DOCUMENTATION FOR ARK PERCEPTION TEAM TASK

TASK-4 Unleash Your Genius: Decode the Hidden Image and Outsmart the Kryptonians!

I.INTRODUCTION

This task focuses on solving a challenging puzzle presented by extraterrestrial visitors from the planet Krypton. The visitors have shared a mysterious system that conceals an image within it. The objective is to develop an efficient algorithm that can decipher the hidden image by analyzing the relationship between the input and output images. By successfully solving this challenge, we can demonstrate our intellectual capabilities and unravel the hidden image.

II. PROBLEM STATEMENT

The problem at hand involves decoding an 8-bit, 3-channel image by understanding the intricate relationship between the hidden and input images. The relationship between the input and output images is defined as follows:

$$R_{input} < R_{hidden} \implies G_{output} = 0$$
$$R_{input} = R_{hidden} \implies G_{output} = 127$$
$$R_{input} > R_{hidden} \implies G_{output} = 255$$

$$G_{input} < G_{hidden} \implies B_{output} = 0$$
$$G_{input} = G_{hidden} \implies B_{output} = 127$$
$$G_{input} > G_{hidden} \implies B_{output} = 255$$

$$B_{input} < B_{hidden} \implies R_{output} = 0$$
$$B_{input} = B_{hidden} \implies R_{output} = 127$$
$$B_{input} > B_{hidden} \implies R_{output} = 255$$

Our task is to implement the solution using the provided ROS template. The template code initializes the necessary variables and provides functions to subscribe to input and result images, apply the decoding algorithm, and publish the generated output image.

III. INITIAL ATTEMPTS

The initial attempt was to start from an image(out) where all the pixels are 127. Then using the information in the image obtained as output from checker node, we need to change the image(out) accordingly.

IV. FINAL APPROACH

The final approach was to convert the result image (obtained from checker node) and the out image into separate color channels: Blue (B), Green (G), and Red (R) and then apply the decoding algorithm to each color channel as follows:

Normalize the channels by dividing them by -127 and adding 1. If input>hidden, output= 255. Here we have to reduce the pixel of input image, hence -1 should be added to input (out image). If input=hidden, output=127. In this case 0 should be added to input image. And if input<hidden, 1 should be added. On dividing the output image by -127 and adding 1 we get the respective numbers 1,0, -1 and we can directly add that matrix to the input image.

The explanation of the player.py code is as follows:

1. The code first retrieves the image height and width parameters from the ROS parameter server and stores them in height and width variables. The img_size variable is assigned a tuple representing the size of the image with three color channels.
2. **CvBridge** class is defined, which provides functions to convert between ROS Image messages and OpenCV images.
3. Then a publisher is created that publishes ROS Image messages on the /guess topic with a queue size of 10.
4. This function **strategy()** represents the main logic of the player node. It initializes variables and performs the image processing strategy. The result variable is used to store the received image message, out is the current output image, and counter keeps track of time. The out image is initialized as a square image with dimensions (height, height, 3) filled with 127 (gray) using np.full(). The node is initialized with the name 'player_node' using rospy.init_node(). It subscribes to the /result topic to receive image messages using rospy.Subscriber(). A sleep of 1 second is used to allow time for the subscriber to initialize. The while loop runs until the node is shutdown.
5. In the strategy function the following things happen:
    - The RGB channels of the result image are extracted into B, G, and R variables.
    - The RGB channels of out are also extracted into B_out, G_out, and R_out variables.
    - The data type of R, G, B, R_out, G_out, and B_out variables is converted to int.
    - The RGB values of R, G, and B are modified by dividing them by 127, negating the values, and adding 1. This operation maps the range [0, 255] to [-1, 1] and inverts the values.
    - The modified RGB values are then added to R_out, G_out, and B_out variables.
    - The data type of R_out, G_out, and B_out variables is converted back to uint8. The modified R_out, G_out, and B_out channels are merged back into a single image out using cv2.merge().

- The modified-out image is converted to a bgr8 format Image message using bridge.cv2_to_imgmsg().
- The out image is published to the topic /guess using pub1.publish().
- The counter is incremented, and a message is printed indicating the waiting time in seconds. A delay of 1 second is introduced using rospy.sleep().

V. RESULTS AND OBSERVATION

Upon executing the implemented decoding algorithm, the program iteratively decodes the hidden image by analysing the input and result images according to the predefined relationships. The algorithm utilizes the ROS framework to communicate with the system and publish the generated output image. The decoding process continues until the hidden image is successfully decoded, as indicated by all color channels matching the value of 127.

VI. FUTURE WORK

The program takes a lot of time to find the final images. A more effective decoding algorithm could be used so as to reach the final image in less time and with a smaller number of iterations.

VII. CONCLUSION

The Kryptonian Image Decoding task presents an intriguing challenge to unravel a hidden image based on the relationship between input and output images. By implementing an efficient decoding algorithm, we could successfully reveal the hidden image. The provided Python template, utilizing the ROS framework, serves as a foundation for solving this puzzle.

VIII. REFERENCES

- https://wiki.ros.org/ROS/Tutorials