Water_edge_final tutorial document

Tutorial videos have been made to help explain how to utilise the water_edge_final script and the LabView code to use the load cells. This document will step through what is happening in each video and explain what each of the videos is demonstrating, pointing out its strengths and weaknesses and explaining the cause where possible.

The title of each section is reasonably self-explanatory and matches with the name of a tutorial video.

Function explanations

- StationaryObject_unsuccessful ____ 2
- StationaryObject_successful ____ 3
- SlopingObject_successful ____ 4
- MovingObject_unsuccessful ____ 5
- MovingObject_successful ____ 5

Feature explanations

- Non-straight_tracking ____ 6
- BadContrast ____ 6
- Batch_processing ____ 6
- Enabling_Edge_detection_video ____ 7

Load cell readings

- Setup ____ 8
- Data_record ____ 8

Function explanations

StationaryObject_unsuccessful

Discussion

This experiment required the tracking of water during a wave run up and should be processed with *SlopingObject.* However for demonstration purposes, it has been processed with *StationaryObject*. This has resulted in the failure of the function to detect the actual intersect between the water and object.

This has been caused because *StationaryObject* requires the object of impact to have vertical edges which are defined as 45º or more from the waters edge. As this slope is only 30º, the edges of the water and slope have been not been differentiated so the only vertical edge that is detected is edge of the edge of the screen.

Description of function

In order for the actual height of the run up to be calculated, a ruler was placed on the tank. From this, using the *drawline* function, the number of pixels per unit can be calculated. The zoom capability which is new to MatLab2018b in plots enable for the ruler to be measured with more precision. Note that this function (along with *roipoly()*) allows for the points to be modified after placement unlike *getrect()*. The user input allows for flexibility, however it is inherently inaccurate due to the pixelation of the ruler, undefined point of the *drawline* function and human error.

Much of the video footage area, due to the nature of the set up and ability for focusing, is in many circumstances not needed. By specifying a ROI using *getrect()* it reduces the computational effort needed to process the video and also increases accuracy by removing non-essential features such as lighting, other movement etc.

An UI approximation of the initial waters edge point is selected. This is actually redundant in *Stationary* and *MovingObject*! It was included very early on when the condition of waves coming from either the left of the right were considered and knowing the position of the water relative to the object was important. However as the project evolved, this condition was dropped for the assumption that the wave will always travel from left to right but it was left in because it was relatively harmless and don't fix what isn't broken! In any case it is used in *SlopingObject,* explained later.

To identify the structure, *roipoly()* is used to black out all the area behind the structure. This function provides a lot of flexibility in the kind of structures can be tracked, however the boundaries are often hard to eliminate and results in false edge detections. *getrect()* is called again to enable the edges of the polygonal shape to be further blacked out. This process is highly laborious and, if any substantial effort was put into it, could be automated. This could be a point of work, however in the grand scheme of things, other packages such as OpenCV and better image enhancement would provide greater improvement to the results rather than automation.

StationaryObject_successful

Discussion

In this experiment, the height of the wave as it passes a point is being investigated. It is processed with *StationaryObject* and it produces very good results except at the initial point of water movement (see 1:28).

This the initial point of water movement causes a bad reading because the meniscus is no longer visible. This means that the edge tracking instead locks onto the water line which is on the tank as the most prominent horizontal edge. There are a number of possible solutions to this which I have thought of but not implemented:

- **Wipe the water off after each time:** simple and doesn't involve any change in coding but is tedious and non-methodical.
- **Incorporate the previous results:** if the edge detected moves by more a set amount then the code automatically disregards these results as false detection.
- **Improve the image contrast enhancement:** this has been talked about quite a lot within this report for general improvement in the quality of tracking across different footage.
- **Perform post-processing smoothing:** applying a processing algorithm over the data, for instance one that removes values that change by more than a certain threshold, would be a band aid solution to the problem rather than fundamentally improving the script.

Description of function

Standard set up. However selecting the area behind the structure using *roipoly()* is in this case optional because only a straight structure is needed. This highlights that some of the steps are optional and depends on the quality of the video and type of object being tracked.

SlopingObject_successful

Discussion

This video successfully demonstrates how *SlopingObject* works. The video being processed is the same as the one described in StationaryObject_unsuccessful, however it is processed with *SlopingObject* and the results (see at 2:00) are far more descriptive of the water movement along the structure.

SlopingObject produces significantly better results in this case because it has an extra step at the start of the function which is used to clearly define the difference between the structure and the water. However, the cost of this is that the water is not tracked during the back wash which is why the water level doesn't go below 0 (see 2.42 for video which shows this failing). It also takes substantially longer to process the video compared to *StationaryObject*. This is likely because of the extra processing required on the detected water edge performed from lines 69-78 of *SlopnigObject*.

Description of function

Same as above, however selecting the water intersect becomes critical in its functionality. The edges to the right of this are set to be the structure. From this, in every frame processed, the edges of the structure are subtracted so as the wave crashes onto the structure, only the edges which are water are processed. The right most point is considered to the be the intersect. This step is critical in allowing slopes of less than 45° to be processed, however it presents the backwash of the wave to be tracked as anything to the left of the initial intersect is considered to be non-structural, ie the waterline.

MovingObject_unsuccessful

Discussion

This experiment tests the ability of *MovingObject* in tracking the intersect of water on an object, in this case a tie down, which is freely moving in the water. Although the graph (1:18) produces results which are believable, the video (1:49) reveals that the tracking does not lock onto the tie down very well.

The reason for this poor tracking is likely due to the misidentification of parts of the tie down being a horizontal edge and also the functions inability to track one side of the moving object.

Description of function

During the selection of the ROI, *getrect()* is used to narrow down the field of view on both sides. This feature is unique to *MovingObject* and can't be used in the other two functions. This is because the constant re-evaluation of the vertical and horizontal edges. In the other two function, narrowing down the field of view results in false detections of water intersects with these removed features which causes the script to misidentify these as part of the structure of impact.

It can get a bit complicated trying to maximise the amount of distracting features to be removed (such as a light source) while minimising false detections so keep this in mind when using these functions:

Remove as much as you need and no more

Future work could involve a step which allows for parts of the image to be removed from processing and not mis-identified as being part of the structure. However as mentioned multiple times, improving the contrast enhancement would have a bigger impact on the overall processing and ensuring the camera is set up correctly with good lighting during experimentation would eliminate this issue in many cases.

MovingObject_successful

Discussion

This video shows *MovingObject* successfully tracking the movement of an objet as it swings in the water to a stand still. Features of this video that make it more successful is that it involves less complex movements as it is a rigid structure swinging, and also that a very good frame around the object was chosen result in very little interference from non-important objects.

Description of function

No area behind the structure was selected by either *roipoly()* or *getrect().* This works well in this particular experiment because the object is thin, has good contrast with its background and there are no other distraction objects in the background to cause processing issues.

Feature explanations

Non-straight_tracking

Discussion

This is a demonstration video showing how the impact of water on complex structure can be tracked. *StationaryObject* was used for this and the UI for selecting the area behind the object is key to drawing the outline of the complex object.

Description of function

Standard set up. *roipoly()* is used to outline a more detailed structure. Again the initial moment of water causes the spike in water level detected which was discussed earlier.

BadContrast

Discussion

This experiment was to test how lower footage quality affects the image tracking of *StationaryObject*. In this experiment, I set up a bad light source and the translucent acrylic structure meant that the contrast enhancing used made the edges hard to detect and resulted in poor results (78% of frames with identified intersects vs the normal ~98%). After taking into account the lengths of the videos, it took twice as long to process this video with bad contrast compared to the video in StationaryObject_successful. This was because as the quality of the video becomes worse, the function performs more calculations to find a fit by “blurring” the detected edges. This is computationally very heavy and is a major point of weakness in all of these functions.

Description of function

Standard set up. It is noticeable in the graph produced (0:48) that there is significant amounts of data missing. This is because if no interest is found, a NaN is returned. This is also shown in the video (1:04) where the red dot visualising the intersect is not plotted for periods of time.

Batch_processing

Discussion

This is a demonstration on how to use this script to process multiple files at the same time. A key requirement of this is that all the files have the same common name (as seen in my directory at 0:06) and it is only some form of version identification that changes.

On the flip side of this, to only process a single video, set the common name to be the exact name of the file including the extension. This will result in only that file being processed.

Description of function

This works on the premise that every video has the EXACT same set up. This is because the first video call is used for the UI to characterise the video features. If there is some structural change between videos then the UI will not be valid and result in poor processing.

Enabling_Edge_detection_video

Discussion

This video is just showing what changes to the code are needed to visualise the edges detected during the processing. It is by default commented out because it adds time to computation and is generally not needed. I found use for it when trying to debug.

When there is an issue processing a video and the cause is not obvious, and it pays to take a look at what the computer is doing. The blue line is the water detected and the red is the structure detected and the red dot is the intersect. In the video there are points that are black (~2:08) which show the other intersects.

It is worth noting that *StationaryObject* and *MovingObject* always take the highest height value intersect while *SlopingObject* takes the height value of the value furthest to the right as the intersect.

Description of function

Previously in the scrip the horizontal and vertical edges are separately found which are used for processing. These are stored as logical matrices. For the video to be generated, the matrices containing these are combined and converted into an RGB image to be put together into a movie.

Load cell readings

Setup

This video is showing how to initialise the data recording LabView function of the load cells and how to calibrate the device.

In the video it is seen that initially the load measured are non-zero. To calibrate these, the programme NI MAX needs to be loaded and if you follow the video instructions it shows you how to zero the load cell readings.

Data_record

Once the load cell is zeroed, the next step is to start recording data. The first step is to type in the directory which the file is to be saved in. It will be saved as a TDMS file and requires a TDMS to excel plugin to perform the conversion.

The DAQ used (C Series Strain/Bridge Input Module) is a high speed module with a lowest data acquisition rate of 1.63kHz. For the data to be recorded from this, a TDMS file is written with a key feature its high stream rate[1]. To access this data requires the use of a TDMS to excel plug in which has to acquired through the vendor with the University license. The computer in the flume lab has this set up but if another computer needs it, contact Stephen Cawley who set this all up for me and is knowledgable in the area.

To record the data, initiate the script first and when required, press “Acquire”. When the data is collected, toggle the logic switch.

The data will be saved in the specified directory and if the TDMS plugin is installed, should automatically open in excel. To plot the data, the time stamp is required which is calculated from the “wf_incriment” which is the size of each time step. Once this is plotted alongside the load cell data, it can be plotted with respect to time.

[1] http://www.ni.com/white-paper/3727/en/