Student Developer: Raghavender Sahdev
Mentors: Dr. Dimiter Prodnov (INCF Belgian Node), Dr. Daniel Sage (EPFL Switzerland)
Working Period: June – August 2015
Public Repository: https://github.com/incfbelgiannode/TimeLapseReg
Project Website: http://neuroinformatics.be/timelapsereg-presented-at-the-imagej-users-and-developers-conference-2015/

WEEK 1,2 (JUNE 1 - 14)

In the initial week a few simple Image J plugins were developed to get an idea about creating imageJ plugins and deploying them as a jar into the ImageJ software. A simple plugin was developed which adds 2 images and put them in a stack. The idea behind developing such small plugins was to get use to the Image J, ImageJ APIs and libraries were studied in the initial week.

Later, the aims, design and development of the plugin were discussed
We here aim to develop a plugin for drift correction of video sequence in time-lapse microscopy. Initial proposal for the plugin identifies the following tasks:

- We focus on Context Ca+ imaging community.
- They need to apply a drift correction to very long sequences of noisy multichannel images
- They have identified StackReg/TurboReg as the most accurate software package
- They require an efficient tool running on a mid-range machine (e.g laptop)
- A nice friendly user interface is required

'They' here refers to the biologists, potential users of our plugin and other medical experts who would use this plugin.

Relevant Plugins
Following 2 plugins exist which we are relevant to our plugin
TurboReg – Aligns 2 images
StackReg – Does recursive alignment
We here want to develop a plugin which uses TurboReg as the computational engine and has features that are not available in StackReg.

TurboReg/ StackReg of Philippe Thevenaz

- Java plugins of ImageJ
- TurboReg: Manual or automatic alignment of 2 images
- StackReg: Recursive (automatic) alignment of 2 consecutive frames of a sequence
- TurboReg contains the engine (computational part of the registration). StackReg uses TurboReg
- TurboReg is not modular and not maintainable
- The geometric transformation is characterized by two sets of points (not by a affine matrix).

Practical Limitations of StackReg

- No choice for the reference image (the reference image is the previous image)
- Require that the complete sequence fit in RAM -> impossible to process long sequence
- Do not allow to compute the transformation on one channel and to apply it on other channels
- No time regularization in case of registration
- Do not prevent large transformation (we know we have a small drift)
- No pre-processing to denoise the noisy image

In the project we do not change/alter the way TurboReg or StackReg work; we build our plugin using the existing TurboReg and StackReg.

Proof of concept for a rigid body transformation

- Call TurboReg
- Referebce Image: provided as an input image
- Register 2-by-2 images, making a loop for the sequence

```
TurboReg_ reg = new TurboReg_();
ref.setTitle("reference");
ref.show();
int nx = ref.getWidth();
int ny = ref.getHeight();
String size = " 0 0 " + (nx-1)  + " " + (ny-1) + " ";
String mark1 = " " + (nx/2) + " " + (ny/2) + " ";
String mark2 = " " + (0)    + " " + (ny/2) + " ";
String mark3 = " " + (nx-1) + " " + (ny/2) + " ";
String sref = " -window reference" + size;
ImageStack stack = new ImageStack(nx, ny);
for(int i=0; i<filenames.size(); i++) {
String name = " -file \"" + filenames.get(i) + "\"" + size;
      IJ.log("Register " + filenames.get(i));
      reg.run("-align" + sref + name + "-rigidBody " + mark1 + mark1 + mark2 + mark2 +
      mark3 + mark3 + "-hideOutput");
      double spts[][] = reg.getSourcePoints();
      double tpts[][] = reg.getTargetPoints();
      IJ.log(" Translation (" + (tpts[0][0] - spts[0][0]) + "," + (tpts[0][1] -
      spts[0][1]) + ") ");
      double sangle = Math.atan2(spts[2][1] - spts[1][1], spts[2][0] - spts[1][0]);
      double tangle = Math.atan2(tpts[2][1] - tpts[1][1], tpts[2][0] - tpts[1][0]);
      IJ.log(" Angle " + (180.0*(tangle-sangle)/Math.PI));
      ImagePlus imp = reg.getTransformedImage();
      stack.addSlice("", imp.getProcessor());
}
new ImagePlus("Aligned ", stack).show();
```

Strategies for choosing the reference image

The reference image is application-dependent

- An image provided by the user(e.g. a blank microscopy image)
- The previous image (StackReg's strategy)
- The average image of sequence
- A moving average around the target
- The brightest or the darkest image
- The first image
- Average of all the images
- ……..etc.

Drift Calculation

1. Load the target frame (image of the sequence)
2. Check the contents of the frame. Is this frame registrable?
3. If necessary, denoise the frame
4. Compute a new reference image if necessary
5. Compute the transformation (TurboReg call)
6. Exclude the non-realistic transformation
7. Store (CSV/XML/JSON) the human-readable transformation (angle, translation)
8. Store (CSV/XML/JSON) the computer-readable transformation (2 sets of points)
9. Go to 1 until the end of the sequence
10. Check the global coherence of the drift (regularization)
11. Provide a report to the user (plot of the drift, quality of registration)

Apply the transformation to multiple channels image

- Load the transformation file
- Recursively, apply it for all images and all channels

WEEK 3,4 (JUNE 15-28)

After discussing the aims and exact requirements of the plugin, our initial focus was the following set of tasks:

1. Build a plugin to align the input stack of images and compute the transformations and store them in a csv / JSON file
2. Build another Plugin to transform the stack of images as per the transformations computed in the alignment step
3. Develop a plugin for simulation of the plugin to test if the above 2 plugins perform correctly or not.

So we then focused on developing the above mentioned 3 plugins. We built the three plugins as stated above.

TimeLapseRegAlign plugin – this plugin initially input the stack of images and called/invoked the famous '*turboreg*' plugin to align the image stack and compute the transformations and store the transformations in a csv (comma separated) file. We did not go with storing the

transformations in a JSON format as csv was simple enough and fulfilled our current requirements.

TimeLapseTransform plugin – this plugin's basic aim was to transform the set of images in the stack and align each of the images as per the transformations computed by the TimeLapseAlign plugin. So this plugin read the transformations from the csv file generated by the TimeLapseAlign plugin. Using those transformations it applied them to each of the images and the resultant was an aligned stack.

Simulation Plugin – this plugin was developed to simulate the previous 2 plugins and check if our proposed system is performing well. We used this plugin to simulate a sample image and made it undergo certain known transformations and checked if we get the computed transformations same as that being initially input to the sample image. This plugin also gave us the limits upto which our proposed system could tolerate transformation. Example for a sample image our system performed well in a scenario where we had transformation upto 50 pixels translation in x and 40 pixels in y and say a rotation upto 10 degrees.

WEEK 5,6,7 (JUNE 29 – JULY 17)

Before moving further we had to ensure that the simulation plugin was performing well, so we initially focused on improving the simulation plugin. We also wanted to know the transformation limits upto which the plugin would perform well. This meant we needed to know the translation and rotational limits upto which we get a good aligned stack.
We considered taking a simple image and subjecting this image to predefined known transformations (translations and rotations). We applied these known transformations to the image and saved the frames that we got after applying each of the transformation. The transformations were applied as follows:

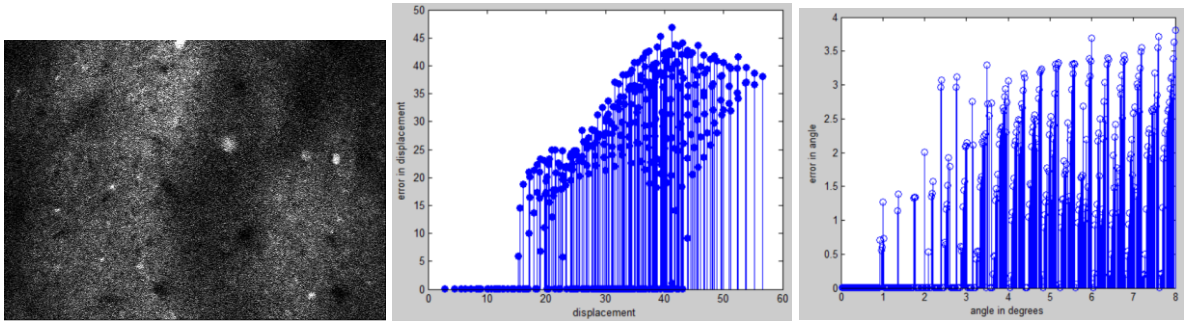
1. Translation of 1px,2px,3px……40 px in x direction only
2. Translation of 1px,2px,3px……40 px in y direction only
3. Rotation of 1degree, 2 degrees, 3degrees ……10 degrees
4. Translation in x and y direction both
5. Translation in x, y direction and rotation also

It was observed that our simulation does perform well up to a certain limit which is good enough for the medical imaging data that we focused on. A rough estimate of these values were up to 15 pixels in x and y with a rotation of 5 degrees.
However these estimates depended on the image we chose. If we chose an image which was much simpler such as a simple rectangle drawn in paint, we even had our plugin perform well up to 100 pixels translation in x and y direction. So this transformation limit really depended on the input image. But for the most difficult image that we had - a really noisy one; we got 15 pixels in x and y and 3-4 degrees rotational limit.

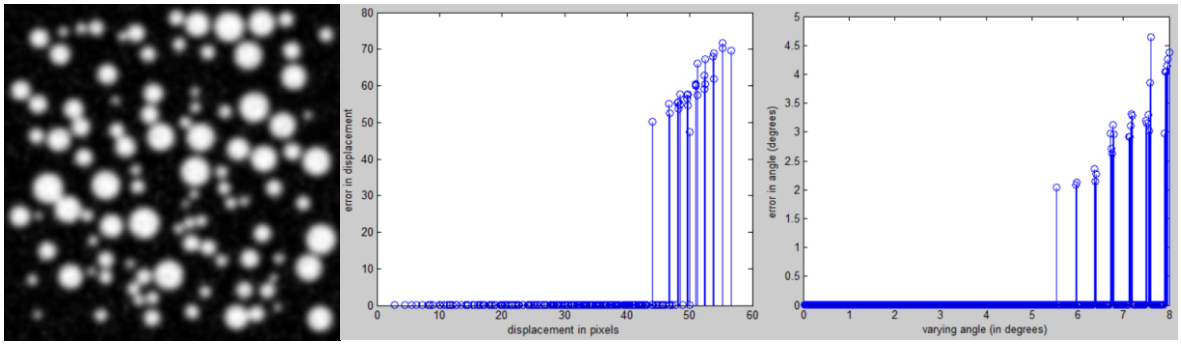
We here performed a validation analysis of our simulation plugin using 3 different cases in order of decreasing difficulty

1. Case1: A noisy image – difficult to compute
2. Case2: A less noisy image easier to compute
3. Case3: A noise free very easy image to compute

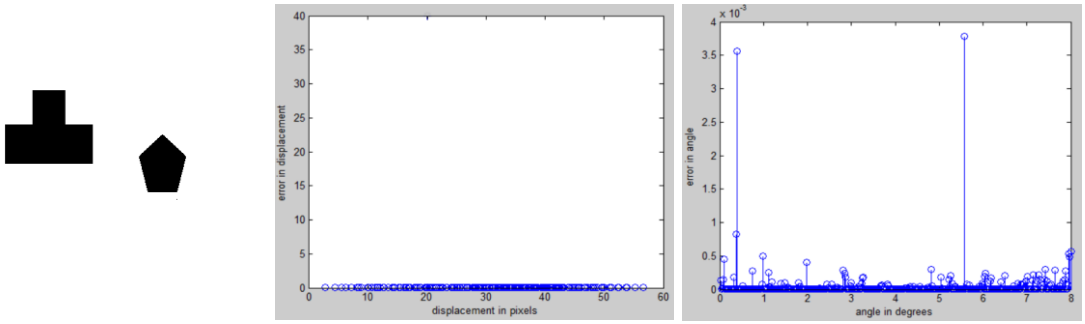
Following graphs illustrate the variation in error



Case 1



Case2



Case3

We here observe that the minimum error is observed in case3 as its not noisy and maximum error is observed in case 1 which is a noisy image.

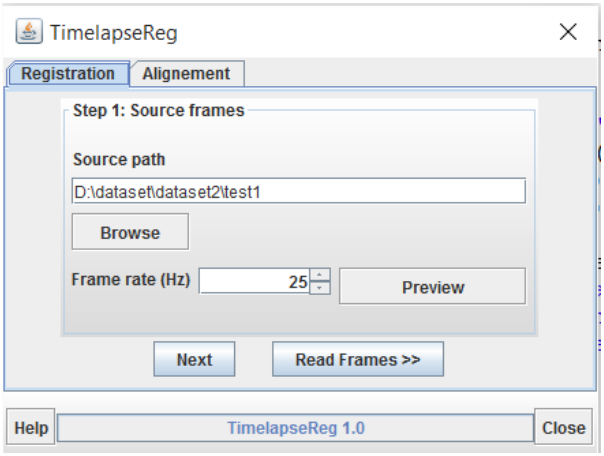
Once we had the simulation plugin working correctly, we knew that the limits under which our system / plugin would perform perfectly well. We then focused on adding the following features to the plugin:

1. Giving the user the option to input the stack of images from a specified folder path
2. Giving the option to the user to choose a reference image to which the entire stack should be aligned
3. Discarding certain bad frames based on a specific criteria
4. Drawing a trajectory on the reference image to give the user some dynamic feedback
5. Computing transformations from one channel and applying it to another channel

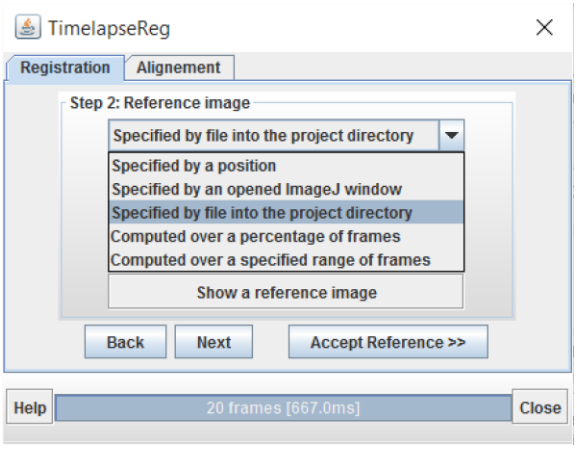
6. Save the transformations so that they do not need to be computed again and again and can be simply used by the user to apply

WEEK 8,9,10 (JULY 18 – AUGUST 9)

To implement the features discussed previously we decided to build the final version of the plugin as a User Friendly Graphical User Interface (GUI) using the java's swing. We put everything that we had built so far into a GUI in order to incorporate the functionalities discussed above.
Following figures demonstrate the plugins:



In Step 1 we give the user the functionality to input the stack of images. The user does so by choosing the path of the folder which contains all the frames of which we want to compute the transformations

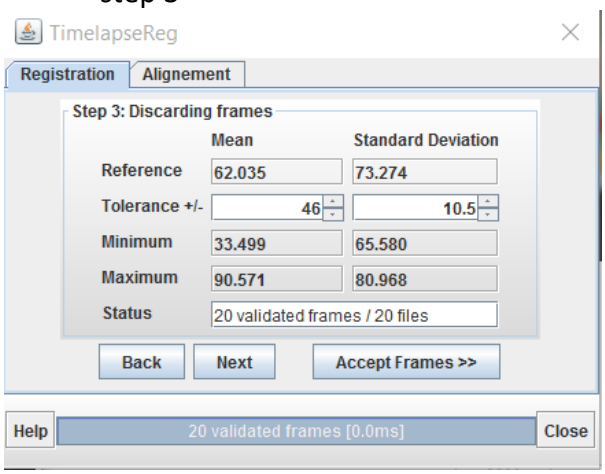


In Step 2 the user has a number of options to choose the reference image as can be seen above

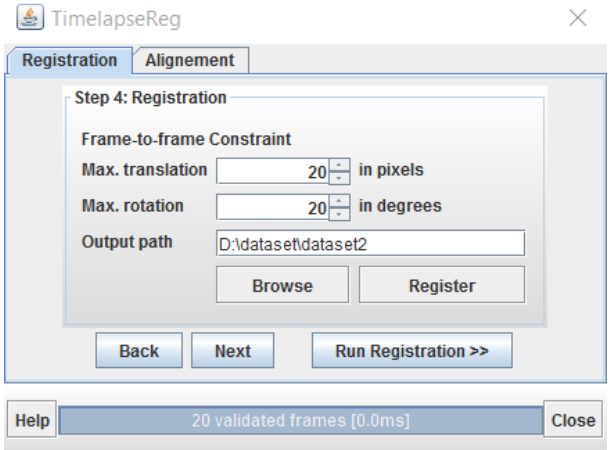
- Specified by a position – this allows the user to specify an index number of the frame which should be the reference image
- Specified by an opened ImageJ window – this allows the user to specify a particular image window which is opened in Image J

- Specified by a file into the project directory – this allows the user to specify a path of an image file which would act as the reference image for the entire input stack.
- Computed over a percentage of frames – this allows the user to choose the average, minimum, maximum of a percentage of the entire stack of images. As an example the user could say reference image to be the average of the first 20% of the frames or maximum over the first 50 % of the frames
- Computed over a specified range of frames – this gives the user an option to specify an exact frame number upper and lower limit and then the user can compute the average, minimum or maximum over that specified range.

  After this step the user can show the reference image and see if he/she likes it, if not choose a different option and choose another image. After finalizing the reference image the user should click on the "*Accept Reference>>*" button, which takes us to step 3

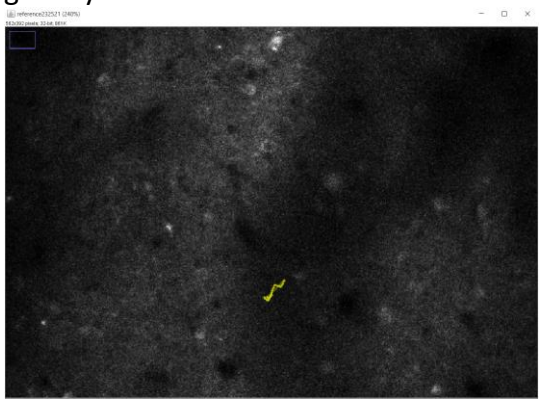


In step 3, we implemented a discarding mechanism to discard a set of frames which may be present due to some noise. We here discard based on the Mean and standard deviation of the frames. The discarded frames essentially get rid of any noisy images which may lead to a bad alignment so we get rid of those



In step 4, we set the translation and rotational limit upto which we would consider a specific transformation as a valid one. All other transformations beyond the ones specified here are considered as bad transformations and not included here.

After this step the user should click on Run registration which starts to compute the transformations and stores them in a csv file. The progress bar in this step shows the progress of the transformation. And while the transformations are being computed the user gets dynamic feedback as shown below –

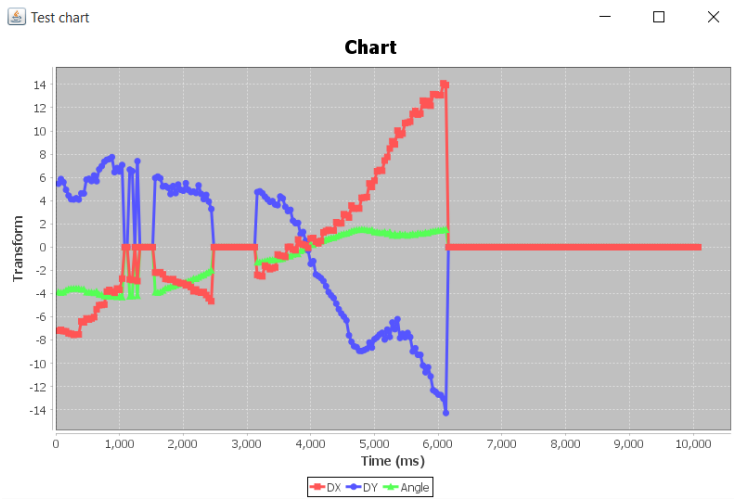


The above trajectory (yellow) in the reference image is being dynamically being drawn here. As computing the transformations takes a lot of time, we cannot have user sitting idle. Hence here we show some dynamic feedback to the user.

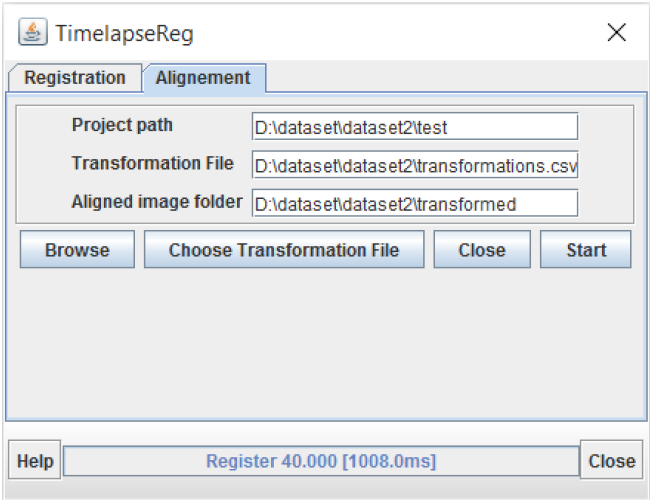
D:\dataset\dataset

#	Time (ms)	Filename	Mean	Sdtdev	Content	Transforma...	RMSE
143	5720.000	img0142.tif	4.550	3.618	OK	11.501, -9....	
144	5760.000	img0143.tif	4.481	3.547	OK	12.609, -10...	
145	5800.000	img0144.tif	4.481	3.650	OK	12.207, -10...	
146	5840.000	img0145.tif	4.224	3.464	OK	12.574, -10...	
147	5880.000	img0146.tif	4.176	3.403	OK	12.169, -11...	
148	5920.000	img0147.tif	4.305	3.374	OK	13.168, -12...	
149	5960.000	img0148.tif	4.223	3.365	OK	13.147, -12...	
150	6000.000	img0149.tif	4.233	3.276	OK	13.061, -12...	
151	6040.000	img0150.tif	4.262	3.242	OK	13.069, -12...	
152	6080.000	img0151.tif	4.289	3.352	OK	14.089, -13...	
153	6120.000	img0152.tif	4.310	3.225	OK	13.937, -14...	
154	6160.000	img0153.tif	4.274	3.170	Invalid tran...	0.000, 0.00...	
155	6200.000	img0154.tif	4.259	3.239	Invalid tran...	0.000, 0.00...	
156	6240.000	img0155.tif	4.274	3.148	Invalid tran...	0.000, 0.00...	
157	6280.000	img0156.tif	4.266	3.283	Invalid tran...	0.000, 0.00...	
158	6320.000	img0157.tif	4.227	3.173	Invalid tran...	0.000, 0.00...	
159	6360.000	img0158.tif	4.299	3.169	Invalid tran...	0.000, 0.00...	
160	6400.000	img0159.tif	4.414	3.243	Invalid tran...	0.000, 0.00...	
161	6440.000	img0160.tif	4.474	3.136	Invalid tran...	0.000, 0.00...	
162	6480.000	img0161.tif	4.482	3.278	Invalid tran...	0.000, 0.00...	
163	6520.000	img0162.tif	4.492	3.178	Invalid tran...	0.000, 0.00...	
164	6560.000	img0163.tif	4.427	3.218	Invalid tran...	0.000, 0.00...	
165	6600.000	img0164.tif	4.371	3.343	Invalid tran...	0.000, 0.00...	
166	6640.000	img0165.tif	4.434	3.450	Invalid tran...	0.000, 0.00...	
167	6680.000	img0166.tif	4.443	3.392	Invalid tran...	0.000, 0.00...	
168	6720.000	img0167.tif	4.524	3.450	Invalid tran...	0.000, 0.00...	
169	6760.000	img0168.tif	4.551	3.516	Invalid tran...	0.000, 0.00...	
170	6800.000	img0169.tif	5.288	3.854	Invalid stdev	0.000, 0.00...	
171	6840.000	img0170.tif	5.845	4.207	Invalid stdev	0.000, 0.00...	
172	6880.000	img0171.tif	7.014	5.001	Invalid stdev	0.000, 0.00...	

The above table is also being dynamically being updated. The green rows indicate a good transformation, the yellow ones indicate an invalid transformation and the red ones indicate the discarded transformations in the pre-processing step. This is to show the user some feedback while the registration is running.

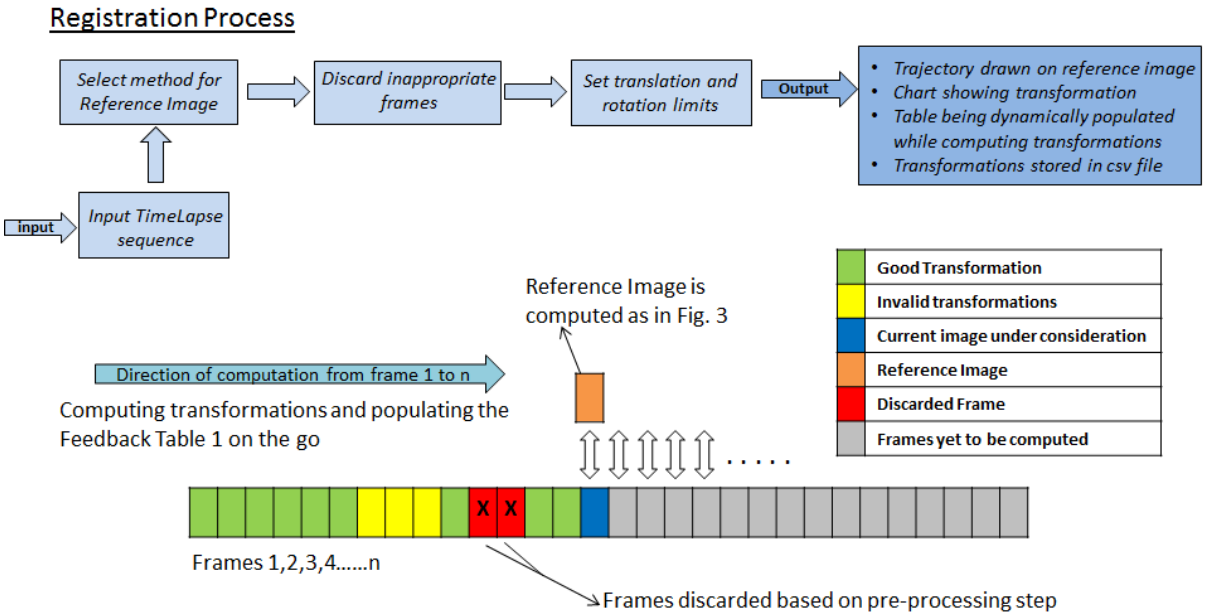


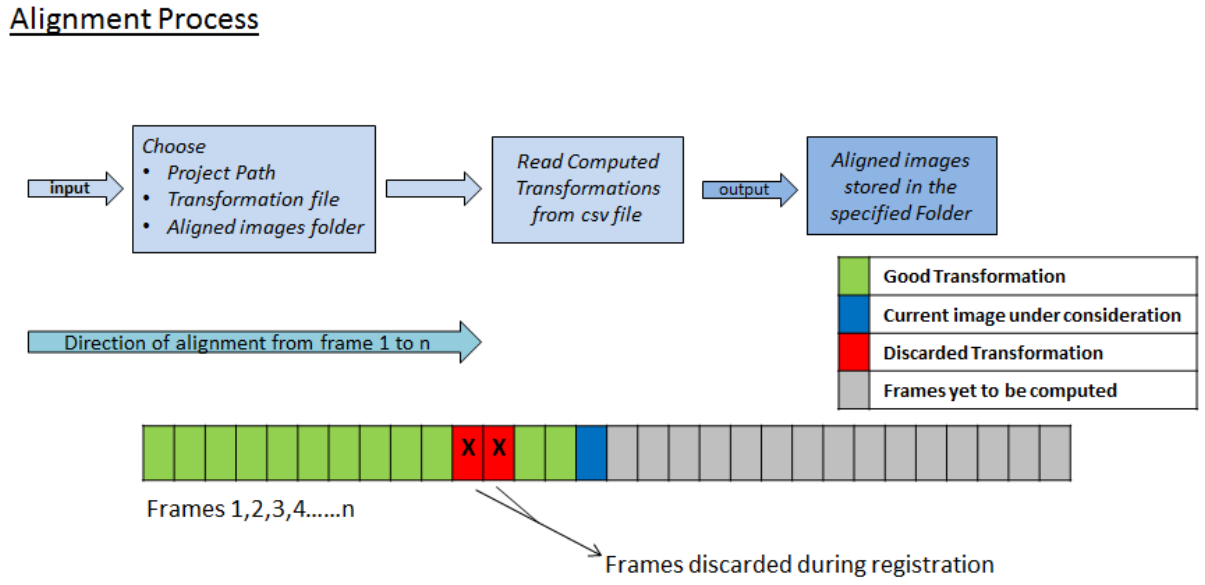
This chart is generated after the transformation is run. This shows the user a graphical representation of the computed transformations (translations in x and y and rotations). After this step the user can use the computed transformations and apply them to a specific channel. So assume he / she computes the transformations on the red channel but wants to apply them to a different channel he can do that in the alignment step shown below in the figure:



WEEK 11,12 (AUGUST 10 – AUGUST 25)

In the last weeks we focused on finishing up the project and discussed about the poster contents to be presented at the ImageJ Users and Developers Conference 2015 in Madison in September. We made a few changes to the code and fixed some bugs in the code.
The following figures illustrate the basic overview of the plugin-





Conclusion

In this project we developed a plugin to compute the transformations from a stack of images and then we used those transformations to align the stack of images. We get good results on the 3 datasets that we tested. Further extension of the project would include testing our plugin with more challenging datasets.

We presented our work at the following 2 places:

- poster at the imageJ Users and developers conference 2015 in Madison, Wisconsin, USA during September 5-6, 2015
- plugin demo at the INCF Booth at Society for Neuroscience Conference 2015 in Chicago, Illinois, USA on October 17-21, 2015.