DLA Report 3

Alexey Yermakov
5 October 2021

1 Description of what was done

1.1 Summary

This report is large, so I've decided to include a summary of what I have completed the past two weeks.

First, I followed Moisy's advice on generating a pattern appropriate for our camera's resolution. I used both objective and subjective measurements: eyeballing the pixels and comparing pattern generation outputs. I then performed research on various lighting solutions for the water table, which we'll be keeping indoors. I believe that an LED panel would be the best solution given its accessibility (in terms of cost and purchasing), portability, and brightness. I then took more outdoor pictures of still and moving water. I selected the two best pictures and processed them in software. I resolved many problems I had in my last report and was able to reconstruct the water surface height. I identified some concerns that I will pursue in my next report, including surface height noise and variable image lighting intensity.

1.2 Right After the Group Meeting

I reached out to the Trades Teaching Lab in Duane about registering for their CNC courses via email. I have not gotten a response and will send a follow up E-Mail on October 6th - two weeks after my original email.

1.3 Generating a new pattern

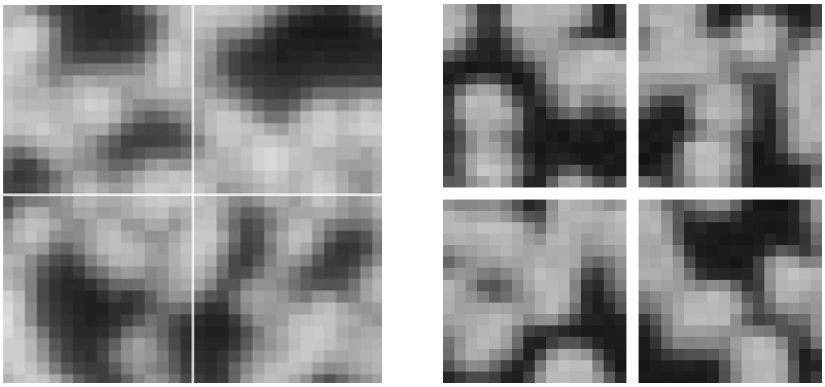
In my last report I was unable to use the APPM printer since I went in during the weekend. Here, I determine the appropriate values for pivmat's `makebospattern` function to generate a new pattern that has dots of size approximately 6 pixels in our camera which will be approximately 103cm away from the pattern (the pattern-camera height from the last time I came into the lab). The camera's resolution in a photo I took last time is 3648×2432 pixels (an aspect ratio of $3:2$). Note that this resolution corresponds to a Medium resolution for our camera - the Canon EOS 70D. To find the proper pattern, I generated different patterns in MATLAB, placed them in a Gimp (similar to Photoshop) workspace of size 3648×2432 pixels, and re-scaled the pattern (which is generated as 210x297mm) to fit in the workspace. I then analyzed the dot density in a 16×16 pixel grid. The values for the pattern I generated containing around 5 dots per 16^2 pixel grid were N=320,000 and D=0.35 (320,000 dots of diameter 0.35mm). I then used the output of the function to match

the default outputs as closely as possible after changing the source code of the function to assume we are using a 3648×2432 resolution camera:

```
% Default output on a 1280x1024 camera
>> makebospattern(50000,1,'w','myfig')

PID pattern parameters:
  50000 white particles on a 210x297 mm (A4) paper
  Particle diameter = 1 mm.
  0.80167 particles / mm^2
  filled surface ratio = 0.62963

Assuming a 1280x1024 camera, with 210 mm = 1024 pixels:
  particle diameter = 4.8762 pixels
  * Warning: particles too big *
  0.033717 particles / pixel^2

Particles per interrogation window:
  1.2138 particles / 6^2-window
  2.1579 particles / 8^2-window
  4.8553 particles / 12^2-window
  8.6317 particles / 16^2-window
  34.5267 particles / 32^2-window
  138.1068 particles / 64^2-window
  Optimal window size: 12.2^2
  Closest window size: 12^2

% Custom output on a 3648x2432 camera
>> makebospattern(335000,0.42,'w','myfig')

PID pattern parameters:
  335000 white particles on a 210x297 mm (A4) paper
  Particle diameter = 0.42 mm.
  5.3712 particles / mm^2
  filled surface ratio = 0.74414

Assuming a 3648x2432 camera, with 210 mm = 2432 pixels:
  particle diameter = 4.864 pixels
  * Warning: particles too big *
  0.033375 particles / pixel^2

Particles per interrogation window:
  1.2015 particles / 6^2-window
  2.136 particles / 8^2-window
  4.806 particles / 12^2-window
  8.544 particles / 16^2-window
```

```
34.176 particles / 32^2-window
136.704 particles / 64^2-window
Optimal window size: 12.2^2
Closest window size: 12^2
```

I tweaked the values in the 3648x2432 camera output to match the optimal window size (12.2) and particle diameter (≈ 4.8) of the recommended output on a 1280x1024 camera. The pattern I found to match these values the closest was generated using N=335,000 and D=0.42 as input parameters.



(a) F. Moisy's pattern

(b) N=335,000 D=0.42 Photo

(c) N=320,000 D=0.35 Software

(d) N=335,000 D=0.42 Software

Figure 1: Comparison of patterns (4 pictures of 4 windows of 16^2 pixels)

See Figure 2 for images of the various patterns I generated. Note that the top row has lower contrast, this is likely because those photos came from a camera whereas the bottom row is from software alone. I printed off both patterns to use in my experimentation.

1.4 Light box research

In my last report I mentioned that taking pictures indoors results in poor image quality. Sunlight resolved the issue, but we can't rely on having the water table outside for experimentation. As a result, we need to consider indoor lighting for the water table.

Surfing the internet, I found that someone used a "matte lamp" to light their experiment, though they then mention they had "colossal hardware failure". Frankly, a matte lamp doesn't seem like it would suit us very well since it doesn't provide uniform lighting. Another article mentions the use of an "LED panel", which appears to be accessible and a good size to fit under the water table to light the pattern. The price range is around $100, as seen by this 2×2 foot LED panel costing around $70 at The Home Depot. Note that the maximum lumen output ("a measure of the total quantity of visible light emitted by a source per unit of time") is 4200 lumens. Compare this to 2400 lumens of the light box created with an LED strip in the lab (measure by my phone's light sensor). (Note: The Home Depot sells troffers at a similar price point with more lumens, though I have concerns about the uniformity of their light output).

I mentioned we could use soft box lights (those used by photography studios), but after learning about LED panels they seem impractical due to their size.

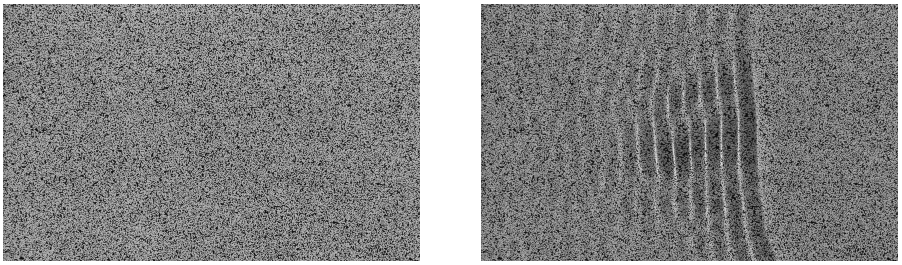
I suggest we use a single LED panel as a light underneath the water table to illuminate the pattern. An LED panel is bright, produces uniform light, and has a small form factor - making it easier to attach to the water table from below. It is also relatively inexpensive. Attaching it from below also removes the possibility of introducing glare on the water surface.

1.5 Experimentation

After the group meeting on September 28th, I went ahead and took another set of photos with the new patterns underneath the glass tray partially filled with water. To alleviate the problems with the photos not lining up last time, I moved the tray to generate a plane wave first. This allowed me to take photos of the wave and then the still water afterwards without having to move the tray in between. This removes the possibility of moving the tray during the two photos, resulting in inconsistent surface height reconstruction. What I did not account for, however, is the possibility of moving the camera as I pressed the button to take the photo. I didn't realize this during the experiment since the shift in photos is minute, but it is noticeable upon inspection on a computer. Another problem I had was leaves falling into the tray, since I was taking photos outside and sometimes wind would blow them into the water.

None of these problems will occur in the final water table experiment. Debris won't fall into the water and no wind will be blowing since the table will be set up indoors. We won't have variable lighting conditions because we will have control over lighting. We won't have to worry about the camera moving since we'll take pictures using a computer. These problems just make it more difficult to test the software before we have the water table set up.

After performing the experiment and looking at the photos in software, I will be using the following two in OpenPIV and pivmat:



(a) Still Water (b) Moving Water

Figure 2: Final photos from experiment

I passed the images through a grayscale filter since we can safely assume the resulting images should have no color. I also cropped the photos significantly because a tiny bit of a leaf fell into the water tray and I didn't notice until I was editing the photos. The leaf bit left a shadow on the pattern and I didn't want the software to be affected by it. I should also note that although I moved the camera slightly between these two photos, the pattern overlapped with each other nearly perfectly, so I am not worried about any human error in this regard. You can verify this yourself by seeing how the pattern looks at the corners of the two photos (they should similar at each corner). I performed this analysis more accurately by placing the two photos over each other and making the top photo be 50% transparent in software. The pattern these two photos use is N=335,000 and D=0.42. The photos with N=320,000 and D=0.35 had too much of a shift between photos for them to be usable.

Note also that since the light source (the sun) is from above, the lighting intensity on the pattern with moving water is uneven. This should not occur in the controlled water table environment where the lighting will be below the pattern.

I reconstruct the surface height using the images in Figure 2. My primary concern with *OpenPIV* from when I last used it was the use of the dt parameter in the function `extended_search_area_piv`. This function generates a velocity field. It turns out, this parameter is only used in the output of the velocity field by dividing each component of each vector in the displacement field by dt. Thus, setting dt to 1 results in a displacement field, which is what we want as the input to *pivmat*'s `surfheight` function when creating the surface height.

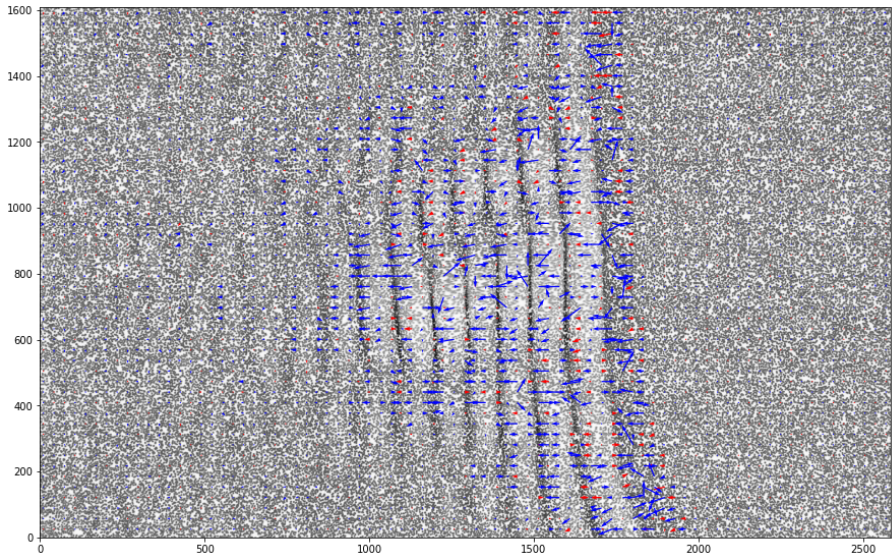


Figure 3: Superimposed Displacement Field (axes are in pixels)

The *OpenPIV* tutorial I used as a foundation for my code contains various filtering and smoothing functions that are worth mentioning. The first is `validation.sig2noise_val`, which "Eliminate[s] spurious vectors from cross-correlation signal to noise ratio". It is possible (and recommended) to view the signal to noise ratio values of each displacement vector in a histogram before deciding the value for the `threshold` parameter in `validation.sig2noise_val`.

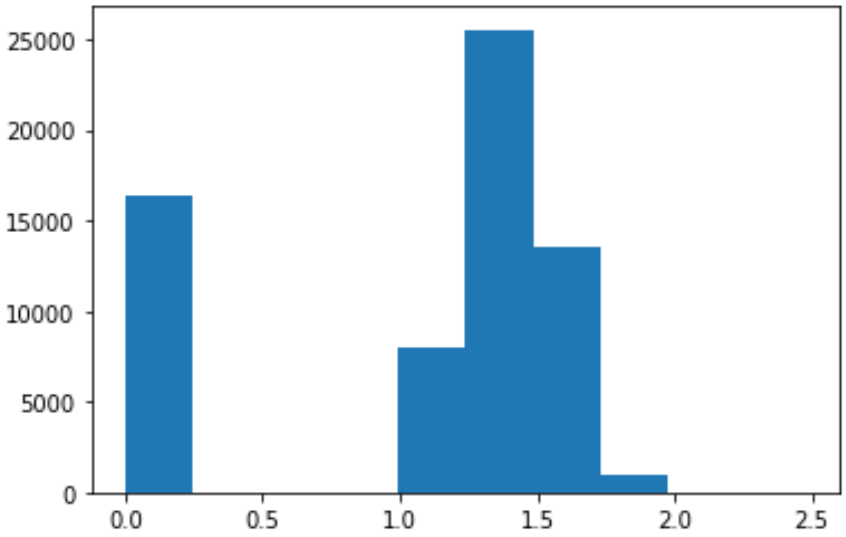


Figure 4: Signal to noise ratio of the displacement field. Bottom axis is the signal to noise ratio. Side axis is the number of vectors.

As you can see, a good value for `threshold` would be any value between 0.3 and 0.9. I chose 0.9 just in case there were any values between 0.3 and 0.9 that don't appear on the histogram and could negatively affect the data. `validation.sig2noise_val` goes through and sets the components of each displacement vector whose signal to noise ratio is below `threshold` to *nan*. The function `filters.replace_outliers` then goes through these values and "Replace[s] invalid vectors in an velocity field using an iterative image inpainting algorithm". The previous two functions combined result in a smoothing effect on the final surface height.

Note: The output of creating the displacement field resulted in 1864 vectors out of 64476 as *nan*. After running `validation.sig2noise_val` this value rose to 16380 (22.51% increase). Running `filters.replace_outliers` caused none of the vectors to be *nan*.

Another problem I encountered in my previous report was the `scaling_factor` parameter in the function `scaling.uniform` in *OpenPIV*. This function is used to convert the unit of the displacement field from pixels to mm. My problem came from the fact that the *OpenPIV* tutorial says that `scaling_factor` should be the ratio of microns per pixel of the image taken. I found this ratio by calculating the number of pixels per 150mm (see Figure 5). Using microns per pixel for `scaling_factor` resulted in my image area being approximately 30x20mm. This is wrong since the final photos I used were approximately 221.22x136.10mm (calculated using pixels per mm and the pixel dimensions of Figure 2). After much trial and error, it turns out that `scaling_factor` should be set to pixels per mm. This makes sense since the function `scaling.uniform` just divides

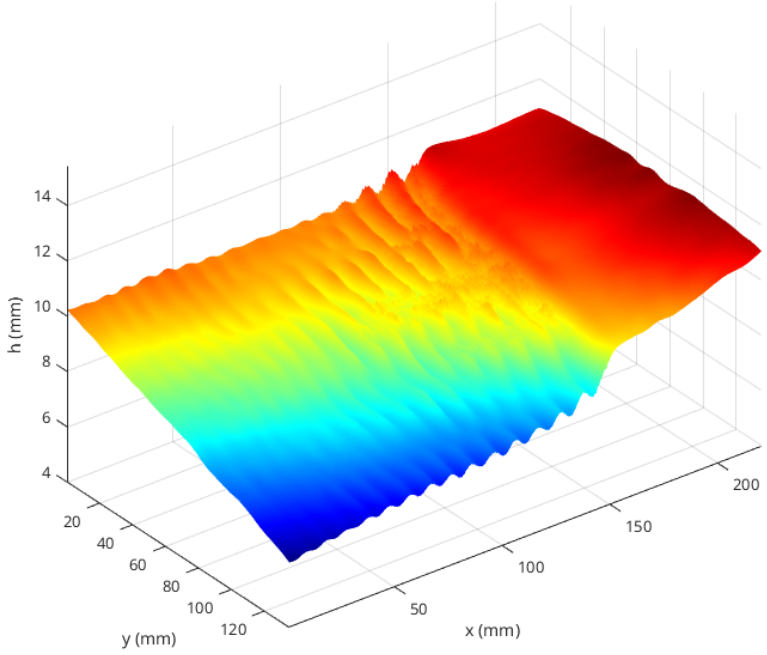
each component of each displacement field vector by `scaling_factor`, converting the units from pixels to mm.

Figure 5: 150mm ruler used to measure pixels per mm

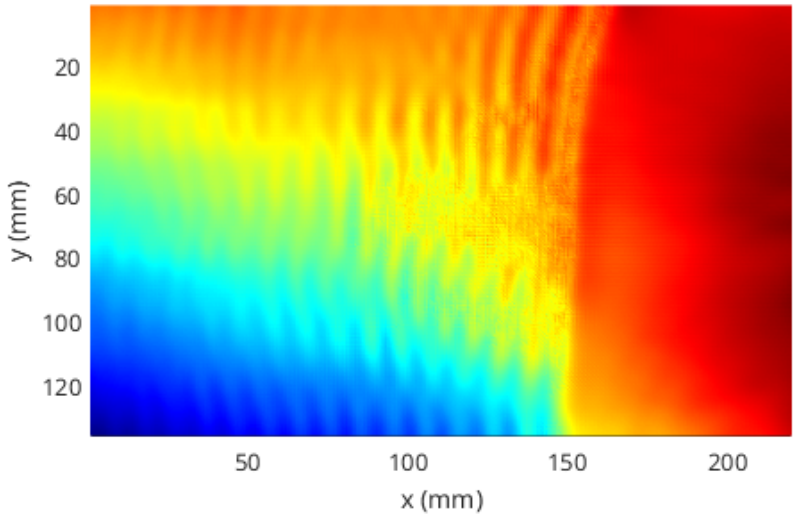
This displacement field can them be used in *pivmat* directly. The only other information I need is the the pattern-surface distance H0 and the pattern-camera distance HC. This was measured by hand. I then ran the following code in MATLAB:

```
dr = loadvec("./fsss_matlab.txt");
h = surfheight(dr, 9.55, 945);
subplot(1,2,1), showf(h, 'Surf')
subplot(1,2,2), showf(h)
```

to get the following images:



(a) 3D Surface



(b) 2D Surface

Two things stand out to me in the resulting surface height:

1. Although post-processing occurred for the surface height, there still appears to be a significant amount of noise around the area containing

x=140mm and y=70mm. This is best seen with Figure 6b. This might be a result of the variable light intensity from the sun due to taking the pictures outside. This hypothesis is supported by the fact that the noise occurs more often in the areas with lower light intensity in the original image.

2. The resulting surface height appears to have a non-zero slope at x=0mm, which I did not expect. This is either a result of human error or the water surface was really just like that. This is best seen with Figure 6a.

I also went back into the lab to take pictures indoors with the light box. I played around with shutter speed (a recommendation by Mark) and the light box resulting in good quality images. I wasn't able to test those pictures much in this report since I spent most of my time generating the pattern and experimenting with the pictures I took outdoors.

2 Description of next steps

Overall, I'm pleased with the progress I made during these past two weeks. I have deciphered many of the problems I encountered in my last report and have a clear path of my next steps.

First, I will discuss with the group before ordering an LED panel for the water table. I believe I can begin taking images using the water table (using the light box for now). Second, I will spend some time tweaking the post-processing functions in *OpenPIV* to see if I can reduce the noise experienced in 6b. Third, I will use software to analyze the indoor photos I took via the same processes outlined here. I hope that since the photos were taken indoors that the noise from uneven lighting will be reduced. Fourth, I need to research how to account for the acryllic/glass between the water and the pattern since *pivmat* doesn't have any parameters for this variable (as far as I'm aware). This research will include looking laminating the patterns I print so that it can be placed above the acrylic/glass. Fifth, I will try to take photos of more minuscule waves (such as a water drop) and see if the software I currently have running can process such small changes in water surface height.

3 Questions

- Is there a way I can laminate paper on campus?
- Am I able to use/run the water table on my own to take pictures?
- What do you think about the water slanting towards the x=0 end of 6a?
- Do you agree with my hypothesis about why there is noise in figure 6b?
- Why, when I am running a MATLAB script, can't I access a file in a sub-directory? The following works as paths when the file is in the same

location as the script: `./file.txt` and `file.txt`. However, when the file is in a sub-directory, neither of these works: `./dir/file.txt` or `dir/file.txt`. I tried adding the sub-directory to the MATLAB PATH and it didn't work. I get the following error:

```
Index exceeds the number of array elements (1).

Error in rdir (line 136)
        pname=name(1:(p-1)); % first pathname of the input string

Error in loadvec (line 236)
    filename=rdir(filename);

Error in FSSS (line 3)
dr = loadvec("OpenPIV/fsss_ns2s_matlab.txt");
```