Мастер-класс «Код полёта»

Уважаемые участники мастер-класса!

Помните про закон Мерфи (Anything that can go wrong will go wrong)!

Программирование квадрокоптеров, так же, как и большая часть робототехники — это компромисс между простотой программы и учетом всех факторов, влияющих на ее работу. Зачастую простая программа может дать лучший результат, нежели чем сложная.

В рамках сегодняшнего мастер-класса мы будем использовать квадрокоптеры Tello, которые являются результатом взаимодействия компании DJI и Intel. Главным элементом дрона является 14-ядерный процессор Intel, который позволяет квадрокоптеру не только эффективно отслеживать свое положение в пространстве, обрабатывая показания встроенных датчиков, но еще и позволяет получить доступ к видеопотоку фронтальной камеры и управлять квадрокоптером посредством отправки ему UDP команд описанных в SDK.

Более подробно ознакомиться с различными моделями серии Tello вы можете на следующих сайтах:

www.ryzerobotics.com
www.dji.com/ru/robomaster-tt

Программировать квадрокоптер мы будем на языке Python, так как он прост для начального изучения и позволяет даже людям ранее не изучавшим языки программирования довольно быстро разобраться в основах языка. Несмотря на то, что программирование посредством передачи UDP команд не вызывает особых затруднений, для экономии времени при написании однотипных команд мы воспользуемся библиотекой DJITelloPy (github.com/damiafuentes/DJITelloPy).

Запрограммируем полёт по квадрату:

Первым делом подключим библиотеку и импортируем из нее класс Tello для работы с квадрокоптером

```
from djitellopy import Tello
```

Далее создадим экземпляр класса Tello и дадим ему имя tello

```
tello = Tello()
```

Теперь подключимся к квадрокоптеру,

```
tello.connect()
```

посмотрим, насколько заряжена у него батарея.

```
print(tello.get_battery())
```

И наконец, взлетим!

```
tello.takeoff()
```

После взлета пролетим вперед 100 см,

```
tello.move_forward(100)
```

пролетим боком вправо 100 см,

```
tello.move_right(100)
```

пролетим назад 100 см,

```
tello.move_back(100)
```

пролетим боком налево 100 см,

```
tello.move_left(100)
```

и, с чувством выполненного долга, посадим коптер на место!

```
tello.land()
```

Сохраните получившийся код в IDLE в папке flight_code_273 под названием square.py и проверьте его, подключившись к вашему квадрокоптеру по Wi-Fi и запустив выполнение кода нажатием в IDLE кнопки F5 (Run->Run module).

Если квадрокоптер не полетел или полетел, но не так, как вы ожидали, то вспомните закон Мерфи и позовите на помощь ведущего мастер-класс или его помощников, они помогут понять, верно ли работает ваш код.

Обратите внимание на то, что каждый новый этап имеет смысл начинать с того, что вы сохранили рабочий код предыдущего этапа и создали новый файл, в котором будет содержаться новый код.

Пришло время дать коптеру возможность вращаться вокруг своей оси:

Воспользуйтесь методом rotate_clockwise(90) в сочетании с методом move_forward(100) для того, чтобы заставить квадрокоптер лететь по квадрату осуществляя пролет вперед и поворот на 90 градусов по часовой стрелке.

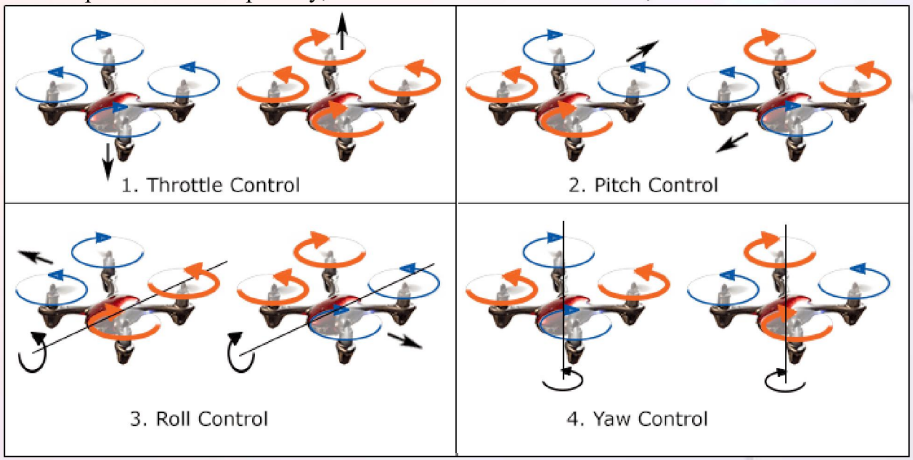
Если вы имеете представление о циклах, воспользуйтесь конструкцией for i in range(4): для того, чтобы сократить количество команд обеспечивающих полёт по квадрату с восьми до трёх. Не забудьте, что команды, входящие в тело цикла в Python, должны иметь отступ в одну табуляцию или 4 пробела относительно уровня отступа управляющей конструкции.

Если же нет, то просто повторите вызовы методов move_forward(100) и rotate_clockwise(90) четыре раза.

Сохраните полученный результат!

Получилось? Теперь вы готовы освоить метод send_rc_control(roll, pitch, throttle, yaw) дающий еще больше контроля над квадрокоптером. Что делает этот метод? – он позволяет передавать целочисленные значения управляющего воздействия на каждый из четырёх компонент, обуславливающих перемещение квадрокоптера в пространстве.

Посмотрите картинку, чтобы понять, какие компоненты за что ответственны



или просто прочтите ниже, если вам лень вникать в разноцветные стрелочки :-)

roll – смещение вправо(+)/влево(-), pitch – смещение вперед(+)/назад(-), throttle – смещение вверх(+)/вниз(-), yaw – поворот по часовой(+)/против часовой(-) стрелке

Квадрокоптер, получив команду send_rc_control, начинает её выполнение немедленно и продолжает её выполнять пока ему не поступит новая команда, либо пока не истечет время ожидания (после истечения времени ожидания, примерно 10 секунд, квадрокоптер осуществляет посадку, предполагая, что соединение с объектом управления потеряно). Как следствие, если вы просто выполните следующие 5 команд, ничего не произойдет и коптер просто повисев в воздухе положенное время сядет на пол.

```
tello.send_rc_control(0, 15, 0, 0)

tello.send_rc_control(15, 0, 0, 0)

tello.send_rc_control(0, -15, 0, 0)

tello.send_rc_control(-15, 0, 0, 0)

tello.send_rc_control(0, 0, 0, 0)
```

Как же быть? Нужно добавить задержку по времени:

Для этого в начале нашего кода подключим библиотеку time, взяв из нее метод sleep

```
from time import sleep
```

и теперь в основном коде между вызовами управляющих команд добавим двухсекундную задержку

```
sleep(2)
```

Квадрокоптер летает, но квадрат какой-то странный, кривой, косой и чем больше скорость и дальше квадрокоптер летит, тем сложнее назвать эту фигуру квадратом. В чём же дело? – дело в инерции! Квадрокоптер, как и бегущий человек не может мгновенно поменять направление своего движения. Не смотря на малый вес (всего 85 гр. с аккумулятором), некоторое время он продолжает двигаться в том же направлении, в котором он уже двигался, при этом смещаясь и в сторону нового управляющего воздействия. Давайте это поправим и хоть с пролетом по инерции пока мы ничего не сделаем, но мы дадим квадрокоптеру время на то, чтобы он остановился и лишь потом отдадим ему команду для движения в новом направлении.

Для этого после каждой временной задержки, обуславливающей время движения в заданном направлении, добавим еще 2 строки:

```
tello.send_rc_control(0, 0, 0, 0)
sleep(1)
```

Наша программа получилась довольно длинной, хотя мы всё еще летаем по квадрату. Перепроектируем её, или, как говорят программисты, подвергнем рефакторингу. Посмотрите на фрагмент кода и сравните с тем, что у вас получился на предыдущем шаге. Правда ведь стало лучше?

```
t = 3
power = 20
controls = [(0, power, 0, 0),
        (power, 0, 0, 0),
        (0, -power, 0, 0),
        (-power, 0, 0, 0)
]
stop = (0, 0, 0, 0)
for control in controls:
    sleep(t)
    tello.send_rc_control(*control)
    sleep(t)
    tello.send_rc_control(*stop)
```

А ещё теперь вы можете записывать в controls сколь угодно сложные комбинации управляющих воздействий для реализации самых разнообразных траекторий, и при этом, в остальной части кода вам не потребуется никаких изменений.

Вооружившись полученными знаниями реализуйте полет по равностороннему треугольнику. А когда у вас это получится, вы будете готовы к участию в районных и городских соревнованиях «Воздушные гонки» по восьмерке (останется лишь немного дописать код:-))

Упорства и удачи в достижении цели!