6.4 Automatic avoid

Note: In the image operating system provided by Yahboom, the APP remote control process is enabled by default, in order to avoid multiple occupations of internal resources, causing some functions to fail to operate normally.

Before you running the code of this course, please follow the following to close the APP remote control process.

If you want to permanently close the function of the APP that starts automatically after booting, execute the following command:

```
sudo systemctl disable jetbotmini_start
```

If you want to permanently open the function of the APP that starts automatically after booting, execute the following command:

```
sudo systemctl enable jetbotmini_start
```

If you do not restart Jetbotmini to restart the APP function, execute the following command:

```
sudo systemctl stop jetbotmini_start
sudo systemctl start jetbotmini_start
```

In Section 4.4, we have already learned about TensorFlow. With the previous foundation, the following things are easy to understand. Let's get started:

First, we import opencv, tensorflow, and control display related libraries.

[1]:
```
import numpy as np
import cv2
import os,time
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_utils
import ipywidgets.widgets as widgets
from image_fun import bgr8_to_jpeg
```

Next, let us import the jetcham library for camera use.

[2]:
```
#from jetcam.usb_camera import USBCamera
from jetcam.csi_camera import CSICamera
from jetcam.utils import bgr8_to_jpeg

#camera = USBCamera(width=320, height=240, capture_fps=30)
camera = CSICamera(width=320, height=240, capture_fps=30)

camera.running = True
```

Then import the tensorflow object recognition-related library, create the camera display control, after the operation is completed, a frame of the camera will be displayed, and the real-time image will be displayed only when the following continuous cycle update is required.

```
image_widget = widgets.Image(format='jpg', width=320, height=240)
display(image_widget)
image_widget.value = bgr8_to_jpeg(camera.value)
```

```
# Init tf model

MODEL_NAME = 'ssdlite_mobilenet_v2_coco_2018_05_09' #fast
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')

NUM_CLASSES = 90
IMAGE_SIZE = (12, 8)
fileAlreadyExists = os.path.isfile(PATH_TO_CKPT)

if not fileAlreadyExists:
    print('Model does not exsist !')
    exit
```

```
# LOAD GRAPH
print('Loading...')
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)
print('Finish Load Graph..')
```

```
print(type(category_index))
```

```
print("dict['Name']: ", category_index[1]['name'])
```

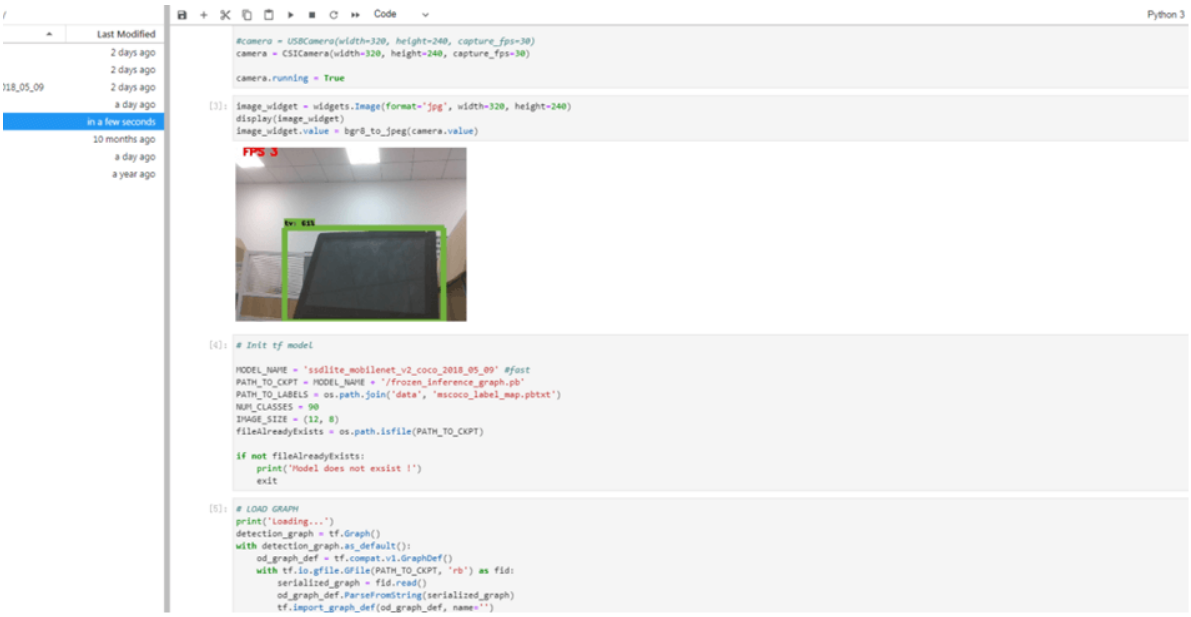
We have two ways to call the camera recognition. The first is to call the camera value through a while loop for recognition. Because it is an infinite loop, it needs to be stopped by the stop button to exit.

```
  # Main
  t_start = time.time()
  fps = 0

  with detection_graph.as_default():
      with tf.compat.v1.Session(graph=detection_graph) as sess:
          while True:
              frame = camera.value
             # ret, frame = cap.read()
  #            frame = cv2.flip(frame, -1) # Flip camera vertically
  #            frame = cv2.resize(frame,(320,240))
              ###############
              image_np_expanded = np.expand_dims(frame, axis=0)
              image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
              detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
              detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
              detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')
              num_detections = detection_graph.get_tensor_by_name('num_detections:0')
  #            print('Running detection..')
              (boxes, scores, classes, num) = sess.run(
                  [detection_boxes, detection_scores, detection_classes, num_detections],
                  feed_dict={image_tensor: image_np_expanded})
  #            print('Done.  Visualizing..')
              vis_utils.visualize_boxes_and_labels_on_image_array(
                      frame,
                      np.squeeze(boxes),
                      np.squeeze(classes).astype(np.int32),
                      np.squeeze(scores),
                      category_index,
                      use_normalized_coordinates=True,
                      line_thickness=8)

              for i in range(0, 10):
                  if scores[0][i] >= 0.5:
                      print(category_index[int(classes[0][i])]['name'])
              ###############
              fps = fps + 1
              mfps = fps / (time.time() - t_start)
              cv2.putText(frame, "FPS " + str(int(mfps)), (10,10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)
              image_widget.value = bgr8_to_jpeg(frame)
```

The second method is to create a value change function for the camera. When the camera value changes, the function will be called for processing. This method is called after the camera value changes, which is more lagging than the above-mentioned loop mode screen.

```
detection_graph.as_default()
sess = tf.compat.v1.Session(graph=detection_graph)

t_start = time.time()
fps = 0

def update_image(change):
    global fps
    global sess
    frame = change['new']
    image_np_expanded = np.expand_dims(frame, axis=0)
    image_np_expanded = np.expand_dims(frame, axis=0)
    image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
    detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
    detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
    detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')
    num_detections = detection_graph.get_tensor_by_name('num_detections:0')

#    print('Running detection..')
    (boxes, scores, classes, num) = sess.run(
        [detection_boxes, detection_scores, detection_classes, num_detections],
        feed_dict={image_tensor: image_np_expanded})

#    print('Done.  Visualizing..')
    vis_utils.visualize_boxes_and_labels_on_image_array(
        frame,
        np.squeeze(boxes),
        np.squeeze(classes).astype(np.int32),
        np.squeeze(scores),
        category_index,
        use_normalized_coordinates=True,
        line_thickness=8)
    for i in range(0, 10):
        if scores[0][i] >= 0.5:
            print(category_index[int(classes[0][i])]['name'])
    ###############
    fps = fps + 1
    mfps = fps / (time.time() - t_start)
    cv2.putText(frame, "FPS " + str(int(mfps)), (10,10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)
    image_widget.value = bgr8_to_jpeg(frame)
```

Only one of the above two methods can be used at the same time. After the successful call, the recognition effect can be seen from the camera interface at the beginning:



The corresponding complete source code is located: