Tutorial_Decolar_Voar

November 1, 2018

1 Decolar 2m de altura e ficar parado

O objetivo dessa lição é decolar o drone a 2 metros de altura e ficar parado. Os objetivos desse documento é esclarecer o desenvolvedor acerca do funcionamento do modo **OFFBOARD** e as consequências *a posteriori* do não cumprimento dos requisitos do modo **OFFBOARD**.

1.1 Importando os módulos necessários

Importaremos aqui as bibliotecas padrão para desenvolvimento.

- A biblioteca **rospy** é padrão para chamarmos os métodos ROS Python.
- A biblioteca **mavros_msgs** é padrão para incorporarmos as mensagens MAVROS no desenvolvimento do node.
- O módulo **srv** é importado de mavros_msgs e é fundamental para incorporar os services da MAVROS a serem utilizados no node.

Nota: é fundamental notar que, se você precisar importar uma mensagem específica de uma classe maior, é recomendado adotar a estrutura:

from (classe_maior).msg **import** (tipo_de_mensagem)

```
In [ ]: import rospy

        import mavros_msgs

        from mavros_msgs import srv # importando services da mavros

        from geometry_msgs.msg import PoseStamped, TwistStamped

        from mavros_msgs.msg import State # importando classe State da Mavros
```

1.2 2. Criando os objetos

Nessa etapa nós criamos os objetos. No ROS, os tipos de mensagens são definidos como classes. Então, para nos inscrevermos em um certo tópico (depois), precisamos nos certificar que criamos estes objetos mensagens referente às classes (representadas pelo tipo de mensagem).

- **PoseStamped** é uma classe de mensagens pertencente à classe maior **geometry_msgs**.
- **State** é uma classe de mensagem que pertence à classe maior **mavros_msgs**.

Para checar informações referente ao tipo de mensagem, sugiro as documentações do ROS, MAVROS. Além disso, é possível rodar no terminal os seguintes comandos:

$ rosmsg list

$ rosmsg show {mensagem}

$ rostopic type (nome do tópico)

O primeiro retorna todas as mensagens que você tem instalado no seu ROS. O segundo retorna todas as informações de uma estrutura de mensagem que você passar como argumento. O terceiro te dá o tipo da mensagem que está sendo publicada em um determinado tópico.

Esses comandos são super úteis na hora que você vai criar aplicações que tenham mensagens que você não trabalhou ainda e desconhece a priori a suas estruturas de mensagem, bem como atributos.

```
In [ ]: goal_pose = PoseStamped()
        current_state = State()
```

1.3 Criando as funções de controle

- A função **set_position** é responsável por setar a as posições do drone modificando os atributos da mensagem goal_pose (da classe PoseStamped) e publicando a mensagem goal_pose no tópico criado na seção dos tópicos (4.)

Funções callback são muito importante em termos da estrutura dos nodes. As funções do tipo callback estão associados aos tópicos subscribers. A cada atualização do tópico em questão em que o node está inscrito, a tal função callback é chamada, promovendo uma atualização de informações.

- A função **state_callback** é do tipo **CALLBACK**. Ela atualiza o estado de vôo do drone (OFFBOARD, Loiter, etc) a cada atualização do tópico '/mavros/set_mode'.

```
In [ ]: def set_position(x, y, z):
            global goal_pose
            goal_pose.pose.position.z = z
            goal_pose.pose.position.y = y
            goal_pose.pose.position.x = x
            local_position_pub.publish(goal_pose)

        def state_callback(state_data):
            global current_state
            current_state = state_data
```

1.4 Inicialização do node e Setup dos tópicos

Nessa etapa do node, a gente vai instanciar os publishers, subscribers, services e clients. Nessa parte, é fundamental sabermos os tópicos em que desejamos inscrever nosso Node ou fazer com que ele publique! Por exemplo, no tópico **'/mavros/setpoint_position/local'** nós publicaremos as posições do drone. No tópico **'/mavros/state'** nós inscrevemos nosso node nesse tópico a fim de captar informações acerca do modo de vôo em que o drone está em um determinado momento, por exemplo.

```
In [ ]: rospy.init_node('Vel_Control_Node', anonymous=True)

        rate = rospy.Rate(20)  # publish at 20 Hz

        local_position_pub = rospy.Publisher(
                '/mavros/setpoint_position/local', PoseStamped, queue_size=10)

        state_status_subscribe = rospy.Subscriber(
                '/mavros/state', State, state_callback)

        arm = rospy.ServiceProxy(
                '/mavros/cmd/arming', mavros_msgs.srv.CommandBool)

        set_mode = rospy.ServiceProxy(
                '/mavros/set_mode', mavros_msgs.srv.SetMode)
```

1.5 Setup do modo de vôo OFFBOARD

Agora devemos nos atenta às peculiaridades do modo OFFBOARD. Vamos entender o que é o modo OFFBOARD. De acordo com o PX4 Developer Guide;

"Offboard mode is primarily used for controlling vehicle movement and attitude, and supports only a very limited set of MAVLink commands (more may be supported in future)."

- Para que o modo OFFBOARD funcione corretamente, precisamos mandar muitas posições para o drone, para o PX4 se certificar que há de fato uma fonte enviando comandos de posição para o drone. Para isso, faremos um laço de envio de posições. Note que sem esse laço o modo OFFBOARD não será corretamente settado.

```
In [ ]: for i in range(300):

            local_position_pub.publish(goal_pose)

            rate.sleep()

        rospy.loginfo("[ ROS] SETUP CONCLUIDO")
```

- Nesse loop principal, é onde as ações de vôo acontecem. O modo **OFFBOARD** requer que posições sejam passadas constantemente. Caso contrário, o PX4 modifica o modo vôo para **AUTO.LOITER** e depois para **AUTO.RTL**, encerrando o vôo.

- Temos um bloco de verificação nesse loop que é responsável por verificar constantemente o modo de vôo e se o drone está armado ou não. Caso o drone não estiver armado ou não estiver no modo offboard, então nós armamos o drone com o service **arm**. Além disso, modificamos o modo de vôo para offboard utilizando o service **set_mode**.

```
In [ ]: while not rospy.is_shutdown():

        # -------- VERIFICAO OFFBOARD E SE ESTA ARMADO -------------- #

        # --------- BLOCO DE VERIFICAÇÃO ------------------------------#
                if current_state.mode != "OFFBOARD" or not current_state.armed:

                        arm(True)

                        set_mode(custom_mode="OFFBOARD")

                print(str(current_state.mode))

                if current_state.armed == True:

                        rospy.loginfo("Drone armed")

                if current_state.mode == "OFFBOARD":

                    rospy.loginfo('OFFBOARD mode setted')
        # ------------------------------------------------------------- #

                set_position(0, 0, 2)

                rate.sleep()
```

Por fim, chamamos constantemente o metodo set_position(0,0,2) a fim de manter a posição do drone fixada em (0,0,2).

Chegamos ao fim da lição. Caso haja alguma dúvida, chame no zap!