Image Registration

Elizabeth Chilcoat, Kelle Clark, Michael Heath

November 28, 2020

1 Image Registration

Image registration is the process of aligning a signal to corresponded to the same basis of the transmission space of another signal so that the information from the two signals can be compared or integrated. The signal to be aligned is the target signal and the second signal is designated the reference or source signal. The methods used to perform the alignment can be classified based on the approach to matching the signals. These processes either focus on matching the signals' intensities or features.

The signal to be aligned in our application is an image uploaded as Image 1 while the source signal is the image uploaded as Image 2, and the application utilizes a feature based approach to align these images. Key points in the source and destination images are used to generate a "preserving" mapping, a homography, between the images. An automatic image registration feature is provided where the homography is constructed with the use of cv2.cornerHarris, cv2.findHomography and the choice of model cv2.RANSAC. RANSAC is an iterative algorithm that calculates parameters of the desired homography. The user can also elect to perform manual image registration after selecting a set number of key points to be used from both images.

2 Usage and System Dependencies

2.1 Dependencies

The user should check to see if the following libraries are installed or do a batch install with

```
pip install -r requirements.txt
```

The libraries and versions used by the team include:

```
opencv-python==4.4.0
numpy==1.19.1
filetype==1.0.7
matplotlib==3.2.2
Pillow==7.2.0
```

Users of MacOS may have to install opencv-python-headless instead of opencvpython. We tested on MacOS using the packages listed below.

```
matplotlib==3.2.2
opencv_python_headless==4.4.0.40
numpy==1.19.1
filetype==1.0.7
Pillow==8.0.1
```

2.2 Usage

To run the application with the given dependencies installed, a user can enter:

```
>python imgreg.py -h
```

This command will signal the script to print instructions of how to use the application and the program's main function to the terminal as shown in Figure 1.

```
>> python imgreg.py -h
```

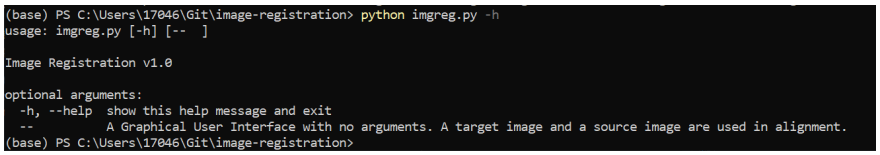

Figure 1: The application's command line help message if the program is run with the optional -h parameter.

Alternatively, a user can run the program without the -h flag, see Figure 2 below. This event will cause the system to launch a Graphical User Interface that provides the user with the option to upload two images. The interface is shown in Fig 1, using the command line instruction below that will generate a description of the code's usage:

```
>> python imgreg.py
```

Two images are required to be uploaded to the system before any image registration can be performed. Alert messages will be displayed if a user attempts to select points, perform alignment or save an image if both images are not uploaded. Once the images have been successfully rendered in the interface, the user may chose to perform image registration on Image 1 either manually or automatically.

For automatic image registration, the user must first chose two images that will be displayed on the left of the interface. The user then selects the Align Automatically button, the "A" or "a" on the keyboard. If the user selects Align Automatically before uploading both images, an alert will be given. In Figure

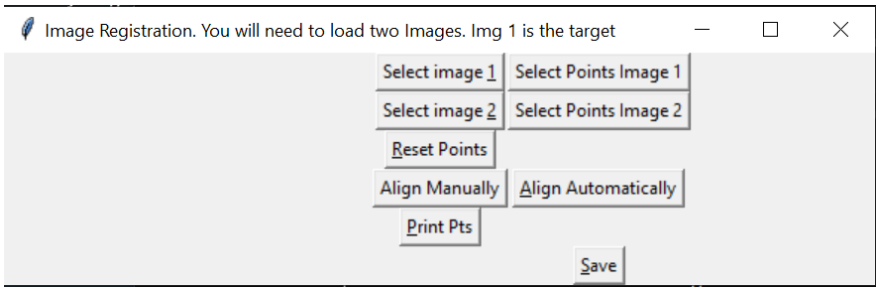

Figure 2: The GUI display when imgreg.py is run without optional -h.

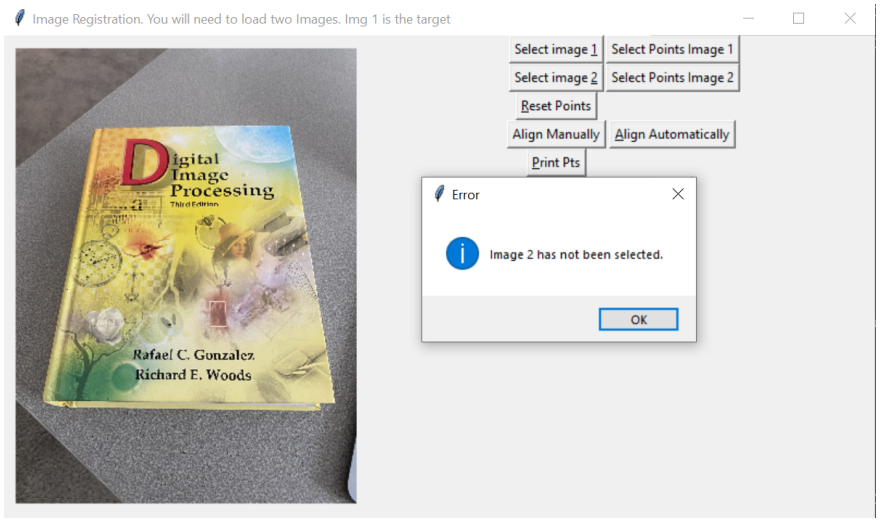

Figure 3: An alert is given when only one image has been uploaded and automatic alignment is chosen.

3, the alert is given when the second image has not yet been uploaded but automatic image registration has been selected.

For automatic alignment, the system uses the Harris Corner Detector library function to identify points of interest in Image 1 then a cv2 library function to find a homography using the RANSAC algorithm. The points given by the Harris Corner Detector are displayed on both of the images in red (Figure 4) and the aligned Image 1 is placed in the interface on the right (Figure 5). Since an aligned image has been created, the user can now access the save feature of the system.

To align an image manually, given the source and target images are uploaded, the user must first chose key points (an exact number of four points per image is hard coded currently). If the user presses the Align Manually button before choosing points in both images, an alert will be displayed prompting the user

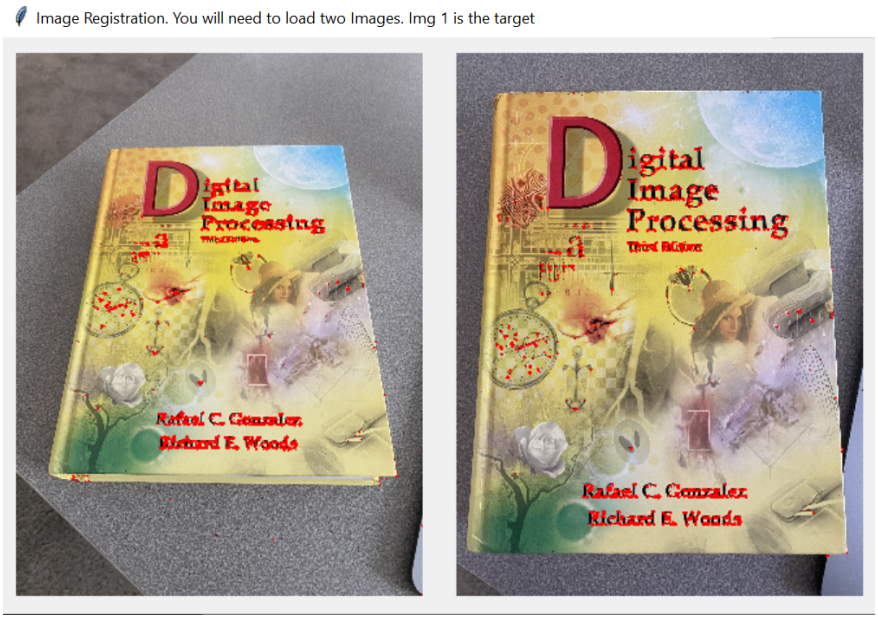

Figure 4: The key points chosen with Harris corner detector.

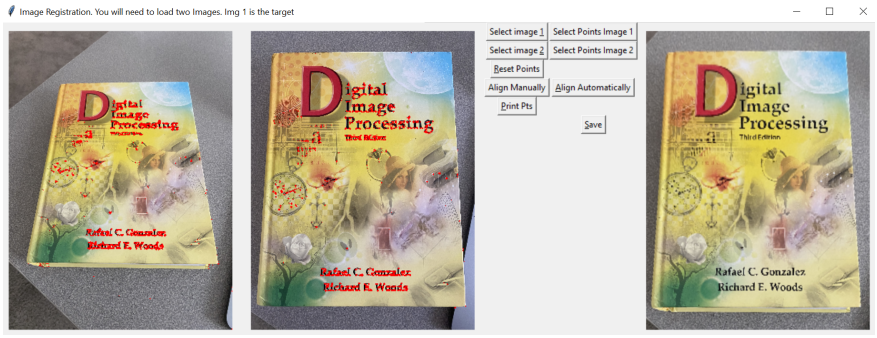

Figure 5: The result of aligning the target Image 1 with the source, Image 2, with Auto Alignment.

to first choose points in an image. In Figure 6, the system was previously displaying the result of Align Automatically on the two images as in Figure 5. The selection of the Align Manual button causes the system to remove the key points created by the Harris Corner Detector and display the warning on the left of Figure 6.

To select points on the images for Align Manually, the user will select one of the buttons for selecting points. A dialogue box will appear that instructs the user to select four points with mouse clicks on the image (see Figure 7). The

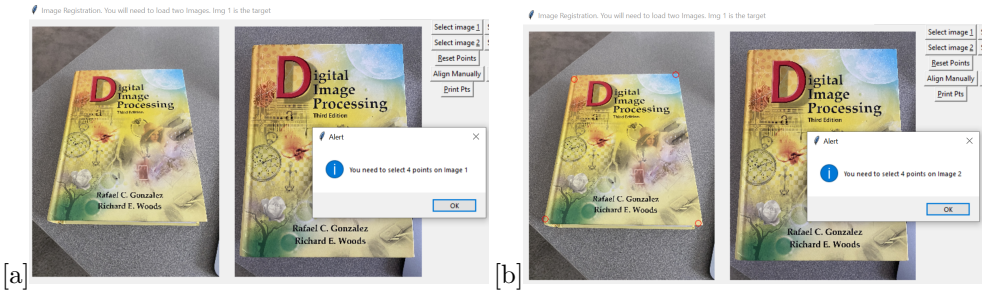

Figure 6: (a) no points chosen in Image 1 prior to manual alignment creates an alert (b) no points chosen in Image 2 prior to manual alignment also creates an alert.

selected points of interest that the user chooses will appear as red circles on the image (Figure 8).

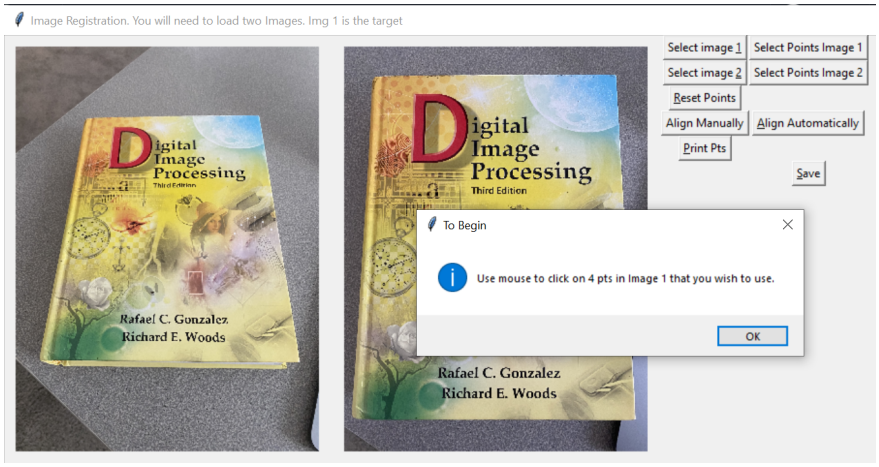

Figure 7: The user is prompted to select points on the images prior to manual alignment.

The user may also "start over" with their point selection. Pressing the button marked Reset removes points from both images.

Points chosen can be printed to the command line with the button labeled Print Pts. The images below, Figures 9 and 10, demonstrate four points selected manually at the four corners of the book in each image. The coordinates of these points are printed to the command line by using the button Print pts.

The user may quit the program with the "q" key on the keyboard or save an image with the "s" key. When saving an image, the user will be given an opportunity to choose a location and name for the file.

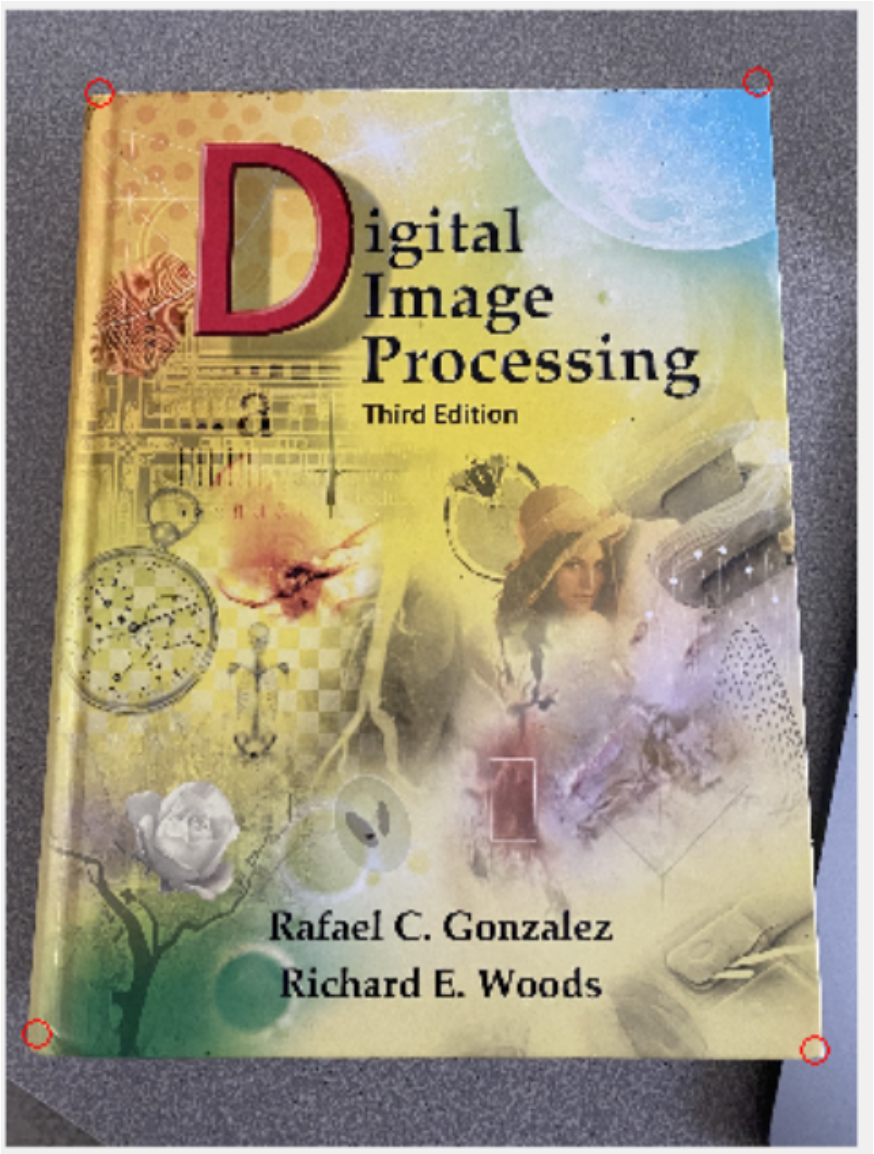

Figure 8: Red circles are placed at the feature locations chosen by the user.

3 User Requirements Log

The design, implementation and testing of the system are based on satisfying the following stated user requirements. The GUI component of the system from project 2 (pixel intensity transformations) was tested previously to ensure that a user is allowed to save a file at the chosen location and name and to display

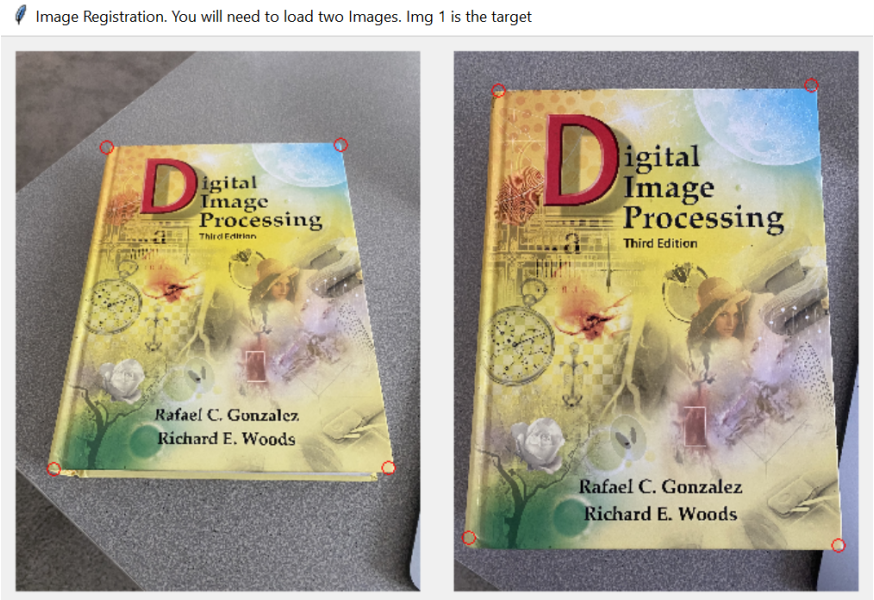

Figure 9: The four corners of the book in each image have been selected as the key points for manual alignment.

```
[(72, 76), (253, 74), (290, 324), (31, 325)]
[(36, 32), (278, 28), (13, 378), (299, 384)]
```

Figure 10: The coordinates chosen on the two images using Print pts.

program information with the optional -h argument.

3.1 Project Requirements:

The project requirements were derived from the provided project description:

https://ispel.cs.ecu.edu/csci4150/pbl-projects/image-registration/

Functional Requirements:

1. The system should run with the command python imgreg.py
2. The system should display help to run the program with the command python imgreg.py -h
3. The system should allow a user to choose two images.

4. The system should allow a user to perform image registration on the target image using the source image automatically and display the resulting image.

5. The system should allow a user to perform image registration on the target image using the source image manually and display the resulting image.

6. The system should allow a user to chose to save a transformed image using the key entry s and prompt the user for the name to save a file as.

7. The system should terminate when the user presses the "q" on the keyboard.

8. The system should perform the expected output, the target image should be visibly aligned with the source image.

4 Software Model, Development Methodology

The team reconfigured and added to the programs written to satisfy the first four assignments for this project that allowed the team to meet the stated requirements.

5 Testing

The system and components of the system were tested manually to determine the behavior when the system is used as expected. To the best of the team's ability, the system was designed to anticipate entry and usage errors and handle these exceptions with prompts to the user. Testing also involved identifying the behavior of the system in these alternative scenarios.

5.1 Test Objective: system should run with the command imgreg.py

Test run: In the directory of the file, on the Anaconda PowerShell Prompt command line:

```
>> python imgreg.py
```

Test result: Graphical User Interface displayed as in Figure 2.

5.2 Test objective: The system should display help to run the program with the command python imgreg.py -h

Test run: In the directory of the file, on the Anaconda PowerShell Prompt command line:

```
>python imgreg.py -h
```

Result: Figure 1

5.3 Test Objective: The system should allow a user to choose two images.

Test run: In the directory of the file, on the Anaconda PowerShell Prompt command line:

```
>> python imgreg.py
```

Select image 1 and Select image 2 buttons were used to choose two files Test results: Images were rendered in the GUI when image files were selected.

Alternate test run: A file that is not an image was selected.

Alternate test result: A warning/alert modal is displayed.

5.4 Test Objective: The system should allow a user to perform image registration on the target image using the source image automatically and display the resulting image.

Test run: In the directory of the file, on the Anaconda PowerShell Prompt command line:

```
>> python imgreg.py
```

Two images were selected and Automatic align was clicked.

Test results: Figure 5

Alternate test run: Images of two different objects were selected and then the Automatic align was chosen.

Test results: The aligned image is not as expected indicating that the objects in the source and target image need to be the same for the best outcome. See Figure 11.

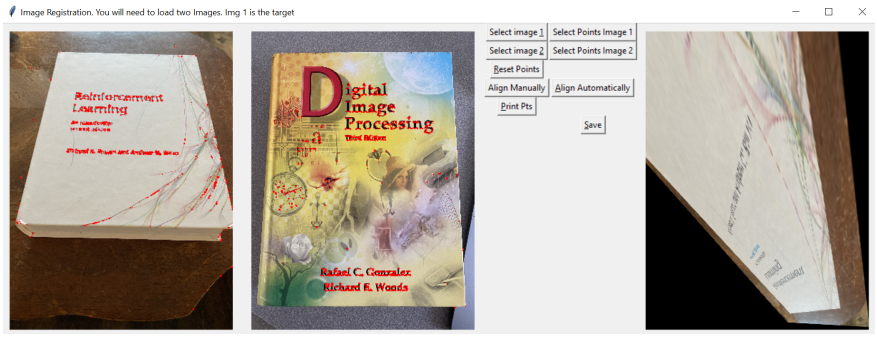

Figure 11: Images do not contain the same object resulting in less than ideal alignment of target image.

5.5 Test Objective: The system should allow a user to perform image registration on the target image using the source image automatically and display the resulting image.

Test run: See section 5.9

Test result: successfully met requirement

5.6 Test Objective: The system should allow a user to perform image registration on the target image using the source image manually and display the resulting image.

Test run: See section 5.9

Test result: successfully met requirement

5.7 Test Objective: The system should allow a user to chose to save a transformed image using the key entry s and prompt the user for the name to save a file as.

Test run: The test was conducted after both a manual and an automatic alignment had been performed followed by the save button or the "s" key.

Test results: In these four tests, the result was the same. There was a slight delay in using the "s" key to save to view the saved files in the directory. A simple file dialog modal was rendered of the local file system. The user was allowed to chose the location and name of the image in the right canvas to be saved. Inspection of the directory confirmed that the image was saved appropriately.

Test run: The test was conducted before alignment had been performed followed by the save button or the "s" key.

Test results: A warning modal is presented to the user, "There is no new image currently."

5.8 Test Objective: The system should terminate when the user presses the "q" on the keyboard.

Test run:

```
>python imgreg.py
```

followed by "q" on the keyboard.

Test result: The program terminated.

5.9 Test Objective: The system should perform as expected.

Test run: Two images were uploaded of the same object, with the object at different perspectives. The automatic alignment and the manual alignment options were compared.

Test results: The result of automatic and manual alignment are visually very close when using the same source and target images. Although these results are dependent, in the case of manual alignment, on the choice of points Figures 12 - 15

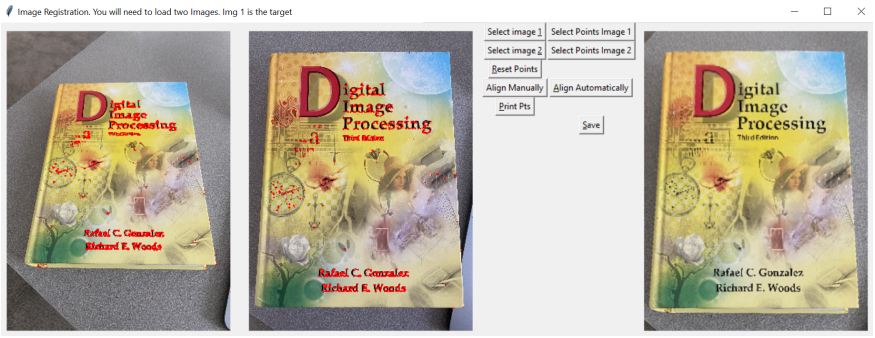

Figure 12: Image registration using automatic alignment and key points at the corners in both source and target images..

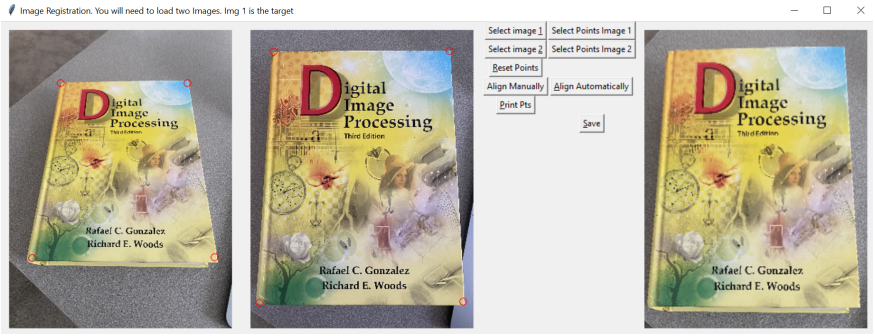

Figure 13: Image registration using manual alignment and key points at the corners in both source and target images.

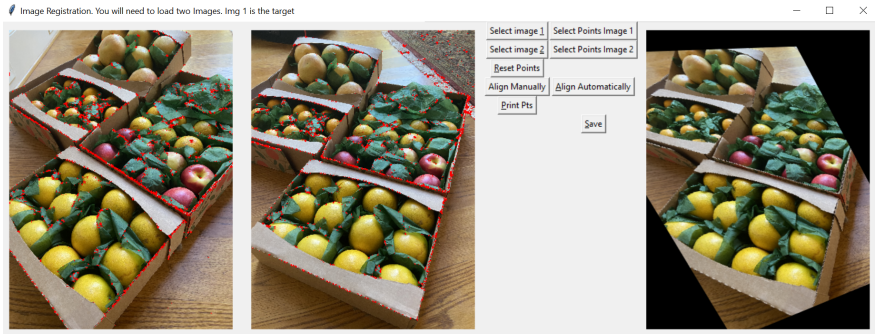

Figure 14: Image registration using automatic alignment and key points at the corners in both source and target images..

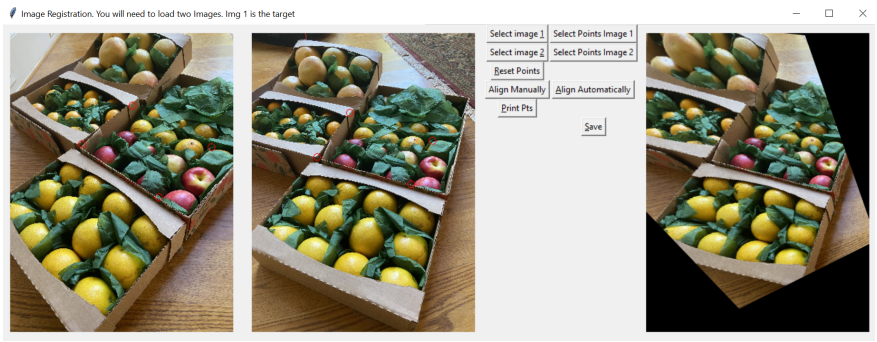

Figure 15: Image registration using manual alignment and key points at the corners in both source and target images.

Test results: The result of manual alignment when using the same source and target images but different choices of key points. The choice of the small number of points, four, compared to the Harris Corner Detector which generates many points, creates a wide range of results. 16 - 18

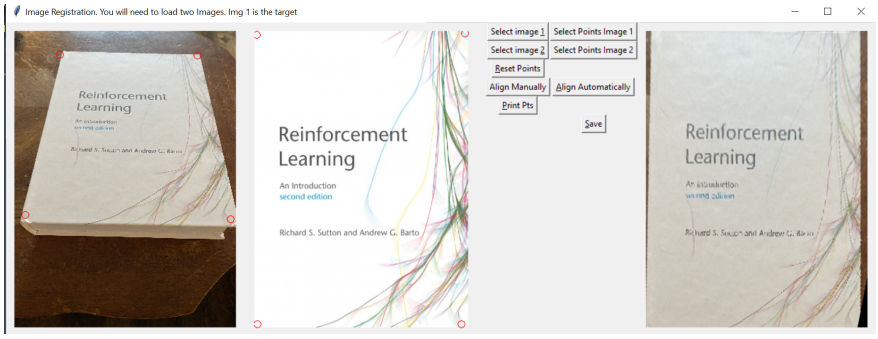

Figure 16: Image registration using manual alignment and key points at four corners.

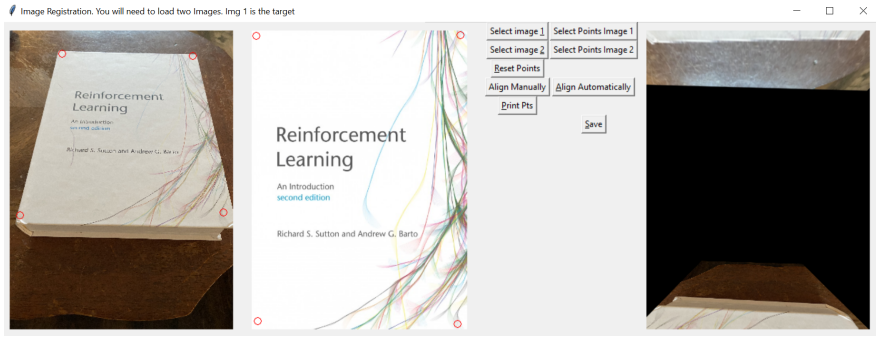

Figure 17: Image registration using manual alignment and key points not exactly at corners, with poor result..

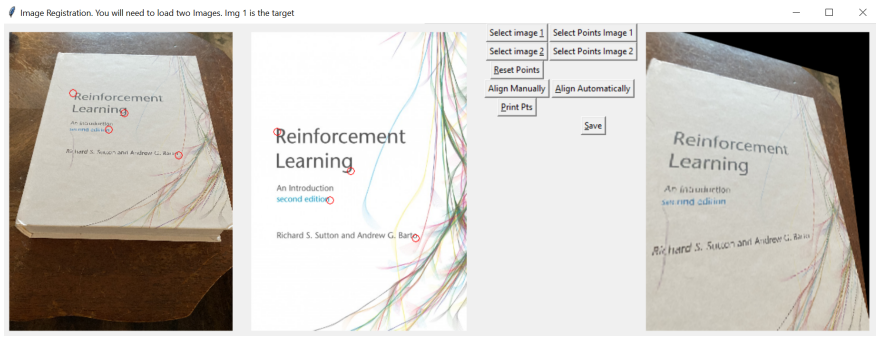

Figure 18: Image registration using manual alignment and key points within the book along text corners.