Decoding the hidden Image

U Skanda Aithal *

Abstract— **Should be half column long. Should be clear enough to explain your whole documentation. Similar to a TL;DR**

Write in brief what you have done and give a small outline about the fields of application. Eg. for object detection , write about the fields where this can be scaled , for solidworks model write about the place where it will be used in ARK or in general it's uses.

I. INTRODUCTION

The task is to develop an efficient algorithm using the provided ROS template to unravel the hidden image by analyzing the feedback generated. The challenge requires an 8-bit, 3-channel input image, and the relationship between the input and output images is given. The problem is to successfully decipher this relationship and reveal the hidden image. I have used an algorithm which works like binary search in every pixel. Initial implementation was to use for loop and traverse through every element. But that was not very efficient and then I used numpy functions to carry on array operations.

II. PROBLEM STATEMENT

The goal is to develop an efficient algorithm that can unravel the hidden image by utilizing appropriate input images and analyzing the generated feedback.

The given problem statement presents an intriguing challenge involving a system that conceals an image within it. The system takes an 8-bit, 3-channel image as input and generates a feedback image based on a complex relationship between the hidden and input images.

The task requires implementing the solution using the Robot Operating System (ROS) and a provided template. The challenge emphasizes the importance of developing an algorithm that can decode the hidden image by understanding the relationship between the input and hidden images.

In this system, each pixel in the images has three color values: red (R), blue (B), and green (G). The relationship between the input and output images is defined as follows:

If the red value of the input image is less than the red value of the hidden image, the green value of the output image is set to 0. If the red values of the input and hidden images are equal, the green value of the output image is set to 127. If the red value of the input image is greater than the red value of the hidden image, the green value of the output image is set to 255. Similarly, the relationship between the green and blue values of the input and hidden images determines the blue and red values of the output image, respectively.

To crack this challenge, an algorithm needs to be devised that can efficiently analyze the input images, compare their color values with the hidden image, and generate the corresponding output image based on the predefined relationships.

We can utilize the provided template and implement the necessary algorithms and logic to unravel the hidden image. By leveraging the given relationships between color values and analyzing the input and hidden images effectively, the algorithm should generate the corresponding output image that reveals the concealed information.

In conclusion, the problem statementis to develop an efficient algorithm to decode a hidden image concealed within a system. By utilizing appropriate input images and analyzing the generated feedback, the task is to generate the output image that unveils the concealed information. The solution is to be implemented using the ROS framework.

III. RELATED WORK

I have implemented an algorithm based on binary search. One can also use an approach based on linear search and any other appropriate search algorithms. But the approach using linear search will be less efficient compared to the approach using binary search.

IV. INITIAL ATTEMPTS

Initially I tried the same algorithm by traversing through each pixel and carrying out the operation. But it turned out to be of very low efficiency and hence I changed the method of implementation.

V. FINAL APPROACH

The implementation utilizes the ROS (Robot Operating System) framework to implement an image decoding strategy.

There is file provided checker.py. The purpose of this program is to implement a checker node in the ROS (Robot Operating System) framework. The code subscribes to the "/guess" topic to receive Image messages containing a guessed image. It generates a random hidden image and compares it with the guessed image using a specific comparison function. The result of the comparison is then published on the "/result" topic.The main functionality is encapsulated within the "check" function. This function initializes the ROS node, sets up a subscriber to receive the guessed image messages, and initializes the hidden image. The hidden image is generated using the "generateRandomImage" function, which creates a random image of the specified height and width.

*Venkatesh

The ”compareImage” function compares the corresponding color channels of the guessed image and the hidden image, calculates the result based on the given comparison formula, and returns the resulting image. The resulting image is then converted back to an Image message using the CvBridge and published on the ”/result” topic. If the resulting image matches a predefined condition (all pixel values are 127), indicating that the guessed image matches the hidden image, the code prints a success message and breaks out of the loop.

The program written in player.py subscribes to the ”/result” topic to receive an Image message, decodes the hidden image within it, and publishes the decoded image on the ”/guess” topic.

The script begins by initializing a ROS node with the name ”player node”. It also sets up a publisher object to publish the decoded image, and creates an instance of the CvBridge class, which provides a bridge between ROS Image messages and OpenCV images.

The main strategy function is defined, which represents the decoding algorithm. Within this function, there is a while loop that continuously processes the received image messages and performs the decoding process. The decoding process involves extracting color channel values (red, green, and blue) from the received image and updating the guess image accordingly.

The algorithm initializes an empty guess image with dimensions 512x512 pixels and sets all pixel values to 127 (a mid-gray color). It also creates a track array to keep track of the end values for each pixel. Then the decoding process begins by itering through the received message and finding the pixels with values 0 or 255 in each layer. To find these pixels I have used a numpy funcion where() to make the process efficient. And then according to the condition if a pixel value is lesser than the actual hidden value the I find the midpoint between the current pixel value and the corresponding upper end and replace the pixel value. If pixel value is greater, then the same process is carried out with the lower end. This follows the principle of binary search.

After updating the guess image, it converts the image to an Image message using the CvBridge and publishes it on the ”/guess” topic. The algorithm then waits for the next image message by looping until a new image message is received. It also includes a slight delay of 0.2 seconds using the rospy.sleep() function to control the frequency of processing and publishing the decoded image.

The code also defines a callback function named guessCallback, which is called whenever a new image message is received on the ”/result” topic. This function converts the received Image message to an OpenCV image format using the CvBridge and stores it in the global variable imgMsg.

Finally, in the main block of the code, the strategy function is called within a try-except block to handle any potential ROS interruptions. This ensures that the decoding process can be gracefully terminated if needed.

VI. Results and Observation

As mentioned earlier, when the comparison was carried out by traversing through each and every pixel it was very inefficient. So it was not guaranteed to provide the solution at all. Maybe there was some upper limit to the number of iterations. Then I used numpy function where() to carry out this process which is more efficient and hence works faster. The approach chosen is based on binary search. One can also implement an algorithm based on linear search. But it will be highly inefficient and slow. So I have chosen binary over linear search.

VII. Future Work

We can further develop this to make it more efficient by using other numpy functions for carrying out array operations if applicable. We can try to avoid loops as much as possible as they affect the efficiency. We can also check for alternative approaches that provide better effeciency.

CONCLUSION

This was a good introduction to ROS. Since this was new it was really challenging. The major problem was to get used to working with ROS. Then even after the implementation of the algorithm correctly, it did not provide answer because of lower efficiency which was later overcome by avoiding looping.

References

[1] http://wiki.ros.org/
[2] https://www.youtube.com/@emilvidmark