**PV4**

**Requirements.txt**

absl-py==2.1.0

attrs==23.2.0

beautifulsoup4==4.12.2

cachetools==5.3.2

centroid-tracker==0.0.9

certifi==2023.11.17

cffi==1.16.0

charset-normalizer==3.3.2

colorzero==2.0

contourpy==1.1.1

cycler==0.12.1

filelock==3.13.1

flatbuffers==23.5.26

fonttools==4.44.3

gdown==4.7.1

google-auth==2.26.2

google-auth-oauthlib==1.0.0

gpiozero==2.0

grpcio==1.60.0

idna==3.4

importlib-metadata==7.0.1

importlib-resources==6.1.1

imutils==0.5.4

Jetson.GPIO==2.1.6

kiwisolver==1.4.5

Markdown==3.5.2

MarkupSafe==2.1.4

matplotlib==3.7.4

mediapipe==0.10.9

numpy==1.24.4

nvidia-pyindex==1.0.9

oauthlib==3.2.2

onnx==1.15.0

onnxscript==0.1.0.dev20231120

opencv-contrib-python==4.9.0.80

opencv-python==4.8.1.78

packaging==23.2

pandas==2.0.3

Pillow==10.1.0

protobuf==3.20.3

psutil==5.9.6

py-cpuinfo==9.0.0

pyasn1==0.5.1

pyasn1-modules==0.3.0

pycparser==2.21

pyparsing==3.1.1

pyrealsense2==2.54.2.5684

pyserial==3.5

PySocks==1.7.1

python-dateutil==2.8.2

pytz==2023.3.post1

PyYAML==6.0.1

requests==2.31.0

requests-oauthlib==1.3.1

RPi.GPIO==0.7.1

rsa==4.9

scipy==1.10.1

seaborn==0.13.0

six==1.16.0

sounddevice==0.4.6

soupsieve==2.5

tensorboard==2.14.0

tensorboard-data-server==0.7.2

thop==0.1.1.post2209072238

tqdm==4.66.1

typing\_extensions==4.8.0

tzdata==2023.3

urllib3==2.1.0

Werkzeug==3.0.1

zipp==3.17.0

**Code for Enabling Different Cluster every 5 second**

import Jetson.GPIO as GPIO

import time

output\_pins = [18, 19, 21, 22, 23, 24, 26, 31, 11, 7]

C1 = [26,31]

C2 = [18, 19]

C3 = [7,11,23,24,21,22]

C4 = [26, 31,23,24,18,19]

C5 = [7,11]

GPIO.setmode(GPIO.BOARD)

for pin in output\_pins:

GPIO.setup(pin, GPIO.OUT, initial=GPIO.LOW)

try:

for i in range(1):

for pin in output\_pins:

GPIO.output(pin, GPIO.LOW)

GPIO.output(C1, GPIO.HIGH)

time.sleep(5)

GPIO.output(C1, GPIO.LOW)

GPIO.output(C2, GPIO.HIGH)

time.sleep(5)

GPIO.output(C2, GPIO.LOW)

GPIO.output(C3, GPIO.HIGH)

time.sleep(5)

GPIO.output(C3, GPIO.LOW)

GPIO.output(C4, GPIO.HIGH)

time.sleep(5)

GPIO.output(C4, GPIO.LOW)

GPIO.output(C5, GPIO.HIGH)

time.sleep(5)

GPIO.output(C5, GPIO.LOW)

except KeyboardInterrupt:

GPIO.cleanup()

**Final Code:**

import cv2

import numpy as np

import pyrealsense2 as rs

from ultralytics import YOLO

import Jetson.GPIO as GPIO

import serial

import time

iter = 0

# Load the YOLOv5 model

model = YOLO("yolov8n.pt")

# Set the reference distance (in meters)

ref\_distance = 5.0

# Define the output pins for GPIO control

output\_pins = [18,19,21,22,23,24,26,31,11,12]

# Define lists of stationary and semi-stationary objects

Stationary\_Objects = ['dining table','bench','couch','traffic light','tv','laptop','refrigerator','book']

Semi\_Stationary\_Objects = ['person','umbrella', 'bicycle', 'motorcycle','car','bus','dog','cat','cow']

# Initialize a serial connection on port '/dev/ttyTHS1' with baud rate 115200

ser = serial.Serial("/dev/ttyTHS1", 115200)

# Initialize GPIO

GPIO.setmode(GPIO.BOARD)

for pin in output\_pins:

GPIO.setup(pin, GPIO.OUT, initial=GPIO.LOW)

# Set the camera matrix and distortion coefficients

camera\_matrix = np.array([[379.45187378, 0, 324.09848022], [0, 379.45187378, 238.02722168], [0, 0, 1]])

distortion\_coefficients = np.array([0.0, 0.0, 0.0, 0.0, 0.0])

# Set the object size (in meters) at the reference distance

obj\_size = 1.0

# Set the video capture

cap = cv2.VideoCapture(1)

# Start the RealSense pipeline

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)

pipeline.start(config)

# Initialize the depth scale

depth\_sensor = pipeline.get\_active\_profile().get\_device().first\_depth\_sensor()

depth\_scale = depth\_sensor.get\_depth\_scale()

while True:

# Read the frame

ret, frame = cap.read()

(H, W) = frame.shape[:2]

# Perform object detection using YOLO

results = model(frame)

# Draw the bounding boxes and labels

frames = pipeline.wait\_for\_frames()

depth\_frame = frames.get\_depth\_frame()

color\_frame = frames.get\_color\_frame()

if not depth\_frame or not color\_frame:

continue

nearest\_object = None

min\_dist = float('inf')

# Calculating distance from lidar

count = ser.in\_waiting

if count > 8:

recv = ser.read(9)

ser.reset\_input\_buffer()

if recv[0] == 0x59 and recv[1] == 0x59: #python3

distance\_l = recv[2] + recv[3] \* 256

strength = recv[4] + recv[5] \* 256

ser.reset\_input\_buffer()

iter = iter + 1

# Iterate over detected objects

for result in results:

print(iter, "> ", result)

for box in result.boxes:

x1, y1, x2, y2 = map(int, box.xyxy[0])

label = model.names[int(box.cls)]

# Calculate the distance

height = y2 - y1

distance\_mm = depth\_frame.get\_distance(x1 + (x2 - x1) // 2, y1 + (y2 - y1) // 2) \* 1000 # Convert to millimeters

dist = distance\_mm \* depth\_scale # Convert to meters

# Update nearest object and distance if closer than current nearest

if dist < min\_dist:

min\_dist = dist

nearest\_object = label

# Reset all GPIO pins to LOW

for pin in output\_pins:

GPIO.output(pin, GPIO.LOW)

# Set specific GPIO pin based on the nearest object

if (nearest\_object == 'person' or nearest\_object == 'umbrella'): #18

GPIO.output(output\_pins[0], GPIO.HIGH)

elif (nearest\_object == 'bicycle' or nearest\_object == 'motorcycle'): #19

GPIO.output(output\_pins[1], GPIO.HIGH)

elif (nearest\_object == 'dining table' or nearest\_object == 'couch' or nearest\_object == 'bench'): #21

GPIO.output(output\_pins[2], GPIO.HIGH)

elif (nearest\_object == 'traffic light'): #22

GPIO.output(output\_pins[3], GPIO.HIGH)

elif (nearest\_object == 'book'): #23

GPIO.output(output\_pins[4], GPIO.HIGH)

elif (nearest\_object == 'dog' or nearest\_object == 'cat' or nearest\_object == 'cow'): #24

GPIO.output(output\_pins[5], GPIO.HIGH)

elif (nearest\_object == 'car'): #26

GPIO.output(output\_pins[6], GPIO.HIGH)

elif (nearest\_object == 'bus'): #31

GPIO.output(output\_pins[7], GPIO.HIGH)

elif (nearest\_object == 'laptop' or nearest\_object == 'tv'): #11

GPIO.output(output\_pins[8], GPIO.HIGH)

elif (nearest\_object == 'refrigerator'): #12

GPIO.output(output\_pins[9], GPIO.HIGH)

# if(distance\_l<15):

# for pin in output\_pins:

# GPIO.output(pin, GPIO.HIGH)

# Draw the object label and distance on the frame

cv2.putText(frame, f"{label} {dist:.2f} m", (x1, y1 - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 2)

# Draw the bounding box color based on distance

if (dist <= 0.5):

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)

elif(0.5<dist<=1.5):

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 255), 2)

elif (dist>1.5):

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

# Print the nearest object and its distance

print(f"Nearest : {nearest\_object}, Distance: {min\_dist:.2f} meters.")

# Display the frame

cv2.imshow("Object Detection and Distance Estimation", frame)

# Exit if the user presses the 'q' key

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release the video capture and destroy all windows

cap.release()

cv2.destroyAllWindows()

**PV5**

**Requirements.txt**

absl-py==2.1.0

attrs==23.2.0

beautifulsoup4==4.12.2

cachetools==5.3.2

centroid-tracker==0.0.9

certifi==2023.11.17

cffi==1.16.0

charset-normalizer==3.3.2

colorzero==2.0

contourpy==1.1.1

cycler==0.12.1

filelock==3.13.1

flatbuffers==23.5.26

fonttools==4.44.3

gdown==4.7.1

google-auth==2.26.2

google-auth-oauthlib==1.0.0

gpiozero==2.0

grpcio==1.60.0

idna==3.4

importlib-metadata==7.0.1

importlib-resources==6.1.1

imutils==0.5.4

Jetson.GPIO==2.1.6

kiwisolver==1.4.5

Markdown==3.5.2

MarkupSafe==2.1.4

matplotlib==3.7.4

mediapipe==0.10.9

numpy==1.24.4

nvidia-pyindex==1.0.9

oauthlib==3.2.2

onnx==1.15.0

onnxscript==0.1.0.dev20231120

opencv-contrib-python==4.9.0.80

opencv-python==4.8.1.78

packaging==23.2

pandas==2.0.3

Pillow==10.1.0

protobuf==3.20.3

psutil==5.9.6

py-cpuinfo==9.0.0

pyasn1==0.5.1

pyasn1-modules==0.3.0

pycparser==2.21

pyparsing==3.1.1

pyrealsense2==2.54.2.5684

pyserial==3.5

PySocks==1.7.1

python-dateutil==2.8.2

pytz==2023.3.post1

PyYAML==6.0.1

requests==2.31.0

requests-oauthlib==1.3.1

RPi.GPIO==0.7.1

rsa==4.9

scipy==1.10.1

seaborn==0.13.0

six==1.16.0

sounddevice==0.4.6

soupsieve==2.5

tensorboard==2.14.0

tensorboard-data-server==0.7.2

thop==0.1.1.post2209072238

tqdm==4.66.1

typing\_extensions==4.8.0

tzdata==2023.3

ultralytics==8.1.23

urllib3==2.1.0

Werkzeug==3.0.1

zipp==3.17.0

**Code for Enabling Different Cluster every 4 second for 2 times:**

import Jetson.GPIO as GPIO

import time

output\_pins = [19,11,7,29,32,35,40]

C1 = [7,32]

C2 = [40,19]

C3 = [11,29,35]

C4 = [7,11,19]

C5 = [29,32,35,40]

GPIO.setmode(GPIO.BOARD)

for pin in output\_pins:

GPIO.setup(pin, GPIO.OUT, initial=GPIO.LOW)

try:

for i in range(2):

for pin in output\_pins:

GPIO.output(pin, GPIO.LOW)

GPIO.output(C1, GPIO.HIGH)

time.sleep(4)

GPIO.output(C1, GPIO.LOW)

GPIO.output(C2, GPIO.HIGH)

time.sleep(4)

GPIO.output(C2, GPIO.LOW)

GPIO.output(C3, GPIO.HIGH)

time.sleep(4)

GPIO.output(C3, GPIO.LOW)

GPIO.output(C4, GPIO.HIGH)

time.sleep(4)

GPIO.output(C4, GPIO.LOW)

GPIO.output(C5, GPIO.HIGH)

time.sleep(4)

GPIO.output(C5, GPIO.LOW)

except KeyboardInterrupt:

for pin in output\_pins:

GPIO.output(pin, GPIO.LOW)

GPIO.cleanup()

**Code for exporting torch model to onnx model:**

import torch

import torchvision.models as models

from ultralytics import YOLO

# Load your PyTorch model

model = YOLO("yolov8n.pt")

model.eval()

# Define dummy input (change this according to your model's input requirements)

dummy\_input = torch.randn(1, 3, 224, 224)

# Export the model to ONNX format

torch.onnx.export(model, dummy\_input, "resnet18.onnx", verbose=True)

print("Model exported successfully to 'resnet18.onnx'")

**Final Code using onnx: -** Unsuccessful due to memory error while converting torch model to onnx model.

import cv2

import numpy as np

import torch

# Load the ONNX model

onnx\_model = torch.onnx.load("yolov8.onnx")

# Set the input name and shape

onnx\_input\_name = "input"

onnx\_input\_shape = (1, 3, 416, 416)

# Create a dummy input tensor with the same shape and data type as the actual input

onnx\_dummy\_input = torch.randn(\*onnx\_input\_shape)

# Create an ONNX session

onnx\_session = torch.onnx.TracingSession(onnx\_model, input\_names=[onnx\_input\_name], output\_names=['output'])

# Set the camera matrix and distortion coefficients

camera\_matrix = np.array([[379.45187378, 0, 324.09848022], [0, 379.45187378, 238.02722168], [0, 0, 1]])

distortion\_coefficients = np.array([0.0, 0.0, 0.0, 0.0, 0.0])

# Set the video capture (replace 0 with your camera index if needed)

cap = cv2.VideoCapture(1)

while True:

# Read the frame

ret, frame = cap.read()

# Perform object detection

onnx\_outputs = onnx\_session.run(None, {onnx\_input\_name: onnx\_dummy\_input.to(device).cpu().numpy()})

onnx\_outputs = [torch.from\_numpy(onnx\_output) for onnx\_output in onnx\_outputs]

onnx\_outputs = [onnx\_output.to(device) for onnx\_output in onnx\_outputs]

results = model.postprocess(onnx\_outputs)

# Draw the bounding boxes and labels

for result in results:

for box in result.boxes:

x1, y1, x2, y2 = map(int, box.xyxy[0])

label = model.names[int(box.cls)]

# Calculate the distance

height = y2 - y1

distance\_mm = frame.get\_distance(x1 + (x2 - x1) // 2, y1 + (y2 - y1) // 2) \* 1000 # Convert to millimeters

dist = distance\_mm \* depth\_scale # Convert to meters

# Draw the distance on the rectangle

cv2.putText(frame, f"{label} {dist:.2f} m", (x1, y1 - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 2)

# Draw the bounding box

if (dist <= 0.5):

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)

elif(0.5<dist<=1.5):

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 255), 2)

elif (dist>1.5):

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

# Display the frame

cv2.imshow("Object Detection", frame)

# Exit if the user presses the 'q' key

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release the video capture and destroy all windows

cap.release()

cv2.destroyAllWindows()

**Final Code:**

import cv2

import numpy as np

import pyrealsense2 as rs

from ultralytics import YOLO

import Jetson.GPIO as GPIO

import serial

import time

iter = 0

# Load the YOLOv5 model

model = YOLO("yolov8n.pt")

# Set the reference distance (in meters)

ref\_distance = 5.0

# Define the output pins for GPIO control

output\_pins = [7,11,19,29,32,35,40]

C1 = [7,32]

C2 = [40,19]

C3 = [11,29,35]

C4 = [7,11,19]

C5 = [29,32,35,40]

# Define lists of stationary and semi-stationary objects

Stationary\_Objects = ['dining table','bench','couch','traffic light','tv','laptop','refrigerator','book']

Semi\_Stationary\_Objects = ['person','umbrella', 'bicycle', 'motorcycle','car','bus','dog','cat','cow']

# Initialize a serial connection on port '/dev/ttyTHS1' with baud rate 115200

ser = serial.Serial("/dev/ttyTHS1", 115200)

# Initialize GPIO

GPIO.setmode(GPIO.BOARD)

for pin in output\_pins:

GPIO.setup(pin, GPIO.OUT, initial=GPIO.LOW)

# Set the camera matrix and distortion coefficients

camera\_matrix = np.array([[379.45187378, 0, 324.09848022], [0, 379.45187378, 238.02722168], [0, 0, 1]])

distortion\_coefficients = np.array([0.0, 0.0, 0.0, 0.0, 0.0])

# Set the object size (in meters) at the reference distance

obj\_size = 1.0

# Set the video capture

cap = cv2.VideoCapture(1)

# Start the RealSense pipeline

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)

pipeline.start(config)

# Initialize the depth scale

depth\_sensor = pipeline.get\_active\_profile().get\_device().first\_depth\_sensor()

depth\_scale = depth\_sensor.get\_depth\_scale()

while True:

# Read the frame

ret, frame = cap.read()

(H, W) = frame.shape[:2]

# Perform object detection using YOLO

results = model(frame)

# Draw the bounding boxes and labels

frames = pipeline.wait\_for\_frames()

depth\_frame = frames.get\_depth\_frame()

color\_frame = frames.get\_color\_frame()

if not depth\_frame or not color\_frame:

continue

nearest\_object = None

min\_dist = float('inf')

# Calculating distance from lidar

count = ser.in\_waiting

if count > 8:

recv = ser.read(9)

ser.reset\_input\_buffer()

if recv[0] == 0x59 and recv[1] == 0x59: #python3

distance\_l = recv[2] + recv[3] \* 256

strength = recv[4] + recv[5] \* 256

ser.reset\_input\_buffer()

# Iterate over detected objects

for result in results:

for box in result.boxes:

x1, y1, x2, y2 = map(int, box.xyxy[0])

label = model.names[int(box.cls)]

# Calculate the distance

height = y2 - y1

distance\_mm = depth\_frame.get\_distance(x1 + (x2 - x1) // 2, y1 + (y2 - y1) // 2) \* 1000 # Convert to millimeters

dist = distance\_mm \* depth\_scale # Convert to meters

# Update nearest object and distance if closer than current nearest

if dist < min\_dist:

min\_dist = dist

nearest\_object = label

# Reset all GPIO pins to LOW

for pin in output\_pins:

GPIO.output(pin, GPIO.LOW)

# Set specific GPIO pin based on the nearest object

if (nearest\_object == 'person' or nearest\_object == 'umbrella'): #C1

GPIO.output(C1, GPIO.HIGH)

elif (nearest\_object == 'tv' or nearest\_object == 'laptop'): #C2

GPIO.output(C2, GPIO.HIGH)

elif (nearest\_object == 'dining table' or nearest\_object == 'couch' or nearest\_object == 'bench'): #C3

GPIO.output(C3, GPIO.HIGH)

elif (nearest\_object == 'traffic light'): #C4

GPIO.output(C4, GPIO.HIGH)

elif (nearest\_object == 'book'): #C5

GPIO.output(C5, GPIO.HIGH)

# elif (nearest\_object == 'dog' or nearest\_object == 'cat' or nearest\_object == 'cow'): #35

# GPIO.output(output\_pins[5], GPIO.HIGH)

# elif (nearest\_object == 'car'): #40

# GPIO.output(output\_pins[6], GPIO.HIGH)

if(distance\_l<15):

for pin in output\_pins:

GPIO.output(pin, GPIO.HIGH)

# Draw the object label and distance on the frame

cv2.putText(frame, f"{label} {dist:.2f} m", (x1, y1 - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 2)

# Draw the bounding box color based on distance

if (dist <= 0.5):

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)

elif(0.5<dist<=1.5):

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 255), 2)

elif (dist>1.5):

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

# Print the nearest object and its distance

print(f"Nearest : {nearest\_object}, Distance: {min\_dist:.2f} meters.")

# Display the frame

cv2.imshow("Object Detection and Distance Estimation", frame)

# Exit if the user presses the 'q' key

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release the video capture and destroy all windows

cap.release()

cv2.destroyAllWindows()

|  |  |
| --- | --- |
| PV4 | PV5 |
| * Exported pytorch model. | * Exported pytorch model and converted it into onnx model. |
| * Used pytorch model to implement the final code. | * Converted pytorch model to onnx model to use in final code but getting memory error while loading the final model. **Tried solving it but at last was not able to resolve it.** |
| * Not used Lidar | * Used Lidar for emergency situations where objects are too close, which will enable all the gpio pins. |
| * Directly implemented the finalized gpio pins in the ‘while’ loop which will make the code robust. | * Created clusters for easy implementation of code in future, where the user just need to change the gpio pin numbers in the variable C1,C2,C3,C4 or C5 in the code according to the usage. |
| * Final code takes approx. 60 seconds to start giving outputs. | * Latency is reduced. The final code takes approx. 40 seconds to start giving the outputs. |
| * 10 Classes are created and used. | * Classes are reduced to 5. |
|  | * Tried converting pytorch model to tensorrt engine form which will improve the accuracy and latency but since the development of tensorrt is no longer in chase, we were unable to export it into the engine form in python 3.8. version, the version which is supported by most of the other libraries such as numpy, ultralytics, matplotlib etc.. . Although tenorrt is pre-installed in python 3.6 version, but in this version other libraries are not supported. |