# Unleash Your Genius: Decode the Hidden Image and Outsmart the Kryptonians!

Owais Makroo

***Abstract*—In this paper, we present an algorithmic solution to the Kryptonian challenge of deciphering a hidden image using an intricate relationship with an input image. The challenge requires the development of an efficient algorithm capable of unraveling the hidden image by analyzing the generated feedback image. We propose a solution implemented using the Robot Operating System (ROS) framework, utilizing an 8-bit, 3-channel input image. Our algorithm follows a set of rules based on the relationships between the input and hidden images' pixel values to generate the corresponding output image. By applying these rules iteratively, we successfully reveal the hidden image within the given system. This paper discusses the algorithm's design, implementation in ROS, and its effectiveness in meeting the challenge.**

## I. INTRODUCTION

The arrival of extraterrestrial visitors from the advanced planet of Krypton presents humanity with an extraordinary challenge: deciphering a hidden image concealed within a system governed by complex relationships with an input image. This paper addresses the challenge by proposing an efficient algorithmic solution implemented using the Robot Operating System (ROS).

## II. PROBLEM STATEMENT

The Kryptonian challenge presents humanity with the task of deciphering a hidden image concealed within a system that involves intricate relationships with an input image. The system operates on 8-bit, 3-channel images, where each pixel holds values for Red (R), Blue (B), and Green (G).

The problem can be summarized as follows: Given an input image and a set of relationships between the pixel values of the hidden image and the input image, the objective is to develop an efficient algorithm that can generate an output image based on these relationships. The output image should reveal the hidden image by following the specified rules for each channel (R, G, B).

To clarify the relationships between the input and hidden images' pixel values, the following rules apply:

1) **Red Channel (R)**
   - If the Red value of the input image (R_input) is less than the Red value of the hidden image (R_hidden), the corresponding Green value in the output image (G_output) should be set to 0.
   - If R_input is equal to R_hidden, G_output should be set to 127.
   - If R_input is greater than R_hidden, G_output should be set to 255.
2) **Green Channel (G)**
   - If the Green value of the input image (G_input) is less than the Green value of the hidden image (G_hidden), the corresponding Blue value in the output image (B_output) should be set to 0.
   - If G_input is equal to G_hidden, B_output should be set to 127.
   - If G_input is greater than G_hidden, B_output should be set to 255.
3) **Blue Channel (B)**
   - If the Blue value of the input image (B_input) is less than the Blue value of the hidden image (B_hidden), the corresponding Red value in the output image (R_output) should be set to 0.
   - If B_input is equal to B_hidden, R_output should be set to 127.
   - If B_input is greater than B_hidden, R_output should be set to 255.

The algorithm's objective is to iteratively apply these rules to each pixel in the input image, generating the corresponding output image that reveals the hidden image within the given system.

By cracking this extraordinary challenge and successfully implementing the algorithm, we demonstrate humanity's intellectual mettle and ability to analyze complex relationships within images. The fate of the world rests on our capability to unravel the hidden image, unless Superman intervenes to save the day.

## III. INITIAL ATTEMPTS

In our initial attempts to unravel the hidden image within the given system, we devised a straightforward algorithm that involved iteratively adjusting pixel values based on the feedback received from the rules. However, this approach proved to be highly inefficient and time-consuming.

Our initial algorithm consisted of initializing the input image as an 8-bit, 3-channel image with all pixel values set to 127. We then passed this image into a compare function that compared the pixel values of the input and hidden images according to the specified relationships. Based on the output feedback for each pixel, we incrementally increased or decreased the pixel value by 1, attempting to converge on the required image.

```
final_r = final_r*(imgMsg_res[:,:,1] == 127)
final_r = (final_r-1)*(imgMsg_res[:,:,1] == 255)
final_r = (final_r+1)*(imgMsg_res[:,:,1] == 0)
```

While this initial attempt was conceptually straightforward, it proved to be impractical for larger images due to the maximum of 128 iterations required. The computational cost per iteration was considerable, resulting in excessive execution time. It became evident that a more efficient and optimized approach was necessary to meet the challenge requirements.

In the subsequent sections of this paper, we present our revised algorithm, developed based on a more optimized approach. We outline the improved methodology, discuss the implementation details within the ROS framework, and provide experimental results that demonstrate the effectiveness of our optimized solution in efficiently unraveling the hidden image within the Kryptonian challenge.

## IV. Final Approach

To overcome the inefficiencies of the initial attempts, we developed a more optimized solution based on the binary search algorithm. This approach significantly reduced the maximum number of iterations required to unveil the hidden image within the given system. The following code snippet outlines the final approach:

```
global imgMsg_res
final_r = 127 * np.ones(
    (img_size[0], img_size[1])
    )
final_g = 127 * np.ones(
    (img_size[0], img_size[1])
    )
final_b = 127 * np.ones(
    (img_size[0], img_size[1])
    )

rospy.init_node('player_node')
rate = rospy.Rate(10)
pub1 = rospy.Publisher(
    'guess',
    Image,
    queue_size=10
    )
rospy.Subscriber('result', Image, guessCallback)

change = 128
while not rospy.is_shutdown():
    guess = cv2.merge([final_b,final_g,final_r])
    msg = bridge.cv2_to_imgmsg(guess)
    pub1.publish(msg)

    rate.sleep()
    if imgMsg_res is not None:
        if (np.sum(imgMsg_res!=127) == 0):
            rospy.signal_shutdown(
                "Task Complete"
                )
            exit()
        final_r = final_r * (imgMsg_res[:,:,1]
            == 127)
        final_r = (final_r - change) * (
            imgMsg_res[:,:,1] == 255)
        final_r =  (final_r + change) * (
            imgMsg_res[:,:,1] == 0)
        final_g = final_g * (imgMsg_res[:,:,0]
            == 127)
        final_g = (final_g - change) * (
            imgMsg_res[:,:,0] == 255)
        final_g = (final_g + change) * (
            imgMsg_res[:,:,0] == 0)
        final_b = final_b * (imgMsg_res[:,:,2]
            == 127)
        final_b = (final_b - change) * (
            imgMsg_res[:,:,2] == 255)
        final_b = (final_b + change) * (
            imgMsg_res[:,:,2] == 0)
        imgMsg_res = None
        change = change / 2
```

In this final approach, we adopted a binary search algorithm to efficiently update the pixel values of the output image channels (red, green, and blue) based on the comparison results obtained from the hidden image (*imgMsg_res*). Here's an overview of the key steps:

1) **Initialization**
   We begin by initializing the *final_r, final_g,* and *final_b* to 1 channel 8 bit images with all pixel values equal to 127 and the change variable to 128, which represents the maximum change value in each iteration. This value allows us to halve the range of possible values at each iteration.
2) **Iterative Process**
   The algorithm iterates until a solution is found or the ROS shutdown signal is received. Within each iteration, the algorithm performs the following steps:
   - Generate Guess
     We generate a guess image by merging the current red, green, and blue channels (*final_r, final_g,* and *final_b*, respectively) into a single image.
   - Publish Guess
     The guess image is published as a ROS image message to be processed.
   - Comparison and Update
     If the *imgMsg_res* is not *None*, indicating that a comparison result is available, we proceed to update the pixel values of the output image channels based on the comparison results. The comparison results are used to selectively apply the rules provided in the problem statement to update the pixel values accordingly.
   - Halving the Change
     After updating the pixel values, the *change* value is halved to reduce the range of possible values for the next iteration. This reduction in range allows for faster convergence towards the hidden image.
3) **Task Completion**
   If the sum of elements in *imgMsg_res* that are not equal to 127 is zero, it indicates that all pixel values have been successfully matched to reveal the hidden image. In this case, the algorithm sends a shutdown signal to ROS and exits.

By employing the binary search algorithm and the optimized update process described above, the final approach significantly reduces the maximum number of iterations required to unveil the hidden image. This approach efficiently converges

towards the solution, showcasing humanity's intellectual mettle in meeting the Kryptonian challenge.

Here is a link to the complete code.

## CONCLUSION

Our proposed solution successfully addresses the Kryptonian challenge of unraveling a hidden image within a complex system. By implementing an algorithm using ROS and following the defined rules, we effectively generate the corresponding output image based on the relationships between the input and hidden images' pixel values. Our solution demonstrates the intellectual mettle of humanity and contributes to our understanding of image analysis and algorithm design in the context of complex image relationships.

## REFERENCES


[1] Open Robotics, "ROS Noetic Ninjemys." https://wiki.ros.org/noetic/, 2020.

[2] Arindam, "DEBUGPOINT.COM", Link, 2022