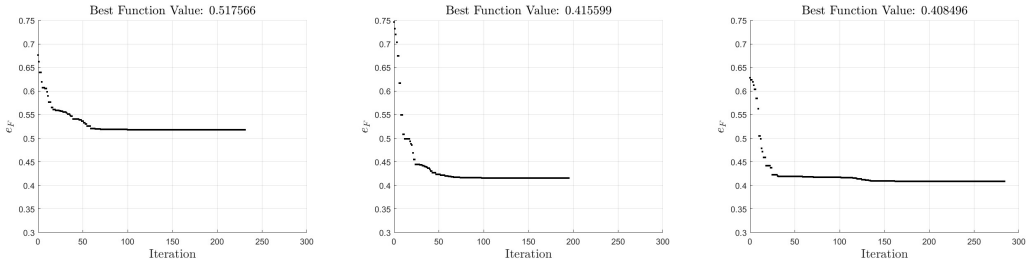
where we note the change from strict inequality and the inclusion of some arbitrarily small constant $\epsilon > 0$. Though an equivalent formulation, writing the constraints in this way makes the problem convenient to be solved using the Global Optimisation Toolbox in Matlab[3]. In particular we employ the *pattern search* algorithm as it uses derivative free methods at each iteration and can be used in a multidimensional search space[3], thus being very convenient for our problem. We solve the constrained optimisation problem to find parameters σ_1, σ_2, δ and p under the prescribed constraints for each image and each RBF. In each case we take the initial parameters in the algorithm as the 'optimal' parameters suggested previously for σ_1 and σ_2, and take initial values for δ and p as used before. Table 1 shows the optimal parameters attained by the algorithm. There does not seem to be any clear relationship between the parameters in the different cases, however we note that δ is indeed very small in all cases. Figure 7 shows the extent of the optimisation algorithm. An interesting point to note is the lower error over all iterations and images for the compactly supported RBF. A possible reason why this may be the case is because we are generally considering very small images, which have been re-scaled to significantly reduce CPU times. Thus the images we use are typically less continuous in terms of changes in colour.

Table 1: Optimal parameters of each image.

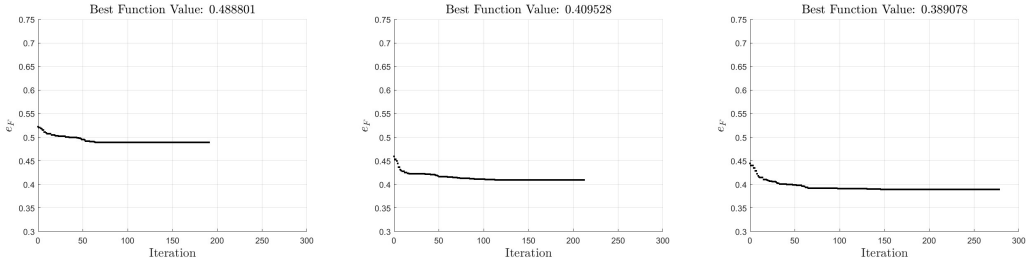
	Gaussian RBF			**Compactly Supported RBF**		
	'field'	*'cats'*	*'rainbow'*	*'field'*	*'cats'*	*'rainbow'*
σ_1	22.4	30.8	24.7	85.0	81.5	109
σ_2	133	6.92	166	5.88	28.1	1030
δ	8.48×10^{-5}	8.00×10^{-5}	4.26×10^{-5}	3.95×10^{-5}	8.51×10^{-6}	2.59×10^{-6}
p	0.996	0.418	0.940	1.00	0.507	1.00

[2]Where inequalities between vectors are defined element-wise.

[3]We simply take ϵ in the numerical solution as the built in machine epsilon in Matlab $\sim 2.2 \times 10^{-16}$.



(a) *'field'* (Gauss) (b) *'cats'* (Gauss) (c) *'rainbow'* (Gauss)



(d) *'field'* (Compact) (e) *'cats'* (Compact) (f) *'rainbow'* (Compact)

Figure 7: Visualisation of the *pattern search* algorithm for finding the optimal solution for all images and each RBF.

6 Extension

As an extension, we re-evaluate one of the key assumptions made when devising the solution to the variational problem. Recall that we found the minimising coefficient vector a^s to the least squares problem

$$F^s = \arg\min_{a^s \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} (F^s(x_i) - f^s(x_i))^2 + \delta \sum_{i,j=1}^{n} K(x_i, x_j) a_i^s a_j^s. \tag{22}$$

by assuming minima occur at turning points, thus leading to the system

$$(K_D + \delta n I) a^s = f^s \tag{23}$$

which can be solved trivially. The issue here is the assumption that minimisers occur at a turning point, since (22) is not actually an unconstrained minimisation problem. Instead we need to find the coefficients under the constraint that resulting colour values are 'true colour', that is 8-bit. Thus we can write the problem as

$$\min_{a^s \in \mathbb{R}^n} J(a^s) \quad \text{subject to} \quad 0 \leq F^s(z_k) = \sum_{j=1}^{n} K(z_k, x_j) a_j^s \leq 255 \tag{24}$$

for all $1 \leq k \leq m$ and $s = r, g, b$, where the objective, J, is defined by

$$J(a^s) = \frac{1}{n} \sum_{i=1}^{n} \left[\sum_{j=1}^{n} K(x_i, x_j) a_j^s - f^s(x_i) \right]^2 + \delta \sum_{i,j=1}^{n} K(x_i, x_j) a_i^s a_j^s. \tag{25}$$

Then, the extension would again be given by (13). Such a formulation of the problem is not so convenient for computing the solution numerically so we need to tidy things up. We first split the constraints into a more suitable form. We have that if k takes all values $1 \leq k \leq m$,

$$\sum_{j=1}^{n} K(z_k, x_j) a_j^s \leq 255 \implies \tilde{K} a_j^s \leq \begin{bmatrix} 255 \\ \vdots \\ 255 \end{bmatrix} \in \mathbb{R}^m$$

$$\text{and} \quad \sum_{j=1}^{n} K(z_k, x_j) a^s \geq 0 \implies -\tilde{K} a^s \leq \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^m,$$

where we define the $m \times n$ inequality constraint matrix $\tilde{K}_{pq} = K(z_p, x_q)$. We also write the vectors on the right hand side as $\mathbf{255}$ and $\mathbf{0}$ for brevity. Note that in this context inequalities between vectors are defined component-wise. The problem is then recast as

$$\min_{a^s \in \mathbb{R}^n} J(a^s) \quad \text{subject to} \quad \begin{bmatrix} \tilde{K} \\ -\tilde{K} \end{bmatrix} a^s \leq \begin{bmatrix} \mathbf{255} \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^{2m} \tag{26}$$

for each $s = r, g, b$. In a practical setting we simply define the objective J as a function in Matlab and define the inequality in matrix form $Ax \leq b$, where now it is very convenient to compute A because it is simply defined by the kernel function we use, similarly we can easily set b. Such definitions are the canonical forms for constraints required by most constrained optimisation solvers in the Global Optimisation Toolbox. As explained previously, problems of this sort can be solved using the *pattern search* algorithm. This alternate method will essentially find the 'true' minimum to the solution given sufficient iterations. The key word here is 'sufficient'. A significant downside to this method is the computation time due to the number of constraints. This trade-off suggests that in a practical setting, for larger images we should employ the derivative method and for smaller images we should use this optimisation method.

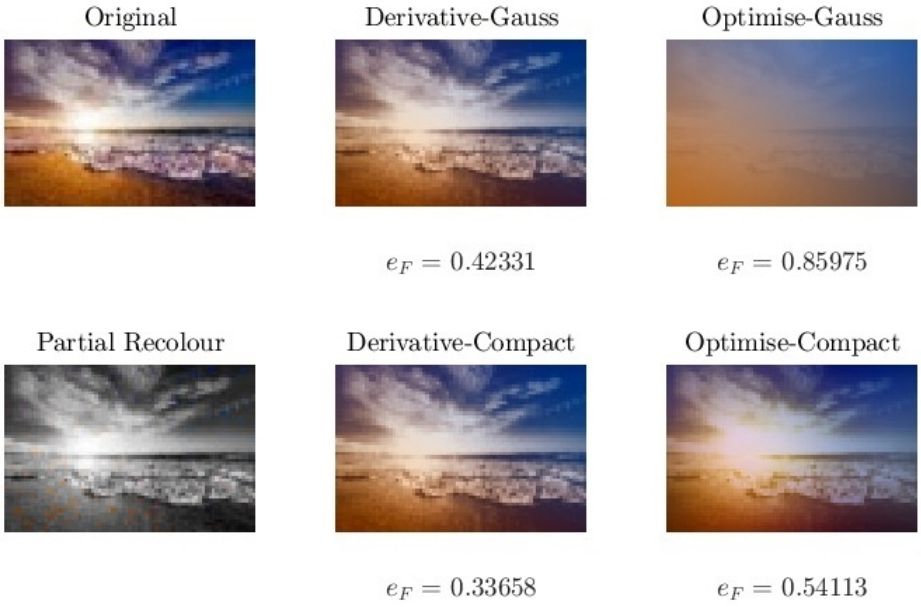


Figure 8: Comparisons of the derivative-based solution with the constrained optimisation problem. Parameters used in all cases: $\sigma_1 = \sigma_2 = 100$, $\delta = 2 \times 10^{-4}$, $p = 0.5$. The optimisation algorithm was prescribed 2000 maximum iterations and terminated by reaching this limit in each case. Image taken from [4].

Figure 10 shows a comparison between the optimisation method presented in this section and the standard derivative method. We note that the starting point of the *pattern search* algorithm was a vector of ones in both cases. We see that given 2000 iterations, the optimisation method failed to beat the error in the derivative method for both radial basis functions, with the compactly supported RBF performing better for this image in both cases. What makes the derivative method seem even more superior is the fact that it took < 0.5s to run, whereas on the same machine the optimisation method took ~ 15 minutes for each RBF. It is important to mention that we prescribed this maximum number of iterations for the sake of computation times, and so the algorithm terminated based on this condition, and not on the condition that a minimum was found. Therefore perhaps not all is lost with this optimisation method - of course it will always take longer to run however maybe this is acceptable if the overall resulting image has a reduced error. One way we could improve the speed is by coupling the derivative method with this optimisation method.

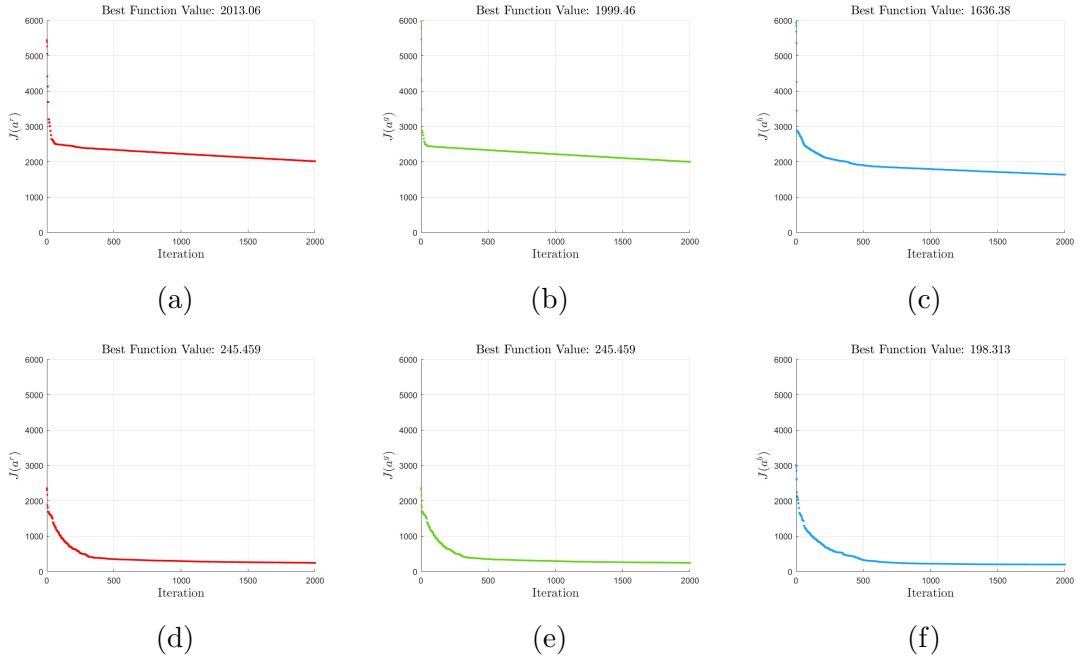


Figure 9: Visualisation of the *pattern search* algorithm over 2000-iterations for finding coefficients a^s for both RBFs. (9a - 9c): Gaussian RBF and (9d - 9f): compactly supported RBF.

6.1 Coupled Derivative and Optimisation Method

The fact that the derivative method provides a very accurate colourisation relative to the optimisation method in much less time, but the optimisation method has the potential[4], given sufficient iterations, to provide a better colourisation motivates this method. In this method we simply use the derivative method to obtain coefficients that can be used as the initial guess in the optimisation algorithm, as such we are guaranteed (given at least one successful iteration) to achieve a lower minimiser to the objective[5]. We have that the initial value in our *pattern search* algorithm satisfies

$$(K_D + \delta n I)\, a_0^s = f^s. \tag{27}$$

An important additional quality of this method is that it will remove the need to compute the solution to a constrained optimisation problem in the case that the derivative method yields a solution within the feasible region, this can be seen in 11b

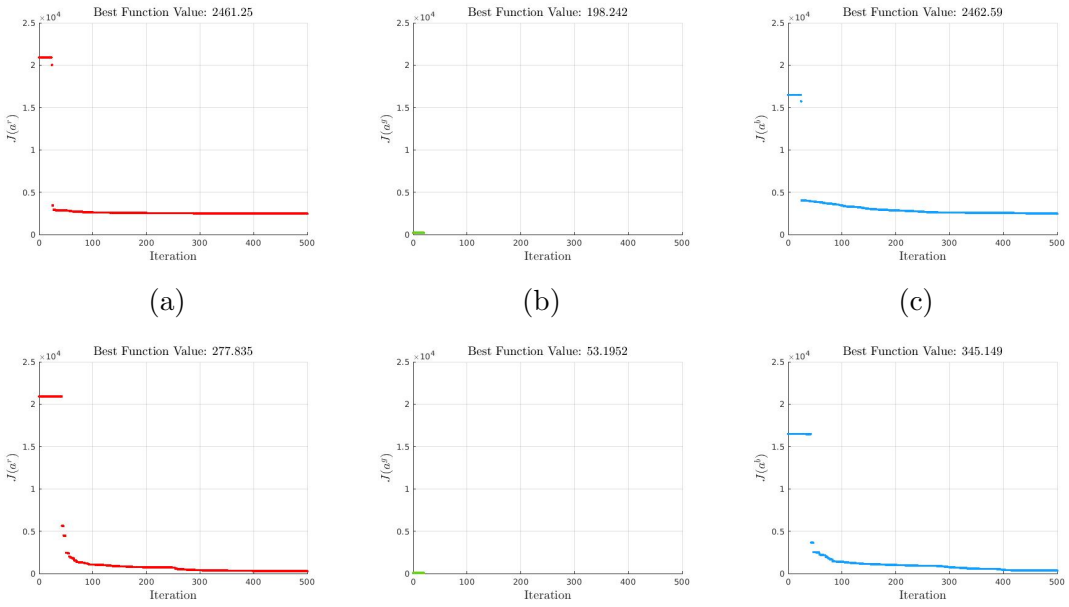
[4]Of course we have not guaranteed that convergence to the true minimum will occur, however for the sake of argument we assume that in most cases it will, eventually, provide a reduced error.

[5]It is important to note that this does not necessarily imply a lower error, in fact in our cases it does not.

and 11e, where the objective function converges almost instantly to a minimiser. Thus one could simply prescribe a required degree of accuracy in the recoloured image and this method could perform optimisation on top of the derivative method if necessary to achieve (or come as close as possible) the value.

Figure 10: Comparisons of the derivative-based solution with the coupled derivative-constrained optimisation problem. Parameters used in all cases: $\sigma_1 = \sigma_2 = 100$, $\delta = 2 \times 10^{-4}$, $p = 0.5$. The optimisation algorithm was prescribed 500 maximum iterations. Image taken from [4].

Figure 10 shows the effect of the coupling on the given image and for each RBF. We note that in this run of the algorithm, we only prescribed 500 maximum iterations. Despite this, the relative error between the derivative method and the constrained method has improved on the introduction of the new starting value, even with only 500 iterations. Figure 11 shows the *pattern search* algorithm, where we note the instant convergence of the objective for a^g, suggesting that $(K_D + \delta nI)^{-1} f^g$ is in the feasible set.



(a) (b) (c)

(d) (e) (f)

Figure 11: Visualisation of the *pattern search* algorithm over 500-iterations for finding coefficients a^s for both RBFs with initial guess given by the derivative method. (9a - 11c): Gaussian RBF and (9d - 11f): compactly supported RBF.

7 Summary

We have introduced the basic ideas behind computational image storage and applications of this through manipulating matrix and tensor representations. Through a detailed review of the literature, we have motivated the connection between image colour recovery and the very abstract field of reproducing kernel Hilbert spaces. We then applied notions from this field to solve the extension problem set by mathematical formalism of image colour recovery, and implemented this in Matlab. We investigated the variation of several parameters set by this problem, and motivated the effects some may have on the final recovered image. As an extension, we considered an alternate approach to solving the minimising problem that emerges. Namely we treated this as a constrained optimisation problem, given the 'true colour' constraint. We compared this approach to the derivative based approach, concluding that it performs worse even when given significantly more CPU time. As a means to improve these computation times, we proposed a coupled model that uses derivative based solutions as starting values in the optimisation algorithm, which we showed to improve the errors given fewer maximum iterations.

References

[1] Riyadh M. Al-saleem, Baraa M. Al-Hilali, and Izz K. Abboud. "Mathematical Representation of Color Spaces and Its Role in Communication Systems". In: *Journal of Applied Mathematics* 2020 (2020), p. 4640175. ISSN: 1110-757X. DOI: 10.1155/2020/4640175. URL: https://doi.org/10.1155/2020/4640175.

[2] Kohji Asakawa. *Flower field flowers - free photo on Pixabay*. URL: https://pixabay.com/photos/flower-field-flowers-field-trees-250016/.

[3] Charles Audet and J. E. Dennis. "Analysis of Generalized Pattern Searches". In: *SIAM Journal on Optimization* 13.3 (2002), pp. 889–903. DOI: 10.1137/S1052623400378742. eprint: https://doi.org/10.1137/S1052623400378742. URL: https://doi.org/10.1137/S1052623400378742.

[4] *Beach Sea Sunset - free photo on Pixabay*. URL: https://pixabay.com/photos/beach-sea-sunset-sun-sunlight-1751455/.

[5] dandelion_tea. *Cats Pets Animals - free vector graphic on Pixabay*. URL: https://pixabay.com/vectors/cats-pets-animals-flower-domestic-7122943/.

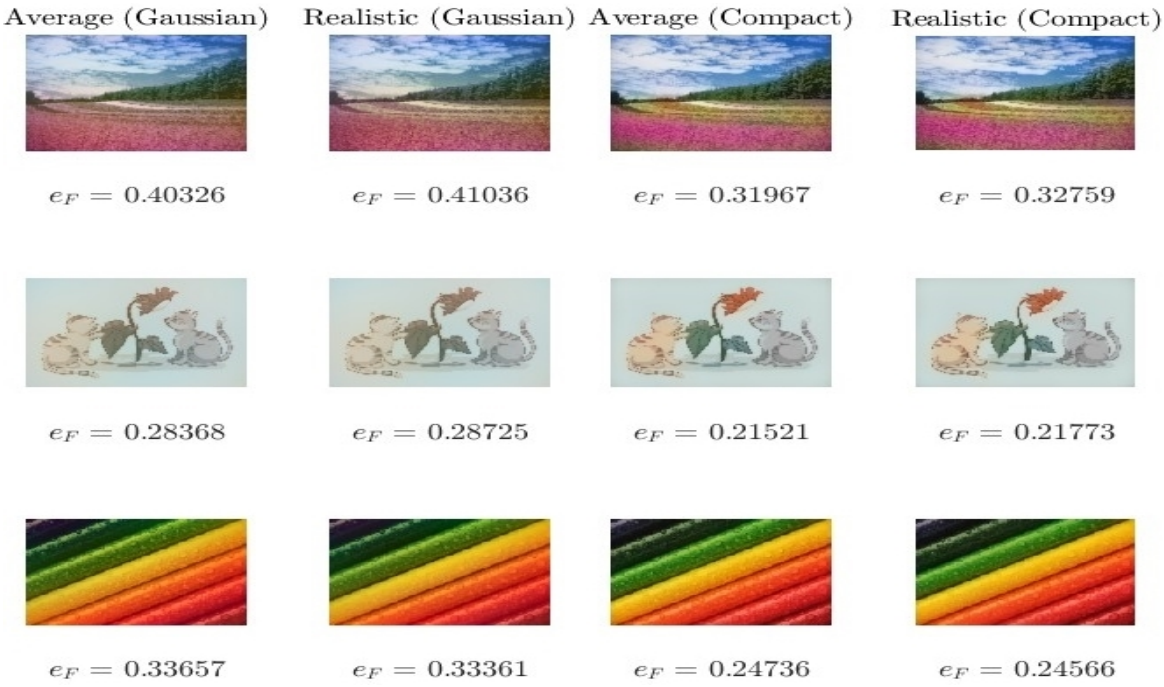
[6] Trang Doan. *Assorted Bundle of Fruits - Free Stock Photo*. URL: https://www.pexels.com/photo/assorted-bundle-of-fruits-793772/.

[7] Minh Ha Quang, Sung Kang, and Triet Le. "Image and Video Colorization Using Vector-Valued Reproducing Kernel Hilbert Spaces". In: *Journal of Mathematical Imaging and Vision* 37 (May 2010), pp. 49–65. DOI: 10.1007/s10851-010-0192-8.

[8] InspiredImages. *Pencils Rainbow Crayons - free photo on Pixabay*. URL: https://pixabay.com/photos/pencils-rainbow-crayons-452238/.

A Varying Grey-Scale Map

Before investigating optimial parameters, we first consider the effect of different grey-scale maps on the final error for each of the radial basis functions. In all cases we use the parameters $\sigma_1 = \sigma_2 = 100$, $p = 0.5$ and $\delta = 2 \times 10^{-4}$. We see from figure 12 the errors and final images when using both the average and realistic grey-scale maps prior to recolouring. Overall we see very minute differences in the errors for the different grey-scale maps for the same choice of RBF. As such we do not discuss further the effect of using a different grey-scale map in this section. A perhaps more interesting observation is the significantly lower error in all cases of using the compactly supported RBF over the Gaussian RBF. We also note that the second image has the lowest error throughout, which perhaps is expected as it appears the 'simplest' in terms of solid boundaries between colours and very few colours in total. It seems to follow that the less 'discrete' the image the worse the error, however this is nothing more than speculation at this point.



(a) Gaussian RBF (b) Compactly supported RBF

Figure 12: Comparisons of the final recoloured image given two different types of initial grey-scale maps. 12a Results for Gaussian RBF. 12b Results for Compactly supported RBF.