INPLANT TRAINING/INDUSTRIAL REPORT

ON

3D Object Scanner Using Intel Depth Camera D435if

At

Vividobots Private Limited

216ECE4244

Submitted in partial fulfilment of the
Requirements for the award of the degree
Of
Bachelor of Technology
In
Electronics and Communication Engineering
By

R Dinesh kumar (9922005319)

Department of Electronics and Communication Engineering
School of Electronics, Electrical and Biomedical Technology
Kalasalingam Academy of Research and Education
(Deemed to be University)
Krishnankoil – 626126
Tamil Nadu

November 2024

KALASALINGAM ACADEMY OF RESEARCH AND EDUCATION

(DEEMED TO BE UNIVERSITY)

SCHOOL OF ELECTRONICS AND ELECTRICAL AND BIOMEDICAL TEHCNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ı l t

Declaration by Student

I hereby declare that this Internship on 3d Object Scanner Using Intel Depth Camera D435if is done as my genuine work and no part of it has been reproduced from any other works.

Sl No	Register Number	Name of the Student	Student Signature
1	9922005319	R Dinesh kumar	

KALASALINGAM ACADEMY OF RESEARCH AND EDUCATION

(DEEMED TO BE UNIVERSITY)

SCHOOL OF ELECTRONICS AND ELECTRICAL AND BIOMEDICAL TEHCNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

Certified that this IPT/Internship report on **3D Object Scanner Using Intel Depth Camera D435if** in **Vividobots Private Limited** is the work of **R Dinesh kumar** (9922005319)

Who conducted the work under my supervision

Signature
Ms.S.Sheeba Jeya Sophia
Assistant Professor,
Internal Guide
Department of ECE,
Kalasalingam Academy of
Research and Education,
Krishnankoil – 626126

Signature
Dr.S.Diwakaran,
Associate Professor,
IPT Coordinator,
Department of ECE,
Kalasalingam Academy of
Research and Education,
Krishnankoil - 626126

Signature
Dr.A.Muthukumar, M.E., PhD.,
Head of the Department,
Department of ECE,
Kalasalingam Academy of
Research and Education,
Krishnankoil - 626126

IPT/Internship Review held on 07/11/2024

a l t

ACKNOWLEDGEMENT

We are indebted to kalasalingam University Founder and Chairman “Kalvivallal” **Thiru T.Kalasalingam**, “Illayvallal” **Dr.K.Sridharan**, Chancellor, **Dr. Shasi Anand**, Vice President, **Dr.S.Narayanan**, Vice-Chancellor, and **Dr. V.Vasudevan**, Registrar, Kalasalingam Academy of Research and Education for providing facilities on my Internship/In plant training.

In preparing this report, we were in contact with cluster of people, researchers, academicians, and industrialists. They have contributed towards my understanding and thoughts. We wish to express our sincere appreciation to our supervisor, **Ms.S.Sheeba Jeya Sophia**, Assistant Professor, for encouragement, guidance, critics. We are also thankful to **Dr.A.Muthukumar**, HoD-ECE and Project Coordinators **Dr.S.Diwakaran**, Associate Professor for his guidance, advice, and motivation. Without their continued support and interest, this internship report would not have been the same as presented here.

We thank all the teaching and non-teaching faculty of ECE department for their help to complete this internship/IPT work. We are also grateful to all our family members, friends, and all others who have aided at various occasions. Their views and tips are useful indeed. Unfortunately, it is not possible to list all of them in this limited space.

Table of Contents

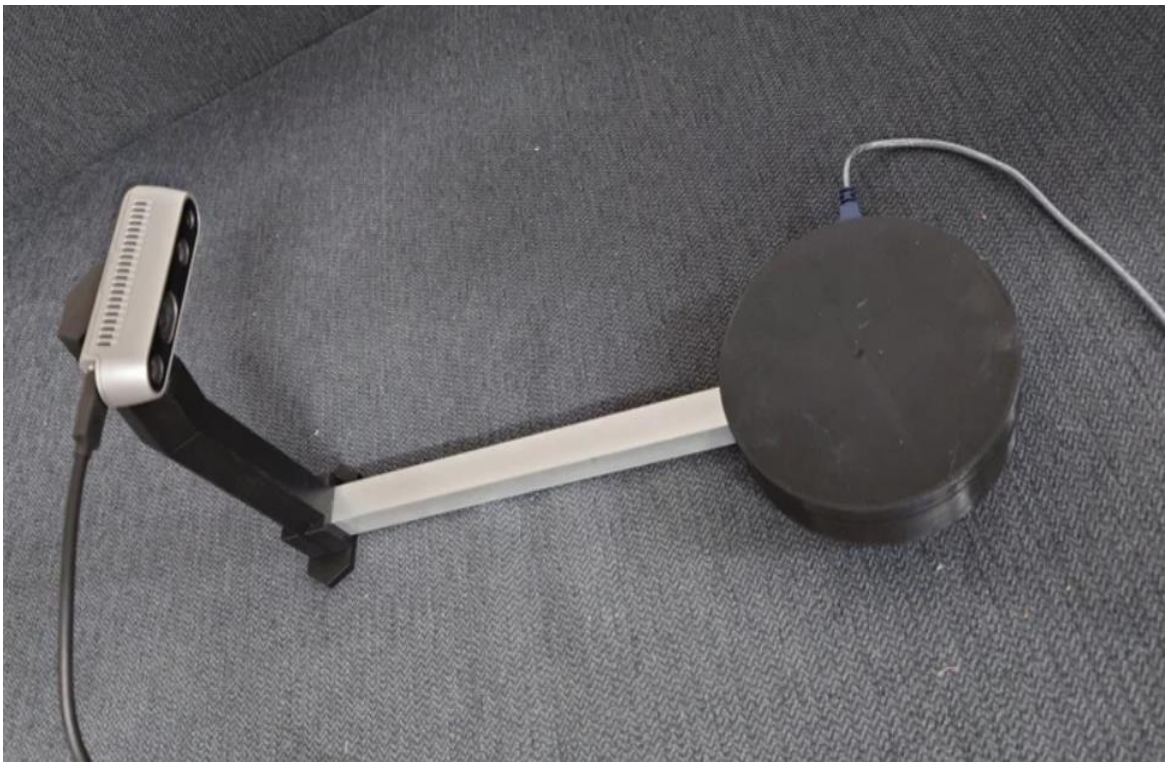
Introduction

- 3D object scanner using an Intel depth camera is a powerful tool for capturing precise and high-resolution 3D models of physical objects. Utilizing Intel's advanced depth-sensing technology

- This scanner measures the distance between the camera and various points on an object's surface, generating a detailed 3D point cloud.

- This data can then be processed to create an accurate, scaled digital replica of the object, ideal for applications in 3D modeling, virtual reality, augmented reality, and manufacturing.

- The Intel depth camera, with its high-resolution sensors and depth accuracy, provides robust scanning capabilities that enhance the speed, accuracy, and quality of 3D scanning, making it a versatile solution for industries such as product design, quality inspection, and digital archiving



Objective

- This project focuses on using an Intel RealSense depth camera to scan physical objects and convert the captured data into 3D mesh files.

- Mesh files are widely used in 3D modeling, printing, and virtual reality. By following this workflow, you can capture real-world objects and create digital 3D models suitable for use in applications that require realistic representations.

Required Components

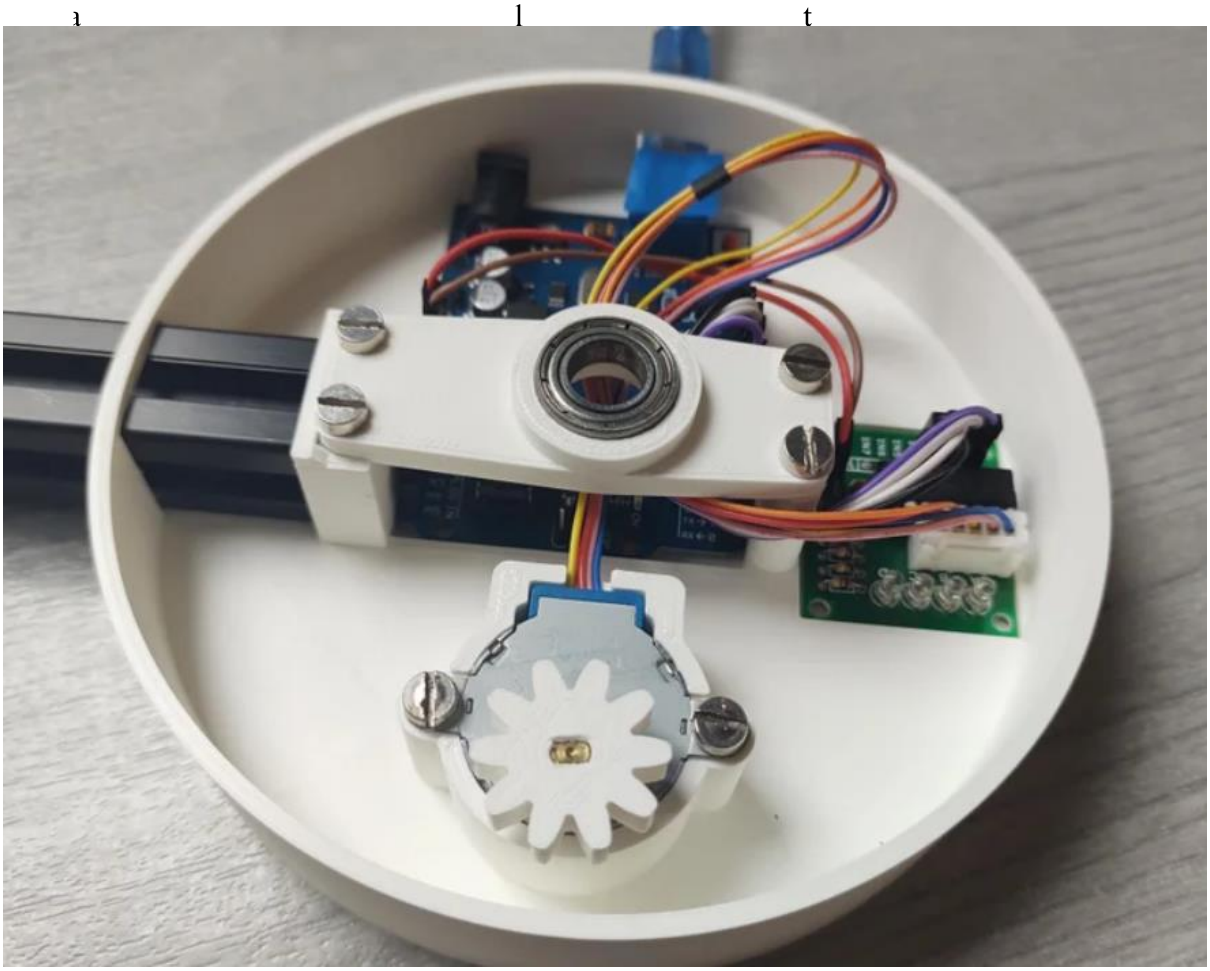
- Intel RealSense Depth Camera (e.g., D415, D435, D455) - This camera is equipped with depth-sensing technology that captures the distance between the camera and objects, providing 3D data for creating point clouds and meshes.In My Case I Used **Intel Depth Camera D435if**

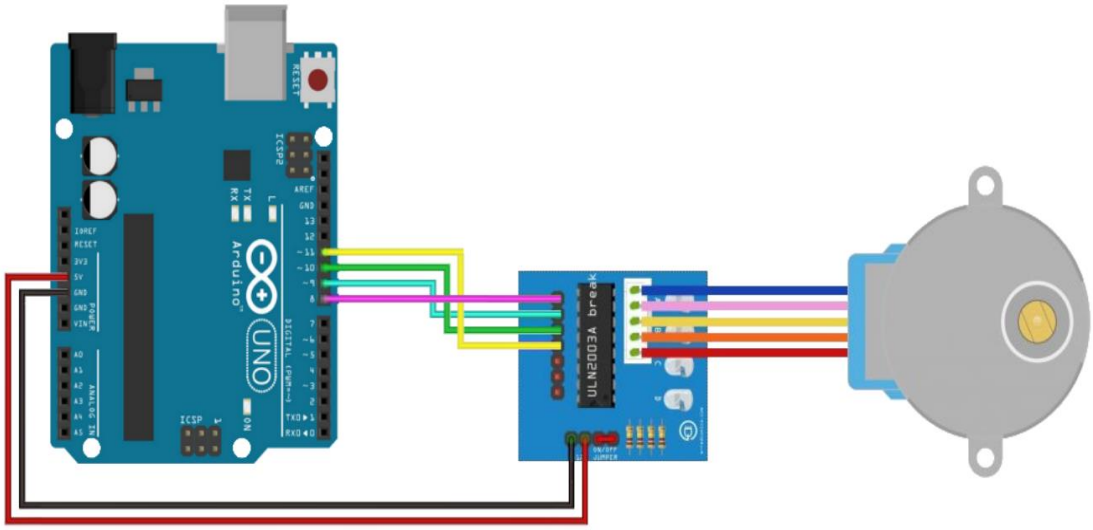
- 3d printer is needed for making turning table And Mount for 3D Object Scanner

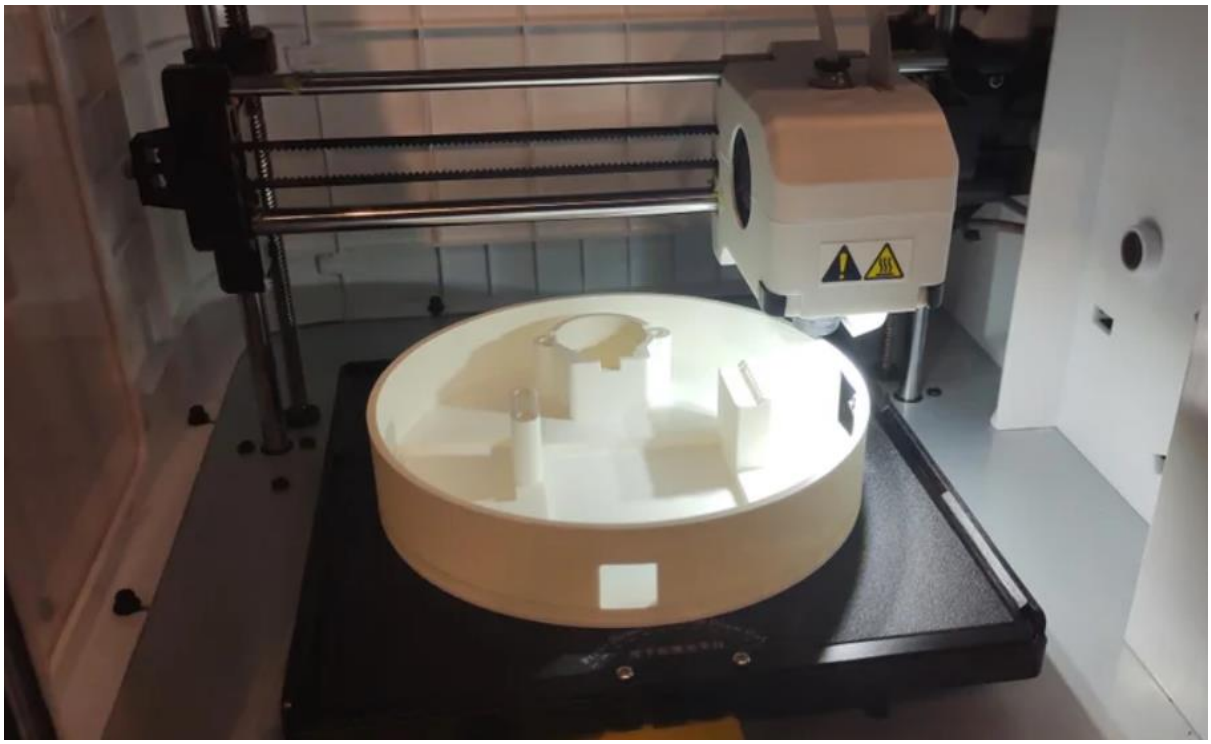
- Electronics Components needed for Making Turning Table For 3D Object Scanner Are

  - **Intel Realsense D435 (with USB-C cable)**
  - **Arduino Mega2560 (with USB-B cable)**
  - **MF-6402411 Stepper motor with ULN2003 Driver**
  - **7 male-female jumper cables**
  - **Skateboard/Fidget spinner ball bearing (8mm (inner diameter) x 22mm (outer diameter) x 7mm (width))**
  - **20x20mm extrusion profile 30cm**
  - **M4 bolts and nuts (screw heads prefered)**
  - **Camera mount bolt**
  - **3D printed PLA parts (minimal 150*150 print bed required)**
  - **Glue**

a l t







Procedure to Make Turning Table And Depth Mount

1. Start by pressing the small gear on the stepper motor.

2. Place the Arduino, Stepper driver and Stepper motor inside the frame. When the Arduino or the Stepper driver don't stay in place by themselves it is possible to glue them in or keep them in place using double-sided tape.

3. Use 2 M4 bolts to lock the Stepper motor in place.

4. Press the ball bearing inside the ball bearing holder.

5. Assemble the ball bearing holder on the frame using 4 M4 bolts. Pay attention to the orientation since the ball bearing isn't symmetric. The middle of the bearing should align with the middle of the frame. (It might also be

handy to put the cables underneath the ball bearing holder to keep them in place)

6. Connect the USB-B cable to the Arduino.

7. Slide the extrusion profile into the frame and secure it using glue.

8. If the camera stand in 2 parts is chosen, connect the 2 parts using 2 M4 bolts and nuts.

9. Put 2 M4 nuts in the nut holder.

10. Slide the nut holder into the bottom groove of the extrusion profile with the nuts facing up.

11. Slide the extrusion profile into the camera stand.

12. Drive 2 bolts in the bottom of the camera stand into the nuts inside the nut holder. Be careful not to tighten the bolts to strong!

13. Secure the camera on the stand using the camera mount bolt.

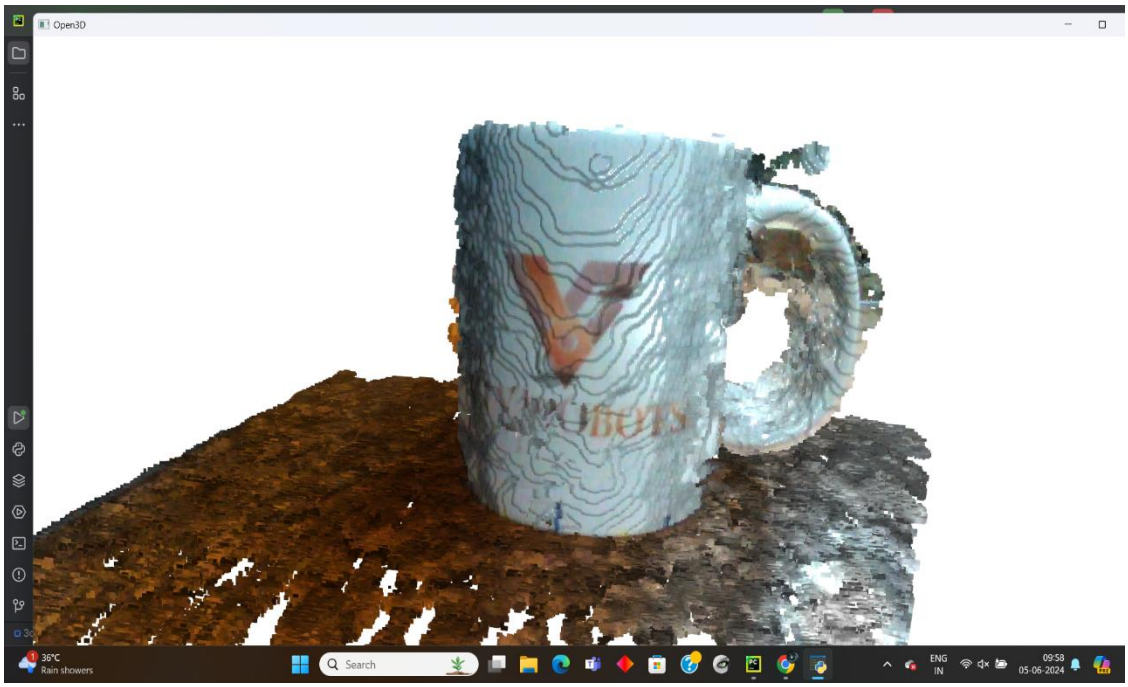
14. Press the turning top inside of the bearing.

15. Connect the USB-C cable to the camera.

Software Libraries:

- **Intel RealSense SDK:** Used for capturing depth frames from the RealSense camera.
- **Open3D:** A powerful library for working with 3D data, including point cloud generation and mesh reconstruction.
- **Point Cloud Library (PCL):** Optional, provides additional tools for point cloud processing , Sample pcl is given below as ply file in the image.

Realtime Image



3D Scanned Single Degree Mesh File

Setup and Configuration

Install Required Software

Intel RealSense SDK: Download and install the SDK from the Intel RealSense GitHub repository.

Open3D: Install Open3D, a library for working with 3D data in Python, by running: `pip install open3d`

Point Cloud Library (PCL):
If additional point cloud processing is needed, install PCL following their official installation guide.

Configure the Intel RealSense Camera After connecting the RealSense camera, test its functionality using sample applications in the Intel RealSense SDK to ensure it captures depth data accurately

Workflow for 3D Object Scanning and Mesh Generation

Capturing Depth Data

To capture depth data from the RealSense camera, use the RealSense SDK to stream the depth frames. This process involves configuring the depth stream and capturing depth images.

Generating a Point Cloud

Using depth data, generate a point cloud - a 3D representation of the object's surface made up of points in space. Point clouds can be visualized directly or used to create a mesh.

Mesh Reconstruction from Point Cloud

After generating the point cloud, convert it into a mesh. This step uses algorithms like Poisson surface reconstruction or Ball-Pivoting to generate a continuous surface from discrete points.

Saving the Mesh File

Once the mesh is created, save it in a standard format like OBJ or PLY for compatibility with other 3D software.

Post-Processing and Mesh Optimization

The generated mesh may need additional processing to remove noise or improve surface smoothness:

Noise Filtering: Remove isolated clusters or outliers in the point cloud. - Smoothing: Apply mesh smoothing algorithms to improve visual quality.

Hole Filling: Some mesh processing tools, like MeshLab or Blender, can fill holes left from incomplete data capture.

Python code for 3D Object Scanning And Mesh Generation

```
import serial   #for using serial function for arduino ide with device's ports i haven't used serial in this code but i used in other code
import keyboard  #for using keyboard function like enter to take photo
import pyrealsense2 as rs #for intel realsense camera user's there is a inbuilt library given by intel for using depth camera functions like rgb, depth module etc
import open3d as o3d #for using 3d point cloud and mesh creation im using open3d library.
import numpy as np #for using basic mathematic functions
import cv2 #for computer vision features used like measuring the camera to object distance and object dimensions.
import time #not used in this code,it is used for time features like date,time etc
import threading #threading is used to thread a function it is used to optimize the code.

class Scan(): #creating a class called called Scan
    def __init__(self, width, height, framerate, autoexposureFrames, backDistance): #im creating a function with self, width, height, framerate, autoexposureFrames, backDistance ,it indicates the self for depth camera.
        self.width = width #width of the depth camera vision
        self.height = height #height of the depth camera vision
        self.framerate = framerate #framerate of the depth camera vision
        self.backDistance = backDistance #it is distance between camera and object.
        self.distance = backDistance / 100 #it is divided by 100 for getting it in the seconds.
        self.autoexposureFrames = autoexposureFrames #This allows the instance to store and manage the autoexposure frames setting.
        self.main_pcd = o3d.geometry.PointCloud() #using open3d library to make point cloud

        self.pipe = rs.pipeline() #realsense depth camera pipeline started.
        self.config = rs.config() #starting realsense with configuration like rgb module, depth module etc..
        self.config.enable_stream(rs.stream.color, self.width, self.height, rs.format.any,
```

self.framerate) #starting to stream color with the width,height,format and framerate given

a l t
self.config.enable_stream(rs.stream.depth, self.width, self.height, rs.format.any, self.framerate) #starting to stream using depth configuration

self.depth_to_disparity = rs.disparity_transform(True) #function that converts depth images to disparity images
self.disparity_to_depth = rs.disparity_transform(False) #disparity to depth is negative
self.dec_filter = rs.decimation_filter() #used to reduce the amount of data in the depth stream, effectively decreasing the resolution of the depth image to improve performance.
self.temp_filter = rs.temporal_filter() #used to reduce temporal noise and smooth the depth data over time.
self.spat_filter = rs.spatial_filter() #used to smooth the depth image spatially, reducing noise and small artifacts while preserving edges
self.hole_filter = rs.hole_filling_filter() #used to fill small gaps (holes) in the depth data where no depth information is available.
self.threshold = rs.threshold_filter(0.17, 0.4) #used to eliminate depth values that fall outside the specified range here in this code i had used 0.17 as minimum and 0.4 as maximum, helping to focus on relevant depth information.

self.dtr = np.pi / 180 #used to convert degrees to radians
self.bbox = o3d.geometry.AxisAlignedBoundingBox((-0.13, -0.13, 0), (0.13, 0.13, 0.2)) #(AABB) using the Open3D library. This bounding box is defined by its minimum (-0.13, -0.13, 0)and maximum (0.13, 0.13, 0.2) corner coordinates.

def startPipeline(self): #startPipeline method im trying to define likely belongs to a class
self.pipe.start(self.config) #pipeline is started with the given configuration
self.align = rs.align(rs.stream.color) #aligning with realsense color stream(camera feed) after this process i will add distance measuring module in this code.
print("Pipeline started") #printing pipeline is started when the rgb came on line

def stopPipeline(self): #i m creating a function called stopPipeline to stop the depth camera
self.pipe.stop() #this line will stop the depth camera streaming.
self.pipe = None #pipeline had stopped
self.config = None #when the pipeline is stopped, the configuration will stop too.
cv2.destroyAllWindows() #by using i m closing the open3d window and camera stream window.
print("Pipeline stopped") #printing the statement that pipeline is stopped.

def takeFoto(self): #i m creating function called takefoto

```
        print("Photo taken!") #printing that photo taken
        for i in range(self.autoexposureFrames): #creating a for loop in the range of the
settings given in the autoexposureFrames
            self.frameset = self.pipe.wait_for_frames() #get a set of frames from the
pipeline

        self.frameset = self.pipe.wait_for_frames() #get a set of frames from the
pipeline
        self.frameset = self.align.process(self.frameset) #ensures that the depth and
color images are spatially aligned
        self.profile = self.frameset.get_profile() #obtain the profile of the frameset which
consist resolution,framerate,backdistance etc..
        self.depth_intrinsics = self.profile.as_video_stream_profile().get_intrinsics()
#obtaining the depth intrinsics(focal length, principal point, and distortion model etc..)
from the stream profile
        self.w, self.h = self.depth_intrinsics.width, self.depth_intrinsics.height #extract
the width and height from the depth intrinsics
        self.fx, self.fy = self.depth_intrinsics.fx, self.depth_intrinsics.fy #transferring the
depth intrinsic from self.depth_intrinsics.fx and self.depth_intrinsics.fy to self.fx and
self.fy, respectively.
        self.px, self.py = self.depth_intrinsics.ppx, self.depth_intrinsics.ppy #self.px,
self.py = self.depth_intrinsics.ppx, self.depth_intrinsics.ppy is assigning the values of
ppx and ppy

        self.color_frame = self.frameset.get_color_frame()
        self.depth_frame = self.frameset.get_depth_frame()

        self.intrinsic = o3d.camera.PinholeCameraIntrinsic(self.w, self.h, self.fx, self.fy,
self.px, self.py)
        self.depth_image = np.asanyarray(self.depth_frame.get_data())
        self.color_image = np.asanyarray(self.color_frame.get_data())

    def processFoto(self, angle):
        print(f"Processing photo at angle {angle} degrees")
        self.angle = angle
        self.depth_frame_open3d = o3d.geometry.Image(self.depth_image)
        self.color_frame_open3d = o3d.geometry.Image(self.color_image)

        self.rgbd_image =
o3d.geometry.RGBDImage.create_from_color_and_depth(self.color_frame_open3d,
                                                   self.depth_frame_open3d,
                                                   convert_rgb_to_intensity=False)
        self.pcd = o3d.geometry.PointCloud.create_from_rgbd_image(self.rgbd_image,
self.intrinsic)

self.pcd.estimate_normals(search_param=o3d.geometry.KDTreeSearchParamHybri
```

```
d(radius=0.1, max_nn=30))


self.pcd.orient_normals_towards_camera_location(camera_location=np.array([0., 0.,
0.]))
        self.getCameraLocation()
        self.rMatrix()
        self.pcd.rotate(self.R, (0, 0, 0))
        self.pcd.translate((self.x, self.y, self.z))
        self.pcd = self.pcd.crop(self.bbox)
        self.pcd, self.ind = self.pcd.remove_statistical_outlier(nb_neighbors=100,
std_ratio=2)
        self.main_pcd = self.main_pcd + self.pcd
        self.measureDimensions()

    def getPointcloud(self):
        return self.main_pcd

    def giveImageArray(self):
        return self.color_image

    def getCameraLocation(self):
        self.x = np.sin(self.angle * self.dtr) * self.distance - np.cos(self.angle * self.dtr) *
0.035
        self.y = -np.cos(self.angle * self.dtr) * self.distance - np.sin(self.angle * self.dtr) *
0.035
        self.z = 0.165
        self.o = self.angle
        self.a = 112.5
        self.t = 0

    def rMatrix(self):
        self.o = self.o * self.dtr
        self.a = (-self.a) * self.dtr
        self.t = self.t * self.dtr
        self.R = [[np.cos(self.o) * np.cos(self.t) - np.cos(self.a) * np.sin(self.o) *
np.sin(self.t),
                   -np.cos(self.o) * np.sin(self.t) - np.cos(self.a) * np.cos(self.t) *
np.sin(self.o),
                   np.sin(self.o) * np.sin(self.a)],
                  [np.cos(self.t) * np.sin(self.o) + np.cos(self.o) * np.cos(self.a) *
np.sin(self.t),
                   np.cos(self.o) * np.cos(self.a) * np.cos(self.t) - np.sin(self.o) *
np.sin(self.t),
                   -np.cos(self.o) * np.sin(self.a)],
                  [np.sin(self.a) * np.sin(self.t), np.cos(self.t) * np.sin(self.a), np.cos(self.a)]]
```

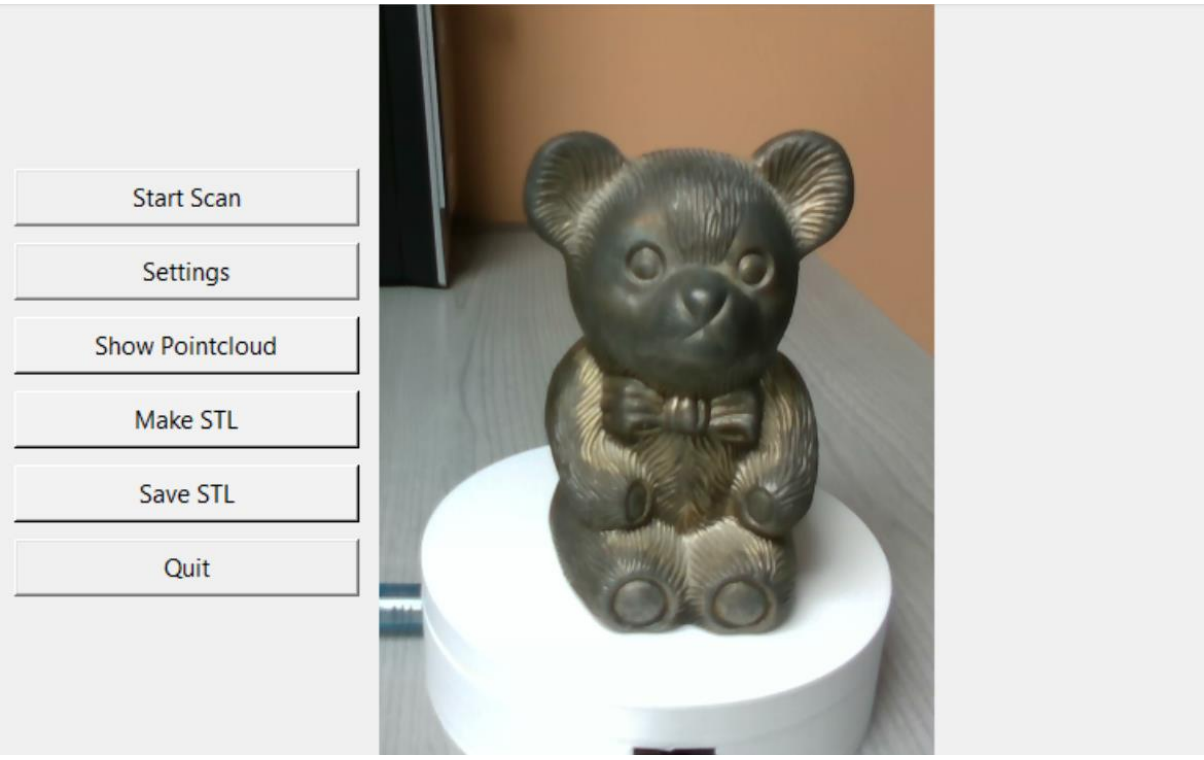
```
    def makeSTL(self, kpoints, stdRatio, depth, iterations):
        self.stl_pcd = self.main_pcd
        self.stl_pcd = self.stl_pcd.uniform_down_sample(every_k_points=kpoints)
        self.stl_pcd, self.ind = self.stl_pcd.remove_statistical_outlier(nb_neighbors=100,
std_ratio=stdRatio)
        self.bbox1 = o3d.geometry.AxisAlignedBoundingBox((-0.13, -0.13, 0), (0.13,
0.13, 0.01))
        self.bottom = self.stl_pcd.crop(self.bbox1)
        try:
            self.hull, self._ = self.bottom.compute_convex_hull()
            self.bottom = self.hull.sample_points_uniformly(number_of_points=10000)

self.bottom.estimate_normals(search_param=o3d.geometry.KDTreeSearchParamHy
brid(radius=0.1, max_nn=30))

self.bottom.orient_normals_towards_camera_location(camera_location=np.array([0.,
0., -10.]))
            self.bottom.paint_uniform_color([0, 0, 0])
            self._, self.pt_map = self.bottom.hidden_point_removal([0, 0, -1], 1)
            self.bottom = self.bottom.select_by_index(self.pt_map)
            self.stl_pcd = self.stl_pcd + self.bottom
        except:
            print("No bottom could be made")
            pass
        finally:
            self.mesh, self.densities =
o3d.geometry.TriangleMesh.create_from_point_cloud_poisson(self.stl_pcd,
depth=depth)
            self.mesh = self.mesh.filter_smooth_simple(number_of_iterations=iterations)
            self.mesh.scale(1000, center=(0, 0, 0))
            self.mesh.compute_vertex_normals()

        return self.mesh

    def measureDimensions(self):
        bbox = self.main_pcd.get_axis_aligned_bounding_box()
        min_bound = bbox.get_min_bound()
        max_bound = bbox.get_max_bound()
        dimensions = max_bound - min_bound
        length, breadth, height = dimensions[0], dimensions[1], dimensions[2]
        print(f"Dimensions of the object - Length: {length:.2f} m, Breadth: {breadth:.2f}
m, Height: {height:.2f} m") #printing the dimensions of the object

    def visualize(self):
        o3d.visualization.draw_geometries([self.main_pcd])
```

```
    def showCameraFeed(self):
        while True:                                frames =
                ı  self.pipe.wait_for_frames()  l                t  color_frame =
frames.get_color_frame()
            depth_frame = frames.get_depth_frame()

            if not color_frame or not depth_frame:
                continue

            color_image = np.asanyarray(color_frame.get_data())
            depth_image = np.asanyarray(depth_frame.get_data())
            center_x, center_y = int(self.width / 2), int(self.height / 2)
            distance = depth_frame.get_distance(center_x, center_y)
            cv2.putText(color_image, f'Distance: {distance:.2f} m', (center_x - 100,
center_y - 20),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2, cv2.LINE_AA)
            cv2.circle(color_image, (center_x, center_y), 5, (0, 255, 0), -1)
            cv2.imshow('Camera Feed', color_image) #a screen is shown as camera
feed with color image,it is shown using cv2 "imshow" keyword
            if cv2.waitKey(1) & 0xFF == ord('q'): #if q key is pressed it will stop camera
feed screen/window
                break #breaking the process
width = 640  #width of the frame is given by me as default
height = 480 #height of the frame is given by me as default
framerate = 30  #framerate is given by me as default
autoexposureFrames = 10  #autoexposureFrames is given by me as a default

backDistance = float(input("Enter the scanning distance (backDistance in cm): "))
#this line get backdistance as input from the user store it as backDistance

scan = Scan(width, height, framerate, autoexposureFrames, backDistance) #Scan
function used with the given configuration and stored in a variable called scan
scan.startPipeline() #starting the depth camera scaning
camera_feed_thread = threading.Thread(target=scan.showCameraFeed) #thread is
created us
camera_feed_thread.start() #threading variable is initializing using start()
angles = [0.0, 7.5, 15.0, 22.5, 30.0, 37.5, 45.0, 52.5, 60.0, 67.5, 75.0, 82.5, 90.0,
97.5, 105.0, 112.5, 120.0, 127.5, 135.0, 142.5, 150.0, 157.5, 165.0, 172.5, 180.0,
187.5, 195.0, 202.5, 210.0, 217.5, 225.0, 232.5, 240.0, 247.5, 255.0, 262.5, 270.0,
277.5, 285.0, 292.5, 300.0, 307.5, 315.0, 322.5, 330.0, 337.5, 345.0, 352.5
] #the angles given in this set will be used for degree turning and photo capture.
for angle in angles: #making a for loop for the angles with name of angle
    input("Press any key to take photo at angle {}: ".format(angle)) #this line will take a
key press as input to take photo in the angle.
    scan.takeFoto() #this will call the takeFoto function it will take angles capture
```

```
    scan.processFoto(angle) #scanning the photo taken with the angle.
    scan.visualize() #visualizing the scanned data
pointcloud = scan.getPointcloud() #this keyword will scan and save the pointcloud in the variable called pointcloud
stl = scan.makeSTL(kpoints=10, stdRatio=0.5, depth=8, iterations=5) #creating stl with this configurations given
o3d.io.write_triangle_mesh("output.stl", stl) #creating stl with the help of open3d with the keyword write triangle mesh

scan.stopPipeline() #depth camera will be stopped
print("3D scanning completed and saved as output.stl") #printing that scanning is complete and saved as output.stl, the file can be located in the location of the python file
```

Implementation

Graphical User Interface of the Python code:



Output of 3D Object Scanner Constructed By 3D Printer

- First Image is the GUI of the python code Created by the Package tkinter.
- Second Image is the Output of 3D Object Scanner Constructed 3D Printer

Applications

1) 3D Printing: Scan objects to create printable 3D models.

2) Virtual Reality: Import meshes into VR environments for realistic simulations.

3) Digital Archives: Preserve real-world artifacts digitally for educational or archival purposes

Conclusion

This 3D scanning workflow enables the transformation of real-world objects into digital mesh files using the Intel RealSense depth camera. The final mesh files can be imported into 3D modeling or CAD software for further refinement, enabling applications in 3D printing, VR/AR, and digital art.