Image Detection Application

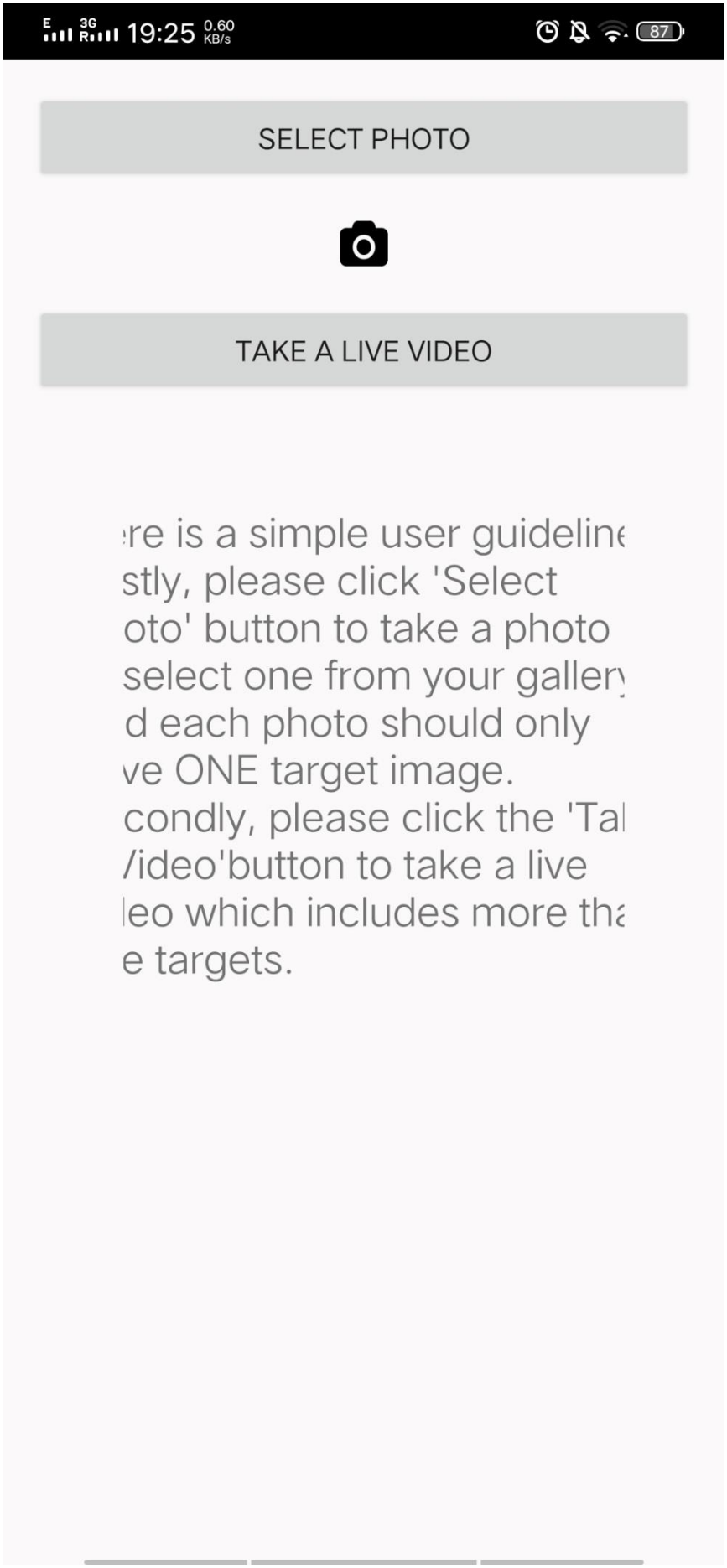
Water

Molan Zhang, Qingli Zeng, Jiaxin Fan

We built an image detection application named Take Image. As the picture blow, this is an android application which has **5 main function:**

1)Take photo and take video

while opening this application, there are two buttons here. Users can take a picture(original image) by pressing the first button (they can also choose the picture from their own albums). They can also be able to start a live video by pressing the second button.

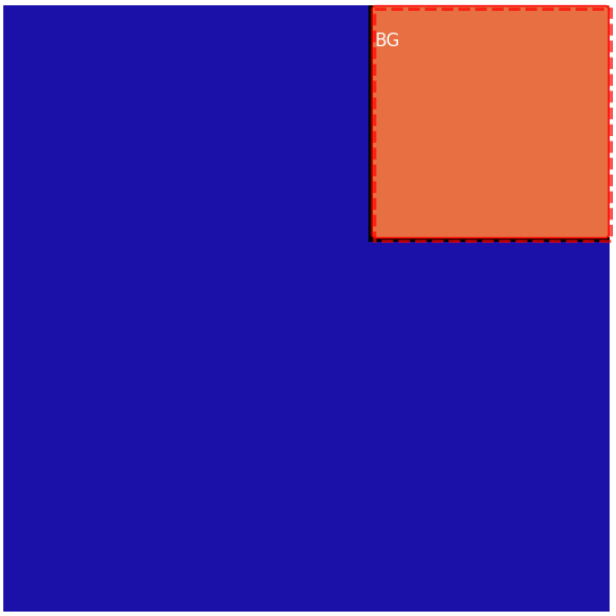

19:25
SELECT PHOTO
TAKE A LIVE VIDEO
re is a simple user guidelin
stly, please click 'Select
oto' button to take a photo
select one from your galler
d each photo should only
ve ONE target image.
condly, please click the 'Ta
/ideo'button to take a live
eo which includes more th
e targets.

2)Image detection

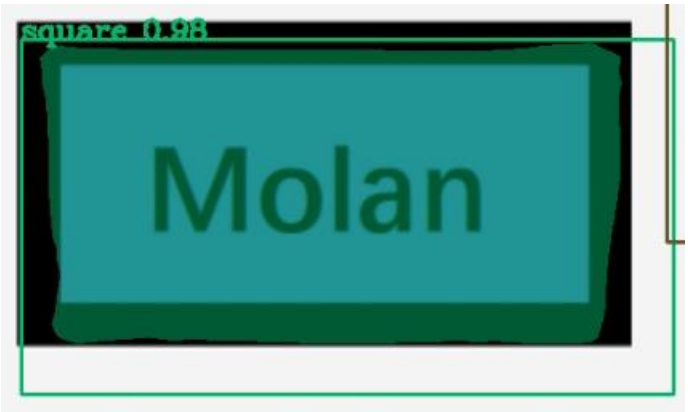
The image detection include picture detection and video detection, we use the deep learning model based on the Mask RCNN network to extract the objects we want from the image and video.And our model’s mAP is over 0.99 which has high accuracy.

There are the three images in the boxes.

This is the image we detect from the photo we took.



There are three images we detect from the live video.



The test shows that the accuracy is pretty high(almost 1.0), and the lost rate is 0.0276

```
Loading weights from  /home/molan/Desktop/mask-rcnn/Mask_RCNN-master/mask_rcnn_s
hapes.h5
30
original_image           shape: (128, 128, 3)         min:    0.00000  max:  223
.00000  uint8
image_meta               shape: (14,)                 min:    0.00000  max:  128
.00000  int64
gt_class_id              shape: (1,)                  min:    1.00000  max:    1
.00000  int32
gt_bbox                  shape: (1, 4)                min:    0.00000  max:  128
.00000  int32
gt_mask                  shape: (128, 128, 1)         min:    0.00000  max:    1
.00000  bool
1
Processing 1 images
image                    shape: (128, 128, 3)         min:    0.00000  max:  223
.00000  uint8
molded_images            shape: (1, 128, 128, 3)      min: -123.70000  max:  106
.20000  float64
image_metas              shape: (1, 14)               min:    0.00000  max:  128
.00000  int64
anchors                  shape: (1, 4092, 4)          min:   -0.71267  max:    1
.20874  float32
1
[[  0  79  47 128]]
mAP:  1.0
```

```
 87/100 [=========================>....] - ETA: 1:57 - loss: 0.3675 - rpn_class_
loss: 0.0089 - rpn_bbox_loss: 0.2708 - mrcnn_class_loss: 0.0357 - mrcnn_bbox_los
 88/100 [=========================>....] - ETA: 1:48 - loss: 0.3678 - rpn_class_
loss: 0.0089 - rpn_bbox_loss: 0.2714 - mrcnn_class_loss: 0.0355 - mrcnn_bbox_los
 89/100 [=========================>....] - ETA: 1:39 - loss: 0.3679 - rpn_class_
loss: 0.0088 - rpn_bbox_loss: 0.2719 - mrcnn_class_loss: 0.0354 - mrcnn_bbox_los
 90/100 [=========================>...] - ETA: 1:30 - loss: 0.3681 - rpn_class_
loss: 0.0088 - rpn_bbox_loss: 0.2721 - mrcnn_class_loss: 0.0354 - mrcnn_bbox_los
 91/100 [=========================>...] - ETA: 1:21 - loss: 0.3692 - rpn_class_
loss: 0.0088 - rpn_bbox_loss: 0.2734 - mrcnn_class_loss: 0.0354 - mrcnn_bbox_los
 92/100 [=========================>...] - ETA: 1:12 - loss: 0.3687 - rpn_class_
loss: 0.0088 - rpn_bbox_loss: 0.2731 - mrcnn_class_loss: 0.0351 - mrcnn_bbox_los
 93/100 [=========================>...] - ETA: 1:03 - loss: 0.3680 - rpn_class_
loss: 0.0088 - rpn_bbox_loss: 0.2727 - mrcnn_class_loss: 0.0350 - mrcnn_bbox_los
 94/100 [=========================>..] - ETA: 54s - loss: 0.3671 - rpn_class_l
oss: 0.0088 - rpn_bbox_loss: 0.2721 - mrcnn_class_loss: 0.0348 - mrcnn_bbox_loss
 95/100 [=========================>..] - ETA: 45s - loss: 0.3660 - rpn_class_l
oss: 0.0088 - rpn_bbox_loss: 0.2714 - mrcnn_class_loss: 0.0347 - mrcnn_bbox_loss
 96/100 [=========================>..] - ETA: 36s - loss: 0.3647 - rpn_class_l
oss: 0.0088 - rpn_bbox_loss: 0.2705 - mrcnn_class_loss: 0.0344 - mrcnn_bbox_loss
 97/100 [=========================>.] - ETA: 27s - loss: 0.3655 - rpn_class_l
oss: 0.0088 - rpn_bbox_loss: 0.2715 - mrcnn_class_loss: 0.0343 - mrcnn_bbox_loss
 98/100 [=========================>.] - ETA: 18s - loss: 0.3663 - rpn_class_l
oss: 0.0088 - rpn_bbox_loss: 0.2723 - mrcnn_class_loss: 0.0343 - mrcnn_bbox_loss
 99/100 [=========================>.] - ETA: 9s - loss: 0.3669 - rpn_class_lo
ss: 0.0088 - rpn_bbox_loss: 0.2730 - mrcnn_class_loss: 0.0343 - mrcnn_bbox_loss:
100/100 [==============================] - 908s 9s/step - loss: 0.3671 - rpn_cla
ss_loss: 0.0088 - rpn_bbox_loss: 0.2735 - mrcnn_class_loss: 0.0341 - mrcnn_bbox_
loss: 0.0196 - mrcnn_mask_loss: 0.0311 - val_loss: 0.3975 - val_rpn_class_loss:
0.0080 - val_rpn_bbox_loss: 0.3153 - val_mrcnn_class_loss: 0.0316 - val_mrcnn_bb
ox_loss: 0.0150 - val_mrcnn_mask_loss: 0.0276
```

3)Image recognition

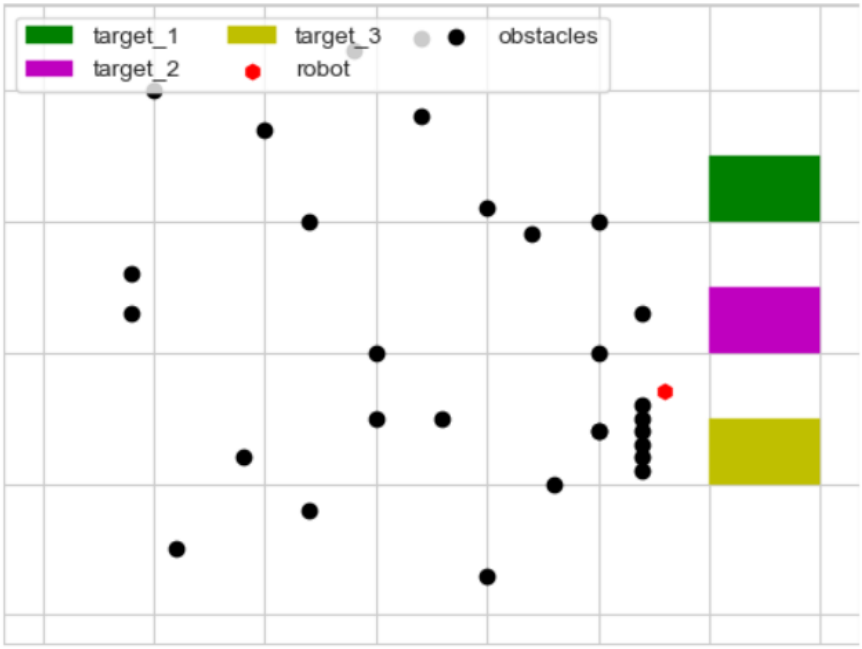
After detecting the images, our application will recognize the most similar image from the three video images to the original image. In this apart, We used the five algorithms that are currently relatively popular to calculate the similarity of the images. And output the most similar one.

We take the photo for the first picture, and the picture above is what the application returned.

4)provide directions for the robot

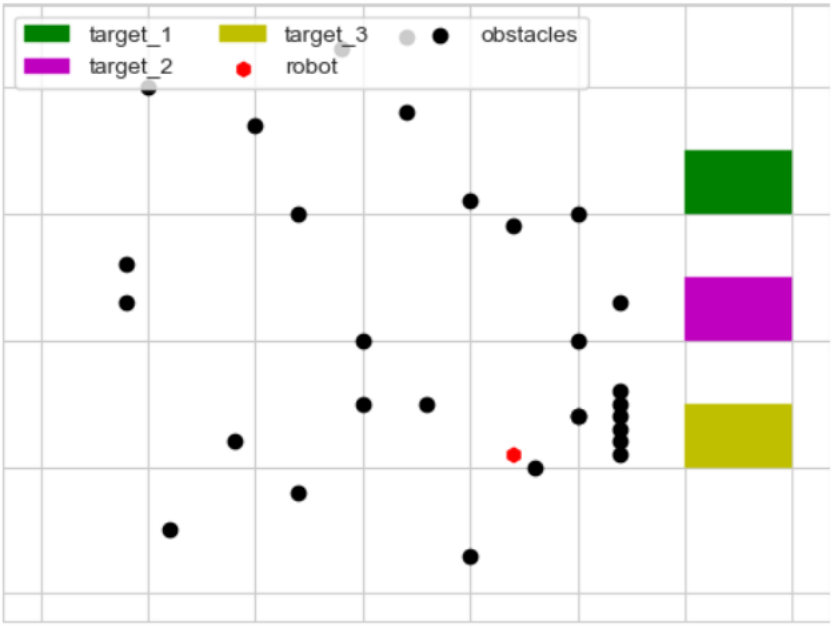
Once the most similar image is found, the app will provide a path to the image. we simulated a similar scene because we don't have a robotic environment. Our results are shown blow. This red point represents our robot, the black point represents the obstacle, the

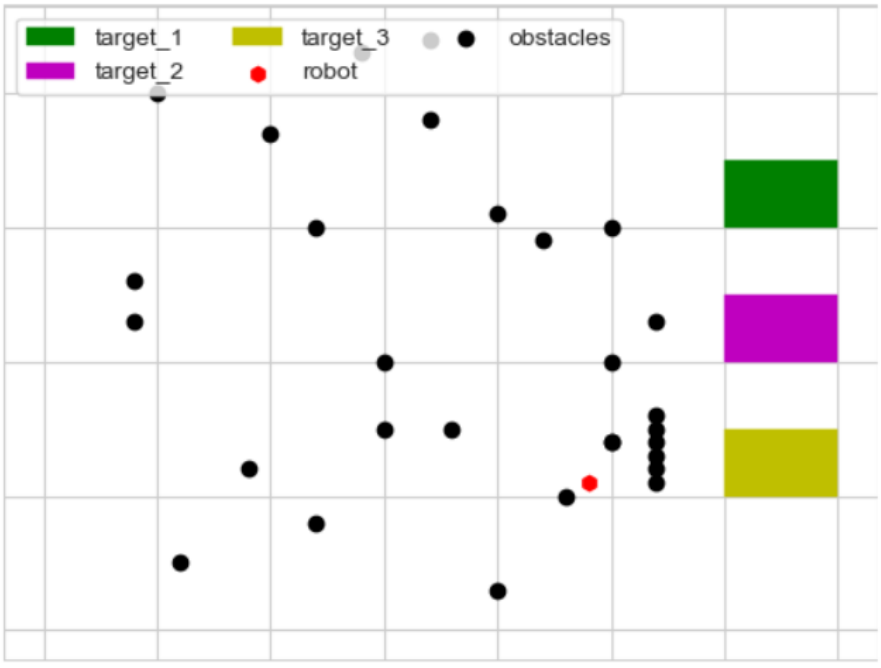
rectangle is the detected box. We will build a map in advance, and the robot will establish its own waypoint(own position) until reach the destination.

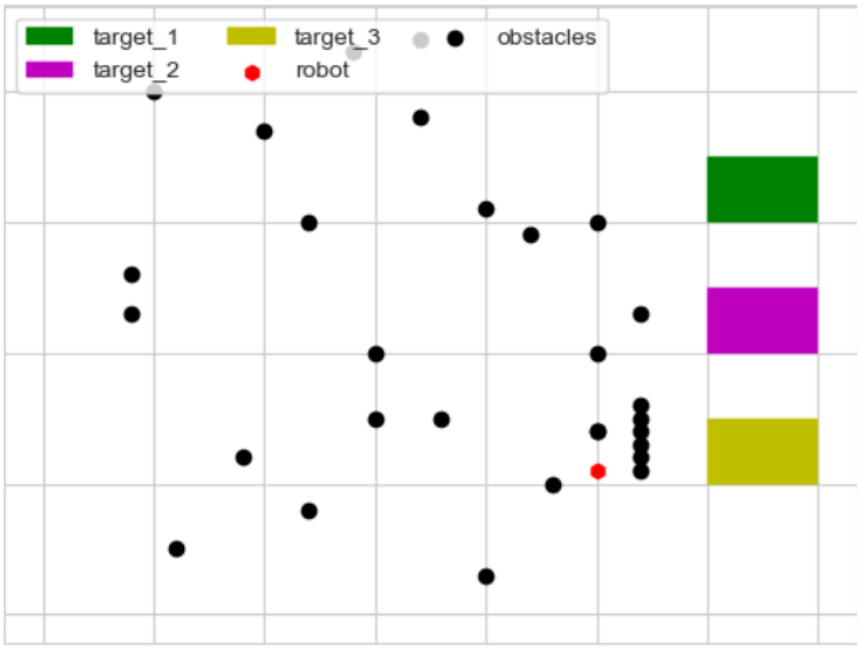


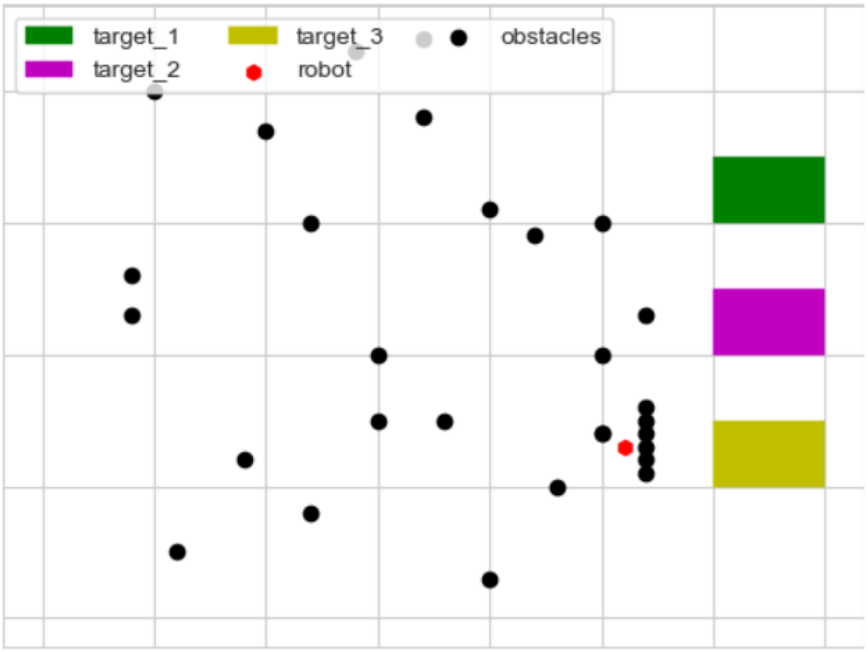
5)Obstacles detection

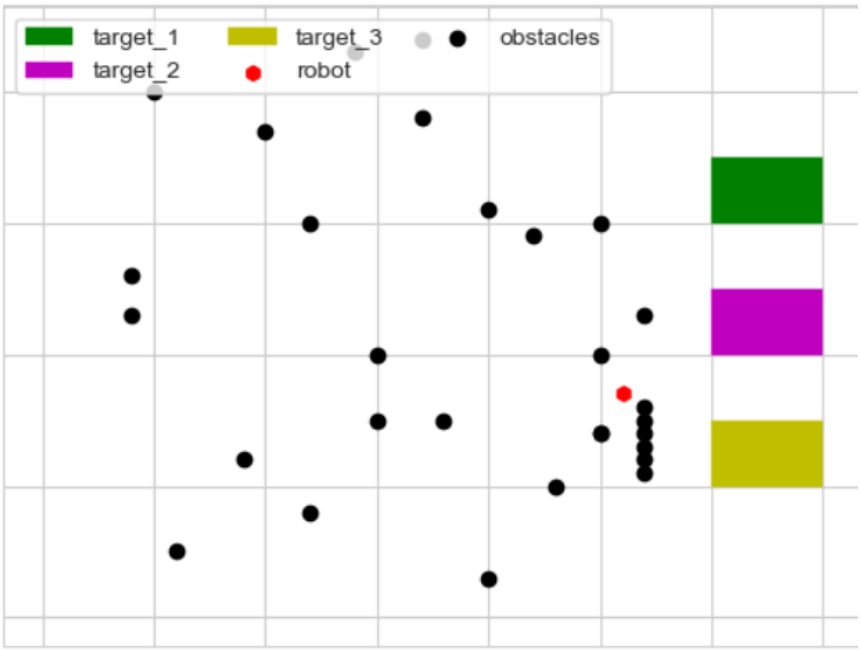
Our application can detect the obstacles while getting closer to the target box. Our results are shown as following GIF. In then reality, the Robot can detect obstacles by laser radar,Once our robot finds an obstacle in front, it will choose to move in parallel until there are no obstacles in front and then move on.

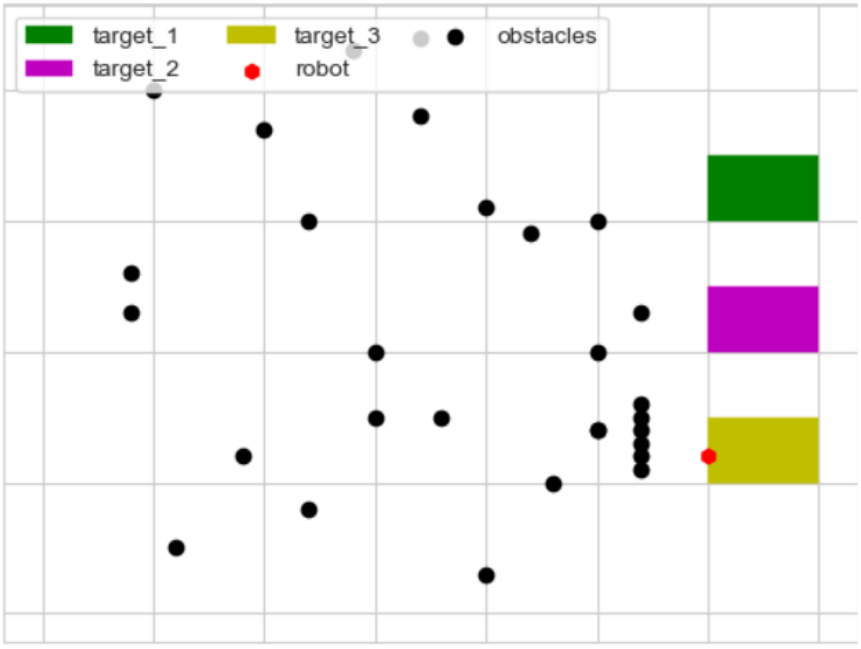

target_1
target_2
target_3
robot
obstacles


target_1
target_2
target_3
robot
obstacles


target_1
target_2
target_3
robot
obstacles


target_1
target_2
target_3
robot
obstacles





Installation and testing steps:

1.Install the require.txt file and configure the MaskRCNN environment.

2.Download pre-training weight

3.Run the Android apk observation

Innovation:

1.The use of MaskRCNN network for semantic segmentation and target detection reduces the problem of pixel-by-pixel feature extraction of CNN network, resulting in global information loss.

2. Using five image similarity calculation methods, the possibility of outputting the correct answer is improved.

3. Simulation realizes automatic obstacle avoidance function of robot

4. Implement the robot control code, but since there is no robot to experiment, you have to perform analog output.

5. The user can see the walking path of the robot

6. Provide a brief user guide and limit the user's wrong click button to ensure the user's successful use of the app.

7. Provide error information warning and information prompts to improve user experience

8.Image smoothing preprocessing is added based on the maskRCNN model, which reduces the possibility of the model being attacked.