2012

Grain wagon fill detection using ultrasonic sensors

John David Gaard
Iowa State University

Grain wagon fill detection using ultrasonic sensors

by

John David Gaard

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Agricultural Engineering (Advanced Machinery Engineering)

Program of Study Committee:

Stuart J. Birrell, Major Professor

Brian L. Steward

Max D. Morris

Iowa State University

Ames, Iowa

2012

Table of Contents

List of Figures

List of Tables

Introduction

In 2011, 88.4x10^6 hectares in the US were harvested with a combine harvester (USDA-NASS, 2012), of which 34.0x10^6 hectares were corn. In search of efficiency, agricultural producers have turned to "unloading on-the-go" which consists of a truck and trailer or tractor and wagon traveling alongside the combine and the combine grain tank being emptied into the trailer or wagon while the combine is moving and harvesting corn. This adds another task for the combine operator, and increases operator stress and fatigue. Not only does the operator have to watch gauges and monitors and steer the combine, the operator now has to make the sure the unloading auger is actually unloading grain into the wagon and not missing the wagon. A well-practiced team consisting of a combine operator and tractor driver can make this operation work, but when one person is inexperienced, the operational capacity of the combine can be significantly decreased. In addition, on-the-go unloading can significantly increase operator fatigue resulting from many hours working in the field, adding to the decrease in combine productivity.

Therefore, a sensing system which allows for automatic even filling of the wagon and automatically shutting off the unloading auger when the wagon is full is desirable, and has the potential to significantly increase combine productivity. Different sensing solutions have been considered. A system using the known GPS position and heading of the combine and the wagon and geometric properties of the wagon for automated control has been patented by Claas of Germany (Behnke, et al., 2004). The same patent

also describes a distance sensor mounted to the side of the combine sensing the distance between the combine and wagon.

Distance measuring devices mounted on the wagon edges to measure the distance down to the grain were considered. These systems would require a microprocessor to gather the information and to wirelessly transmit the information to another microprocessor mounted on the combine. Maintaining consistent and reliable communication between the tractor and combine could be an issue. Outfitting multiple wagons with the sensor and transmitter system would be costly, and would require the manual input of calibration information such as wagon dimensions. The system could transmit fill level information to the combine, but still does not know where the unloading auger is relative to the wagon to ensure even filling.

A sensor package mounted on the unloading auger would deliver depth and geometric information of the wagon to a combine mounted microprocessor that would also control the automatic function of the auger. This system would allow for any type of wagon or cart or truck to pull alongside the combine. Only one control system has to be purchased and mounted on the combine. This requires a sensor system capable of determining the grain cart edge locations, and the grain levels within the carts. A number of different sensing technologies including machine vision, radar, or ultrasonic sensing technologies are possible candidates.

A monocular vision system which uses a camera mounted either on the combine or the unloading auger and image processing techniques could be used to extract features such as the wagon edges and the grain level inside is one option. However, these systems would not have any depth information unlike a stereo camera system which uses two cameras to triangulate points in the image to develop three-dimensional profiles. An alternative is integration of two-dimensional image processing techniques with depth information from auxiliary sensors to produce a three-dimensional scene. Other three-dimensional image generating systems could include using multiple depth measuring devices, such as ultrasonic sensors or laser sensors, to recreate the three-dimensional scene.

Literature Review

Vision Systems

Vision systems using cameras are used in a wide variety of applications including facial recognition and identification systems, feature detection, vehicle guidance systems, medical image processing, and in manufacturing and control. In manufacturing, vision systems have been utilized for a variety of uses such as: object recognition, part dimension inspection, identification of relative position among objects, general work piece manipulation, and even robot position sensing using a known calibration mark on a robot arm (Nilsson and Holmberg, 1994). Vision systems are also used in robot guidance applications such as home cleaning robots and indoor service robots (Ahn et al., 2008).

Although camera vision systems have been successful in a large number of applications, they do have shortcomings, particularly when operating in an outdoor environment. Camera systems have difficulty in dealing with changes in scene illumination and are very sensitive to changes in ambient light intensity and direction. They also suffer from slow update rates due to high computational burdens (Ahn et al., 2008). The alternative is to lower the computational burden at the expense of feature detection accuracy. The proposed application on a combine harvester will experience a large amount of dust in the air. Camera vision systems are easily affected by dust: dust can shroud the view of the desired object and dust can collect on the lens of the camera degrading the effectiveness of the camera system.

Ultrasonic Applications

An ultrasonic sensor transmits a sound wave at frequencies above 20 kHz, beyond the range of most humans hearing ability (Banner Engineering, 2011). Ultrasonic sensors can be driven up to 10 MHz for non-destructive testing of materials, which consists of finding cracks, mechanical flaws, material thickness, and elastic and metallurgical properties of metals (Krautkrämer and Krautkrämer, 1969). Paul Langevin (1918), a French scientist commissioned by the naval forces of France during WW I to develop a system to detect submerged German u-boats, is credited as the father of acoustical imaging technology (Langevin, 1918). Though others were first to experiment, Langevin was the first to succeed by using piezoelectric transducers

operating at resonance which created an adequately intense acoustical transmitting beam.

Ultrasonic sensors have been found to be useful in automotive applications in determining occupant position and occupant velocity during a crash. When combined with the vehicle's crash sensors, ultrasonic sensors add to the airbag control methodology (Breed et al., 1997). They can also be found on the outside of the vehicle used as crash avoidance devices and to detect obstructions in blind spots. As early as 1964, patents were issued detailing ultrasonic sensors mounted at the rear of a vehicle to detect obstacles when the vehicle is travelling in reverse (Grieg, 1964; Cudworth, 1965). Modern automobiles also have "active cruise control" whereby a distance sensor mounted on the front of the vehicle is used as an additional input to the vehicle speed controller (Laiou et al., 2009). When the vehicle approaches another vehicle the speed controller will automatically slow the vehicle to maintain a safe distance behind the vehicle ahead.

Ultrasonic sensors are very useful in the marine industry to map the sea floor and to detect man-made objects such as mines (Murino and Trucco, 2000). Multiple sensors are used to recreate a three-dimensional scene. By adjusting the output frequency and amplitude, ultrasonic sensors can be used for long-range, two-dimensional object detection.

The medical industry uses ultrasonic sensors to create two-dimensional images of the human body. Speed of sound, attenuation, and impedance depend on the body tissue the sound wave is passing through. By using these attributes a 2D image of the human body can be created (Quistgaard, 1997). Either a "scanning" transducer or a fixed array is used to create a B-scan (brightness) image of the area of interest. The brightness of each pixel corresponds to the strength of the reflected sound wave (Schueler et al., 1984). An array consisting of between 48 and 200 active elements is used to transmit and receive steered and focused acoustic beams. A "phased array" uses a small aperture (transmitting elements surrounded by receiving elements) around 15 mm, and forms beams originating from a single point. A "linear array" utilizes a larger aperture around 40 mm, uses more elements than the phased array, and forms acoustic beams normal to the flat surface of the transducer. A "curved linear array" is a linear array formed over a convex curved surface which provides a wider field of view. Three-dimensional images are created by layering two-dimensional images together. However, three-dimensional images prove difficult to create due to excessive processing time (Schueler et al., 1984). Many 3D imaging solutions also suffer from low SNR (signal to noise ratio) because diverging sound waves are used and energy is lost in the soft biological tissue (Lu, 1997).

Ultrasonic sensors have also been used in the robotic field, both in industrial applications and autonomous vehicle applications. In industrial applications, genetic algorithms such as neural networks are used in identifying objects found on conveyer

belts, their orientation, and other features (Brudka and Pacut, 2002). Brudka and Pacut (2002) outline a system which uses an array of 40 ultrasonic sensors in order to achieve high enough resolution. By using certain algorithms, inexpensive ultrasonic sensors can be used in controlled industrial applications. In autonomous robot vehicle applications, ultrasonic sensors can be fused with a vision system for many benefits (Ahn et al., 2008). Ultrasonic sensors are an inexpensive alternative to a stereo camera, add to the accuracy of the vision system, and allow for lowering of the update rate of the vision camera.

Ultrasonic sensors have also been used in autonomous vehicles as standalone sensing devices. By using as few as one to three ultrasonic sensors, the autonomous vehicle can guide itself around its closed environment collecting information about the environment's boundaries and obstacles. A grid map is built and constantly updated using this method (Borenstein and Koren, 1991). The poor spatial resolution of the ultrasonic sensor limits the grid map, so researchers have reduced the two dimensional grid down to a weighted polar histogram which shows the polar obstacle density in a given direction. Obviously, a fully automated vehicle which requires no training to build a world map is desirable. By creating an array of sensors in a circle around the vehicle, the overlap of cones allows for an increase in resolution on an autonomous vehicle (Bank, 2002). The sensor array can then create a 360 degree view of the world around vehicle and the vehicle can make guidance decisions in real time. Prassler et al. (1999) outline an intelligent autonomous wheel chair for transporting elderly and disabled

persons in crowded public environments. The environment is very dynamic where thousands of people and objects are constantly moving, requiring the system to detect obstacles and make guidance decisions quickly and in real time. The group of researchers hopes to extend the design to a people taxi in crowded airports. Bank and others also developed a hospital bed guided by ultrasonic sensors (Bank et al., 1999).

Ultrasonic Feature Detection Methods

In reconstructing three-dimensional scenes in underwater applications, three algorithms and corresponding sensor set-ups have been identified in accurately creating three-dimensional scenes (Murino and Truco, 2000). An “acoustic lens” algorithm is used with an acoustic lens and retina of acoustic sensors. The “focused beam-forming” algorithm and the “acoustic holography” algorithm are used with a two-dimensional array of sensors. In both cases, in order to recreate a three-dimensional scene with acceptable spatial resolution, the number of sensors can reach the thousands, increasing the hardware cost and being computationally intensive. This may be acceptable for mapping a static item such as the sea floor, but for mapping the grain inside the wagon, a sufficiently fast update rate is very important. Computational approaches have been developed to reduce the number of sensors, but systems currently on the market still contain up to 1600 sensors. Regardless of the sensor system, underwater imaging systems usually employ statistical methods which take advantage of the known physical characteristics of objects in order to perform segmentation and scene reconstruction. For example, man-made objects tend to have

less texture, and therefore return less speckle noise, than natural occurring objects such as the sea floor. Other methods involve frequency analysis in which different surfaces return a unique frequency signature. With a high enough resolution, typical image processing techniques such as feature extraction, edge detection, and classification are used. In Bank's previously mentioned system, the Hough transform is used to find edges, but only in two-dimensions (Bank, 2002).

The Canny edge detector, a breakthrough edge detection method when introduced in 1986, first smoothed the image with a Gaussian filter. Edges were then identified by finding the local maxima of the gradient (Canny, 1986). The Canny edge detector suffers from having to manually select a scaling factor which determines the amount of smoothing. As the scaling factor is increased, the sensitivity to noise is decreased but the error in the edge detection increases. As the scaling factor decreases, the reverse is true (Yi et al., 2009). Canny and other techniques have trouble detecting edges in the presence of noise and when edges are near to each other or cross each other. (Ziou and Tabbone, 1998). The Gaussian filtering causes edges close to each other to distort into a single edge. Gaussian filtering also leads to difficulty in determining the orientation of edges, particularly when they cross each other at edges and intersections.

Edge detection in three dimensions is an area of active research. Bahnisch et al. (2009) explains edge detection in three dimensions lacks either speed or accuracy. Extending the Canny edge detector to three dimensions is straightforward but is

computationally intensive. The alternative is to lower the computational burden at the expense of accuracy. New methods continue to be developed, as interest in processing biomedical images is quite high (Cheng and Ma, 2010).

Existing Patents

Existing patents led us to a vision system, and to focus on a sensor package that is fixed to the unloading auger. Therefore, the vision system will mount rigidly to the spout and will not move independent of the spout or the combine. A 1996 patent by Claas of Germany details a spout control system with an optical/acoustic range finder sensor package mounted on the spout (Pollklas, 1996). The sensor package moves independently of the spout. Another patent from Claas in 1998 outlines a similar system, but with a sensor package mounted on the combine (Pollklas, 1998). The sensor package moves relative to the combine and provides a scanning function, viewing the wagon that the combine is filling. As already mentioned, a third patent from Claas describes a spout control system which uses GPS position of the combine and fill cart to determine spout position (Behnke et al., 2004). This precluded us from using GPS position of any vehicle to control spout position. Fortunately, John Deere holds a 2005 patent describing a spout control system utilizing image processing (Alexia et al., 2005).

A sensor package using ultrasonic sensors was chosen over a camera system. Dust and lighting issues with cameras favored ultrasonics. Ultrasonic sensors are not without their faults, however. Sensors can be easily misled by multiple reflections in

moderately complex environments (Knoll, 1991). Sound waves reflect away from the sensor when reflecting off a surface not orthogonal to the sensor. Ultrasonic sensors suffer from poor spatial resolution due to the cone shape of the propagating sound wave. And, ultrasonic sensors experience divergence problems which result in distorted images (Brudka and Pacut, 1999). A camera vision system is still possible if the dust and lighting issues are resolved. An adjustable aperture could partly solve the lighting issue, although operation after dark would still only be possible if adequate lighting was mounted on the combine. Filtering can partly overcome the dust issue, but dust collecting on the camera lens would still need to be resolved.

Objectives

The goal of this project was to build a low-cost ultrasonic sensor package that could accurately identify and model wagon edges and the grain contained in the wagon. In production-ready form, this system could control automatically the swing of the unloading auger to be able to fill the wagon evenly and to avoid the ends of the wagon, and finally, alert the operator and shut off the unloading auger when the wagon is full. To accomplish these items, the following tasks must be completed:

1. Design, develop, and test an ultrasonic sensor package.

2. Develop an algorithm to correctly identify and model the wagon edges using distance information obtained by the ultrasonic sensors.

3. Develop an algorithm to identify and accurately model the grain surface inside the wagon using distance information obtained by the ultrasonic sensors.

4. Using the calculated wagon edge model and grain surface model, determine the grain depth below a particular wagon edge.

Materials

The sensor used in this project was the UC6000-30GM-IUR2-V15 ultrasonic sensor from Pepperl and Fuchs, Mannheim, Germany (Figure 1). The UC6000 has a sensing range of 350 mm to 6000 mm, operates at 65 kHz and has a beam angle of 13 degrees. This sensor was chosen for its distance range, its robustness, and its ability to withstand outdoor weather. The sensor outputs an analog voltage proportional to the distance sensed. The sensor uses internal conditioning to create a very repeatable and consistent measurement.

It was determined that an angle of vision of 104 degrees was necessary to properly view the wagon. This was accomplished with a set of four sensors mounted in a rotating base, in which each sensor was mounted at different fixed angles to the axis of rotation: 6.5 degrees, 19.5 degrees, 32.5 degrees, and 45.5 degrees (Figure 2). This created no overlap or under lap between sensors, and the resolution of each sensor along its circular travel path was limited to the update rate of the sensor, which was approximately 3.0 Hz.

A small electrical motor, part number 6409K13 (McMaster-Carr, Elmhurst, Illinois) rotated the sensor base at approximately 4 revolutions per minute and an HB6M -1000 rotary encoder (US Digital, Vancouver, Washington) recorded the sensor position. An Orbex Group (Fremont, CA) 12 circuit hollow bore slip ring, part number 512-1200, allowed for communication between the sensors and the data acquisition device while the sensor base rotated. A USB-1408FS data acquisition device

(Measurement Computing Corporation, Norton, Massachusetts) was used to record the data (Figure 3). The encoder and the outputs of the sensors were connected to the analog inputs of the USB-1408FS and those channels were measured and recorded at 536 Hz. The encoder channels were digital outputs which consisted of an Index, A, and B signal. The Index channel goes high once every revolution; A and B channels pulse 1000 times per revolution. The analog sampling rate was increased until the digital pulses from the encoder could be read. The digital outputs of the encoder and the analog output of the ultrasonic sensors were all read on the analog channels of the USB-1408FS to make post processing simpler.

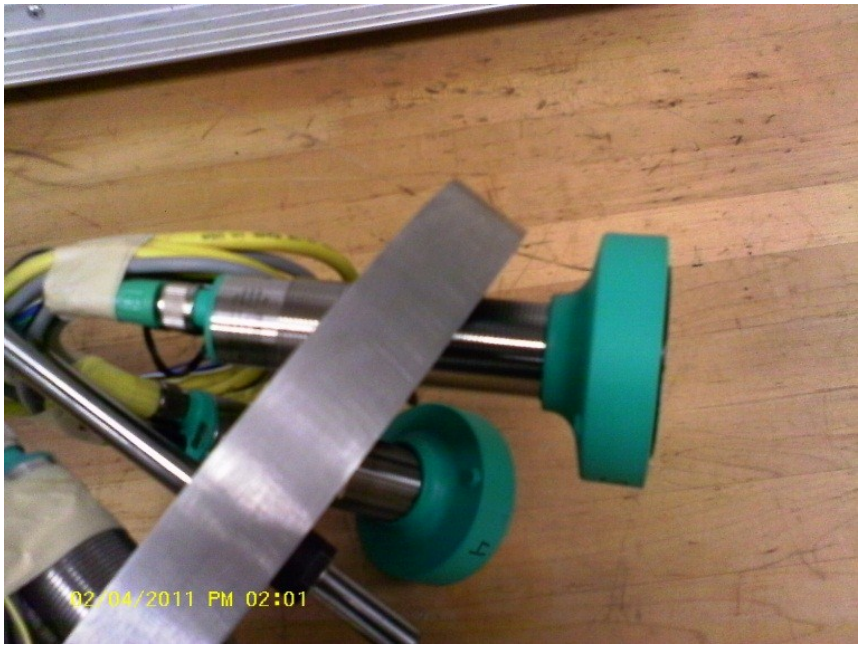

Figure 1. Pepperl and Fuchs UC6000-30GM-IUR2-V15 ultrasonic sensor.

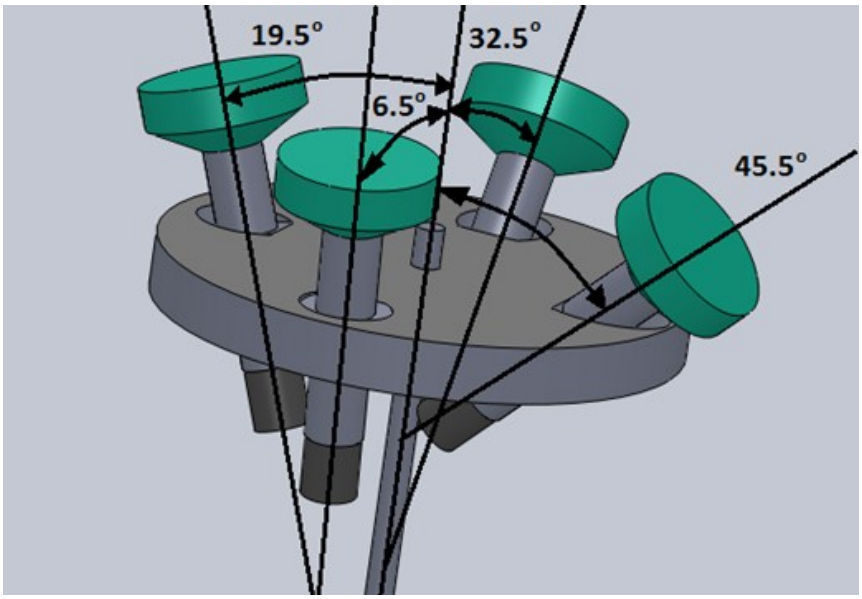

Figure 2. Sensors installed in rotating base.

Figure 3. The ultrasonic sensor package with encoder, electrical motor, and USB1408FS data acquisition device.

Methods and Analysis

Test Procedure

The sensor package was mounted on the unloading auger of a John Deere 9860 STS combine harvester (Deere and Company, Moline, Illinois) approximately 1.2 meters from the end of the auger and approximately 4.9 meters from the auger pivot point (Figure 4). A 640 bushel Brent gravity wagon (Unverferth Manufacturing, Kalida, Ohio) filled with shelled corn was positioned under the unloading auger. The sensor package viewed a different section of the wagon at each different measurement position. The range of positions selected was typical for unloading grain into a wagon (Figure 5). Data was collected at each position. The analog output of each sensor was recorded for one revolution of the sensor package.

Earlier testing revealed that having the center of rotation of the sensor package perpendicular to the ground produced the best results. This position provided more dynamic depth information at the expense of a reduced sensing window. Testing with the sensor package mounted more diagonal to the ground to gain a larger sensing window produced less dynamic depth information which proved to be undesirable.

Figure 4. Sensor package mounted on combine harvester unloading auger.

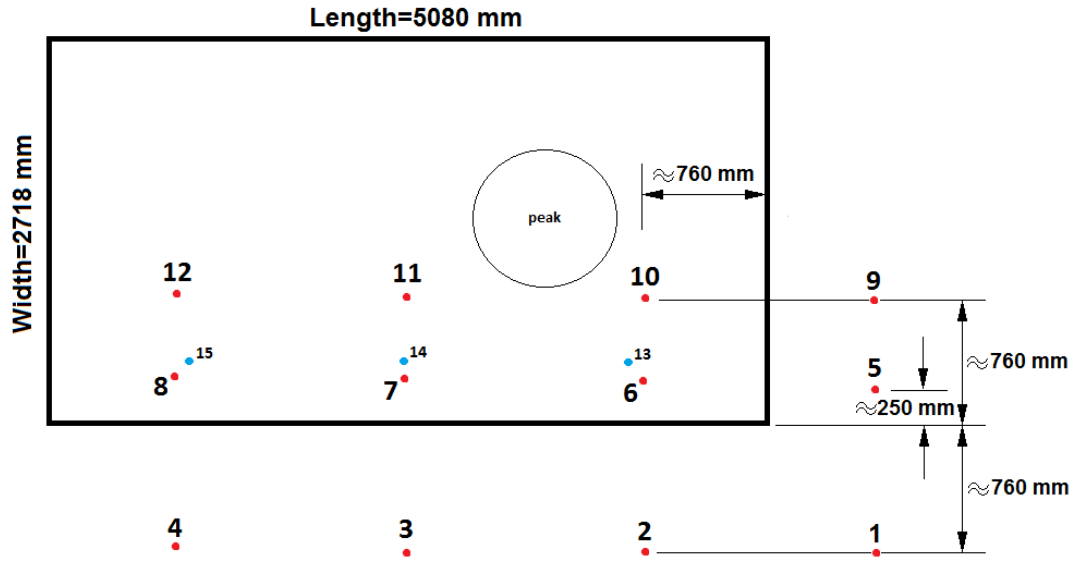

Figure 5. Red dots represent sensor positions relative to the wagon edges while data was collected. Blue dots at positions thirteen through fifteen represent sensor position while data was collected with grain flowing from unloading auger.

Figure 6. View of grain pile inside wagon from front of wagon.

Figure 7. View of grain from rear of wagon.

Data Analysis

The data consisted of values of seven analog channels collected at 536 Hz. The data, located in a comma delimited file, was opened in Matlab for processing. The encoder produced 1000 pulses per revolution, so the analog value of each sensor was sampled at each encoder pulse. The analog array was then down-sampled to 56 samples per revolution giving an angular resolution of 1 sample per 6.4 degrees. Each analog value was then assigned to the correct angular value in radians. The analog values were converted from a binary value to a distance in millimeters using equations (1) and (2). The data acquisition board used 14 bit channels that had a voltage range of -10 to +10 volts. The voltage to distance equation was determined from the theoretical calibration of the sensor and confirmed with empirical testing in the laboratory.

$$vdc = \left(\frac{bit}{16384}\right) 20 - 10 \qquad (1)$$

$$distance\ (mm) = 565(vdc) + 350 \qquad (2)$$

The x, y, and z coordinates were then calculated using the known geometry of the sensor plate and the measured position of the sensor package provided by the encoder. Although the ultrasonic sensor distance response could be from any object within the sensor response cone (+/- 6 degrees), the nominal x,y,z location equations 3-5, assumed a reflection from the center of the response cone. The nominal x,y,z location of the response surface was estimated with the following equations:

$$z = (dist + SH)\cos(\text{FSA}) \quad (3)$$

$$x = \left(\sqrt{(dist + SH)^2 - z^2} + radius\right)\sin\theta \quad (4)$$

$$y = \left(\sqrt{(dist + SH)^2 - z^2} + radius\right)\cos\theta \quad (5)$$

Where, dist=distance returned from sensor
SH=fixed height of sensor head from center of sensor plate
FSA=fixed sensor angle measured from rotation axis
radius=distance of sensor from center of rotation
Θ=angle returned from encoder

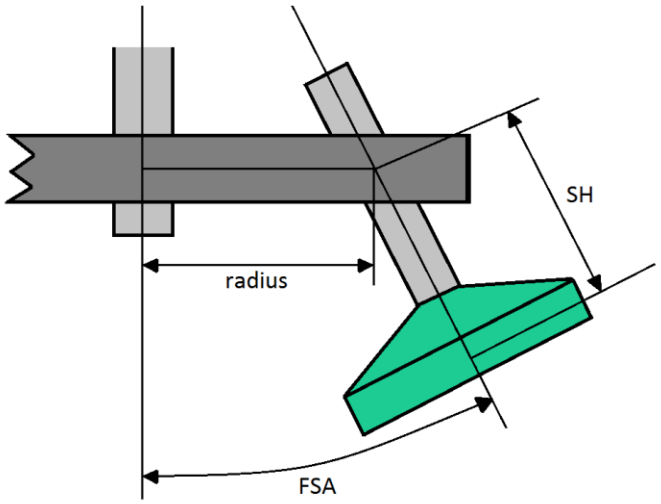

Figure 8. Sensor mount dimensions where, *SH* = fixed height of sensor head from center of plate, *FSA* = fixed sensor angle relative to rotation axis, and *radius* = distance of sensor from center of rotation.

Once the nominal x,y,z location of the response surface was estimated, a process was developed to extract the desired information. Features were extracted from the data; then the features were labeled as wagon edge or grain. It was assumed that the grain would form a consistent, smooth surface which could be easily distinguished from the wagon edges. The flow chart in Figure 9 provides a summary of the procedure.

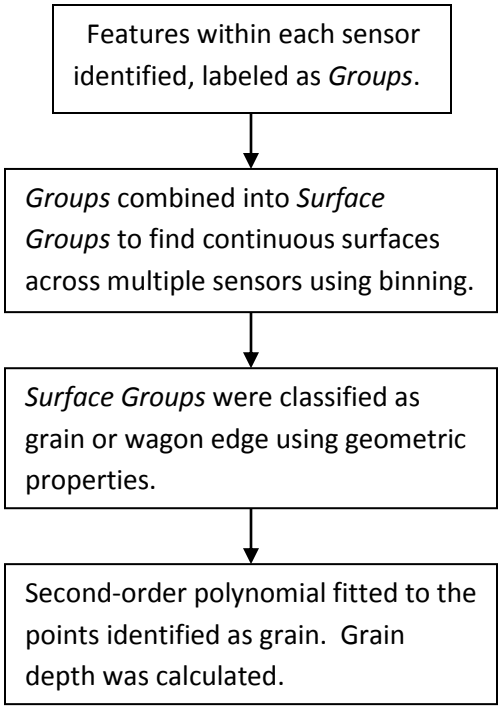

Figure 9. Flow chart of data analysis.

The points were then segmented into groups within each sensor. The resultant vector of all three coordinates was found and the first and second derivative of the resultant vector was calculated:

$$d_i(x,y,z) = \sqrt{{x_i}^2 + {y_i}^2 + {z_i}^2} - \sqrt{{x_{i-1}}^2 + {y_{i-1}}^2 + {z_{i-1}}^2} \quad (6)$$

$$d_i{}^2(x,y,z) = \{\sqrt{{x_i}^2 + {y_i}^2 + {z_i}^2} - \sqrt{{x_{i-1}}^2 + {y_{i-1}}^2 + {z_{i-1}}^2}\} \quad (7)$$

$$-\{\sqrt{{x_{i-1}}^2 + {y_{i-1}}^2 + {z_{i-1}}^2} - \sqrt{{x_{i-2}}^2 + {y_{i-2}}^2 + {z_{i-2}}^2}\}$$

A pair of corresponding negative and positive peaks of the second derivative above a particular threshold specified the end of one group and the start of another

group (Figure 10). These groups were determined to find the possible location where the sensor response distance moved from one continuous surface to a different surface.

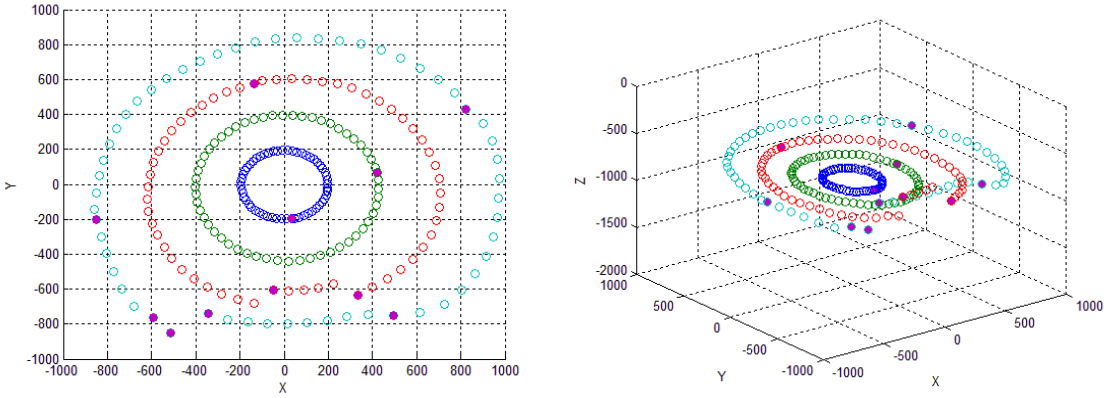

Figure 10. Each color circle represents a different sensor. Solid purple circles represent last points in each group.

Points from all four sensors were combined into one array and labeled according to sensor number and group number which were determined from the second derivative of the resultant vector. Once the points were separated into groups, the points were then combined into one array in order to recognize continuous surfaces across different sensors.

To find continuous surfaces across different sensors, the points were separated into bins in the x and y direction. It was assumed that the wagon edges were parallel with the defined axes. This would make the binning method the most effective at recognizing edges. The derivative, dz/dx and dz/dy, was calculated within the bins to further classify the points. The x and y ranges of the points were divided into sixteen equal parts; sixteen rows were created in the y direction and sixteen columns were

created in the y direction. The width of the rows and columns depended on the full range divided by sixteen, but were approximately 120 mm wide. The points from the sensor forming the outside ring determined the number of rows or columns. The smaller the width of the column or row represented, the greater the maximum resolution/accuracy possible. However, the maximum number of rows and columns was limited if each row and column was required to have at least one point from the outside ring. In these tests, the smallest possible width was approximately 120mm wide while still including a point from the outside ring in each column or row.

For the columns, the x value was ignored, the points were assumed to be collinear. The derivative, dz/dy was found for each point in each column. The same was true for the rows, and dz/dx was found for each point in each row. If the derivative was below a certain threshold, the two *groups* that the two points belonged to were considered a match; above the threshold the groups were considered a non-match. The matches and non-matches were weighted against each other to combine the *groups* into new *surface groups*. Each *surface group* was assumed to be reflections from the same surface plane (i.e. wagon edge or grain surface). To check against false matches, the most populous *surface group* was tested. For each *group* within the *surface group*, matches and non-matches were found between it and the other *groups*. If the non-matches outnumbered the matches, the matching group pair was thrown out and the most populous *surface group* was recalculated. The excluded matching group pairs were then replaced in the array and the remaining *surface groups* were calculated.

Individual groups not matched up were labeled as a *surface group* also. *Surface groups* with less than three points were excluded from further processing. The *surface groups* now had to be classified as grain or an edge (Figure 11).

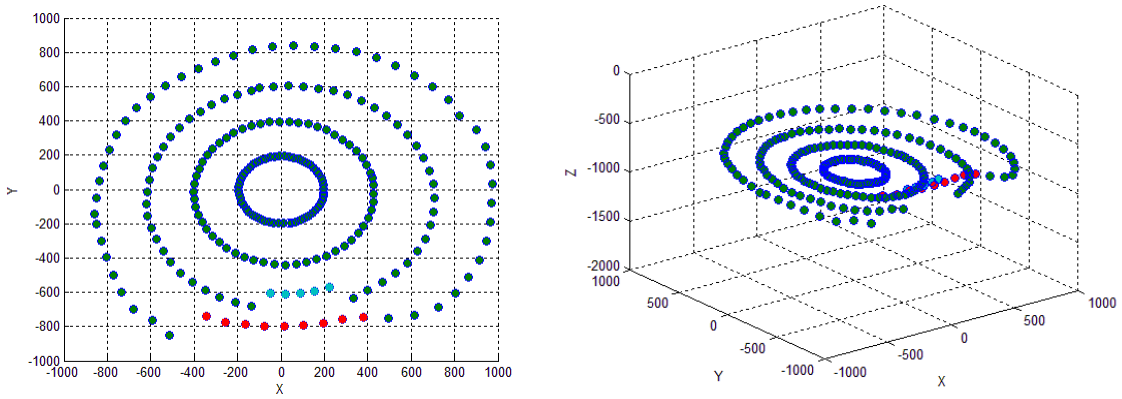

Figure 11. *Groups* combined into *Surface Groups*. Green points are one group, blue points are another, and red points are a third group.

Since the y axis was set in line with the axial direction of the auger and the x axis perpendicular to it, the edges of the wagon were assumed to be parallel to the x or y axes. So, *surface groups* were classified by comparing the ratio of the range in the x direction to the range in the y direction. A *surface group* with an x-y ratio close to one was considered as grain. A range was set enclosing a ratio of one; within the range the *surface group* was classified as grain, and outside the range the *surface group* was classified as an edge. The range was set wide so that anything classified as an edge was definitely an edge, while an actual edge could be misclassified as grain. If the *surface group* was classified as potential grain, the number of sensors that made up the *surface group* was examined. If the *surface group* consisted of just one sensor, it was simply

classified as grain. If the surface group consisted of more than one sensor, the x-y ratio was checked again with a tighter grain classification range. This was done because edges which were found by two sensors had an x-y ratio closer to 1 than edges found with just one sensor (Figure 12). This also prevented *surface groups* which consisted of one sensor and an x-y ratio further away from 1 from being misclassified as edges. *Surface groups* classified as an edge were also classified at that time as a vertical edge (front or rear edge of wagon) or a horizontal edge (near or far edge of wagon).

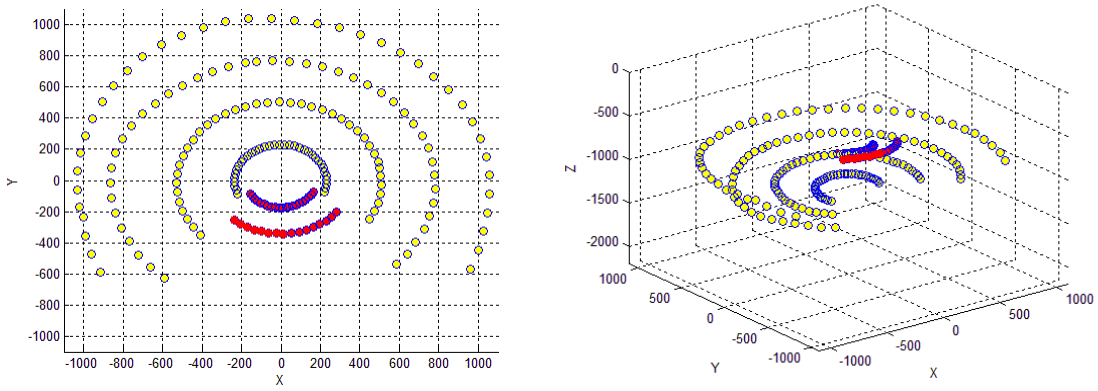

Figure 12. Horizontal edge identified in red. Edge was found by two sensors. Grain identified in yellow.

If more than one horizontal or vertical edge was identified, false edges were identified and thrown out. False edges were merely found by calculating the average z value of each edge and the edge with the highest z value was considered the true edge and all others were considered false. Horizontal and vertical edges were also compared against each other. If the average z value of each differed by more than 100 mm, the lower of the two was thrown out.

With true edges and true grain identified, grain depth could now be calculated. A second order polynomial was fitted to the points identified as grain. A line was fitted to the edges in two different orientations, x by z and y by z. The two line equations were combined to form a three-dimensional line equation (Figure 13). In production form, the grain depth, which is the difference in height between the edge and the grain, could now be calculated at the midpoint of the edge. For evaluation purposes, the grain depth was calculated at the point of measured grain depth nearest to the midpoint of the edge.

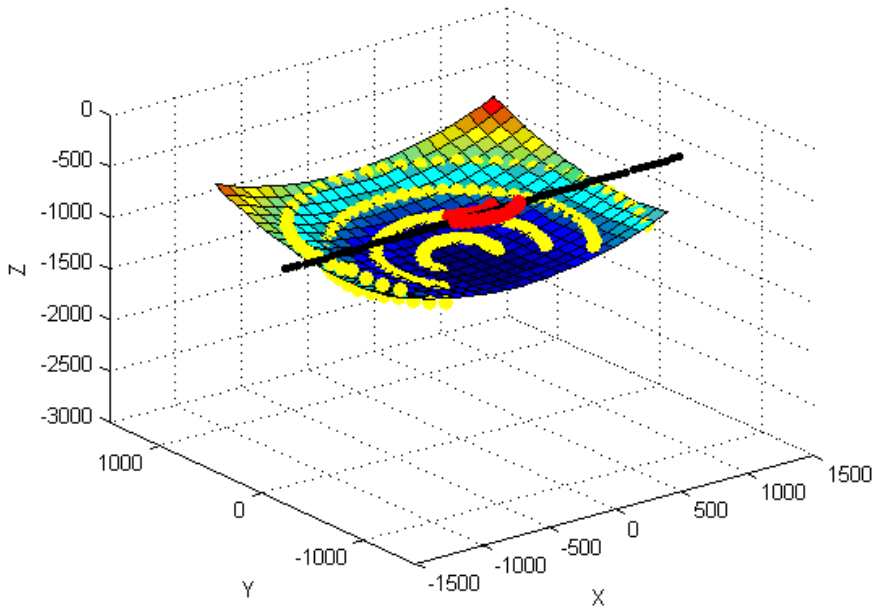

Figure 13. Second order surface fitted to grain and line fitted to edge.

Results

The sensor package was successfully developed and tested. Data was collected at representative locations with the sensor package looking into a wagon. A grid pattern was created and the grain depth below the top wagon edge was measured at intersection points on the grid (Figure 14). At each new sensor position the distance to three of the four wagon corners was measured, which determined the position of the wagon relative to the sensors. The predicted grain surface could then be compared to the known points on the grid, and the predicted grain-edge intersection could be compared to the known grain-edge intersection. The measured grain surface profile is depicted in Figure 15.

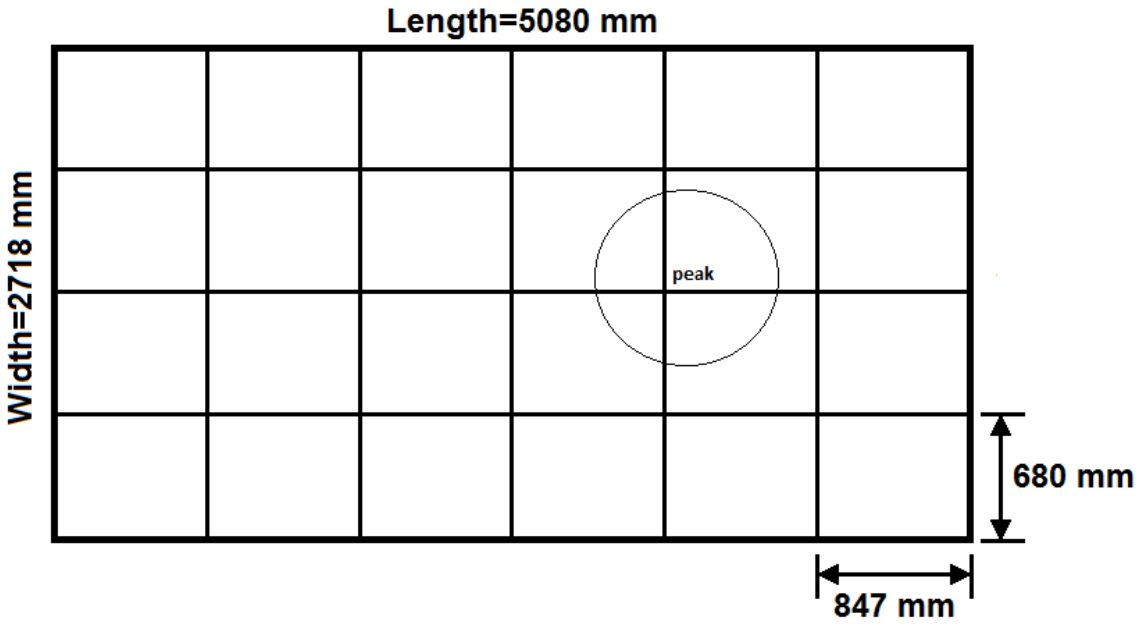

Figure 14. Grain depth was measured at each intersection of a gridline and at the intersection of the gridline with a wagon edge.

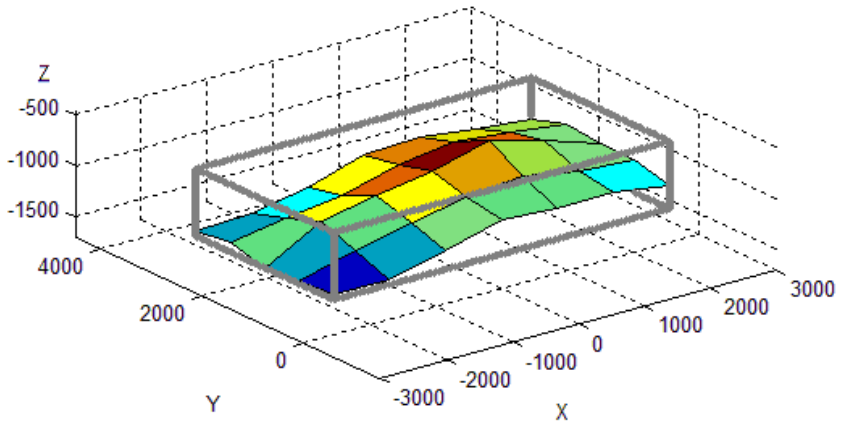

Figure 15: Measured actual grain surface and wagon edges plotted in Matlab.

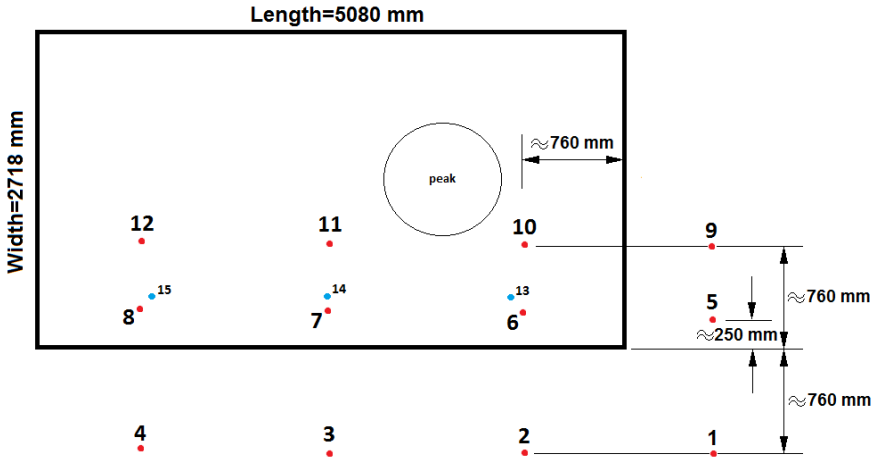

Figure 16. Red dots represent sensor positions relative to the wagon edges while data was collected. Blue dots at positions thirteen through fifteen represent sensor position while data was collected with grain flowing from unloading auger.

No sensor measurements were received at positions 1 and 5. With the sensors at the front corner of the wagon, it is assumed that the sound waves were refracted because the wagon surfaces were at too great of an angle to reflect the sound waves back to the ultrasonic transducer. The sensor package was outside the wagon at positions two through four; therefore, no grain was detected. The edge was successfully detected within 53 mm for positions two through four.

Table 1. Results summary table. All numbers are in millimeters in the z direction with the sensor being the origin. Average error in Grain Depth (edge to grain distance) was within 75 millimeters.

Position	Actual Grain-Edge Intersection	Predicted Grain-Edge Intersection	Error	Actual Edge	Predicted Edge	Error	Actual Grain Depth	Predicted Grain Depth	Error
1		no results			no results			no results	
2		no results		-704.0	-680.8	23.2		no results	
3		no results		-726.0	-680.2	45.8		no results	
4		no results		-739.0	-686.2	52.8		no results	
5		no results			no results			no results	
6 Near Edge	-1140.0	-995.3	144.7	-709.0	-704.8	4.2	-431.0	-290.5	140.5
6 Front Edge	-1140.0	-995.3	144.7	-709.0	-671.8	37.2	-431.0	-323.4	107.6
7	-1044.0	-1036.2	7.8	-714.0	-701.5	12.5	-330.0	-334.6	-4.6
8	-1326.0	-1224.5	101.5	-716.0	-698.3	17.7	-610.0	-526.2	83.8
9 Front Edge	-1064.0	-1259.6	-195.6	-734.0	-678.1	55.9	-330.0	-581.5	-251.5
10	-1077.0	-915.2	161.8	-721.0	-677.0	44.0	-356.0	-238.2	117.8
11	-1031.0	-862.1	168.9	-701.0	-684.9	16.1	-330.0	-177.2	152.8
12	-1275.0	-1052.7	222.3	-665.0	-675.5	-10.5	-610.0	-377.1	232.9
Average			94.5			27.2			72.4

NOTE: UNITS ARE IN MILLIMETERS

Edges were successfully identified at all positions except for one and five. The average error between the actual edge z height and the predicted edge z height was 27.2 mm across all sensor positions. The error was calculated at the measured value closest to the midpoint of the sensing range. At positions six and fifteen, the side and end edge of the wagon were both successfully identified (Figure 17). In positions 12 and 13, the sensor may have been receiving reflections from an end edge, but the edge was dropped due to its height being lower than the other sensed edge (Figure 18). A false edge was identified in position 11. A *surface group* of points were segmented out, rather than combined with other grain groups. The *surface group* passed the x-y ratio test and was identified as an edge.

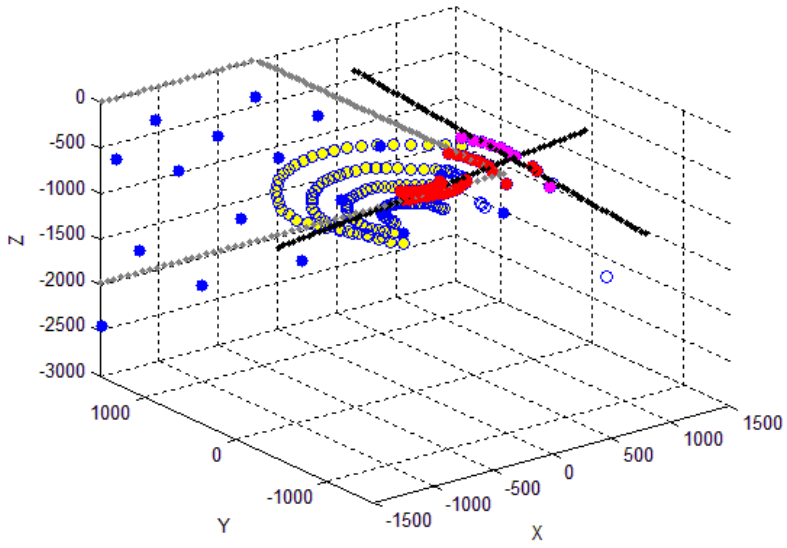

Figure 17. Front and side edges were identified at position six. Actual wagon edges are indicated by grey lines. Points indentified as grain are in yellow. Blue points are the measured grain surface.

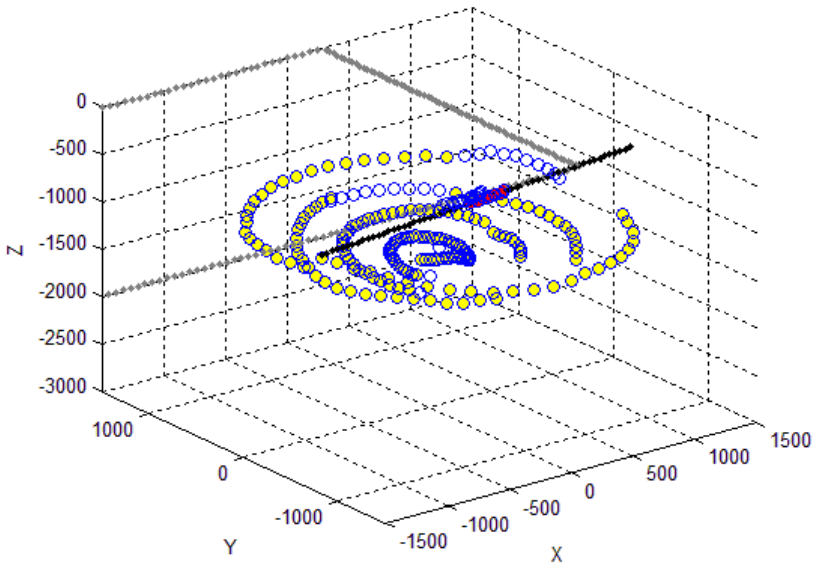

Figure 18. Sensor may have received reflections from front wagon edge but the *surface group* was dropped due to selection criteria at position thirteen. Blue outlined circles with no color fill inside represent unclassified measurement points.

A second order polynomial was fitted to the points identified as grain. The second order surface was then compared to the measured grain depth points (Figure 14) that were enclosed by the sensor range in the x and y direction. The predicted surface (z) was calculated using the second order prediction equation at the x and y coordinates of each measured grain depth point. For all positions, the predicted surface points were compared to the nine measured points in JMP statistical software by fitting a first order linear regression to the measured points versus the predicted points. The RMSE ranged from a best of 69.19 mm up to a high of 1011.06 mm (Table 2). The particularly poor RMSE of 1011.06 at position 9 results from too few points being identified as grain, resulting in a poor fit. This was of no consequence since the system properly identified that the sensors were outside the wagon edges and therefore no grain should be flowing from the unloading auger. This analysis was also restricted by limited degrees of freedom. The model accounted for one degree of freedom out of a total of only nine.

Table 2. Grain surface fit summary for static grain tests.

Position	# of observations	RMSE
1		
2		
3		
4		
5		
6	9	126.20
7	9	148.12
8	9	226.36
9	9	1011.06 *
10	9	69.19
11	9	137.07
12	9	144.00

* The system correctly identified that the sensors were outside the wagon edge so grain surface information was unimportant.

Further skewing results was the cone angle of each ultrasonic sensor. By the nature of a propagating sound wave, the sensing window of the sensor widens as the distance from the sensor is increased. When the sound wave reflects off a surface not orthogonal to the sensor, the distance predicted is shorter than the actual distance because the distance is assumed to be on the centerline of the sensor (Figure 19). This effect worsens as the angle between the sensor and the reflecting surface increases. Therefore, the sensors mounted at greater angles in the sensor mounting plate under-predicted distance more than the sensors mounted more perpendicular to the sensor mounting plate. Figure 20 shows a second order model fit to error between predicted values and measured values versus the distance from the centerline of the sensor assembly. R^2 value of the second order fit was 0.70. This graphic shows that the

distance became more under-predicted the further away from the centerline of the sensor package.

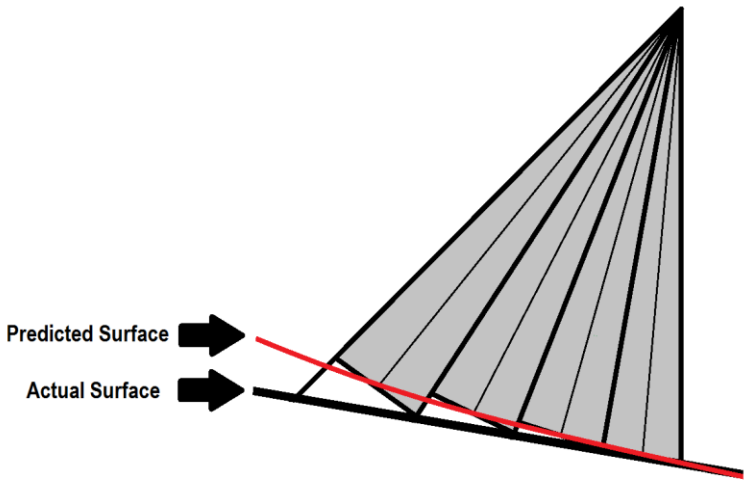

Figure 19. Distance measured assumed to be on the centerline of the sensor.

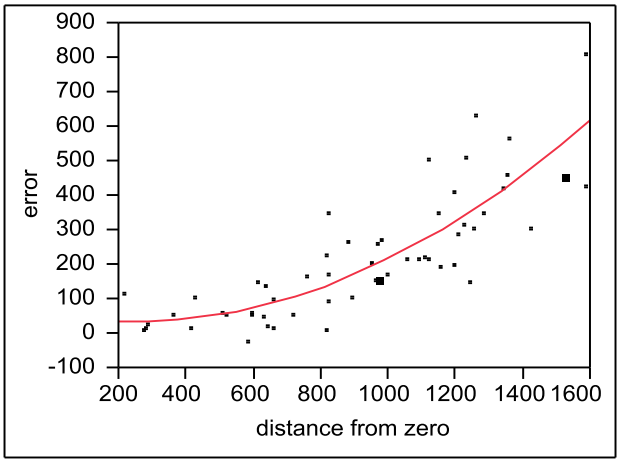

Figure 20. Error between predicted value and measured value versus distance from centerline of sensor assembly. R^2 of second order fit was 0.70.

Next, the grain surface equation was solved where the grain surface met the wagon edge and at the measured grain-edge intersection nearest to the middle of the sensing range. Because of the poor grain surface fit, the grain-edge measurement was calculated at one point and at the center of the sensing range. The grain-edge intersection was calculated at the measured grain depth nearest to the center of the sensing range only for comparison. The average grain-edge intersection error was 94.5 mm, with the worst error being 222.3 mm and the least error being 7.8 mm

Combining the wagon edge z position and the grain-edge intersection created the grain depth measurement which a controller would use to decide when the wagon was full. The average error for grain depth was 72.4 mm, with -251.5 mm being the greatest error and -4.6 mm being the least error.

Data was collected at positions thirteen through fifteen while grain was flowing out of the unloading auger to confirm that the system would function properly. At each position the grain and the wagon edges were correctly identified (Figure 21). No numbers exist to report as the grain depth was not constant.

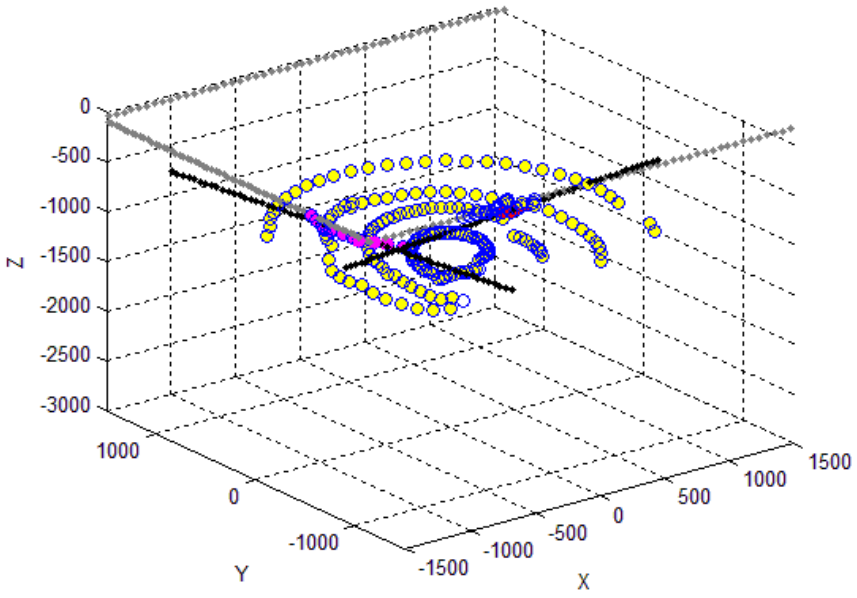

Figure 21. Near wagon edge, rear wagon edge, and grain correctly identified at position 15 while grain was flowing.

Conclusions

Summary

A low cost ultrasonic sensor package was successfully developed and tested. This project showed that this sensor package could sense the wagon edge and the grain surface inside to determine the grain depth. The grain depth information could then be used to automatically control the swing of the unloading auger. It was shown that these features could be successfully extracted even with the low number of sensing elements. The wagon edges were identified with an acceptable amount of error in the predicted z height. The grain surface was predicted with a disappointing amount of

error. Perhaps the error induced by the sensor cone angle could be reduced by more processing. By using the angle between the predicted surface and the sensor, the amount of the cone angle error could be predicted and the sensed distance from each sensor adjusted accordingly. When the edge prediction and the grain-edge intersection prediction were combined, the grain depth was predicted with an average error of 72.4 mm, which is a workable amount of error. The error of 72.4 mm means the grain depth was under-predicted. Of the eight grain depth predictions, six were under-predicted, which is desirable as an input to a controller which will turn off grain flow automatically once grain depth reaches a pre-determined limit. It is helpful that the system under-predicts consistently so that it behaves consistently for the user and a simple offset can be applied to tune the system to the user's liking.

Recommendations for Future Work

The effects of the cone angle need to be reduced with further processing. Once this is done and the grain surface can be predicted with greater accuracy, more automatic control could be done. The control system could start by filling the wagon on one end. Once one end of the wagon was full, the profile of the grain could be sensed, and the auger automatically swung away from the edge of the wagon or the speed of the combine adjusted. This could continue until the wagon is filled evenly.

The update rate of the sensors also needs to be increased to allow for real time control system decisions. The current system updates once every 15 seconds or at 0.067 Hz. The speed of rotation of the sensor base could be increased, but the accuracy

of the distance measurement would have to be analyzed. A critical look at the sensor package is also needed to reduce cost and increase robustness for the real-world environment. The rotating aspect of the sensor package adds complexity and reduces reliability. A static array of a larger number of ultrasonic sensors could be pursued.

The feature extraction methodology needs to become more robust and be less based on assumptions. The ideal methodology would identify wagon edges regardless of wagon orientation relative to the combine, regardless of wagon tilt relative to the combine (from a front view), and regardless of the type of grain contained in the wagon. Different grains would present different grain pile shapes and different reflective properties.

APPENDIX A

Table 3: Predicted and actual Z values summary table.

UNITS ARE IN MILLIMETERS

		x	y	actual z	predict z
Position 6	1	569	-237	-1140	-995
	2	569	442	-1039	-989
	3	569	1122	-1039	-737
	4	-278	-237	-1065	-1016
	5	-278	442	-1014	-967
	6	-278	1122	-861	-672
	7	-1124	-237	-1065	-720
	8	-1124	442	-912	-627
	9	-1124	1122	-709	-289
Position 7	1	913	-267	-1070	-869
	2	913	412	-917	-752
	3	913	1092	-714	-416
	4	66	-267	-1044	-1036
	5	66	412	-968	-956
	6	66	1092	-866	-657
	7	-781	-267	-1197	-854
	8	-781	412	-1070	-812
	9	-781	1092	-968	-549
Position 8	1	527	-282	-1199	-1150
	2	527	397	-1072	-1063
	3	527	1077	-970	-773
	4	-319	-282	-1326	-1224
	5	-319	397	-1199	-1144
	6	-319	1077	-1072	-862
	7	-1166	-282	-1326	-918
	8	-1166	397	-1351	-846
	9	-1166	1077	-1376	-572

Table 3 continued

UNITS ARE IN MILLIMETERS

		x	y	actual z	predict z
Position 9	1	-683	-693	-1165	-1001
	2	-683	-14	-1064	-1260
	3	-683	666	-1064	-1306
	4	-1530	-693	-1090	-718
	5	-1530	-14	-1039	-572
	6	-1530	666	-886	-214
	7	-2376	-693	-1090	1087
	8	-2376	-14	-937	1637
	9	-2376	666	-734	2400
Position 10	1	1057	-732	-1152	-805
	2	1057	-53	-1051	-840
	3	1057	627	-1051	-739
	4	210	-732	-1077	-915
	5	210	-53	-1026	-916
	6	210	627	-873	-781
	7	-636	-732	-1077	-822
	8	-636	-53	-924	-788
	9	-636	627	-721	-620
Position 11	1	1106	-785	-1057	-604
	2	1106	-106	-904	-689
	3	1106	574	-701	-555
	4	259	-785	-1031	-862
	5	259	-106	-955	-945
	6	259	574	-853	-810
	7	-588	-785	-1184	-920
	8	-588	-106	-1057	-1001
	9	-588	574	-955	-864
Position 12	1	575	-775	-1148	-996
	2	575	-96	-1021	-1051
	3	575	584	-919	-912
	4	-271	-775	-1275	-1053
	5	-271	-96	-1148	-1124
	6	-271	584	-1021	-1003
	7	-1118	-775	-1275	-712
	8	-1118	-96	-1300	-801
	9	-1118	584	-1325	-696

APPENDIX B

Table 4: Linear fit between predicted and actual z values at Position 6.

Bivariate Fit of predict By actual

Linear Fit
predict = 774.90635 + 1.581387*actual

Summary of Fit

RSquare	0.758908
RSquare Adj	0.724466
Root Mean Square Error	126.2007
Mean of Response	-779.07
Observations (or Sum Wgts)	9

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	350935.13	350935	22.0345
Error	7	111486.34	15927	**Prob > F**
C. Total	8	462421.47		0.0022*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	774.90635	333.7114	2.32	0.0532
actual	1.581387	0.336889	4.69	0.0022*

Table 5: Linear fit between predicted and actual z values at Position 7.

Bivariate Fit of predict By actual

Linear Fit
predict = 219.29279 + 1.007048*actual

Summary of Fit

RSquare	0.50646
RSquare Adj	0.435954
Root Mean Square Error	148.1158
Mean of Response	-766.943
Observations (or Sum Wgts)	9

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	157588.10	157588	7.1832
Error	7	153568.12	21938	**Prob > F**
C. Total	8	311156.22		0.0315*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	219.29279	371.2742	0.59	0.5733
actual	1.007048	0.375742	2.68	0.0315*

Table 6: Linear fit between predicted and actual z values at Position 8.

Bivariate Fit of predict By actual

Linear Fit
predict = -1054.652 - 0.0862194*actual

Summary of Fit

RSquare	0.003517
RSquare Adj	-0.13884
Root Mean Square Error	226.3599
Mean of Response	-950.317
Observations (or Sum Wgts)	9

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	1265.82	1265.8	0.0247
Error	7	358671.49	51238.8	**Prob > F**
C. Total	8	359937.31		0.8795

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	-1054.652	668.0852	-1.58	0.1584
actual	-0.086219	0.548554	-0.16	0.8795

Table 7: Linear fit between predicted and actual z values at Position 9.

Bivariate Fit of predict By actual

Linear Fit
predict = 7445.703 + 7.3832842*actual

Summary of Fit

RSquare	0.516689
RSquare Adj	0.447644
Root Mean Square Error	1011.055
Mean of Response	5.813618
Observations (or Sum Wgts)	9

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	7649793	7649793	7.4834
Error	7	7155626	1022232	**Prob > F**
C. Total	8	14805420		0.0291*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	7445.703	2740.473	2.72	0.0299*
actual	7.3832842	2.698979	2.74	0.0291*

Table 8: Linear fit between predicted and actual z values at Position 10.

Bivariate Fit of predict By actual

Linear Fit
predict = -328.1422 + 0.477226*actual

Summary of Fit

RSquare	0.488149
RSquare Adj	0.415028
Root Mean Square Error	69.19035
Mean of Response	-802.823
Observations (or Sum Wgts)	9

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	31959.405	31959.4	6.6759
Error	7	33511.136	4787.3	**Prob > F**
C. Total	8	65470.542		0.0363*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	-328.1422	185.1584	-1.77	0.1196
actual	0.477226	0.184701	2.58	0.0363*

Table 9: Linear fit between predicted and actual z values at Position 11.

Bivariate Fit of predict By actual

Linear Fit
predict = -190.0089 + 0.6370341*actual

Summary of Fit

RSquare	0.324079
RSquare Adj	0.227518
Root Mean Square Error	137.0718
Mean of Response	-805.596
Observations (or Sum Wgts)	9

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	63059.20	63059.2	3.3562
Error	7	131520.72	18788.7	**Prob > F**
C. Total	8	194579.92		0.1096

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	-190.0089	339.1109	-0.56	0.5927
actual	0.6370341	0.347725	1.83	0.1096

Table 10: Linear fit between predicted and actual z values at Position 12.

Bivariate Fit of predict By actual

Linear Fit

predict = -1563.632 - 0.5488871*actual

Summary of Fit

RSquare	0.261129
RSquare Adj	0.155576
Root Mean Square Error	144.0028
Mean of Response	-927.411
Observations (or Sum Wgts)	9

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	51301.12	51301.1	2.4739
Error	7	145157.71	20736.8	**Prob > F**
C. Total	8	196458.83		0.1597

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	-1563.632	407.3355	-3.84	0.0064*
actual	-0.548887	0.348972	-1.57	0.1597

APPENDIX C

MATLAB CODE

```
clear all
raw=xlsread('PFtest_15_analog.csv');

%find the index pulse
[M,N]=size(raw);   %longest dimension in array
s=0;
for a=1:M;
    if raw(a,1)>11470;
        if s==0;
            start=a;
            s=1;
        else
            finish=a;
        end
    end
end

%pull out a sample at every encoder pulse
a=1;
old=0;
for x=start:finish;
    if raw(x,2)>11470;
        if x==old+1;
        else
            sensor(a,1)=raw(x,4);
            sensor(a,2)=raw(x,5);
            sensor(a,3)=raw(x,6);
            sensor(a,4)=raw(x,7);
            a=a+1;
        end
        old=x;
    end
end

%pull out a sample at a specified interval
a=1;
x=1;
sensor_condense(a,1)=x;
sensor_condense(a,2)=sensor(x,1);
sensor_condense(a,3)=sensor(x,2);
sensor_condense(a,4)=sensor(x,3);
sensor_condense(a,5)=sensor(x,4);
a=2;
old=0;
for x=1:1000;
    if x==old+18;
        sensor_condense(a,1)=x;
        sensor_condense(a,2)=sensor(x,1);
        sensor_condense(a,3)=sensor(x,2);
        sensor_condense(a,4)=sensor(x,3);
```

```
        sensor_condense(a,5)=sensor(x,4);
        a=a+1;
        old=x;
    end
end

ind1=0;
ind2=0;
ind3=0;
ind4=0;
ind_reject=0;

%convert binary to mm
sensor_condense_mm(:,2:5)=(sensor_condense(:,2:5))*.689697-5300;
%convert encoder pulse to radian
sensor_condense_mm(:,1)=(sensor_condense(:,1))/159.155;

%find x,y,z, coordinates
 [M,N]=size(sensor_condense_mm);
 a=1;
 %sensor 1
 for x=1:M;
     if sensor_condense_mm(x,2)>50;
         if sensor_condense_mm(x,2)<6000;
             H=sensor_condense_mm(x,2)+63.5;  %offset from plate
             sa=0.113446; %fixed sensor angle
             z1=-(H*cos(sa));
             if z1>-2500
                 ind1=1;
                 coord_array1(a,3)=-(H*cos(sa));
                 Z=coord_array1(a,3);
                 delay=0.17;  %(0.33) response delay in radians
                 offset=3.14159264;
                 theta=sensor_condense_mm(x,1)-offset-delay;
                 radius=95.2;    %sensor radius
                 coord_array1(a,1)=((((H^2)-
(Z^2))^0.5)+radius)*sin(theta);
                 coord_array1(a,2)=((((H^2)-
(Z^2))^0.5)+radius)*cos(theta);
                 a=a+1;
             end
         end
     end
 end
%sensor 2
a=1;
for x=1:M;
    if sensor_condense_mm(x,3)>50;
        if sensor_condense_mm(x,3)<6000;
            H=sensor_condense_mm(x,3)+54.0;
            sa=0.340339; %fixed sensor angle
            z2=-(H*cos(sa));
            if z2>-2500
                ind2=1;
                coord_array2(a,3)=-(H*cos(sa));
```

```
                Z=coord_array2(a,3);
                delay=0.17;  %(0.33) response delay in radians
                offset=4.71238898;
                theta=sensor_condense_mm(x,1)-offset-delay;
                radius=106.0;    %sensor radius
                coord_array2(a,1)=((((H^2)-
(Z^2))^0.5)+radius)*sin(theta);
                coord_array2(a,2)=((((H^2)-
(Z^2))^0.5)+radius)*cos(theta);
                a=a+1;
            end
        end
    end
end
 %sensor 3
 a=1;
 b=1;
 for x=1:M;
     if sensor_condense_mm(x,4)>50;
         if sensor_condense_mm(x,4)<6000;   %5840
             H=sensor_condense_mm(x,4)+50.8;
             sa=0.567232; %fixed sensor angle
             z3=-(H*cos(sa));
             if z3>-2500
                 ind3=1;
                 coord_array3(a,3)=-(H*cos(sa));
                 Z=coord_array3(a,3);
                 delay=0.17;  %(0.33) response delay in radians
                 offset=0;
                 theta=sensor_condense_mm(x,1)-offset-delay;
                 radius=115.3;    %sensor radius
                 coord_array3(a,1)=((((H^2)-
(Z^2))^0.5)+radius)*sin(theta);
                 coord_array3(a,2)=((((H^2)-
(Z^2))^0.5)+radius)*cos(theta);
                 a=a+1;
             else
                 ind_reject=1;
                 reject3(b,3)=-(H*cos(sa));
                 Z=reject3(b,3);
                 delay=0.17;  %(0.33) response delay in radians
                 offset=0;
                 theta=sensor_condense_mm(x,1)-offset-delay;
                 radius=115.3;    %sensor radius
                 reject3(b,1)=((((H^2)-(Z^2))^0.5)+radius)*sin(theta);
                 reject3(b,2)=((((H^2)-(Z^2))^0.5)+radius)*cos(theta);
                 b=b+1;
             end
         end
     end
 end
 %sensor 4
 a=1;
 for x=1:M;
     if sensor_condense_mm(x,5)>50;
```

```
        if sensor_condense_mm(x,5)<6000;   %5840
            H=sensor_condense_mm(x,5)+50.8;
            sa=0.7941248; %fixed sensor angle
            z4=-(H*cos(sa));
            if z4>-2500
                ind4=1;
                coord_array4(a,3)=-(H*cos(sa));
                Z=coord_array4(a,3);
                delay=0.17;  %(0.33) response delay in radians
                offset=1.570796;
                theta=sensor_condense_mm(x,1)-offset-delay;
                radius=124.2;    %sensor radius
                coord_array4(a,1)=((((H^2)-
(Z^2))^0.5)+radius)*sin(theta);
                coord_array4(a,2)=((((H^2)-
(Z^2))^0.5)+radius)*cos(theta);
                a=a+1;
            end
        end
    end
end
%    Sensor 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ind1==1
    % FIND HYPOTENUSE
    [M,N]=size(coord_array1);
    for x=1:M

hyp1(x,1)=((coord_array1(x,1))^2+(coord_array1(x,2))^2+(coord_array1(x,
3))^2)^0.5;
    end

    %find frist derivative
    [M,N]=size(hyp1);
    for x=2:M;
        b=x-1;
        hyp1_deriv(b,1)=hyp1(x,1)-hyp1(b,1);
    end

    %find second derivative
    hyp1_deriv2(1,1)=0;
    [M,N]=size(hyp1_deriv);
    for x=2:M;
        b=x-1;
        hyp1_deriv2(b,1)=hyp1_deriv(x,1)-hyp1_deriv(b,1);
    end

    %find peaks
    [M,N]=size(hyp1_deriv2);
    a=1;
    hyp1_peak(1,1)=0;
    for x=1:M;
        if hyp1_deriv2(x,1)>24;
            hyp1_peak(a,1)=x;
            hyp1_peak(a,2)=hyp1_deriv2(x,1);
```

```
        a=a+1;
    else
        if hyp1_deriv2(x,1)<-24;
            hyp1_peak(a,1)=x;
            hyp1_peak(a,2)=hyp1_deriv2(x,1);
            a=a+1;
        end
    end
end

% find pairs, add points to peak_array1
[M,N]=size(hyp1_peak);
hyp1_peak(M+1,1)=0;
hyp1_peak(M+2,1)=0;
a=1;
peak_array1(1,1)=1;
for x=1:M;
    if hyp1_peak(x,1)+1==hyp1_peak(x+1,1);
        if hyp1_peak(x,2)>0;
            if hyp1_peak(x+1,2)<0;
                peak_array1(a,1)=hyp1_peak(x+1,1);
                a=a+1;
            end
        else
            if hyp1_peak(x+1,2)>0;
                peak_array1(a,1)=hyp1_peak(x+1,1);
                a=a+1;
            end
        end
    else
        % if pair is seperated by 2
        if hyp1_peak(x,1)+2==hyp1_peak(x+1,1);
            if hyp1_peak(x,2)>0;
                if hyp1_peak(x+1,2)<0;
                    peak_array1(a,1)=hyp1_peak(x,1)+1;
                    a=a+1;
                end
            else
                if hyp1_peak(x+1,2)>0;
                    peak_array1(a,1)=hyp1_peak(x,1)+1;
                    a=a+1;
                end
            end
        end
        % if pair is seperated by 3
        if hyp1_peak(x,1)+3==hyp1_peak(x+1,1);
            if hyp1_peak(x,2)>0;
                if hyp1_peak(x+1,2)<0;
                    peak_array1(a,1)=hyp1_peak(x,1)+1;
                    a=a+1;
                end
            else
                if hyp1_peak(x+1,2)>0;
                    peak_array1(a,1)=hyp1_peak(x,1)+1;
                    a=a+1;
```

```
                    end
                end
            end
        end
    end

    %%% CONFIGURE PEAK ARRAYS into something usable
    [M,N]=size(peak_array1);
    for x=1:M;
        peak_array1a(x)=peak_array1(x,1);
    end
    peak_array1b=sort(peak_array1a);

end
%    Sensor 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ind2==1
    % FIND HYPOTENUSE
    [M,N]=size(coord_array2);
    for x=1:M

hyp2(x,1)=((coord_array2(x,1))^2+(coord_array2(x,2))^2+(coord_array2(x,
3))^2)^0.5;
    end

    %find frist derivative
    [M,N]=size(hyp2);
    for x=2:M;
        b=x-1;
        hyp2_deriv(b,1)=hyp2(x,1)-hyp2(b,1);
    end

    %find second derivative
    hyp2_deriv2(1,1)=0;
    [M,N]=size(hyp2_deriv);
    for x=2:M;
        b=x-1;
        hyp2_deriv2(b,1)=hyp2_deriv(x,1)-hyp2_deriv(b,1);
    end

    %find peaks
    [M,N]=size(hyp2_deriv2);
    a=1;
    hyp2_peak(1,1)=0;
    for x=1:M;
        if hyp2_deriv2(x,1)>24;
            hyp2_peak(a,1)=x;
            hyp2_peak(a,2)=hyp2_deriv2(x,1);
            a=a+1;
        else
            if hyp2_deriv2(x,1)<-24;
                hyp2_peak(a,1)=x;
                hyp2_peak(a,2)=hyp2_deriv2(x,1);
                a=a+1;
```

```
        end
    end
end

% find pairs, add points to peak_array2
[M,N]=size(hyp2_peak);
hyp2_peak(M+1,1)=0;
hyp2_peak(M+2,1)=0;
a=1;
peak_array2(1,1)=1;
for x=1:M;
    if hyp2_peak(x,1)+1==hyp2_peak(x+1,1);
        if hyp2_peak(x,2)>0;
            if hyp2_peak(x+1,2)<0;
                peak_array2(a,1)=hyp2_peak(x+1,1);
                a=a+1;
            end
        else
            if hyp2_peak(x+1,2)>0;
                peak_array2(a,1)=hyp2_peak(x+1,1);
                a=a+1;
            end
        end
    else
        % if pair is seperated by 2
        if hyp2_peak(x,1)+2==hyp2_peak(x+1,1);
            if hyp2_peak(x,2)>0;
                if hyp2_peak(x+1,2)<0;
                    peak_array2(a,1)=hyp2_peak(x,1)+1;
                    a=a+1;
                end
            else
                if hyp2_peak(x+1,2)>0;
                    peak_array2(a,1)=hyp2_peak(x,1)+1;
                    a=a+1;
                end
            end
        end
        % if pair is seperated by 3
        if hyp2_peak(x,1)+3==hyp2_peak(x+1,1);
            if hyp2_peak(x,2)>0;
                if hyp2_peak(x+1,2)<0;
                    peak_array2(a,1)=hyp2_peak(x,1)+1;
                    a=a+1;
                end
            else
                if hyp2_peak(x+1,2)>0;
                    peak_array2(a,1)=hyp2_peak(x,1)+1;
                    a=a+1;
                end
            end
        end
    end
end
```

```
    %%% CONFIGURE PEAK ARRAYS into something usable
    [M,N]=size(peak_array2);
    for x=1:M;
        peak_array2a(x)=peak_array2(x,1);
    end
    peak_array2b=sort(peak_array2a);

end
%    Sensor 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ind3==1;
    % FIND HYPOTENUSE
    [M,N]=size(coord_array3);
    for x=1:M

hyp3(x,1)=((coord_array3(x,1))^2+(coord_array3(x,2))^2+(coord_array3(x,
3))^2)^0.5;
    end

    %find frist derivative
    [M,N]=size(hyp3);
    for x=2:M;
        b=x-1;
        hyp3_deriv(b,1)=hyp3(x,1)-hyp3(b,1);
    end

    %find second derivative
    hyp3_deriv2(1,1)=0;
    [M,N]=size(hyp3_deriv);
    for x=2:M;
        b=x-1;
        hyp3_deriv2(b,1)=hyp3_deriv(x,1)-hyp3_deriv(b,1);
    end

    %find peaks
    [M,N]=size(hyp3_deriv2);
    a=1;
    hyp3_peak(1,1)=1;
    for x=1:M;
        if hyp3_deriv2(x,1)>24;
            hyp3_peak(a,1)=x;
            hyp3_peak(a,2)=hyp3_deriv2(x,1);
            a=a+1;
        else
            if hyp3_deriv2(x,1)<-24;
                hyp3_peak(a,1)=x;
                hyp3_peak(a,2)=hyp3_deriv2(x,1);
                a=a+1;
            end
        end
    end

    % find pairs, add points to peak_array3
    [M,N]=size(hyp3_peak);
```

```
hyp3_peak(M+1,1)=0;
hyp3_peak(M+2,1)=0;
a=1;
peak_array3(1,1)=0;
for x=1:M;
    if hyp3_peak(x,1)+1==hyp3_peak(x+1,1);
        if hyp3_peak(x,2)>0;
            if hyp3_peak(x+1,2)<0;
                peak_array3(a,1)=hyp3_peak(x+1,1);
                a=a+1;
            end
        else
            if hyp3_peak(x+1,2)>0;
                peak_array3(a,1)=hyp3_peak(x+1,1);
                a=a+1;
            end
        end
    else
        % if pair is seperated by 2
        if hyp3_peak(x,1)+2==hyp3_peak(x+1,1);
            if hyp3_peak(x,2)>0;
                if hyp3_peak(x+1,2)<0;
                    peak_array3(a,1)=hyp3_peak(x,1)+1;
                    a=a+1;
                end
            else
                if hyp3_peak(x+1,2)>0;
                    peak_array3(a,1)=hyp3_peak(x,1)+1;
                    a=a+1;
                end
            end
        end
        % if pair is seperated by 3
        if hyp3_peak(x,1)+3==hyp3_peak(x+1,1);
            if hyp3_peak(x,2)>0;
                if hyp3_peak(x+1,2)<0;
                    peak_array3(a,1)=hyp3_peak(x,1)+1;
                    a=a+1;
                end
            else
                if hyp3_peak(x+1,2)>0;
                    peak_array3(a,1)=hyp3_peak(x,1)+1;
                    a=a+1;
                end
            end
        end
    end
end

%%% CONFIGURE PEAK ARRAYS into something usable
[M,N]=size(peak_array3);
for x=1:M;
    peak_array3a(x)=peak_array3(x,1);
end
peak_array3b=sort(peak_array3a);
```

```
end
%    Sensor 4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ind4==1
    % FIND HYPOTENUSE
    [M,N]=size(coord_array4);
    for x=1:M

hyp4(x,1)=((coord_array4(x,1))^2+(coord_array4(x,2))^2+(coord_array4(x,
3))^2)^0.5;
    end

    %find frist derivative
    [M,N]=size(hyp4);
    for x=2:M;
        b=x-1;
        hyp4_deriv(b,1)=hyp4(x,1)-hyp4(b,1);
    end

    %find second derivative
    hyp4_deriv2(1,1)=0;
    [M,N]=size(hyp4_deriv);
    for x=2:M;
        b=x-1;
        hyp4_deriv2(b,1)=hyp4_deriv(x,1)-hyp4_deriv(b,1);
    end

    %find peaks
    [M,N]=size(hyp4_deriv2);
    a=1;
    hyp4_peak(1,1)=1;
    for x=1:M;
        if hyp4_deriv2(x,1)>24;
            hyp4_peak(a,1)=x;
            hyp4_peak(a,2)=hyp4_deriv2(x,1);
            a=a+1;
        else
            if hyp4_deriv2(x,1)<-24;
                hyp4_peak(a,1)=x;
                hyp4_peak(a,2)=hyp4_deriv2(x,1);
                a=a+1;
            end
        end
    end

    % find pairs, add points to peak_array4
    [M,N]=size(hyp4_peak);
    hyp4_peak(M+1,1)=0;
    hyp4_peak(M+2,1)=0;
    a=1;
    peak_array4(1,1)=0;
    for x=1:M;
        if hyp4_peak(x,1)+1==hyp4_peak(x+1,1);
```

```
            if hyp4_peak(x,2)>0;
                if hyp4_peak(x+1,2)<0;
                    peak_array4(a,1)=hyp4_peak(x+1,1);
                    a=a+1;
                end
            else
                if hyp4_peak(x+1,2)>0;
                    peak_array4(a,1)=hyp4_peak(x+1,1);
                    a=a+1;
                end
            end
        else
            if hyp4_peak(x,1)+2==hyp4_peak(x+1,1);
                if hyp4_peak(x,2)>0;
                    if hyp4_peak(x+1,2)<0;
                        peak_array4(a,1)=hyp4_peak(x+1,1);
                        a=a+1;
                    end
                else
                    if hyp4_peak(x+1,2)>0;
                        peak_array4(a,1)=hyp4_peak(x+1,1);
                        a=a+1;
                    end
                end
            end
            if hyp4_peak(x,1)+3==hyp4_peak(x+1,1);
                if hyp4_peak(x,2)>0;
                    if hyp4_peak(x+1,2)<0;
                        peak_array4(a,1)=hyp4_peak(x+1,1);
                        a=a+1;
                    end
                else
                    if hyp4_peak(x+1,2)>0;
                        peak_array4(a,1)=hyp4_peak(x+1,1);
                        a=a+1;
                    end
                end
            end
        end
    end

    %%% CONFIGURE PEAK ARRAYS into something usable
    [M,N]=size(peak_array4);
    for x=1:M;
        peak_array4a(x)=peak_array4(x,1);
    end
    peak_array4b=sort(peak_array4a);

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%    Pull peak points out to graph
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ind1==1
    [M,N]=size(peak_array1b);
    graph_array(1,1)=coord_array1(1,1);
```

```
    graph_array(1,2)=coord_array1(1,2);
    graph_array(1,3)=coord_array1(1,3);
    a=2;
    for x=1:N;
        b=peak_array1b(x);
        graph_array(a,1)=coord_array1(b,1);
        graph_array(a,2)=coord_array1(b,2);
        graph_array(a,3)=coord_array1(b,3);
        a=a+1;
    end
end

if ind2==1
    [M,N]=size(peak_array2b);
    graph_array(a,1)=coord_array2(1,1);
    graph_array(a,2)=coord_array2(1,2);
    graph_array(a,3)=coord_array2(1,3);
    a=a+1;
    for x=1:N;
        b=peak_array2b(x);
        graph_array(a,1)=coord_array2(b,1);
        graph_array(a,2)=coord_array2(b,2);
        graph_array(a,3)=coord_array2(b,3);
        a=a+1;
    end
end

if ind3==1
    [M,N]=size(peak_array3b);
    [M1,N1]=size(coord_array3);
    graph_array(a,1)=coord_array3(M1,1);
    graph_array(a,2)=coord_array3(M1,2);
    graph_array(a,3)=coord_array3(M1,3);
    a=a+1;
    for x=1:N;
        b=peak_array3b(x);
        graph_array(a,1)=coord_array3(b,1);
        graph_array(a,2)=coord_array3(b,2);
        graph_array(a,3)=coord_array3(b,3);
        a=a+1;
    end
end

if ind4==1
    [M,N]=size(peak_array4b);
    [M1,N1]=size(coord_array4);
    graph_array(a,1)=coord_array4(M1,1);
    graph_array(a,2)=coord_array4(M1,2);
    graph_array(a,3)=coord_array4(M1,3);
    a=a+1;
    for x=1:N;
        b=peak_array4b(x);
        graph_array(a,1)=coord_array4(b,1);
        graph_array(a,2)=coord_array4(b,2);
        graph_array(a,3)=coord_array4(b,3);
```

```
        a=a+1;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% IDENTIFY groups
%%%%%%%%%%%%%%%%%%%%%%%%%%%%  1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ind1==1
    %assign group numbers
    [M,N]=size(peak_array1b);
    [M1,N1]=size(coord_array1);
    b=1;
    c=101;
    for x=1:M1
        group_array1(x,1)=coord_array1(x,1);
        group_array1(x,2)=coord_array1(x,2);
        group_array1(x,3)=coord_array1(x,3);
        group_array1(x,4)=c;
        if b==N+1
        else
            if peak_array1b(b)==x;
                c=c+1;
                b=b+1;
            end
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%  2   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ind2==1
    %assign group numbers
    [M,N]=size(peak_array2b);
    [M1,N1]=size(coord_array2);
    b=1;
    c=201;
    for x=1:M1
        group_array2(x,1)=coord_array2(x,1);
        group_array2(x,2)=coord_array2(x,2);
        group_array2(x,3)=coord_array2(x,3);
        group_array2(x,4)=c;
        if b==N+1
        else
            if peak_array2b(b)==x;
                c=c+1;
                b=b+1;
            end
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%  3   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ind3==1
    %assign group numbers
    [M,N]=size(peak_array3b);
    [M1,N1]=size(coord_array3);
    b=1;
    c=301;
```

```
    for x=1:M1
        group_array3(x,1)=coord_array3(x,1);
        group_array3(x,2)=coord_array3(x,2);
        group_array3(x,3)=coord_array3(x,3);
        group_array3(x,4)=c;
        if b==N+1
        else
            if peak_array3b(b)==x;
                c=c+1;
                b=b+1;
            end
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    4    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ind4==1
    %assign group numbers
    [M,N]=size(peak_array4b);
    [M1,N1]=size(coord_array4);
    b=1;
    c=401;
    for x=1:M1
        group_array4(x,1)=coord_array4(x,1);
        group_array4(x,2)=coord_array4(x,2);
        group_array4(x,3)=coord_array4(x,3);
        group_array4(x,4)=c;
        if b==N+1
        else
            if peak_array4b(b)==x;
                c=c+1;
                b=b+1;
            end
        end
    end
end

%%% check for erroneous points (sensor 3 dropped points)
if ind_reject==1
    reject3_2=sort(reject3);
    [M,N]=size(reject3_2);
    max_x=reject3_2(M,1);
    min_x=reject3_2(1,1);
    max_y=reject3_2(M,2);
    min_y=reject3_2(1,2);

    x_range=max_x-min_x;
    y_range=max_y-min_y;

    if x_range>y_range
        group_array3_2=sort(group_array3);
        group_min_y=group_array3_2(1,2);
        [M,N]=size(group_array4);
        a=1;
        for x=1:M
            if group_array4(x,2)>group_min_y
```

```
                group_array4_buffer(a,1)=group_array4(x,1);
                group_array4_buffer(a,2)=group_array4(x,2);
                group_array4_buffer(a,3)=group_array4(x,3);
                group_array4_buffer(a,4)=group_array4(x,4);
                a=a+1;
            end
        end

        clear group_array4
        group_array4=group_array4_buffer;
        clear coord_array4
        coord_array4(:,1)=group_array4(:,1);
        coord_array4(:,2)=group_array4(:,2);
        coord_array4(:,3)=group_array4(:,3);
    end

    if y_range>x_range
        group_array3_2=sort(group_array3);
        [M,N]=size(group_array3_2);
        group_max_x=group_array3_2(M,1);
        [M,N]=size(group_array4);
        a=1;
        for x=1:M
            if group_array4(x,1)<group_max_x
                group_array4_buffer(a,1)=group_array4(x,1);
                group_array4_buffer(a,2)=group_array4(x,2);
                group_array4_buffer(a,3)=group_array4(x,3);
                group_array4_buffer(a,4)=group_array4(x,4);
                a=a+1;
            end
        end

        clear group_array4
        group_array4=group_array4_buffer;
        clear coord_array4
        coord_array4(:,1)=group_array4(:,1);
        coord_array4(:,2)=group_array4(:,2);
        coord_array4(:,3)=group_array4(:,3);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%COMBINE individual sensors into one GROUP array
a=1;
if ind1==1
    [M,N]=size(group_array1);
    for x=1:M
        group_array(a,1)=group_array1(x,1);
        group_array(a,2)=group_array1(x,2);
        group_array(a,3)=group_array1(x,3);
        group_array(a,4)=group_array1(x,4);
        a=a+1;
    end
end
if ind2==1
```

```
    [M,N]=size(group_array2);
    for x=1:M
        group_array(a,1)=group_array2(x,1);
        group_array(a,2)=group_array2(x,2);
        group_array(a,3)=group_array2(x,3);
        group_array(a,4)=group_array2(x,4);
        a=a+1;
    end
end
if ind3==1
    [M,N]=size(group_array3);
    for x=1:M
        group_array(a,1)=group_array3(x,1);
        group_array(a,2)=group_array3(x,2);
        group_array(a,3)=group_array3(x,3);
        group_array(a,4)=group_array3(x,4);
        a=a+1;
    end
end
if ind4==1
    [M,N]=size(group_array4);
    for x=1:M
        group_array(a,1)=group_array4(x,1);
        group_array(a,2)=group_array4(x,2);
        group_array(a,3)=group_array4(x,3);
        group_array(a,4)=group_array4(x,4);
        a=a+1;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
if ind1==1
    scatter3(coord_array1(:,1),coord_array1(:,2),coord_array1(:,3));
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    grid on
    hold on;
end
if ind2==1
scatter3(coord_array2(:,1),coord_array2(:,2),coord_array2(:,3));
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    grid on
    hold on;
end
if ind3==1
scatter3(coord_array3(:,1),coord_array3(:,2),coord_array3(:,3));
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    grid on
    hold on;
end
```

```
if ind4==1
scatter3(coord_array4(:,1),coord_array4(:,2),coord_array4(:,3));
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    grid on
    hold on;
end
scatter3(graph_array(:,1),graph_array(:,2),graph_array(:,3),'filled');
```

```
Start new M-file

clear all
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
           num=4;
input=load('pos4.mat');
raw=input.group_array;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
max=max(raw);
min=min(raw);

xrange=max(1,1)-min(1,1);
yrange=max(1,2)-min(1,2);

deltax=xrange/16+.07;  %0.07 makes last group +1 beyond last point
deltay=yrange/16+.07;  %0.07 makes last group +1 beyond last point

xmin=min(1,1)-1;
ymin=min(1,2)-1;

xmax=max(1,1)+1;    %for plotting
ymax=max(1,2)+1;    %for plotting

[M,N]=size(raw);
for x=1:M
    raw(x,5)=x;
end
%%%%%%%%   GROUPS IN X DIRECTION    %%%%%%%%%%%
for x=1:M
    if raw(x,1)>xmin
        if raw(x,1)<xmin+deltax
            raw(x,6)=1;
        else
            if raw(x,1)<xmin+2*deltax
                raw(x,6)=2;
            else
                if raw(x,1)<xmin+3*deltax
                    raw(x,6)=3;
                else
                    if raw(x,1)<xmin+4*deltax
                        raw(x,6)=4;
                    else
                        if raw(x,1)<xmin+5*deltax
                            raw(x,6)=5;
```

```
                        else
                            if raw(x,1)<xmin+6*deltax
                                raw(x,6)=6;
                            else
                                if raw(x,1)<xmin+7*deltax
                                    raw(x,6)=7;
                                else
                                    if raw(x,1)<xmin+8*deltax
                                        raw(x,6)=8;
                                    else
                                        if raw(x,1)<xmin+9*deltax
                                            raw(x,6)=9;
                                        else
                                            if raw(x,1)<xmin+10*deltax
                                                raw(x,6)=10;
                                            else
                                                if
raw(x,1)<xmin+11*deltax
                                                    raw(x,6)=11;
                                                else
                                                    if
raw(x,1)<xmin+12*deltax
                                                        raw(x,6)=12;
                                                    else
                                                        if
raw(x,1)<xmin+13*deltax

raw(x,6)=13;
                                                        else
                                                            if
raw(x,1)<xmin+14*deltax

raw(x,6)=14;
                                                            else
                                                                if
raw(x,1)<xmin+15*deltax

raw(x,6)=15;
                                                                else
                                                                    if
raw(x,1)<xmin+16*deltax

raw(x,6)=16;
                                                                end
                                                            end
                                                        end
                                                    end
                                                end
                                            end
                                        end
                                    end
                                end
                            end
                        end
```

```
                    end
                end
            end
        end
    end
end


%%%%%%%%%   GROUPS IN Y DIRECTION    %%%%%%%%%%%%

for x=1:M
    if raw(x,2)>ymin
        if raw(x,2)<ymin+deltay
            raw(x,7)=1;
        else
            if raw(x,2)<ymin+2*deltay
                raw(x,7)=2;
            else
                if raw(x,2)<ymin+3*deltay
                    raw(x,7)=3;
                else
                    if raw(x,2)<ymin+4*deltay
                        raw(x,7)=4;
                    else
                        if raw(x,2)<ymin+5*deltay
                            raw(x,7)=5;
                        else
                            if raw(x,2)<ymin+6*deltay
                                raw(x,7)=6;
                            else
                                if raw(x,2)<ymin+7*deltay
                                    raw(x,7)=7;
                                else
                                    if raw(x,2)<ymin+8*deltay
                                        raw(x,7)=8;
                                    else
                                        if raw(x,2)<ymin+9*deltay
                                            raw(x,7)=9;
                                        else
                                            if raw(x,2)<ymin+10*deltay
                                                raw(x,7)=10;
                                            else
                                                if
raw(x,2)<ymin+11*deltay
                                                    raw(x,7)=11;
                                                else
                                                    if
raw(x,2)<ymin+12*deltay
                                                        raw(x,7)=12;
                                                    else
                                                        if
raw(x,2)<ymin+13*deltay

raw(x,7)=13;
                                                        else
```

```
                                                            if
raw(x,2)<ymin+14*deltay

raw(x,7)=14;
                                                            else
                                                                if
raw(x,2)<ymin+15*deltay

raw(x,7)=15;
                                                                else
                                                                    if
raw(x,2)<ymin+16*deltay

raw(x,7)=16;
                                                                    end
                                                                end
                                                            end
                                                        end
                                                    end
                                                end
                                            end
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%     Pull points out into each row or column
a=1;
b=1;
c=1;
d=1;
e=1;
f=1;
g=1;
h=1;
i=1;
j=1;
k=1;
l=1;
m=1;
n=1;
o=1;
p=1;

column1(1,7)=0;
column2(1,7)=0;
column3(1,7)=0;
```

```
column4(1,7)=0;
column5(1,7)=0;
column6(1,7)=0;
column7(1,7)=0;
column8(1,7)=0;
column9(1,7)=0;
column10(1,7)=0;
column11(1,7)=0;
column12(1,7)=0;
column13(1,7)=0;
column14(1,7)=0;
column15(1,7)=0;
column16(1,7)=0;

for x=1:M
    if raw(x,6)==1
       column1(a,:)=raw(x,:);
       a=a+1;
    end
    if raw(x,6)==2
       column2(b,:)=raw(x,:);
       b=b+1;
    end
    if raw(x,6)==3
       column3(c,:)=raw(x,:);
       c=c+1;
    end
    if raw(x,6)==4
       column4(d,:)=raw(x,:);
       d=d+1;
    end
    if raw(x,6)==5
       column5(e,:)=raw(x,:);
       e=e+1;
    end
    if raw(x,6)==6
       column6(f,:)=raw(x,:);
       f=f+1;
    end
    if raw(x,6)==7
       column7(g,:)=raw(x,:);
       g=g+1;
    end
    if raw(x,6)==8
       column8(h,:)=raw(x,:);
       h=h+1;
    end
    if raw(x,6)==9
       column9(i,:)=raw(x,:);
       i=i+1;
    end
    if raw(x,6)==10
       column10(j,:)=raw(x,:);
       j=j+1;
    end
```

```
    if raw(x,6)==11
       column11(k,:)=raw(x,:);
       k=k+1;
    end
    if raw(x,6)==12
       column12(l,:)=raw(x,:);
       l=l+1;
    end
    if raw(x,6)==13
       column13(m,:)=raw(x,:);
       m=m+1;
    end
    if raw(x,6)==14
       column14(n,:)=raw(x,:);
       n=n+1;
    end
    if raw(x,6)==15
       column15(o,:)=raw(x,:);
       o=o+1;
    end
    if raw(x,6)==16
       column16(p,:)=raw(x,:);
       p=p+1;
    end
end

a=1;
b=1;
c=1;
d=1;
e=1;
f=1;
g=1;
h=1;
i=1;
j=1;
k=1;
l=1;
m=1;
n=1;
o=1;
p=1;

row1(1,7)=0;
row2(1,7)=0;
row3(1,7)=0;
row4(1,7)=0;
row5(1,7)=0;
row6(1,7)=0;
row7(1,7)=0;
row8(1,7)=0;
row9(1,7)=0;
row10(1,7)=0;
row11(1,7)=0;
row12(1,7)=0;
```

```
row13(1,7)=0;
row14(1,7)=0;
row15(1,7)=0;
row16(1,7)=0;

for x=1:M
    if raw(x,7)==1
       row1(a,:)=raw(x,:);
       a=a+1;
    end
    if raw(x,7)==2
       row2(b,:)=raw(x,:);
       b=b+1;
    end
    if raw(x,7)==3
       row3(c,:)=raw(x,:);
       c=c+1;
    end
    if raw(x,7)==4
       row4(d,:)=raw(x,:);
       d=d+1;
    end
    if raw(x,7)==5
       row5(e,:)=raw(x,:);
       e=e+1;
    end
    if raw(x,7)==6
       row6(f,:)=raw(x,:);
       f=f+1;
    end
    if raw(x,7)==7
       row7(g,:)=raw(x,:);
       g=g+1;
    end
    if raw(x,7)==8
       row8(h,:)=raw(x,:);
       h=h+1;
    end
    if raw(x,7)==9
       row9(i,:)=raw(x,:);
       i=i+1;
    end
    if raw(x,7)==10
       row10(j,:)=raw(x,:);
       j=j+1;
    end
    if raw(x,7)==11
       row11(k,:)=raw(x,:);
       k=k+1;
    end
    if raw(x,7)==12
       row12(l,:)=raw(x,:);
       l=l+1;
    end
    if raw(x,7)==13
```

```
        row13(m,:)=raw(x,:);
        m=m+1;
    end
    if raw(x,7)==14
        row14(n,:)=raw(x,:);
        n=n+1;
    end
    if raw(x,7)==15
        row15(o,:)=raw(x,:);
        o=o+1;
    end
    if raw(x,7)==16
        row16(p,:)=raw(x,:);
        p=p+1;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%% SORT ACCORDING TO Y (for columns), X (for rows)
column1=sortrows(column1,2);
column2=sortrows(column2,2);
column3=sortrows(column3,2);
column4=sortrows(column4,2);
column5=sortrows(column5,2);
column6=sortrows(column6,2);
column7=sortrows(column7,2);
column8=sortrows(column8,2);

column9=sortrows(column9,2);
column10=sortrows(column10,2);
column11=sortrows(column11,2);
column12=sortrows(column12,2);
column13=sortrows(column13,2);
column14=sortrows(column14,2);
column15=sortrows(column15,2);
column16=sortrows(column16,2);

row1=sortrows(row1,1);
row2=sortrows(row2,1);
row3=sortrows(row3,1);
row4=sortrows(row4,1);
row5=sortrows(row5,1);
row6=sortrows(row6,1);
row7=sortrows(row7,1);
row8=sortrows(row8,1);

row9=sortrows(row9,1);
row10=sortrows(row10,1);
row11=sortrows(row11,1);
row12=sortrows(row12,1);
row13=sortrows(row13,1);
row14=sortrows(row14,1);
row15=sortrows(row15,1);
row16=sortrows(row16,1);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%   FIND DERIVATIVE (dz/dy for column, dz/dx for row)
column1(1,8)=0;
[M,N]=size(column1);
for x=2:M
    column1(x,8)=(column1(x,3)-column1(x-1,3))/(column1(x,2)-column1(x-
1,2));
end
column2(1,8)=0;
[M,N]=size(column2);
for x=2:M
    column2(x,8)=(column2(x,3)-column2(x-1,3))/(column2(x,2)-column2(x-
1,2));
end
column3(1,8)=0;
[M,N]=size(column3);
for x=2:M
    column3(x,8)=(column3(x,3)-column3(x-1,3))/(column3(x,2)-column3(x-
1,2));
end
column4(1,8)=0;
[M,N]=size(column4);
for x=2:M
    column4(x,8)=(column4(x,3)-column4(x-1,3))/(column4(x,2)-column4(x-
1,2));
end
column5(1,8)=0;
[M,N]=size(column5);
for x=2:M
    column5(x,8)=(column5(x,3)-column5(x-1,3))/(column5(x,2)-column5(x-
1,2));
end
column6(1,8)=0;
[M,N]=size(column6);
for x=2:M
    column6(x,8)=(column6(x,3)-column6(x-1,3))/(column6(x,2)-column6(x-
1,2));
end
column7(1,8)=0;
[M,N]=size(column7);
for x=2:M
    column7(x,8)=(column7(x,3)-column7(x-1,3))/(column7(x,2)-column7(x-
1,2));
end
column8(1,8)=0;
[M,N]=size(column8);
for x=2:M
    column8(x,8)=(column8(x,3)-column8(x-1,3))/(column8(x,2)-column8(x-
1,2));
end
column9(1,8)=0;
[M,N]=size(column9);
for x=2:M
```

```
    column9(x,8)=(column9(x,3)-column9(x-1,3))/(column9(x,2)-column9(x-
1,2));
end
column10(1,8)=0;
[M,N]=size(column10);
for x=2:M
    column10(x,8)=(column10(x,3)-column10(x-1,3))/(column10(x,2)-
column10(x-1,2));
end
column11(1,8)=0;
[M,N]=size(column11);
for x=2:M
    column11(x,8)=(column11(x,3)-column11(x-1,3))/(column11(x,2)-
column11(x-1,2));
end
column12(1,8)=0;
[M,N]=size(column12);
for x=2:M
    column12(x,8)=(column12(x,3)-column12(x-1,3))/(column12(x,2)-
column12(x-1,2));
end
column13(1,8)=0;
[M,N]=size(column13);
for x=2:M
    column13(x,8)=(column13(x,3)-column13(x-1,3))/(column13(x,2)-
column13(x-1,2));
end
column14(1,8)=0;
[M,N]=size(column14);
for x=2:M
    column14(x,8)=(column14(x,3)-column14(x-1,3))/(column14(x,2)-
column14(x-1,2));
end
column15(1,8)=0;
[M,N]=size(column15);
for x=2:M
    column15(x,8)=(column15(x,3)-column15(x-1,3))/(column15(x,2)-
column15(x-1,2));
end
column16(1,8)=0;
[M,N]=size(column16);
for x=2:M
    column16(x,8)=(column16(x,3)-column16(x-1,3))/(column16(x,2)-
column16(x-1,2));
end
%%%%%%%%%%%%%%%%%%%
row1(1,8)=0;
[M,N]=size(row1);
for x=2:M
    row1(x,8)=(row1(x,3)-row1(x-1,3))/(row1(x,1)-row1(x-1,1));
end
row2(1,8)=0;
[M,N]=size(row2);
for x=2:M
    row2(x,8)=(row2(x,3)-row2(x-1,3))/(row2(x,1)-row2(x-1,1));
```

```
end
row3(1,8)=0;
[M,N]=size(row3);
for x=2:M
    row3(x,8)=(row3(x,3)-row3(x-1,3))/(row3(x,1)-row3(x-1,1));
end
row4(1,8)=0;
[M,N]=size(row4);
for x=2:M
    row4(x,8)=(row4(x,3)-row4(x-1,3))/(row4(x,1)-row4(x-1,1));
end
row5(1,8)=0;
[M,N]=size(row5);
for x=2:M
    row5(x,8)=(row5(x,3)-row5(x-1,3))/(row5(x,1)-row5(x-1,1));
end
row6(1,8)=0;
[M,N]=size(row6);
for x=2:M
    row6(x,8)=(row6(x,3)-row6(x-1,3))/(row6(x,1)-row6(x-1,1));
end
row7(1,8)=0;
[M,N]=size(row7);
for x=2:M
    row7(x,8)=(row7(x,3)-row7(x-1,3))/(row7(x,1)-row7(x-1,1));
end
row8(1,8)=0;
[M,N]=size(row8);
for x=2:M
    row8(x,8)=(row8(x,3)-row8(x-1,3))/(row8(x,1)-row8(x-1,1));
end
row9(1,8)=0;
[M,N]=size(row9);
for x=2:M
    row9(x,8)=(row9(x,3)-row9(x-1,3))/(row9(x,1)-row9(x-1,1));
end
row10(1,8)=0;
[M,N]=size(row10);
for x=2:M
    row10(x,8)=(row10(x,3)-row10(x-1,3))/(row10(x,1)-row10(x-1,1));
end
row11(1,8)=0;
[M,N]=size(row11);
for x=2:M
    row11(x,8)=(row11(x,3)-row11(x-1,3))/(row11(x,1)-row11(x-1,1));
end
row12(1,8)=0;
[M,N]=size(row12);
for x=2:M
    row12(x,8)=(row12(x,3)-row12(x-1,3))/(row12(x,1)-row12(x-1,1));
end
row13(1,8)=0;
[M,N]=size(row13);
for x=2:M
    row13(x,8)=(row13(x,3)-row13(x-1,3))/(row13(x,1)-row13(x-1,1));
```

```
end
row14(1,8)=0;
[M,N]=size(row14);
for x=2:M
    row14(x,8)=(row14(x,3)-row14(x-1,3))/(row14(x,1)-row14(x-1,1));
end
row15(1,8)=0;
[M,N]=size(row15);
for x=2:M
    row15(x,8)=(row15(x,3)-row15(x-1,3))/(row15(x,1)-row15(x-1,1));
end
row16(1,8)=0;
[M,N]=size(row16);
for x=2:M
    row16(x,8)=(row16(x,3)-row16(x-1,3))/(row16(x,1)-row16(x-1,1));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%  Combine column gradients into one array and plot
%%%  COLUMNS
[M,N]=size(column1);
a=1;
for x=1:M
    combine_col(a,:)=column1(x,:);
    a=a+1;
end
[M,N]=size(column2);
for x=1:M
    combine_col(a,:)=column2(x,:);
    a=a+1;
end
[M,N]=size(column3);
for x=1:M
    combine_col(a,:)=column3(x,:);
    a=a+1;
end
[M,N]=size(column4);
for x=1:M
    combine_col(a,:)=column4(x,:);
    a=a+1;
end
[M,N]=size(column5);
for x=1:M
    combine_col(a,:)=column5(x,:);
    a=a+1;
end
[M,N]=size(column6);
for x=1:M
    combine_col(a,:)=column6(x,:);
    a=a+1;
end
[M,N]=size(column7);
for x=1:M
    combine_col(a,:)=column7(x,:);
    a=a+1;
```

```
end
[M,N]=size(column8);
for x=1:M
    combine_col(a,:)=column8(x,:);
    a=a+1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[M,N]=size(column9);
for x=1:M
    combine_col(a,:)=column9(x,:);
    a=a+1;
end
[M,N]=size(column10);
for x=1:M
    combine_col(a,:)=column10(x,:);
    a=a+1;
end
[M,N]=size(column11);
for x=1:M
    combine_col(a,:)=column11(x,:);
    a=a+1;
end
[M,N]=size(column12);
for x=1:M
    combine_col(a,:)=column12(x,:);
    a=a+1;
end
[M,N]=size(column13);
for x=1:M
    combine_col(a,:)=column13(x,:);
    a=a+1;
end
[M,N]=size(column14);
for x=1:M
    combine_col(a,:)=column14(x,:);
    a=a+1;
end
[M,N]=size(column15);
for x=1:M
    combine_col(a,:)=column15(x,:);
    a=a+1;
end
[M,N]=size(column16);
for x=1:M
    combine_col(a,:)=column16(x,:);
    a=a+1;
end

%%%  ROWS
[M,N]=size(row1);
a=1;
for x=1:M
    combine_row(a,:)=row1(x,:);
    a=a+1;
end
```

```
[M,N]=size(row2);
for x=1:M
    combine_row(a,:)=row2(x,:);
    a=a+1;
end
[M,N]=size(row3);
for x=1:M
    combine_row(a,:)=row3(x,:);
    a=a+1;
end
[M,N]=size(row4);
for x=1:M
    combine_row(a,:)=row4(x,:);
    a=a+1;
end
[M,N]=size(row5);
for x=1:M
    combine_row(a,:)=row5(x,:);
    a=a+1;
end
[M,N]=size(row6);
for x=1:M
    combine_row(a,:)=row6(x,:);
    a=a+1;
end
[M,N]=size(row7);
for x=1:M
    combine_row(a,:)=row7(x,:);
    a=a+1;
end
[M,N]=size(row8);
for x=1:M
    combine_row(a,:)=row8(x,:);
    a=a+1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[M,N]=size(row9);
for x=1:M
    combine_row(a,:)=row9(x,:);
    a=a+1;
end
[M,N]=size(row10);
for x=1:M
    combine_row(a,:)=row10(x,:);
    a=a+1;
end
[M,N]=size(row11);
for x=1:M
    combine_row(a,:)=row11(x,:);
    a=a+1;
end
[M,N]=size(row12);
for x=1:M
    combine_row(a,:)=row12(x,:);
    a=a+1;
```

```
end
[M,N]=size(row13);
for x=1:M
    combine_row(a,:)=row13(x,:);
    a=a+1;
end
[M,N]=size(row14);
for x=1:M
    combine_row(a,:)=row14(x,:);
    a=a+1;
end
[M,N]=size(row15);
for x=1:M
    combine_row(a,:)=row15(x,:);
    a=a+1;
end
[M,N]=size(row16);
for x=1:M
    combine_row(a,:)=row16(x,:);
    a=a+1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%  combine groups using GRADIENT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  COLUMNS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[M,N]=size(combine_col);
a=1;
b=1;
for x=2:M
    if combine_col(x,8)>.42
        if combine_col(x,6)==combine_col(x-1,6) %group edge
            non_match(a,1)=combine_col(x,4);
            non_match(a,2)=combine_col(x-1,4);
            a=a+1;
        end
    else
        if combine_col(x,8)<-.42
            if combine_col(x,6)==combine_col(x-1,6) %group edge
                non_match(a,1)=combine_col(x,4);
                non_match(a,2)=combine_col(x-1,4);
                a=a+1;
            end
        else
            if combine_col(x,6)==combine_col(x-1,6) %group edge
                match(b,1)=combine_col(x,4);
                match(b,2)=combine_col(x-1,4);
                b=b+1;
            end
        end
    end
end
%%% Eliminate group pairs (401,401)
match1(1,1)=0;
match1(1,2)=0;
```

```
[M,N]=size(match);
a=1;
for x=1:M
    if match(x,1)==match(x,2)
    else
        match1(a,1)=match(x,1);
        match1(a,2)=match(x,2);
        a=a+1;
    end
end

non_match1(1,1)=0;
non_match1(1,2)=0;
[M,N]=size(non_match);
a=1;
for x=1:M
    if non_match(x,1)==non_match(x,2)
    else
        non_match1(a,1)=non_match(x,1);
        non_match1(a,2)=non_match(x,2);
        a=a+1;
    end
end
%%% Sort each row (lower group number first)
[M,N]=size(match1);
a=1;

for x=1:M
    if match1(x,1)>match1(x,2)
        match2(a,1)=match1(x,2);
        match2(a,2)=match1(x,1);
        a=a+1;
    else
        match2(a,1)=match1(x,1);
        match2(a,2)=match1(x,2);
        a=a+1;
    end
end

[M,N]=size(non_match1);
a=1;

for x=1:M
    if non_match1(x,1)>non_match1(x,2)
        non_match2(a,1)=non_match1(x,2);
        non_match2(a,2)=non_match1(x,1);
        a=a+1;
    else
        non_match2(a,1)=non_match1(x,1);
        non_match2(a,2)=non_match1(x,2);
        a=a+1;
    end
end
%%% Sort by first row
```

```
match2=sortrows(match2,1);
non_match2=sortrows(non_match2,1);
%%% Combine pairs
[M,N]=size(match2);
a=2;
b=0;
row=0;
match3(1,1)=match2(1,1);
match3(1,2)=match2(1,2);
match3(1,3)=1;

for x=2:1:M
    [M1,N1]=size(match3);
    for y=1:1:M1
        if match3(y,1)==match2(x,1)
            if match3(y,2)==match2(x,2)
                row=y;
                b=1;
            end
        end
    end

    if b==1;
        match3(row,3)=match3(row,3)+1;
        b=0;
    else
        match3(a,1)=match2(x,1);
        match3(a,2)=match2(x,2);
        match3(a,3)=1;
        a=a+1;
        b=0;
    end
end

[M,N]=size(non_match2);
a=2;
b=0;
row=0;
non_match3(1,1)=non_match2(1,1);
non_match3(1,2)=non_match2(1,2);
non_match3(1,3)=1;

for x=2:1:M
    [M1,N1]=size(non_match3);
    for y=1:1:M1
        if non_match3(y,1)==non_match2(x,1)
            if non_match3(y,2)==non_match2(x,2)
                row=y;
                b=1;
            end
        end
    end

    if b==1;
```

```
        non_match3(row,3)=non_match3(row,3)+1;
        b=0;
    else
        non_match3(a,1)=non_match2(x,1);
        non_match3(a,2)=non_match2(x,2);
        non_match3(a,3)=1;
        a=a+1;
        b=0;
    end
end
%%%%%%%%%%%%%%%%%%   REPEAT PROCEDURE ON ROWS
%%%%%%%%%%%%%%%%%%%%%%%%
[M,N]=size(combine_row);
a=1;
b=1;
for x=2:M
    if combine_row(x,8)>.42  %.65
        if combine_row(x,7)==combine_row(x-1,7) %group edge
            rnon_match(a,1)=combine_row(x,4);
            rnon_match(a,2)=combine_row(x-1,4);
            a=a+1;
        end
    else
        if combine_row(x,8)<-.42    %.65
            if combine_row(x,7)==combine_row(x-1,7) %group edge
                rnon_match(a,1)=combine_row(x,4);
                rnon_match(a,2)=combine_row(x-1,4);
                a=a+1;
            end
        else
            if combine_row(x,7)==combine_row(x-1,7) %group edge
                if abs(combine_row(x,1)-combine_row(x-1,1))<1000
                    rmatch(b,1)=combine_row(x,4);
                    rmatch(b,2)=combine_row(x-1,4);
                    b=b+1;
                end
            end
        end
    end
end
%%% Eliminate group pairs (401,401)
rmatch1(1,1)=0;
rmatch1(1,2)=0;
[M,N]=size(rmatch);
a=1;

for x=1:M
    if rmatch(x,1)==rmatch(x,2)
    else
        rmatch1(a,1)=rmatch(x,1);
        rmatch1(a,2)=rmatch(x,2);
        a=a+1;
    end
end
```

```
rnon_match1(1,1)=0;
rnon_match1(1,2)=0;
[M,N]=size(rnon_match);
a=1;

for x=1:M
    if rnon_match(x,1)==rnon_match(x,2)
    else
        rnon_match1(a,1)=rnon_match(x,1);
        rnon_match1(a,2)=rnon_match(x,2);
        a=a+1;
    end
end
%%% Sort each row (lower group number first)
[M,N]=size(rmatch1);
a=1;

for x=1:M
    if rmatch1(x,1)>rmatch1(x,2)
        rmatch2(a,1)=rmatch1(x,2);
        rmatch2(a,2)=rmatch1(x,1);
        a=a+1;
    else
        rmatch2(a,1)=rmatch1(x,1);
        rmatch2(a,2)=rmatch1(x,2);
        a=a+1;
    end
end

[M,N]=size(rnon_match1);
a=1;

for x=1:M
    if rnon_match1(x,1)>rnon_match1(x,2)
        rnon_match2(a,1)=rnon_match1(x,2);
        rnon_match2(a,2)=rnon_match1(x,1);
        a=a+1;
    else
        rnon_match2(a,1)=rnon_match1(x,1);
        rnon_match2(a,2)=rnon_match1(x,2);
        a=a+1;
    end
end
%%% Sort by first row
rmatch2=sortrows(rmatch2,1);
rnon_match2=sortrows(rnon_match2,1);
%%% Combine pairs
[M,N]=size(rmatch2);
a=2;
b=0;
row=0;
rmatch3(1,1)=rmatch2(1,1);
rmatch3(1,2)=rmatch2(1,2);
rmatch3(1,3)=1;
```

```
for x=2:1:M
    [M1,N1]=size(rmatch3);
    for y=1:1:M1
        if rmatch3(y,1)==rmatch2(x,1)
            if rmatch3(y,2)==rmatch2(x,2)
                row=y;
                b=1;
            end
        end
    end

    if b==1;
        rmatch3(row,3)=rmatch3(row,3)+1;
        b=0;
    else
        rmatch3(a,1)=rmatch2(x,1);
        rmatch3(a,2)=rmatch2(x,2);
        rmatch3(a,3)=1;
        a=a+1;
        b=0;
    end
end

[M,N]=size(rnon_match2);
a=2;
b=0;
row=0;
rnon_match3(1,1)=rnon_match2(1,1);
rnon_match3(1,2)=rnon_match2(1,2);
rnon_match3(1,3)=1;

for x=2:1:M
    [M1,N1]=size(rnon_match3);
    for y=1:1:M1
        if rnon_match3(y,1)==rnon_match2(x,1)
            if rnon_match3(y,2)==rnon_match2(x,2)
                row=y;
                b=1;
            end
        end
    end

    if b==1;
        rnon_match3(row,3)=rnon_match3(row,3)+1;
        b=0;
    else
        rnon_match3(a,1)=rnon_match2(x,1);
        rnon_match3(a,2)=rnon_match2(x,2);
        rnon_match3(a,3)=1;
        a=a+1;
        b=0;
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%%%%    combine match and non-match from rows and columns
%%%%%%%%
combine_match(:,1)=match3(:,1);
combine_match(:,2)=match3(:,2);
combine_match(:,3)=match3(:,3);

[M,N]=size(rmatch3);
[M1,N1]=size(combine_match);
a=M1+1;
b=0;
row=0;

for x=1:1:M
    for y=1:1:M1
        if combine_match(y,1)==rmatch3(x,1)
            if combine_match(y,2)==rmatch3(x,2)
                row=y;
                b=1;
            end
        end
    end

    if b==1;
        combine_match(row,3)=combine_match(row,3)+rmatch3(x,3);
        b=0;
    else
        combine_match(a,1)=rmatch3(x,1);
        combine_match(a,2)=rmatch3(x,2);
        combine_match(a,3)=rmatch3(x,3);
        a=a+1;
        b=0;
    end
end
%%% REPEAT FOR ROWS
combine_non_match(:,1)=non_match3(:,1);
combine_non_match(:,2)=non_match3(:,2);
combine_non_match(:,3)=non_match3(:,3);

[M,N]=size(rnon_match3);
[M1,N1]=size(combine_non_match);
a=M1+1;
b=0;
row=0;

for x=1:1:M
    for y=1:1:M1
        if combine_non_match(y,1)==rnon_match3(x,1)
            if combine_non_match(y,2)==rnon_match3(x,2)
                row=y;
                b=1;
            end
        end
```

```
    end

    if b==1;

combine_non_match(row,3)=combine_non_match(row,3)+rnon_match3(x,3);
        b=0;
    else
        combine_non_match(a,1)=rnon_match3(x,1);
        combine_non_match(a,2)=rnon_match3(x,2);
        combine_non_match(a,3)=rnon_match3(x,3);
        a=a+1;
        b=0;
    end
end
%%% Sort by first row
combine_match2=sortrows(combine_match,1);
combine_non_match2=sortrows(combine_non_match,1);
%%% Combine match and non-match to determine if groups match
%%% check match against non-match
[M,N]=size(combine_match2);
[M1,N1]=size(combine_non_match2);
a=1;
b=0;
row=0;

for x=1:1:M
    for y=1:1:M1
        if combine_non_match2(y,1)==combine_match2(x,1)
            if combine_non_match2(y,2)==combine_match2(x,2)
                row=y;
                b=1;
            end
        end
    end

    if b==1;
        combine(a,1)=combine_match2(x,1);
        combine(a,2)=combine_match2(x,2);
        combine(a,3)=combine_match2(x,3);
        combine(a,4)=combine_non_match2(row,3);
        a=a+1;
        b=0;
    else
        combine(a,1)=combine_match2(x,1);
        combine(a,2)=combine_match2(x,2);
        combine(a,3)=combine_match2(x,3);
        combine(a,4)=0;
        a=a+1;
        b=0;
    end
end
%%% Check non-match against match (continue filling 'combine')
b=0;
```

```
for y=1:1:M1
    for x=1:1:M
        if combine_non_match2(y,1)==combine_match2(x,1)
            if combine_non_match2(y,2)==combine_match2(x,2)
                b=1;
            end
        end
    end

    if b==1;
        b=0;
    else
        combine(a,1)=combine_non_match2(y,1);
        combine(a,2)=combine_non_match2(y,2);
        combine(a,3)=0;
        combine(a,4)=combine_non_match2(y,3);
        a=a+1;
        b=0;
    end
end
%%%%%%%%%%%%%%%%%%%  create a group array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
class_group(1,1)=combine(1,1);
[M,N]=size(class_group);
[M1,N1]=size(combine);
for x=1:M1
    fill(x,1)=0;
end
x=1;

while x<=N  %class_group column
    y=1;
    a=2;
    while y<=M  %class_group row
        for z=1:M1  %combine row
            if combine(z,1)==class_group(y,x)
                if combine(z,3)>combine(z,4)
                    if fill(z,1)==0
                        class_group(a,x)=combine(z,2);
                        fill(z,1)=1;
                        a=a+1;

                    end
                end
            end
            if combine(z,2)==class_group(y,x)
                if combine(z,3)>combine(z,4)
                    if fill(z,1)==0
                        class_group(a,x)=combine(z,1);
                        fill(z,1)=1;
                        a=a+1;

                    end
                end
```

```
            end
        end
        [M,N]=size(class_group);
        y=y+1;
        if y>M
            for aa=1:M1
                if fill(aa,1)==0;
                    if combine(aa,3)>combine(aa,4)
                        class_group(1,x+1)=combine(aa,1);
                    end
                end
            end
        end
    end
    [M,N]=size(class_group);
    x=x+1;
end
%%% ELIMINATE DOUBLES
[M,N]=size(class_group);

b=0;
row=0;

for v=1:1:N
    a=2;
    class_group2(1,v)=class_group(1,v);
    for x=2:1:M
        [M1,N1]=size(class_group2);
        for y=1:1:M1
            if class_group2(y,v)==class_group(x,v)
                    b=1;
            end
        end

        if b==1;
            b=0;
        else
            class_group2(a,v)=class_group(x,v);
            a=a+1;
            b=0;
        end
    end
end

class_group3=sort(class_group2,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% TEST most populous group to see if they belong together
%%% Find number of groups in each column
[M,N]=size(class_group3);
for x=1:N %class_group3 column
    a=0;
    for y=1:M %class_group3 row
        if class_group3(y,x)>0
            a=a+1;
```

```
        end
    end
    class_group3_count(1,x)=a;
end

%%% find the most populous column
[M,N]=size(class_group3_count);
class_group3_pop(1,1)=class_group3_count(1,1);
class_group3_pop(2,1)=1;
for x=1:N
    if class_group3_count(1,x)>class_group3_pop(1,1)
        class_group3_pop(1,1)=class_group3_count(1,x);
        class_group3_pop(2,1)=x;
    end
end
%%% find number of matches and non-matches
[M,N]=size(class_group3);
[M1,N1]=size(combine);
row=1;
x=class_group3_pop(2,1);

class_group3_buffer(:,row)=class_group3(:,x);
class_group3_buffer(:,row+1)=0;
class_group3_buffer(:,row+2)=0;
for y=1:M %class_group3 row
    for z=1:M %class_group3 row
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for a=1:M1 %combine
            if class_group3(y,x)==combine(a,1)
                if class_group3(z,x)==combine(a,2)

class_group3_buffer(y,row+1)=combine(a,3)+class_group3_buffer(y,row+1);

class_group3_buffer(y,row+2)=combine(a,4)+class_group3_buffer(y,row+2);
                end
            end
            if class_group3(y,x)==combine(a,2)
                if class_group3(z,x)==combine(a,1)

class_group3_buffer(y,row+1)=combine(a,3)+class_group3_buffer(y,row+1);

class_group3_buffer(y,row+2)=combine(a,4)+class_group3_buffer(y,row+2);
                end
            end
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end
end
%%% remove rejected groups; fill 'reject' array
reject(1,1)=0;
[M,N]=size(class_group3_buffer);
a=1;
b=1;
for x=1:M
    if class_group3_buffer(x,2)>class_group3_buffer(x,3);
```

```
        class_group3_buffer2(a,1)=class_group3_buffer(x,1);
        a=a+1;
    else
        reject(b,1)=class_group3_buffer(x,1);
        b=b+1;
    end
end
%%% Scrub rejected groups from 'combine'
combine_scrub(:,:)=combine(:,:);
[M,N]=size(reject);
[M1,N1]=size(combine_scrub);

for x=1:M
    for y=1:M1
        if combine_scrub(y,1)==reject(x,1)
            combine_scrub(y,3)=0;
            combine_scrub(y,4)=0;
        end
        if combine_scrub(y,2)==reject(x,1)
            combine_scrub(y,3)=0;
            combine_scrub(y,4)=0;
        end
    end
end
%%% RECALCULATE 'class_group' column one (class_group_new)
class_group_new(1,1)=class_group3_buffer2(1,1);

[M,N]=size(class_group_new);
[M1,N1]=size(combine_scrub);

for x=1:M1
    fill(x,1)=0;
end

x=1;
y=1;
a=2;
while y<=M  %class_group10 row
    for z=1:M1  %combine_scrub row
        if combine_scrub(z,1)==class_group_new(y,x)
            if combine_scrub(z,3)>combine_scrub(z,4)
                if fill(z,1)==0
                    class_group_new(a,x)=combine_scrub(z,2);
                    fill(z,1)=1;
                    a=a+1;

                end
            end
        end
        if combine_scrub(z,2)==class_group_new(y,x)
            if combine_scrub(z,3)>combine_scrub(z,4)
                if fill(z,1)==0
                    class_group_new(a,x)=combine_scrub(z,1);
                    fill(z,1)=1;
```

```
                    a=a+1;

                end
            end
        end
    end
    [M,N]=size(class_group_new);
    y=y+1;
end
%%% set 'fill' to 1 if rejected group matches with column 1
[R,C]=size(reject);
for x=1:R
    for y=1:M1
        if combine(y,1)==reject(R,1)
            for z=1:M
                if combine(y,2)==class_group_new(M,1)
                    fill(y,1)=1;
                end
            end
        end
        if combine(y,2)==reject(R,1)
            for z=1:M
                if combine(y,1)==class_group_new(M,1)
                    fill(y,1)=1;
                end
            end
        end
    end
end
%%% second column (go back to combine)
[M,N]=size(class_group_new);
b=0;
for aa=1:M1
    if fill(aa,1)==0;
        if combine(aa,3)>combine(aa,4)
            for bb=1:M
                if combine(aa,1)==class_group_new(bb,1)
                    b=1;
                end
            end
            if b==1
                b=0;
            else
            class_group_new(1,2)=combine(aa,1);
            b=0;
            end
        end
    end
end
[M,N]=size(class_group_new);
x=2;
c=0;
while x<=N  %class_group column
    y=1;
    a=2;
```

```
while y<=M  %class_group10 row
    for z=1:M1  %combine row

        if combine(z,1)==class_group_new(y,x)
            if combine(z,3)>combine(z,4)
                if fill(z,1)==0
                    for cc=1:N
                        for bb=1:M
                            if combine(z,2)==class_group_new(bb,cc)
                                b=1;
                            end
                        end
                    end
                    if b==1
                        b=0;
                    else
                        class_group_new(a,x)=combine(z,2);
                        b=0;
                        c=1;
                        fill(z,1)=1;
                        a=a+1;
                    end
                end
            end
        end

        if combine(z,2)==class_group_new(y,x)
            if combine(z,3)>combine(z,4)
                if fill(z,1)==0
                    for cc=1:N
                        for bb=1:M
                            if combine(z,1)==class_group_new(bb,cc)
                                b=1;
                            end
                        end
                    end
                    if b==1
                        b=0;
                    else
                        class_group_new(a,x)=combine(z,1);
                        b=0;
                        c=1;
                        fill(z,1)=1;
                        a=a+1;
                    end
                end
            end
        end

    end

    %%% set up for next column
    [M,N]=size(class_group_new);
    y=y+1;
```

```
        if y>M
            if c==1;
                for aa=1:M1
                    if fill(aa,1)==0;
                        if combine(aa,3)>combine(aa,4)
                            for cc=1:N
                                for bb=1:M
                                    if
combine(aa,1)==class_group_new(bb,cc)
                                        b=1;
                                    end
                                end
                            end
                            if b==1
                                b=0;
                            else
                                class_group_new(1,x+1)=combine(aa,1);
                                b=0;
                                c=0;
                            end
                        end
                    end
                    c=0;
                end
            end
        end
        [M,N]=size(class_group_new);
        x=x+1;
end
%%% ELIMINATE DOUBLES
[M,N]=size(class_group_new);
b=0;
for v=1:1:N
    a=2;
    class_group_new2(1,v)=class_group_new(1,v);
    for x=2:1:M
        [M1,N1]=size(class_group_new2);
        for y=1:1:M1
            if class_group_new2(y,v)==class_group_new(x,v)
                    b=1;
            end
        end

        if b==1;
            b=0;
        else
            class_group_new2(a,v)=class_group_new(x,v);
            a=a+1;
            b=0;
        end
    end
end
class_group_new3=sort(class_group_new2,1);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%%%%    label RAW with appropriate group numbers to graph
%%%%%%%%
 [M,N]=size(class_group_new3);
[M1,N1]=size(raw);
for g=1:1:N
    for x=1:1:M
        for y=1:1:M1

            if raw(y,4)==class_group_new3(x,g)
                raw(y,8)=g;
            end

        end
    end
end
%%% find single groups (labeled as group zero)
clear buffer
a=1;
ind=0;
for x=1:1:M1
    if raw(x,8)==0
        buffer(a,1)=raw(x,4);
        a=a+1;
        ind=1;
    end
end

if ind==1
    [M,N]=size(buffer);
    a=2;
    b=0;
    row=0;
    buffer2(1,1)=buffer(1,1);
    buffer2(1,2)=1;

    for x=2:1:M
        [M1,N1]=size(buffer2);
        for y=1:1:M1
            if buffer2(y,1)==buffer(x,1)
                row=y;
                b=1;
            end
        end

        if b==1;
            buffer2(row,2)=buffer2(row,2)+1;
            b=0;
        else
            buffer2(a,1)=buffer(x,1);
            buffer2(a,2)=1;
            a=a+1;
            b=0;
```

```
        end
    end
    %%% drop single groups with 3 or less points
    [M,N]=size(buffer2);
    a=1;
    buffer3(1,1)=0;
    for x=1:1:M
        if buffer2(x,2)>3
            buffer3(1,a)=buffer2(x,1);
            a=a+1;
        end
    end
    %%% label points in RAW with single groups (over 3 points)
    [M,N]=size(buffer3);
    [M1,N1]=size(raw);
    [M2,N2]=size(class_group_new3);

    for g=1:1:N
        for y=1:1:M1
            if raw(y,4)==buffer3(1,g)
                raw(y,8)=g+N2;
            end
        end
    end
end
%%% find amount of points in each group
[M,N]=size(raw);
a=2;
b=0;
row=0;
cumul(1,1)=raw(1,8);
cumul(1,2)=1;
for x=2:1:M
    [M1,N1]=size(cumul);
    for y=1:1:M1
        if cumul(y,1)==raw(x,8)
                row=y;
                b=1;
        end
    end

    if b==1;
        cumul(row,2)=cumul(row,2)+1;
        b=0;
    else
        cumul(a,1)=raw(x,8);
        cumul(a,2)=1;
        a=a+1;
        b=0;
    end
end
%%% eliminate groups with 3 or less points
[M,N]=size(cumul);
drop_group(1,1)=0;
a=1;
```

```
for x=1:1:M
    if cumul(x,2)<4
        drop_group(a,1)=cumul(x,1);
        a=a+1;
    end
end

[M,N]=size(drop_group);
[M1,N1]=size(raw);
for x=1:1:M
    for y=1:1:M1
        if raw(y,8)==drop_group(x,1)
            raw(y,8)=0;
        end
    end
end

%%% Sort (cumul1)
[M,N]=size(cumul);
a=1;
for x=1:1:M
    if cumul(x,1)==0
    else
        if cumul(x,2)>3
            cumul1(a,1)=cumul(x,1);
            cumul1(a,2)=cumul(x,2);
            a=a+1;
        end
    end
end
cumul2=sortrows(cumul1,-2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%%             IDENTIFY GROUPS AS GRAIN OR EDGE
%%%%
%%% find x coordinates of each group
[M,N]=size(raw);
[M1,N1]=size(cumul2);
col=1;
a=1;
for x=1:1:M1
    for y=1:1:M
        if raw(y,8)==cumul2(x,1)
            group_x(a,col)=raw(y,1);
            a=a+1;
        end
    end
    col=col+1;
    a=1;
end
%%% find max and min of x of each group
group_x2=sort(group_x,1);
[M,N]=size(group_x2);
rev=M;
ind_min=0;
```

```
ind_max=0;
for x=1:1:N %%% group_x2 column
    for y=1:1:M %%% group_x2 row
        if ind_min==0;
            if group_x2(y,x)==0;
            else
                group_x_limits(1,x)=group_x2(y,x);
                ind_min=1;
            end
        end
        if ind_max==0;
            if group_x2(rev,x)==0;
            else
                group_x_limits(2,x)=group_x2(rev,x);
                ind_max=1;
            end
        end
        rev=rev-1;
    end
    ind_min=0;
    ind_max=0;
    rev=M;
end
%%% find range in x direction
[M,N]=size(group_x_limits);
for x=1:1:N %%% group_x_limits2 column
    group_x_range(1,x)=group_x_limits(2,x)-group_x_limits(1,x);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% find y coordinates of each group
[M,N]=size(raw);
[M1,N1]=size(cumul2);
col=1;
a=1;
for x=1:1:M1
    for y=1:1:M
        if raw(y,8)==cumul2(x,1)
            group_y(a,col)=raw(y,2);
            a=a+1;
        end
    end
    col=col+1;
    a=1;
end
%%% find max and min of y of each group
group_y2=sort(group_y,1);
[M,N]=size(group_y2);
rev=M;
ind_min=0;
ind_max=0;
for x=1:1:N %%% group_y2 column
    for y=1:1:M %%% group_y2 row
        if ind_min==0;
            if group_y2(y,x)==0;
            else
```

```
                group_y_limits(1,x)=group_y2(y,x);
                ind_min=1;
            end
        end
        if ind_max==0;
            if group_y2(rev,x)==0;
            else
                group_y_limits(2,x)=group_y2(rev,x);
                ind_max=1;
            end
        end
        rev=rev-1;
    end
    ind_min=0;
    ind_max=0;
    rev=M;
end
%%% find range in y direction
[M,N]=size(group_y_limits);
for x=1:1:N %%% group_y_limits2 column
    group_y_range(1,x)=group_y_limits(2,x)-group_y_limits(1,x);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% find z coordinates of each group
[M,N]=size(raw);
[M1,N1]=size(cumul2);
col=1;
a=1;
for x=1:1:M1
    for y=1:1:M
        if raw(y,8)==cumul2(x,1)
            group_z(a,col)=raw(y,3);
            a=a+1;
        end
    end
    col=col+1;
    a=1;
end
%%% find max and min of z of each group
group_z2=sort(group_z,1);
[M,N]=size(group_z2);
rev=M;
ind_min=0;
ind_max=0;
for x=1:1:N %%% group_z2 column
    for y=1:1:M %%% group_z2 row
        if ind_min==0;
            if group_z2(y,x)==0;
            else
                group_z_limits(1,x)=group_z2(y,x);
                ind_min=1;
            end
        end
        if ind_max==0;
```

```
            if group_z2(rev,x)==0;
            else
                group_z_limits(2,x)=group_z2(rev,x);
                ind_max=1;
            end
        end
        rev=rev-1;
    end
    ind_min=0;
    ind_max=0;
    rev=M;
end
%%% find range in z direction
[M,N]=size(group_z_limits);
for x=1:1:N %%% group_z_limits2 column
    group_z_range(1,x)=group_z_limits(2,x)-group_z_limits(1,x);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%% find ratio between x and y
[M,N]=size(group_x_range);
for x=1:1:N
    group_xy_ratio(1,x)=group_x_range(1,x)/group_y_range(1,x);
end

%%% determine if edge and direction
[M,N]=size(group_xy_ratio);
a=1;
b=1;
c=1;
yes_horiz_edge=0;
yes_vert_edge=0;
yes_grain=0;
for x=1:1:N
    if group_xy_ratio(1,x)>3.0;  %1.7
        group_horiz_edge(a,1)=x;
        group_horiz_edge(a,2)=group_xy_ratio(1,x);
        a=a+1;
        yes_horiz_edge=1;
    else
        if group_xy_ratio(1,x)<0.27  %.5
            group_vertical_edge(b,1)=x;
            group_vertical_edge(b,2)=group_xy_ratio(1,x);
            b=b+1;
            yes_vert_edge=1;
        else
            [M1,N1]=size(raw);
            var=1;
            num_group=0;
            for y=1:M1
                if raw(y,8)==cumul2(x,1)
                    dif=abs(raw(y,4)-var);
                    if dif>85
                        var=raw(y,4);
                        num_group=num_group+1;
```

```
                    end
                end
            end
            if num_group>1
                if group_xy_ratio(1,x)>1.538;  %1.7
                    group_horiz_edge(a,1)=x;
                    group_horiz_edge(a,2)=group_xy_ratio(1,x);
                    a=a+1;
                    yes_horiz_edge=1;
                else
                    if group_xy_ratio(1,x)<0.45  %.5
                        group_vertical_edge(b,1)=x;
                        group_vertical_edge(b,2)=group_xy_ratio(1,x);
                        b=b+1;
                        yes_vert_edge=1;
                    else
                        group_grain(c,1)=x;
                        group_grain(c,2)=group_xy_ratio(1,x);
                        c=c+1;
                        yes_grain=1;
                    end
                end
            else
                group_grain(c,1)=x;
                group_grain(c,2)=group_xy_ratio(1,x);
                c=c+1;
                yes_grain=1;
            end
        end
    end
end
%%% Find average z
[M,N]=size(group_z2);
group_z_sum(1,:)=group_z2(1,:);

for x=1:1:N
    a=1;
    for y=2:1:M
        if group_z2(y,x)==0
        else
            a=a+1;
            group_z_sum(1,x)=group_z2(y,x)+group_z_sum(1,x);
        end
    end
    group_z_avg(1,x)=group_z_sum(1,x)/a;
end
%%% Find highest edge
if yes_horiz_edge==1;
    [M,N]=size(group_horiz_edge);
    for x=1:1:M
        col=group_horiz_edge(x,1);
        group_horiz_edge(x,3)=group_z_avg(1,col);
    end
    group_horiz_edge2=sortrows(group_horiz_edge,-3);
end
```

```
if yes_vert_edge==1;
    [M,N]=size(group_vertical_edge);
    for x=1:1:M
        col=group_vertical_edge(x,1);
        group_vertical_edge(x,3)=group_z_avg(1,col);
    end
    group_vertical_edge2=sortrows(group_vertical_edge,-3);
end
%%% CHECK for false edge: find delta between vert and horz edges
if yes_horiz_edge==1;
    if yes_vert_edge==1;
        group_delta=group_horiz_edge2(1,3)-group_vertical_edge2(1,3);
        if group_delta>100
            yes_vert_edge=0;
        end
        if group_delta<-100
            yes_horiz_edge=0;
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if yes_horiz_edge==1;
    horiz_edge_group=group_horiz_edge2(1,1);

    [M,N]=size(raw);
    a=1;
    for x=1:1:M
        if raw(x,8)==cumul2(horiz_edge_group,1)
            raw_horiz_edge(a,1)=raw(x,1);
            raw_horiz_edge(a,2)=raw(x,2);
            raw_horiz_edge(a,3)=raw(x,3);
            a=a+1;
        end
    end
    %%% p = polyfit(x,y,n)
    %%% fit y by x

horiz_line_fit_coeff1=polyfit(raw_horiz_edge(:,1),raw_horiz_edge(:,2),1
);
    %%% fit z by x

horiz_line_fit_coeff2=polyfit(raw_horiz_edge(:,1),raw_horiz_edge(:,3),1
);

    a=1;
    for x=-1500:50:1500
       horiz_edge(a,1)=x;

horiz_edge(a,2)=horiz_line_fit_coeff1(1,1)*x+horiz_line_fit_coeff1(1,2)
;

horiz_edge(a,3)=horiz_line_fit_coeff2(1,1)*x+horiz_line_fit_coeff2(1,2)
;
       a=a+1;
```

```
    end
end
%%% repeat for vertical edge
if yes_vert_edge==1;
    vert_edge_group=group_vertical_edge2(1,1);

    [M,N]=size(raw);
    a=1;
    for x=1:1:M
        if raw(x,8)==cumul2(vert_edge_group,1)
            raw_vert_edge(a,1)=raw(x,1);
            raw_vert_edge(a,2)=raw(x,2);
            raw_vert_edge(a,3)=raw(x,3);
            a=a+1;
        end
    end
    %%% p = polyfit(x,y,n)
    %%% fit x by y

vert_line_fit_coeff1=polyfit(raw_vert_edge(:,2),raw_vert_edge(:,1),1);
    %%% fit z by y

vert_line_fit_coeff2=polyfit(raw_vert_edge(:,2),raw_vert_edge(:,3),1);
    a=1;
    for x=-1500:50:1500

vert_edge(a,1)=vert_line_fit_coeff1(1,1)*x+vert_line_fit_coeff1(1,2);
       vert_edge(a,2)=x;

vert_edge(a,3)=vert_line_fit_coeff2(1,1)*x+vert_line_fit_coeff2(1,2);
       a=a+1;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%%    PLOT!!!
figure,scatter3(raw(:,1),raw(:,2),raw(:,3));
xlabel('X');
ylabel('Y');
zlabel('Z');
axis([-1500 1500 -1500 1500 -3000 0])
grid on
hold on



[M,N]=size(raw);
[M1,N1]=size(cumul2);

if M1>=1
    a=1;
    for x=1:1:M
        if raw(x,8)==cumul2(1,1)
            raw_group_one(a,1)=raw(x,1);
```

```
            raw_group_one(a,2)=raw(x,2);
            raw_group_one(a,3)=raw(x,3);
            a=a+1;
        end
    end
    scatter3(raw_group_one(:,1),raw_group_one(:,2),raw_group_one(:,3),
'filled');
end
if M1>=2
    a=1;
    for x=1:1:M
        if raw(x,8)==cumul2(2,1)
            raw_group_two(a,1)=raw(x,1);
            raw_group_two(a,2)=raw(x,2);
            raw_group_two(a,3)=raw(x,3);
            a=a+1;
        end
    end
    scatter3(raw_group_two(:,1),raw_group_two(:,2),raw_group_two(:,3),
'filled');
end
if M1>=3
    a=1;
    for x=1:1:M
        if raw(x,8)==cumul2(3,1)
            raw_group_three(a,1)=raw(x,1);
            raw_group_three(a,2)=raw(x,2);
            raw_group_three(a,3)=raw(x,3);
            a=a+1;
        end
    end

scatter3(raw_group_three(:,1),raw_group_three(:,2),raw_group_three(:,3)
, 'filled');
end
if M1>=4
    a=1;
    for x=1:1:M
        if raw(x,8)==cumul2(4,1)
            raw_group_four(a,1)=raw(x,1);
            raw_group_four(a,2)=raw(x,2);
            raw_group_four(a,3)=raw(x,3);
            a=a+1;
        end
    end

scatter3(raw_group_four(:,1),raw_group_four(:,2),raw_group_four(:,3),
'filled');
end
if M1>=5
    a=1;
    for x=1:1:M
        if raw(x,8)==cumul2(5,1)
            raw_group_five(a,1)=raw(x,1);
            raw_group_five(a,2)=raw(x,2);
```

```
            raw_group_five(a,3)=raw(x,3);
            a=a+1;
        end
    end

scatter3(raw_group_five(:,1),raw_group_five(:,2),raw_group_five(:,3),
'filled');
end
if M1>=6
    a=1;
    for x=1:1:M
        if raw(x,8)==cumul2(6,1)
            raw_group_six(a,1)=raw(x,1);
            raw_group_six(a,2)=raw(x,2);
            raw_group_six(a,3)=raw(x,3);
            a=a+1;
        end
    end
    scatter3(raw_group_six(:,1),raw_group_six(:,2),raw_group_six(:,3),
'filled');
end
if M1>=7
    a=1;
    for x=1:1:M
        if raw(x,8)==cumul2(7,1)
            raw_group_seven(a,1)=raw(x,1);
            raw_group_seven(a,2)=raw(x,2);
            raw_group_seven(a,3)=raw(x,3);
            a=a+1;
        end
    end

scatter3(raw_group_seven(:,1),raw_group_seven(:,2),raw_group_seven(:,3)
, 'filled');
end
if M1>=8
    a=1;
    for x=1:1:M
        if raw(x,8)==cumul2(8,1)
            raw_group_eight(a,1)=raw(x,1);
            raw_group_eight(a,2)=raw(x,2);
            raw_group_eight(a,3)=raw(x,3);
            a=a+1;
        end
    end

scatter3(raw_group_eight(:,1),raw_group_eight(:,2),raw_group_eight(:,3)
, 'filled');
end



if yes_horiz_edge==1;
    scatter3(horiz_edge(:,1),horiz_edge(:,2),horiz_edge(:,3),5,
'filled','MarkerFaceColor',[0 0 0]);
end
```

```
if yes_vert_edge==1;
    scatter3(vert_edge(:,1),vert_edge(:,2),vert_edge(:,3),5,
'filled','MarkerFaceColor',[0 0 0]);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if num==2
    Fx=808;     %FRONT edge (1 x)
    Rx=-4272;   %REAR edge (3 x)
    Ny=726;     %NEAR edge (3 y)
    Fy=3444;    %FAR edge (1 y)
    Z=-704;
end
if num==3
    Fx=2357;    %FRONT edge (1 x)
    Rx=-2723;    %REAR edge (3 x)
    Ny=737;    %NEAR edge (3 y)
    Fy=3454;    %FAR edge (1 y)
    Z=-726;
end
if num==4
    Fx=4161;    %FRONT edge (1 x)
    Rx=-922;    %REAR edge (3 x)
    Ny=744;    %NEAR edge (3 y)
    Fy=3462;    %FAR edge (1 y)
    Z=-739;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if num==5
    Fx=-627;     %FRONT edge (1 x)
    Rx=-5707;   %REAR edge (3 x)
    Ny=-221;     %NEAR edge (3 y)
    Fy=2497;    %FAR edge (1 y)
    Z=-665;
end
if num==6
    Fx=569;    %FRONT edge (1 x)
    Rx=-4511;    %REAR edge (3 x)
    Ny=-237;    %NEAR edge (3 y)
    Fy=2479;    %FAR edge (1 y)
    Z=-709;
end
if num==7
    Fx=2606;    %FRONT edge (1 x)
    Rx=-2474;    %REAR edge (3 x)
    Ny=-267;    %NEAR edge (3 y)
    Fy=2451;    %FAR edge (1 y)
    Z=-714;
end
if num==8
    Fx=3914;    %FRONT edge (1 x)
    Rx=-1166;    %REAR edge (3 x)
    Ny=-282;    %NEAR edge (3 y)
    Fy=2436;    %FAR edge (1 y)
    Z=-716;
```

```
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if num==9
    Fx=-683;     %FRONT edge (1 x)
    Rx=-5763;   %REAR edge (3 x)
    Ny=-693;     %NEAR edge (3 y)
    Fy=2024;    %FAR edge (1 y)
    Z=-734;
end
if num==10
    Fx=1057;    %FRONT edge (1 x)
    Rx=-4023;    %REAR edge (3 x)
    Ny=-732;    %NEAR edge (3 y)
    Fy=1986;    %FAR edge (1 y)
    Z=-721;
end
if num==11
    Fx=2799;    %FRONT edge (1 x)
    Rx=-2281;    %REAR edge (3 x)
    Ny=-785;    %NEAR edge (3 y)
    Fy=1933;    %FAR edge (1 y)
    Z=-701;
end
if num==12
    Fx=3962;    %FRONT edge (1 x)
    Rx=-1092;    %REAR edge (3 x)
    Ny=-775;    %NEAR edge (3 y)
    Fy=1943;    %FAR edge (1 y)
    Z=-665;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if num==13
    Fx=945;    %FRONT edge (1 x)
    Rx=-4135;    %REAR edge (3 x)
    Ny=-361;    %NEAR edge (3 y)
    Fy=2357;    %FAR edge (1 y)
    Z=-635;
end
if num==14
    Fx=2746;    %FRONT edge (1 x)
    Rx=-2334;    %REAR edge (3 x)
    Ny=-409;    %NEAR edge (3 y)
    Fy=2309;    %FAR edge (1 y)
    Z=-673;
end
if num==15
    Fx=4153;    %FRONT edge (1 x)
    Rx=-927;    %REAR edge (3 x)
    Ny=-412;    %NEAR edge (3 y)
    Fy=2306;    %FAR edge (1 y)
    Z=-630;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
actual_grain(1,1)=Fx;
actual_grain(1,2)=Ny;
```

```
actual_grain(1,3)=Z-431;
actual_grain(2,1)=Fx;
actual_grain(2,2)=Ny+679.45;
actual_grain(2,3)=Z-330;
actual_grain(3,1)=Fx;
actual_grain(3,2)=Ny+1358.9;
actual_grain(3,3)=Z-330;
actual_grain(4,1)=Fx;
actual_grain(4,2)=Ny+2038.35;
actual_grain(4,3)=Z-305;
actual_grain(5,1)=Fx;
actual_grain(5,2)=Ny+2717.8;
actual_grain(5,3)=Z-406;

actual_grain(6,1)=Fx-846.66;
actual_grain(6,2)=Ny;
actual_grain(6,3)=Z-356;
actual_grain(7,1)=Fx-846.66;
actual_grain(7,2)=Ny+679.45;
actual_grain(7,3)=Z-305;
actual_grain(8,1)=Fx-846.66;
actual_grain(8,2)=Ny+1358.9;
actual_grain(8,3)=Z-152;
actual_grain(9,1)=Fx-846.66;
actual_grain(9,2)=Ny+2038.35;
actual_grain(9,3)=Z-229;
actual_grain(10,1)=Fx-846.66;
actual_grain(10,2)=Ny+2717.8;
actual_grain(10,3)=Z-356;

actual_grain(11,1)=Fx-1693.3;
actual_grain(11,2)=Ny;
actual_grain(11,3)=Z-356;
actual_grain(12,1)=Fx-1693.3;
actual_grain(12,2)=Ny+679.45;
actual_grain(12,3)=Z-203;
actual_grain(13,1)=Fx-1693.3;
actual_grain(13,2)=Ny+1358.9;
actual_grain(13,3)=Z;
actual_grain(14,1)=Fx-1693.3;
actual_grain(14,2)=Ny+2038.35;
actual_grain(14,3)=Z-178;
actual_grain(15,1)=Fx-1693.3;
actual_grain(15,2)=Ny+2717.8;
actual_grain(15,3)=Z-279;

actual_grain(16,1)=Fx-2540;
actual_grain(16,2)=Ny;
actual_grain(16,3)=Z-330;
actual_grain(17,1)=Fx-2540;
actual_grain(17,2)=Ny+679.45;
actual_grain(17,3)=Z-254;
actual_grain(18,1)=Fx-2540;
actual_grain(18,2)=Ny+1358.9;
actual_grain(18,3)=Z-152;
```

```
actual_grain(19,1)=Fx-2540;
actual_grain(19,2)=Ny+2038.35;
actual_grain(19,3)=Z-254;
actual_grain(20,1)=Fx-2540;
actual_grain(20,2)=Ny+2717.8;
actual_grain(20,3)=Z-305;

actual_grain(21,1)=Fx-3386.6;
actual_grain(21,2)=Ny;
actual_grain(21,3)=Z-483;
actual_grain(22,1)=Fx-3386.6;
actual_grain(22,2)=Ny+679.45;
actual_grain(22,3)=Z-356;
actual_grain(23,1)=Fx-3386.6;
actual_grain(23,2)=Ny+1358.9;
actual_grain(23,3)=Z-254;
actual_grain(24,1)=Fx-3386.6;
actual_grain(24,2)=Ny+2038.35;
actual_grain(24,3)=Z-406;
actual_grain(25,1)=Fx-3386.6;
actual_grain(25,2)=Ny+2717.8;
actual_grain(25,3)=Z-483;

actual_grain(26,1)=Fx-4233.3;
actual_grain(26,2)=Ny;
actual_grain(26,3)=Z-610;
actual_grain(27,1)=Fx-4233.3;
actual_grain(27,2)=Ny+679.45;
actual_grain(27,3)=Z-483;
actual_grain(28,1)=Fx-4233.3;
actual_grain(28,2)=Ny+1358.9;
actual_grain(28,3)=Z-356;
actual_grain(29,1)=Fx-4233.3;
actual_grain(29,2)=Ny+2038.35;
actual_grain(29,3)=Z-483;
actual_grain(30,1)=Fx-4233.3;
actual_grain(30,2)=Ny+2717.8;
actual_grain(30,3)=Z-584;

actual_grain(31,1)=Fx-5080;
actual_grain(31,2)=Ny;
actual_grain(31,3)=Z-610;
actual_grain(32,1)=Fx-5080;
actual_grain(32,2)=Ny+679.45;
actual_grain(32,3)=Z-635;
actual_grain(33,1)=Fx-5080;
actual_grain(33,2)=Ny+1358.9;
actual_grain(33,3)=Z-660;
actual_grain(34,1)=Fx-5080;
actual_grain(34,2)=Ny+2038.35;
actual_grain(34,3)=Z-559;
actual_grain(35,1)=Fx-5080;
actual_grain(35,2)=Ny+2717.8;
actual_grain(35,3)=Z-610;
%%% PLOT for PAPER
```

```
figure,scatter3(raw(:,1),raw(:,2),raw(:,3));
xlabel('X');
ylabel('Y');
zlabel('Z');
axis([-1500 1500 -1500 1500 -3000 0])
grid on
hold on
if yes_grain==1
    [M,N]=size(group_grain);
    [M1,N1]=size(raw);
    b=1;
    c=1;
    for x=1:1:M
        a=cumul2(group_grain(x,1),1);
        for y=1:1:M1
            if raw(y,8)==a
                raw_grain(b,1)=raw(y,1);
                raw_grain(b,2)=raw(y,2);
                raw_grain(b,3)=raw(y,3);
                b=b+1;
            end
        end
    end
    scatter3(raw_grain(:,1),raw_grain(:,2),raw_grain(:,3),
'filled','MarkerFaceColor',[1 1 0]);
end

if yes_horiz_edge==1;

scatter3(raw_horiz_edge(:,1),raw_horiz_edge(:,2),raw_horiz_edge(:,3),
'filled','MarkerFaceColor',[1 0 0]);
    scatter3(horiz_edge(:,1),horiz_edge(:,2),horiz_edge(:,3),5,
'filled','MarkerFaceColor',[0 0 0]);
end

if yes_vert_edge==1;
    scatter3(raw_vert_edge(:,1),raw_vert_edge(:,2),raw_vert_edge(:,3),
'filled','MarkerFaceColor',[1 0 1]);
    scatter3(vert_edge(:,1),vert_edge(:,2),vert_edge(:,3),5,
'filled','MarkerFaceColor',[0 0 0]);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PLOT actual wagon edges
a=1;
for x=Rx:60:Fx
    near_wag(a,1)=x;
    near_wag(a,2)=Ny;
    near_wag(a,3)=Z;
    far_wag(a,1)=x;
    far_wag(a,2)=Fy;
    far_wag(a,3)=Z;
    a=a+1;
end

a=1;
```

```
for y=Ny:40:Fy
    front_wag(a,1)=Fx;
    front_wag(a,2)=y;
    front_wag(a,3)=Z;

    rear_wag(a,1)=Rx;
    rear_wag(a,2)=y;
    rear_wag(a,3)=Z;
    a=a+1;
end

scatter3(near_wag(:,1),near_wag(:,2),near_wag(:,3),5,
'filled','MarkerFaceColor',[0.5 0.5 0.5]);
scatter3(far_wag(:,1),far_wag(:,2),far_wag(:,3),5,
'filled','MarkerFaceColor',[0.5 0.5 0.5]);
scatter3(front_wag(:,1),front_wag(:,2),front_wag(:,3),5,
'filled','MarkerFaceColor',[0.5 0.5 0.5]);
scatter3(rear_wag(:,1),rear_wag(:,2),rear_wag(:,3),5,
'filled','MarkerFaceColor',[0.5 0.5 0.5]);
if num<13
    scatter3(actual_grain(:,1),actual_grain(:,2),actual_grain(:,3),50,
'filled','MarkerFaceColor',[0 0 1]);
end
%%%%%%%%%%%%%   pull out grain for SURFACE PLOT
%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   z=f(x,y)
if yes_grain==1
    x=raw_grain(:,1);
    y=raw_grain(:,2);
    [M,N]=size(raw_grain);
    for a=1:1:M
        x2(a,1)=x(a,1)*x(a,1);
        y2(a,1)=y(a,1)*y(a,1);
        xy(a,1)=x(a,1)*y(a,1);
    end


    X=[ones(size(x2)) x2 y2 xy x y ];
    poly_coef=X\raw_grain(:,3);
    a=1;
    for x=-1500:100:1500
        b=-1500;
        for y=1:1:31
            grain_surf(a,1)=x;
            grain_surf(a,2)=b;

grain_surf(a,3)=poly_coef(1,1)+poly_coef(2,1)*x*x+poly_coef(3,1)*b*b+po
ly_coef(4,1)*x*b+poly_coef(5,1)*x+poly_coef(6,1)*b;
            a=a+1;
            b=b+100;
        end
    end
    [M,N]=size(grain_surf);
    a=1;
```

```
    row=1;
    for x=1:1:M
        if a==32
            row=row+1;
            a=1;
        end
        grain_surf_z(a,row)=grain_surf(x,3);
        a=a+1;
    end

    a=1;
    for x=-1500:100:1500
        grain_surf_x(a,1)=x;
        grain_surf_y(a,1)=x;
        a=a+1;
    end


    figure,scatter3(raw(:,1),raw(:,2),raw(:,3));
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    axis([-1500 1500 -1500 1500 -3000 0])
    grid on
    hold on
    surf(grain_surf_x,grain_surf_y,grain_surf_z);

    scatter3(raw_grain(:,1),raw_grain(:,2),raw_grain(:,3),
'filled','MarkerFaceColor',[1 1 0]);

    if yes_horiz_edge==1;

scatter3(raw_horiz_edge(:,1),raw_horiz_edge(:,2),raw_horiz_edge(:,3),
'filled','MarkerFaceColor',[1 0 0]);
        scatter3(horiz_edge(:,1),horiz_edge(:,2),horiz_edge(:,3),5,
'filled','MarkerFaceColor',[0 0 0]);
    end

    if yes_vert_edge==1;

scatter3(raw_vert_edge(:,1),raw_vert_edge(:,2),raw_vert_edge(:,3),
'filled','MarkerFaceColor',[1 0 1]);
        scatter3(vert_edge(:,1),vert_edge(:,2),vert_edge(:,3),5,
'filled','MarkerFaceColor',[0 0 0]);
    end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%  compare predicted grain to actual grain

upper_x=0;
lower_x=0;
a=0;
```

```
c(1)=0;
c(2)=0;
c(3)=0;
c(4)=0;
c(5)=0;
c(6)=0;
c(7)=0;

while a<3
    a=0;
    upper_x=upper_x+10;
    lower_x=lower_x-10;
    if actual_grain(1,1)>lower_x
        if actual_grain(1,1)<upper_x
            a=a+1;
            c(1)=1;
        end
    end
    if actual_grain(6,1)>lower_x
        if actual_grain(6,1)<upper_x
            a=a+1;
            c(2)=1;
        end
    end
    if actual_grain(11,1)>lower_x
        if actual_grain(11,1)<upper_x
            a=a+1;
            c(3)=1;
        end
    end
    if actual_grain(16,1)>lower_x
        if actual_grain(16,1)<upper_x
            a=a+1;
            c(4)=1;
        end
    end
    if actual_grain(21,1)>lower_x
        if actual_grain(21,1)<upper_x
            a=a+1;
            c(5)=1;
        end
    end
    if actual_grain(26,1)>lower_x
        if actual_grain(26,1)<upper_x
            a=a+1;
            c(6)=1;
        end
    end
    if actual_grain(31,1)>lower_x
        if actual_grain(31,1)<upper_x
            a=a+1;
            c(7)=1;
        end
    end
end
```

```
upper_y=0;
lower_y=0;
a=0;
r(1)=0;
r(2)=0;
r(3)=0;
r(4)=0;
r(5)=0;

while a<3
    a=0;
    upper_y=upper_y+10;
    lower_y=lower_y-10;
    if actual_grain(1,1)>lower_y
        if actual_grain(1,2)<upper_y
            a=a+1;
            r(1)=1;
        end
    end
    if actual_grain(6,1)>lower_y
        if actual_grain(2,2)<upper_y
            a=a+1;
            r(2)=1;
        end
    end
    if actual_grain(11,1)>lower_y
        if actual_grain(3,2)<upper_y
            a=a+1;
            r(3)=1;
        end
    end
    if actual_grain(16,1)>lower_y
        if actual_grain(4,2)<upper_y
            a=a+1;
            r(4)=1;
        end
    end
    if actual_grain(21,1)>lower_y
        if actual_grain(5,2)<upper_y
            a=a+1;
            r(5)=1;
        end
    end
end

for x=1:7
    if c(x)==1
        colo=(x-1)*5+1;
        actual_grain(colo,4)=1;
        actual_grain(colo+1,4)=1;
        actual_grain(colo+2,4)=1;
        actual_grain(colo+3,4)=1;
        actual_grain(colo+4,4)=1;
    end
```

```
end

for x=1:5
    if r(x)==1
        actual_grain(x,5)=1;
        actual_grain(x+5,5)=1;
        actual_grain(x+10,5)=1;
        actual_grain(x+15,5)=1;
        actual_grain(x+20,5)=1;
        actual_grain(x+25,5)=1;
        actual_grain(x+30,5)=1;
    end
end

a=1;
for y=1:35
    x=actual_grain(y,1);
    b=actual_grain(y,2);

actual_grain(y,6)=poly_coef(1,1)+poly_coef(2,1)*x*x+poly_coef(3,1)*b*b+
poly_coef(4,1)*x*b+poly_coef(5,1)*x+poly_coef(6,1)*b;
    if actual_grain(y,4)==1
        if actual_grain(y,5)==1
            plot_actual(a,1)=actual_grain(y,1);
            plot_actual(a,2)=actual_grain(y,2);
            plot_actual(a,3)=actual_grain(y,3);
            compare(a,1)=actual_grain(y,3);
            compare(a,2)=actual_grain(y,6);

compare(a,3)=(((actual_grain(y,1))^2)+((actual_grain(y,2))^2))^.5;
            compare(a,4)=compare(a,2)-compare(a,1);
            a=a+1;
        end
    end
end

figure,scatter3(raw(:,1),raw(:,2),raw(:,3));
xlabel('X');
ylabel('Y');
zlabel('Z');
axis([-1500 1500 -1500 1500 -3000 0])
grid on
hold on
scatter3(plot_actual(:,1),plot_actual(:,2),plot_actual(:,3),
'filled','MarkerFaceColor',[0 0 0]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
if yes_horiz_edge==1;

x_midpoint=(group_x_limits(2,horiz_edge_group)+group_x_limits(1,horiz_e
dge_group))/2;
    edge_upper_x=x_midpoint;
    edge_lower_x=x_midpoint;
    a=0;
```

```
    while a<3
        a=0;
        edge_upper_x=edge_upper_x+10;
        edge_lower_x=edge_lower_x-10;
        if actual_grain(1,1)>edge_lower_x
            if actual_grain(1,1)<edge_upper_x
                a=a+1;
                actual_grain(1,7)=1;
            end
        end
        if actual_grain(6,1)>edge_lower_x
            if actual_grain(6,1)<edge_upper_x
                a=a+1;
                actual_grain(6,7)=1;
            end
        end
        if actual_grain(11,1)>edge_lower_x
            if actual_grain(11,1)<edge_upper_x
                a=a+1;
                actual_grain(11,7)=1;
            end
        end
        if actual_grain(16,1)>edge_lower_x
            if actual_grain(16,1)<edge_upper_x
                a=a+1;
                actual_grain(16,7)=1;
            end
        end
        if actual_grain(21,1)>edge_lower_x
            if actual_grain(21,1)<edge_upper_x
                a=a+1;
                actual_grain(21,7)=1;
            end
        end
        if actual_grain(26,1)>edge_lower_x
            if actual_grain(26,1)<edge_upper_x
                a=a+1;
                actual_grain(26,7)=1;
            end
        end
        if actual_grain(31,1)>edge_lower_x
            if actual_grain(31,1)<edge_upper_x
                a=a+1;
                actual_grain(31,7)=1;
            end
        end
        if a==1
            c=1;
            for b=1:35
                if actual_grain(b,7)==1
                    compare_horiz_edge(c,1)=actual_grain(b,1);  %%%x
                    compare_horiz_edge(c,2)=actual_grain(b,3);
%%%actual grain-edge
```

```
                    compare_horiz_edge(c,3)=actual_grain(b,6);
%%%predict grain-edge
                    compare_horiz_edge(c,4)=near_wag(1,3);
%%%actual edge
                    x=compare_horiz_edge(c,1);

compare_horiz_edge(c,5)=horiz_line_fit_coeff2(1,1)*x+horiz_line_fit_coe
ff2(1,2);    %%%predicted edge
                    c=c+1;
                end
            end
        end
    end

    a=1;
    for y=1:35
        if actual_grain(y,7)==1
            compare_edgeh(a,1)=actual_grain(y,1);
            compare_edgeh(a,2)=actual_grain(y,2);
            compare_edgeh(a,3)=actual_grain(y,3);
            compare_edgeh(a,4)=actual_grain(y,6);
            a=a+1;
        end
    end

end
if yes_vert_edge==1;

y_midpoint=(group_y_limits(2,vert_edge_group)+group_y_limits(1,vert_edg
e_group))/2;
    edge_upper_y=y_midpoint;
    edge_lower_y=y_midpoint;
    a=0;

    while a<3
        a=0;
        edge_upper_y=edge_upper_y+10;
        edge_lower_y=edge_lower_y-10;
        if actual_grain(1,2)>edge_lower_y
            if actual_grain(1,2)<edge_upper_y
                a=a+1;
                actual_grain(1,8)=1;
            end
        end
        if actual_grain(2,2)>edge_lower_y
            if actual_grain(2,2)<edge_upper_y
                a=a+1;
                actual_grain(2,8)=1;
            end
        end
        if actual_grain(3,2)>edge_lower_y
            if actual_grain(3,2)<edge_upper_y
                a=a+1;
                actual_grain(3,8)=1;
            end
```

```
        end
        if actual_grain(4,2)>edge_lower_y
            if actual_grain(4,2)<edge_upper_y
                a=a+1;
                actual_grain(4,8)=1;
            end
        end
        if actual_grain(5,2)>edge_lower_y
            if actual_grain(5,2)<edge_upper_y
                a=a+1;
                actual_grain(5,8)=1;
            end
        end
        if a==1
            c=1;
            for b=1:35
                if actual_grain(b,8)==1
                    compare_vert_edge(c,1)=actual_grain(b,2);  %%%y
                    compare_vert_edge(c,2)=actual_grain(b,3);
%%%actual grain-edge
                    compare_vert_edge(c,3)=actual_grain(b,6);
%%%predict grain-edge
                    compare_vert_edge(c,4)=front_wag(1,3);
%%%actual edge
                    x=compare_vert_edge(c,1);

compare_vert_edge(c,5)=vert_line_fit_coeff2(1,1)*x+vert_line_fit_coeff2
(1,2);    %%%predicted edge
                    c=c+1;
                end
            end
        end
    end

    a=1;
    for y=1:35
        if actual_grain(y,8)==1
            compare_edgev(a,1)=actual_grain(y,1);    %%% plus 30 for
back edge
            compare_edgev(a,2)=actual_grain(y,2);
            compare_edgev(a,3)=actual_grain(y,3);
            compare_edgev(a,4)=actual_grain(y,6);
            a=a+1;
        end
    end

end
```

References

Ahn, S., J. Choi, N.L. Doh, and W.K. Chung. 2008. A Practical Approach for EFK-SLAM in an Indoor Environment: Fusing Ultrasonic Sensors and Stereo Camera. *Autonomous Robots* 24: 315-335.

Alexia, B.M., A.J. Brislen, J.R. Wicking, and W.J. Frandsen. 2005. Image Processing Spout Control System. US Patent No. 6,943,824 B2.

Bähnisch, C., P. Stelldinger, U. Köthe. 2009. Fast and Accurate 3D Edge Detection for Surface Reconstruction. *Pattern Recognition 31st DAGM Symposium 2009 Proceedings*, 111-120. Paderborn, Germany: DAGM.

Bank, D. 2002. A Novel Ultrasonic Sensing System for Autonomous Mobil Systems. *IEEE Sensors Journal* 2(6): 597-606.

Bank, D., M. Strobel, and E. Prassler. 1999. AutoBed-An Automatically Guided Hospital Bed. *IASTED International Conference on Robotic Applications*, 324-329. Calgary, Alberta: International Association of Science and Technology for Development.

Banner Engineering. 2011. Q&A: Ultrasonic Basics. Available at: http://www.bannerengineering.com/training/faq.php?faqID=34&div=1. Accessed 6 June 2011.

Behnke, W., N. Diekhans, J. Huster, and G. Quincke. 2004. Automatic Adjustment of a Transfer Device on an Agricultural Harvesting Machine. US Patent No. 6,682,416,B2.

Borenstion, J., and Y. Koren. 1991. The Vector Field Histogram-Fast Obstacle Avoidance for Mobile Robots. *IEEE Transactions on Robotics and Automation* 7 (3): 278-288.

Breed, D., V. Castelli, W. Johnson, W. DuVall, R.M. Patel. 1997. Vehicle Occupant Position and Velocity Sensor. US Patent No. 5,653,462.

Brudka, M., and A. Pacut. 2002. Intelligent Robot Control Using Ultrasonic Measurements. *IEEE Transactions on Instrumentation and Measurement* 51(3): 454-459.

Canny, J. 1986. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8(6): 679-698.

Cheng, L., L. Wang, and Y. Ma. 2010. A New Strategy for Boundary Surface Detection in 3D Biomedical Images. *IEEE 2010 3rd International Conference on Biomedical Engineering and Informatics*, 51-55. Piscataway, New Jersey: IEEE.

Cudworth, A.L. 1965. Device for Indicating Objects Rearwardly of a Vehicle. US Patent No. 3,226,673.

Grieg, D. 1964. Distance Measuring Device. US Patent No. 3,122,719.

Knoll, A.C. 1991. Ultrasonic Holography Techniques for Localizing and Imaging Solid Objects. *IEEE Transactions on Robotics and Automation* 7(4): 449-467.

Krautkrämer, J., and H. Krautkrämer. 1969. *Ultrasonic Testing of Materials*. New York, New York: Springer-Verlage.

Langevin, P. 1918. French patent: 505 703.

Laiou, M., A. Meske, and F. Langer. 2009. Method for Controlling Speed and/or Distance in Motor Vehicles. US Patent Application Publication No. US 2009/0228185 A1.

Lu, J. 1997. 2D and 3D High Frame Rate Imaging with Limited Defraction Beams. *IEEE Transactions on Ultrasonics, Ferroelectronics, and Frequency Controls* 44(4): 839-856.

Murino, V., and Trucco, A. 2000. Three-Dimensional Image Generation and Processing in Underwater Acoustic Vision. *Proceedings of the IEEE* 88 (12): 1903-1946.

Nilsson, A., and P. Holmberg. 1994. Combining a Stable 2-D Vision Camera and an Ultrasonic Range Detector for 3-D Position Estimation. *IEEE Transactions on Instrumentation and Measurement* 43(2): 272-276.

Pollklas, M. 1996. Device for Automatic Filling of Containers. US Patent No. 5,575,316.

Pollklas, M. 1998. Device for Automatic Filling of Load Containers. US Patent No. 5,749,783.

Prassler, E., J. Sholz, M. Strobel, and P. Fiorini. 1999. An Intelligent (Semi-) Autonomous Passenger Transportation System. *IEEE International Conference on Intelligent Transportation Systems*: 374-379. Piscataway, New Jersey: IEEE.

Quistgaard, J.U. 1997. Signal Acquisition and Processing in Medical Diagnostic Ultrasound. *IEEE Signal Processing Magazine* 14: 67-74.

Schueller, C.F., H. Lee, and G. Wade. 1984. Fundamentals of Digital Ultrasonic Imaging. *IEEE Transactions on Sonics and Ultrasonics* 31(4): 195-217.

USDA-NASS. 2012. Crop Production. Washington, D.C.: USDA National Agricultural Statistics Service.

Yi, S., D. Labate, G.R. Easley, and H. Krim. 2009. A Shearlet Approach to Edge Analysis and Detection. *IEEE Transactions on Image Processing* 18(5): 929-941.

Ziou, D., and S. Tabbone. 1998. Edge Detection Techniques-An Overview. *International Journal of Pattern Recognition and Image Analysis* 8(4): 537-559.