API Description

1 UAV networking and communication

1.1 Network foundation and 6G network introduction

With the rapid development of the mobile Internet era and the deepening and expansion of information globalization, mobile communication technology is developing rapidly, profoundly influencing, and changing human's way of work and life. Reviewing The development of mobile communication technology, The First Generation (1G) mobile communication system was developed in the 1980s, using analog signals to transmit voice information and provide call services, which opened the prelude to the development and evolution of mobile communication technology. The Second Generation (2G) mobile communication was born in the 1990s, completing the transition from analog system to digital system. It can provide a wider coverage and more stable voice call service, and at the same time, it begins to expand the service dimension of support. At The beginning of the 21st century, The Third Generation (3G) mobile communication system came into being, thanks to the application of key technologies such as code division multiple access, can achieve more than 2Mbps data transmission rate, and expand the support for mobile multimedia data services, to achieve the combination of mobile communication and the Internet. In 2012, The International Telecommunication Union officially released the fourth Generation (The Fourt h Generation, 4G) mobile communication standard, marking the 4G mobile communication system into commercial deployment. The Orthogonal Frequency Division Multiplexing (OFDM) and Multiple-Input Multiple-Output (MIMO) are the key technologies in 4G mobile communication. The data transmission rate has been increased to more than 100Mbps, which has become an important infrastructure for carrying mobile Internet data services while achieving great commercial success. Driven by the diversified growth of demand for mobile communication services, The Fifth Generation (5G) mobile communication standard R15 and the first evolution standard R16 have been officially determined in 2019 and 2020 respectively, and the 5G communication system will fully enter the commercial stage in 2020. 5G Mobile communication technology based on supporting the enhanced Mobile Broadband (EBB) service, further expand the technology application scenario to Ultra-Reliable & Low Latency Communication, u RLLC) and Massive Machine Type Communication (MTC). In Massive MIMO (Massive MIMO), Millimeter wave (Millimeter wave, With the support of key technologies such as mm Wave, 5G mobile communication technology has achieved an all-round improvement in peak rate, user experience rate, spectral efficiency, mobility management, delay, connection density, network energy efficiency, regional service capacity and other performance, and has gradually penetrated vertical industries to provide large-traffic mobile broadband communication services. And is widely used in driverless, industrial control, Internet of Things, and other fields, to provide support for the Internet of everything.

With The large-scale commercial use of 5G mobile communication, the research and development of The Sixth Generation (6G) mobile communication technology is also officially launched in various countries. Finland, Japan, the United States and South Korea have started the research on 6G related technologies, as shown in the figure. China launched the national 6G special research plan in November 2019, and established the national 6G Technology research and development promotion working group and expert group, which marked the official launch of China's 6G mobile communication technology research and development work. 6G mobile communication technology will be committed to deepening mobile Internet based on 5G, constantly expanding the boundaries and scope of the Internet of everything,

and eventually realizing the intelligent connection of everything. From the point of view of peak transmission rate, 6G will reach the ultra-high transmission rate (huge traffic) of Tbps level; From the perspective of access density, 6G will reach 10 million/km-2 ~10 million/km-2 (huge connection); From the perspective of coverage, 6G will achieve global deep coverage (wide coverage); From the point of view of intelligence, 6G will eventually achieve high intelligence (high intelligence) by the initial intelligence of the 5G era. To meet the key characteristics of the 6G mobile communication system and improve the service experience of mobile communication users, the 6G wireless communication network will adopt a new paradigm and adopt new enabling technologies. The new paradigm focuses on the four major trends of full coverage, full spectrum, full application, and strong security, as shown in the figure. Among them, full coverage means that 6G will build an integrated "ubiquitous fusion information network" of air, earth, and sea through enabling technologies such as satellite communication, drone communication and maritime communication based on local coverage of 5G on land, to achieve deep network coverage on a global scale.

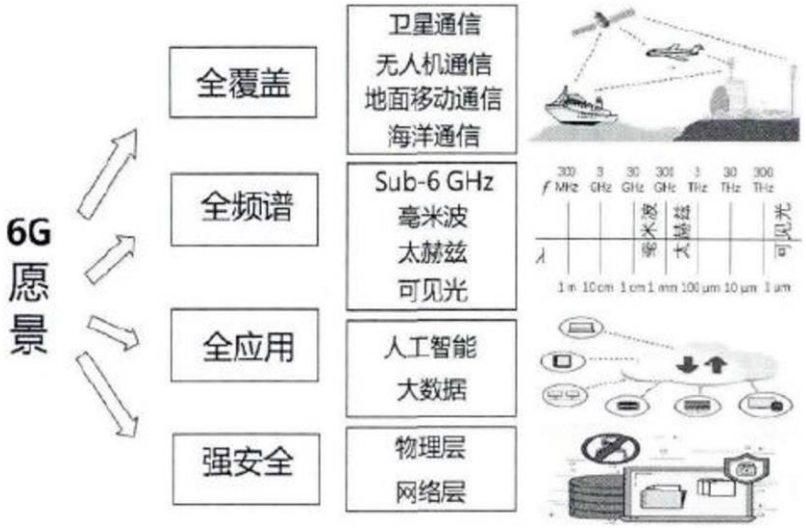

Figure 1

Uav communication is an important part of 6G wireless communication network, which can improve the efficiency of UAV mission execution and expand the coverage of wireless network. Compared with ground communication scenes, UAVs communication scenes have unique channel characteristics, which brings challenges to the design of UAVS communication systems.

1.2 Wireless channel modeling

In the field of wireless communication, the phenomenon that the amplitude of the received signal changes randomly due to the change of the channel is called channel fading. Signal in the channel generally through electromagnetic wave propagation, the way will produce fading, fading is an amount of power, or the amount of signal strength, when a signal in the process of transmission signal strength will continue to decrease.

What we have to do is how to model the channel fading, this is the channel modeling, can be divided into the following categories:

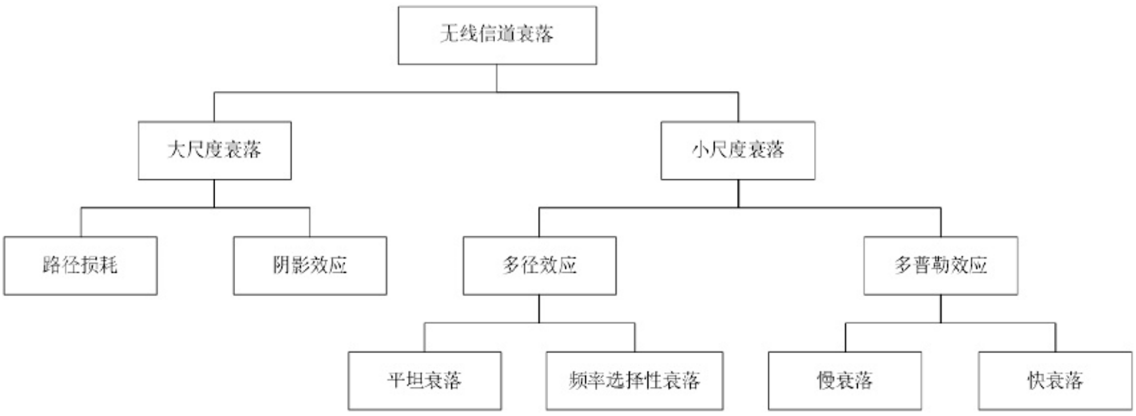

Figure 2 Division of channel fading

The causes of large-scale fading are path loss and shadow effect. The reasons of small-scale fading are multi-path effect and Doppler effect.

Large-scale fading refers to the phenomenon of fading when the mobile device moves through a relatively long distance (for example, a large distance in a cell); The so-called small-scale fading means that the mobile station moves a relatively short distance, and the cancellation or tensioning interference of multiple signal propagation paths will cause rapid fluctuations in signal levels. In this section, we only introduce large-scale fading, small-scale fading belongs to the micro level of the signal, involving the analysis of the signal field, there is no specific suitable modeling, the need for specific problem specific analysis.

According to Figure 2, large-scale fading is composed of path loss and shadow effect:

Path loss refers to the loss caused by the propagation of radio waves in space, generated by reflection, diffraction and scattering in the propagation, mainly determined by the distance between the mobile device and the antenna, and related to the environment of the electromagnetic wave transmission path, electromagnetic wave frequency, antenna, and other factors. For distance, the farther the distance, the greater the electromagnetic wave loss, and the closer the distance, the smaller the electromagnetic wave loss. For frequency, the greater the frequency, the greater the loss, and the smaller the frequency, the smaller the loss. For environmental factors, the path loss is relatively greater on rugged paths, such as mountains and hills, and relatively smaller on gentle paths, such as grasslands and fields. Path loss has good predictability. For a specific environment, it can usually be estimated by empirical formula. To obtain more accurate loss, it can be obtained by theoretical analysis and field measurement fitting.

Shadow effect refers to in the wireless communication system, the mobile station in the case of movement, due to the large buildings and other objects to the radio wave transmission path block and form a semi-blind area in the transmission receiving area, so as to form the electromagnetic field shadow, this with the changing position of the mobile station caused by the median amplitude of the receiving point fluctuations called shadow effect. The shadow effect is generally subject to the logarithmic plus distribution, and the fluctuations are large-scale fluctuations, so they are also predictable.

First, we analyze path loss. Before analyzing its principle, we need to understand the propagation behavior of electromagnetic wave: visible propagation and invisible propagation. Visible propagation refers to the propagation of electromagnetic waves in the channel without significant obstacles blocked, invisible propagation refers to the propagation of electromagnetic waves in the channel by a certain obstacle blocked. The judgment of electromagnetic wave propagation behavior can be made by the first Fresnel region for reference.

As early as the 16th century, the Dutch physicist Huygens proposed that each point of a wave front (surface source) can be regarded as the central wave source of a secondary spherical wave that produces spherical surface waves, and the wave speed and frequency of the secondary wave source are equal to the wave speed and frequency of the primary wave. Moreover, the position of the wavefront at any subsequent time is the envelope of all such seed waves. This shows that the state of the wave in any one part of the medium is determined by the fluctuations in all parts of the medium.

Then, in the 18th century, the French physicist Freel made a corresponding supplement to Huygens' principle, giving a mathematical form to calculate the fluctuation of a certain point, and at the same time, put forward a new concept of Fresnel region.
Fresnel principle believes that the radiation field at any point in space is all the secondary waves generated on any closed surface surrounding the wave source, and the superposition of the field generated at that point can be regarded as a secondary wave source at each point of any closed surface surrounding the wave source, and each emits spherical secondary waves.

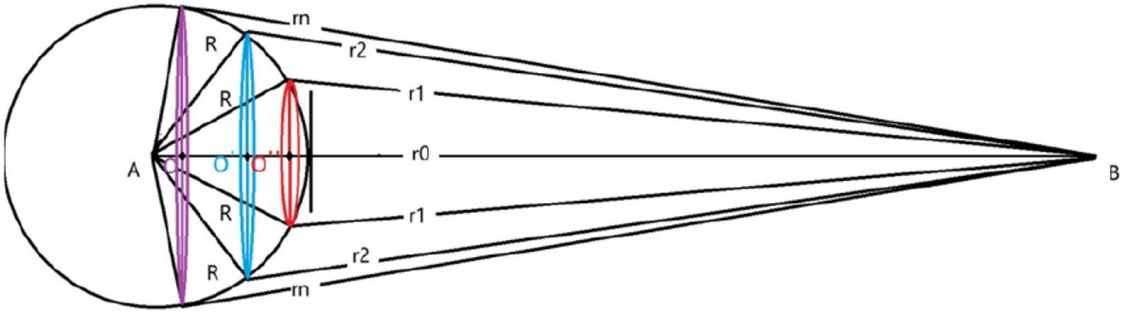

Figure 3 Fresnel channel and radius

Figure 3 depicts the Fresnel zone channel and radius. Points A and B are the sending and receiving electromagnetic wave sources respectively. It is assumed that there are O'', $O'$and O on the straight path of A and B. According to Fresnel half-wave band method, point A is taken as the center of the sphere as A sphere with radius R, so that A can reach B through the point on the sphere:

$$r0 + R = r0 + R$$

$$r1 + R = r0 + R + \frac{\lambda}{2}$$

$$r2 + R = r0 + R + 2 \times \frac{\lambda}{2}$$

.....

$$rn + R = r0 + R + n \times \frac{\lambda}{2}$$

Where λ is the wavelength of an electromagnetic wave. When n is 1, the red circle in the figure is o" the first Fresnel channel of the point whose radius is called o". Similarly, when n is 2, the blue circle in the figure is the second Fresnel channel of o 'and its radius is called the second Fresnel radius of o'. By this analogy, the purple circle represented by n is the NTH Fresnel channel of o, and its radius is the NTH Fresnel radius of o. In fact, in the field of communication, we are more concerned about the radius of the first Fresnel zone. The formula for calculating the radius of the first Fresnel zone of a certain point is as follows:

$$r = \sqrt{\frac{\lambda d_1 d_2}{d}} \quad (9\text{-}2)$$

$$d = d_1 + d_2 \quad (9\text{-}3)$$

Where d is the distance of two points A and B, $d_1$and d_2 are the distance from A to this point and the distance from B to this point respectively, as shown in the Figure 3, the red circle represents o "point of the first Fresnel zone channel, its radius of $r = \sqrt{\frac{\lambda|Ao''||Bo''|}{|Ao''|+|Bo''|}}$

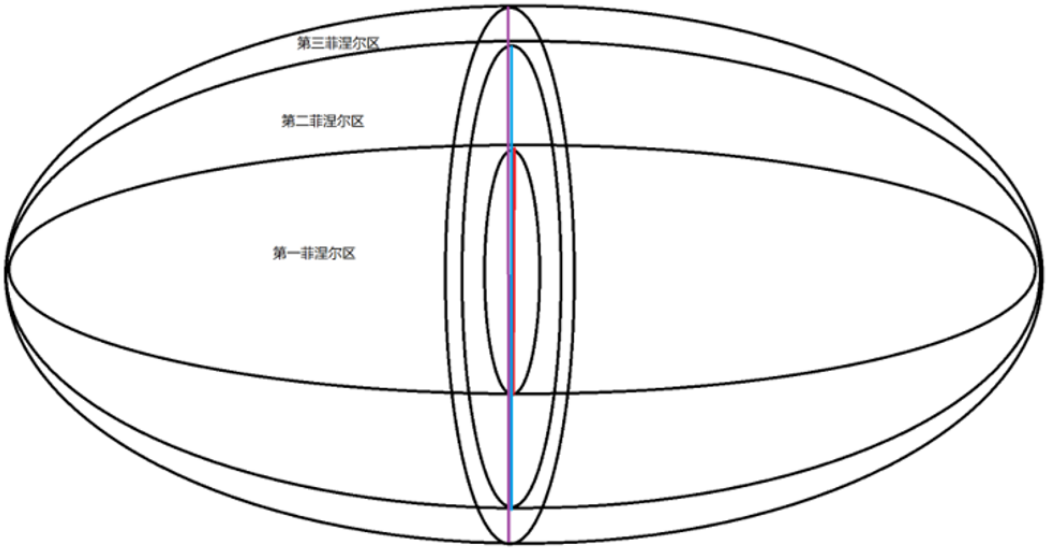

Figure 4 Division of Fresnel District

Figure 4 describes the division of Fresnel zone. From the profile, a Fresnel zone is composed of an infinite number of Fresnel channels. The following transformation is done in equation below:

$$rn + R = r0 + R + n * \frac{\lambda}{2} = |AB| + n * \frac{\lambda}{2}$$

Where, $|AB|$ is the straight-line distance between two points AB. When n is determined, $|AB| + n * \frac{\lambda}{2}$ in

the equation is a constant, and the right side of equation is a constant, so that the left side of equation is also a constant. According to the ellipse characteristic that the sum of distances between two focal points and any point is a constant, the trajectory composed of all points in this equation can be regarded as an ellipsoid with A and B as the focal points. When n is 1, the coverage area of the ellipsoid is the first Fresnel region, and by analogy, the NTH Fresnel region can be obtained.

According to Fresnel principle, the field strength of any point can be calculated, and the amplitude of the point is half of the amplitude generated by the first Fresnel half band, indicating that the influence of the first Fresnel region is very large. In judging whether the sight distance condition is satisfied, the first Fresnel region is mainly studied.

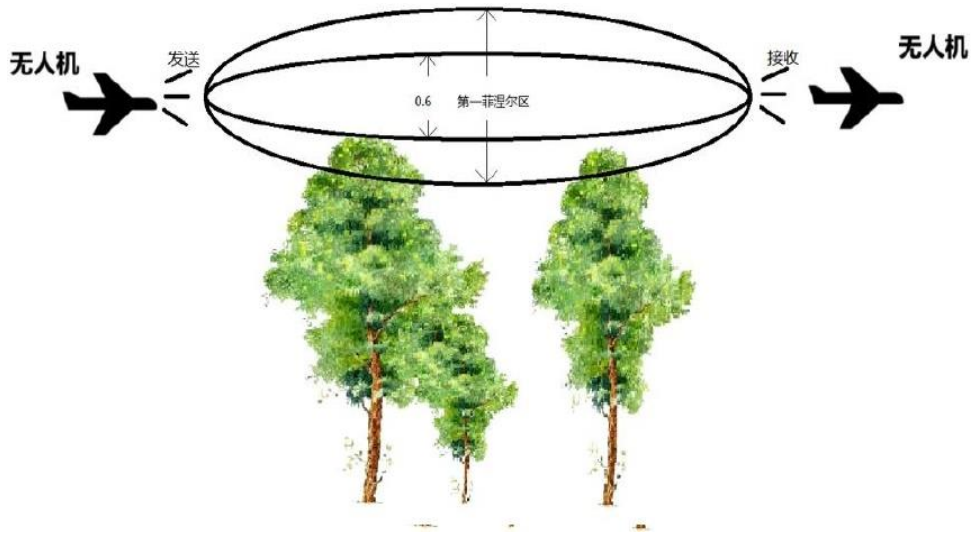

Figure 5 Schematic diagram of blocking Fresnel zone in communication

FIG. 5 demonstrates the influence of the first Fresnel zone (Fresnel zone in the figure) in communication. Under visual conditions, wireless signals propagate in a straight line between the sending end and the receiving end without obstruction, which requires objects that do not block radio waves in the first Fresnel zone, such Propagation is LOS Propagation; If the conditions are not met, the signal strength will decrease significantly, when the obstacle completely occlades the first Fresnel zone, this propagation is called non-line-of-sight propagation (NLOS Propagation). In general, we want the obstruction to block no more than 20% of the first Fresnel zone boundary.

In the case of determining that the propagation behavior is visible, in general, regardless of the communication in any situation, we can use the Free space propagation Model and the Log-distance Path Loss Model. Which is the most widely applicable model.

The free space propagation model is used to predict the field strength of the received signal when the line-of-sight path between the receiver and the transmitter is completely unblocked and belongs to the model of radio wave propagation with large scale path loss. Its formula is as follows:

$$P_r = P_t \frac{G_t G_r (\lambda)^2}{(4\pi)^2 d^2 L}$$

Where, Pt is the transmitting power, Gt is the transmitting antenna gain, Gr is the receiving antenna gain, λ is the wavelength (m), d is the linear distance between the transmitting antenna and the receiving antenna (m), and L is the system loss coefficient, including the total attenuation or loss of the system hardware system.

In the equation, logarithms of both sides of the equation are converted to db, and the following expression can be obtained:

$$10logP_r = 10logP_t + 10log_{10}\left(\frac{G_t G_r (\lambda)^2}{(4\pi)^2 d^2 L}\right)$$

$$P_r[db] = P_t[db] - L_{Free}[db]$$

L_{Free} is the path loss of the free space propagation model:

$$L_{Free}[db] = -10log_{10}\left(\frac{G_t G_r (\lambda)^2}{(4\pi)^2 d^{2L}}\right) = 10log_{10}\left(\frac{(4\pi)^2 d^2 L}{G_t G_r (\lambda)^2}\right)$$

L_{Free} is used to predict the power received by the receiving antenna under the line-of-sight condition.

The logarithmic distance path loss model amends the free space propagation model by introducing the path loss index which varies with the environment. The formula is as follows:

$$P_r[db] = P_t[db] - L_{LogDistance}[db]$$

$$L_{LogDistance}[db] = L_{Free}(d_0)[db] + 10\gamma log_{10}\left(\frac{d}{d_0}\right)$$

$L_{LogDistance}$ for distance path loss models of path loss, γ for path loss index, d, d_0 respectively transmitting antenna and receiving antenna linear distance (m) and the antenna far field reference distance (m). Due to the scattering phenomenon in the near field of the antenna, the channel will be more complicated. The general formula is only applicable to the case that the sending distance $d>$ d_0. d_0 for indoor generally 1m~10m, for outdoor generally 10m~100m or even larger.

So, let's talk about the modeling of shadow effect.

The shadow effect occurs when the receiving area is shaded by obstacles, and the electromagnetic wave will decrease the signal after passing through the shadow area. Shadow effect of wastage to logarithmic fall commonly, use $X\ \sigma\ [d\ b]$ said that assumes that averages $\bar{X}[d\ b]$, variance for σ, its probability density function is:

$$P(X_\sigma) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(X_\sigma - \bar{X})^2}{2\sigma^2}\right)$$

Therefore, in the case of obstructions, it is necessary to consider the influence of X_σ.

1.3 Commonly used UAV application layer protocols (DDS, MAVlink)

DDS (Data Distribution Service for Real-Time Systems) is a type of data-oriented middleware that provides a standard way for real-time systems to publish, discover, and subscribe to distributed data.

The core concept of DDS is the data publish/subscribe model, in which publishers publish data to topics, and subscribers receive data from those topics. DDS uses a technique called Service Discovery Protocol (SMP or SDP) to discover other DDS entities on the network (such as publishers, subscribers, etc.). This means that when a DDS node is started, it can automatically find topics and services available on the network. DDS can communicate using different transport layer protocols, including TCP/IP, UDP, shared memory, etc. This allows DDS to adapt to a variety of network environments and performance requirements. DDS provides a variety of Quality of service (QoS) policies, such as reliability, persistence, sequence assurance, etc. These policies can be adapted to the specific needs of the application to optimize data transfer. DDS supports defining data types and ensuring that these types are consistent across platforms. This allows DDS to handle complex, heterogeneous systems. One of the design goals of DDS is to provide real-time communication with low latency and high throughput.

As an application layer protocol for UAVS, DDS has the following advantages: Real-time performance: DDS is designed for real-time systems and can meet the low latency requirements of UAVs in flight control and other critical tasks. Loose coupling: DDS allows for loose coupling between publishers and subscribers, which means that components in the system can be developed and updated independently without affecting other parts. Flexibility: DDS supports dynamic network topologies, so nodes can be added or removed on the fly. This is important for drones that need the flexibility to adjust their mission configuration. Scalability: Since DDS is a topic-based data release/subscription model, it can easily handle more data streams and participants as the system grows in complexity and size. Quality of Service (QoS) policies: DDS allows users to select different QoS policies according to specific needs, such as reliability, persistence, sequence guarantee, etc., to optimize data transmission. Standardization: DDS is a standard developed by the OMG group, which helps to ensure interoperability and compatibility between different vendors. ROS integration: DDS has good integration with the Robot Operating System (ROS), especially in ROS 2, where DDS is widely adopted as the default communication framework. Defense and Industrial applications: Given that DDS was originally developed in the military sector and has been successfully used in multiple industries, its reliability and security are an important advantage for drone applications.

The limitations of the use of DDS are mainly in the following areas. DDS has many advanced features

and configuration options that make it relatively complex. Compared to some simple messaging systems, DDS may require more compute and memory resources to run. To support real-time performance, DDS introduces some additional overhead such as cache management, network protocol processing, etc. These overheads may affect the overall efficiency of the system.

The MAVLink protocol is a lightweight communication protocol specifically designed for small unmanned aerial systems (UAS). Its role in the UAS application layer is to provide a standardized way to exchange flight control information, sensor data, and other mission-relevant data.

MAVlink has the following advantages as a UAV application layer protocol: Compact message structure: The message body is usually small and suitable for wireless links with low bandwidth and high latency. Cross-platform support: It can run on a variety of hardware platforms, including embedded devices and Windows systems. Ease of implementation: There are many open-source libraries and tools available, simplifying the development process.

MAVlink has the following main limitations. Lack of quality-of-service assurance: It does not provide QoS mechanisms to ensure reliable transmission of critical data. Lack of real-time performance: While it can be used in real-time systems, it is not specifically designed to meet strict real-time requirements. No built-in security features: external mechanisms are required to protect the security and integrity of the data.
In comparison, the two protocols have similar functions, but have different scope of application. The main difference is that DDS is a more complex and powerful real-time data distribution protocol. DDS is designed to solve the problem of data sharing in large-scale distributed systems, so it is more suitable for large and complex distributed systems that need strong data management and distribution capabilities. It also allows users to configure different QoS policies and customize data transmission according to their own needs. MAVlink is more suitable for simple drone applications, especially in resource-constrained environments.

1.4 Common UAV routing protocols (AODV, OLSR, DSR, DSDV)

AODV: Ad-Hoc On-Demand Distance Vector Routing (AODV) is a routing protocol used in mobile Ad hoc networking (MANET). In such a network, there is no fixed infrastructure or central control node, and each node must be able to forward packets as a router to enable communication across the network. Since nodes in this network environment can move around at will, the network topology may change frequently.

Here's how it works: When the source node needs to send data to the destination node and there are no routes available in the local routing table, it broadcasts a "Routing Request" (RREQ) message. The RREQ message is passed along the network, and each node that receives it records the source of the request and forwards it to its neighbors. After receiving the RREQ, the destination node replies with a "Routing Reply" (RREP) message that includes information about the shortest path to the source node. The RREP is returned along the original path, and each node updates its routing table after receiving the RREP so that data can be sent directly through that path in the future.

In UAV AD hoc network, the application of AODV protocol is very important. Since UAVs are usually highly mobile, the communication network between them must be able to quickly adapt to changing topologies. In addition, since drones may be on remote missions or in remote areas, network bandwidth and energy efficiency are important considerations.

The advantage of an AODV is its on-demand nature, which means it creates and maintains routes only when they are needed, which helps reduce unnecessary network overhead. However, for highly dynamic scenarios, such as multiple UAVs flying in tandem, AODVs may need to be optimized to improve routing stability, reduce latency, and improve overall performance.

OLSR: Optimized Link State Routing Protocol (OLSR) is a link-state routing protocol for Mobile AD hoc networking (MET). Unlike distance-vector protocols such as AODV, OLSR uses global network topology information to make routing decisions.

Here's how it works: Each node periodically sends Hello messages to its neighbors to detect and update the list of neighbors. Based on the neighbor list, each node selects a set of MPR nodes and announces this selection in the next Hello message. When a Hello message is received, the node updates its set of neighbors and MPRS. The node generates a TC message based on the neighbor and MPR information, which contains information about all reachable nodes. The TC message is forwarded through the MPR node, thus propagating topological information throughout the network. Each node updates its routing table based on the received TC messages to find the best path for the packets.

In drone communications, OLSR may be more suitable for some scenarios than AODV, as it provides better network visibility and faster convergence. However, the main disadvantage of OLSR is that it requires more bandwidth and computing resources to maintain and propagate topological information.

DSR: Dynamic Source Routing (DSR) is a source routing protocol for mobile AD hoc networking (MANET). Unlike traditional routing protocols such as AODV and OLSR, DSR does not require each node to maintain topology information for the entire network, and instead lets packets carry complete path information.

DSR works as follows: When a source node needs to send data to a destination node, it looks in its routing cache to see if there is a valid route to the destination node. If a valid route is found, the packet is sent directly using that route. If no valid Route is found, the source node broadcasts a "Route Request" (RREQ) message. The RREQ message travels along the network, and each node that receives it records the source of the request and forwards it to its neighbors. After receiving the RREQ, the destination node replies with a "Route Reply" (RREP) message that includes information about the shortest path to the source node. The route reply is returned along the original path, and each node updates its route cache after receiving the reply so that data can be sent directly through that path in the future.

In drone communications, DSR may be more suitable for some scenarios than other routing protocols because it offers lower control overhead and greater flexibility. However, the main disadvantage of DSR is that it is very sensitive to loops in the network and extra measures need to be taken to avoid loops. In addition, since each packet needs to carry complete path information, this can result in the header of the

packet being too large, thus affecting transmission efficiency.

DSDV: Destination-Sequenced Distance Vector (DSDV) is a distance-vector routing protocol for mobile AD hoc networking (MA NET). Unlike on-demand routing protocols such as AODV and OLSR, DSDV uses a periodic update mechanism to maintain routing tables in the network.

Here's how DSDV works: Each node periodically sends a "Route Update" (RU) message to its neighbors, which contains routing information to all other known nodes. Each node that receives the RU message updates its routing table and forwards those updates to its neighbors. Each route in the routing table has a sequence number, and newly received routes that have a larger sequence number than the currently stored route is considered better routes and are updated into the routing table. When the source node needs to send data to the destination node, it looks up the best path to the destination node in the local routing table and uses that path to send a packet.

In drone communications, DSDV may be more suitable for some scenarios than other routing protocols because it offers greater robustness and lower latency. However, the main disadvantage of DSDV is that it requires a lot of bandwidth and computing resources to maintain and propagate network-wide routing information. In addition, since DSDV is based on a distance vector algorithm, it can be very sensitive to link fluctuations and errors in the network.

2 UAV networking development platform and simulation design

2.1 coarse-grained network communication simulation interface

2.1.1 NetUavAPI.NetTransNode:__init__()

This interface is used to initialize a new communication node. PX4MavCtrl.PX4MavCtrler()

2.1.2 startNetServ()

This interface is used to turn on the network emulator. ListenMsgLoop and SendMsgLoop loops are enabled.
Parameters: self, RecPort=-1, netSimPort=20030, netSimIP='224.0.0.10'
RecPort: Listening port.
netSimPort: send port.
netSimIP: send IP.

2.1.3 endHeartSer()

This interface is used to close the network emulator. The ListenMsgLoop and SendMsgLoop loops are closed.

2.1.4 NetUavAPI.HeartServer:__init__()

This interface is used to initialize the heartbeat server.
Parameters:
self.ReceiverIP: Receive IP
self.ReceiverPort: Receive Port
self.SendIP: Send IP
self.SendPort: Send Port
self.RecHeartThreadFlag: Enable receiving flag
self.SendHeartThreatFlag: Enable sending flag
self.CheckStateThreadFlag: Enable Checking flag

2.1.5 startHeartSer()

Used to turn on heartbeat detection server emulation.
Three processes will be started: ReceiveHeartSer, SendHeartSer, CheckConnectState
ReceiveHeartSer: Receive data from the bound ReceiveIP, ReceivePort. When RecHeartThreadFlag is 1, turn on receive.
SendHeartSer: Send a message to the specified SendIP, SendPort. SendHeartThreadFlag is enabled when SendHeartThreadflag is 1.
CheckConnectState: When CheckStateThreadFlag is 1, the detection of communication delays with other aircraft is enabled.

2.1.6 endHeartSer()

Used to stop the status detection server. The ReceiveHeartSer, SendHeartSer, CheckConnectState processes are stopped.

2.2 Coarse-grained UE signal Attenuation simulation interface

2.2.1 CoarseNetworkSimulation:__init__()

Interface to initiate the coarse-grained simulation.
Parameters: numberOfNodes:int, IDstring= "", maxRange:float=1.5, bandwidth:float=100.0
numberOfNodes: Number of emulation nodes
IDstring: Node ID
maxRange: indicates the maximum communication distance amplification factor.
bandwidth: indicates the communication bandwidth.

2.2.2 runSimulation()

To start the simulation. The sendRoutingTable and receiveFromCopterSim processes are opened. Create Redis communication by calling RedisUtils.RedisUtils().

2.2.3 sendRoutingTable()

Used to send signal attenuation data to the appropriate key. Function calls calculateRoutingTable(), calculate PacketLossRate(), GetPos(), calculatePathLose(), Dijkstra(). Used to calculate UE signal attenuation information and its own position information.
calculateRoutingTable() : Attenuation rate calculation.
calculatePacketLossRate() : Calculation of packet loss rate.
GetPos() : Obtain the position information of the aircraft itself.
calculatePathLose() : calculation of signal attenuation information.
Dijkstra() : Maintains optimal communication between nodes.

2.2.4 CreateTarget()

To create a Redis communication instance. And send some information to the Redis server.

2.3 Coarse-grained Network Signal Simulation Interface

2.3.1 PublicUavData()

Publish information about the aircraft itself.

```
UAV1 = {
    "timeDelay":TimeUnix,
    "CopterID":mav.CopterID,    # Vehicle ID
    "uavTimeStmp":mav.uavTimeStmp,      # double uavTimeStmp
    "uavAngEular":mav.uavAngEular,    # float uavAngEular[3]
    "uavVelNED":mav.uavVelNED,    # float uavVelNED[3]
    "uavPosGPSHome":mav.uavPosGPSHome, # double uavPosGPSHome[3]
    "uavPosNED":mav.uavPosNED,    # double uavPosNED[3]
    "uavGlobalPos":mav.uavGlobalPos} # double uavGlobalPos[3]
```

2.3.2 sub_callback()

Store the received data separately to the appropriate path locally, while recording the communication delay. Channel as the identifier, data structure is explained in PublicUavData().
Parameters:
Channel: The channel to which you subscribe.
Data: The data received by the channel.

2.3.3 sub_data_multiple_channels()

To subscribe to multiple channels. To subscribe to incoming channels, when the data type of the received information is the same as the expected type, the callback function is called to process the information. sub_callback() will be called.
Parameters:
message_type: indicates the data type of the subscription.
Channels: The subscribed channel.

Callback: The callback function to call.

2.4 MQTT network communication simulation interface

2.4.1 mqtt.client.Client:__init__()

This is the interface for Mqtt to initialize the client.
Parameters: self, client_id="", clean_session=None, userdata=None, protocol=MQTTv311, transport="tcp", reconnect_on_failure=True
client_id: This argument acts as the unique client ID string used when the current node connects to the proxy terminal. This parameter can be specified on its own or left empty. If it is empty, the proxy terminal will generate the ID for the client when using MQTT v3.1.1. If MQTT v3.1 is used, the ID will be generated randomly by itself. If id is empty, clean_session must be True or it will cause a Value Error error.
clean_session: This argument is a Boolean value. When True, the broker clears all information about the client ID when the client is disconnected from the broker. When true, the information under the current client ID is saved.
Userdata: This is user-defined data. It will be passed to the callback function user_data_set().
The Protocol: argument allows you to explicitly set the version of MQTT that this client should use. The default is v3.1.1.
Transport: This parameter specifies the transport mechanism. There are two options: websockets, tcp. The default is tcp.
reconnect_on_failure: This parameter is a Boolean value. When True, the client automatically tries to reconnect after the connection fails. When False, the client does not repeat the connection attempt. The default is True.

2.4.2 connect()

This interface is used to connect the created client to the remote agent.
Parameters: self, host, port=1883, keepalive=60, bind_address="", bind_port=0, clean_start=MQTT_CLEAN_START_FIRST_ONLY, properties=None
Host: This parameter represents the host name or IP address of the remote agent.
Port: is the network port of the server host you want to connect to. The default is 1883.
Keepalive: This is the maximum interval between the client and the agent to communicate. If no message is exchanged during this time, the client will send a ping message to the proxy.
bind_address: This parameter will be used to determine the local IP address that the client will use to send the data. If this parameter is not specified, the system will select it for itself.
bind_port: This parameter determines the local port on which the client will send data. If this parameter is not specified, the system will select it.
clean_start: This is a connection parameter in MQTT v5.0 that controls the behavior of the session between client and server. When True, the client requests a completely new session, and when connected, the server will clear the previous session data with that client ID. When False, a persistent session will be established, which means that the agent will save the previous conversation data with the client ID. With MQTT_CLEAN_START_FIRST_ONLY, this parameter only takes effect when the client first connects to the broker.

Properties: The MQTT v5.0 property sent in the MQTT connection package.

2.4.3 publish()

This function is used to publish the topic information to the proxy.
Parameters: self, topic, payload=None, qos=0, retain=False, properties=None
topic: The topic that the message should be published on.
Payload: This is the message to send.
Qos: This is the quality of service level to be selected. At 0, the client can send the message at most once, the transmission speed is the fastest, the message may be lost, and the client will not confirm whether the agent received the message. At 0, the client sends the message at least once, which increases some latency and provides higher reliability, and the client resends the message before acknowledging that the agent segment has received the message. At 2, the client sends the message only once, with the highest reliability, and the client will have complex interactions with the proxy segment to ensure data consistency.
Retain: This parameter will be used to retain whether to retain this message as a "known good" message for the subject. If True, the message will be retained.
Properties: (MQTT v5.0 only) MQTT v5.0 properties to be included. Use the Properties class.

2.4.4 loop_start()

This is part of the threaded client interface. You only need to call this method once to start a new thread to handle network traffic. This provides an alternative to repeatedly calling the loop() method yourself. This function doesn't have to pass in arguments.

2.4.5 disconnect()

This function disconnects the client from the agent.
Parameters: self, reasoncode=None, properties=None
Reasoncode: (MQTT v5.0 only) An instance of ReasonCodes to set the MQTT v5.0 reason code to send with disconnect.
properties: (MQTT v5.0 only) A Properties instance for setting MQTT v5.0 properties to be included. If it is not set, no properties are sent.

2.4.6 tls_set()

This function is used to configure network encryption and authentication. Enable SSL/TLS support. Function must be called before the connect() function.
Parameters: self, ca_certs=None, certfile=None, keyfile=None, cert_reqs=None, tls_version=None, ciphers=None, keyfile_password=None
ca_certs: A string path to the certificate file that the client should treat as a trusted certificate authority.
certfile: A string that points to the PEM encoded client certificate.
keyfile: string that points to the private key of the PEM encoded client. This parameter will and certfile be used as client information for TLS based authentication. Supporting this feature is up to the broker.
cert_reqs: Allows you to change the certificate requirements imposed by the client on the agent. The default value is ssl.CERT_REQUIRED, which means that the agent must provide a certificate.

tls_version: Allows specifying the SSL/TLS protocol version to be used. TLS v1.2 is used by default.
ciphers: is a string that specifies the encryption password allowed for this connection.
keyfile_password: If either of certfile and keyfile is encrypted and requires password decryption, then you can use the keyfile_password parameter to pass the password. If keyfile_password is not provided, the password will be requested in the terminal window.

2.5 DDS Network Communication simulation interface

2.5.1 UavMessageWriter.Writer: __init__()

The main role is to set up and configure a DDS publisher, including creating domain participants, topics, publishers, and data writers, and registering data types and supported listeners.
Parameters: def __init__ (self, domain, machine, UavMessageDataType, TopicType)
domain: said DDS (Data Distribution Service) in the domain ID.
Machine: Identifies or describes the machine or device that executes this publisher.
UavMessageDataType: Used to specify the name of the data type to be published.
TopicType: indicates the name of the topic to be created.

2.5.2 UavMessageWriter.Writer.write()

Writes the given Data to the Data Distribution Service (DDS) system so that other subscribers can receive it.
Parameters: def write (self, timeDelay, CopterID uavTimeStmp, uavAngEular, uavVelNED, uavPosGPSHome, uavPosNED, uavGlobalPos)
timeDelay: Delay
CopterID: Drone ID
uavTimeStmp: Drone time stamp
uavAngEular: Euler Angle for drone
uavVelNED: Velocity of the drone in the northeast Earth coordinate system
uavPosGPSHome: initializes the position. Dimension, longitude, height
uavPosNED: Position of the drone in the northeast Earth coordinate system
uavGlobalPos: Position in the drone's global coordinate system

2.5.3 UavMessageReader.Reader: __init__()

The main role is to set up and configure a DDS subscriber, including creating domain participants, topics, subscribers, and data readers, and registering data types and supported listeners.
Parameters: def __init__ (self, domain, data_callback UavMessageDataType, TopicType) domain: Indicates the domain ID in the Data Distribution Service (DDS).
domain: Indicates the domain ID in the Data Distribution Service (DDS).
data_callback: This is a function object argument, supplied by the user. When the subscriber receives new data, this callback function will be called.
UavMessageDataType: Used to specify the name of the data type to subscribe to.
TopicType: The name of the topic to subscribe to.

2.5.4 UavMessageReader.ReaderListener:__init__()

Defines a class called ReaderListener, this class inherits from fastdds. DataReaderListener. DataReaderListener is the class that handles events and callbacks related to data readers.
Create an instance of the ReaderListener object, which is typically used to handle events and callbacks related to data readers. The data_callback argument here is a user-supplied callback function that will be called when new data is received.
Parameters: def __init__(self,data_callback,TopicType)
data_callback: The callback function that is expected to be called when triggered by the subscriber
TopicType: The title of the subscription

2.6 Redis network communication simulation interface

2.6.1 RedisUtils:__init__()

It is used to set up and establish a connection to the Redis database when an instance of the class is created.
Parameters:
Host: Set the host name or IP address of the Redis server
Port: Set the port number of the Redis server
Db: Set the number of the Redis database to connect to
Password: Set the password of the Redis server
Rdb: Create an instance of the redis.StrictRedis object and connect to the Redis server using the host name, port number, database number, and password you set earlier. This object (self.rdb) is used to execute Redis commands
Pubsub: Calls the pubsub() method of self.rdb (i.e., a connected instance of Redis), returning a Pub/Sub (publish/subscribe) object instance (self.pubsub). This object is used to handle the publish/subscribe function of Redis

2.6.2 RedisUtils. sub_data()

What it does is subscribe to data on the specified Redis channel and call a callback function to process messages when they are received that match a particular type.
Parameters:
Channel: The channel to which you subscribe.
Callback: The callback function to call.

2.6.3 RedisUtils. pub_data()

Publishes the data on the specified channel in Redis, and the published data is a string encoded in JSON format.
Patameters: def pub_data(self,key, data)
Key: The specified publishing channel
Data: The data to be published

2.6.4 RedisUtils. sub_data_multiple_channels()

The role is to subscribe to data on multiple Redis channels specified, and to call callback functions to process messages when they are received that match a particular type.
Parameters: def sub_data_multiple_channels(self, message_type, channels, callback)
Channels: Channels to subscribe to the channel, the data type should be a list or a collection.
The remaining arguments have the same meaning as the arguments of RedisUtils.sub_data ().

2.6.5 RedisUtils.set_data()

Used to create or update the corresponding channel data in redis.
Parameter: def set_data(self,key, data)
Key: The key value to be created or updated.
Data: The data to be written.
Unlike publish, data will not be stored in the Redis server, set data will be stored in the Redis server.

2.6.6 RedisUtils.get_data()

Gets the value of the corresponding key value.
Arguments: def get_data(self,key)
Key: The target to get.
Set_data and get_data as the corresponding communication methods. Different from pub and sub, but both can realize the communication function.

2.7 NS3 Communication simulation interface for fine-grained networking

2.7.1 ns3_transition()

The function is to send data packets from one UAV (src_copterID_Uav) to another UAV (dst_copterID_Uav) in NS3 network simulation environment.
Patameters: void ns3_transition(uint 16_t src_copterID_Uav, unit16_t dst_copterID_Uav, Ptr<Packet> packet)
src_copterID_Uav: The Drone that send data
dst_copterID_Uav: The targer drone ID
packet: the data packet to send

2.7.2 ReceiveUav()

Function is to receive by specifying the Socket (sock) from the packet, and according to the content for further processing and forwarding data packets.
Parameter: void ReceiveUav(Ptr<Socket> sock)
Sock: interface that needs to receive data

2.7.3 parameter configuration

uint32_t routing_protocol = 1; //Routing protocol used at the network layer

```
        switch (routing_protocol)
        case 1: "OLSR";
        case 2: "AODV";
        case 3: "DSDV";
        case 4: "AODVKMEANS";
        case 5: "PARROT";
        case 6: "GPSR";
        case 7: "DSR";
        default: exit(1);
int adhocNodes_N = 4;          //>> Number of AD hoc nodes (default value: 4)
int BS_N = 1;                  //>> Number of base station nodes (default is 1, that is, only 1 node is required to receive physical network data)
uint64_t frequency = 5180;     //>> frequency          Mhz
double txPowerBaseDbm = 20; //>> Low transmission power      dbm
double txPowerEndDbm = 20;    //>> High transmission power      dbm
double Start_time = 5;    //>> Simulation start time
double Total_time = 1000; //>> Simulation end time
double init_energy = 1 * 11 * 3600;          //>> Maximum initial energy
```

2.8 Communication Simulation interface for formation networking

2.8.1 PX4MavCtrler: __init__()

Parameters: self, ID=1, ip='127.0.0.1',Com='udp',port=0 ID: emulation ID
Ip: The IP address from which the data is sent out
Com: Connection mode with Pixhawk
Port: port number
Function to initialize aircraft, aircraft communication mode.

2.8.2 InitMavLoop()

Parameters: UDPMode=2
UDPMode: 0 and 1 correspond to UDP_Full and UDP_Simple Mode, 2 and 3 correspond to MAVLink_Full an d MAVLink_Simple mode, and 4 correspond to MAVLink_NoSend
isCom: indicates the flag for serial communication
isRealFly: indicates the symbol of the direct network connection mode
isRedis: symbol of the Redis mode
Function to establish communication. The default mode is MAVLink_Full

2.8.3 initOffboard()

This function requires no input arguments to start.
The function sends Offboard commands to put the aircraft into Offboard mode, and it starts sending Offboard messages at a frequency of 30Hz.
The function sends the message by calling sendMavOffboardAPI().

2.8.4 sendMavOffboardAPI()

Parameters: type_mask = 0, coordinate_frame = 0, pos = 0, 0, vel = 0, 0, acc = 0, 0, yaw = 0, yawrate = 0
type_mask: control mode, such as local position control, global position control, etc
coordinate_frame: Coordinate system type
pos: Position information
vel: Speed information
acc: Acceleration information
yaw: yaw Angle information
yawrate: yaw Angle rate information
The function determines the offboard mode according to the offMode variable and calls the corresponding function to send information.

2.8.5 SendPosNED()

Parameters: x=0,y=0,z=0,yaw=0
x: target location information
y: target location information
z: target location information
yaw: yaw Angle
Function will assign the offMode variable to 0, representing entry into the local NORtheasterly coordinate position control.

2.8.6 endOffboard()

No arguments are required for the call, which is called directly.
The function will send PX4 to exit offboard mode, stopping the message sending loop.

2.8.7 stopRun()

No arguments are required for the call, just call it directly.
The function will stop mavlink's message listening loop.

2.8.8 initPointMassModel()

Parameters:
intAlt=0
intState=[0,0,0]
intAlt: initial altitude of aircraft
intState: initial position of aircraft, initial yaw Angle function is used to start particle model simulation.

2.8.9 PointMassModelLoop()

Function is the update loop of the particle model. Will be called by the initPointMassModel() function. The user does not need to start.

2.8.10 sendUE4PosNew()

Parameters: copterID=1, vehicleType=3, PosE=[0,0,0], AngEuler=[0,0,0], VelE=[0,0,0], PWMs=[0]*8, runnedTime=-1, windowID=-1
copterID: Simulated aircraft ID
vehicleType: Aircraft style
PosE: NORD coordinate position
AngEuler: aircraft attitude Angle
VelE: Aircraft speed
PWMs: Rotational speed
runnedTime: current time
windowID: window number
The function packages the incoming data and sends it to RflySim3D to create a new 3D model or update the status of the model.

2.8.11 InitTrueDataLoop()

This function initializes the UDP real data listening loop. Listen to data from CopterSim through the 30100 series port.

2.8.12 EndTrueDataLoop()

Function closes the UDP real data listening loop.

2.8.13 endMavLoop()

Function has the same function as the stopRun() function. Data listening on series 20100 ports will be turned off.

2.8.14 SendMavCmdLong()

Parameters: command, param1=0, param2=0, param3=0, param4=0, param5=0, param6=0, param 7=0
For the definitions of command and param1~7, refer to:
https://mavlink.io/en/messages/common.html#COMMAND_LONG
https://mavlink.io/en/messages/common.html#MAV_CMD
function is used to send a message to PX4

2.8.15 sendMavOffboardCmd()

Parameters: type_mask,coordinate_frame, x, y, z, vx, vy, vz, afx, afy, afz, yaw, yaw_rate
type_mask: control mode
coordinate_frame: coordinate system information x: position information
y: position information
z: location information
vx: speed information
vy: Speed information

vz: Speed information
afx: Acceleration information
afy: acceleration information
afz: acceleration information
yaw: yaw Angle
yaw_rate: yaw Angle rate
Function to send offboard instructions to PX4

2.8.16 sendUDPSimpData()

Parameters: ctrlMode,ctrls
ctrlMode: Airplane mode
ctrls: Control parameters. Example: If the flight mode is take-off, the control parameters should be [x, y, z, yaw] Function sends the simple UDP message to the communication port.

2.8.17 SendVelNED()

Parameters: vx=0,vy=0,vz=0,yawrate=0 vx: speed control instruction
vy: Speed control instruction
vz: Speed control instruction yawrate: yaw Angle rate instruction
Function is used to switch flight mode to earth speed control mode, switch offboard mode, send message discriminant bit, set coordinate system.

2.8.18 SendVelNEDNoYaw()

Parameters: vx,vy,vz
vx: Speed control instruction
vy: Speed control instruction
vz: Speed control instruction
Function similar to SendVelNED(). The difference is that the function does not set the yaw Angle rate instruction, only sends the speed control instruction.

2.8.19 SendVelFRD()

Parameters: vx=0,vy=0,vz=0,yawrate=0 vx: speed control instruction
vy: Speed control instruction
vz: Speed control instruction yawrate: yaw Angle rate instruction
Function will switch flight mode to body speed control mode and offboard mode to speed control mode. Switch the working coordinate system to the collective coordinate system. The positive direction of the coordinate system is front, right and down.

2.8.20 SendAttPX4()

参数：att=[0,0,0,0],thrust=0.5,CtrlFlag=0,AltFlg=0

att：根据 CtrlFlag 确定

CtrlFlag 0 : att 是三维向量，包含横滚角，俯仰角，偏航角，单位是角度

CtrlFlag 1 : att 是三维向量，包含横滚角，俯仰角，偏航角，单位是弧度

CtrlFlag 2 : att 是四维向量，包含四元数

CtrlFlag 3 : att 是三维向量，包含横滚角速率，俯仰角速率，偏航角速率，单位是 rad/s

CtrlFlag 4 : att 是三维向量，包含横滚角速率，俯仰角速率，偏航角速率，单位是 degree/s

thrust：根据 AltFlg 确定

AltFlg 0 ：总推力，归在 0-1 范围内，（-1~1 适用于有反向推力的飞机）

AltFlg>0：期望高度

CtrlFlag：用于确定输入 att 定义的标志

AltFlg：用于确定输入 thrust 定义的标志

函数将切换飞机的飞行模式为姿态控制模式，切换 offboard 模式。将 FRD（前，右，下）坐标系下飞机目标姿态信息发送到 PX4

2.8.21 SendAccPX4()

Parameters：afx=0,afy=0,afz=0,yawValue=0,yawType=0,frameType=0
afx: Acceleration information
afy: acceleration information
afz: acceleration information
yawValue: yaw information
yawType: 1, yaw Angle control. 2, yaw Angle rate control frameType: 0, ground north east ground coordinate system. 1, body FRD coordinate system
The function will switch the aircraft's flight mode to acceleration control mode and switch offboard mode. Toggle coordinate system according to frametype. Send acceleration control information to PX4

2.8.22 SendVelNoYaw()

Parameters: vx,vy,vz
vx: Speed control instruction
vy: Speed control instruction
vz: Speed control instruction
Function has the same function as SendVelNEDNoYaw(), except that the function sets the current coordinate system to the body FRD coordinate system.

2.8.23 SendVelYawAlt()

Parameters: vel=10, yaw=6.28,alt=-100 vel: speed instruction

yaw: yaw Angle instruction
alt: Altitude command
Function will set the flight mode to speed altitude yaw mode. Toggle offboard mode. Switch the coordinate system to the NE Earth coordinate system. Send the aircraft's yaw information, speed information and altitude information to PX4

2.8.24 SendPosGlobal()

Parameters: lat=0, lon=0,alt=0,yawValue=0,yawType=0 lat: Latitude information
lon: Longitude information
alt: Altitude
yawvalue: yaw
yawType: 0, no yaw. 1, yawvalue is the yaw Angle. 2, yawvalue is the yaw Angle rate
Function will switch flight mode to Earth position control mode, switch offboard mode and coordinate system to Northeast Earth coordinate system. Send the aircraft's position control information and yaw information to PX4

2.8.25 SendPosNEDNoYaw()

Parameters: x=0, y=0, z=0,
x: target location information
y: target location information
z: target location information
Function similar to 2.1.5 SendPosNED() function. Fly with position control mode. The difference is that the function does not send the yaw information of the aircraft.

2.8.26 SendPosFRD()

Parameters: x=0,y=0,z=0,yaw=0
x: target position information
y: target position information
z: target position information
yaw: yaw information
Function sets the flight mode to the body position control mode. Set offboard mode to local NORtheast-coordinate position control mode. Set the working coordinate system as the FRD coordinate system of the body, and the function sends the position control instructions and yaw Angle control instructions to PX4.

2.8.27 SendPosFRDNoYaw()

Parameters: x=0,y=0,z=0
x: target location information
y: target location information
z: target location information
The function is similar to the 2.1.26 SendPosFRD() function, except that the function does not send aircraft yaw Angle control instructions.

2.8.28 SendPosNEDExt()

Patameters: x=0,y=0,z=0,mode=3,isNED=True
x: target location information
y: target location information
z: target location information
mode: 0, glide mode. 1, Take off mode. 2, landing mode. 3, gyrocopter hover mode, fixed wing hover mode. 4, fixed wing aircraft, no throttle, no roll/pitch
isNED: True, local northeast Earth coordinate system. False: the body coordinate system is northeast
Function is used to publish aircraft position control. The position control command is issued after resetting the aircraft's operating mode according to the input.

2.8.29 enFixedWRWTO()

Function to send takeoff permission to an aircraft on the runway. The function is called without arguments.

2.8.30 SendCruiseSpeed()

Parameter: Speed=0
Speed: The cruising speed of the aircraft. Function used to change the cruising speed of an aircraft.

2.8.31 SendCruiseRadius()

Parameter: rad=0
rad: Cruising radius of the aircraft. Function to modify the cruising radius of an aircraft.

2.8.32 sendMavTakeOffLocal()

Parameters: xM=0,yM=0,zM=0,YawRad=0,PitchRad=0,AscendRate=2
xM: location information
yM: Location information
zM: location information
YawRad: yaw Angle rate
PitchRad: Pitch Angle rate
AscendRate: rate of rise
Function to send the desired local position command to the aircraft to take off to the desired position.