Unleash Your Genius: Decode the Hidden Image and Outsmart the Kryptonions!

Jyotishko Banerjee

***Abstract*—**

We have always wondered what is beyond Earth. Is there any life far away from us? What if they are more advanced than us? What if they seek our destruction?

Well, the project was brought to us involving a very intriguing storyline involving the Kryptonites from the timeline of Superman or The Man of Steel.

When the Kryptonians have decided to attack Earth, they have been impressed by the developments that Earthlings have made with technology and have offered us a chance to save our Earth if we can solve their challenge.

They have provided a special system called ROS (Robot Operating System), which contains an hidden image and our task is to unravel the hidden image by performing constant operations that they have set.

Stories apart, on a serious note, this project basically involves developing an efficient algorithm in ROS to reveal a hidden image after some operations are performed between the input image given by us and the hidden image and the result is given back to us and we need to operate further. Overall, this project has taught me a lot about ROS system, scripts and ROS Topics. And definitely perhaps one of most challenging tasks till now due to its interface, using not so previously used Terminal to get our work done and a completely new application of ROS was definitely challenging. But none the less I worked on this project day and night learning ROS and finally I am able to present this project as explained in the following sections.

I. INTRODUCTION

The Robot Operating System also known as ROS is a set of tools and libraries which is used to build robot applications easily. ROS is a free and open-source software that defines the essential components, interfaces and tools to make advanced robots.

The functions of the robots are usually made using scripts (made using programming languages like Python) and connecting them through ROS Topics and Messages. ROS Topics are ways to for scripts to receive and send data and operate and receive feedback accordingly.

The Messages can easily be recorded using ROS Bag Files or logs for easy training and testing for robots. ROS also provides options to work with simulated robots to avoid usage of robots and wastage in real life unnecessarily and make the utilisation more effective.

ROS supports hardware interfaces for many common robot components like cameras, LIDARs, and controllers.

ROS also supports micro-ROS, a variant of ROS works in embedded micro-controllers running in real-time Operating Systems (OS). ROS is used in variety of place starting from big corporate houses to academia and industry to personal use due to its user-specific need fulfilment abilities and easy interface.

II. PROBLEM STATEMENT

Skipping the Storyline, lets come straight to the problem. We have been given a script checker.py that takes an input image from ROS Topic guess and performs an operation of it with the hidden image and publishes it to result ROS Topic. Explaining more about checker.py,

- Necessary libraries like rospy, sensor msgs, cv bridge, cv2, numpy have been imported
- Next, the height and width of the to be generated hidden image has been set through rospy.get_param from the launch file
- Then, Publisher is declared to publish Image type data to ROS Topic result
- In the next function generateRandomImage(), randimage library functions have been used to generate a random hidden image
- Here, it is to be mentioned that rqndimage is an external library that needed to be downloaded from GitHub and generation of image of standard size takes about 30 secs
- Next function compareImage() compares the input and hidden image and returns the result after performing the "Operation". The image uses BGR scheme and operates on each channel individually on a set of conditions and accordingly sets the value of 'result'. The 'Operation' is as follows:

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

$$R_{input} = R_{hidden} \implies G_{output} = 127$$

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

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

$$G_{input} = G_{hidden} \implies B_{output} = 127$$

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

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

$$B_{input} = B_{hidden} \implies R_{output} = 127$$

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

• Next, we define the check() function. We initial the node 'checker_node', set imgMsg as None and Subscribe to "/guess", then generate the hidden_image by calling function generateRandomImage()

• We, put the following part in a while loop

• We keep the program trapped in a loop until some value comes from the "/guess" and then it proceeds further, this is done in many parts of the program to avoid getting error as in a continuous publishing and subscribing, a slight delay in timing might generate errors that might prevent the program from moving forward

• We then call compareImage on the 2 images and put it in result and converting it we publish it to "/result"

• Then, we define the "Successfully Identified" condition that if all elements of result are 127, i.e. hidden image and input image becomes same, function prints "Successfully identified the image" and the loop breaks

• Then we define the standard callback function

• Finally, in man script is call function check

Now, in many parts of the function, a few rospy.sleep() has been provided, as we will see in solutions time delays play a major role in image determination and slight changes in time delays can actually damage the picture and we may never obtain the proper hidden image.

III. Related Work

This whole task was challenging not only because it was just about designing the strategy, but it was also about installing Ubuntu, that too a proper version, with slight fear of loosing your data (well, thanks to Virtual Box), then installing ROS Noetic with terminal, countless tutorials and documents about the installations and using Ubuntu itself. Coming across errors and fixing them were tiring.

Next came learning ROS, it took some time, about 2-3 days at a continuous stretch to know about ROS Topics, scripts, catkin workspaces, publishers and subscribers, their proper utilisations and syntaxes. I mainly learnt about them using Turtlesim package from tutorials available and focused on learning how and in what flow they work.

IV. INITIAL ATTEMPTS

I had to face a lot of difficulties while trying to install and use new applications, operating systems and softwares.

After I installed Virtual Box, I by mistake installed the 22.04 version of Ubuntu and then even tried installing ROS Iron Irwini on it, which failed. I re-read the instructions properly and had to reinstall 20.04 version of Ubuntu on Virtual Box then again ROS Noetic by watching the tutorials.

After ROS was setup, it took around 2 days to learn about ROS and related stuff, faced a lot of problems with creating and running scripts on my own and executing them properly. But with time, things started becoming sorted out.

After everything was working fine, it came very late realisation to me that I had misinterpreted the question which made me think the question is lot easier than it actually was. It took me multiple readings to actually understand the question properly, relate the checker.py script to it. I had started attempting the problem with limited knowledge of how ROS Topic worked. Hence, although I was informed that checker.py need no changes, I was almost certain that there are errors in checker.py, due to which I changed checker.py and ultimately lead to more complicacies.

I experimented with ROS Topics a lot before attempting the real problem, like sending and receiving small messages, modifying them and again sending them to Topics, etc to ensure I grasp the concepts properly.

And I am glad I made so many mistakes because they ultimately helped me learn a lot and make me understand the concepts after I got my hands dirty while random experimenting with player.py and checker.py and trying desperately to properly put them on a loop through ROS Topics. None the less, I started filling in the player.py with doubts in mind and reached towards the final approach

V. Final Approach

After all the problems faced, as I have mentioned in INITIAL ATTEMPTS, I made sure to first understand the concepts clearly, I understood ROS Topics which are basically used as temporary data storage elements to send and receive data to and from scripts with the help of Publishers and Subscribers.

I also successfully learnt to put 2 scripts in a loop through multiple ROS Topics such that data sent by a script will be received by another, will be operated on and passed to the first script again through Topic. This is the main concept on which this problem is based upon.

The flow is as follows:

• Player.py generates an image and publishes to the Topic "/guess" and checker.py takes the image from "/guess" and compares it with the generated hidden image, generates the result image and publishes it to the topic "/result".

• Further, player.py is Subscribed to "/result", hence receives the result image, analyses it and accordingly changes the input image, at the pixels where satisfactory result has not been obtained and is again published to "/guess"

• As the cycle runs again and again, the pixels in input image takes the values of the hidden image in corresponding coordinates and finally as the result image indicates so, the message of Successful Identification is printed'

Player.py:

• The usual libraries like rospy, sensor_msgs, cv_bridge, cv2 and numpy are imported

• Publisher is declared being published to "/guess" of datatype "Image" and bridge is declared as cv_bridge() instance

• I define imgMsg as global and assign it as a np.uint8 matrix with all 0's, i.e. completely black

• Then, the strategy() function, is defined

• Then the standard callback function is defined where the result is taken in and converted to BGR format

• Finally, in the main script, strategy function is called

Initially, I could have taken any input image, but I chose completely black because then, all pixel will only have to increase in value or reman same, eliminating cases of decreasing value which might have complicated the situation. But I have provided case to deal with pixel value more than that of the hidden image, due to a shortcoming of ROS Topics about which I will be explaining later.

Lets now explain the strategy() function:

• We first declare the result variable as global where we will store the result got from "/result" Topic.

• We then initial the node 'player_node' using rospy

• We, then print a few basic instructions and warning about waiting till the full image generates and it may take upto 2 mins to generate the full image and about visibility of slow transformation from black screen to the hidden image

• We then Subscribe to "/result" and put the following statements in a loop

• If the result is none, then result will be assigned uniform matrix of value 1, which is done to ensure that it matches with none of the standard values of 0,127 or 255 and hence, no operation is further performed on imgMsg (the input image)

• Next, with np.where() I determine the co-ordinates from result where the input image has lesser and more value than that of hidden image respectively

• Then, using that data, I change the pixel values as follows: In co-ordinates where value is lesser, I increase the pixel value by 1, and where it is greater, I decrease it by one.

• Finally, I show the image imgMsg (input image), which displays the slow transformation of image from black screen to hidden image when the program is run

• I publish the input image and provide a delay

Now, a general question would be, why do I give the greater condition, when I am starting with all 0's which are smaller than or just equal to pixel values? This question is also intertwined with the question, Why proper delays are necessary?

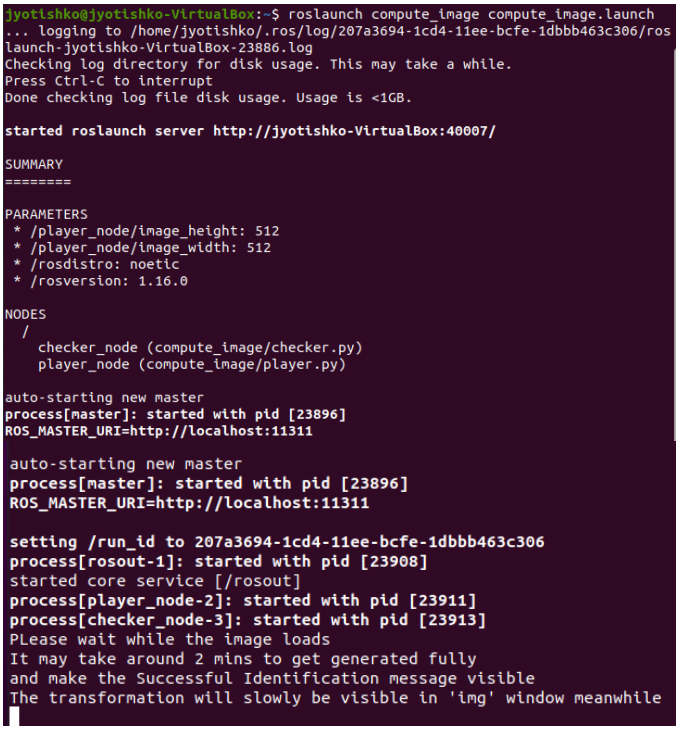
For answering this, proper understanding of how ROS Topics work is necessary. When Script A publishes to a Topic and Script B is Subscribed to it, the Topic works like a hub. As soon as the Topic receives a data, it transfers it to Script B, the topic cannot hold the data permanently. Now suppose if we run both the scripts and a message from A is published long before B is able to subscribe to Topic, the message given by A will be lost and will not be received by B. This is a reason why scripts are put in a loop while Publishing and Subscribing, because continuous data transfer is required.

Now, next comes time delays, time delays done by rospy.sleep() can delay the time of the program and vary when the data will be published to the Topic. In our project, the ideal case would be that a single input image passes, received by checker, result generated, passed to Topic and received by player, then again a single modified image generates and then passes again. But, the execution time of the 2 scripts are definitely not the same, which results in anomaly that might let 2 input images pass while result is being created, and the next image alter the present image's result. For example, if player.py's execution time is more, and suppose input image is published, received and result generated and then quickly another input image comes in, changes the result and then published, we see that a result skips itself and makes inappropriate changes that ultimately damages the image, which results in pixel values getting more than necessary, hence the 'more' condition is necessary to put.

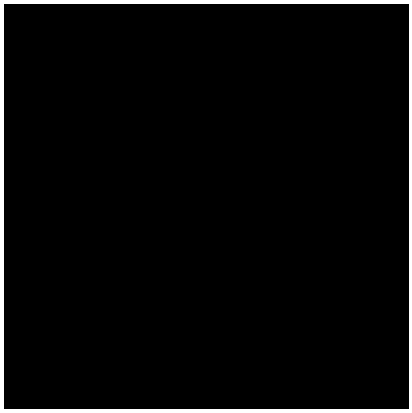
And as for time delay is concerned, execution time of player.py should be slightly more than checker.py, we avoid damaging the image.
So, I had to get a proper delay that wouldn't stretch the program too long but also not damage the image.

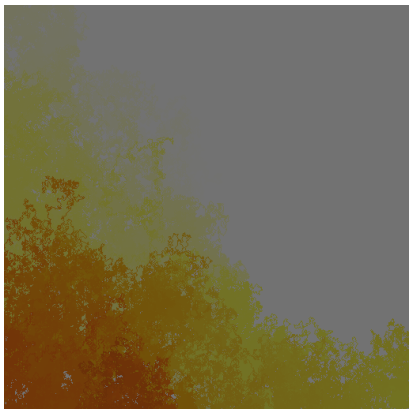
VI. Results and Observation

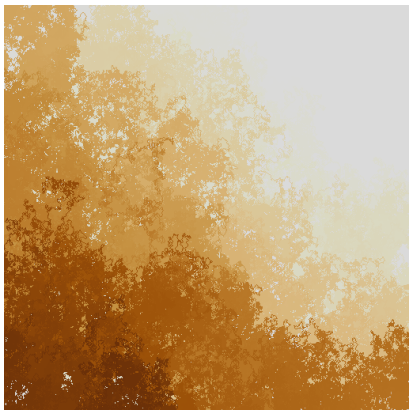
For executing all the files and ROS Master together, we launch the .launch file. It also sets the parameters of the height and width of the 2 images. So, the scripts are executed and the program starts.
A typical output in terminal is shown below.

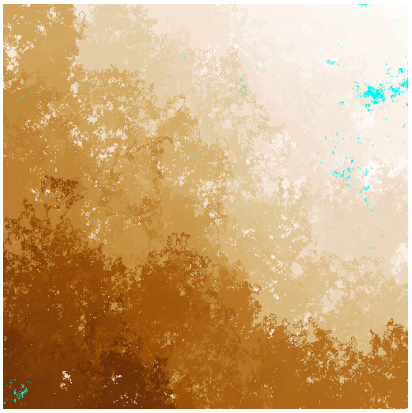


Meanwhile, the following transformation takes place in the "img" window in a span of 2 minutes

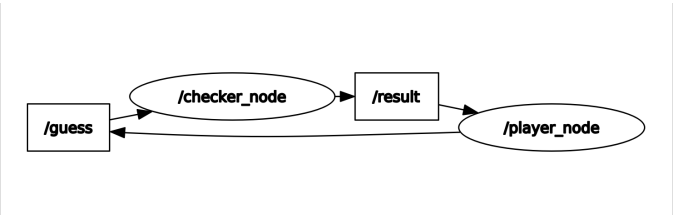








The rqt graph looks like:



Finally, the terminal looks as follows:

```
setting /run_id to 207a3694-1cd4-11ee-bcfe-1dbbb463c306
process[rosout-1]: started with pid [23908]
started core service [/rosout]
process[player_node-2]: started with pid [23911]
process[checker_node-3]: started with pid [23913]
PLease wait while the image loads
It may take around 2 mins to get generated fully
and make the Successful Identification message visible
The transformation will slowly be visible in 'img' window meanwhile
Successfully identified the image
[checker_node-3] process has finished cleanly
log file: /home/jyotishko/.ros/log/207a3694-1cd4-11ee-bcfe-1dbbb463c306/checker_
node-3*.log
```

VII. Future Work

Important Precaution: In line number 67 of **player.py**, I have provided rospy.sleep(0.3) due to reasons of image getting corrupted as I have explained in FINAL APPROACH. This drags the execution time of the program to around 2 mins.

However, this may vary from system to system and time to time and many a times, the delay is not even required. So, it is requested if the program takes a lot of time, then kindly **comment out (#)** the **line number 67** i.e. **rospy.sleep(0.3)** to get the program execute quicker or you can reduce the time less than 0.3. But, if by doing this, **the image gets corrupted**, spawning not expected colors suddenly out of nowhere, please consider keeping the delay within the program.

If image still gets corrupted by even keeping rospy.sleep(0.3), please consider increasing the time to 0.5 more.

To understand this anomaly and solve this, I have to go to more depths of how the ROS works. More reading and understanding is necessary to do this and also to build more advanced projects on ROS.

ROS will be used in ARK to simulate drones and virtual environment to test it virtually and perform various test and

operations. So, ROS is definitely one of the most important applications used in ARK.

CONCLUSION

This was one of the most challenging tasks I faced till now which challenged me from every aspect possible. The success was made possible due to my dedication and long hours of sitting and learning for about almost a week. But overall, I enjoyed every bit of the process and learnt a lot new things that would help me immensely in my skill development and in the future. Still, a lot remains to be learnt in the platform for solving more bigger problems and building more advanced programs.

But, as of now, I am certain that the Kryptonians would be impressed by humanity's intellect and ability to crack their challenge in the perplexing system and finding out the hidden image. The day has been saved and humanity rests freely.

REFERENCES

[1] Programming Knowledge, "How to Install Ubuntu 22.04 LTS on VirtualBox in Windows 11", YouTube, 2022

[2] Programming Knowledge, "How to Install Ubuntu 20.04 LTS on VirtualBox in Windows 10", YouTube, 2020

[3] Robotics Back-End, "ROS Tutorials - ROS Noetic For Beginners", YouTube, 2022

[4] Emil Vidmark, "ROS Tutorials Python - Beginner", YouTube, 2021