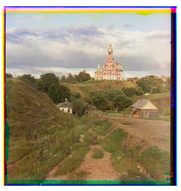
Aligning Images with Exhaustive Search

Using an exhaustive search given a window [-*x*, *x*], the algorithm loops over the x and y axis to determine a shift (dx, dy) such that dx, dy are in [-*x*, *x*]. The ideal shift is chosen by a given metric, two of which I used for the project are sum of squared differences and the zero-normalized correlation coefficient. The ZNCC metric is useful for varying brightnesses in the channels, as the images are first normalized before maximizing their dot product. The exhaustive search is effective for small images such as the jpg files.





```
tobolsk ((2,3),(3,6),2.088016833004076)
monastery ((2,-3),(2,3),2.144031499999828)
cathedral ((2,5),(3,12),2.0376386670031934)
```

Runtime Data

The output format is: `image_name (ag_disp, ar_disp, runtime)`.

- `image_name` : Name of the image
- `ag_disp` : Displacement of the green channel to align with the blue channel. This is a tuple of the form (dx, dy) such that `np.roll(g, ag_disp, (1, 0))` is the realigned green channel.
- `ar_disp` : Displacement of the red channel to align with the blue channel. This is a tuple of the form (dx, dy) such that `np.roll(r, ar_disp, (1, 0))` is the realigned red channel.
- `runtime` : Runtime in seconds.

Image Pyramid

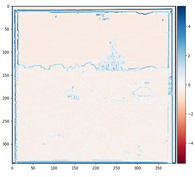
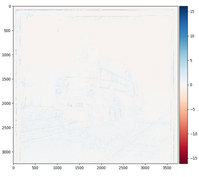
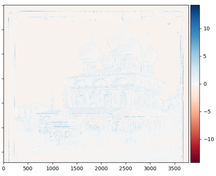
For larger images, larger windows will slow down the result. Using an image pyramid, we can recursively downsample the image (by a factor of 2) to a given depth, at each step aligning the image and returning the shift. Propagation upwards will account for the total displacement of the image, which we can use to shift the image. This makes the process much faster, as at a depth (d), determining the ideal displacement is fastest, and accounts for (# pixel shifts) * 2^d of the displacement. Although faster, this algorithm struggles with the borders that are present in some images. This affects the metric used, making the result inaccurate, especially on train.tif, lady.tif, and emir.tif.




```
emir ((8,27),(17,107),8.82453879196602)
monastery ((1,-3),(1,7),0.09240904200123623)
church ((-5,-1),(-6,53),8.345933124997828)
three_generations ((5,52),(7,108),8.609514625000884)
melons ((4,83),(7,176),8.837978958006715)
onion_church ((22,52),(35,108),8.366027333002421)
train ((-2,41),(2,97),8.19836729200324)
tobolsk ((2,3),(3,6),0.09251366559346511)
icon ((16,42),(22,89),8.482053167004779)
cathedral ((-1,3),(-1,9),0.09229854200838458)
self_portrait ((-2,62),(-4,147),8.955216917005 6)
harvesters ((10,59),(7,120),8.143220707999717)
sculpture ((-11,33),(-26,140),8.693139582996082)
lady ((-8,57),(-17,113),8.282611499998893)
```

Edge Detection for Alignment

One improvement is to use the sobel filter to detect edges in the image. We can preprocess the image with the filter, feed this into our alignment algorithm and obtain a result more resistant to border colors. The filter uses the kernel matrix `[[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]`, derived from adjustments to the discrete form of the derivative: `f_x(x, y) = (f(x + 1, y) - f(x - 1, y)) / 2`. The kernel is convolved with the image matrix to produce the x component of the gradient. The same is done for the y component, and the normalized euclidean distance between both images is used to obtain the edges. Here are some examples of using `scipy.signal.convolve2d` to do the convolution then finding the magnitude:

And here is the result using `sk.filters.sobel`:




```
emir ((23,49),(40,107),8.819455124998058)
monastery ((0,-3),(1,3),0.08822054199845297)
church ((-2,25),(-14,54),8.329349291001563)
three_generations ((0,54),(7,107),8.813686957997561)
melons ((5,78),(11,177),8.124893291998887)
onion_church ((24,52),(35,107),8.005964750002022)
train ((1,41),(29,85),7.973955954199601)
tobolsk ((2,3),(3,6),0.11289779200160 41)
icon ((16,40),(23,90),7.895021333002666)
cathedral ((-1,5),(-3,17),0.09654333300568396)
self_portrait ((-3,79),(37,176),7.863586999999825)
harvesters ((-6,71),(4,123),7.604994499977015)
sculpture ((-11,33),(-26,140),8.136666292004 52)
lady ((-10,59),(-21,77),7.642337625002256)
```

Naive Cropping and Results

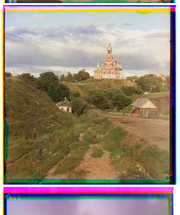
To achieve stronger results, cropping the image by 15% on all sides can remove the issue. This is the final improvement of the program and here are all results listed:











```
emir ((24,49),(40,107),3.607143584000296)
monastery ((2,-3),(2,3),0.05113158300082432)
church ((4,25),(-4,58),3.3933759999999893)
three_generations ((12,53),(9,111),3.8013116670044838)
melons ((10,80),(13,177),3.582036874999176)
onion_church ((26,51),(36,107),3.538803707997431)
train ((2,42),(30,86),3.5046957909944467)
tobolsk ((3,3),(3,6),0.05030783300026087)
icon ((17,42),(23,90),3.546664124995044)
cathedral ((2,5),(3,12),0.048385875001 25868)
self_portrait ((29,78),(37,176),3.664792374998 7967)
harvesters ((17,60),(14,124),3.449887957998726)
sculpture ((-11,33),(-27,140),3.623725582998304)
lady ((9,56),(13,119),3.5158652090003955)
```