Optimizing Image Processing Algorithms to Explore the Stability of a Granular System

Marissa M. Singh
Physics Department, University of California Davis

Granular material, such as sand and dirt, is a common type of material that is seemingly easy to describe. In reality, granular material behaves in ways unexplainable using conventional physics or mathematics. To efficiently contribute to the research of granular materials, knowing how to employ new technology can greatly improve accuracy of analysis. This research focuses on optimizing algorithms used to explore the stability of a particular granular system. Mainly implementing methods from Open CV, image processing algorithms were modernized to improve the efficiency of a two-part Python program that analyzes the avalanches of granular material.

I. INTRODUCTION

In order to explore the question: "What makes a granular pile more or less stable?", the question that must first be explored is: "What is the best way to explore the stability of a granular system?"

A granular system is a collection of distinct macroscopic particles. Although these systems are simple to describe, they are extremely complicated to study as they exhibit complex behavior. This is because granular materials hold characteristics of solids, liquids, and sometimes gases. A pile of sand for instance has the ability to flow, however it also has the ability to stabilize and remain stationary. What makes them hard to study is the way force runs through the pile. When a force is applied to the system, it spreads through the pile in a chain-like manner. These unique chains are called force chains [1]. Because these force chains are unpredictable and don't affect every ball in the pile, we are forced to look at each ball individually. Once you begin to consider other forces, such as frictional force, the system becomes even more challenging to study. Some scientists have turned to conventional statistical mechanics to try and analyze these systems. However, the particles are macroscopic; temperature plays no significant role on the motion of the particles and the particles have inelastic collisions. As a result, the energy and temperature of the system are not well defined, and conventional statistical mechanics would have to be modified to apply. In order to contribute to the study of this research, Professor Rena J. Zieve has set out to explore the stability of granular systems in hopes to understand more about granular material as well as to create guidelines for future research.

The granular system that we are exploring resides in a container that consists of two square sheets of Plexiglas, red construction paper, a square of metal aluminum that is 1/8 of an inch thick with a hole cut out in the center, and white shelf paper over the aluminum. Inside the aluminum hole is the granular material [2][3]. The reason the metal is so thin is so that we can model a 2D granular system and have the balls confined to a single layer (Figure 1). We are currently working with two structures: hexagons that are painted green and made up of seven ball bearings welded together, and doubles that are left silver and made up of two ball bearings welded together. The entire apparatus is rotated about its center using a stepper motor that receives pulses from a wave function generator. As the system rotates, we observe an avalanche. The speed of rotation is slow so that we can observe each avalanche as an individual event. With the current speed, it takes approximately 30 minutes for the entire apparatus to make a complete revolution. Our goal is to explore how the exact arrangement of the balls makes the pile more or less stable. More specifically, we hope to analyze 10,000 avalanches, organize them by angle of collapse, and create a heat map of the hex-center locations for each angle. The way we plan to achieve this is through data analysis and data acquisition programs. Because this project has been going on for many years, the programs were first written in IDL, translated to Python2, and I caught it in the middle of being translated to Python3. My role throughout this process has largely been to finish translating, debug, modernize, and optimize these programs.

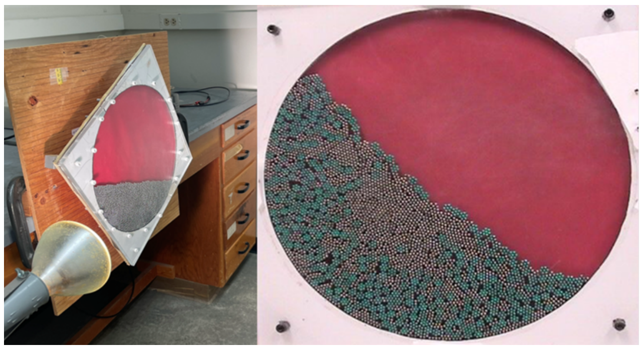
FIG. 1: This is the physical apparatus that contains the granular system of interest. The granular material consists of silver dimers and green hexagons. All material is confined to a single layer, and the entire apparatus is rotated about its center.

II. DATA ACQUISITION

The Data Acquisition program acquires data to be later analyzed by the main analysis program. It is a threaded program that functions in the following manner: (1) Initializes through a setup process that finds the border of the drum (2) Continuously reads in images and finds the angle that the balls are heaped at (3) Looks at the last four documented angles for a net decrease in angle value – if there is a decrease, this means an avalanche has occurred and the program will take 45 frames before the trigger point, 55 frames after, and send this over to the avalanche thread (4) The avalanche thread further analyzes the 100 frames to find the start and end frame of an avalanche – relevant data is saved. When I first started working with this program, it was not able to run and needed to be modernized.

The first algorithm I changed was the method for finding the border of the drum. Originally, the program scanned across the image looking for derivative differences in pixel color. This process was highly contingent on lighting and because our entire setup has been moved from room to room over the years, the lighting we work with is not consistent. To resolve this issue, I turned to an image processing library called Open CV. I first took the image and applied a Gaussian Blur to the image using cv2.GaussianBlur. I then found the edges of the image using cv2.CannyEdges (Figure 2). Because the image is blurred, a lot of the noise in the image is removed, and I'm able to get a nice outline of the border of the drum. The Canny Edges method does not store any coordinate information about the edges it finds, so to get the coordinates of the drum, I found the contours of the image using cv2.FindContours. This method gives you all the contours of a particular image, so to find the exact contour of the drum, I iterated through the contours looking for the contour with the largest length. The longest contour is always that of the circle, so I'm able to get the red trace of the border seen in Figure 3.

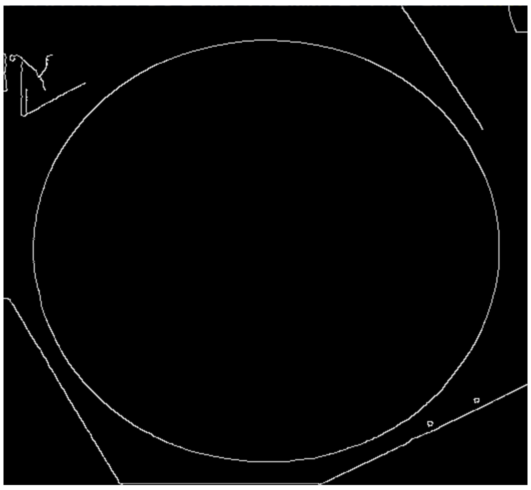

FIG. 2: This is what the image looks like when the methods Gaussian Blur and Canny Edges are applied. The border of the drum is very clear cut, and all other noise is blurred out.

This is an efficient method for finding the border of the drum because lighting no longer affects the accuracy of the trace. Furthermore, the trace is accurate to the exact pixel – this is extremely useful for our research because the border of the drum later determines where the program will look when scanning for balls. If the drum is off by a pixel, it can cut off some parts of the balls on the edge of the drum, and those balls will be missed in the scan.

The second algorithm I changed was the methods for finding the edge of the balls. Similar to the previous algorithm, the program originally scanned down the image looking for differences in pixel color. I changed it to use methods from Open CV. The program first finds the edges of the image. In this case, I did not blur the image so that the pile of the balls would not be mistaken for noise. Instead I directly used cv2.CannyEdges to produce the clear cut image in black and white seen in Figure 4. Because the image is in black and white, the pixel array is in terms of 0s and 255s – 0 corresponding with black pixels and 255 corresponding with white pixels. Instead of scanning the image looking for differences in pixel color, I scanned the image looking for white pixels. The iterations begin at the top most point of the drum. From this point, the algorithm scans down the pixel array until it hits a 255. Once it hits a 255, the coordinate of the white pixel is logged into a list. It then steps up 15 pixels, to the left 1 pixel, and scans down again until it hits a 255. This process is repeated up until the distance between the last point logged into the list and the center of the drum is equal to the radius of the drum. This ensures that the program doesn't run outside of the drum looking for white pixels. Once the scan finds the left half of the edge of the balls, it repeats the same process to find the right half.

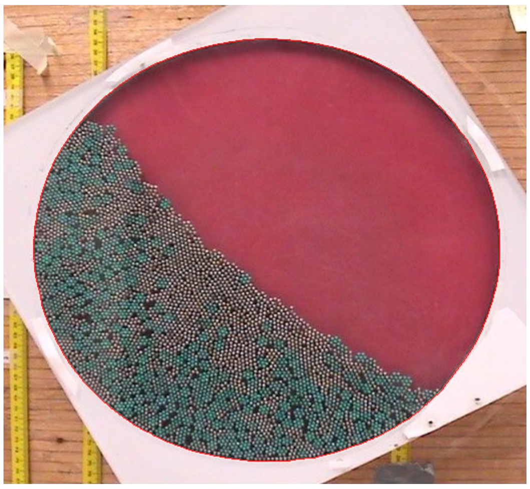

FIG. 3: In this image I overlaid the coordinates of the border against the original frame. The coordinates match up perfectly with the border of the drum, confirming that the new algorithm is precise and accurate.

Although this process is a bit more accurate than the previous algorithm, there is still room for improvement. Because we are now looking for single white pixels, it is possible for the trace to fall through the cracks in between the balls. This results in random low logged points. Although this doesn't severely affect the ball scan as it is only a single low logged point, it can still be fixed. Another fault is that sometimes the trace cuts through balls as seen at the top of Figure 5. I believe that this can be resolved by altering the image of the edges to see the balls as a collective white chunk instead of individual balls. This would allow the scan to work as we had imagined, and accuracy would be improved.

Once these two algorithms were altered, the data acquisition program was running. Our next task was to create an

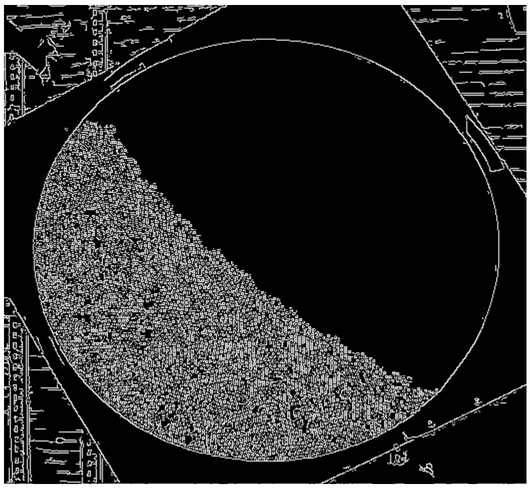

FIG. 4: This is what the image looks like when the method Canny Edges is applied without first applying the Gaussian Blur. Most of the noise is back in the picture, however, where the balls are heaped is clearly visible in white. This allows the new algorithm to find the edge of the balls by only looking for white pixels.

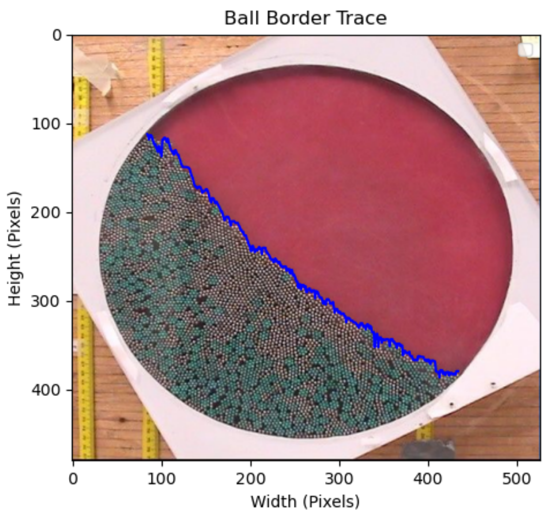

FIG. 5: In this image I overlaid the coordinates of the ball border trace against the original frame. The trace is fairly accurate, but it still cuts out a few balls. The cut out balls can be seen at the very top of the trace.

algorithm for identifying the starting and ending frame of an avalanche. From observing many avalanches, something that we noticed is that there is a rapid increase in the angle data just as the avalanche collapses. Recall that for each avalanche, there are 100 frames that are further analyzed to find the starting and ending frames of an avalanche. Each frame is run through the ball border trace algorithm described above, and the trace of the edge of the balls is converted into a line. The slope of that line is converted to Radians. For a particular avalanche, we are able to graph the progression of angle measurements over an avalanche period. This can be seen in Figure 6.

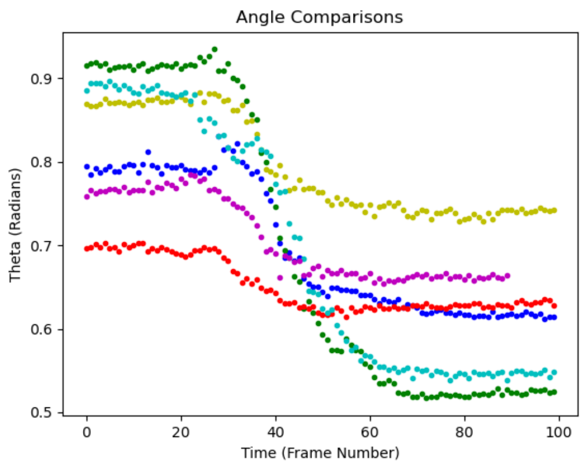

FIG. 6: This graph compares the angle measurements for 6 different avalanches. Each point represents the angle the balls are heaped at for a certain frame. The first thing to note here is how different each avalanche is. All avalanches above reach different max angles before collapsing to different ending angles. The duration of collapse is similar amongst the avalanches, but the intensity of collapse differs. The second thing to note is the behavior of the angle measurements just before collapse. Each avalanche has a small jump in angle measurement just before collapse. In that hump is where the highest angle measurement is, and 20 frames before that point is where we place the beginning frame for the start of an avalanche.

This is because during an avalanche, the upper heap collapses to the bottom. When the program finds the angles of each frame taken during an avalanche, the moving upper heap is included in the angle measurements. As a result, the angle rapidly increases with the max angle signifying the start of collapse. This means that the frame a couple of seconds before the max angle is a still frame that best represents the start of an avalanche. To find this frame, we first created a linear fit of the first 20 angles in the list of avalanche angles being analyzed in the avalanche class. The slope of the fit will remain positive until the avalanche starts to collapse. Because we are fitting 20 angles at a time, the slope starts to decrease when the first half of the angles begin dropping to lower values. We move the window of 20 angles over 1 frame until the slope of the fit is less than a certain threshold. After experimenting with many numbers, we found -0.0035 to be the best threshold number that results in a fit that is well aligned with the avalanche. This slope represents the window of frames midway into the start of an avalanche. At this point, we're able to find the max angle out of the 20 angles. This angle tends to be the highest angle just before collapse. From the max angle, we look back a couple of seconds to find the still frame just before collapse. A couple of seconds corresponds to approximately 20 frames, so we select this frame and mark it as the starting frame.

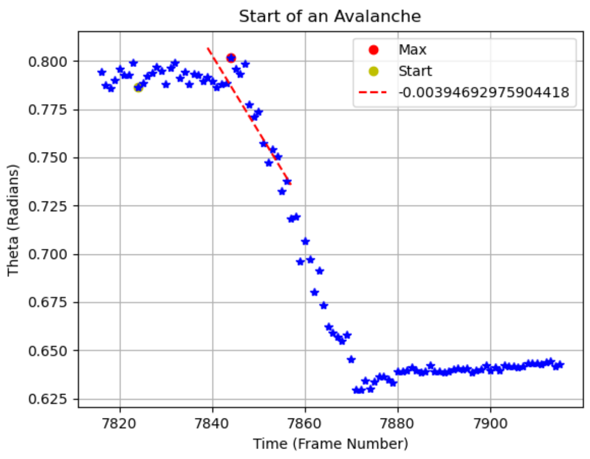

FIG. 7: This graph shows what the linear fit of 20 frames looks like just as it passes the threshold of -0.0035. The max angle is easily found because the threshold ensures that the window of angles in the fit includes the jump of angle measurements just before collapse. The max angle is highlighted in red, and 20 frames before that, the beginning frame is highlighted in yellow.

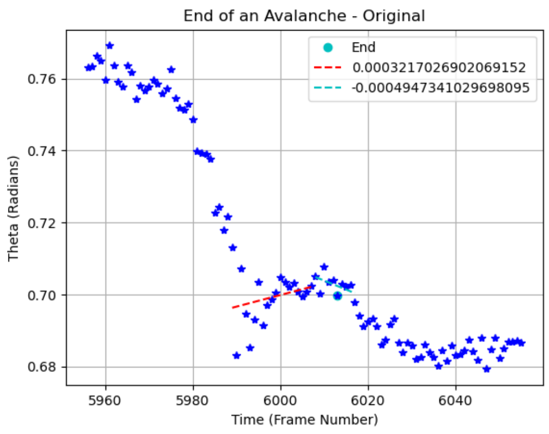

FIG. 8: This image shows the current algorithm for the choosing the ending frame of an avalanche. It currently moves the window of angles to the right until the slope of the angles surpasses -0.0004. 20 frames after this point is where the ending frame is marked. In the avalanche above, there were two tumbles during collapse. The current algorithm places the ending frame, highlighted in blue, in the middle of the second tumble. This results in a blurry ending frame that is difficult to analyze.

We continue moving the window of angles forward until the slope of the fit becomes greater than -0.0004. We look 20 frames after this point and mark it as the ending frame. This algorithm works great for the standard avalanche, however, not every avalanche is perfect. Sometimes an avalanche has a small initial tumble, and is immediately followed by a larger tumble. In avalanches like these, there is a hump in the avalanche angle data. Using the current method for finding the ending frame would result in the ending frame being marked at the middle of the second tumble. A frame pulled from the middle of a tumble tends to be blurry and difficult to analyze (Figure 8). We are currently looking to implement a new algorithm that looks at the potential ending frame, takes a fit of the all the angles between 5 frames before the potential end and 5 frames after. It would then move the smaller window of 10 angles forward until the slope of the new fit is positive (Figure 9).

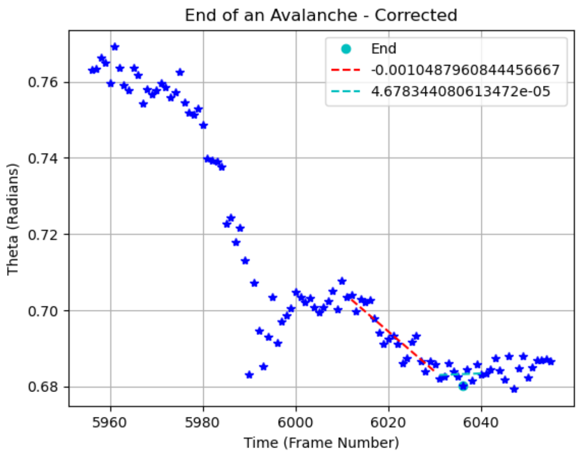

FIG. 9: This image shows the new algorithm that we are working to implement. This algorithm follows the original algorithm up to the point where the ending frame is selected. Once the slope surpasses -0.0004, the algorithm examines the potential ending point by creating a smaller fit surrounding the ending point. It does not mark the ending point until the slope of the smaller fit is positive. The new ending point is highlighted in blue, and we can see that it accurately marks the end of the avalanche.

III. MAIN ANALYSIS PROGRAM

The main analysis program currently analyzes the starting frames of an avalanche and outputs all hex-center locations for a particular frame. This program works in the following manner: (1) Finds the border of the drum (2) Traces the edge of the balls (3) Scans for all ball center locations (4) Finds the balls directly surrounding a ball (nearest neighbors) and the balls surrounding the nearest neighbors of the ball (second nearest neighbors) for each ball (5) Determines which balls are green or silver through a neural network (6) All information is sent to a final module that finds hex-center locations with the help of 5 neural networks based on color and position (7) Program outputs all hex-center locations. After working through many bugs, we got the program running, however, it was only finding two hexagons in the pile. One of the main things that got the program working up to par was an algorithmic change in the module that finds the neighbors of each ball. This module operates by using a Delaunay triangulation to find the neighbors of each ball. It first locates the centers of each ball and marks those with purple dots. It then draws green lines connecting each ball center as seen in the left image of Figure 10. Perpendicular bisectors are drawn through each green line, best seen in right image of Figure 10 with the focus on the upper left ball. The orange vertices mark where the perpendicular bisectors intersect.

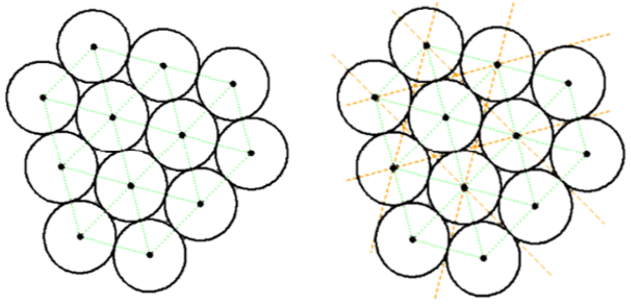
FIG. 10: This is a simplified visual of the Delaunay Triangulation process. The centers of each ball are first connected by the green lines. Perpendicular bisectors are drawn through each green line, depicted in orange. The nearest neighbors of a particular ball are the ball centers that can be reached by crossing exactly one orange line. This process is shown for the top left group of balls above.

For a particular ball, the nearest neighbors are the ball centers that can be reached by crossing exactly one orange line. This process was working great at the center of the pile, however, at the edge of the pile, the orange vertices were extending extremely far from the pile (Figure 11).

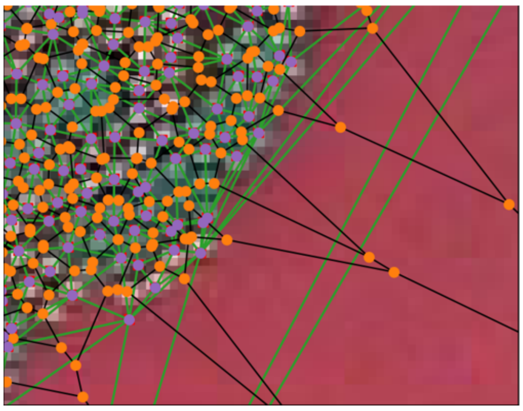
FIG. 11: This is what the Delaunay Triangulation process looks like on a real image of the heaped granular balls. The purple vertices represent the center of each ball, and the orange vertices represent the nearest neighbors of each ball. At the edge of the balls, the algorithm places orange vertices far into the center of the drum where no balls are located. This results in the program incorrectly making different hexagons second nearest neighbors of each other. The program would then fail as two hexagons cannot be second nearest neighbors.

This was causing the balls on the edge to have very long lists of neighbors, mostly comprised of balls that were not neighboring balls. The reason this resulted in many issues is because two hexagons cannot be nearest neighbors or second nearest neighbors of each other. The final module uses this information and throws out any hexagons that are too close together. With the error at the border, the final module was throwing out a large amount of real hexagons. We resolved this issue by creating a new module that layered fake balls along the edge of the real balls. This allowed the Delaunay Triangulation to function as it normally did, however, instead of attaching erroneous balls to the real balls, it attached them to the fake balls. At the end of the process, we swept out the fake balls, and the program was able to function smoothly. This minor change got the program to find 84.194% of the hexagons in the pile. Of the hexagons it found, 99.240% were correctly identified. Something important to note: these numbers are results from the program analyzing the image the neural networks were trained on. This means that these will be the highest amount and most accurate hexagons the program finds, as the neural networks are most familiar with the specific variation of hexagon locations in the training image.

IV. RESULTS AND DISCUSSION

With the new improvements in place, we were able to run the data acquisition program and read in 2600 avalanches. A noticeable difference from the previous algorithms is the speed at which the avalanche thread analyzes an avalanche. Previously, there were issues with the camera buffer filling. When this happened, all threads other than the avalanche thread had to be halted - this means that the camera would stop reading in new frames. Now, the avalanche thread analyzes the 100 frames tied to an avalanche within seconds. There is no longer a need to halt the other threads, and we're able to fully read in every single frame that the camera captures. We are still unsure if this improvement in speed is a result of the newer algorithms or the newer computer that we are now operating on.

Because the data analysis program used the same algorithms as the data acquisition program, we were able to implement the new methods in the analysis. Before we could really see if they improved things, we first had to retrain the neural networks. Since 2013, the neural networks had been trained on images from the older setup. All algorithmic changes between then and now were made to fit the old lighting and cleaner container. Once the neural networks were retrained on a current image, naturally the older algorithms failed. After implementing the new algorithms, the program was able to find 82.0% of the hexagons in the pile. Of the hexagons it found, 95.5% were correctly identified. Again, these numbers are from running the analysis program on the image that the neural networks were trained on. These numbers are very close to the older algorithms, however, they are not as great. This is not entirely the fault of the new algorithms, but rather both the new algorithms and other algorithms that have not yet been optimized. The following issues are the most pressing:

Balls found outside of the border

One of the issues with the old algorithms that we aimed to fix is that the program would find balls outside of the drum. The program classifies balls by looking for the bright spot in a ball and marking that as the center. The older method for finding the border of the drum would include some parts of the white container inside the scan for ball center locations. As a result, the program would see the chunk of white and classify some of the chunk as ball centers. The new method for finding the border of the drum partly fixes this as the new trace is accurate to the exact pixel. Because it is accurate to the exact pixel, the scan sometimes has trouble differentiating exactly when the pixel changes from being a pixel on the border(black) to being a pixel outside of the border(white). This is because when you zoom in on the image, the border of the drum blends a bit into white, so some border pixels are grey. As a result, the trace of the border will have small vertical lines to cover the blend. In order for the scan to work, we have to tell the algorithm to ignore a certain number of pixels at the top of the vertical line so that the entire border is included in the trace. This works perfect for some images, but for others, some white is included in the trace and we end up finding balls outside of the drum.

Balls found outside of the trace

Another issue was that the old algorithm for finding the trace of the edge of the balls would miss balls at the edge of the pile. This is different from the last issue because this is not dealing with the border algorithm. This algorithmic issue comes from the scan where the code is looking for white pixels to classify the edge of the balls inside the drum. The current algorithm has just a bit more accuracy than the previous algorithm as most balls inside the pile are being found, however, it is now finding a few balls outside of the trace. Although this is not significantly affecting how many hexagons are found, it can still be improved. Because most balls inside of the pile are being found, it is up to the remaining algorithms to determine which balls are hexagons and which are not. Hence, the accuracy of the program will improve once the remaining algorithms are optimized.

When images other than the training image are ran through the data analysis program, it finds around 60-80% of the hexagons in the drum. We are not yet able to determine how accurate the hexagons are because in order to get a percentage of accuracy, we have to manually identify all hexagon locations for an image. This takes a lot of time, so we plan to manually locate hexagons for five or more images and gauge the accuracy with those percentages. There are however some images that are not able to get through the analysis. This is because the module that finds the trace of the edge of the balls occasionally fails. With an incomplete trace of the edge of the balls, the program isn't able to run the algorithm that finds all ball center locations, so it halts the analysis. I believe that the module for finding the trace fails because in some images, there are enough cracks in the pile for the scan to completely run through the pile and miss white pixels. Recall that this module documents coordinates of the trace by scanning the image for white pixels. Ideas for fixing this issue are detailed in the following section.

V. CONCLUSION AND FUTURE WORK

Although much progress has been made, there is still much to be done. All algorithmic changes have accomplished one thing: they have modernized and compressed the older code. There was originally tons of folders, excess lines of code, and different modules overlaid in single files. The modernization of both programs has brought a lot of organization to the project, and it will now be easier for someone else to pick this up and take off running. As mentioned above, there are still pressing issues that need to be resolved before we can draw significant conclusions about the stability of granular piles. I believe that the border algorithm can either be improved with a better camera, or adjustments to the current algorithm. A better camera may help eliminate the grey blend of pixels that is seen at the border of drum. Other adjustments could be combining the old algorithm with the new algorithm. We could start off with the new algorithm and first obtain the coordinates for the border of the drum. From there, new code can be implemented to closely examine any small vertical lines in the border. The old code would be used to look for pixel color differences with specialized threshold values. Wherever the switch from inside to outside of the border is detected would be the new point logged in for the border - this would override the vertical line. Another resolution could be counting the pixels in a vertical line, finding the median, and logging that point to replace the vertical line. For the second issue of finding balls outside of the trace of the edge of the balls inside the drum, applying a special filter to the image may help. Currently there's issues where the algorithm can run through the cracks between the balls, resulting in low logged points for the edge. I believe this same issue is causing balls to be found outside of the edge. With a special filter applied to the image, we could alter the image so that the heap of balls becomes a collective clump of white. This would make it very clear to the algorithm where the edge is at, and we would end up with a clearly defined trace of the edge.

Outside of the current algorithmic issues, it would be interesting to explore how changing the colors of the granular material would affect the analysis. The silver and green colors of the balls can sometimes be difficult to differentiate depending on lighting. Painting the balls to drastically different colors, such as bright purple and bright yellow, may help other parts of the program work better. Painting the center of each hexagon a different color could also greatly assist the program's accuracy.

With the code now fully translated and modernized, I believe that we are much closer to reaching our end goal of analyzing the stability of our particular granular system. Once things are working with optimal accuracy, this research will be able to contribute to the mass of research being done on establishing the behavior of granular material.

VI. ACKNOWLEDGEMENTS

I would like to thank Professor Rena J. Zieve for welcoming me into the world of granular material. I have learned a vast amount about how to analyze complex systems under her instruction, and for that, I am very thankful. I would also like to thank the University of California Davis for running the Research Experiences for Undergraduates Program, and the National Science Foundation for funding this incredible journey.

[1] Qicheng Sun, Feng Jin, Jiangui Liu and Guohua Zhang, "Understanding Force Chains in Dense Granular Materials," *International Journal of Modern Physics B* **Vol. 24, No. 29**, 5743 (2010).

[2] A.G. Swartz, J.B. Kalmbach, J. Olson, and R.J. Zieve,"Segregation and stability of a binary granular heap," *Granular Matter* **Vol. 11, No. 3**, 185 (2009).

[3] J. Olson, M. Priester, J. Luo, S. Chopra, and R.J. Zieve, "Packing fractions and maximum angles of stability of granular materials," *Physical Review E* **Vol. 72, No. 3**, 031302 (2005).