Group 5

Indoor ICT building Object Detection for Visual Impairment

ITCS498 - Special Topics in Computer Science

Team Members

and our main roles

Nipphit Apisitpuwakul

Semantic segmentation model

Leenawat Honglerdnapakul

Depth estimation model

Phuvanarth Wangmuk

Data preprocessing

Introduction

Problem

- Individuals with visual impairments face challenges in navigating indoor environments
- The risk of accidents and injuries increases without proper navigation aids in indoor environments

Objective

Design and implement technology-based visual aids to enhance:

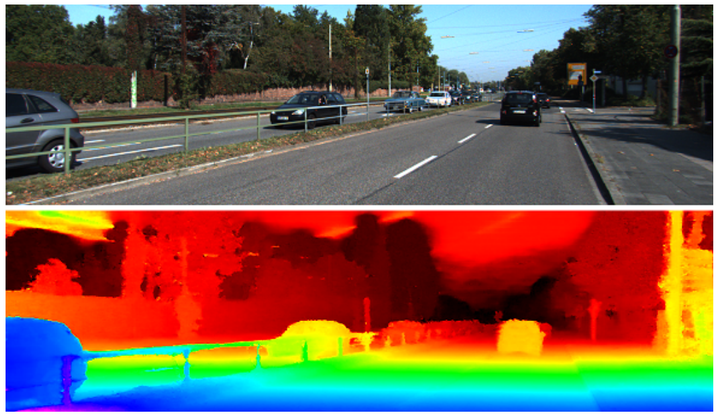
- Safety
- Convenience

Proposed solution

Semantic segmentation

a process in which an image is divided into different regions or segments

Depth Estimation



Measuring the distance to objects in the environment

Scope

- Indoor ICT building

  3 Classes:

  - Floors/Ceilings
  - Walls
  - Obstacle (including people)

3

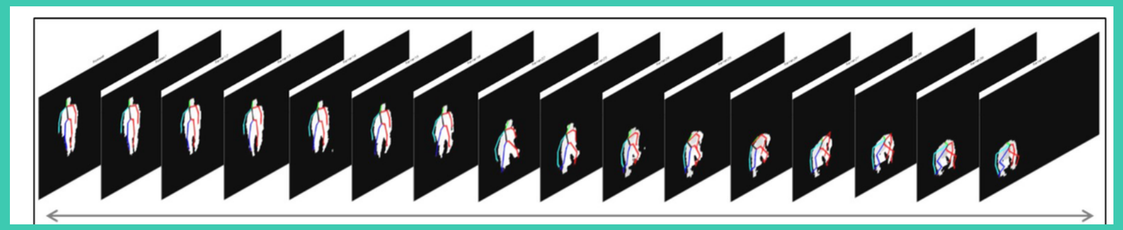
Classes

Methodology

Input

- RGB indoor Images of ICT building
- Corresponding segmentation images as ground truth segmentation masks

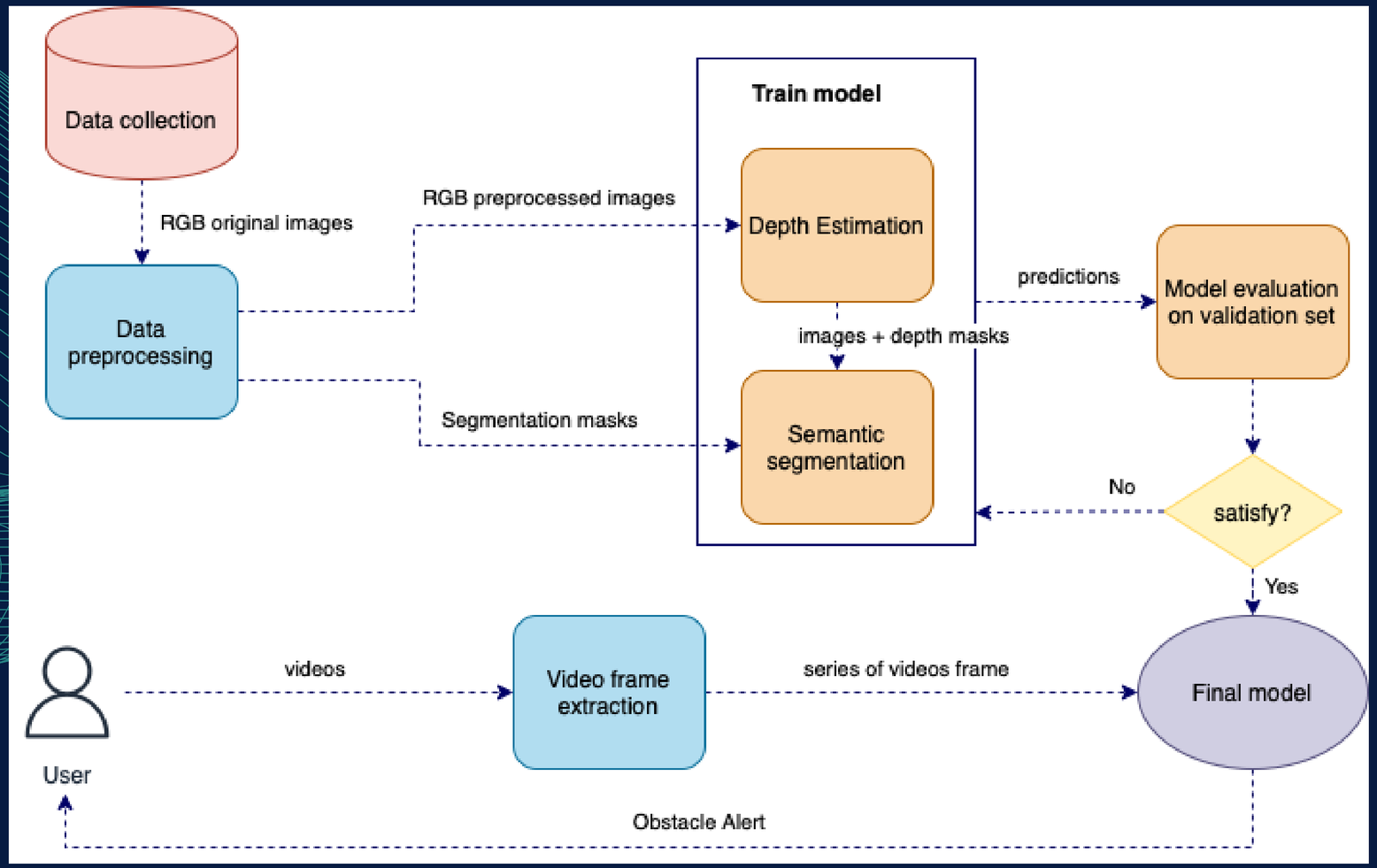
Output



- Sequence of Segmentation maps
- ~~Sequence of Depth maps~~

an alert telling that the obstacles ~~are close~~ to the user

Overall Architecture



Technology used

Data Preprocessing

- Adobe illustrator : segmentaion mask

Model

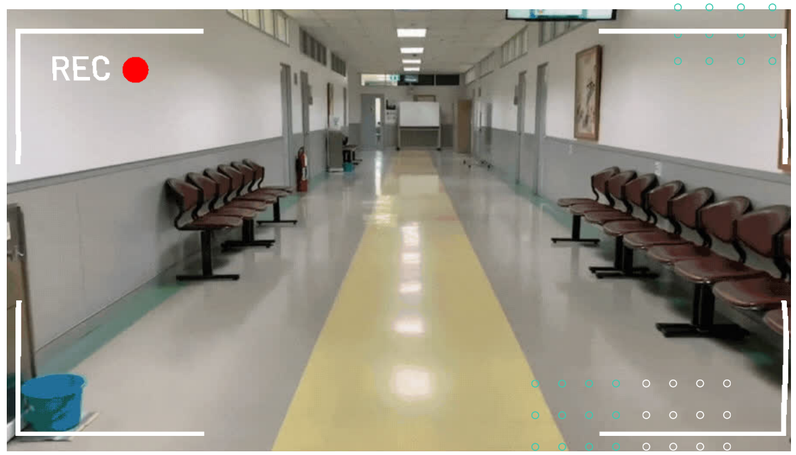
- Pytorch

Video frame extraction

- OpenCV (Open Source Computer Vision)

Data Collection

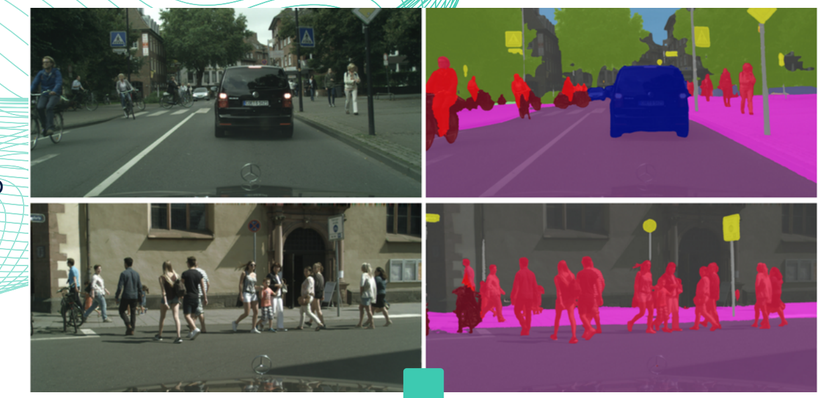
Collect new data by capturing images and videos of the building and its surroundings from multiple angles and lighting conditions.



Labelled the object is important to assign categories to objects in the scene, such as obstacles, floors and walls.

Implementation

Semantic segmentation



depth estimation

1. Semantic segmentation

- **U-net** : suitable for small dataset

2. Depth estimation

- **MiDaS** (Monocular Depth Estimation in Real-Time with Adaptive Scale Network): a pre-trained model that can estimate depth in real-time

3. Combining U-net & MiDaS

- **Concatenate output of MiDaS to input of U-net:** dept estimation can help improve performance in distinguishing the objects

Implementation

Depth estimation

```python
def get_depth_map(self, img):
    # Process the image with the MiDaS model
    input_batch = self.transform(img).to(self.device)

    with torch.no_grad():
        prediction = self.midas(input_batch)
        prediction = torch.nn.functional.interpolate(
            prediction.unsqueeze(1),
            size=img.shape[:2],
            mode="nearest-exact",
        ).squeeze()
    depth_map = prediction.cpu().numpy()

    return depth_map
```

When Loading IndoorDataset

```python
# Load the depth map with MiDaS
depth_map = self.get_depth_map(image)

# Apply transforms to the image, mask, and depth map
if self.data_transform is not None:
    augmentations = self.data_transform(image=image, mask=mask)
    transformed_image = augmentations['image']
    mask = augmentations['mask']
    mask = np.floor(mask / 80)

    depth_transforms = A.Compose([
      A.Resize(IMAGE_HEIGHT, IMAGE_WIDTH),
      A.Rotate(limit=35, p=1.0),
      A.HorizontalFlip(p=0.5),
      A.VerticalFlip(p=0.1),
      ToTensorV2()
    ])

    depth_map = depth_transforms(image=depth_map)["image"]

# Convert the transformed image and depth map to tensors
image = transformed_image
depth_map = depth_map.float()

# Concatenate the depth map with the image along the channel dimension
image = torch.cat([image, depth_map], dim=0)

return image, mask
```

Concat depth map to the images

Implementation

Data transformation

```
train_transform = A.Compose([
    A.Resize(IMAGE_HEIGHT,IMAGE_WIDTH),
    A.Rotate(limit=35,p=1.0),
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.1),
    A.Normalize(
        mean = [0.0,0.0,0.0],
        std = [1.0,1.0,1.0],
        max_pixel_value=255.0
    ),
    ToTensorV2()
])
validation_transform = A.Compose([
    A.Resize(IMAGE_HEIGHT,IMAGE_WIDTH),
    A.Normalize(
        mean = [0.0,0.0,0.0],
        std = [1.0,1.0,1.0],
        max_pixel_value=255.0,
    ),
    ToTensorV2()
])
```

Unet model (partial)

```
class UNet(nn.Module):
    def __init__(self, n_channels=4, n_classes=3, bilinear=True):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bilinear = bilinear

        self.inc = DoubleConv(n_channels, 64)
        self.down1 = Down(64, 128)
        self.down2 = Down(128, 256)
        self.down3 = Down(256, 512)
        self.down4 = Down(512, 512)
        self.up1 = Up(1024, 256, bilinear)
        self.up2 = Up(512, 128, bilinear)
        self.up3 = Up(256, 64, bilinear)
        self.up4 = Up(128, 64, bilinear)
        self.outc = OutConv(64, n_classes)

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        logits = self.outc(x)
        return logits
```

```
model = UNet(4,3)
model = model.to(DEVICE)
```

Implementation

Training + Testing on validation set

```
loss_fn = nn.CrossEntropyLoss()
scaler = torch.cuda.amp.GradScaler()

optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

```
def train_fn(loader, model, optimizer, loss_fn, scaler):
    model.train()
    loop = tqdm(loader)

    for batch_idx, (data, targets) in enumerate(loop):
        data = data.to(device=DEVICE)
        targets = targets.long().to(device=DEVICE)

        # forward
        with torch.cuda.amp.autocast():
            predictions = model(data)
            loss = loss_fn(predictions, targets)

        # backward
        optimizer.zero_grad()
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        # update tqdm loop
        loop.set_postfix(loss=loss.item())
```

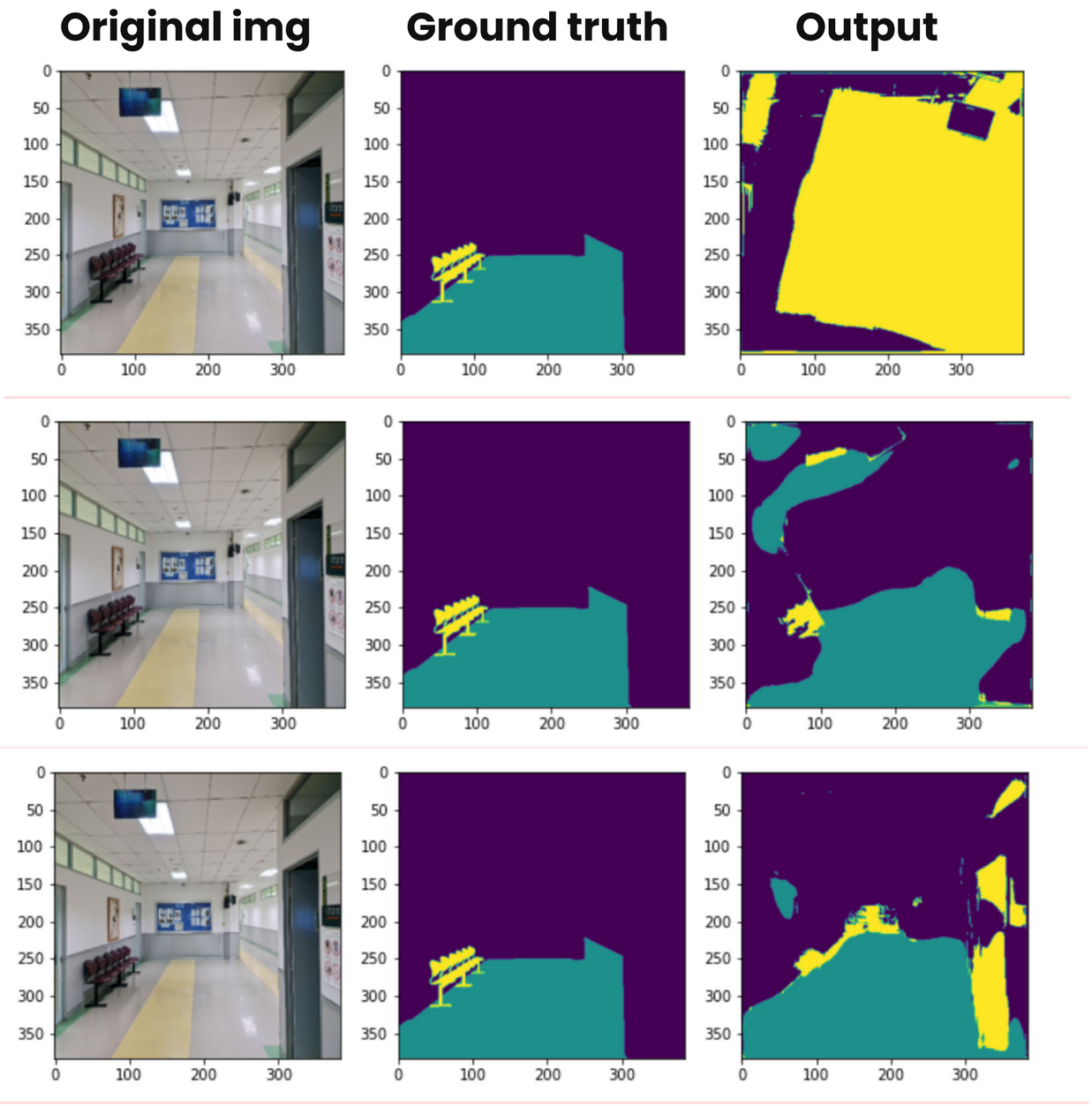
```
for epoch in range(EPOCHS):
    train_fn(train_dataloader, model, optimizer, loss_fn, scaler)

    # save model
    checkpoint = {
        "state_dict": model.state_dict(),
        "optimizer":optimizer.state_dict(),
    }
    save_checkpoint(checkpoint)

    # check accuracy
    check_accuracy(valid_dataloader, model, device=DEVICE)

    # test on validation set
    model.eval()
    imgs, masks = next(iter(valid_dataloader))
    imgs = imgs.to(device=DEVICE)
    logits = model(imgs)
    pred_masks = torch.argmax(logits, dim=1)
    i = 0
    fig, axs = plt.subplots(1, 3, figsize=(12,6))
    axs[0].imshow(imgs[i].cpu().numpy().transpose(1,2,0))
    axs[1].imshow(masks[i].cpu().numpy())
    axs[2].imshow(pred_masks[i].cpu().numpy())
    plt.show()
    plt.close('all')
```

Example of training process



Implementation

Testing on Video

```
vidcap = cv2.VideoCapture(os.path.join(input_folder_path+'vdo_2.mp4'))
frame_count = int(vidcap.get(cv2.CAP_PROP_FRAME_COUNT))
target_img_w = 720
target_img_h = 480
```

Create an alert
if the class is obstacle

```
success, image = vidcap.read()
with tqdm(total=frame_count, position=0, leave=True) as pbar:
    predictions = []
    while success:
        image = cv2.resize(
            image, (target_img_w, target_img_h),
            interpolation=cv2.INTER_LINEAR)

        # Generate the depth map using the MiDaS model
        depth_map = get_depth_map(image, midas, transform, DEVICE)

        # Convert the image and depth map to a PyTorch tensor and normalize them
        image_tensor = torch.tensor(image, dtype=torch.float32).permute(2, 0, 1).unsqueeze(0) / 255.0
        depth_map_tensor = torch.tensor(depth_map, dtype=torch.float32).unsqueeze(0).unsqueeze(0)
        image_tensor = torch.cat([image_tensor, depth_map_tensor], dim=1)
        image_tensor = image_tensor.to(DEVICE)

        # Uss U-Net model for segmentation
        logits = model(image_tensor)
        pred_masks = torch.argmax(logits, dim=1).squeeze(0).cpu().numpy()

        # Visualize the segmentation results
        out_image = np.zeros_like(image)
        for i in range(0, 3):
            masks = pred_masks == i
            if i == 0:
                color = [61, 11, 81]
            elif i == 1:
                color = [69, 142, 139]
                font = cv2.FONT_HERSHEY_SIMPLEX
                text = "Detected"
                position = (50, 50)
                font_scale = 1
                font_color = (255, 255, 255)
                line_type = 2
                cv2.putText(out_image, text, position, font, font_scale, font_color, line_type)
            else:
                color = [250, 230, 85]

            out_image[masks] = color  # Set the color for the current class

        out_stream.stdin.write(
            out_image
```

Model Deployment

Implement on

Library

```
pip install timm
pip install pyyaml==5.1
pip install ffmpeg-python

import os
import torch
import torch.nn as nn

import cv2
import ffmpeg
from tqdm import tqdm
```

Required steps

- Upload my_checkpoint.pth.tar in google drive
- Define the UNET model
  make sure the model architecture matches the architecture of the saved checkpoint
- Load the saved checkpoint
- Import midas


Project
Demonstration

Evaluation

Data splitting

~ 70%

~ 30%

Training set	Validation set	Test set
30 pictures	6 pictures	6 pictures

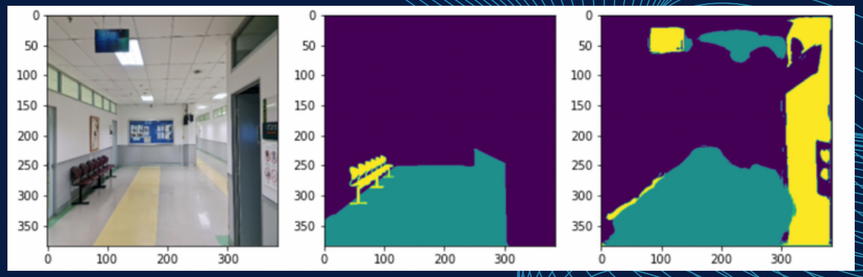
Performance metrics

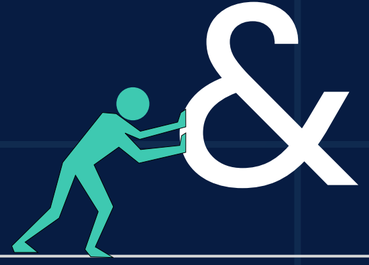
- **Accuracy**

  Compare output with ground truth pixel by pixel

- **Results**

  Accuracy on validation set: 83.58%
  Accuracy on test set: 64.72%





& Limitations Future Works

Limitations

- Limited time for gathering and labeling a comprehensive dataset, and implementing a model that can accurately classify multiple classes of indoor environments
- Limited access to visually impaired individuals for user testing and feedback, which can affect the accuracy and usability of the system

Future work

- Gathering more image/video datasets
- Expanding the scope to cover more classes of indoor environments
- Fixing the Unet model architecture and Fine-tuning the Midas model to improve accuracy
- Implementing it to be able to test on real-time video