Скрипт Inventorization

Скрипт inventorization.py является примером, того как можно провести инвентаризацию складских помещений по QR-кодам, содержащихся в этих каждой из ячеек.

Скрипт позволяет квадрокоптеру в автономном режиме провести инвентаризацию складского ряда с QR-кодами. т. е. набора предметов, расположенных горизонтально друг за другом на одинаковом расстоянии. Определённые предметы при этом могут отсутствовать, в таком случае ячейка будет считаться пустой. Полётная траектория определяется в скрипте заранее.

Разбор скрипта

1. Импортируем необходимые библиотеки и определяем их назначение:

   - **Pioneer_sdk** – библиотека для управления квадрокоптером;

   Описание библиотеки Pioneer_sdk - https://pioneer-doc.readthedocs.io/ru/master/programming/python/pioneer-sdk-methods.html;

   - **NumPy** – библиотека для работы с массивами данных;

   Описание библиотеки NumPy - https://numpy.org/doc/stable/;

   - **Cv2** – библиотека машинного зрения;

   Описание библиотеки NumPy - https://numpy.org/doc/stable/;

   - **More_itertools** – библиотека с инструментами для итераций массивов;

   Описание библиотеки more_itertools — https://more-itertools.readthedocs.io/en/stable/api.html;

   - **Time** – библиотека для создания искусственных задержек;

   Описание библиотеки time - https://docs.python.org/3/library/time.html.

```
from pioneer_sdk import Pioneer, Camera
import numpy as np
import cv2
from more_itertools import locate
import time
```

2. Создаём ряд констант и массивы:

   - STORAGE_WIDTH = 3 — кол-во ячеек в ряду.
   - *X_INC = float(0.6)* — расстояние между ячейками в ряду.
   - *HEIGHT = float(0.8)* — высота ряда относительно пола.
   - *names = []* — массив для сохранения имён хранящихся предметов.
   - *quantities = []* — массив для сохранения количеств хранящихся предметов.

```
STORAGE_WIDTH = 3
X_INC = float(0.6)
HEIGHT = float(0.8)
names = []
quantities = []
```

3. Далее используем конструкцию **if __name__ == „__main__“:**, которая является точкой входа в программу. Всё, что идёт до этого условия, выполнятся всегда: и при вызове в качестве модуля и при вызове, как исполняемый файл.

Подробное описание данной конструкции:
https://docs.python.org/3/library/__main__.html

```
if __name__ == '__main__':
```

4. Создаём экземпляр класса Pioneer и Camera, чтобы начать работать с квадрокоптером и его камерой непосредственно.

```
pioneer_mini = Pioneer()
camera = Camera()
```

С понятием, что такое класс и его экземпляры можно ознакомиться по ссылке
https://docs.python.org/3/tutorial/classes.html

5. Запускаем моторы и взлетаем:

```
pioneer_mini.arm()
pioneer_mini.takeoff()
```

Создадим функция для инвентаризации ряда. Для этого заранее объявим её в функции **if __name__ == „__main__“:** после создания экземпляров классов и передадим туда созданные экземпляры. Выше объявляем одноимённую функцию где и продолжим работу.

```
inventorize(pioneer_mini, camera)
```

Выше, вне функции **if __name__ == „__main__“:**

```
def find_item(drone):
```

6. Объявляем следующие переменные и экземпляр класса:

- counter = 1 — счётчик ячеек (счёт начинается от первой ячейки).
- *command_x = float(0)* — переменная, отвечающая за перемещение по оси x.
- detector = cv2.QRCodeDetector() — экземпляр класса для поиска и сканирования QR-кодов.
- *new_point = True* — флаг, сигнализирующий о том, что ячейка просканирована.

```
counter = 1
command_x = float(0)
detector = cv2.QRCodeDetector()
new_point = True
```

Также создадим бесконечный цикл, а в нём пропишем **try:** Эта структура будет улавливать ошибки модуля OpenCV в процессе сканирования и обрабатывать их согласно структуре **except:**, в нашем случае — игнорировать ошибку и продолжить работу скрипта. Без этих структур программа бы предварительно завершилась и квадрокоптер мог бы остаться в воздухе:

```
while True:
    try:
        # основное тело цикла...
    except cv2.error:
        continue
```

7. В переменную **camera_frame** передаём изображение от квадрокоптера путём обращения к методу **get_cv_frame()** класса Camera. Сделаем проверку на наличие кадра проверив сумму массива изображения. Если кадр проверку не прошёл, цикл перезапустим с новой итерации. Так же выведем изображение на экран:

```
camera_frame = cam.get_cv_frame()
if np.sum(camera_frame) == 0:
   continue
cv2.imshow('QR Reading', camera_frame)  # QR Reading — название окна
```

Если дрон просканировал ячейку и готов лететь к следующей, даём ему эту команду. Также она срабатывает при первой итерации и позволяет квадрокоптеру подлететь к первой ячейке для начала инвентаризации:

```
if new_point:
   new_point = False
   drone.go_to_local_point(x=command_x, y=0, z=HEIGHT, yaw=0)
```

Создадим проверку на достижение квадрокоптером заданной точки. Метод **point_reached()** класса Pioneer вернёт True только в случае, если текущие координаты Пионера совпадают с заданными ему методом **go_to_local_point(x, y, z, yaw)**, где для инвентаризации ряда нам важны только x (меняется от ячейки к ячейке) и z (не меняется в случае одного ряда предметов):

```
if drone.point_reached():
```

8. Так как при подлёте к точке квадрокоптер какое-то время стабилизируется и его камера может не охватить интересующий QR-код, нужно сканировать метку не единожды, а в течении какого-то времени. Один из возможных способов достичь этого — создать таймер и цикл для сканирования изображения в течении 2.5 секунд. Минус реализации заключается в том, что во время подобного сканирования Пионер не показывает картинку с камеры и не воспринимает команды.

Сперва инициализируем таймер, т. е. запоминаем время, когда началось выполнения сканирования:

```
timer = time.time()
```

В теле цикла для сканирования мы снова получаем кадр и обесцвечиваем его с помощью метода OpenCV **cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)**. Далее пытаемся найти и прочитать QR-код, записав полученные данные в переменную string. В условии указываем, что если string содержит хоть какой-то текст ИЛИ если текущее время превышает время в timer на более чем 2.5 секунды, то мы покидаем цикл:

```
while True:
   try:
      gray = cv2.cvtColor(cam.get_cv_frame(), cv2.COLOR_BGR2GRAY)
      string, _, _ = detector.detectAndDecode(gray)
      if ((string is not None) and (string != '')) or (time.time() - timer > float(2.5)):
         break
   except:
      continue
```

9. Если при сканировании предмет не был найден, то ячейка объявляется пустой. Так же нам не интересны ячейки не содержащие два фрагмента — Название и Количество предметов на полке (пример правильного формата - Camera 8). Добавляем в массивы names и quantities характерные значения None и 0 для пустых ячеек:

```
if (string is None) or (string == '') or (len(string.split(' ')) != 2):
    print("[INFO] На данной полке предмет не найден")
    names.append("None")
    quantities.append(int(0))
```

10. Если при сканировании предмет был найден и текст в ней в нужном формате, то добавляем в массивы names и quantities значения первого фрагмента текста (Название) и второго фрагмента текста (Количество):

```
else:
    text = string.split(' ')
    print("[INFO] Найден предмет", text[0], "в количестве", text[1])
    names.append(text[0])
    quantities.append(int(text[1]))
```

11. Далее необходимо предусмотреть условие окончания инвентаризации. Если счётчик counter, объявленный до тела цикла, достиг числа ячеек в ряду, то выводим все собранные данные и подаём квадрокоптеру команду вернуться в изначальную позицию и в бесконечном цикле ждать, пока он не достигнет этой позиции, после чего приземлиться и завершить выполнение функции:

```
if counter == STORAGE_WIDTH:
    print("[INFO] Инвентаризация завершена:")
    print(names)
    print(quantities)
    drone.go_to_local_point(x=0, y=0, z=HEIGHT, yaw=0)
    while True:
        if drone.point_reached():
            drone.land()
            return None
```

Если ячейка не была последней, то переменная x для следующей точки будет отличаться на величину шага от предыдущей, т. е. квадрокоптер пролетит направо на расстояния шага X_INC. После этого выставляем флаг new_point и повышаем счётчик на 1, сигнализируя о том, что сканирование завершено:

```
else:
    command_x += X_INC
new_point = True
counter += 1
```

12. После структуры except, объявленной ранее, создадим проверку на нажатие Escape. Это позволит посадить квадрокоптер и завершить программу предварительно:

```
key_ip = cv2.waitKey(1)
if key_ip == 27:
    print('[INFO] ESC нажат, программа завершается')
    drone.land()
    time.sleep(5)
    cv2.destroyAllWindows()
    exit(0)
```

На этом этапе функция инвентаризации окончена.

13. Создадим функция для поиска нужного предмета по запросу. Для этого заранее объявим её в функции **if __name__ == „__main__“:** после вызова функции инвентаризации:

```
find_item(pioneer_mini)
```

Выше, вне функции **if __name__ == „__main__“:**

```
def find_item(drone):
```

14. Объявляем переменную item_found = False, которая будет использована в дальнейшем для выхода из двойного цикла:

```
item_found = False
```

Далее в бесконечном цикле будем опрашивать пользователя о том, какой именно предмет ему нужно найти. Встроенный метод input() присваивает переменной значение, введённое в терминал пользователем:

```
while True:
   item = input("Какой предмет нужно найти? ")
```

15. Перебираем имена, содержащиеся в массиве names и сверяем их с запросом пользователя. Соответственно, если имя находится в этом массиве, опрашиваем пользователя необходимое ему количество данного предмета. Так как предметов с одинаковым названием может быть несколько, нужно проверить количества всех таких предметов из массива quantities. Следующая строка **indexes = list(locate(names, lambda x: x == item)):** присвоит переменной indexes значения всех ячеек, где содержится запрошенный пользователем предмет, в виде списка:

```
if item in names:
 quantity = int(input("В каком количестве? "))
 indexes = list(locate(names, lambda x: x == item))
```

Далее сравниваем значения количеств предметов в найденных ячейках с требуемым и в случае соответствия требованиям по количеству запишем номер ячейки в переменную index и просигнализируем о завершении поиска с помощью флага item_found, что выведет нас из общего цикла. В случаях, если условия не выполнены, выведем в терминал соответствующую информацию:

```
   for i in range(len(indexes)):
      if quantity <= quantities[indexes[i]]:
         index = indexes[i]
         item_found = True
         print("Ячейка", index+1, "содержит", item, "в нужном количестве")
         break
   if item_found:
      break
   print(item, "в нужном количестве не найден, повторите попытку")
else:
   print("Такого предмета нет на складе, повторите попытку")
```

16. Получив номер ячейки с необходимым предметом, отправляем квадрокоптеру команду взлететь и направиться к этой точке, аналогично предыдущим полётам ожидая флага **point_reached()**. Здесь значение x — номер ячейки умноженный на шаг, что совпадает с реальным положением предмета:

```
drone.arm()
drone.takeoff()
command_x = X_INC*index
drone.go_to_local_point(x=command_x, y=0, z=HEIGHT, yaw=0)
while True:
   if drone.point_reached():
      break
```

17. Посветим светодиодом напротив этой ячейки, после чего функция нахождения предмета будет окончена:

```
drone.led_control(r=0, g=255, b=0)
time.sleep(3)
drone.led_control(r=0, g=0, b=0)
```

18. В функции **if __name__ == „__main__“:** после вызова всех пользовательских функций садим квадрокоптер и завершаем программу:

```
pioneer_mini.land()
time.sleep(5)
cv2.destroyAllWindows()
exit(0)
```

Возможные ошибки:

1. Данный скрипт рассчитан на то, что единожды квадрокоптер будет сканировать только один QR-код, и при попытке просканировать сразу несколько QR-кодов возможно появление ошибок. Чтобы избежать этого, QR-коды рекомендуется ставить на значительном расстоянии друг от друга и/или ставить Пионер как можно ближе к сетке.

2. Перемещение квадрокоптера по системе координат осуществляется по системе оптического потока, которая является сравнительно неточной. На качество достоверности перемещения Пионера на заданные координаты влияют прежде всего такие факторы, как освещение и индивидуальные дефекты моторов и пропеллеров.

На практике это значит, что при отправке команды **drone.go_to_local_point(x=1, y=0, z=1, yaw=0)** со стартовой позиции (0, 0, 1) квадрокоптер может переместиться вправо на 1,05 метров, а его реальная координата по y может измениться в ходе полёта, несмотря на то, что команды на это не было.

Для того, чтобы бороться с этим, используйте максимально хорошо освещённое пространство со светлыми полами и/или учитывайте небольшие отклонения по дистанции при проведении инвентаризации. При исправных параметрах складского ряда, заданных вначале скрипта, на успешную инвентаризацию может понадобиться несколько попыток.

3. Из-за плохого освещения или некачественного для оптического потока покрытия пола квадрокоптер может подняться на высоту, отличающейся от заданной и выровняться по ходу своего полетного задания. Чтобы избежать этого, можно изменять параметр начальной

высоты полёта в параметрах автопилота. Для этого подключаем Пионер Мини к Pioneer Station и меняем параметр Flight_com_takeoffAlt во вкладке Flight на новое значение, например 1,5 или 1,6.