ECE 364 Project: Steganography

Phase II

Due: December 3, 2017

GUI Application

Completing this project phase will satisfy course objectives CO2, CO3 and CO4

Instructions

- Work in your Lab11 directory.
- Copy the UI file from ∼ee364/labfiles/Lab11 into your Lab11 directory. You may use the following command:
  `cp ∼ee364/labfiles/Lab11/SteganographyGUI.ui .`
- Remember to add and commit all **required** files to SVN. **We will grade the version of the file that is in SVN!**
- Make sure you file compiles. **You will not receive any credit if your file does not compile.**
- This is the second of two phases for the course project for ECE 364.
- This is an **individual** project. All submissions will be checked for plagiarism.
- Make sure you are using Python 3.4 for your project. In PyCharm, go to:

  File Menu → Settings → Project Interpreter

  And make sure that Python 3.4 (`/usr/local/bin/python3.4`) is selected.

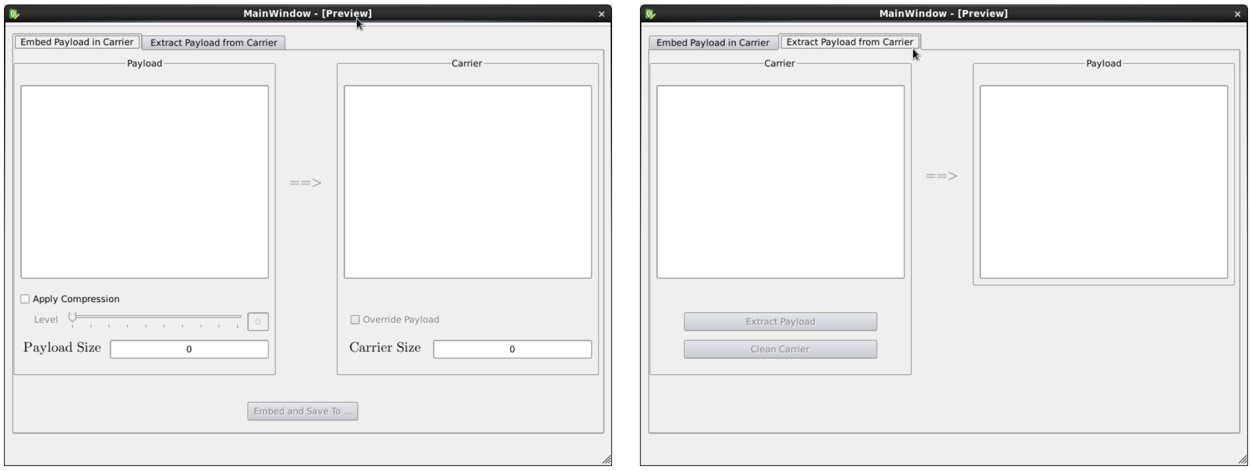
Introduction

By now, you should have completed the first phase of the project, where you have implemented payload embedding and extraction within an image carrier, without affecting its visual appearance. In this phase, you are going to implemented a simple GUI Application, using PySide and the Qt Framework, that facilitates using your "Phase I" code[1] via a visual interface. The main new task that you will have to investigate is the functionality of drag-and-drop of image files over the `GraphicsView` widget.

While your Steganography code supports embedding image and text files as payload types, we will only utilize images, both grayscale and color, with the extension `".png"` as payloads in this phase.

The GUI Design

You are given the UI file `SteganographyGUI.ui`[2] that has been designed using the Qt Designer. Convert it into `SteganographyGUI.py` to be usable by Python, and create a file called `Processor.py` and do your work in it. The GUI window contains two tabs (Figure 1 shows each tab in its initial state,) where each tab performs independent functionality. The user will select the tab that he/she wants to use, and carry on the desired work.



(a) Embedding Tab

(b) Extraction Tab

Figure 1: The Initial State of the Application for Both Tabs.

Tab 1: Payload Embedding

In this part of the application, the user can perform embedding of a payload into a carrier. In order to do that, the user must provide both the payload and the carrier images, as well as some additional options to make sure the action can be performed.

Dropping a Payload

Acquiring a payload in the application should happen via dragging the desired image file, and dropping it on the `GraphicsView` widget. If the user drops any file that is not a valid image, the drop should be ignored.

[1]Please note that if your Phase I code is not working, we will *not* be able to grade your code for this phase, and you will not receive any credit.

[2]You are encouraged to view the UI file in the Qt designer, as this would give you faster access to widget properties. You are not expected, though, to modify the content of the UI. If you feel that you need to do so, please let your TA knows before you carry on any changes.

Additionally, the user can repeat the drag-and-drop process, and at any time a drop is accepted, the image must be displayed. Figure 2 shows the application state after accepting a payload image drop.

Notes:

- If a drop is accepted, the "Payload Size" must be immediately calculated, and the value must be shown in the text box. (The payload size is the length of the JSON string, which, obviously, needs to be constructed.) At this point, no compression should be applied to the payload.
- At any point a drop is accepted, the compression check box should be unchecked, and the compression-level widgets below the check box (the label, the slider and the text box) must all disabled. Also, the slider value should be reset to 0, and the text box should display 0 as well.

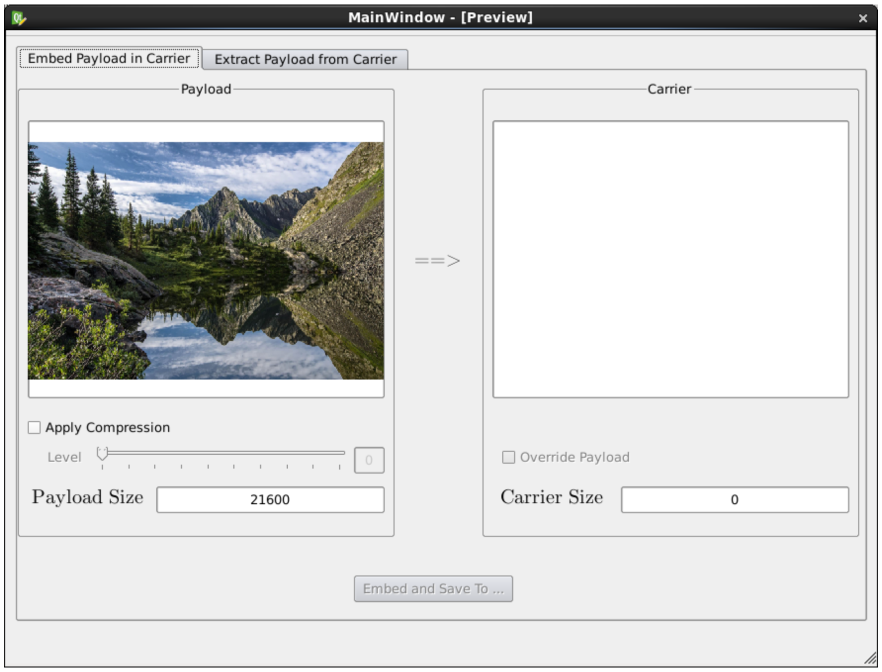


Figure 2: Application State after Dropping a Payload Image.

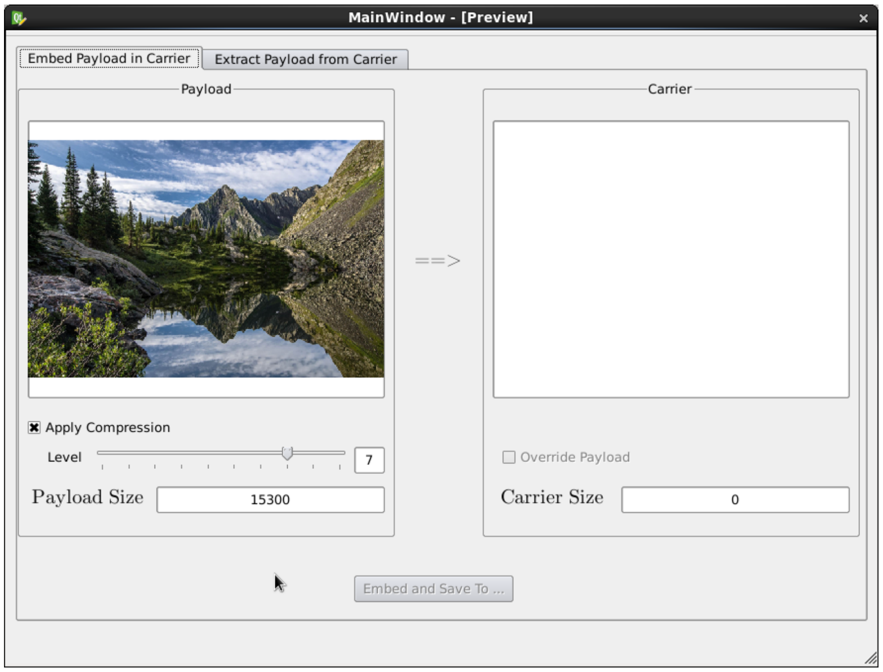
Applying Compression

The compression behavior is controlled via the compression check box and the compression-level widgets. If the user desires to apply compression, he/she will check the compression check box, at which point the compression-level widgets should all be enabled. Figure 3 shows the application state when working with compression.

Notes:

- The slider does not display the selected value. Hence, at any point the slider changes its value, you should update the text box to display that value.
- The moment compression is enabled, the "Payload Size" should be re-calculated with compression-level 0, and the value updated in the text box.

- The user can change the compression-level via the slider, which goes between 0 and 9 inclusive. At any point the user changes the slider, the "Payload Size" should be re-calculated with the corresponding compression-level.
- If the compression check box is unchecked, while a payload is present, simply disable the compression-level widgets, and recalculate the "Payload Size". There is no need to reset the slider and the text box values, as the user may re-check the check box again, at which point you can simply re-use the present compression-level value.



[✓] Apply Compression
Level: 7
Payload Size: 15300
[ ] Override Payload
Carrier Size: 0

Figure 3: Application with Compression Applied to the Payload.

Dropping a Carrier

Similar to the payload, acquiring a carrier image should happen via dragging the desired image file, and dropping it on the `GraphicsView` widget. Moreover, the checking for the dropped file type, and the process repeatability is also applicable to the carrier. Figure 4 shows the application state after accepting a carrier image drop.

Notes:

- Similar to the payload case, if a drop is accepted, the "Carrier Size" must be immediately calculated, and the value must be shown in the text box. The carrier size is the maximum message size it can hold, which equals the total number of its pixels.
- If the carrier already includes a payload, the label below the image should display a message indicating that fact, and the "Override" check box enabled, as shown in Figure 4. If the carrier does not have a payload, then the label should not be displayed, and the override check box should be unchecked and disabled.

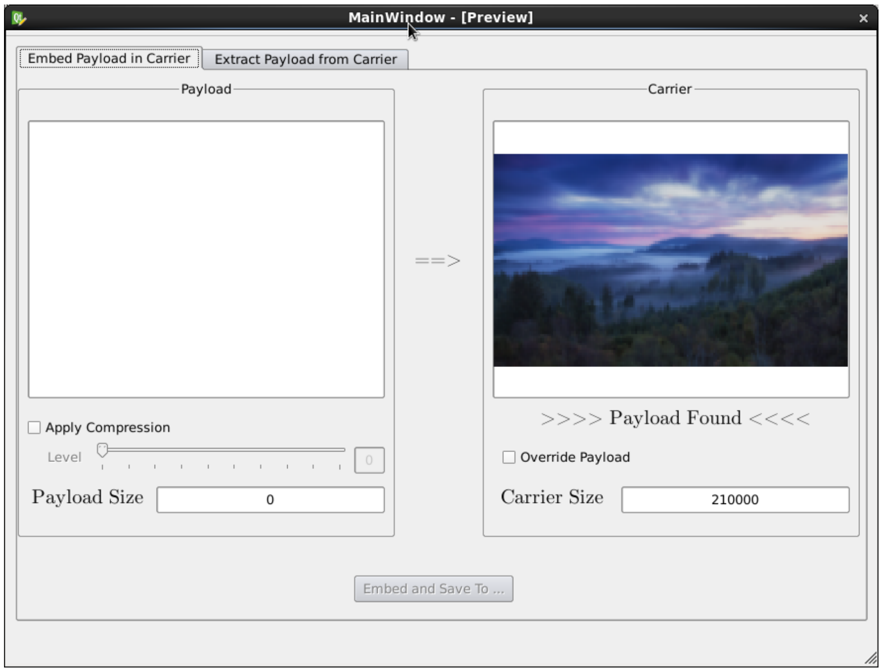


Figure 4: Application State after Dropping a Carrier Image.

Embedding and Saving

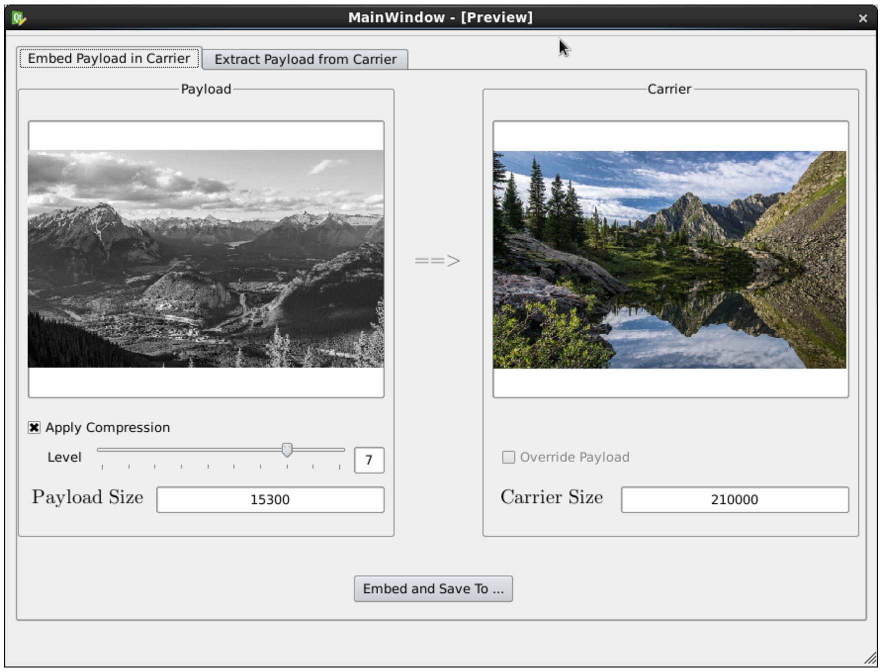
Given both, the payload and the carrier, the application can perform the embedding process. If the user clicks on the "Embed and Save To" button, a file dialog should appear asking the user for a target file name and location. If the user provides one, the application should embed the given payload in the carrier and save the result in the provided file name. Note, however, that the application must check for the eligibility of the embedding process before performing it. There are two conditions that must be satisfied:

- The carrier size is larger than, or equal to, the payload size.
- The carrier does not have a previous payload, or it does contain one, but the override check box is checked.

If both of these conditions are met, the embedding button can be enabled, otherwise, it must be disabled. Figures 5 and 6 show the two eligible cases.

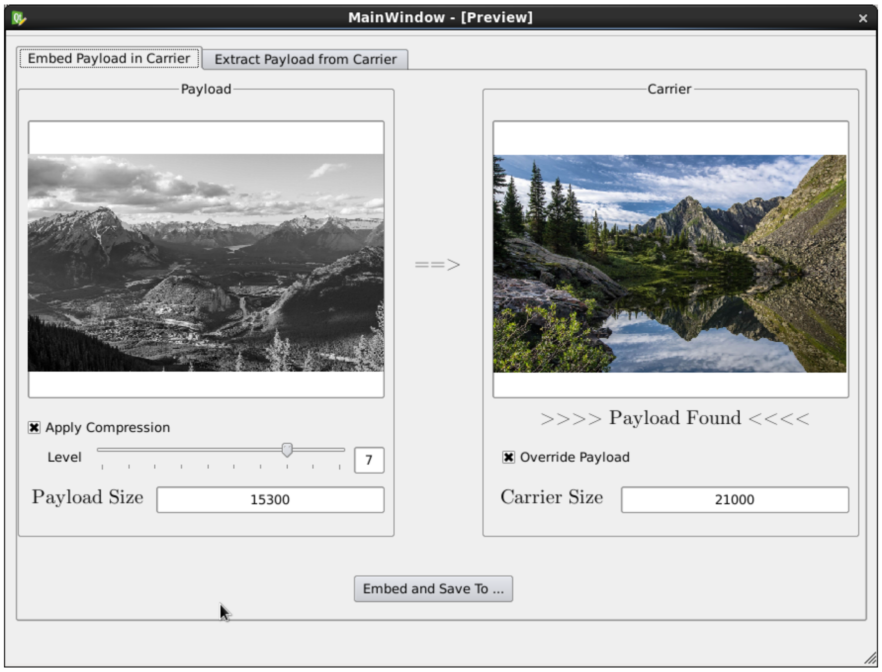
Notes:

- Any change in the payload image (re-drop, applying/removing compression or changing the compression-level) will change the payload size, and hence must trigger the validation of the embedding process.
- Similarly, any change in the carrier image (re-drop or modifying the Override flag,) must trigger the validation of the embedding process.



Apply Compression [✓]
Level 7
Payload Size 15300
Override Payload [ ]
Carrier Size 210000

Figure 5: Application Ready to Save with no Prior Payload.



Apply Compression [✓]
Level 7
Payload Size 15300
>>>> Payload Found <<<<
Override Payload [✓]
Carrier Size 21000

Figure 6: Application Ready to Save with a Prior Payload.

Tab 2: Payload Extraction

In this part of the application, the user can perform the extracting of a payload from a carrier. The user must *only* provide the carrier image, and decide whether to extract the payload, or clean the image.

Dropping a Carrier

As explained in the previous section, acquiring a carrier image should happen via dragging the desired image file, and dropping it on the `GraphicsView` widget. Moreover, the checking for the dropped file type, and the process repeatability is also applicable. Once a carrier drop is accepted, there exists one of two possibilities:

- The carrier image may be empty, which means that we cannot perform any operation on it. In this case, the label below the carrier image should be displayed, and it should provide a message indicating that it is empty, and both of the buttons should be disabled as well. Figure 7 shows an example of an empty carrier.
- The carrier image does contain a payload. In this case, the label below the carrier image should not be displayed, and both of the buttons should be enabled. Figure 8 shows an example of a carrier with a payload.

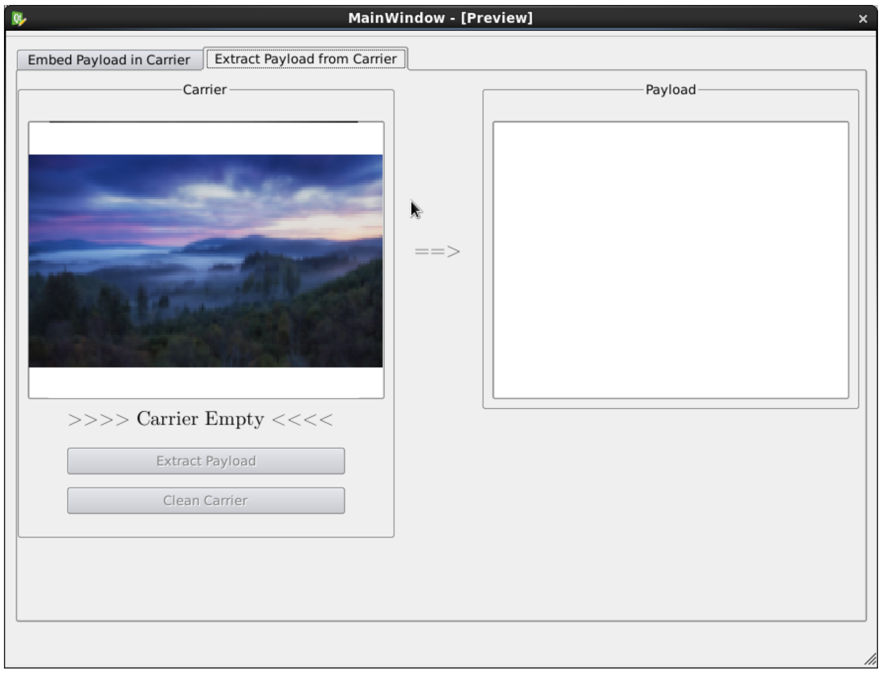


Figure 7: Empty Carrier Image in the Extraction Tab.

Cleaning and Extraction

If a payload is present in the carrier, there are two operations that can be performed:

- The user can extract the payload by clicking on the button. Doing that should display the payload in the relevant `GraphicsView`. Note that the Payload view in the extraction tab is the only view that does not need to support the drag-and-drop functionality. Figure 9 shows an example of an extracted payload from a carrier. (You may choose to disable the Extraction button after displaying the payload, but that's not required. If it stays enabled and the user re-clicks on it, you can just repeat the extraction process.)

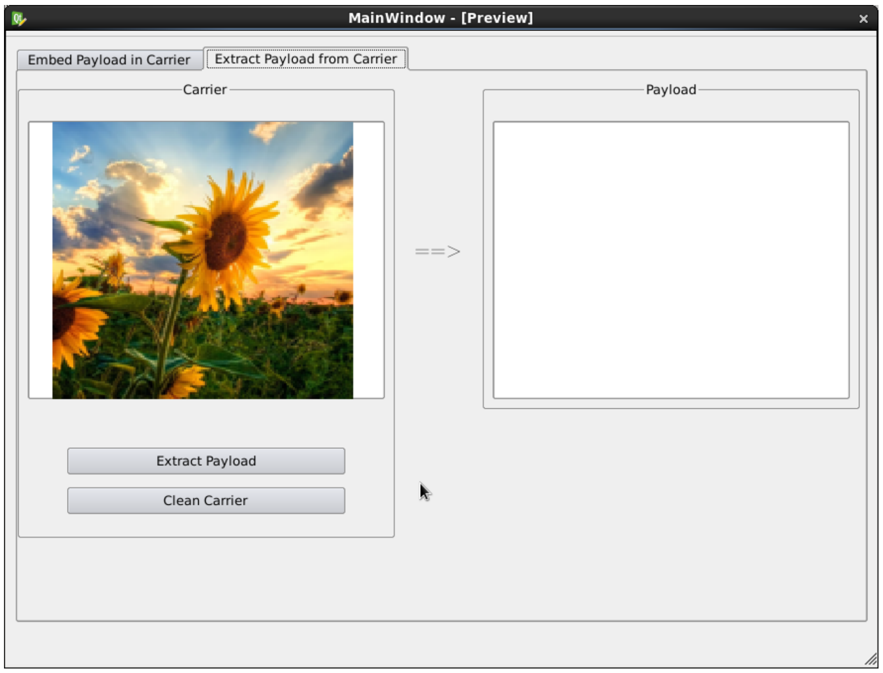


Figure 8: Carrier Image with a Present Payload.

- The user can choose to clean the carrier. Clicking on the Clean button should perform the cleaning operation on the carrier and then should save it in place of the original carrier image file. This would render the carrier without a payload, so the application should be updated to be as in Figure 10. Notice how a cleaned carrier shows up as if it is empty to begin with, as in Figure 7.

Finally, if the carrier image is changed (via another drop) after a payload has been extracted, the payload view should be cleared, since it will not belong to the new carrier.

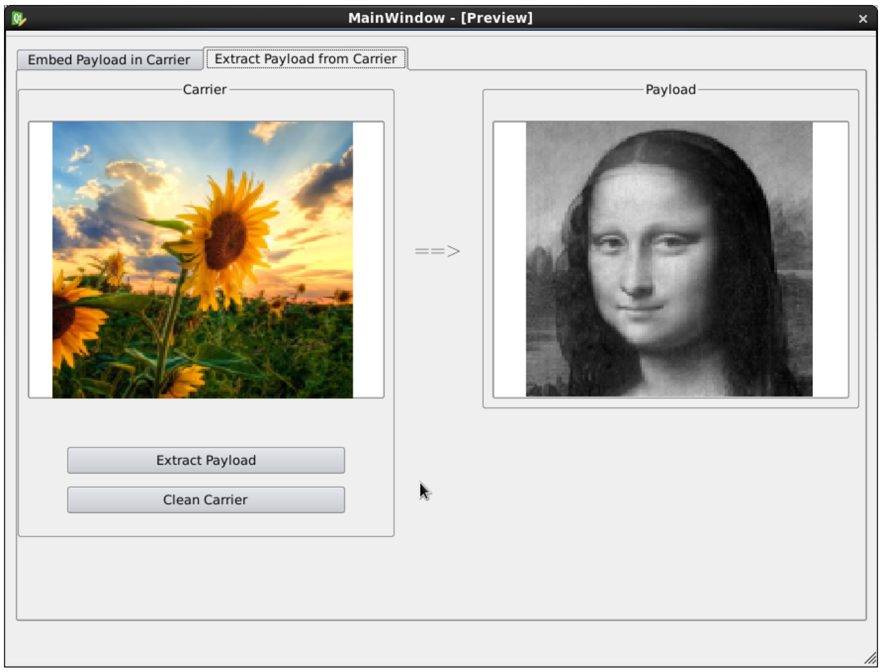


Figure 9: Carrier Image with the Payload Extracted.

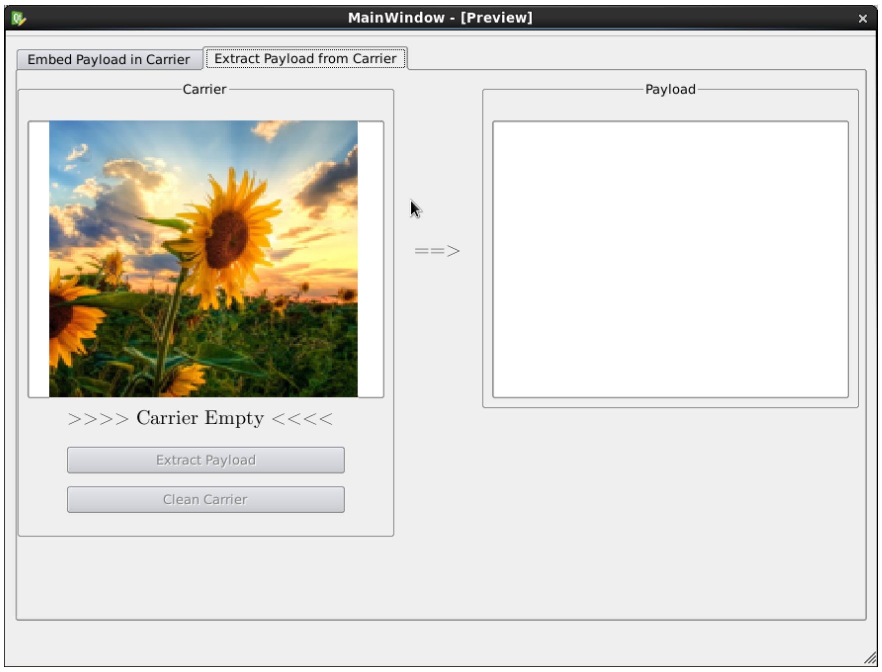


Figure 10: Carrier Image after Cleaning.

Comments

- In the Embedding Tab, to perform an extraction, both the payload and the carrier images must be present. The order of their presence should be of no importance, since handling of each one is independent. The user may either start with dropping a payload or a carrier, on its respective widget.
- When you display the image in the widget, please note the following:
  - It is acceptable for the image to be smaller than the boundaries and not fill the whole widget.
  - It is *not* acceptable for the image to be larger than the widget, which will cause scrollbars to be visible in the widget.
  - It is *not* acceptable for the image to be deformed. In other words, the aspect ratio of the image must be maintained.
- Your grade for this phase is based on your demo to the TA. **Failing to demonstrate your work to your TA will result in a zero credit for this phase**. Moreover, please note that the final judgment for passing the use-cases rest with the TAs. If you have any doubt about the application behavior, it pays to ask first before you find out at demo time.
- Your submission must consist of the files:
  1. `SteganographyGUI.ui`.
  2. `SteganographyGUI.py`.
  3. `Processor.py`.