Object Detection using SICK S-300 Laser Rangefinder simulated in CoppeliaSim (V-REP) & MATLAB

Written by ***Nandhito Adiyatma Rahadi*** *(18/428626/TK/47128)*

I. INTRODUCTION

In this assignment, students are asked to simulate object detection using laser rangefinder. Coppelia Robotics Simulator (or commonly known as V-REP) is used in this simulation, due to its direct compatibility with MATLAB as an independent data processing environment. In instance, CoppeliaSim will be used to design the simulation environment, featuring the SICK S-300 laser rangefinder and some primitive shaped objects. The sensor measurement will be transferred to the MATLAB Workspace to be able to process the data further.

II. SIMULATION METHODS

CoppeliaSim Environment – Cuboid (Non-Threaded) Child Script

```
function sysCall_init()
    simExtRemoteApiStart(19999)
end

function sysCall_actuation()
    -- put your actuation code here
end

function sysCall_sensing()
    -- put your sensing code here
end

function sysCall_cleanup()
    -- do some clean-up here
end

-- See the user manual or the available code snippets for additional callback
functions and details
```

To able to connect the CoppeliaSim Environment with the MATLAB Workspace, a connection between ports were needed to be established in the first place. From the CoppeliaSim Environment side, user need to establish the connection using **`simExtRemoteApiStart()`** command, while declaring the used port number between parentheses (**19999** is the default port number). This can be done in any objects within the environment (The parent objects).

CoppeliaSim Environment – SICK S300 Fast Child Script

```
function sysCall_init()
    visionSensor1Handle=sim.getObjectHandle("SICK_S300_sensor1")
    visionSensor2Handle=sim.getObjectHandle("SICK_S300_sensor2")
    joint1Handle=sim.getObjectHandle("SICK_S300_joint1")
    joint2Handle=sim.getObjectHandle("SICK_S300_joint2")
    sensorRefHandle=sim.getObjectHandle("SICK_S300_ref")

    maxScanDistance=5

sim.setObjectFloatParam(visionSensor1Handle,sim.visionfloatparam_far_clipping,maxSc
anDistance)

sim.setObjectFloatParam(visionSensor2Handle,sim.visionfloatparam_far_clipping,maxSc
anDistance)
    maxScanDistance_=maxScanDistance*0.9999

    scanningAngle=180*math.pi/180

sim.setObjectFloatParam(visionSensor1Handle,sim.visionfloatparam_perspective_angle,
scanningAngle/2)

sim.setObjectFloatParam(visionSensor2Handle,sim.visionfloatparam_perspective_angle,
scanningAngle/2)

    sim.setJointPosition(joint1Handle,-scanningAngle/4)
    sim.setJointPosition(joint2Handle,scanningAngle/4)
    red={1,0,0}
    lines=sim.addDrawingObject(sim.drawing_lines,1,0,-1,1000,nil,nil,nil,red)

    showLines=true
end

function sysCall_cleanup()
    sim.removeDrawingObject(lines)
end

function sysCall_sensing()
    measuredData={}
```

```
if notFirstHere then
    -- We skip the very first reading
    sim.addDrawingObjectItem(lines,nil)
    r,t1,u1=sim.readVisionSensor(visionSensor1Handle)
    r,t2,u2=sim.readVisionSensor(visionSensor2Handle)

    m1=sim.getObjectMatrix(visionSensor1Handle,-1)
    m01=sim.getObjectMatrix(sensorRefHandle,-1)
    sim.invertMatrix(m01)
    m01=sim.multiplyMatrices(m01,m1)
    m2=sim.getObjectMatrix(visionSensor2Handle,-1)
    m02=sim.getObjectMatrix(sensorRefHandle,-1)
    sim.invertMatrix(m02)
    m02=sim.multiplyMatrices(m02,m2)
    if u1 then
        p={0,0,0}
        p=sim.multiplyVector(m1,p)
        t={p[1],p[2],p[3],0,0,0}
        for j=0,u1[2]-1,1 do
            for i=0,u1[1]-1,1 do
                w=2+4*(j*u1[1]+i)
                v1=u1[w+1]
                v2=u1[w+2]
                v3=u1[w+3]
                v4=u1[w+4]
                if (v4<maxScanDistance_) then
                    p={v1,v2,v3}
                    p=sim.multiplyVector(m01,p)
                    table.insert(measuredData,p[1])
                    table.insert(measuredData,p[2])
                    table.insert(measuredData,p[3])
                end
                if showLines then
                    p={v1,v2,v3}
                    p=sim.multiplyVector(m1,p)
                    t[4]=p[1]
                    t[5]=p[2]
                    t[6]=p[3]
                    sim.addDrawingObjectItem(lines,t)
                end
            end
        end
    end
    if u2 then
        p={0,0,0}
        p=sim.multiplyVector(m2,p)
        t={p[1],p[2],p[3],0,0,0}
        for j=0,u2[2]-1,1 do
            for i=0,u2[1]-1,1 do
                w=2+4*(j*u2[1]+i)
                v1=u2[w+1]
                v2=u2[w+2]
                v3=u2[w+3]
```

```
                    v4=u2[w+4]
                    if (v4<maxScanDistance_) then
                        p={v1,v2,v3}
                        p=sim.multiplyVector(m02,p)
                        table.insert(measuredData,p[1])
                        table.insert(measuredData,p[2])
                        table.insert(measuredData,p[3])
                    end
                    if showLines then
                        p={v1,v2,v3}
                        p=sim.multiplyVector(m2,p)
                        t[4]=p[1]
                        t[5]=p[2]
                        t[6]=p[3]
                        sim.addDrawingObjectItem(lines,t)
                    end
                end
            end
        end
    end
    notFirstHere=true
end
```

The Script written above is provided as default by the Coppelia Robotics Simulator, users are recommended to only change the angle of observation and its range, which in this case is changed into 180 degrees from the default configuration of 270 degrees. The sensor itself is built with 2 Orthographic Type-Vision Sensor. Both sensors child script is kept default.

MATLAB Workspace – Initiation and Data Collecting (SICKread.m)

```
sim=remApi('remoteApi');
sim.simxFinish(-1);

clientID=sim.simxStart('127.0.0.1',19999,true,true,5000,5);

if (clientID>-1)
    disp('Connected to remote API server');

    % Object Handle
    [returnCode,sensor1] =
sim.simxGetObjectHandle(clientID,'SICK_S300_sensor1',sim.simx_opmode_blocki
ng);
```

```
    [returnCode,sensor2] =
sim.simxGetObjectHandle(clientID,'SICK_S300_sensor2',sim.simx_opmode_blocki
ng);

    % Object Command
    [returnCode,resolution,image1] =
sim.simxGetVisionSensorImage2(clientID,sensor1,0,sim.simx_opmode_streaming)
;
    [returnCode,resolution,image2] =
sim.simxGetVisionSensorImage2(clientID,sensor2,0,sim.simx_opmode_streaming)
;

    for i=1:10
        [returnCode,resolution,image1] =
sim.simxGetVisionSensorImage2(clientID,sensor1,0,sim.simx_opmode_buffer);
        [returnCode,resolution,image2] =
sim.simxGetVisionSensorImage2(clientID,sensor2,0,sim.simx_opmode_buffer);
        figure(1)
        imshow(image1)
        figure(2)
        imshow(image2)
        pause(0.1);
    end

    sim.simxFinish(-1);
end

sim.delete();
disp('Program ended');
```

In the MATLAB Workspace side, user could then establish the connection with CoppeliaSim Environment using **`simxStart()`** command, while declaring the used port and connection within the parentheses. To use objects from the CoppeliaSIm Environment, user need to declare it in the first place with **`simxGetObjectHandle()`** command. Since the used objects are vision sensors, so the simulation data will be transferred as images using **`simxGetVisionSensorImage2()`** to the MATLAB Workspace.

MATLAB Workspace – Image Processing & Performs Hough Transform (ImageProcess.m)

```
% Displaying Combined Image (from 2 seperated vision sensor in SICK S-300).
image = imfuse(image2,image1,'montage');
figure(1)
imshow(image)

% Turning Image into Numerical Vector and plot it.
image_row_vector = image(:,:,2);
figure(2)
plot(image_row_vector)

% Performing Hough Transform.
[H,theta,rho] = hough(image_row_vector);

figure(3)
imshow(imadjust(rescale(H)),[],...
       'XData',theta,...
       'YData',rho,...
       'InitialMagnification','fit');
xlabel('\theta (degrees)')
ylabel('\rho')
axis on
axis normal
hold on
colormap(gca,hot)

P = houghpeaks(H,5,'threshold',ceil(0.3*max(H(:))));
x = theta(P(:,2));
y = rho(P(:,1));
plot(x,y,'s','color','black');
```

After getting the image data from the vision sensors, user could then merge the images into one image to represents the overall sensor vision using **`imfuse()`** command. The image will be represented as 512 columns and 3 rows uint8 binary image. Since both 3 rows have the same value (numerically), so user could just pick one row as a new vector to redeclare it as a 2D matrix which can be plotted as overall reading values of the sensors. From the same image, user could also perform the Hough Transform to see other characteristics of object(s) detected by the sensor.

III. SIMULATION RESULTS

CASE 1 – DETECTING A CUBOID OBJECT

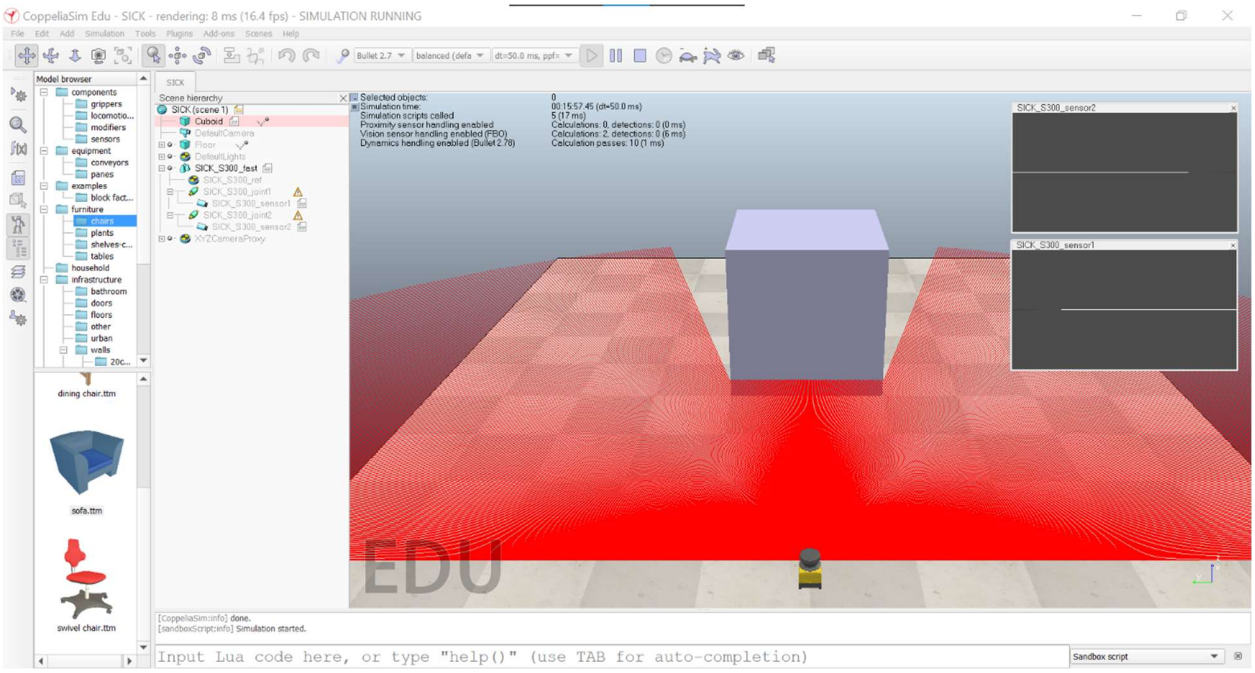

Figure 1 - First Case CoppeliaSim Environment

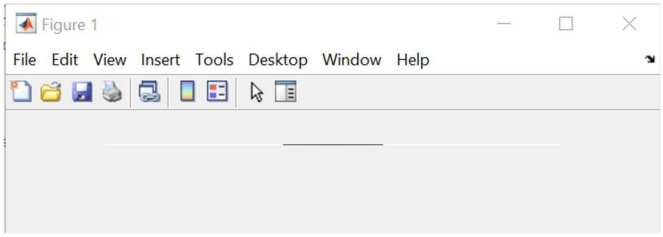

Figure 2 - First Case Recorded Image

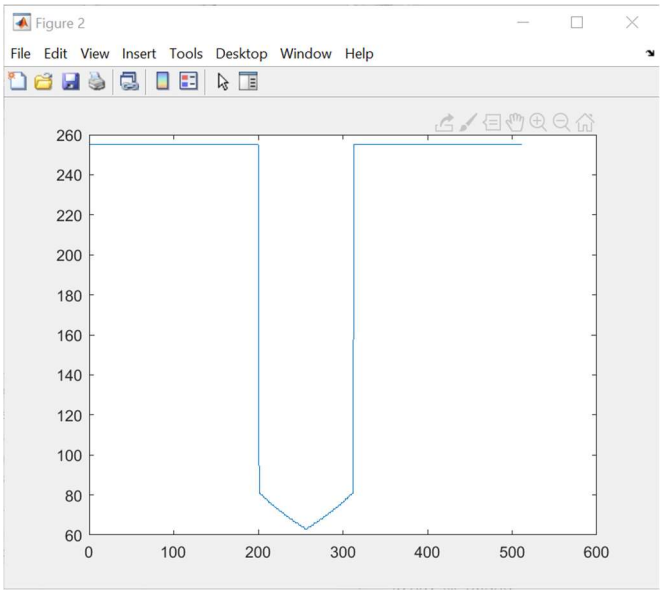

Figure 3 - First Case Image Vector

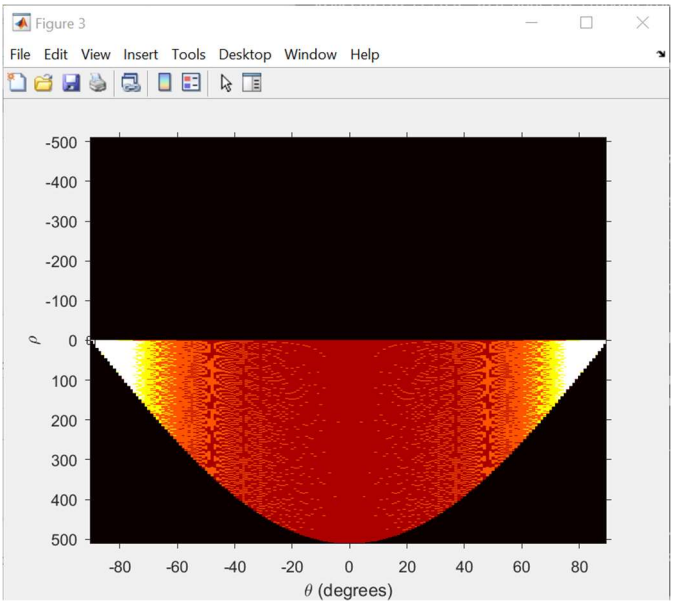

Figure 4 - First Case Hough Transformed Plot

CASE 2 – DETECTING A CYLINDRICAL OBJECT

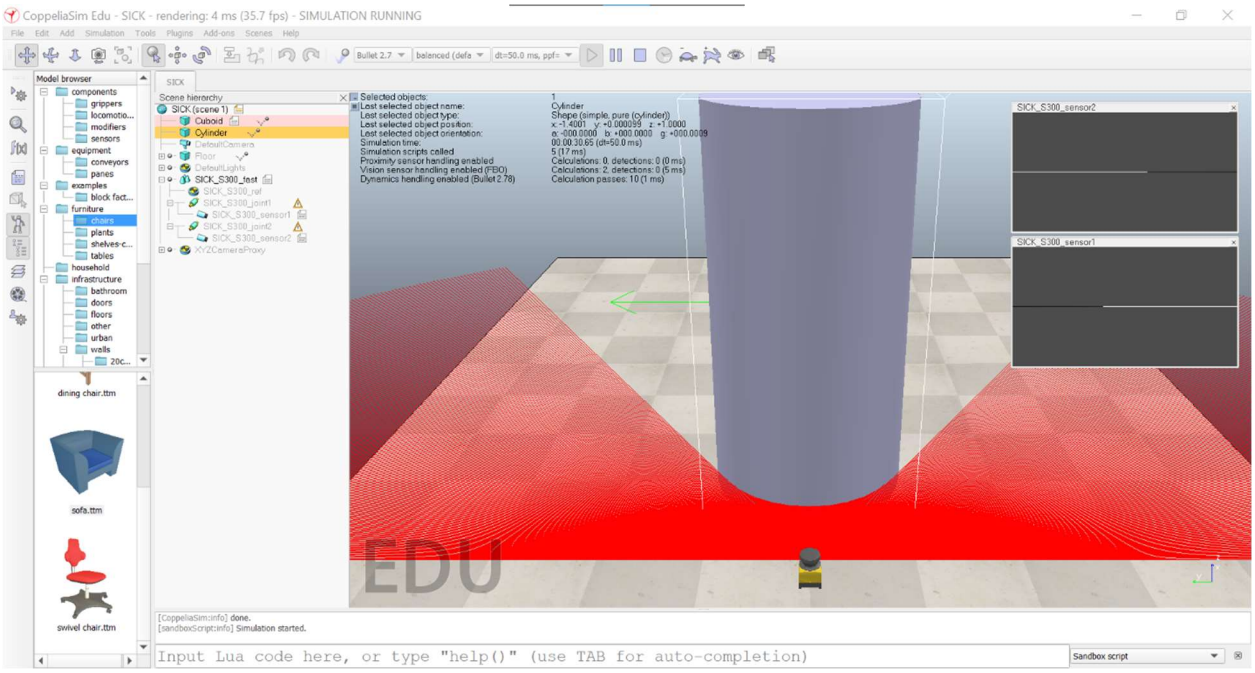

Figure 5 - Second Case CoppeliaSim Environment

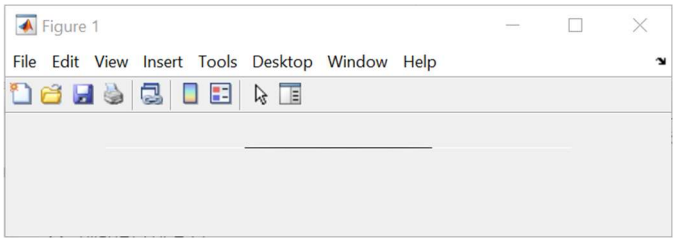

Figure 6 - Second Case Recorded Image

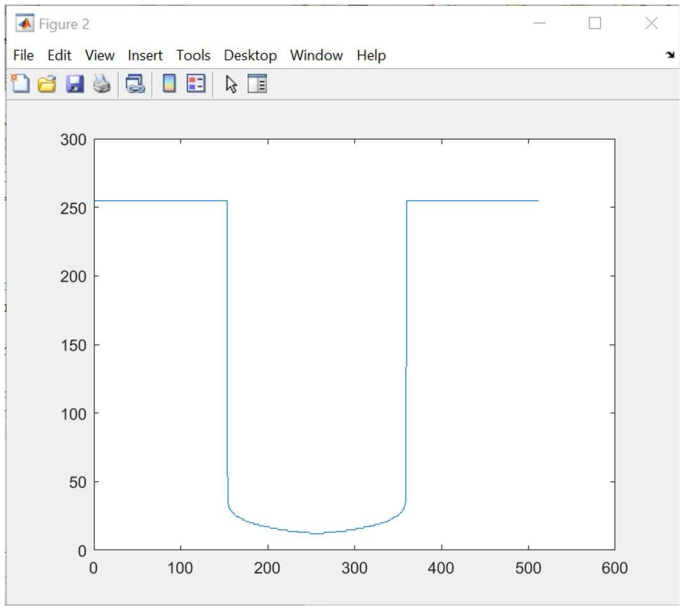

Figure 7 - Second Case Image Vector

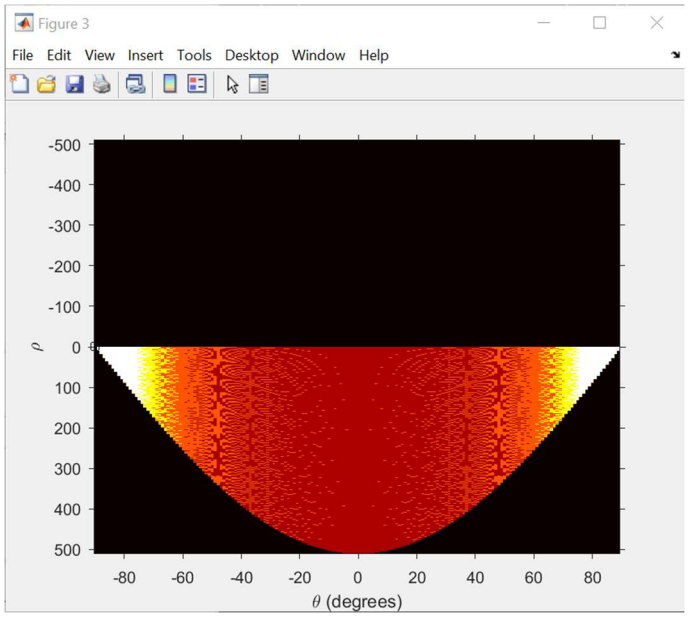

Figure 8 - Second Case Hough Transformed Plot

CASE 3 – DETECTING MULTIPLE OBJECTS

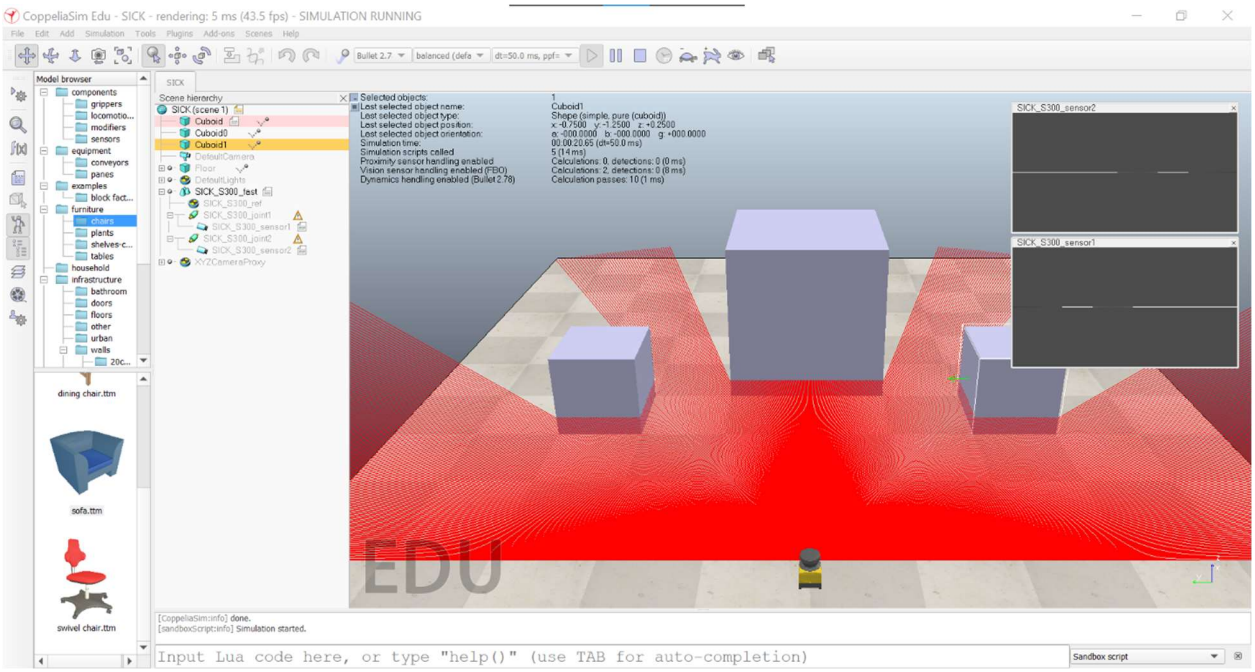

Figure 9 - Third Case CoppeliaSim Environment

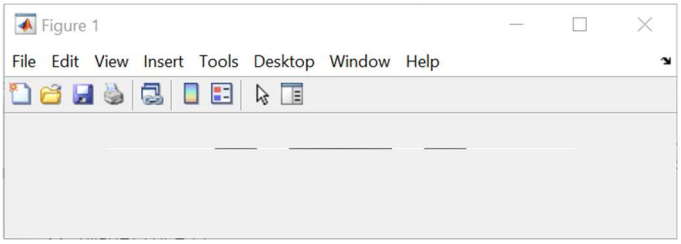

Figure 10 - Third Case Recorded Image

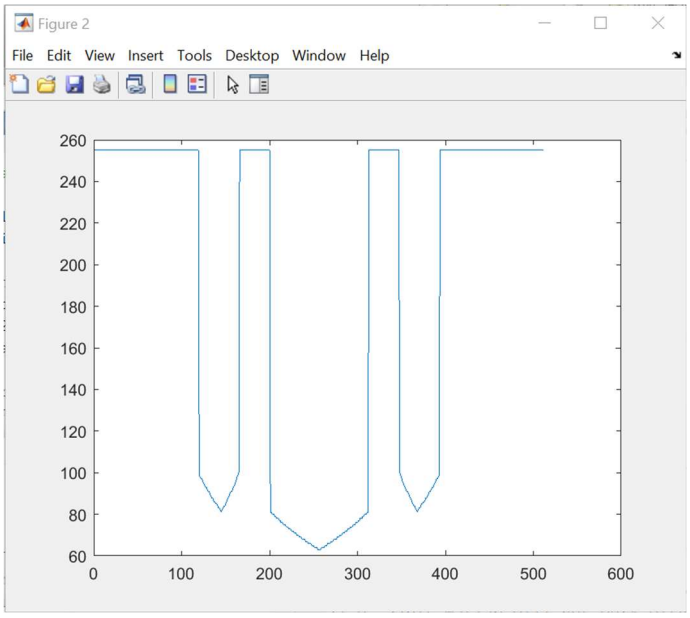

Figure 11 - Third Case Image Vector

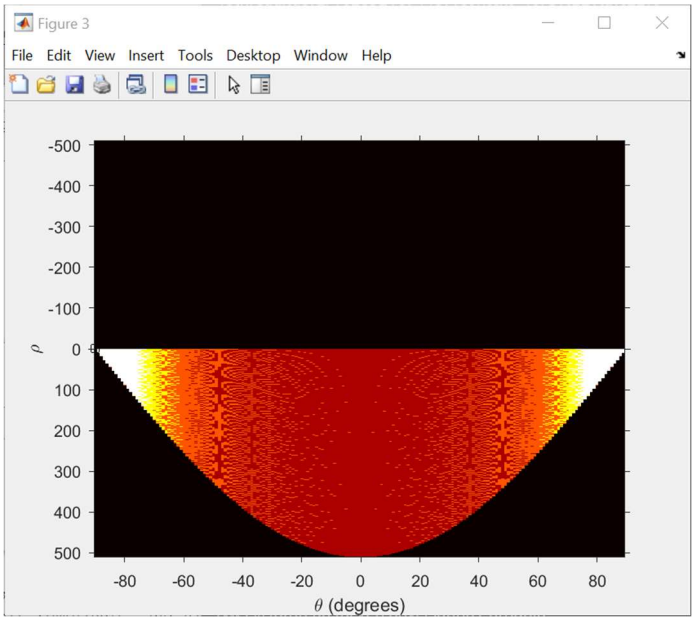

Figure 12 - Third Case Hough Transformed Plot

In the first and third case, we use cuboids as the object of detection, from Figure 3 and Figure 11, we could see that the detection plot seen as a sharp elbow. This is due to the relative position between the object surface to the sensor position, which the sensor does the scanning in 180 degrees radial angle. In instance, the sharp elbow represents a flat surface, which likely belongs to a cuboid type objects. The only

difference is that in the third case, there are 3 spikes, representing 3 different objects. But since the plot shows similar sharp elbows, we could conclude that all of them are cuboid type objects.

On the other hand, in the second case, which used a cylinder as the object of detection, we could see that in Figure 7, the shape is pretty much rounded, represents the rounded nature of the cylinder surface itself. Another thing that we need to look at is that in all three cases, the Hough Transform plot shows the same image, represents the 180 degrees of observation, but that's all. The problem with this Hough Transform is the image used. Hough Transform use lines to be able to classify the shape of the object, while the Orthographic Type-Vision Sensor only show a line of image, this image cannot be transformed further by the Hough Algorithm, since there are not enough features that could be extracted from just a line. That explains why in all three cases, the Hough Transform plots show all the same results.

IV. CONCLUSIONS

In conclusions, by using vision sensor based-laser rangefinder, we could predict the shape and how much object is detected by simply analyzing the distance readings of the sensor position, relative to the detected object surface. While Hough Transform is proven to be not effective to be used in an Orthographic Type-Vision Sensor, because that kind of sensor only produce a line that contains insufficient features to be extracted from. On the other hand, Hough Transform would be effective if used along with a Perspective Type-Vision Sensor or simply a camera that depict the complete image in front of the sensor, and thus is able to extract sufficient features to then predict the shape of object detected by the sensor.