2.1.3 Mobile tracking

Code path

/home/pi/Yahboom_Project/3.AI_Visual_course/03.Mobile_tracking.ipynb

```
# Low-level driving method
from Raspblock import Raspblock
robot = Raspblock()
```

```
#bgr8 to jpeg format
import enum
import cv2

def bgr8_to_jpeg(value, quality=75):
    return bytes(cv2.imencode('.jpg', value)[1])
```

```
# Display camera components
import cv2
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display
import time
# Thread function operation library
import threading
import inspect
import ctypes

image_widget = widgets.Image(format='jpeg', width=300, height=300)
display(image_widget)
```

```
def _async_raise(tid, exctype):
    """raises the exception, performs cleanup if needed"""
    tid = ctypes.c_long(tid)
    if not inspect.isclass(exctype):
        exctype = type(exctype)
    res = ctypes.pythonapi.PyThreadState_SetAsyncExc(tid,
ctypes.py_object(exctype))
    if res == 0:
        raise ValueError("invalid thread id")
    elif res != 1:
        # """if it returns a number greater than one, you're in trouble,
        # and you should call it again with exc=NULL to revert the effect"""
        ctypes.pythonapi.PyThreadState_SetAsyncExc(tid, None)

def stop_thread(thread):
    _async_raise(thread.ident, SystemExit)
```

```
image = cv2.VideoCapture(0)
```

```
image.set(3, 320)
image.set(4, 240)
image.set(5, 120)    # set frame rate
# fourcc = cv2.VideoWriter_fourcc(*"MPEG")
image.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter.fourcc('M', 'J', 'P', 'G'))
# image.set(cv2.CAP_PROP_BRIGHTNESS, 40) # set brightness -64 - 64    0.0
# image.set(cv2.CAP_PROP_CONTRAST, 50) # set contrast -64 - 64    2.0
# image.set(cv2.CAP_PROP_EXPOSURE, 156) #set exposure 1.0 - 5000    156.0
ret, frame = image.read()
image_widget.value = bgr8_to_jpeg(frame)
```

```
global color_x, color_y, color_radius
color_x = color_y = color_radius = 0
global target_valuex
target_valuex = 1500
global target_valuey
target_valuey = 1500
global g_mode
g_mode = 0
```

```
# Create an array to store HSV color gamut color classification data
import numpy as np
global color_lower
color_lower = np.array([156, 43, 46])
global color_upperv
color_upper = np.array([180, 255, 255])
```

```
# Create a PID control instance
import PID
xservo_pid = PID.PositionalPID(1.1, 0.2, 0.8)
yservo_pid = PID.PositionalPID(0.8, 0.2, 0.8)
```

```
# Color selection button configuration
Redbutton = widgets.Button(
        value=False,
        description='Red',
        disabled=False,
        button_style='', # 'success', 'info', 'warning', 'danger' or ''
        tooltip='Description',
        icon='uncheck' )
Greenbutton = widgets.Button(
        value=False,
        description='Green',
        disabled=False,
        button_style='', # 'success', 'info', 'warning', 'danger' or ''
        tooltip='Description',
        icon='uncheck' )
Bluebutton = widgets.Button(
        value=False,
        description='Blue',
        disabled=False,
        button_style='', # 'success', 'info', 'warning', 'danger' or ''
        tooltip='Description',
        icon='uncheck' )
Yellowbutton = widgets.Button(
        value=False,
        description='Yellow',
        disabled=False,
        button_style='', # 'success', 'info', 'warning', 'danger' or ''
        tooltip='Description',
        icon='uncheck' )
Orangebutton = widgets.Button(
        value=False,
        description='Orange',
        disabled=False,
        button_style='', # 'success', 'info', 'warning', 'danger' or ''
        tooltip='Description',
        icon='uncheck' )
Closebutton = widgets.Button(
        value=False,
        description='Close',
        disabled=False,
        button_style='', # 'success', 'info', 'warning', 'danger' or ''
        tooltip='Description',
        icon='uncheck' )
output = widgets.Output()
display(Redbutton, Greenbutton, Bluebutton, Yellowbutton, Orangebutton, Closebutton, output)

def ALL_Uncheck():
        Redbutton.icon = 'uncheck'
        Greenbutton.icon = 'uncheck'
        Bluebutton.icon = 'uncheck'
        Yellowbutton.icon = 'uncheck'
        Orangebutton.icon = 'uncheck'

def on_Redbutton_clicked(b):
        global color_lower, color_upper, g_mode
        global target_valuex, target_valuey
        ALL_Uncheck()
        b.icon = 'check'
        color_lower = np.array([0, 43, 46])
```

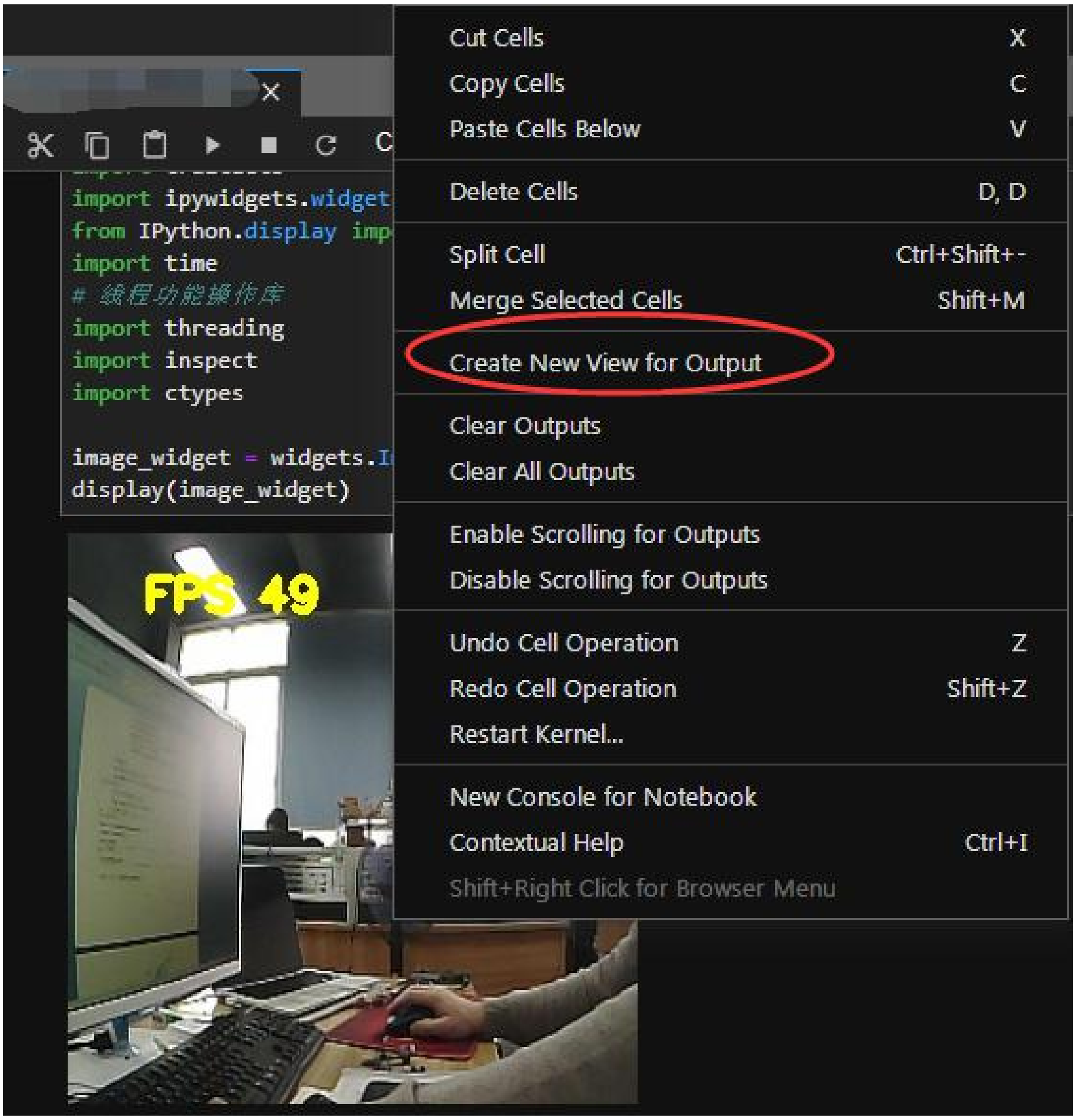
```
    color_upper = np.array([10, 255, 255])
    target_valuex = target_valuey = 2048
    robot.Servo_control(1000,1500)
    g_mode = 1
    with output:
        print(g_mode)
        print(color_lower)
        print("RedButton clicked.")

def on_Greenbutton_clicked(b):
    global color_lower, color_upper, g_mode
    global target_valuex, target_valuey
    ALL_Uncheck()
    b.icon = 'check'
    color_lower = np.array([35, 43, 46])
    color_upper = np.array([77, 255, 255])
    target_valuex = target_valuey = 2048
    robot.Servo_control(1500,1500)
    g_mode = 1
    with output:
        print("GreenButton clicked.")

def on_Bluebutton_clicked(b):
    global color_lower, color_upper, g_mode
    global target_valuex, target_valuey
    ALL_Uncheck()
    b.icon = 'check'
    color_lower=np.array([100, 43, 46])
    color_upper = np.array([124, 255, 255])
    target_valuex = target_valuey = 2048
    robot.Servo_control(1500,1500)
    g_mode = 1
    with output:
        print("Bluebutton clicked.")

def on_Yellowbutton_clicked(b):
    global color_lower, color_upper, g_mode
    global target_valuex, target_valuey
    ALL_Uncheck()
    b.icon = 'check'
    color_lower = np.array([26, 43, 46])
    color_upper = np.array([34, 255, 255])
    target_valuex = target_valuey = 150
    robot.Servo_control(1500,1500)
    g_mode = 1
    with output:
        print("Yellowbutton clicked.")

def on_Orangebutton_clicked(b):
    global color_lower, color_upper, g_mode
    global target_valuex, target_valuey
    ALL_Uncheck()
    b.icon = 'check'
    color_lower = np.array([11, 43, 46])
    color_upper = np.array([25, 255, 255])
    target_valuex = target_valuey = 2048
    robot.Servo_control(1500,1500)
    g_mode = 1
    with output:
        print("Orangebutton clicked.")

def on_Closebutton_clicked(b):
    global g_mode

    ALL_Uncheck()
    g_mode = 0
    with output:
        print("CloseButton clicked.")

Redbutton.on_click(on_Redbutton_clicked)
Greenbutton.on_click(on_Greenbutton_clicked)
Bluebutton.on_click(on_Bluebutton_clicked)
Yellowbutton.on_click(on_Yellowbutton_clicked)
Orangebutton.on_click(on_Orangebutton_clicked)
Closebutton.on_click(on_Closebutton_clicked)
```

```
# PTZ motion main process
def Color_track():
    global color_lower, color_upper, g_mode
    global target_valuex, target_valuey
    t_start = time.time()
    fps = 0
    while True:
        ret, frame = image.read()
        frame = cv2.resize(frame, (300, 300))
        frame_ = cv2.GaussianBlur(frame,(5,5),0)
        hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(hsv,color_lower,color_upper)
        mask = cv2.erode(mask,None,iterations=2)
        mask = cv2.dilate(mask,None,iterations=2)
        mask = cv2.GaussianBlur(mask,(3,3),0)
        cnts = cv2.findContours(mask.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)[-2]
        if g_mode == 1: # Button switch
            if len(cnts) > 0:
                cnt = max (cnts, key = cv2.contourArea)
                (color_x,color_y),color_radius = cv2.minEnclosingCircle(cnt)
                if color_radius > 10:
                    # Mark the detected color
                    cv2.circle(frame,(int(color_x),int(color_y)),int(color_radius),(255,0,255),2)
                    # Proportion-Integration-Differentiation
                    xservo_pid.SystemOutput = color_x
                    xservo_pid.SetStepSignal(150)
                    xservo_pid.SetInertiaTime(0.01, 0.1)
                    target_valuex = int(1500+xservo_pid.SystemOutput)
                    # Input Y axis direction parameter PID control input
                    yservo_pid.SystemOutput = color_y
                    yservo_pid.SetStepSignal(150)
                    yservo_pid.SetInertiaTime(0.01, 0.1)
                    target_valuey = int(1500 - yservo_pid.SystemOutput)
                    # PTZ ratate PID Rotate the gimbal to the PID adjustment and calibration position
                    robot.Servo_control(target_valuex,target_valuey)
        fps = fps + 1
        mfps = fps / (time.time() - t_start)
        cv2.putText(frame, "FPS " + str(int(mfps)), (40,40), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,255,255), 3)
        # Display returned data in real time
        image_widget.value = bgr8_to_jpeg(frame)
        # print(g_mode)
```

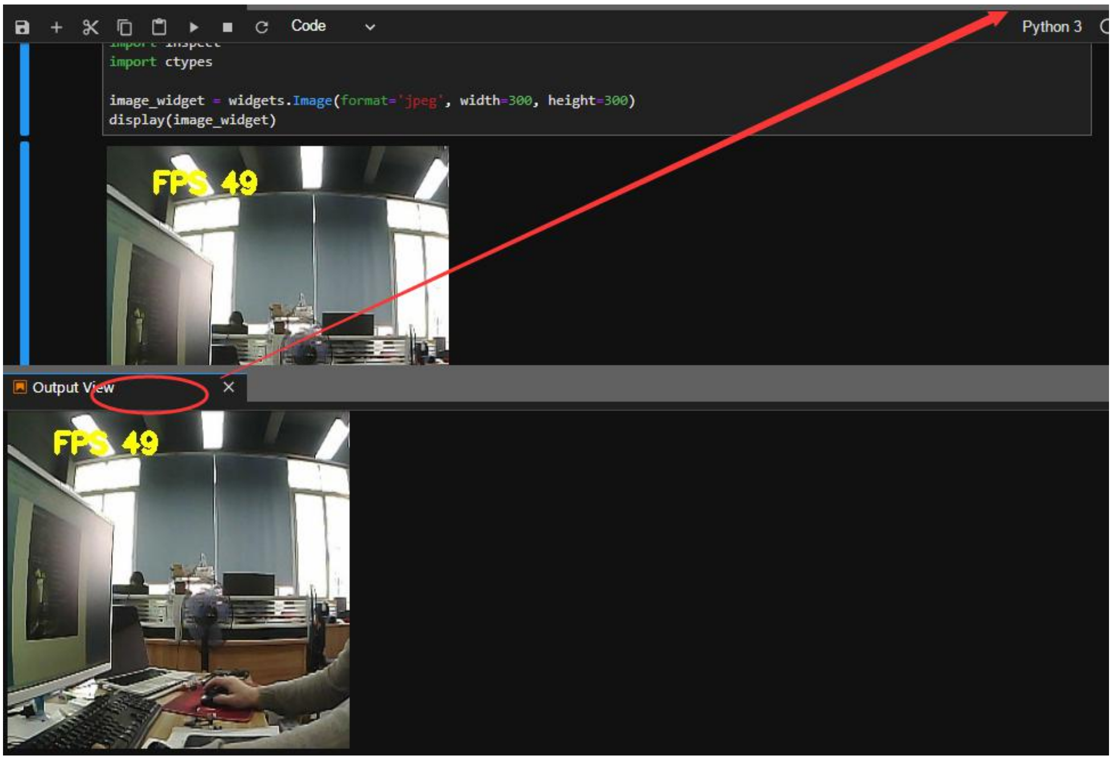
```
## Start process
thread1 = threading.Thread(target=Color_track)
thread1.setDaemon(True)
thread1.start()
```

```
## Close process
stop_thread(thread1)
```

Tip: We can put the components in other Windows as shown below.
Click “right button” of mouse -->【Create New View for Output】.



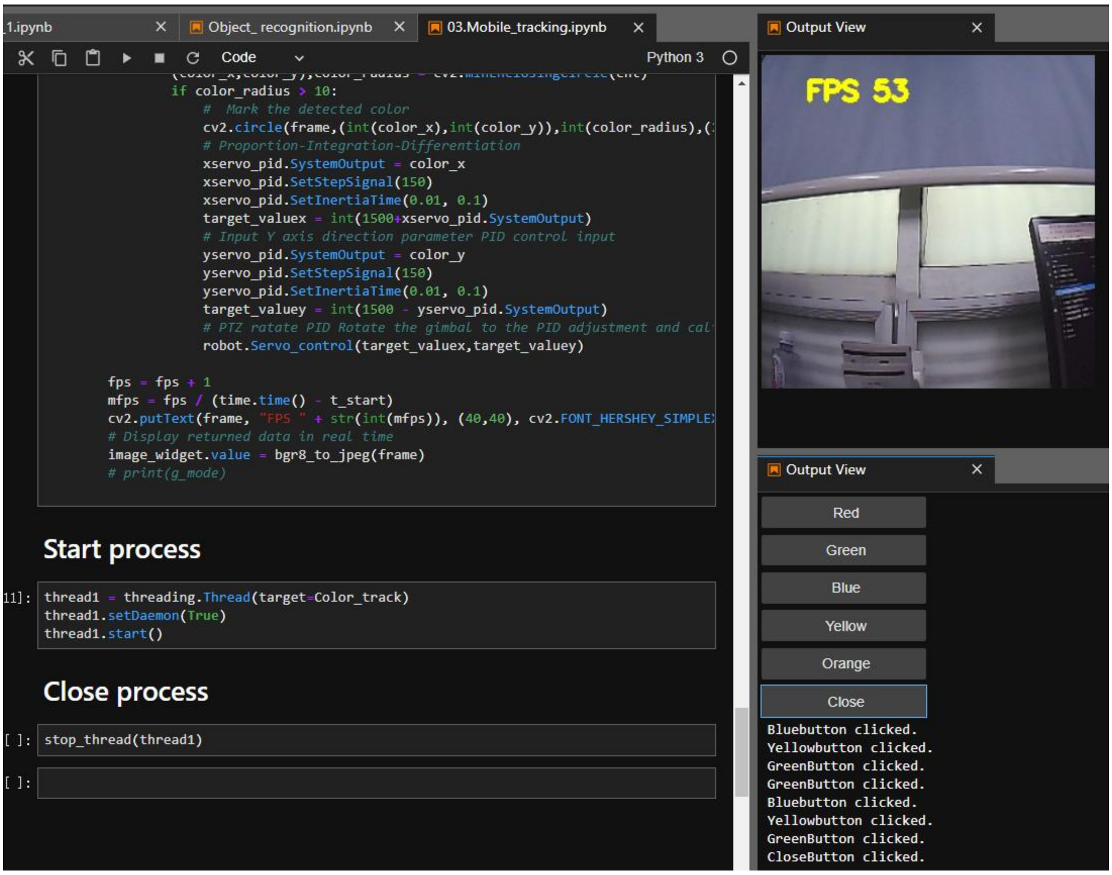
```
import ipywidgets.widget
from IPython.display imp
import time
# 线程功能操作库
import threading
import inspect
import ctypes

image_widget = widgets.I
display(image_widget)
```



```
import inspect
import ctypes

image_widget = widgets.Image(format='jpeg', width=300, height=300)
display(image_widget)
```

We also pulled button interface to the right together, as shown below.

We can choose different color, camera will tracking it.



```
        (color_x,color_y),color_radius = cv2.minEnclosingCircle(cnt)
        if color_radius > 10:
            #  Mark the detected color
            cv2.circle(frame,(int(color_x),int(color_y)),int(color_radius),(
            # Proportion-Integration-Differentiation
            xservo_pid.SystemOutput = color_x
            xservo_pid.SetStepSignal(150)
            xservo_pid.SetInertiaTime(0.01, 0.1)
            target_valuex = int(1500+xservo_pid.SystemOutput)
            # Input Y axis direction parameter PID control input
            yservo_pid.SystemOutput = color_y
            yservo_pid.SetStepSignal(150)
            yservo_pid.SetInertiaTime(0.01, 0.1)
            target_valuey = int(1500 - yservo_pid.SystemOutput)
            # PTZ ratate PID Rotate the gimbal to the PID adjustment and cal
            robot.Servo_control(target_valuex,target_valuey)

    fps = fps + 1
    mfps = fps / (time.time() - t_start)
    cv2.putText(frame, "FPS " + str(int(mfps)), (40,40), cv2.FONT_HERSHEY_SIMPLEX
    # Display returned data in real time
    image_widget.value = bgr8_to_jpeg(frame)
    # print(g_mode)
```

Start process

```
11]: thread1 = threading.Thread(target=Color_track)
     thread1.setDaemon(True)
     thread1.start()
```

Close process

```
[ ]: stop_thread(thread1)

[ ]:
```

```
Bluebutton clicked.
Yellowbutton clicked.
GreenButton clicked.
GreenButton clicked.
Bluebutton clicked.
Yellowbutton clicked.
GreenButton clicked.
CloseButton clicked.
```