ECE 364 Project: Homography
Phase II

Due: December 07, 2015

Homography GUI

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

Instructions

- Work in your Lab11 directory.
- There are no files to copy for this lab.
- This is the second of two phases for the course project for ECE 364. Each phase is worth 12% of your overall course grade.
- This is an **individual** project. All submissions will be checked for plagiarism using an online service.
- Remember to add and commit all files to SVN. Also remember to use your header file to start all scripts. **We will grade the version of the file that is in SVN!**
- Name and spell your scripts exactly as instructed. When you are required to generate output, make sure it matches the specifications exactly, since your scripts might be graded by a computer.
- Make sure you are using Python 3.4 for your project. To verify/modify the Python version, 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 performing a transformation of a source image onto a container image. In this phase, you are going to implement a simple GUI Application, using PySide and the Qt Framework, that uses the functionality you have completed. The purpose of the GUI Application is to quickly perform several homographies and save the results. For this phase, all of the functionality that you will implement must be accessible from the GUI, and not from the command-line.

The GUI Design

Create a UI file called `HomographyGUI.ui`, using the Qt Designer, that contains the layout shown in Figure 1. Convert it into `HomographyGUI.py` so that you can import it into your Python code, and create a file called `HomographyApp.py` that would consume the GUI file[1]. The figure shown uses the following widgets[2]: `QPushButton`, `QGraphicsView`, `QLineEdit`, `QLabel` and `QComboBox`. Check with your TA if you prefer to use other widgets that satisfy the required functionality. Note that the text boxes should be read-only, and the button "Acquire Points" checkable (i.e. has the attribute checkable set to true,) to enable the toggle behavior. Also, note that the "Effect" drop-down widget should contain the options shown in Figure 2.

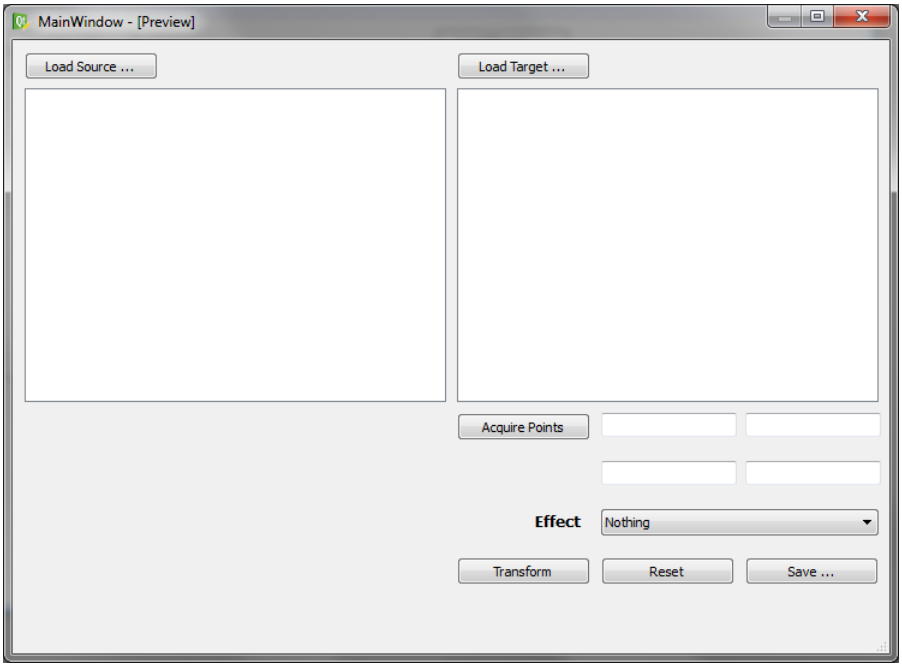

Figure 1: The Homography GUI.

[1] Remember to only work in the consumer file, as you may override the GUI Python file and lose your work.
[2] Please refer to the PySide documentation to understand the functionality of the available widgets.

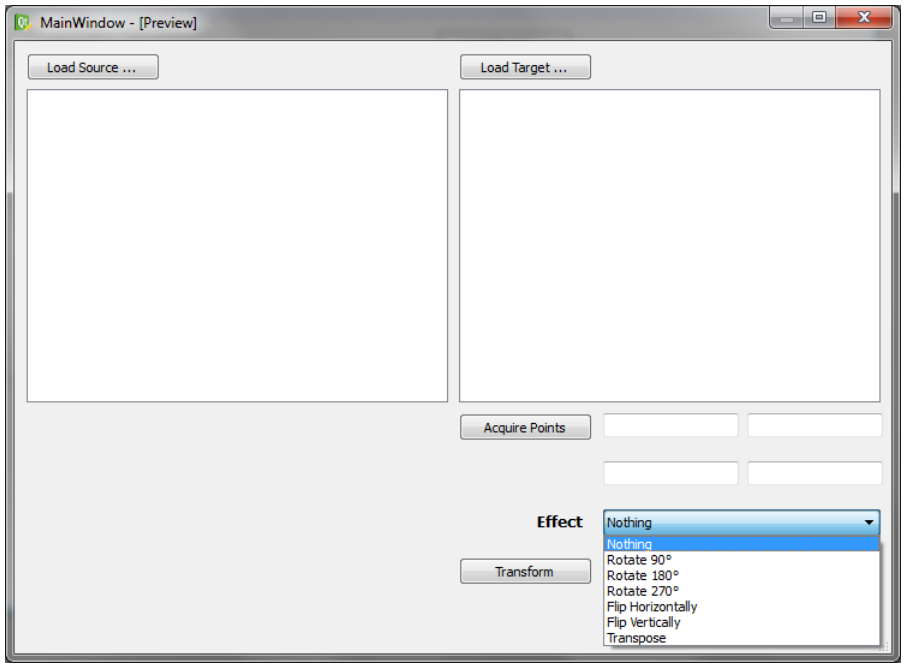

Figure 2: Available Transformation Effects.

Application Behavior

Designing a functioning and a stable GUI application is a non-trivial task that grows with every widget you add to the application. The simplest way to control the application behavior is to model it after a "Finite State Machine." In this phase, we are going to define a set of application states, and your job will be to make sure that the application is always in one of those states.

Once you understand the application states, it is recommended that you draw a state-transition diagram and verify the behavior of the application with your TA before you start coding.

Initial State

When you first run the application, it should start with all widgets disabled, except for the two loading buttons, as shown in Figure 3. This will force the user to load the source and target images to be able to continue. Clicking on these buttons should pop up a file selection dialog box that allows the user to select an image, and once the image is selected it should be displayed on the widget below the button. You may assume that both images will either be grayscale or color images.

Loaded State

Once you verify that both images have been selected by the user, you should enable the button "Acquire Points" and the text boxes next to it, as shown in Figure 4. In this state, there are three possible actions that the user can do:

- Reload a different source image, via clicking again on the "Load Source ..." button.
- Reload a different target image, via clicking again on the "Load Target ..." button.
- Select the target points, via clicking on the "Acquire Points" button, which will change the application state into the "Point-Selection State".

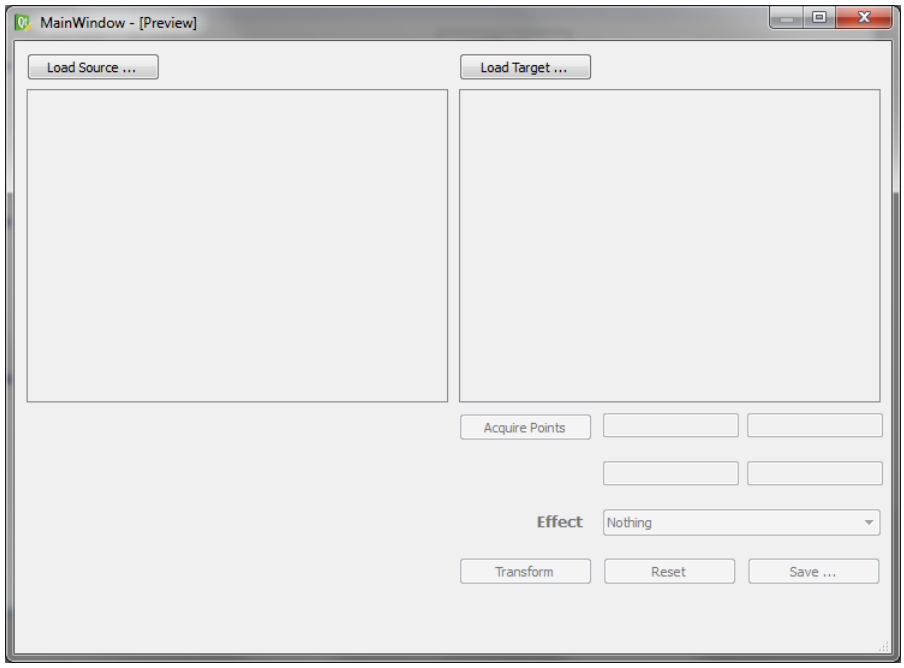

Figure 3: The Initial State of the Application.

Point-Selection State

Clicking on the "Acquire Points" button will disable the loading buttons, to prevent the reloading while selecting the target points, and bring the application into the "Point-Selection State." The behavior for selecting the points should be as follows:

- Using the mouse, the user will hover over the target image and click to select the first point.
- Once the point is selected, populate the first text box with the coordinates of that point.
- The selection behavior is repeated until all points are selected.
- If at any point in time the user is not satisfied with the point selection, he/she can press the backspace key on the keyboard once, and the last point will be deleted, i.e. the last text box populated will be cleared.
- If the backspace key is pressed again, the prior point is deleted, and so on. (If no more points are available to delete, pressing the backspace key must have no effect on the application.)
- The order of the points selected should be as defined in Phase I: Upper-Left, Upper-Right, Lower-Left, Lower-Right.
- If the "Acquire Points" button is un-toggled before all four points are selected, reset all text boxes, and enable the loading buttons, i.e. send the application into the "Loaded State".
- If the "Acquire Points" button is un-toggled after all four points are selected, enable all widgets, which will send the application into the "Ready State."

Figure 5 shows the application in the "Point-Selection State" with three points selected.

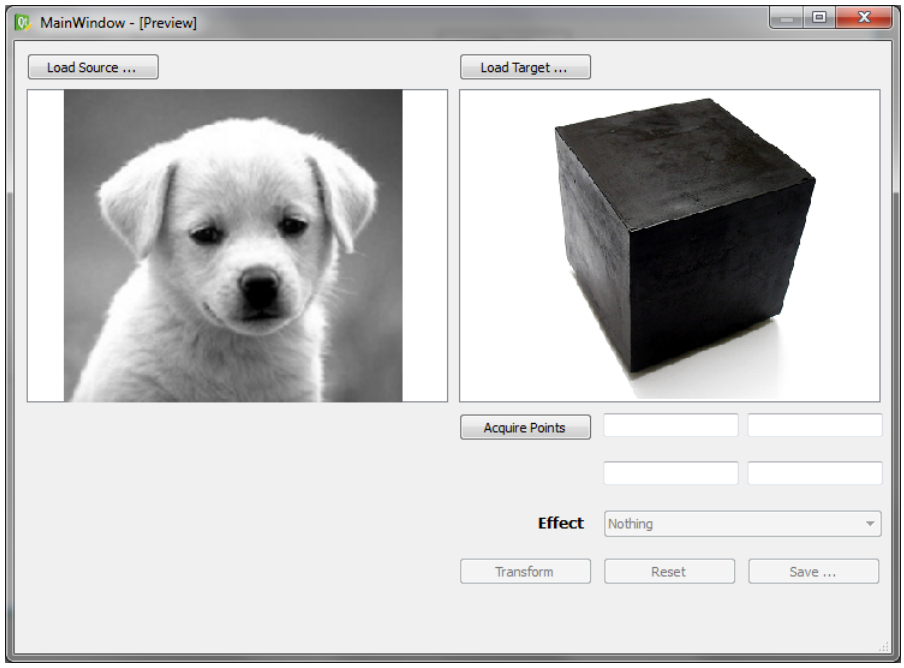

Figure 4: The application window with the source and target loaded.

Ready State

The application now has all the information it needs to perform the homography as shown in Figure 6. The expected behavior at this state is as follows:

- Clicking on "Load Source ..." will allow for reloading a new source image.
- Clicking on "Load Target ..." will allow for reloading a new target image. If a new target image is indeed selected, reset the text boxes, and send the application into the "Loaded State."
- Clicking on "Acquire Points" will send the application into the "Point-Selection State."[3]
- Changing the option in the "Effect" drop-down should have no additional side-effects, i.e. it will not affect any other parts of the GUI.
- Clicking on "Reset" will have no effect in this state.
- Clicking on "Save..." will pop up a dialog box to select a target file name to save the target image to a different file.
- Clicking on "Transform" will apply the homography, with the selected effect, and show the result. It will also disabled the point selection mechanism, which will bring the application into the "Transformed State."

Transformed State

The main difference between this state and the "Ready State" is that we have applied the homography. While this is a subtle difference, it will have a significant impact on the application behavior, which should be as follows:

[3]While the "Point-Selection State" has the text boxes as empty, you may want to keep the point data intact in case the user clicked on the button by mistake. This, however, is an optional behavior.

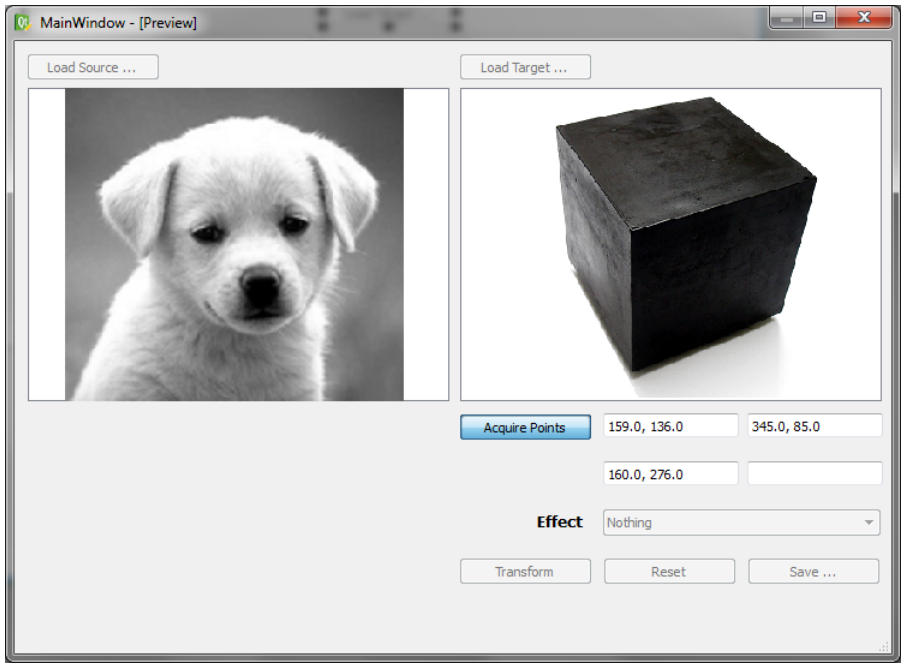

Figure 5: The application window state for target points' selection.

- Clicking on "Load Source ..." or "Load Target ..." will allow for reloading a new image. If either a new source or a target image is selected, reset the text boxes, and send the application into the "Loaded State." Additionally, reset the target image to its original content, i.e. without the homography.
- Clicking on "Save..." will pop up a dialog box to select a target file name to save the target image, with the applied homography, to a file.
- Clicking on "Reset" will reset the target image to the pre-transformed state, i.e. it will send the application into the "Ready State."
- Changing the option in the "Effect" drop-down should have no additional side-effects, i.e. it will not affect any other parts of the GUI.
- Clicking on "Transform" will apply the homography, with the selected effect, and show the result. The application will still be in the same state afterwards.

Figure 7 shows the application in the "Transformed State." Note how the "Acquire Points" button, and the text boxes are all disabled.

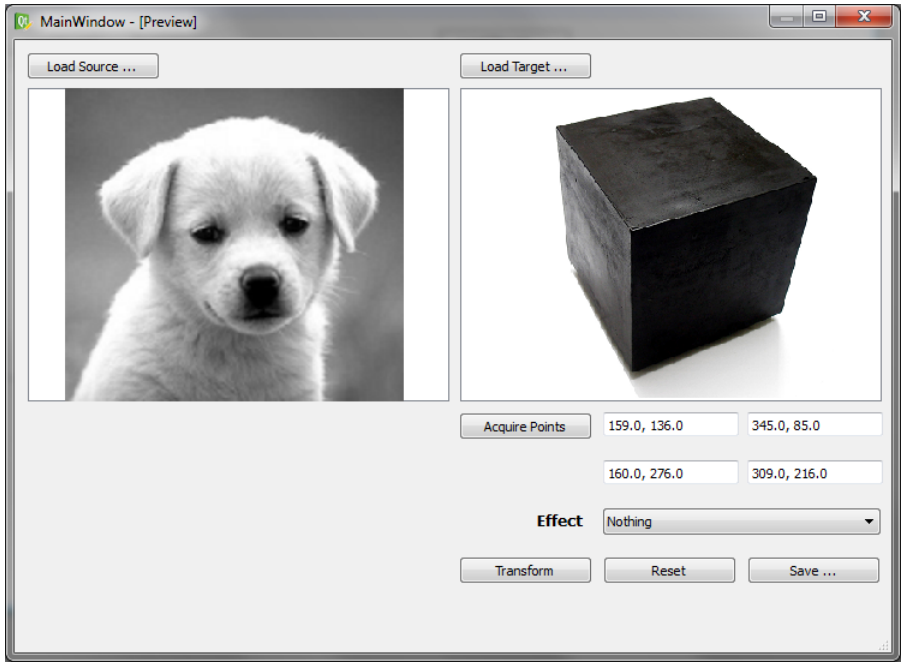

Figure 6: The application window in "Ready State."

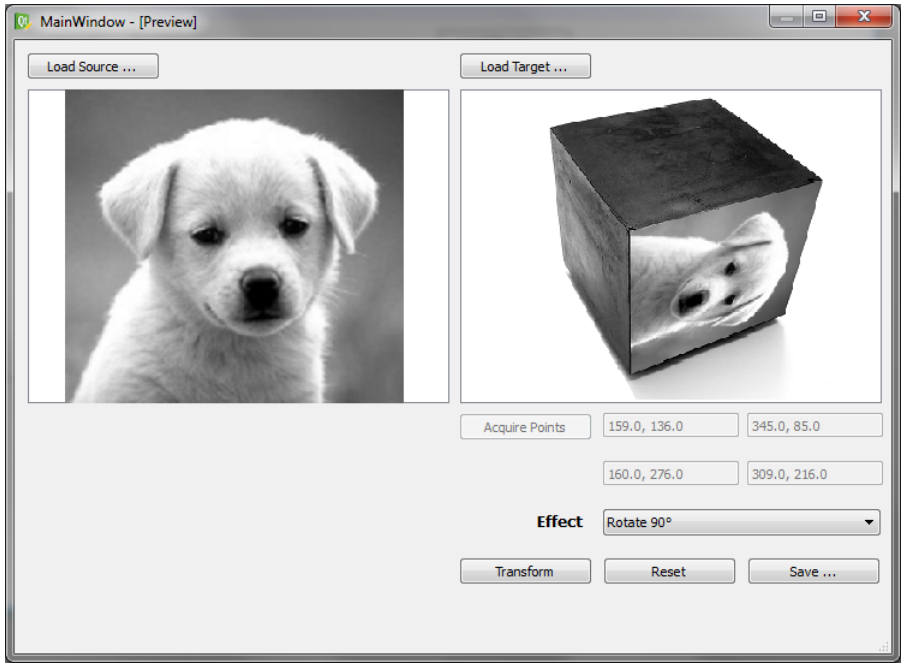

Figure 7: The application window in "Transformed State."

Extra Credit

Subregion Transformation

The code you have written so far performs the transformation over the whole source image. Extend your code to allow for transforming only a subregion of the source image, where the subregion is a rectangle defined by two points: the upper-left and the lower-right. You will need to extend your Phase I code and modify the GUI to account for the new functionality.

GeneralTransformation Class

Extend your `Transformation` class to allow for passing in a set of source points. Create the new class named `GeneralTransformation`, in your `HomographyApp.py` file, that inherits from `Transformation` and has the following initializer:

- `__init__(self, sourceImage, sourcePoints=None)`:

  Initialize a new instance of the class with the given source points. If no source points are passed, use the bounds of the whole image.

Note that this new class must work with grayscale and color images[4].

GUI Enhancement

Update your GUI to allow for acquiring the subregion in the source image, as shown in Figure 8. The behavior of the application when clicking on the button "Acquire Points" for the source image should follow the behavior of the same functionality in the target.

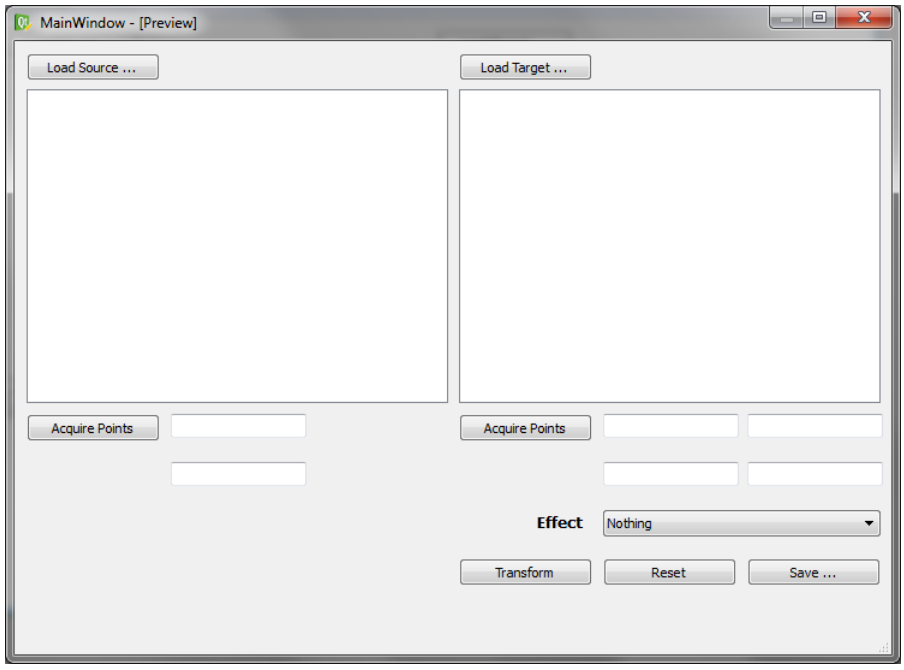

Figure 8: The enhanced GUI that allows for acquiring source image subregion.

[4]You might want to investigate multiple inheritance to avoid code duplication.

Grayscale & Color Mismatch

The standard behavior of the application assumes that both the source and target images are either both grayscale, or both color images. Extend the behavior to account for the following:

- If the source is a grayscale image and the target is a color image, embed the source in every channel in the target, and save the transformation result as a color image.
- If the source is a color image, and the target is a grayscale image, convert the target image into a color image, where each channel is a replica of the original grayscale image, and save the transformation result as a color image as well.

Comments

- When you display the image in the widget, please note the following:
  - It is acceptable for the image to be smaller 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. `HomographyGUI.ui`.
  2. `HomographyGUI.py`.
  3. `HomographyApp.py`.