Robotics Project Report: LIDAR

Matt Delaney, Ian Averman, Derek Schumacher, Taylor Nightingale

Design:

Task 1: Specify System Specification

In order for the MARS (Multi-unit Autonomous System) to most effectively traverse the obstacle zone at the competition, the transport device must be able to locate all rocks larger than 15 cm and pits deeper than 15 cm in the obstacle zone. A LIDAR-based system will be developed in order to accomplish this task, so that the MARS can successfully avoid such hazards. Finding smaller rocks and pits is desirable but not required if doing so would compromise the system's ability to find larger obstacles. The system should also be able to detect these obstacles in 15 seconds or less.

Task 2: Specify the Required Hardware Components

In order to recognize obstacles in the competition zone we will be using LIDAR--specifically, the Hokuyo UTM-30LX Scanning range finder. This LIDAR unit is rated for scanning in an indoor environment, has a 30 meter range, and 270° scanning area [1]. At the competition, the scans will be done in a diffuse natural lighting environment. For the purposes of this report, data was collected in an indoor fluorescent light environment. This difference in environmental light is not a concern at this time.

After the measurements were taken, the processing was run on a desktop computer in MATLAB running Windows 7 with 8GB of RAM and a 3.5 GHz Intel i5 processor. Since the image processing is currently performed using MATLAB, the program runs very slow. In addition, the majority of the logged data being read into MATLAB is not being used. The large amount of logged data causes the algorithm to take around 40 seconds to run currently, of which 30 seconds are the reading in of this logged sensor data. In the final system, when equipped with a live sensor and less logged data, the system is expected to be able to detect obstacles in less than 10 seconds. For the purposes of prototyping, the processing is not being done in real time and has no time constraint at this time. As such, we have no minimum required CPU speed.

Task 3: Required Image Processing Operations

An algorithm was developed to extract the elevation angle from the 2D laser scanning plane (LIDAR_CSV_v8.m, lines 39-104). This algorithm relies on a specific image feature shown in Figure 1 that can be used with trigonometry to determine the angle of elevation. In the competition, this step will not be necessary; the LIDAR mount will be able to provide the elevation angle relative to the robot. With the angle of elevation and LIDAR-provided azimuth

and distance (radius), the spherical coordinates of each point were obtained and converted to cartesian coordinates.

After the data was converted, a point map was stitched together using the `scatter3` command in MATLAB. Figure 2 shows the results of the initial plotting. However, this point cloud has over 220,000 points, so to speed in the visualization of the data, a meshgrid was created based on a scattered interpolant function, and displayed using the `mesh` command. The results of this operation are displayed in Figure 3. In order to smooth the noise found in our data, a 2x2 maximum box filter is applied. This filter was chosen because it gave the data the best smoothing while preserving the data.

Implementation:

Task 4: Feature Extraction Algorithms

Since the pits and rocks are not uniformly deep or tall, feature extraction based on height is not easily obtainable without modification of the data. In order to easily find the features, the image is modified to contain three values. These values are obtained using thresholding.

Maximum and minimum threshold values are chosen based on the desired obstacle height or depth. A pass is run over the data to set any Z-values above or below the respective thresholds to that threshold value. Any value in-between the thresholds are set to the midpoint between the two thresholds. The result of this transformation is shown in Figure 4.

After turning the image into this ternary form, the MATLAB function `regionprops` is used to find the diameter, location, and orientation of each remaining region two pixels or greater. The resulting ovals are then drawn over the LIDAR data, as shown in Figure 5.

Task 5: Representation and Recognition

The features are represented as circles on the two-dimensional grid of our competition zone. The origin is placed at the bottom center of the competition zone. The obstacle type (pit or rock), size (diameter) and centroid (position) are known; these are the data types necessary for the robot to autonomously avoid the obstacles. A graphical representation is provided for human operators so that system performance can be evaluated. Once the system is fully developed, no MATLAB graphical interface will be necessary; only the centroid and diameter should be necessary.

Task 6: System Performance Assessment

Although the system was not perfect, it did meet several of our goals. The data gathered from the LIDAR took around 30 seconds to read into MatLab. However, this should not be a

problem for our robot, as the data will be available through direct access. The processing time took less than 10 seconds, so completely met our 15 second limit. The data gathered from the LIDAR was noisey, yet gave good results after smoothing and thresholding. Lastly, the walls of the arena were slightly skewed, and perceived by the algorithm to be large rocks. Since the robot should still avoid the walls, the results are still useful, despite the misinformation.

Summary:

A system was developed to take 2D LIDAR data and, using a specific image feature, extract elevation. With the LIDAR plane and the elevation angle, a cartesian point cloud representation of the data was generated. This point cloud was processed to identify pit and boulder regions; these regions were analyzed using standard region processing techniques to extract obstacle size and position information in a format that could be passed to a pathfinding algorithm. The system performance exceeds expectations and has the potential for further improvement.

References:

[1] ***Hokuyo UTM 30LX Scanning Range Finder*. Available at** https://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html

[2] ***NASA's Fifth Annual Robotic Mining Competition Rules & Rubrics 2014*. Available at** http://www.nasa.gov/sites/default/files/robotics_mining_competition_rules_2014_1.pdf

Appendix:

Figures:

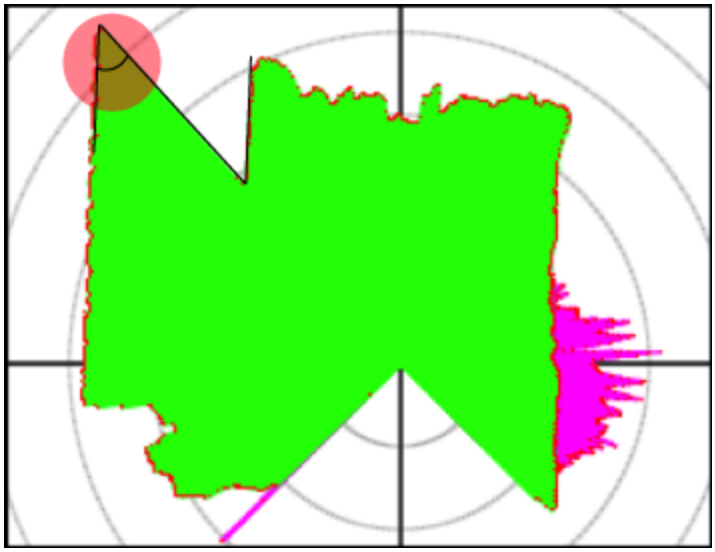

Figure 1. A planar scan using the LIDAR. The highlighted angle was used to determine the angle of measurement.

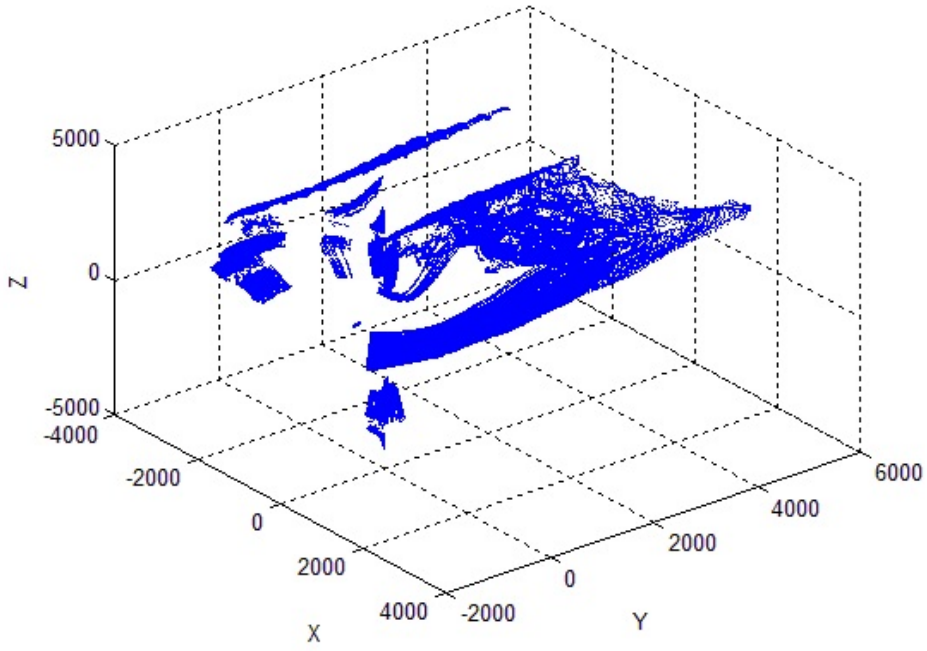

Figure 2. Point Cloud of LIDAR Data.

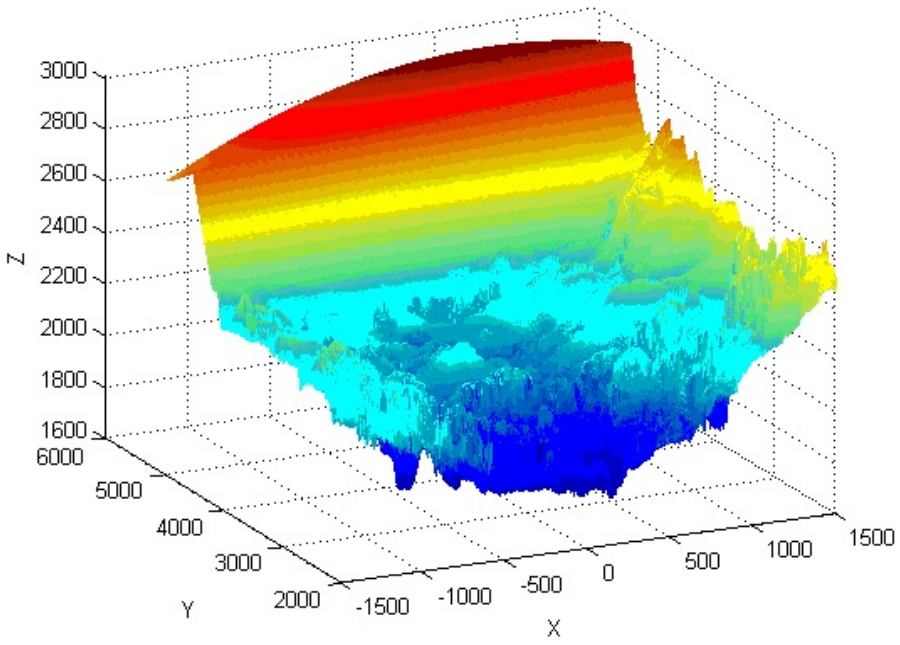

Figure 3. A meshgrid visualization of the LIDAR data. The color corresponds to the Z-value of the data point.

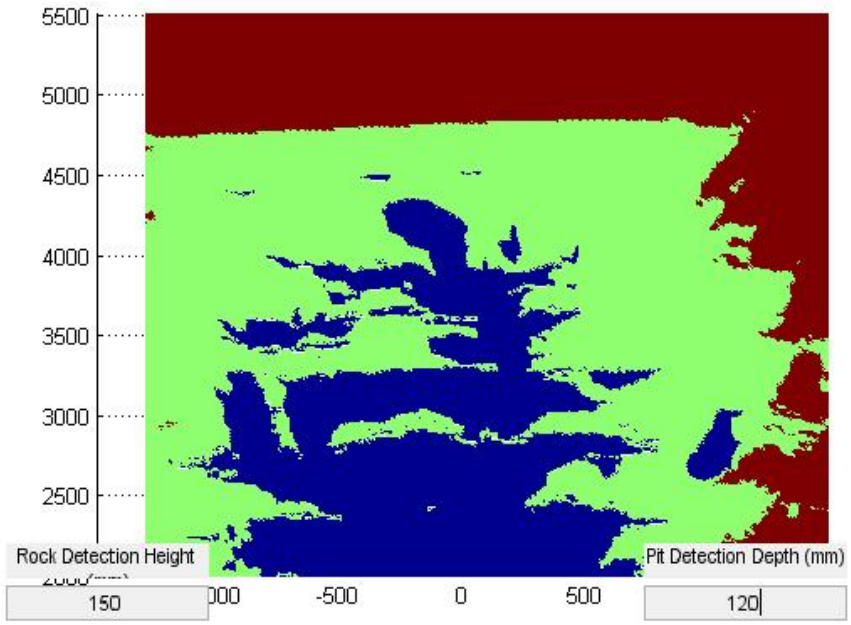

Figure 4. Thresholded Mesh Data. Blue represents low (pit) obstacles; red represents high (boulder) obstacles.

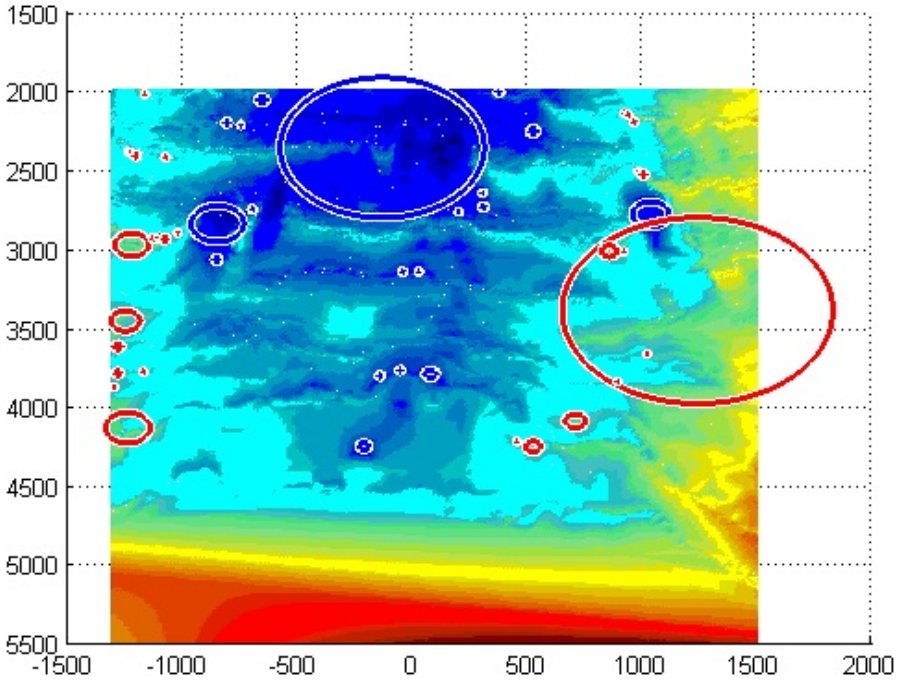

Figure 5. Detected Obstacles Overlaid on the Meshgrid.

MatLab Code:

```
function LIDAR_CSV_v8(~)
%Implemented:
% -Read / compute Input Data (distance and angle) (Ian Averman / Matt
% Delaney)
% -Sample code to compute (X,Y) coordinates for each point in a scan (Ian
% Averman)
% -Sample code to compute the top-left angle in a scan (Matt Delaney)
%Todo:
% -Improve left point selection using a median filter of some sort
% -Compute angles for every scan that has useable data
% -Determine which frames have usable data
   % -Possibly by detecting if By-Cy and Bx-Cx are sufficiently large

close all
clc

file = 'C:\Users\delaney15\Desktop\2014_11_11_10_54_57_359.csv';
global DATA;
if 1 ~= exist('DATA','var')
    DATA = xlsread(file);
end
[rows,cols] = size(DATA);

slits = rows-5;
```

```
scans = cols/2;
resol = 270/slits;
ANGLE = -45:resol:225;
if 1 ~= exist('DIST','var')
    DIST=zeros(1081,cols/2);
    %DIST (Scan, Data)
    for i = 1:scans
        DIST(:,i) = DATA(5:slits+5,2*i-1);
        %INTENS(:,i) = DATA(5:slits+5,2*i);
    end
end
%test scans are 1221 to 1427 on file 2014_11_11_10_54_57_359.csv
%SCAN Corresponds to scan # (SCAN - 1)
SCAN_MIN = 1222;
SCAN_MAX = 1428;

%preallocate memory for speed.
X = zeros(1,slits);
Y = zeros(1,slits);
%absolute (relative-to-the-arena) positions
Xabs = zeros(1,slits*(SCAN_MAX-SCAN_MIN));
Yabs = zeros(1,slits*(SCAN_MAX-SCAN_MIN));
Zabs = zeros(1,slits*(SCAN_MAX-SCAN_MIN));

for SCAN=SCAN_MIN:1:SCAN_MAX

    for j = 1:slits
        X(j) = DIST(j,SCAN)*cosd(ANGLE(j));
        Y(j) = DIST(j,SCAN)*sind(ANGLE(j));
    end

    % Building this triangle to find angle c:
    %      C
    %     / \
    %    /   B
    %   /
    %  /
    % A
    %
    % A is the due-west pixel. C is the top-left-most pixel. B is the pixel
    % where the "Jump" occurs.
    % Will use the law of cosines (http://en.wikipedia.org/wiki/Law_of_cosines)
    % to calculate.


    % The slit value for "due west" is roughly 900.
    %TODO: Change the slitA value dynamically (median filter?).
    slitA = 850; %changed from 900!!!!!!!
    Ax = X(slitA);
    Ay = Y(slitA);

    %The top-left pixel is the pixel in the top-left quadrant with the largest
    %distance value.
    [cDistGuess, cSlitGuess] = max(DIST(565:890,SCAN));
    %Correct for the fact that MATLAB thinks slit 565 is slit 1
    cSlitGuess = cSlitGuess + 565;
    %Find x and y values
```

```
    Cx = X(cSlitGuess);
    Cy = Y(cSlitGuess);

    %Find the bottom corner from the "sudden drop"
    [bDistGuess, bSlitGuess] = min(DIST(540:cSlitGuess,SCAN));
    bSlitGuess = bSlitGuess + 539;
    % We now have x and y values.
    Bx = X(bSlitGuess);
    By = Y(bSlitGuess);

    % Calculate the angle!
    AB = pdist([Ax,Ay;Bx,By],'euclidean');
    BC = pdist([Bx,By;Cx,Cy],'euclidean');
    AC = pdist([Ax,Ay;Cx,Cy],'euclidean');
    DISTANCE_FROM_LEFT_WALL = 1850;
    HEIGHT = 1100;
    %Law of cosines!
    cosOfAngleC = (AB*AB - BC*BC - AC*AC)/((-2)*AC*BC);
    angleC = acosd(cosOfAngleC);

    %calculate distance to the wall point
    y = DISTANCE_FROM_LEFT_WALL * sind(90-angleC) / sind(angleC);
    elevation = atand(HEIGHT / y);
    %code below is under development


    %we have spherical coordinates. Convert to XYZ.
    for j = 1:slits
        [Xt,Yt,Zt] = sph2cart(ANGLE(j)*(pi/180),(elevation)*(pi/180),DIST(j,SCAN));
        Xabs(j+slits*(SCAN-SCAN_MIN)) = Xt;
        Yabs(j+slits*(SCAN-SCAN_MIN)) = Yt;
        Zabs(j+slits*(SCAN-SCAN_MIN)) = Zt;
    end
end

%resolution to create the mesh at.
MESH_RESOLUTION = 500;
%Hard-coded values to try and focus on the arena surface and not the walls.
%Ian: X(-2750,2000),Y(2000,6000)
%FieldOnly: X(-1500,1500),Y(2000,5500)
XMIN = -1300;
XMAX = 1500;
YMIN = 2000;
YMAX = 5500;
xlin = linspace(XMIN,XMAX,MESH_RESOLUTION);
ylin = linspace(YMIN,YMAX,MESH_RESOLUTION);
%xlin = linspace(min(Xabs),max(Xabs),MESH_RESOLUTION);
%ylin = linspace(min(Yabs),max(Yabs),MESH_RESOLUTION);
[Xmesh,Ymesh] = meshgrid(xlin,ylin);
f = scatteredInterpolant(Xabs.',Yabs.',Zabs.');
Zmesh = f(Xmesh,Ymesh);
figure(1);
mesh(Xmesh,Ymesh,Zmesh);
xlabel('X');
ylabel('Y');
zlabel('Z');
```

```
%2x2 max filter
ZmeshFiltered = ordfilt2(Zmesh,4,ones(2,2));
figure(2);
xlabel('X');
ylabel('Y');
zlabel('Z');
title('2D box maximum filter');
mesh(Xmesh,Ymesh,ZmeshFiltered);
view([0 -90]);
%average value is 2266, although the walls mess with this quite a bit.
% median is roughly 2170 for the test data.
medianValue = median(median(ZmeshFiltered(:)));
%low pass: 2170 - detection level (mm)
low_pass_offset = 230;
LOW_PASS_THRESHOLD = medianValue-low_pass_offset;
%high pass: 2170 + detection level (mm).
high_pass_offset = 70;
HIGH_PASS_THRESHOLD = medianValue+high_pass_offset;
% Apply the thresholds
ZmeshThresholdedLOW = ZmeshFiltered < LOW_PASS_THRESHOLD;
ZmeshThresholdedHIGH = (ZmeshFiltered > HIGH_PASS_THRESHOLD);
ZmeshThresholded = (ZmeshThresholdedLOW *(-1000)) + (ZmeshThresholdedHIGH * (1000));
figure(3);
subplot(1,2,1);
title('Basic Threshold Filter');
mesh(Xmesh,Ymesh,ZmeshThresholded);
xlabel('X');
ylabel('Y');
zlabel('Z');
view([0 90]);
%add some user interface tools
low_pass_box = uicontrol('Style', 'edit',...
    'Position', [400 0 120 20],...
    'String', num2str(low_pass_offset),...
    'Callback', @LowPassThreshUpdate);
low_pass_box_label = uicontrol('Style','text',...
    'Position',[400 25 120 20],...
    'String','Pit Detection Depth (mm)');
high_pass_box = uicontrol('Style', 'edit',...
    'Position', [20 0 120 20],...
    'String', num2str(high_pass_offset),...
    'Callback', @HighPassThreshUpdate);
high_pass_box_label = uicontrol('Style','text',...
    'Position',[20 25 150 20],...
    'String','Rock Detection Height (mm)');
figure(4);
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Facility Depth Mesh with Pits highlighted');
mesh(Xmesh,Ymesh,ZmeshFiltered);

extractObstacles(ZmeshThresholdedLOW, 'b');
extractObstacles(ZmeshThresholdedHIGH, 'r');
    function extractObstacles(inputThresholdedMesh,circleColor)
        %FEATURE EXTRACTION TIME
        global PitMatrix;
```

```
        PitMatrix = bwmorph(inputThresholdedMesh, 'erode');
        newRows = YMAX - YMIN;
        newCols = XMAX - XMIN;
        PitMatrix = imresize(PitMatrix, [newRows newCols]);
        % Only pay attention to the obstacle area
        PitMatrix = PitMatrix(1:2500,:);
        global PIT_STATS;
        PIT_STATS = regionprops(PitMatrix, 'Area', 'Centroid', 'MajorAxisLength',
'MinorAxisLength', 'Orientation','EquivDiameter');

        %imagesc(flip(PitMatrix,1));
        hold on
        for k= 1:size(PIT_STATS)
            %draw circles to the plots we've made indicating where the pits
            %are
            figure(4);
            viscircles(PIT_STATS(k).Centroid + [XMIN YMIN], PIT_STATS(k).EquivDiameter
/2,'LineStyle','-', 'EdgeColor',circleColor);
            figure(3);
            subplot(1,2,1);
            viscircles(PIT_STATS(k).Centroid + [XMIN YMIN], PIT_STATS(k).EquivDiameter
/2,'LineStyle',':','EdgeColor',circleColor);
            subplot(1,2,2);
            viscircles(PIT_STATS(k).Centroid + [XMIN YMIN], PIT_STATS(k).EquivDiameter
/2,'LineStyle',':','EdgeColor',circleColor);
        end
        figure(4);
        view([1 -90]);
        hold off

    end



        function LowPassThreshUpdate(src, ~)
            updateFilterVal=get(src,'String');
            if isempty(str2double(updateFilterVal))
                set(src,'string','0');
                warndlg('Input must be numerical');
            else
                low_pass_offset = str2double(updateFilterVal);
                %recalculate the thresholds
                LOW_PASS_THRESHOLD = medianValue-low_pass_offset;
                %high pass: 2170 + detection level (mm).
                HIGH_PASS_THRESHOLD = medianValue+high_pass_offset;
                ZmeshThresholdedLOW = (ZmeshFiltered < LOW_PASS_THRESHOLD);
                ZmeshThresholdedHIGH = (ZmeshFiltered > HIGH_PASS_THRESHOLD);
                ZmeshThresholded = (ZmeshThresholdedLOW *(-1000)) + (ZmeshThresholdedHIGH *
(1000));
                figure(3);
                mesh(Xmesh,Ymesh,ZmeshThresholded);
                view([0 90]);
            end
        end
            function HighPassThreshUpdate(src, ~)
            updateFilterVal=get(src,'String');
            if isempty(str2double(updateFilterVal))
```

```
                set(src,'string','0');
                warndlg('Input must be numerical');
            else
                high_pass_offset = str2double(updateFilterVal);
                %recalculate the thresholds
                LOW_PASS_THRESHOLD = medianValue-low_pass_offset;
                %high pass: 2170 + detection level (mm).
                HIGH_PASS_THRESHOLD = medianValue+high_pass_offset;
                ZmeshThresholdedLOW = (ZmeshFiltered < LOW_PASS_THRESHOLD);
                ZmeshThresholdedHIGH = (ZmeshFiltered > HIGH_PASS_THRESHOLD);
                ZmeshThresholded = (ZmeshThresholdedLOW *(-1000)) + (ZmeshThresholdedHIGH *
(1000));
                 figure(3);
                mesh(Xmesh,Ymesh,ZmeshThresholded);
                view([0 90]);
            end
        end
%figure(3);
%scatter3(Xabs,Yabs,Zabs,1,'.');
%xlabel('X');
%ylabel('Y');
%zlabel('Z');
   end
```