ENGR3390: Fundamentals of Robotics Report 2
SENSE Lab

Web Cam and LiDAR (from left to right)

Octopus Team: Alexis Wu, Braden Vaccari, Dowling Wong, Eliyahu Suskind, and Mahderekal Regassa

Report Author: Mahderekal Regassa

April 1, 2023

Table of Contents **Page**

SENSE Lab

In this lab, we were given a suit of sensors to use with the goal of finding a target different color targets (or holes) on a rotating turntable. Generally, sensors allow robots to find targets or holes to then either follow/ avoid a target or go through a hole. Sensors also help the robot answer these fundamental questions: “Where am I”, “What’s around me?” and “How am I?”.

Web Camera

Theory

We used a high resolution low cost Microsoft Lifetouch Cinema USB Web Camera for this lab. It automatically adjusts exposure for brightness and provides a 360° rotation for an all-rounded view. Leveraging the high quality images from this sensor, we use Matlab which stores images in matrix. Each element of the matrix usually stores the color for a pixel of an image.

Fig.1.0 Lifetouch USB Web Cam with 6 Sharp IR sensors

Code

After plugging the web cam to our computer using a USB, we connect it to matlab and get started.

StepUp

```
clc       % clear command window
clear     % clear MATLAB workspace
imaqreset % clear images from memory
```

Set Up Control System

```
% Detect and create webcam object
camList = webcamlist
roboCam = webcam(2)
setUpWebcam(roboCam);

% Test camera, wait for user confirm
preview(roboCam);
CamTest = input( ...
    "Press enter when ready to rumble.","s" ...
    );
clc;
closePreview(roboCam);
```

- Adjust the whitebalance or exposure in the sense function

```
function setUpWebcam(roboCam)
% Sense function to set up roboCam balance mode
% Input: roboCam is webcam object to be set up

    roboCam.WhiteBalanceMode = "manual";
    roboCam.Brightness = 100;

end
```

- Play with the imtool, create a standalone figure for images, and setup number of tests for the control loop

```
% Test IMTool, wait for user confirm
img = snapshot(roboCam);
imtool(img);
CamTest = input( ...
    "Experiment with IMTool, then hit enter when ready","s" ...
    );
clc;

% Create free floating figure for livescript update help
camWindow = figure( ...
    "name","Robot USB Camera","NumberTitle","off","Visible","on" ...
    );
figure(camWindow)
imshow(img,"Border","tight")
```

```
% Configure loop to test n data points
nTest = input( ...
    "DO NOT CLOSE FIGURE. Enter small number of camera tests, " + ...
    "and hit enter when ready: " ...
    );
clc;
r= rateControl(0.1);
reset(r);
```

Robot Control Loop

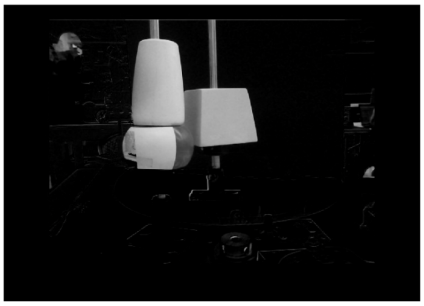
```
controlFlag = 1;

while (controlFlag < nTest +1)  % loop till ntests data captured
    % SENSE finds centroid and area of target
    [centroids, targetArea] = SENSE(roboCam, camWindow)

    CamTest = input("Press Enter to advance tests.","s");
    clc;

    waitfor(r);
    controlFlag = controlFlag + 1;
end
```

- Using Matlab's Image Processing Toolbox, we created a masking function to remove the noise and **find the target.** The image shown in black and white has the target in white.

```
% reading images
img1 = imread('background.jpg');
img2 = imread('withObjects.jpg');
```

```
% display color images
figure
imshow(img1)
figure
imshow(img2)
```

```
% convering image to grayscale
img1BW = rgb2gray(img1);
img2BW = rgb2gray(img2);
```



```
% subtract images
imgDiff = abs(img2BW -img1BW);
figure
imshow(imgDiff)
```

Robot Functions

- We created a colorMask function using the color threshold app, meaning the program tracks yellow color targets.

Sense Functions

```
function [centroids, targetArea] = SENSE(roboCam, myFigure)

% Create filtered webcam image, prompt user input
robotImage = snapshot(roboCam);
[targetMask, yellowMasked] = yellowMask2(robotImage);
figure(myFigure)
imshow(targetMask)
CamTest = input("Ensure filter is OK, then hit enter.","s");
clc;

% Fill noise, prompt user input, if bad re-do woth new fill value
fillFlag = 1;
testPixels = 30;
while fillFlag
cleanImage = bwareaopen(targetMask,testPixels);
figure(myFigure)
imshow(cleanImage)
testPixels = input(...
    "If filter is OK, hit enter. Else enter size in pixels of noise: " ...
    );
clc;
```

```
if isempty(testPixels)
    break
end
end

% Find centroid, prompt user input
targetCenter = regionprops(cleanImage, 'centroid');
centroids = cat(1, targetCenter.Centroid);
figure(myFigure)
imshow(robotImage)
hold on
plot(centroids(:,1), centroids(:,2), 'b*')
text(centroids(:,1), centroids(:,2), 'Centroid');
hold off
CamTest = input('Check out target center and hit enter.', 's');
clc;
```

```
% Find object area(s), prompt user input
targetArea = regionprops(cleanImage, 'area');
area = targetArea.Area;
centroids = cat(1, targetCenter.Centroid);
figure(myFigure)
imshow(robotImage)
```

```
hold on
plot(centroids(:,1), centroids(:,2), 'b*')
text(centroids(:,1), (centroids(:,2)+10), num2str(area));
hold off
CamTest = input('Check out target area and hit enter.', 's');
clc;

end

function cleanImage = noImSense(roboCam, myFigure, testPixels)
% Sense without IMShows or Centroid/Area Finding for Live Cam

% Create filtered webcam image, prompt user input
robotImage = snapshot(roboCam);
[targetMask, yellowMasked] = yellowMask2(robotImage);
figure(myFigure)
clc;

% Fill noise in filtered image
cleanImage = bwareaopen(targetMask,testPixels);
figure(myFigure)
clc;
end
```

```
function setUpWebcam(roboCam)
% Sense function to set up roboCam balance mode
% Input: roboCam is webcam object to be set up

    roboCam.WhiteBalanceMode = "manual";
    roboCam.Brightness = 100;

end
```

Results

- Full binary mask using ColorThresholder app; first image shows yellow mask applied to the image with the yellow target shown in a white mask.

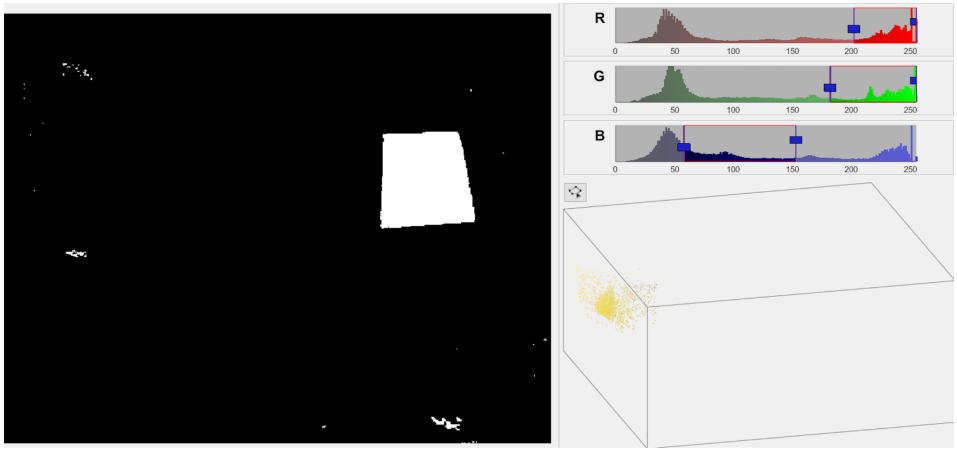


Fig.1.2 White mask for a yellow object

LiDAR

Theory

Hokuyo LIDAR range sensor for this lab has a 240° polar range measurements with a 0.36° pitch and 10m range. The distance between this piece of equipment and any object/ wall infront of it is calculated by the measure of the phase difference between the projected and reflected infrared laser light. It acquires a wide range of data for extreme accuracy.

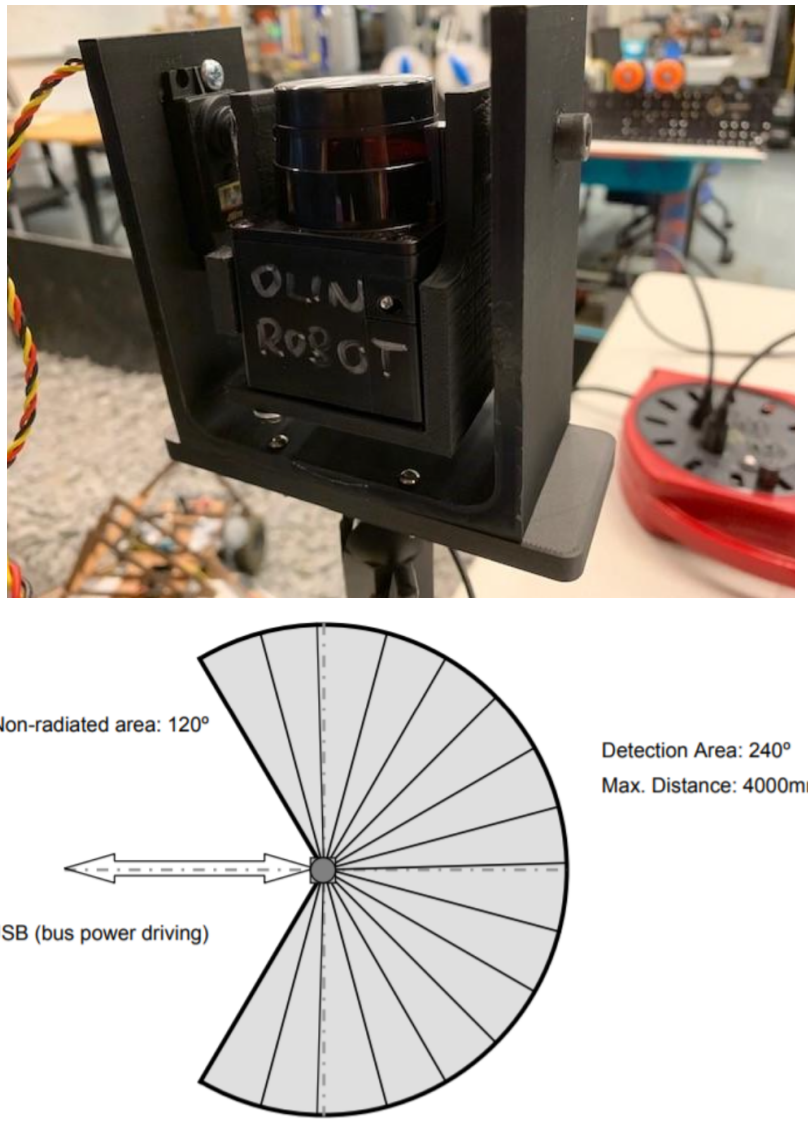


Fig.1.3 Hokuyo LiDAR Sensor (top image) and scanning zone (bottom image)

Code

Live Scan

We first connect to the LiDAR through a USB cable and bluetooth, then install the UrgBenriPlus installer to get live LiDAR scans from it.

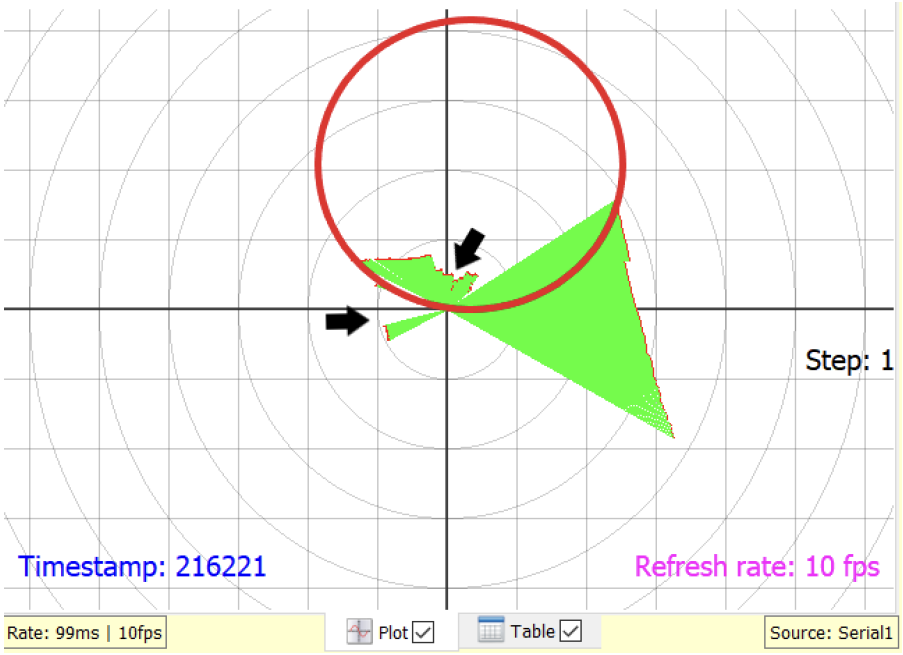


Fig.1.4 UrgBenriPlus live snapshot of the environment; red circle encompasses the side where the scanner is pointing from

The code for the LiDAR is throughly documented so I will refrain from adding more explanation.

StepUp

Mahdi, 3/8/2023, Personal Revision

```
clc       % clear command window
clear     % clear MATLAB workspace
```

Set Up Lidar

```
disp("program running"); % indicate lidar code starts ok
warning('off', 'all');   % temporarily turn matlab warnings off as it will
                         % throw nonfatal errors when you run the program
                         % becasue the code is a bit wonky

% there might be old objects in the lidar serial object so we need to find
% it and delete to prevent any conflicts
serialObjs = instrfind; % reads serial port objects
if ~isempty(serialObjs) % a conditional to check if there are any old objects
    fclose(serialObjs); % closes the serial port connected to old Lidar
    delete(serialObjs); % delete the old lidar objects
end

%clearing any old objects to avoid conflicts
clear
clc
```

Specifiying the USB serial port that the Hokuyo Lidar is plugged into

---

```
comPort = "COM4";    % COM port that the lidar is connected to
                     % (from windows settings)
disp("Com Port Set") % display that we're all set
```

Connect to Lidar and setting serial communication parameters

```
disp("connecting to lidarand setting Lidar parameters");
disp("expect a short delay while setting...");

lidar = serial(comPort, 'baudrate', 115200); % creatig a serial port object called lidar
set(lidar, 'Timeout', 2); % setting communication link timeout
set(lidar, 'InputBufferSize', 200000); % setting data input buffer size
set(lidar, 'Terminator', 'LF/CR'); % setting data stream terminator for fprintf and fscanf

% serial communications complexity for devices are driven by the
% manufacturer. In this case we are setting Hokuyo parameters to configure Lidar
fopen(lidar); % connects Lidar or the serial port object
pause(0.3); % pause for command transmission
fprintf(lidar, 'SCIP2.0'); % writes a string cmd to lidar
pause(0.3); % pause for the command to be sent
fscanf(lidar); % reads ASCII data from the device connected to Lidar
fprintf(lidar, 'W');
pause(0.3);
fscanf(lidar);
fprintf(lidar,'BW');
```

```
pause(0.3);
fscanf(lidar);
fprintf(lidar, 'MD0044072500');
pause(0.3);
fscanf(lidar);
clc
disp("Lidar set")
% in the code above, we essentially create the serial port object (lidar)
% and configure it to allow serial communication with Hokuyo through the
% USB port. In the second chunk of the code, we open that communication
% link and tell (send it coded texts) the Hokuyo lidar to send us back a
% set of range data
```

Inititalize Lidar display figure Window

```
% here, we are creating a stand-alone figure window that will be used to
% plot once we get the Lidar data
LaserPlot1.figure = figure('Name', 'Hokuyu URG-04LX data', 'NumberTitle', 'off', ...
    'MenuBar', 'figure', 'units', 'normalized', 'Visible', 'on');
LaserPlot1.axis1 = axes('parent', LaserPlot1.figure, 'units', 'normalized', 'NextPlot', ...
    'replace');
grid(LaserPlot1.axis1, 'on')
```

```
LaserPlot1.axis1.Title.String='Laser Scans';
LaserPlot1.axis1.XLabel.String='X Axis';
LaserPlot1.axis1.YLabel.String='Y Axis';

% we create a primitive line object which is a matlab synatax that plots a line
% based on x and y data. In this case our x and y data is XData and YData
% respectively and both of them are empty vectors waiting to be filled with
% Lidar data
laserRange = line('Parent', LaserPlot1.axis1, 'XData', [], 'YData', [], 'LineStyle', ...
    'none', 'marker', '.', 'color', 'b', 'LineWidth', 2);
grid on
axis([-3000 3000 -3000 3000])
xlabel('x (mm)')
ylabel('y (mm)')
disp("Laser Scans figure set");
```

Sample and Visualize Data

This code's main task is to read lidar range data over serial link and plot it to a figure

```
disp("Read and Plot Lidar Data, type and hold cntrl-c to stop")
angles = (-120:240/682:120-240/682)*pi/180; % convert sensor steps (distances) to angles
tStart = tic; % start experiment time using the tic function
iscan = 1;
while (iscan == 1) % this loop continous for 60 seconds
    [A] = FunRoboLidarScan(lidar); % lidar scan range data is stored in [A]
    laserRange.XData = A .* cos(angles); % finding the x coordinate using trig
    laserRange.YData = A .* sin(angles); % finding the y coordinate using trig
    drawnow % draws laserRange in the Laser Scan window
    pause(0.2) % pausing for serial communication to keep up
    tElapsed = toc(tStart); % returns time elapsed since the tic function was triggered
    if (tElapsed > 60) % condition checks to make sure the experiment doesn't run for
                       % longer than 60 seconds
        iscan = 0;     % break the loop by setting iscan variable to 0 once condition is met
    end
end
disp("Laser scan ended");
```

- The LiDAR scan data once processed in the code above, we have it plotted nicely in the empty floating window we created earlier as shown below. Other than what's annotated in the images below, the rest is random obstacles, including chairs, people, etc.

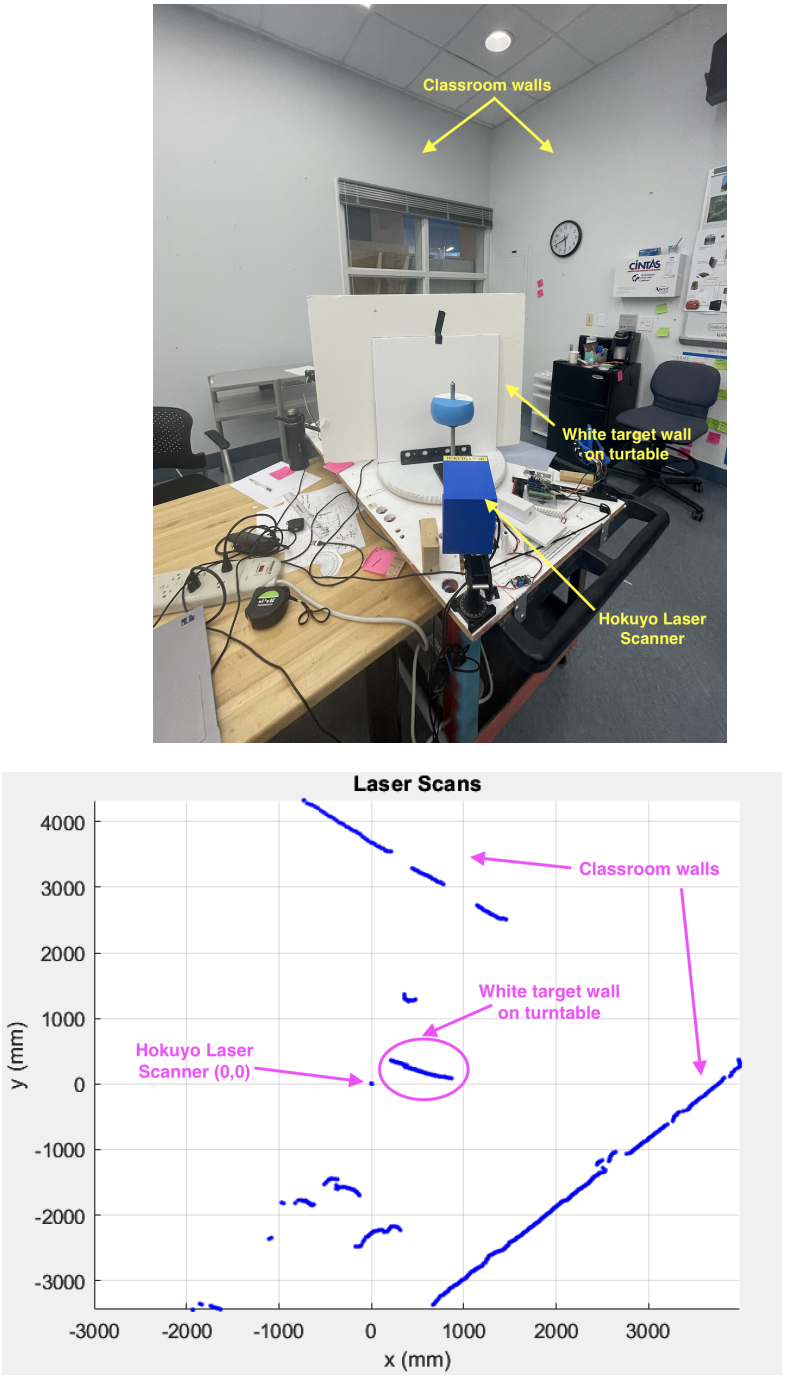


Fig.1.5 What we see from the classroom in the direction that the LiDAR is pointed (top image) vs what the LiDAR sees and plots

Find Object

- After this our target is to detect objects infront of the white wall and return it's x and y coordinates.

Run this code first with no objects infront of the wall

```
% capture the laser x and y coordinate readings for the wall with no
% objects infront on x1 and y1 and save the data on wallonlydata.mat
x1 = [laserRange.XData]
y1 = [laserRange.YData]
save('wallonlydata','x1','y1')
```

This code plots the data for wall without obstacles in front

```
% set x and y limits to take the data we only need and disregard the rest
x1highestlim = 700
x1lowestlim = 0
y1highestlim = 400
y1lowestlim = -400

% creating a matrix of x1 and y1 and apply the limits
combinedata = [x1;y1]
combinedata = combinedata(:,combinedata(1,:)>x1lowestlim)
combinedata = combinedata(:,combinedata(1,:)<x1highestlim)
combinedata = combinedata(:,combinedata(2,:)>y1lowestlim)
combinedata = combinedata(:,combinedata(2,:)<y1highestlim)
```

```
% plots on the previous diagram
hold on
wallplot = line('Parent', LaserPlot1.axis1, 'XData', combinedata(1,:), 'YData', ...
    combinedata(2,:), 'LineStyle', 'none', 'marker', '.', 'color', 'r', 'LineWidth', 2);
```

- The mask the rest of the noise and plotting only the area of focus (target wall infront of it), we get this plot:

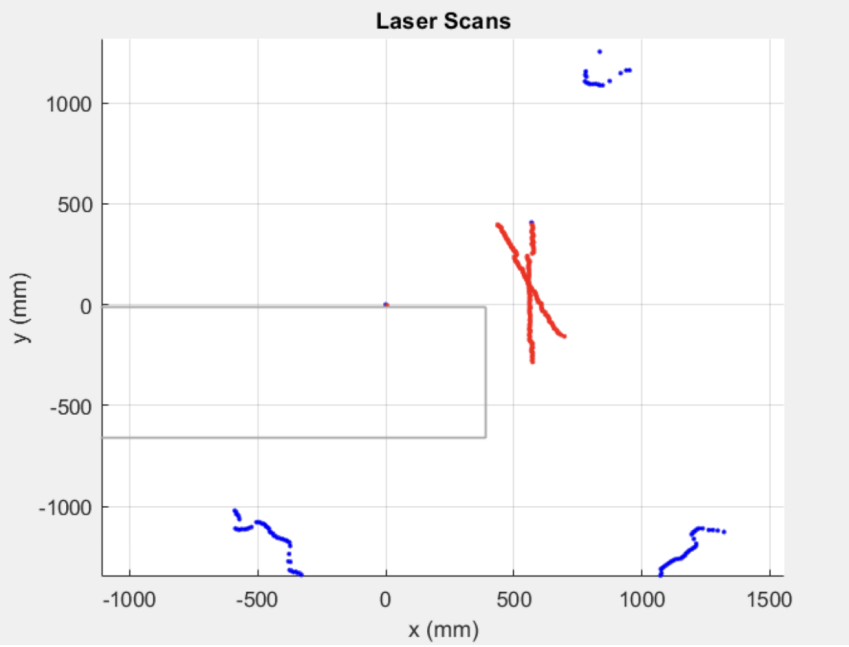


Fig. 1.6 Laser scan with the area of focus plotted in red

Run this code after placing an object infront of the wall

```
% new x and y data with obstacles infront of the wall
x2 = [laserRange.XData]
y2 = [laserRange.YData]
combinedata2 = [x2;y2]

% creating a matrix of x2 and y2 and apply the limits
combinedata2 = combinedata2(:,combinedata2(1,:)>x2lowestlim)
combinedata2 = combinedata2(:,combinedata2(1,:)<x2highestlim)
combinedata2 = combinedata2(:,combinedata2(2,:)>y2lowestlim)
combinedata2 = combinedata2(:,combinedata2(2,:)<y2highestlim)

% plots again on the same plot
hold on
wallplot = line('Parent', LaserPlot1.axis1, 'XData', combinedata2(1,:), 'YData', ...
combinedata2(2,:), 'LineStyle','none', 'marker', '.', 'color', 'g', 'LineWidth', 2);
```

- We repeat the same process for wall with objects infront of it and we get a masked plot over the previous one, in green. To note, the code is a bit finicky and when the LiDAR glitches, the code does too which is why I have double red lines for these plots. The desired plot is the small green line (object reading) around x=250mm.

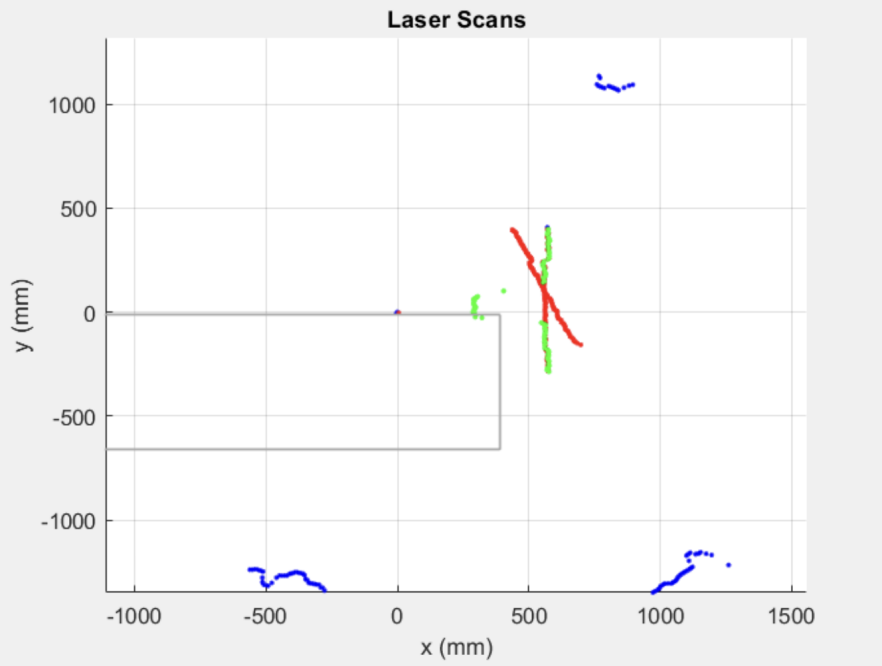


Fig. 1.7 Laser scan with the area of focus (object readings) plotted in green

```
% we subtract the two combinedata matrices to get the values of the object
% (minus the wall coordinates)
mask = combinedata - combinedata2
object = zeros(1,200);

% iterating through the matrix with objects (combinedata2) and taking the x
% values above a certain treshold
for index = 1 : length(combinedata2)
    x = mask(1,:);
    if x(index) > 200
       object(index) = x(index);
    %disp("object is at " + object)
       end
end
% finding the middle point of the object by taking the mean of the max and
% min x values we now have
finder = find(object > 200)
loc = round( (max(finder)+ min(finder))/2)
```

```
% returns the x and y coordinates of the object detected
location = combinedata2(:,loc)
```

- This last chunk of code then returns the final result: the x and y coordinates of the object. See the result below for the demo objects I used in this code section.

```
location = 2×1
   298.0474
    23.8483
```

- Lastly, we clean and shut down.

Clean and Shutdown

End program by safely disconnecting from LIDAR

```
fprintf(lidar, 'QT'); % commands the lidar to quit through the string quit
fclose(lidar); % disconnects lidar object from serial port object
clear lidar; % clears lidar obj from memory
warning('on'); % turns warning back on
disp("program ended");
```