Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

# Computer Vision Assignment

Robotics 814
Report

Dylan Charl Eksteen
22623906

May 11, 2023

Department of Mechanical and Mechatronic Engineering
Stellenbosch University
Private Bag X1, Matieland 7602, South Africa.

# Plagiarism declaration

I have read and understand the Stellenbosch University Policy on Plagiarism and the definitions of plagiarism and self-plagiarism contained in the Policy [Plagiarism: The use of the ideas or material of others without acknowledgement, or the re-use of one's own previously evaluated or published material without acknowledgement or indication thereof (self-plagiarism or text-recycling)].

I also understand that direct translations are plagiarism, unless accompanied by an appropriate acknowledgement of the source. I also know that verbatim copy that has not been explicitly indicated as such, is plagiarism.

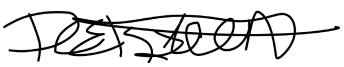
I know that plagiarism is a punishable offence and may be referred to the University's Central Disciplinary Committee (CDC) who has the authority to expel me for such an offence.

I know that plagiarism is harmful for the academic environment and that it has a negative impact on any profession.

Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully (acknowledged); further, all verbatim copies have been expressly indicated as such (e.g. through quotation marks) and the sources are cited fully.

I declare that, except where a source has been cited, the work contained in this assignment is my own work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment. I declare that have not allowed, and will not allow, anyone to use my work (in paper, graphics, electronic, verbal or any other format) with the intention of passing it off as his/her own work.

I know that a mark of zero may be awarded to assignments with plagiarism and also that no opportunity be given to submit an improved assignment.

Signature: 

Name: DC Eksteen Student no: 22623906

Date: 11/05/2023

# Table of contents

# List of figures

# Assignment Overview

This report is for completing the Computer Vision Assignment for Robotics 814. The project aimed to explore different applications and techniques of computer vision.

The project was implemented using Python scripts and the openCV library containing the required tools to complete the assignment.

## Question 1: Stereo vision

1. Use your phone to take two photographs. The photographs will serve as a stereo-vision dataset.
2. Calculate the disparity map.
3. Calculate and match SIFT features.
4. Calculate the disparity between the matched features.
5. Compare the result in 2 with the result in 4.
6. Discuss the results.

## Question 2: Where's Wally

1. Download at least three "Where's Wally" posters.
2. Download or create a template for Wally.
3. See if template matching can be used to find Wally in the three posters.
4. Discuss the results. (If you find that template matching cannot detect Wally, explain why that is the case and discuss the objects that the algorithm identifies as Wally).

## Question 3: Filter Smarties

1. Put a handful of Smarties on a desk.
2. Use your phone to take two photographs (P1 and P2) of the Smarties. For P1, the lighting conditions should be very good (a well-lit desk), for P2, it should be reasonable (there should be enough light that you can clearly identify the colour of the Smarties, but there can be shadows or nonideal lighting conditions).
3. Pick two colourspaces (C1 and C2). C1 should work very well to implement a colour filter, while C2 should be a poor choice for a colour filter. Motivate why you expect C1 to work well and C2 to not work well.
4. Create a colour filter in Colourspace C1 to remove all but the brown and red smarties in each photograph.
5. Create a colour filter in Colourspace C2 to remove all but the brown and red smarties in each photograph.
6. Discuss the effectiveness of colourspace and the effect of lighting conditions.

## Submission Requirements

- The code used for the project
- A short report that lists and explains key parts of the code.

# Question 1: Stereo Vision

This question investigates the use of stereo vision in order to demonstrate the application thereof. The main application is in finding the distance to different objects when stereo images are available.

## Dataset Overview

In order to better illustrate the working of the techniques, two sets of stereo images were used. The first was of a bunch of smarties that were placed on a table with a plain background:

Left View Right View

Figure 1: Stereo Image of Smarties

Then another was taken with a more complex background and a book as a second object in the images:


Left View Right View

Figure 2: Stereo Image of Book and Smarties

# Method

Block matching (BM) is a common technique used in stereo vision to estimate the disparity between corresponding pixels in a pair of stereo images. It works based on the assumption that corresponding pixels in the left and right images have similar intensities.

## Block Matching

OpenCV has two block-matching functions as part of its library, stereoBM and stereoSGBM. A general overview of the block-matching process is:

1. Image Preprocessing
    - Convert the left and right images to grayscale if they are not already in grayscale format.
    - This simplifies the matching process by considering pixel intensities only.
2. Block Selection
    - Divide the left image into small blocks or windows of a fixed size (e.g., 3x3, 5x5, etc.). Each block typically contains a few pixels.
    - For each block in the left image, define a corresponding search region in the right image.
    - This search region is the same size as the block and is usually located within a certain disparity range from the block's position.

3. Cost Calculation

    - For each block in the left image, calculate a matching cost between the pixels in the block and the corresponding pixels in the search region of the right image.
    - The cost is usually based on the intensity differences between the pixels.

4. Disparity Estimation

    - Identify the disparity value that corresponds to the lowest matching cost for each block.
    - This disparity value represents the horizontal shift needed to align the block with its corresponding region in the right image.

5. Disparity Refinement

    - Apply post-processing techniques to refine the disparity map.
    - This can involve filtering, interpolation, or other methods to improve the accuracy and smoothness of the disparity values.

## BM vs Semi-Global BM

The two functions can be compared in terms of the algorithm implementation, flexibility and performance.

### Algorithm

StereoBM (Block Matching) is a simpler algorithm that works by comparing blocks of pixels between the left and right images. It assumes that the disparity between corresponding pixels is constant within a block. It uses a window of a fixed size and searches for the best match by comparing pixel intensities.

StereoSGBM (Semi-Global Block Matching) is a more advanced algorithm that improves upon the limitations of StereoBM. It takes into account not only left-to-right matching but also right-to-left matching, which helps to handle occlusions and textureless regions better. It uses a cost aggregation approach and considers multiple disparity hypotheses for each pixel.

### Flexibility

StereoBM has fewer parameters to tune, making it easier to use. The main parameters are the block size and the number of disparities. However, it may not handle challenging scenarios with large disparities or occlusions as effectively.

StereoSGBM offers more parameters for fine-tuning the algorithm. These parameters include the block size, the number of disparities, the uniqueness ratio, the P1 and P2 penalties for the cost aggregation, etc. This flexibility allows for better adaptation to different stereo-image pairs and scene characteristics.

**Performance**

StereoBM is generally faster compared to StereoSGBM because of its simpler matching strategy. It can be suitable for real-time applications or situations where computational resources are limited.

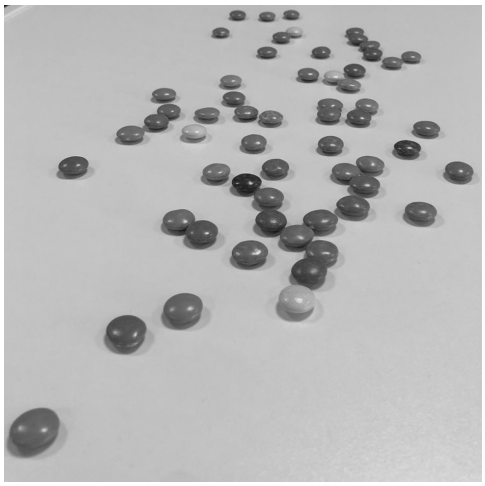
StereoSGBM is more computationally intensive due to its semi-global optimization approach. It involves additional steps such as cost aggregation and disparity optimization, which make it slower compared to StereoBM. However, it often produces more accurate disparity maps in complex scenarios.

In summary, StereoBM is a simpler and faster algorithm with fewer parameters to adjust, while StereoSGBM is more advanced, provides better results in challenging situations, and offers more control through additional parameters.

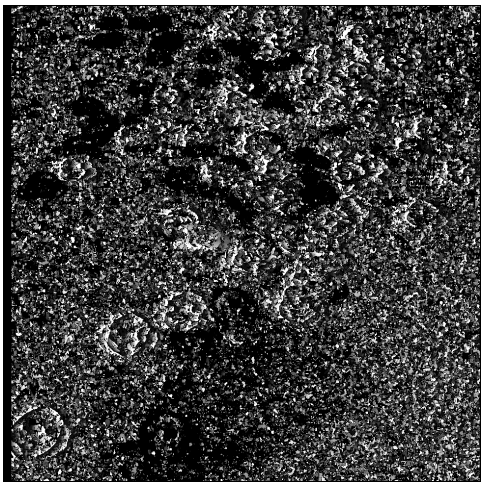
The choice between them depends on the specific requirements of your application, the complexity of the stereo image pair, and the available computational resources. In this case, the normal block matching (StereoBM) was implemented due to its simplicity. The results were compared to SGBM results and no notable improvement was observed, and it was thus omitted from this report.

# Disparity Map

The disparity maps for the two sets of stereo images are shown in Figure 3 and Figure 4 respectively.



Right Image



Disparity Map

Figure 3: Disparity Map of Smarties - BM

Right Image

Disparity Map

Figure 4: Disparity Map of Book and Smarties - BM

In both of the disparity maps above we can see that the objects are faintly recognisable, with objects and features that are closer appearing brighter.

Since the backgrounds of both images do not have a lot of texture, and the contrast between the objects is not very large, we get a lot of noise in the disparity map.

# SIFT Feature Matching

SIFT (Scale-Invariant Feature Transform) is an algorithm in computer vision to detect and describe local features in images. The algorithm was published by David Lowe in 1999. (openimaj, 2020) It is a method for extracting distinctive invariant features from images, which can be used to perform reliable matching between different views of an object or scene.

The stages of computation used to generate these features are: (Singh, 2023)

### Scale-space Extremes Detection

The first step of SIFT features extraction algorithm is to identify key locations in the scale space, which are invariant to scale and orientation. This is done by convolving the image with Gaussian filters at different scales (resolutions), and looking for local maxima and minima of the Difference of Gaussians (DoG) that occur at multiple scales.

Each pixel in the image is compared with its eight neighbours in the current image and nine neighbours in the scale above and below. It is selected only if it is larger than all these pixels or smaller than all these pixels. These key locations are called "candidate keypoints".

### Keypoint Localization

Once candidate key points locations are found, they have to be refined to get more accurate results. This is done by interpolating nearby data for a more accurate position and rejecting points of low contrast or poorly localized on an edge. This increases stability.

### Orientation Assignment

One or more orientations are assigned to each keypoint location based on local image gradient directions to achieve invariance to image rotation. The dominant direction of the gradient is taken as the key point's orientation. This step creates key points, which are invariant to image scale, location, and rotation.

### Keypoint Descriptor

Now, a descriptor is created for each key point to allow for robust matching. This descriptor is a measurement of the local image region around each key point and is computed as a set of orientation histograms on (4x4) pixel neighbourhoods. The orientation histograms are relative to the keypoint orientation and scale, resulting in invariance to these parameters.

**Keypoint Matching**

Keypoints between different images are matched by identifying their nearest neighbours. However, in some cases, the second closest match may be very near to the first. Such cases can give rise to false matches. To eliminate these, a ratio is calculated of the distance from the closest neighbour to the distance of the second closest. If this ratio is greater than 0.8 (as suggested by Lowe), they are rejected. 0.8 has been found through empirical analysis to yield the least number of false matches.

Figure 5 and Figure 6 show the SIFT features that have been detected in the two pairs of images.

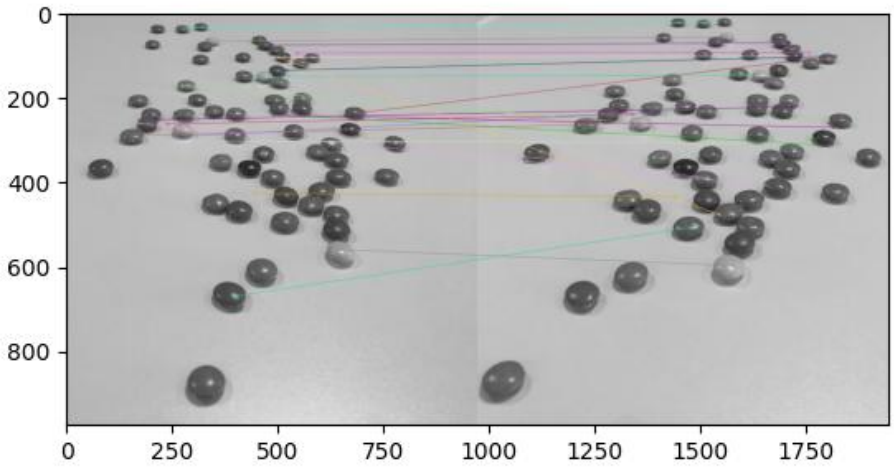



Figure 5: SIFT Features on Smarties

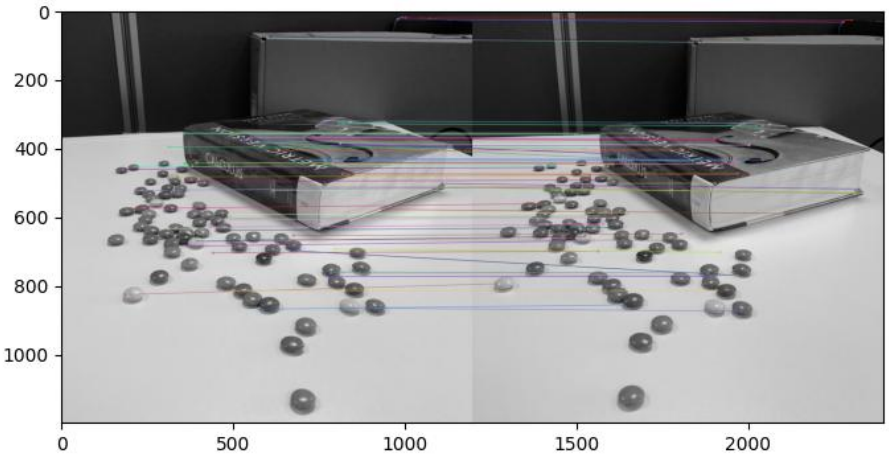



Figure 6: SIFT Features on Book and Smarties

As we can see from these figures it is much better at detecting the accurate features than Block Matching was. This is probably due to the nature of the images and the algorithms involved. Block matching expects that the y-coordinate of the different features remains constant, where SIFT finds the most likely match for certain features independently of position.

Since both images were taken with a smartphone that was held by a hand, the y-coordinates were probably not constant enough for block matching to perform optimally.

A possible solution would be to combine both algorithms, where SIFT is used to identify features and then rectify the images before BM or SGBM is applied to determine the disparity map.

# Code Implementation

The code was implemented in Python and utilises the OpenCV library for computer vision tools.

## Block Matching

```
def update_disparity_map(block_size, num_disparities):
    # Ensure the number of disparities is divisible by 16
    if num_disparities % 16 != 0:
        num_disparities = (num_disparities // 16) * 16

    # Create a StereoBM object for disparity calculation
    stereo = cv2.StereoBM_create(
        numDisparities=num_disparities,
        blockSize=block_size)

    # Compute the disparity map
    disparity_map = stereo.compute(left_image, right_image)

    # Normalize the disparity map for better visualization
    disparity_map_normalized = cv2.normalize(
        disparity_map,
        None,
        alpha=0,
        beta=255,
        norm_type=cv2.NORM_MINMAX,
        dtype=cv2.CV_8U)

    # Display the disparity map with updated parameters
    cv2.imshow('result', disparity_map_normalized)
```

## SIFT

```
# Step 1: Load the images
img1 = cv2.imread('<image_path>', 0)  # queryImage
img2 = cv2.imread('<image_path>', 0)  # trainImage

# Check if the images have different sizes
if img1.shape != img2.shape:
   # Resize the images to a common size
   width = min(img1.shape[1], img2.shape[1])
   height = min(img1.shape[0], img2.shape[0])
```

```
    img1 = cv2.resize(img1, (width, height))
    img2 = cv2.resize(img2, (width, height))

# Step 2: Detect and compute SIFT features
sift = cv2.SIFT_create()

kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)

# Step 3: Match the features using FLANN
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=50)

flann = cv2.FlannBasedMatcher(index_params,
    search_params)
matches = flann.knnMatch(des1, des2, k=2)

# Store all the good matches as per Lowe's ratio test.
good = []
for m, n in matches:
    if m.distance < 0.5 * n.distance:
        good.append(m)

# Step 4: Compute the disparity
# (difference in x-coordinates) for the matched points
if len(good) > 10:
    src_pts = np.float32(
        [kp1[m.queryIdx].pt
        for m in good]).reshape(-1, 1, 2)
    dst_pts = np.float32(
        [kp2[m.trainIdx].pt
        for m in good]).reshape(-1, 1, 2)

    # Calculate disparity (x-coordinates difference)
    disparity = np.abs(src_pts[:, :, 0] - dst_pts[:, :, 0])

    print('Disparity:', disparity)

    # Draw matches
    img3 = cv2.drawMatches(img1,
        kp1, img2, kp2,
        good, None,
        Flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
```

```
# Display the result
plt.imshow(img3), plt.show()
```

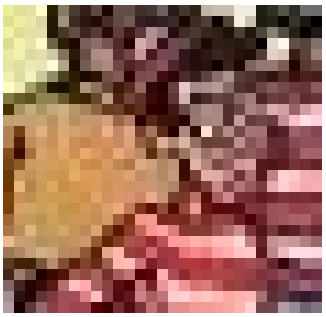
# Question 2: Where's Wally

For this question, 3 Where's Waldo images were found on the internet? Template matching was then used in order to find Waldo in the images.

## Dataset Overview

The following images were extracted from the selected Waldo maps and were used as templates for template matching.

Template from Map 1



Template from Map 2

Template from Map 3

Figure 7: Waldo templates from the chosen maps

The following three images were used in trying to perform template matching. All of the images were selected to be 1920 by 1080 pixels.

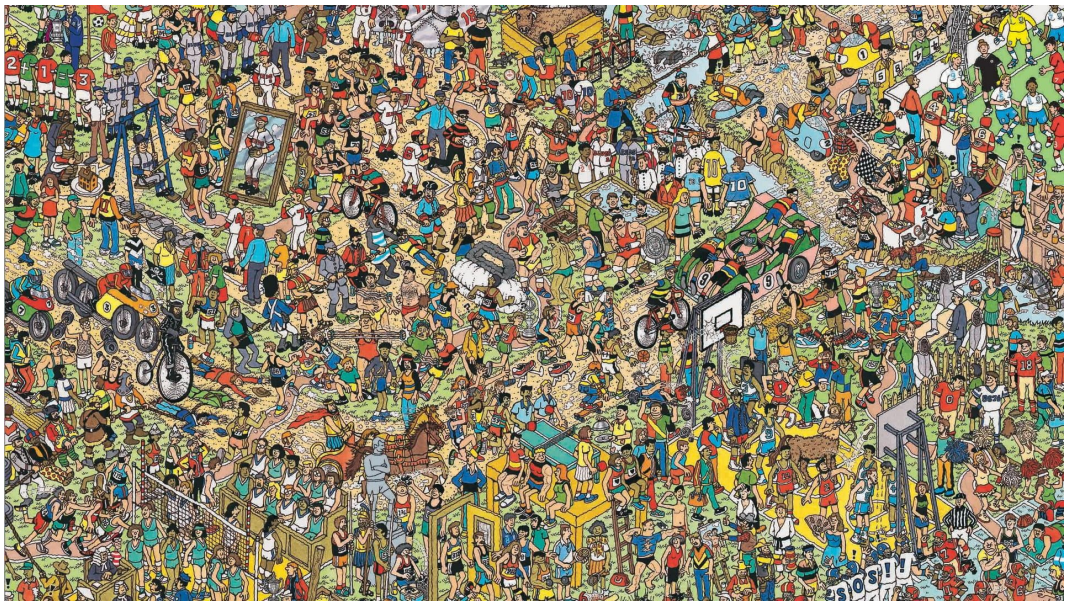


Figure 8: Map 1

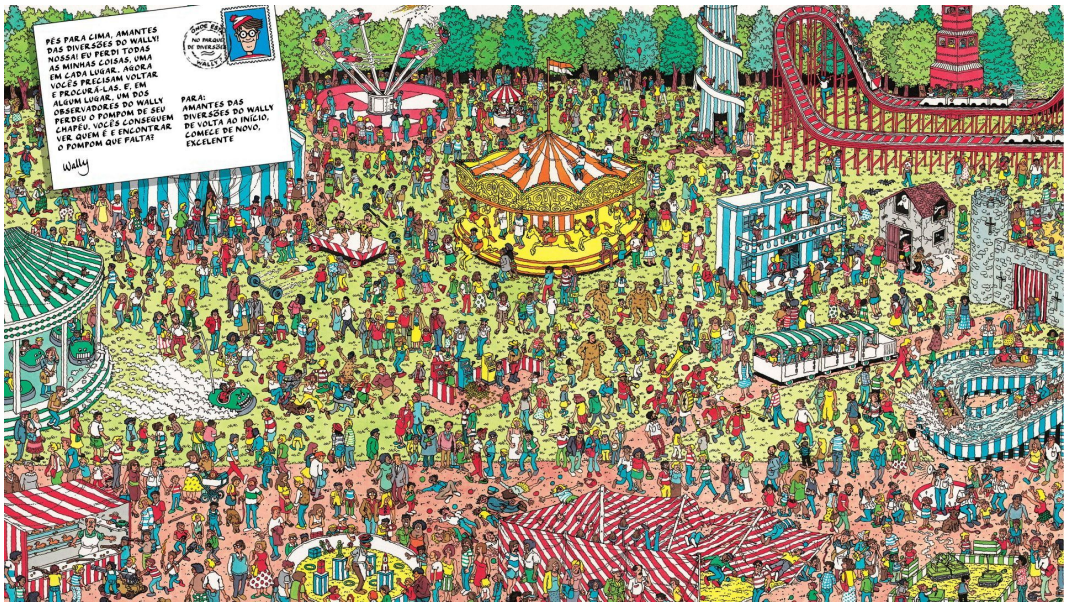



Figure 9: Map 2

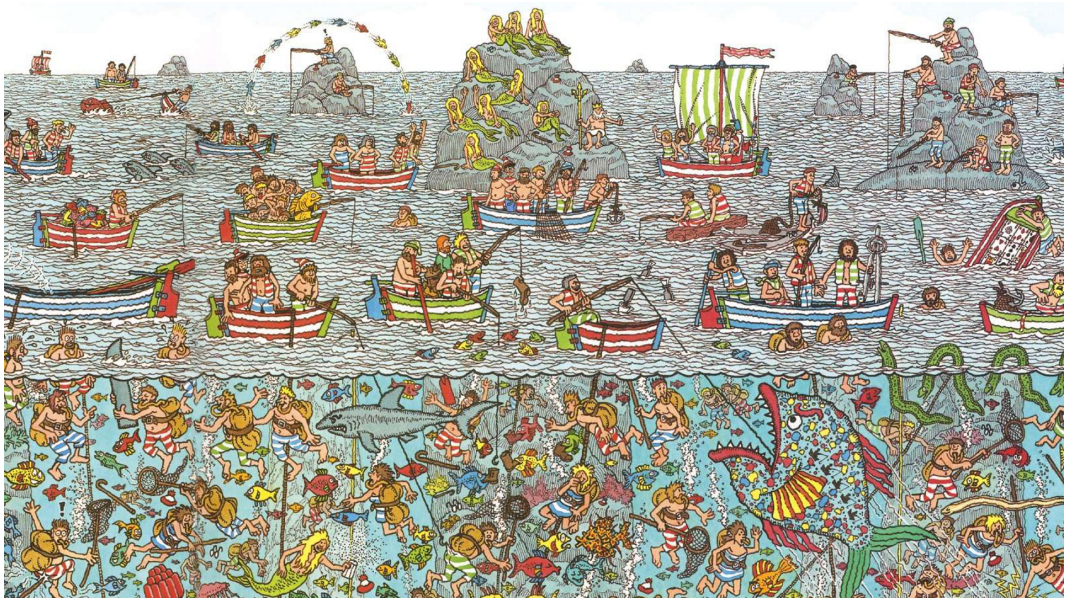


Figure 10: Map 3

## Method

Template matching works by calculating the similarity between the given template and all of the corresponding pixels in the search area. These results are then mapped to a matrix according to the resultant similarity.

This is achieved by taking a cross-product of the search space and the template. This is done in a loop until every possible search space in the search image has been computed and the similarity value stored.

## Application

When applying template matching to the selected images, we expect the template matching to work perfectly when using a template generated from the specific search image. Figure 11 below shows the calculated similarity matrix when using template 1 and map 1.

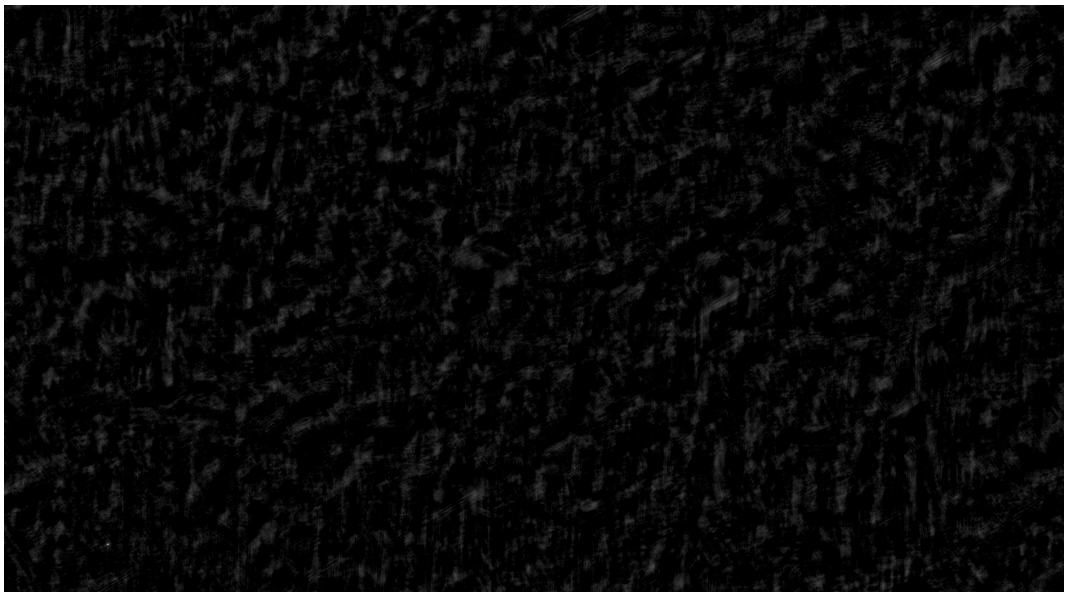


Figure 11: Map 1 & Template 1 Similarity Map

The white pixel on the left and bottom of the matrix is the spot where the template matches exactly. This can be confirmed by drawing a rectangle on the search image showing where the template was identified. This is shown in Figure 12.

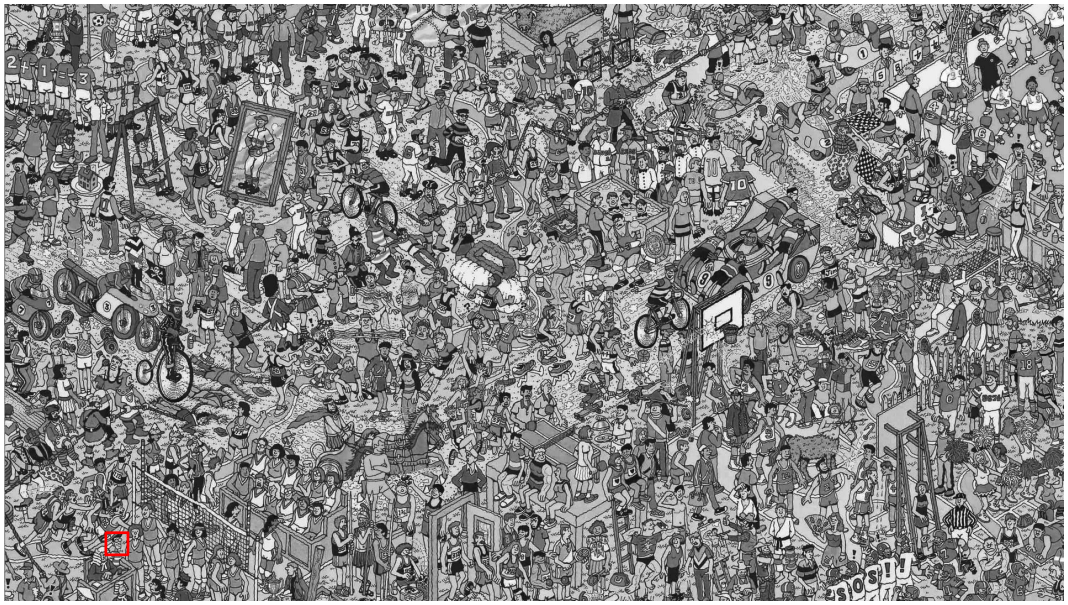


Figure 12: Map 1 & Template 1 Match

If we try and apply the same search to Map 2, we are not able to find Waldo.
In fact, the identified search area with the highest similarity score (0.39329639077186584)

is not even close to the correct position.

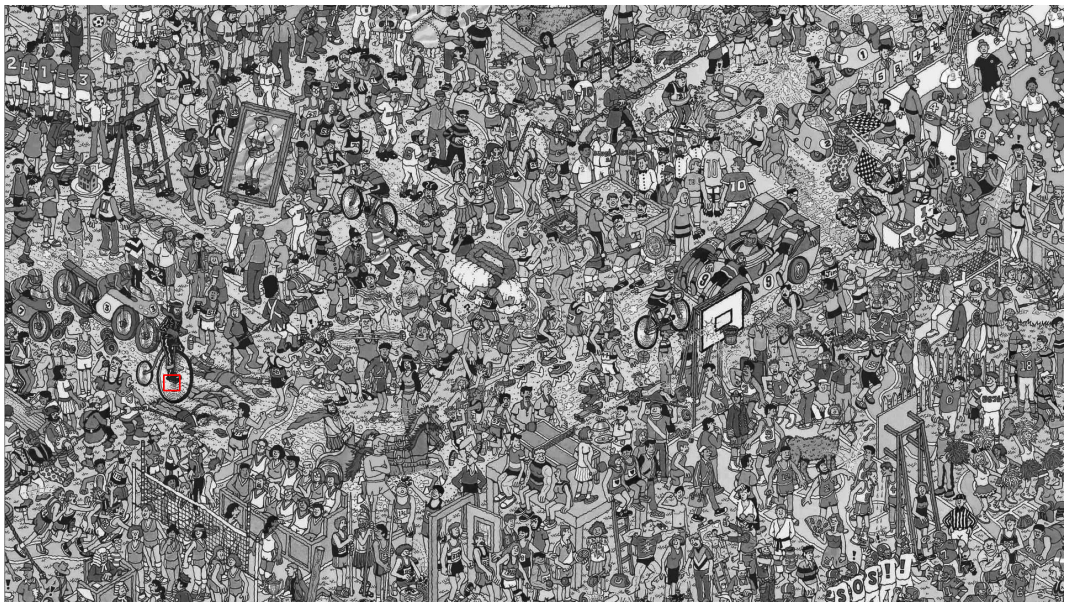
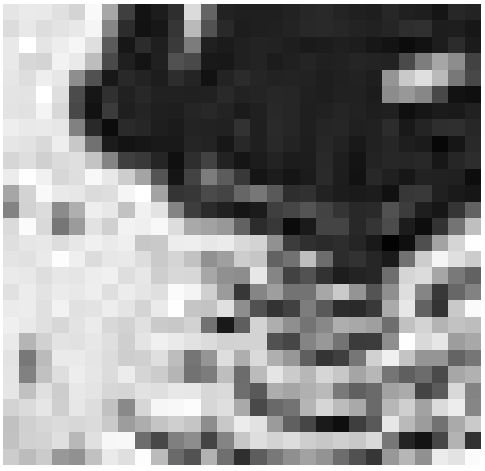


Figure 13: Map 2 & Template 1 Match

As we can see in Figure 13, the identified area is not where Waldo actually is. Figure 14 shows the search area that was identified by the template matching.

Gray-scale Template 1



Identified Area

Figure 14: Map 2 & Template 1 Identified Area

If we visually compare the two images we can see that the only real similarity is the dark area in the top red corner. This however is not relevant to Wally.

# Discussion

Template matching tries to find a very specific image in a search image by calculating the similarity between the given template and the different search areas in the search image.

The similarity score that is determined is a pretty good indicator of whether the found match is a good match or not. If the value is below 0.5, it is safe to say in most cases that the match is not very reliable.

## Template Background

One large problem with template matching is that the search also takes the background of the template into account. This means that if the template has a dark background and the search area has a light background, we will not get a very high similarity score if the background makes up a large portion of the template.

One approach to solving this would be to filter out different colour channels in order to reduce the effect of different colour backgrounds. Then the best match overall different colour channels can be taken to be the best match. This however is computationally very expensive and does not guarantee a better result.

## Template Size and Orientation

Another factor that proves to be a challenge for template matching is that the search area is the same as the template size. Thus if the template object is larger than the object in the search image, the similarity will not be detected. The same is true for the orientation of the template.

One way to solve this is to once again iterative search with the template scaled to different sizes and orientations and to select the best result. This is again very computationally expensive and does not guarantee a better result.

## Template Resolution

The final problem with template matching that has been identified is that the resolution of the template and search image also plays a role in the similarity score calculations.

This means that if the template has a much higher or lower resolution than the search object in the search image, the similarity score will not be very high since the corresponding pixel values will likely not be very similar.

## Code Implementation

The code for this question was implemented in Python using the OpenCV library.

```
# Load the main image and template image - read as bgr
image = cv2.imread(<image_path>)
template = cv2.imread(<template_path>)

# Convert the images to Greyscale
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
template = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)

# Get the width and height of template
w, h = template.shape[::-1]

# Perform template matching
res = cv2.matchTemplate(image,
                template,
                cv2.TM_CCOEFF_NORMED) # Matching method

# Find the location and match value of the identified area
~, max_val, ~, max_loc = cv2.minMaxLoc(res)

# Convert grayscale image to colour
image_color = cv2.cvtColor(image,
                cv2.COLOR_GRAY2BGR)

# Add red rectangle around identified area
cv2.rectangle(image_color,     # target image
            max_loc,          # top left corner
            (max_loc[0]+w,max_loc[1]+h), # right bottom
            (0,0,255),        # frame colour
            2)                # line thickness

# Extract the identified image area
cropped_image = image[max_loc[1]:max_loc[1]+h,
                    max_loc[0]:max_loc[0]+w]

# Show the image
cv2.imshow('Map_With_Rectangle', image_color)
cv2.imshow('Cropped_Area', cropped_image)

# Display frames until key is pressed
cv2.waitKey(0)
cv2.destroyAllWindows()
```

# Question 3: Filter Smarties

## Overview

The question requires two images of Smarties. One with good lighting and one with bad lighting. The following images were used as a starting point for the question

Good lighting

Bad Lighting

Figure 15: Stereo Image of Smarties

## Colourspaces

The question requires the selection of two colourspaces. The first colour space is one that we might expect to do well in the problem and another that we expect to perform poorly. The two colour spaces that were selected were: HSV and RGB, where HSV is expected to perform better than RGB.

HSV stands for Hue, Saturation and Value. Figure 16 below shows an overview of the HSV colourspace. (rosedeepy, 2021)

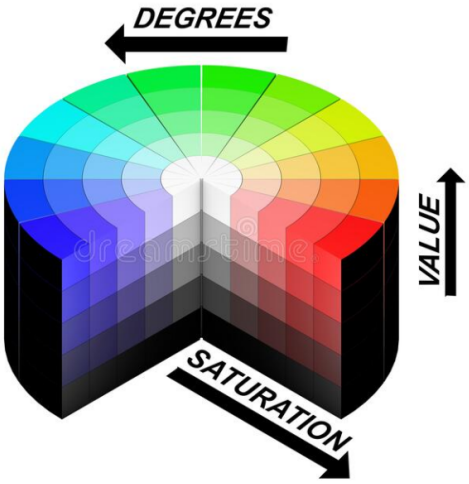



Figure 16: HSV Colour Space

This is expected to perform well since we can filter out with the Hue varying between 0 and 180 degrees and Saturation and Value between 0 and 255.

RGB is the Red Green Blue colourspace and is the most commonly used colourspace. Figure 16 below shows the typical overview of the RGB space. (rosedeepy, 2021)

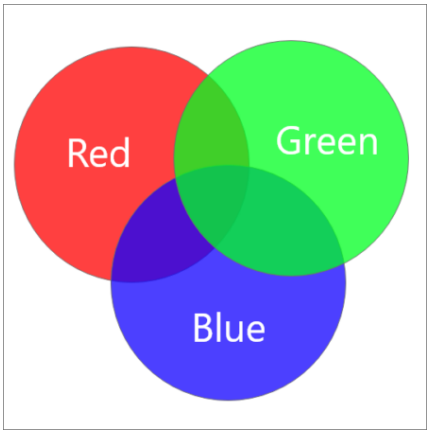



Figure 17: RGB Colour Space

This space has Red, Green and Blue values ranging between 0 and 255. This is expected to perform worse for the filtering application due to the ratio between the values being used to determine the darkness of the colour shown in the image, where HSV would experience a constant Hue with varying brightness and saturation.

The differences between the two colourspaces are shown in Table 2 below.

Table 1: Colourspace Comparison

| | RGB | | HSV | |
|---|---|---|---|---|
| Index | Channel | Range | Channel | Range |
| 1 | Blue | 0 - 255 | Hue | 0 - 179 |
| 2 | Red | 0 - 255 | Saturation | 0 - 255 |
| 3 | Green | 0 - 255 | Value | 0 - 255 |

# Implementation Strategy

The strategy for this question is to determine a mask that can be applied to the original image in order to determine the areas that should be kept and what should be filtered out.

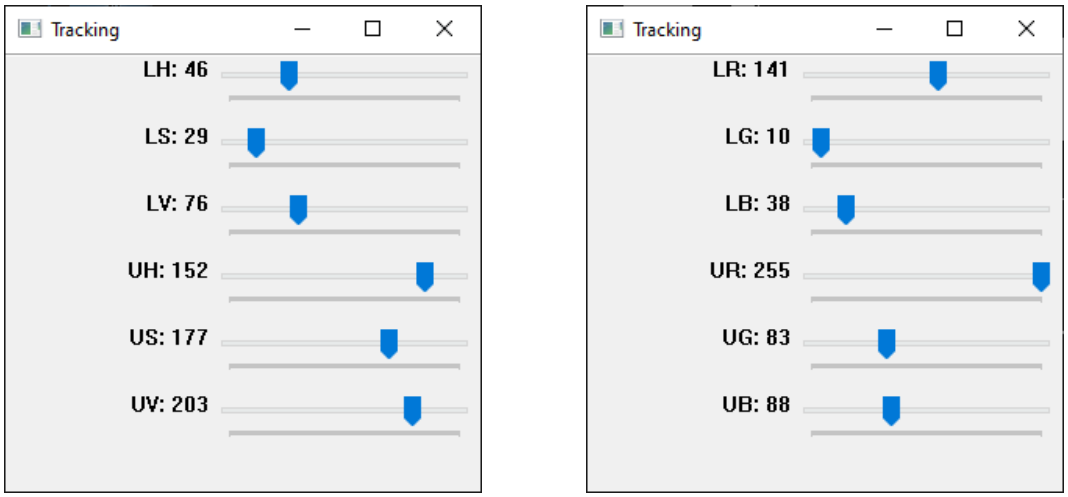
Once this mask has been determined it can be applied to the original image by performing a logical AND on all of the bytes of the image. The result is an image with only the desired pixels still visible.

When filtering for two different colours, two masks can be determined and added together before applying them to the image. This would result in both colours being displayed on the resulting image.

The initial mask will be determined by tuning the upper and lower channel values based on the bright image. Then these same filter values will be used for the non-ideal image and the results compared and analysed.

Next, a filter can be applied considering both circumstances at the same time to determine more robust filter bounds. This strategy will then also be compared and analysed.

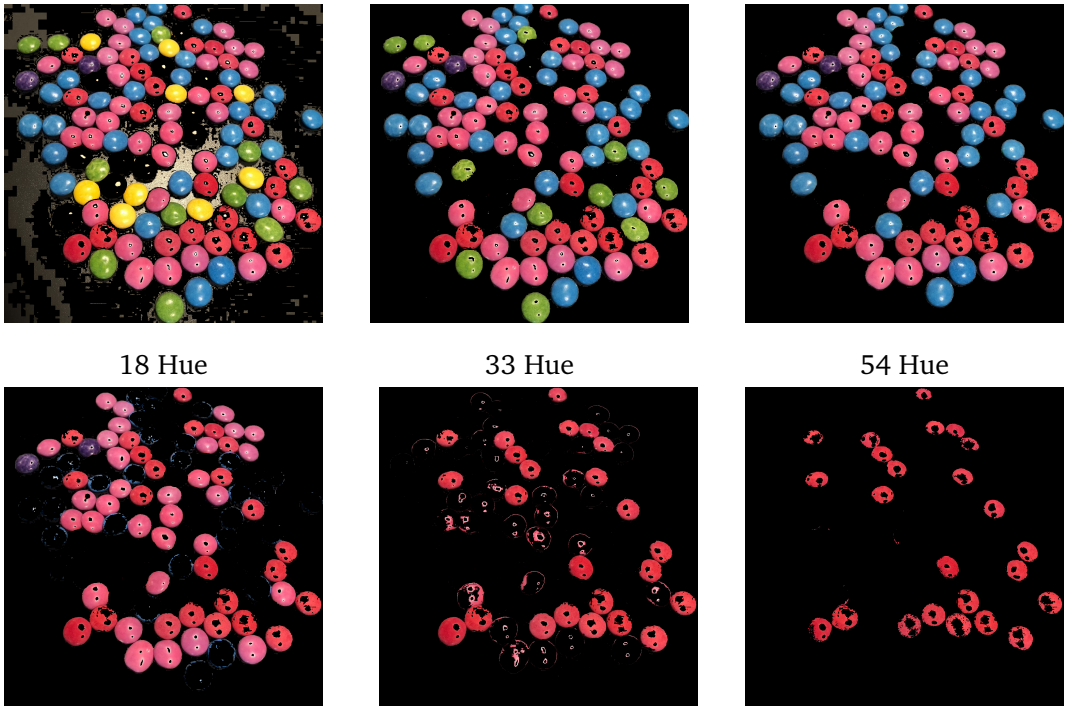
A Python script implementing sliding bars to tune the HSV and RGB upper and lower limit values was implemented. The interface can be seen in Figure 18 below.




HSV Tuner RGB Tuner

Figure 18: Tuning Script Graphical Interface

The process of tuning the values for the HSV colourspace is shown in Figure 19 below as the filter values for the red smarties.



18 Hue 33 Hue 54 Hue

105 Hue 175 Hue Final

Figure 19: Good lighting HSV Tuning

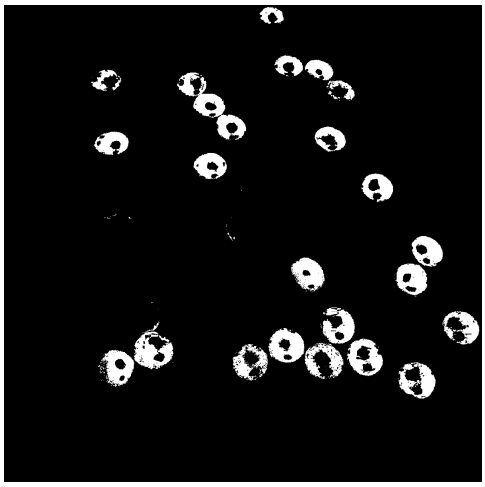
## Filter Parameters

Following this approach for both the red and brown smarties for HSV and RGB, using the image with good lighting, the following results were achieved:

Table 2: HSV Filter Values

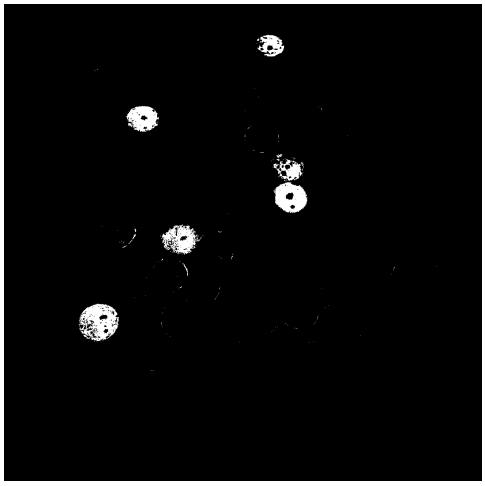
| | Red | | Brown | |
|---|---|---|---|---|
| Channel | Lower | Upper | Lower | Upper |
| H | 175 | 179 | 12 | 15 |
| S | 170 | 255 | 108 | 182 |
| V | 203 | 255 | 128 | 226 |

These values resulted in the following masks and corresponding result images:

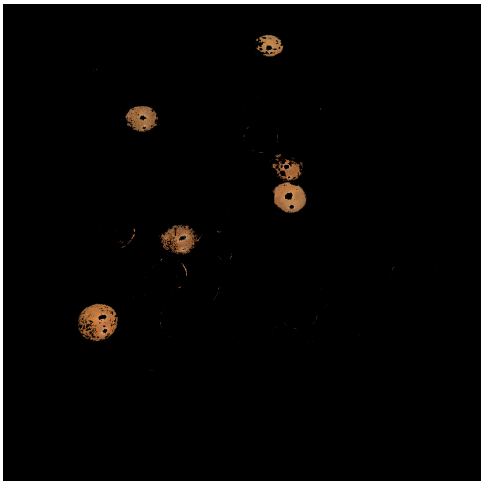


Red Smarties Mask

Red Smarties Result



Brown Smarties Mask



Brown Smarties Result

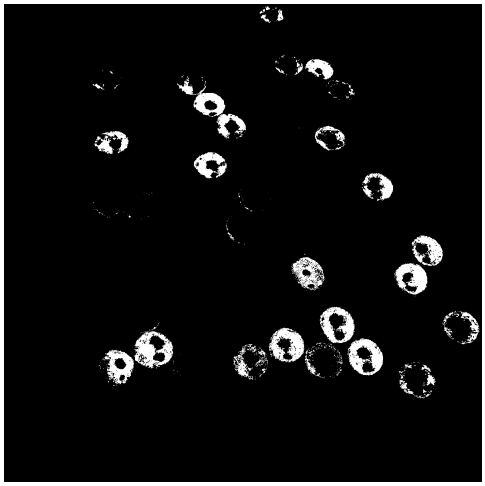
Figure 20: HSV Filter Results

Following the same steps as with the HSV images, the values for the RGB colourspace could be found. These results are shown in Table 3 below:

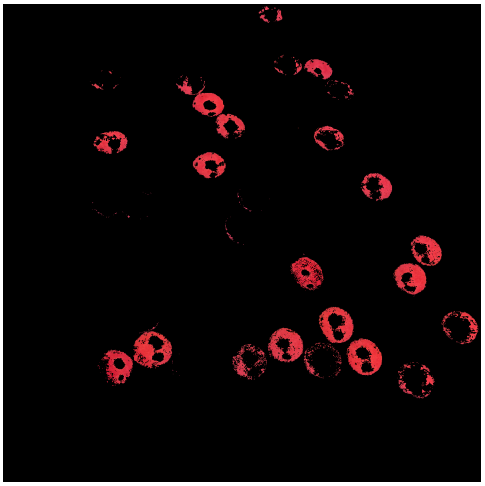
Table 3: RGB Filter Values

| | Red | | Brown | |
|---|---|---|---|---|
| Channel | Lower | Upper | Lower | Upper |
| R | 167 | 255 | 113 | 173 |
| G | 38 | 71 | 75 | 114 |
| B | 52 | 80 | 33 | 66 |

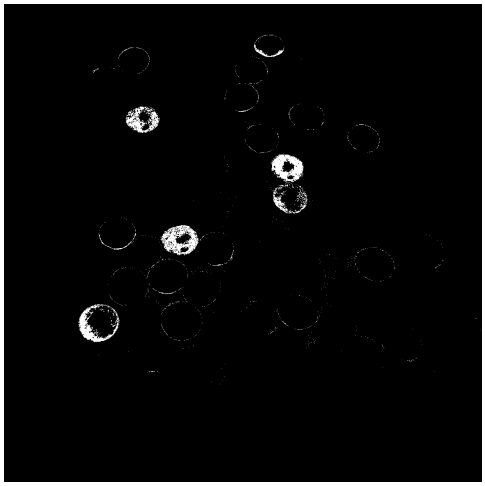
These values resulted in the following masks and corresponding result images:



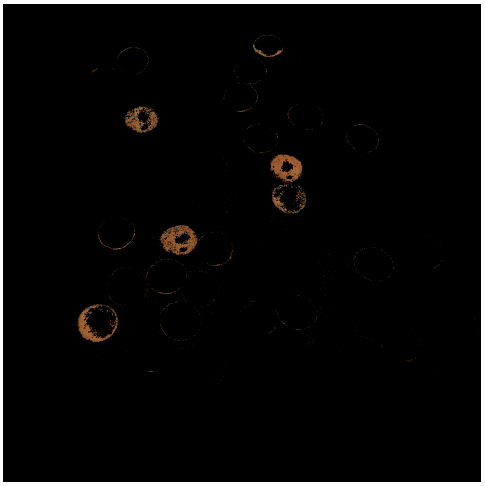
Red Smarties Mask



Red Smarties Result



Brown Smarties Mask



Brown Smarties Result

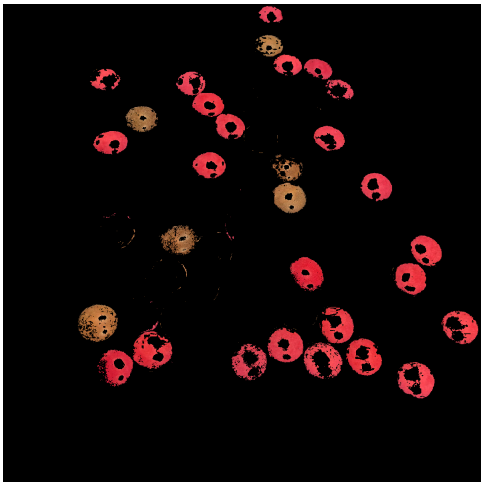
Figure 21: HSV Filter Results

Comparing the results in Figure 20 and Figure 21 above, we can see that the result using the HSV tuning is seemingly already yielding better results. The process of tuning the HSV values was also a lot simpler as it proved to be a more intuitive way of thinking about the colours seen in the image.

## Filter Implementation

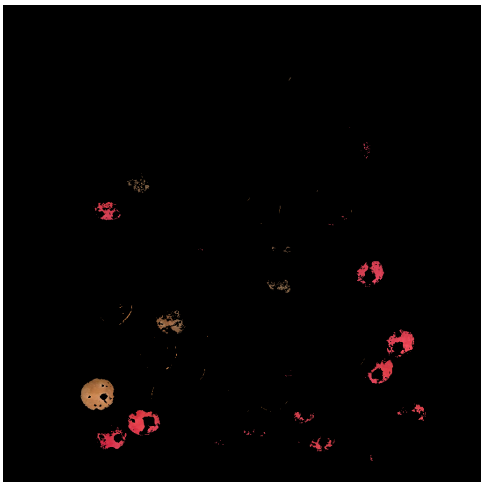
The next step would be to combine the two masks and determine the results for both the good lighting and non-ideal lighting images. Figure 22 below shows the implementation of the VHS filter on the bad and good lighting images.

Good Lighting Original



Good Lighting Result

Bad Lighting Original



Bad Lighting Result

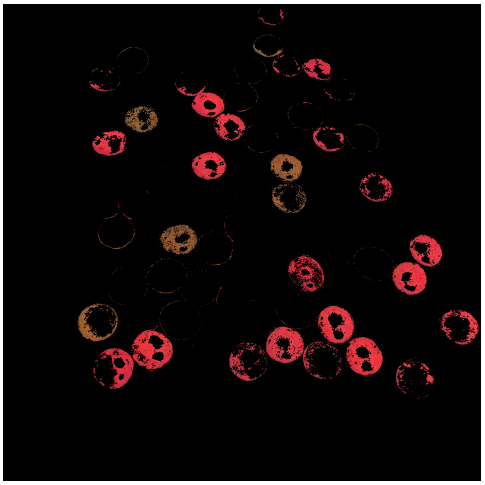
Figure 22: HSV Filter Results

As can be expected, the good lighting image shows a good result containing

only the red and brown Smarties and nothing else. The same is not true for the bad lighting image, where very few of the smarties are actually visible. The conclusion from this is that the parameter selection based on the good lighting image does not deliver a robust filter.

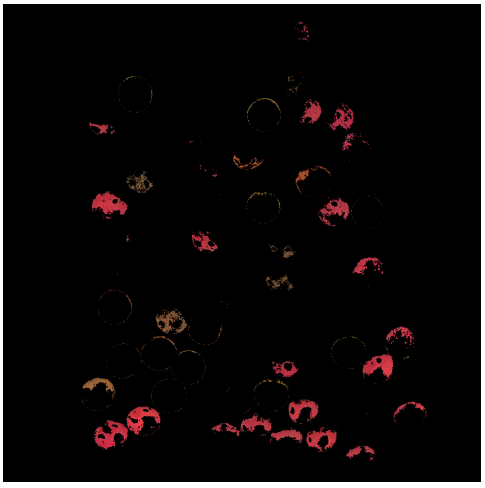
Figure 23 below shows the same results now using the RGB based filter:

Good Lighting Original



Good Lighting Result

Bad Lighting Original



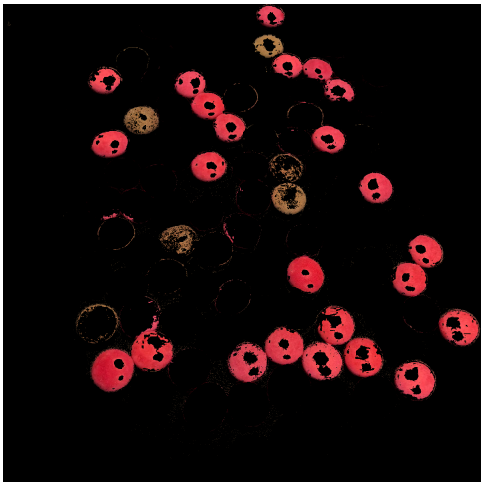
Bad Lighting Result

Figure 23: RGB Filter Results

From Figure 23 we can see that again the filter performs reasonably well for the good lighting case. In this case, however, the filter seems to perform better than the HSV filter when applied to the non-ideal lighting case. There are more red and brown Smarties visible in the resulting image.

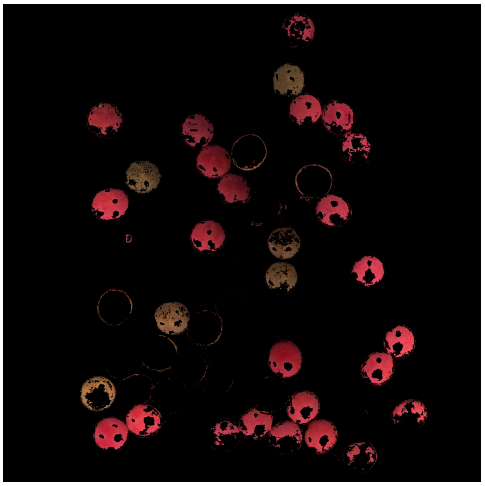
The final step would be to tune the parameters for a filter using both the good and bad lighting images.

Good Lighting Original



Good Lighting Result

Bad Lighting Original



Bad Lighting Result

Figure 24: Optimised HSV Filter Results

From Figure 24 above we can now see that with a robust selection of our H, S and V range we can achieve relatively good colour filtering in both ideal and non-ideal lighting conditions.

## Code Implementation

The code for this problem was also implemented in Python using the OpenCV library.

```
# Load the image
frame = cv2.imread('<image_path>')

# Convert the image to hsv
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# Set the Red Filter Values:
# Lower Bounds
l_v = 175
l_h = 165
l_s = 0
# Upper Bounds
u_v = 179
u_h = 255
u_s = 255

# Define the bounds for the color space filter:
l_mask_brown = np.array([l_v, l_h, l_s])
u_mask_brown = np.array([u_v, u_h, u_s])

# Set the Brown Filter Values:
# Lower Bounds
l_v = 0
l_h = 118
l_s = 49
# Upper Bounds
u_v = 17
u_h = 165
u_s = 184

# Define the bounds for the color space filter:
l_mask_red = np.array([l_v, l_h, l_s])
u_mask_red = np.array([u_v, u_h, u_s])

# Define the masks:
mask_red = cv2.inRange(hsv, l_mask_red, u_mask_red)
mask_brown = cv2.inRange(hsv, l_mask_brown, u_mask_brown)
mask = cv2.bitwise_or(mask_brown, mask_red)
```

```
# Get the result after the mask is applied:
result = cv2.bitwise_and(frame, frame, mask=mask)

# Show the results of the masking:
cv2.imshow("frame", frame)
cv2.imshow("mask_1", mask_red)
cv2.imshow("mask_2", mask_brown)
cv2.imshow("mask", mask)
cv2.imshow("result", result)

# Display frames until key is pressed
cv2.waitKey(0)
cv2.destroyAllWindows()
```

# Conclusions

## Stereo Vision

Block Matching and Semi Global Block Matching were used to create disparity maps between two sets of stereo images. Next, SIFT feature matching was applied to match features between these two sets of images.

It was determined that for the given application, SIFT performs better due to the non-idealised nature of the images that were taken. In actual applications, these two might be used in the same pipeline. Better and specialised cameras will also lead to better disparity map results.

## Where's Wally

Template matching is a very powerful technique that is used in many applications from fault detection to object detection. The application to the problem of finding Wally proved to not be very practical as the resolution, size and orientation across many different Wally maps differ significantly.

Factors such as the background of the template also differ substantially between different search maps, making the similarity scores unreliable. Different techniques such as feature detection of machine learning models would possibly provide better results, as a higher complexity and computational cost.

## Smarties Filter

The implementation of a mask based on a pre-specified colour channel range proves to be relatively good at filtering out certain colours of Smarties given good and bad lighting images. When extending this application to the real world it is important to select the colour filtering ranges based on a variety of different lighting conditions.

All of the code that was implemented during this project can be found at `https://github.com/TheMechatronic/Computer_Vision`

# List of references

openimaj (2020). Sift and feature matching.
Available at: http://openimaj.org/tutorial/sift-and-feature-matching.html

rosedeepy (2021). What are color spaces and applying grayscale? | pyimagedata.
Available at: https://www.pyimagedata.com/what-are-color-spaces-and-applying-grayscale/

Singh, A. (2023). Sift algorithm | how to use sift for image matching in python.
Available at: https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/