Derek Sayler
Calvin Kwong
EE146 Winter 2019
Bir Bhanu

EE146 Project: Obstacle Detection and Depth Estimation

Introduction

Our project is obstacle detection and distance estimation for robotic applications. Instead of using machine learning to detect objects in a video stream we wanted to combine RGB and depth image data to detect objects on the fly. We believe that machine learning algorithms used for object detection takes too long to train and only detects objects you train it to detect. We wanted to make an algorithm that can detect any objects so a rover can safely path around it. Our project involves the Intel Realsense to retrieve RGB and depth image stream which we use to detect objects.

Technical Approach

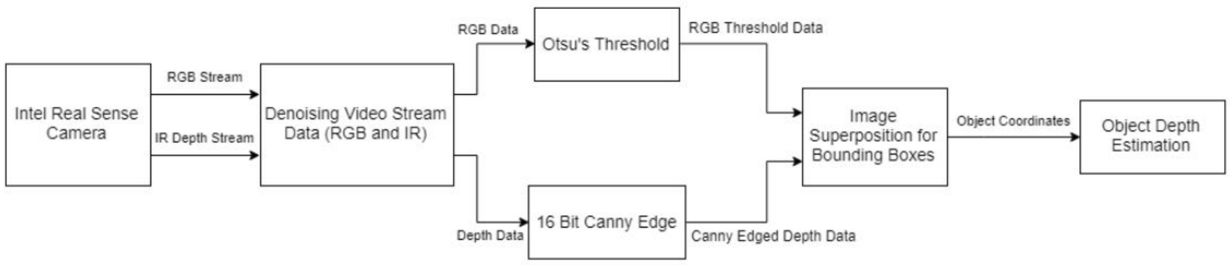

Figure 1: Algorithm Flowchart

Our approach is quite simple. First we read in the RGB and depth image data from the RealSense and begin removing noise from the data. For the depth image we align the images so pixels match across images. The we fill in any holes in so we get don't have area in the image without depth information. Next we delete the background by setting any depth element that is beyond 2 meters depth to 2 meters. Next we do histogram equalization to make any objects in the image stand out more for later Canny edge detection. Following this we do an open and close morphology to remove any leftover noise and make any object have smoother edges. For the RGB image we simply take the corresponding pixels from the depth image that were over 2 meters away and set them to grey in the RGB image. Following this allows for a clear foreground in both images and removes interference from objects in the background. The figure below shows an example of this process.

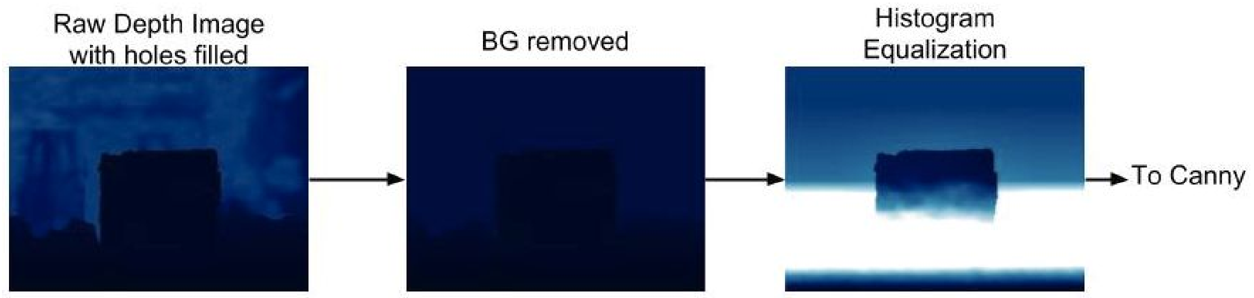

Figure 2: Noise Removal, BG removal, and histogram equalization of Depth

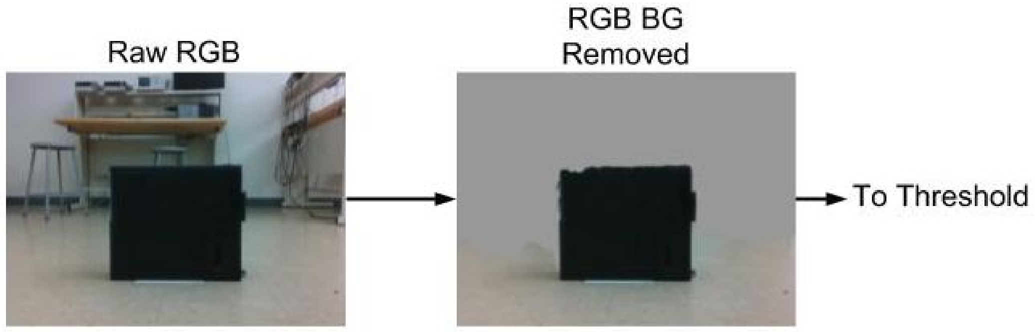

Figure 3: BG removal of RGB image

After removing noise and background from both RGB and depth images, we take the depth image and perform Canny edge detection on it. This should produce object “peaks” that we can perform contouring and extract bounding boxes from. Alongside this we also perform grayscale transformation, Otsu’s threshold algorithm, and open/close morphology on the RGB images. This should produce a binary image with minimal noise from ground interference and give low intensity objects a clear foreground area.

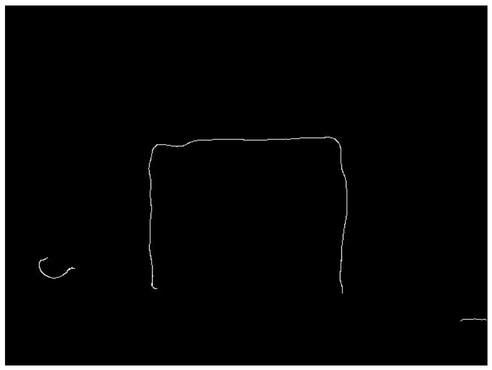

Figure 4: Canny edge of depth image

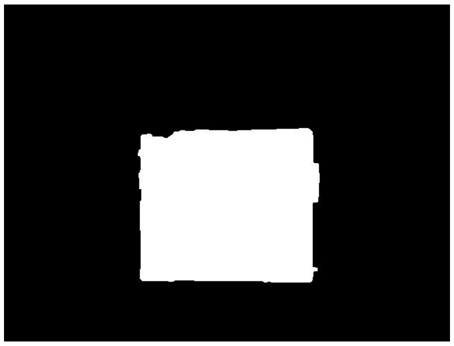

Figure 5: Ostu's Threshold and morphology clean of RGB image

Next we contour both the thresholded RGB and edge depth images and get bounding boxes in each image of detected objects. We then use intersection over union to find the percent of overlap between bounding boxes from both images. Any bounding boxes that have an overlap of over 70% we save. Then we take all saved bounding boxes and find the biggest one between the saved boxes and set that as our predicted object. The algorithm only detects one object at a time.

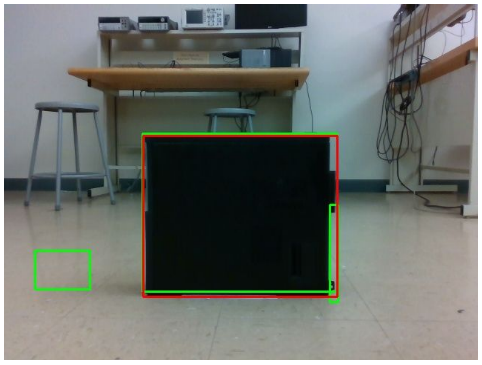

Figure 6: Bounding Boxes before decision box

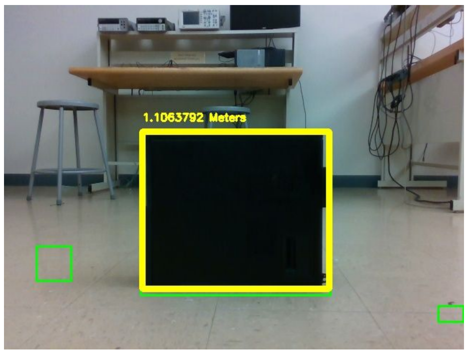

Figure 7: Decision Box and Distance Estimation

Results

We took 4 videos as test data. Two with a single object, one with two objects, and one without an object to detect. Our algorithm detects only one object at a time so these test vids should only detect one object and do it robust enough to be detected a majority of the time.

Also for these results we had to have the camera about 2 feet off the ground to reduce the number of false edges creates during the canny edge detection portion of our code. First we needed to find an optimal threshold for the canny edge detection. The following figure shows the threshold boundaries tested for the canny edge detection and their respective accuracies.

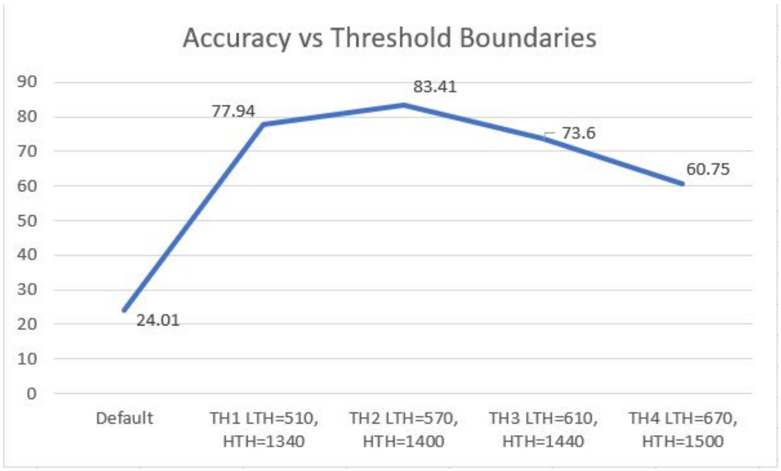

Figure 8: Accuracy vs Threshold

Next we wanted to create a confusion matrix to determine the performance of our algorithm. The figure below shows the confusion matrix base on our test videos. As you can see the performance of our algorithm isn't too bad but it has very specific conditions in order for it to work well.

Confusion Matrix

(image frames) N = 1020	Pred: Yes	Pred: No	
Actual: Yes	TP= 529	FP=145	674
Actual: No	FN= 133	TN=213	346
	662	358	

Figure 9: Performance Matrix

Conclusion

This was a really tough project. There is very little references that use simple algorithms to detect objects using RGBD image data with the RealSense. We instead had to take parts from many different pieces of literature and implement them into our project. Although our implementation works well when there is an object on screen, it does not work well with many objects, and when there is no pronounced objects on screen. To fix this ground deletion is necessary for isolating objects in depth image although we did not have success implementing it in a real time environment.

References

B. Bhanu, S. Lee, C. C. Ho and T. Henderson, "Range data processing: Representation of surfaces by edges," *Proceedings 8th International Conference on Pattern Recognition,* 1986

W. Liu, X. Chen, Q. Wu, J. Yang, “Fast robust detection of edges in noisy depth images”, Journal of Electronic Imaging, 2016

Hsieh-Chang Huang , Ching-Tang Hsieh , and Cheng-Hsiang Yeh, “An Indoor Obstacle Detection System Using Depth Information and Region Growth”, NIH, 2015

X. Rivera, R. Cadubla, J. Alemania, R. Valdellon, R. Villanueva, R. Vicerra, E. Roxas , A. Cruz, “Depth Map Characterization of RGB-D Sensor for Obstacle Detection System”, IEEE, 2015

S. Vorapatratorn, A. Suchato, P. Punyabukkana, “Real-time Obstacle Detection in Outdoor Environment for Visually Impaired using RGB-D and Disparity Map”, ACM, 2016

Code

```
import pyrealsense2 as rs
import numpy as np
import cv2 as cv
from skimage import feature
from skimage import img_as_ubyte
from foos import intersection_over_union, area_of_box

f = 1.93  # focal length of RealSense D435
confidence = 0.70

# Configure depth and color streams
pipeline = rs.pipeline()
config = rs.config()

config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)

# Fill holes in depth
holeFilter = rs.hole_filling_filter()
holeFilter.set_option(rs.option.holes_fill, 1)

# Start streaming
profile = pipeline.start(config)

# Getting the depth sensor's depth scale (see rs-align example for explanation)
depth_sensor = profile.get_device().first_depth_sensor()
depth_scale = depth_sensor.get_depth_scale()

# We will be removing the background of objects more than
#  clipping_distance_in_meters meters away
clipping_distance_in_meters = 2  # 2 meters
clipping_distance = clipping_distance_in_meters / depth_scale
print(int(clipping_distance+1))

# Define the codec and create VideoWriter object.The output is stored in 'outpy.avi' file.

# Create an align object
# rs.align allows us to perform alignment of depth frames to others frames
# The "align_to" is the stream type to which we plan to align depth frames.
align_to = rs.stream.color
align = rs.align(align_to)

def nothing(x):
   pass

cv.namedWindow("Options")
cv.createTrackbar('Lower Thresh', 'Options', 570, 2000, nothing)
cv.createTrackbar('Upper Thresh', 'Options', 1400, 2000, nothing)
cv.createTrackbar('Sigma', 'Options', 7, 15, nothing)

try:
   while True:

       #Make sliders
       lowerCannyThresh = cv.getTrackbarPos('Lower Thresh', 'Options')
       upperCannyThresh = cv.getTrackbarPos('Upper Thresh', 'Options')
       sigma = cv.getTrackbarPos('Sigma', 'Options')

       # Wait for a coherent pair of frames: depth and color
       frames = pipeline.wait_for_frames()

       # Align the depth frame to color frame
       aligned_frames = align.process(frames)

       # Get aligned frames
       depth_frame = aligned_frames.get_depth_frame()  # aligned_depth_frame is a 640x480
depth image
       color_frame = aligned_frames.get_color_frame()

       # Validate that both frames are valid
       if not depth_frame or not color_frame:
           continue

       # Let RealSense fill holes in depth map
       depth_frame = holeFilter.process(depth_frame)

       # Convert images to numpy arrays
       depth_image = np.asanyarray(depth_frame.get_data())
       color_image = np.asanyarray(color_frame.get_data())
```

```
    # Background elimination
    # Remove background of color image - Set pixels further than clipping_distance to grey
    grey_color = 153
    depth_image_3d = np.dstack(
        (depth_image, depth_image, depth_image))  # depth image is 1 channel, color is 3
channels
    color_bg_removed = np.where((depth_image_3d > clipping_distance) | (depth_image_3d
<= 0), grey_color, color_image)

    # Eliminate bg of depth image over clipping_distance away
    depth_bg_removed = np.where((depth_image > int(clipping_distance)) | (depth_image <=
0), int(clipping_distance), depth_image)

    ################ DEPTH OBJ DETECTION #########################
    # Histogram Equilization
    clahe = cv.createCLAHE(clipLimit=50.00, tileGridSize=(3, 3))
    depth_eq = clahe.apply(depth_bg_removed)

    # Open and Closing Depth Image
    strel = cv.getStructuringElement(cv.MORPH_RECT, (5, 5))
    depth_clean = cv.morphologyEx(depth_eq, cv.MORPH_OPEN, strel, 3)
    depth_clean = cv.morphologyEx(depth_clean, cv.MORPH_CLOSE, strel, 3)

    # Edge detection
    depth_edge = feature.canny(depth_clean, sigma=sigma,
low_threshold=lowerCannyThresh, high_threshold=upperCannyThresh)
    depth_edge = img_as_ubyte(depth_edge)

    ################ COLOR OBJ DETECTION #########################
    # convert bg removed color image to greyscale
    color_bg_gray = cv.cvtColor(color_bg_removed, cv.COLOR_BGR2GRAY)

    # blur
    color_bg_gray = cv.blur(color_bg_gray, (3, 3))

    # threshold greyscale image
    ___, th = cv.threshold(color_bg_gray, 127, 255, cv.THRESH_OTSU)

    # Close and open threshold image
    th = cv.morphologyEx(th, cv.MORPH_CLOSE, strel, 3)
    th = cv.morphologyEx(th, cv.MORPH_OPEN, strel, 3)
    # bitwise not because why not
    th = cv.bitwise_not(th)

    ############# BOUNDING BOXES ################
    color_result = np.copy(color_image)
    # Get contours of depth image
    contours, hierarchy = cv.findContours(depth_edge, cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)

    # Approximate Contours to Polygons + Get Bounding Boxes of depth image
    edgeRect = [None] * len(contours)
    contours_poly = [None] * len(contours)
    iou = []
    for i, c in enumerate(contours):
        contours_poly[i] = cv.approxPolyDP(c, 3, True)
        edgeRect[i] = cv.boundingRect(contours_poly[i])
    # Draw polygonal contour + bonding rects + circles
    for i in range(len(contours)):
        color = (0, 255, 0)
        cv.rectangle(color_result, (int(edgeRect[i][0]), int(edgeRect[i][1])), (int(edgeRect[i][0] +
edgeRect[i][2]), int(edgeRect[i][1] + edgeRect[i][3] + 20)), color, 2)

    # Get contours of threshold image
    contours, hierarchy = cv.findContours(th, cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)

    # Approximate Contours to Polygons + Get Bounding Boxes of threshold image
    thRect = [None] * len(contours)
    contours_poly = [None] * len(contours)
    for i, c in enumerate(contours):
        contours_poly[i] = cv.approxPolyDP(c, 3, True)
        thRect[i] = cv.boundingRect(contours_poly[i])
    drawing = np.zeros((depth_edge.shape[0], depth_edge.shape[1], 3), dtype=np.uint8)
    # Draw polygonal contour + bonding rects + circles
    for i in range(len(contours)):
        color = (0, 0, 255)
        cv.rectangle(color_result, (int(thRect[i][0]), int(thRect[i][1])),
                     (int(thRect[i][0] + thRect[i][2]), int(thRect[i][1] + thRect[i][3])),
                     color, 2)

    # Chose boxes from depth edge and thresh images that have an overlap percentage of
```

```
50%
    boxes = []
    for i in range(len(edgeRect)):
        for j in range(len(thRect)):
            iou = (intersection_over_union(
                [edgeRect[i][0], edgeRect[i][1], (edgeRect[i][0] + edgeRect[i][2]),
                 (edgeRect[i][1] + edgeRect[i][3])],
                [thRect[j][0], thRect[j][1], (thRect[j][0] + thRect[j][2]),
                 (thRect[j][1] + thRect[j][3])]))
            if(iou > confidence):
                boxes.append([edgeRect[i], thRect[j]])

    # Take choice boxes and choose the largest one
    choice = []
    dist = 0.0
    choice_dist = 0.0
    for i in range(len(boxes)):
        a1 = area_of_box(boxes[i][0])
        a2 = area_of_box(boxes[i][1])
        if a1 > a2:
            if a1 > 500:
                choice = boxes[i][0]
        else:
            if a2 > 500:
                choice = boxes[i][1]

    # Select Ground Truth
    # r = cv.selectROI('GT', color_image, showCrosshair=True, fromCenter=False)

    # display choice bounding box and estimate depth
    if len(choice) != 0:
        startX = int(choice[0])
        startY = int(choice[1])
        endX = int(choice[0] + choice[2])
        endY = int(choice[1] + choice[3])
        # Display choice bounding box
        color = (0, 255, 255)
        cv.rectangle(color_result, (startX, startY), (endX, endY), color, 7)
        # display coice bounding box and estimate depth

        # Crop depth data
        depth = depth_image[startY:endY, startX:endX].astype(float)
```

```
            # Get distance of object and display
            depth = depth * depth_scale
            # display choice bounding box and estimate depth
            dist, _, _, _ = cv.mean(depth)
            label = "%f2" % dist + " Meters"
            y = startY - 15 if startY - 15 > 15 else startY + 15
            cv.putText(color_result, label, (startX, y), cv.FONT_HERSHEY_SIMPLEX, 0.5, color,
2)

        cv.imshow("Depth", depth_clean)
        cv.imshow('Edge', depth_edge)
        cv.imshow('Bin', th)
        cv.imshow("RGB", color_result)

        cv.waitKey(1)
finally:
    # Stop streaming
    pipeline.stop()
```