Experiment Report on Obstacle Avoidance Procedures

Zening Lu Yifan Wang Chongjie Zhao Changwei Lv Wen Wen

January 15, 2024

Contents

Abstract

This project investigates a method for guiding blindness based on depth image segmentation and human body model orthographic projection algorithm. Based on the depth image segmentation, the project divides the depth image acquired by the depth camera into passable areas according to the location of obstacles and provides relevant information for blindness guidance, thus improving the accuracy of blindness guidance and obstacle avoidance.

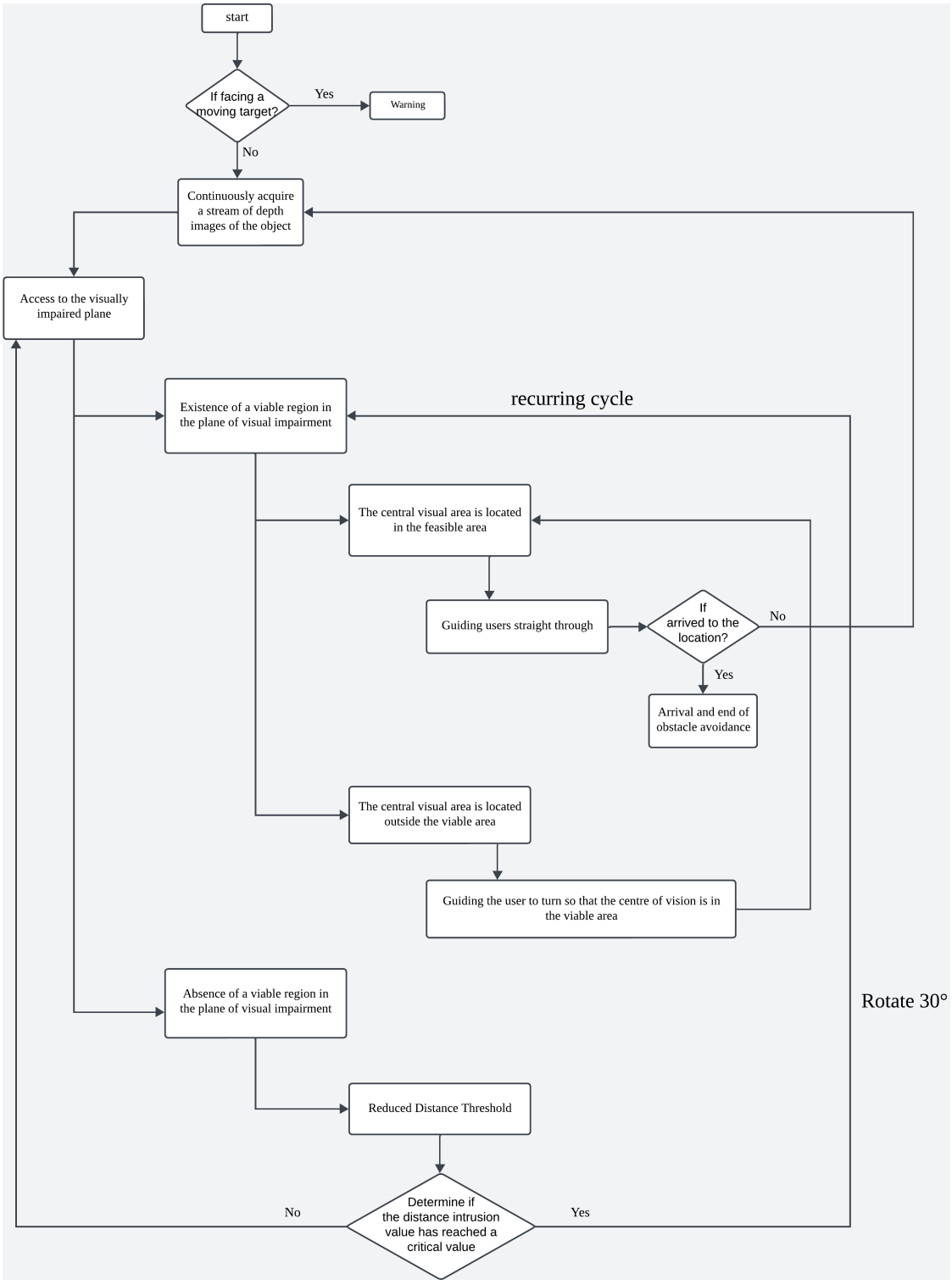

Figure 1: Flowchart of The Whole Process

1 Function Introduction With Examples

The present embodiment will provide a method for recognising if the project faced by the camera is moving, and if yes, there is a following method for guiding blindness based on depth image segmentation and human body model orthographic projection algorithm. Here are 3 examples where the first one shows the moving condition detection part while the second and the third one show the obstacle-avoidance part.

1.1 Example 1

Recognising moving or not At the beginning of the procedure, our program will identify whether the object it is facing is moving or not. If the object is moving, then the program will give feedback and end the process. If the object in front of it is stationary, then the object will run the next obstacle avoidance procedure to avoid obstacles. Example: At the entrance to the basketball court of a school, a moving person and a bicycle are recognised in an image captured by the camera and feedback is given. The screen cut of the identification result is shown in FIG2

Figure 2: Detecting Result in Example1

1.2 Example 2

Obstacle-Avoidance If the facing project is static, the program will run to step2, which means the process goes to obstacle-avoidance. In the scene shown in FIG3.

Figure 3: Depth Image

At a distance from the 3m left front of the depth camera there exists an area height of 2.5m and width of 1m a doorway, and the two sides of the doorway are impenetrable obstacles; blind guide and obstacle avoidance are realized in this scene, and the specific flow is as follows:

Step1 Place a small square with a side length of 10 cm at a distance of 100 cm from the Orbbec Gemini2 binocular structured light depth camera, record the minimum depth value d_1 on the small square and the number of pixel points n_1 occupied by the side length of the square in the depth image at this time; move the small square backward to a distance of 300 cm from the depth camera, and find out the actual size and its corresponding pixel points in the depth image from the minimum depth value and the number of pixel points in the side length of the square based on the depth image at this time. According to the depth image, record the minimum depth value d_2 and the number of pixel points n_2 occupied by the side length of the square; through the minimum depth value and the number of pixel points, find the scaling ratio f(d) between the actual size and its corresponding number of pixel points in the depth image. Where f(d) is represented by:

$$f(d) = kd + b, \quad k = \frac{n_1 - n_2}{(d_1 - d_2)a}\ ;\ b = \frac{n_2 d_1 - n_1 d_2}{(d_1 - d_2)a}$$

d is the depth value of the resulting depth image; the user stands facing the depth camera,

obtains the minimum depth value of the human body in the depth image at this time d_0, the number of horizontal pixel points n_x, and the number of vertical pixel points n_y, and substitutes d_0, n_x, and n_y into the scaling ratio to obtain the height of the user of 176 cm as well as the shoulder width of 46 cm.

Step2 Set the initial threshold distance to 300 cm, which is a distance that the depth camera can image stably; consider the reaction time of the human body, set the minimum safety distance to 100 cm, and set the current threshold distance to D. According to the scaling ratio f(d) and the current threshold distance D, appropriately zoom in on the collision area on the front side of the human body, the The human body frontal projection matrix A_{321*79} is obtained, and all the element values in the human body frontal projection matrix A_{321*79} are initialised to 1.

Step3 The image obtained by the depth camera is transferred to the standard coordinate system, and then the relative position coordinates of the object and the depth camera are obtained by camera transformation, and then the camera-altered object is projected onto the $[-1,1]^3$ three-dimensional space by perspective projection transformation, and finally the coordinates of the ground are shifted to the u-w plane by the viewport transformation, and the specific processing is as follows:

Step3-1 Define the depth camera view direction as g and the vertical upward direction along the view direction as t. Establish the standard coordinate system (u,v,x) as defined below:

$$u = -\frac{t \times w}{\|t \times w\|}, \quad v = w \times u, \quad w = -\frac{g}{\|g\|};$$

Step3-2 Set the depth camera position e as the origin and define the transformation matrix as V_{cam} with the following equation: where x_u, x_v, x_w, y_u,y_v , y_w,z_u ,z_v , $z_w$are the projections

$$V_{cam} = \begin{bmatrix} u & v & w & e \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

of the x y z components of the object under the original coordinate system under the standard coordinate system O_{uvw}, respectively; x_e, y_e, $z_e$is the coordinate position e of the depth camera position; the right matrix indicates that the first step of the operation moves the depth camera

position to the origin, and the left matrix indicates that the second step of the operation rejoins it with the standard coordinate system O_{uvw}.

Step3-3 Define the visual space in which the camera is located as follows:

l = left plane ; r = right plane; b = bottom plane ; t = top plane; n = near plane ; f = far plane

Step3-4 The depth camera view matrix T_{view} is converted to 3D space $[-1, 1]^3$ and convolution calculation is performed to obtain the orthogonal transformation matrix V_{ortho} with the following formula:

$$V_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step3-5 The viewpoint is regarded as the v axis origin, and the place plane where w=-n is regarded as the projection plane, using the similar triangle property and the two-point property, substituting any two points of the far and near planes, respectively, to obtain the projection transformation matrix $V_{persp->ortho}$ with the following formula:

$$V_{persp \to ortho} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Step3-6 Define the perspective projection transformation matrix V_{per} with the following formula, with the result:

$$V_{per} = V_{ortho} V_{persp \to ortho}$$

$$V_{per} = V_{ortho}V_{persp\to ortho} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Step3-7 The depth camera viewpoint matrix in 3D space$[-1,1]^3$ is converted to the standard coordinate system to obtain the viewport transformation matrix V_{view} with the following formula:

$$\begin{bmatrix} \frac{w}{2} & 0 & 0 & \frac{w}{2} \\ 0 & \frac{h}{2} & 0 & \frac{h}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step3-8 Define the coordinate system change matrixT_{view} with the following equation:

$$T_{view} = V_{viewport}V_{per}V_{cam}$$

The standard viewing angle matrix $V_s$in the standard coordinate system is obtained by calculating the following equation:

$$V_s = T_{view}V_{in}$$

Step4 A depth image matrix of the road obstacle in front of the user and the depth value of each pixel point in the depth image matrix are obtained by the depth camera of the guide device, and the plumb distance of each pixel point in the depth image from the depth camera is calculated, and at the same time the depth image matrix is binarised, and the specific processing is as follows:

Step4-1 A depth image matrix $M_{480*640}$ of the road obstacle in front of the user is obtained by the depth camera of the guide device, and the value of each pixel point in the depth image matrix $M_{480*640}$ indicates the size of the distance from the corresponding point on the object to the depth camera, and the horizontal distance z from the object to the camera in the depth image is calculated according to the collinear theorem in three-dimensional space;

Step4-2 The depth image matrix $M_{480*640}$ is binarised to obtain the binarised matrix $M_{480*640}$ to obtain the binarised image as shown in Fig4;

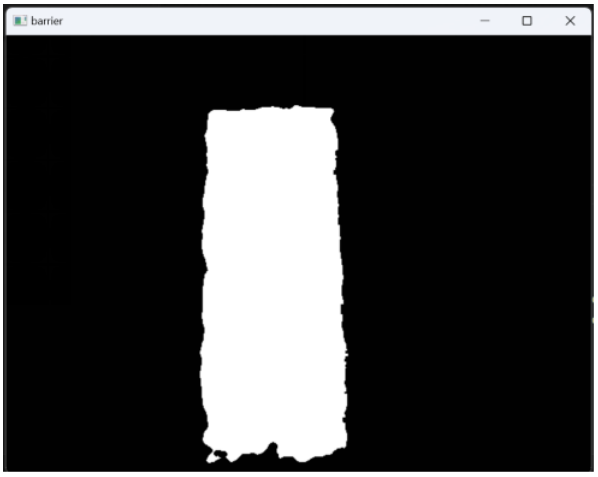

Figure 4: The Binarised Image

The values greater than or equal to the initial threshold distance z are set to 1, and the horizontal distances less than the initial threshold distance z are set to 0.

Step5 According to the human body positive projection matrix $A_{321\times79}$, take the binarized matrix $N_{480\times640}$ from the 126th row to the 448th row. Obtain the human body positive projection matrix $A_{321\times79}$'s rows from the same matrix $C_{321\times640}$.

In the matrix $C_{321\times640}$, perform a traversal scanning operation on the human body positive projection matrix $A_{321\times79}$. Define i as the number of times of traversal scanning. If i consecutive scanning results in the ONES matrix, record the leftmost column of the human body positive projection matrix during the ith traversal as the number of columns X_i.

Define the number of columns where the line in the binarized matrix $N_{480\times640}$ is $X_0 = 320$. Define the number of columns where the line in the region can be passed as

$$Y_0 = \frac{(X_{79}-X_1+321)}{2}.$$

When the scanning appears to be interrupted, calculate the difference between the two horizontal coordinate distances of X_0 and Y_0:

$$\delta_X = X_0 - \frac{(X_{79}-X_1+321)}{2} \ ;\delta_\theta = \arctan\left(\frac{\delta_X}{Df(D)}\right).$$

By calculating the rotational angle δ_θ, reset $X_i = 0, i = 0$. Iteratively update the rotational angle through iterative traversal scanning to obtain the absolute minimum value of the rotational angle.

Define i as the number of times of traversal scanning. The absolute minimum of $|\delta_\theta| = 30°$ and $\delta_{\theta_{\min}} = -7°$.

Step6 Guide the user in the direction of the left front $-7°$ and re-execute steps S4-S6

1.3 Example 3

Direction Changing The present embodiment will provide a method for guiding a blind person based on depth image segmentation and human body model orthographic projection algorithm according to the flowchart shown in FIG5, in a scenario as shown in FIG4.

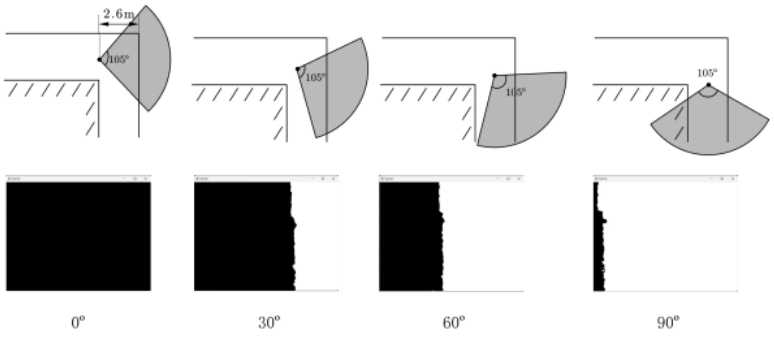

Figure 5: The position information

There exists a narrow right-turning road in front of the user, there is an impenetrable wall at a distance of 260 cm in front of the depth camera worn by the user, and there exists a road on the right side of the road that can be passed by the user in a safe way; in such a scenario In this scenario, the user is safely guided to complete the right turn, the specific process is as follows:

Step1-Step4 are same as example 2

Step5 According to the human body positive projection matrix $A_{321\times79}$, take the binarized matrix $N_{480\times640}$ from the 126th row to the 448th row. Obtain the human body positive projection matrix $A_{321\times79}$'s rows from the same matrix $C_{321\times640}$.

In the matrix $C_{321\times640}$, perform a traversal scanning operation on the human body positive projection matrix $A_{321\times79}$. Define i as the number of times of traversal scanning. If i consecutive

scanning results in the ONES matrix, record the leftmost column of the human body positive projection matrix during the ith traversal as the number of columns X_i.

Define the number of columns where the line in the binarized matrix $N_{480\times640}$ is $X_0 = 320$. Define the number of columns where the line in the region can be passed as

$$Y_0 = \frac{(X_{79}-X_1+321)}{2}.$$

When the scanning appears to be interrupted, calculate the difference between the two horizontal coordinate distances of X_0 and Y_0:

$$\delta_X = X_0 - \frac{(X_{79}-X_1+321)}{2} \ ; \ \delta_\theta = \arctan\left(\frac{\delta_X}{Df(D)}\right).$$

By calculating the rotational angle δ_θ, reset $X_i = 0, i = 0$. Iteratively update the rotational angle through iterative traversal scanning to obtain the absolute minimum value of the rotational angle.

Define i as the number of times of traversal scanning. The absolute minimum of $|\delta_\theta| = 180°$ Judging that the current route has no passable area.Set the current threshold distance D = 300cm - 50cm = 250cm. judging that the current threshold D distance is not less than the minimum safe distance C = 100cm, re-execute steps S4-S5 , iteratively updating the angle of rotation by traversing the scanning iteratively, and obtaining the absolute minimum value of the angle of rotation, $|\delta_\theta| = 30°$

Step6 Guide the user in the direction of the right front 30° and re-execute steps S4-S6 until the user is safely guided through the right steering 90°.

2 Environment installation and programme usage

1. Please download the model weights file (yolov5s/yolov5x etc.) from the following address https://github.com/ultralytics/yolov5 and copy them to yolov5 and deepth directories, or use the existing weights in deepth directly.

2. Please configure the relevant dependencies in anaconda environment according to the requirements, or use the command line command Pip install -r requirements

```
# Base
----------------------------------------
gitpython
ipython  # interactive notebook
matplotlib>=3.2.2
```

```
numpy>=1.18.5
opencv-python>=4.1.1
Pillow>=7.1.2
psutil  # system resources
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
thop>=0.1.1
torch>=1.7.0
torchvision>=0.8.1
tqdm>=4.64.0
# protobuf<=3.20.1

# Logging
--------------------------------------------------------------------
tensorboard>=2.4.1
# clearml>=1.2.0
# comet

# Plotting
--------------------------------------------------------------------
pandas>=1.1.4
seaborn>=0.11.0
#matplotlib

# Export
----------------------------------------------------------------------
# coremltools>=6.0  # CoreML export
# onnx>=1.9.0  # ONNX export
# onnx-simplifier>=0.4.1  # ONNX simplifier
# nvidia-pyindex  # TensorRT export
# nvidia-tensorrt  # TensorRT export
# scikit-learn<=1.1.2  # CoreML quantization
# tensorflow>=2.4.1  # TF exports (-cpu, -aarch64, -macos)
# tensorflowjs>=3.9.0  # TF.js export
# openvino-dev  # OpenVINO export

# Deploy
----------------------------------------------------------------------
# tritonclient[all]~=2.24.0

```

```
# Extras
    ----------------------------------------------------------------------
# mss  # screenshots
# albumentations>=1.0.3
# pycocotools>=2.0  # COCO mAP
# roboflow
# ultralytics

# Deepth
    ----------------------------------------------------------------------
#cv2
#mpl-toolkits mplot3d.Axes3D

# visualization
    ----------------------------------------------------------------------
#tkinter
```

Listing 1: r-requirements

(a) Please configure and install the driver or SDK of Orbbec Gemini2 camera to make sure the camera can connect and capture images properly $https://developer.orbbec.com.cn/develop_details.htm$ 2

(b) Disable the Webcam (if any) in PC Settings - Bluetooth and Other Devices - Camera interface to ensure that there will be no camera call conflicts when the code is running.

Project use:

Directly in vscode/pycharm to run deepth in the three source code files can be, or in the path of deepth in the command line window to run e.g: python object_detection.py –source 0 (according to the number of cameras connected to the computer in the setup interface to determine the serial number, the first 0, and so on)

3 Implementation and Code Analysis

3.1 Condition Detecting

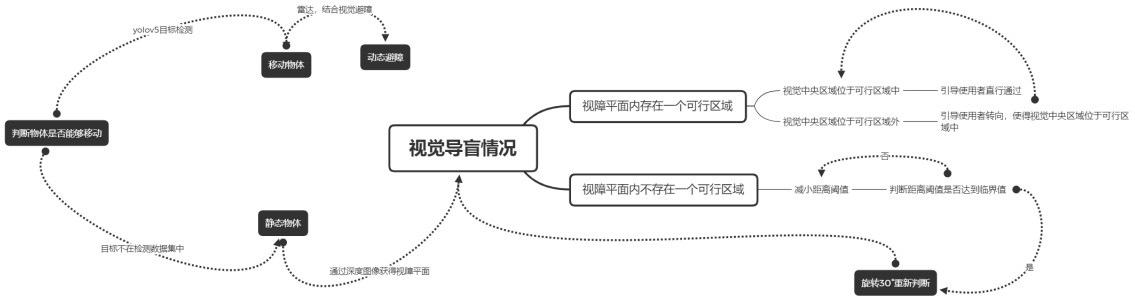

Figure 6: Process Flowchart of The Detecting Part

For recognising moving objects, we used yolov5 to aid detection. We called the coco128 dataset in yolo and selected typical moving objects on general road surfaces contained in the dataset to train the model.

```
weights=ROOT / 'yolov5s.pt',  # model path or triton URL
        source=ROOT / "0",  # file/dir/URL/glob/screen/0(webcam)
        data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
        imgsz=(640, 640),  # inference size (height, width)
        conf_thres=0.25,  # confidence threshold
        iou_thres=0.45,  # NMS IOU threshold
        max_det=1000,  # maximum detections per image
        device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
        view_img=False,  # show results
        save_txt=False,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=[0,1,2,3,5,7,15,16],  # filter by class: --class 0, or --class 0 2
    3
```

Listing 2: Using Coco Data Set

To allow the model to identify the moving objects on the road surface in the captured images. Because of the uncertainty of movable objects, our subsequent obstacle avoidance procedure can only avoid obstacles on static road conditions, so when a moving object is detected on the road, the procedure will feedback "Moving object detected !" when a moving object is detected on the road, the programme will give a feedback "Moving object detected !" and sound a beep to stop the programme. On the contrary, if the programme detects that the road condition is static, the programme will feedback "No risk of mobility disturbance" and make a voice prompt to continue the next obstacle avoidance step.

```
1
2           if len(det):
3               text = "Moving object detected !"
4               # Rescale boxes from img_size to im0 size
5               det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).
    round()
6               for c in det[:, 5].unique():
7                   n = (det[:, 5] == c).sum()  # detections per class
8                   s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string
9               for *xyxy, conf, cls in reversed(det):
10                  if save_txt:  # Write to file
11                      xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).
    view(-1).tolist()  # normalized xywh
12                      line = (cls, *xywh, conf) if save_conf else (cls, *xywh)
    # label format
13                      with open(f'{txt_path}.txt', 'a') as f:
14                          f.write(('%g ' * len(line)).rstrip() % line + '\n')
15                  if save_img or save_crop or view_img:  # Add bbox to image
16                      c = int(cls)  # integer class
17                      label = None if hide_labels else (names[c] if hide_conf
    else f'{names[c]} {conf:.2f}')
18                      annotator.box_label(xyxy, label, color=colors(c, True))
19                  if save_crop:
20                      save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c]
     / f'{p.stem}.jpg', BGR=True)
21          else:
22              text ="No risk of mobility disturbance"
```

Listing 3: Feedback from The Detecting Result

3.2 Obstacle-Avoidance

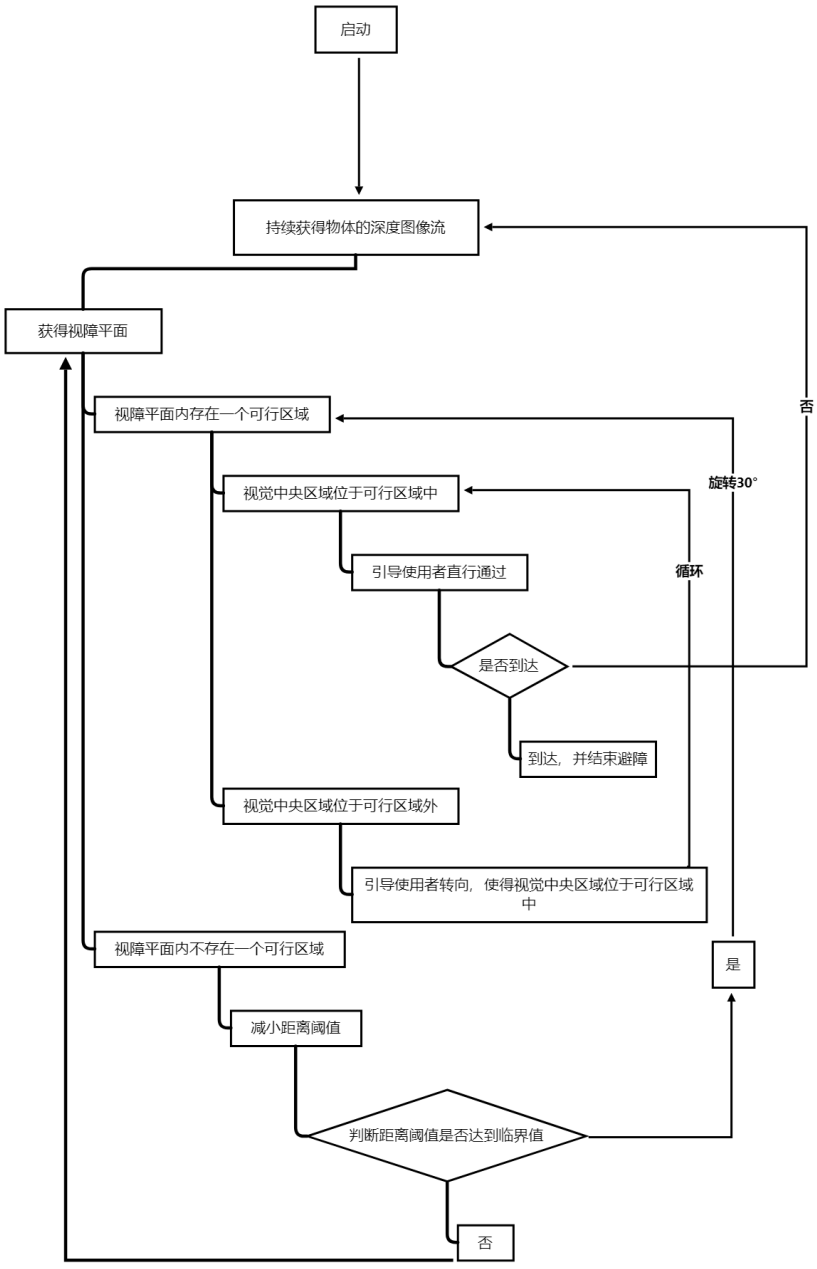

Figure 7: Process Flowchart of The Obstacle-Avoidance Part

Use the TwoInputBox class to obtain user input for shoulder width and height.

1. Display a dialog box for users to input shoulder width and height.

2. Retrieve the input values and convert them to floating-point numbers.

```
two_input_dialog = TwoInputBox(root, "Information Input")
shoulder_width = float(two_input_dialog.value1)
height = float(two_input_dialog.value2)
```

Retrieve the depth image Open the depth camera, retrieve the depth image, and obtain the color image.

```
orbbec_cap = cv.VideoCapture(0, cv.CAP_OBSENSOR)
ret_bgr, bgr_image = orbbec_cap.retrieve(flag=cv.CAP_OBSENSOR_BGR_IMAGE)
ret_depth, depth_map = orbbec_cap.retrieve(flag=cv.CAP_OBSENSOR_DEPTH_MAP)
```

Cloud map Based on the obtained depth map, a point cloud map is generated.As shown in FIG8(Point cloud example (non-avoidable)) and FIG9(Point cloud example (avoidable)).

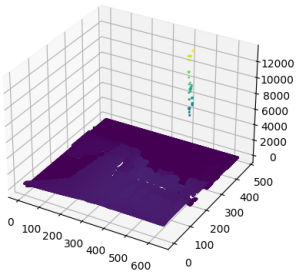

Figure 8: Process Flowchart of The Obstacle-Avoidance Part

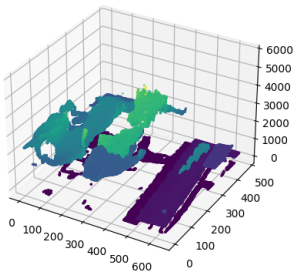

Figure 9: Process Flowchart of The Obstacle-Avoidance Part

```
                if ret_depth:
                    current_time = time.time()
                    if current_time - last_update_time > update_interval:
                        last_update_time = current_time

                        X, Y, Z = depth_to_pointcloud(depth_map)
                        update_pointcloud_plot( X, Y, Z)
                        pointcloud_image_counter +=1
                color_depth_map = cv.normalize(depth_map, None, 0, 255, cv.
    NORM_MINMAX, cv.CV_8UC1)
                cv.imshow("Depth",  color_depth_map)

                cv.imshow('barrier', barrier_mat)

```

Manually or automatically set the depth threshold Users can manually set the depth threshold or the program can automatically adjust based on the scene. The program creates an adjustable slider for manually setting the depth threshold.

```
cv.namedWindow(window_name)
cv.createTrackbar("dist1", window_name, initial_threshold, 1000, set_thre1)
```

Indicating areas Generate an obstacle matrix based on the depth threshold, indicating areas with depth less than the threshold as obstacles.

Convert the depth map into an obstacle matrix based on the depth threshold.

```
barrier_mat = np.zeros(depth_map.shape)
for i in range(len(depth_map)):
    for j in range(len(depth_map[0])):
        if depth_map[i][j] < current_threshold:
            barrier_mat[i][j] = 0
        else:
            barrier_mat[i][j] = 1
```

Display Display the depth image, obstacle matrix, and current threshold. Use OpenCV to display the depth image, obstacle matrix, and current threshold.

```
cv.imshow("Depth", color_depth_map)
cv.imshow('barrier', barrier_mat)
cv.putText(bgr_image, f"Threshold: {current_threshold}", (10, 30), cv.
    FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
cv.imshow(window_name, bgr_image)
```

Obstacle avoidance achieving Achieve obstacle avoidance by automatically adjusting the threshold, detecting a specific-sized rectangular box, and automatically decreasing the threshold if it exists.

```
if find_square(barrier_mat, m, n) and current_threshold >= min_threshold:
    current_threshold -= 50  # Automatically decrease the threshold
    cv.setTrackbarPos("dist1", window_name, current_threshold)
    if current_threshold <= min_threshold:
        print("Turn Warning: Minimum threshold reached. Please turn to other
    directions")
        cv.setTrackbarPos("dist1", window_name, initial_threshold)  # Reset to the
     initial value
if current_threshold <= min_threshold:
    print("Turn Warning: Minimum threshold reached. Please turn to other
    directions")
    cv.setTrackbarPos("dist1", window_name, 1000)
```

4 Division

Zening Lu : Programming /Charts Producing

Yifan Wang: Programming

Chongjie Zhao: Reporting /Charts Producing

Changwei Lv: Video Cliping

Wen Wen: Charts Producing