Apollo 用户手册

—V1.0

1 概述

为降低无人机的落地门槛，让无人机进一步赋能行业应用，拓攻采用“大脑+小脑”的新一代系统架构，依托拓攻卓越的飞控系统（小脑）和强大的 Apollo 机载计算机（大脑），将无人机构建为面向各行各业的开放飞行平台。

作为开放飞行平台的大脑，Apollo 具备强大的运算能力、丰富的接口、高速联网和高效的二次开发能力，配合 M2 飞控，可以让无人机应用场景的开发变得更加便捷，让无人机飞跃无界、纵横无疆。Apollo 典型能力如下：

运算处理能力：Apollo 采用 Freescale i.MX6Q 处理器，由 4 颗 ARM Cortex-A9 处理器组成，主频高达 1.2GHZ，同时还搭载 2GBDDR3 运行内存和 16GB EMMC 存储空间，具备强大的运算能力。此外，Apollo 还搭载一颗 STM32F103 MCU 作为协处理器，用于实现和强实时设备的交互，确保整个系统的实时性。

飞行控制能力：Apollo 通过接入 M2 飞控，实现无人机飞行控制和状态监控。

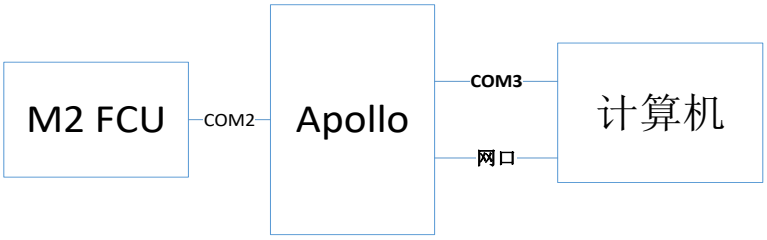
高速联网能力：Apollo 内置高速 4G 模块，可支持中国移动、中国电信和中国联通。通过 4G 通信，用户可实现对无人机的超远程控制、集群控制、远程实时监控等应用，让无人机变成真正的在线智能平台。

外设扩展能力：Apollo 提供了丰富的接口支持，方便接入机载设备（任务载荷、传感器、控制模块等）进行管理和控制。

便捷开发能力：Apollo 搭载 Linux 操作系统，并集成了拓攻 SDK，非常方便机载应用软件的开发。同时 Apollo 还搭载了 ROS，为无人机应用应用软件的开发提供便利。

2 环境搭建

2.1 开发态



如上图所示，将 Apollo 串口 2 接入 M2 FCU,将串口 3 接入用户计算机，同时将 Apollo 网口接入用户计算机。串口默认配置如下

默认配	波特率	数据位	停止位	校验位	流控制	文件系统中名	备注

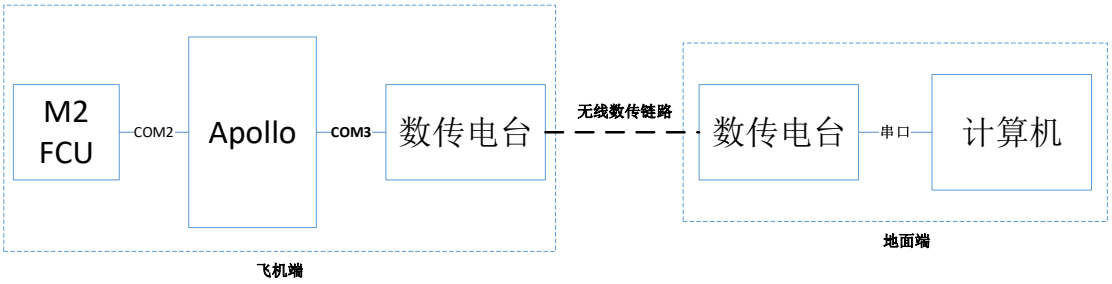
置	(bps)	(bits)	(bits)			称	
串口 1	—	—	—	—	—	/dev/ttymxc4	用户使用
串口 2	256000	8	1	NONE	NONE	/dev/ttymxc0	飞控接入专用
串口 3	115200	8	1	NONE	NONE	/dev/ttymxc2	超级终端专用

在用户计算机上，通过串口超级终端（如 XShell5）登录 Apollo，默认为 root 用户，用户在此时可配置 Apollo 网口 IP 地址，将其 IP 地址配置成与用户计算机同一网段。IP 的配置可使用 linux ifconfig 命令进行。

在用户计算机上，通过 ssh（如 XShell5）远程登录 Apollo，Apollo 默认配置了 linaro 和 root 用户，密码均为 linaro。若无法登录，请确保 Apollo 网口 IP 地址配置正确，且 sshd 服务已运行。通过 ssh 远程登录 Apollo 后，可使用 linux 命令进行软件开发、调试，以及软件安装、卸载等操作。

注：用户可在计算机上安装 Xshell5，使用 Xshell5 通过串口、SSH 接入 Apollo。

2.2 飞行态



如上图所示，Apollo 安装在飞机上，其串口 2 接入 M2 FCU，串口 3 接入数传电台。地面端用户计算机通过串口接入数传电台，串口配置如下。

默认配置	波特率(bps)	数据位(bits)	停止位(bits)	校验位	流控制
Apollo 串口 2	256000	8	1	NONE	NONE
Apollo 串口 3	115200	8	1	NONE	NONE
计算机串口	115200	8	1	NONE	NONE

在用户计算机，用户可以使用串口终端（如 XShell5）接入 Apollo，并进行远程控制。

注意：

1） 在飞行态下，请确保飞机处于空旷环境，保证足够的飞行空间，确保用户同飞机保持安全距离。

2） 飞机处于姿态模式(STABILIZE)时，Apollo 无法对飞机进行飞行控制，仅可进行参数查询、遥测数据获取以及电机加解锁。只有通过遥控器将飞机设置为 GPS 模式

(HYBRID)后，Apollo 方可对飞机进行飞行控制。Apollo 在飞行控制过程中，用户可以通过将飞机切入姿态模式(STABILIZE)，实现对飞机的接管控制。

3）油门低位时，Apollo 下达电机解锁命令后，飞控会自动进行加锁，因此 Apollo 控制时需将油门调到中位。

3 API 手册

3.1 飞行控制

3.1.1 基本控制

Apollo 提供基本的飞行控制功能，主要包括获取 Apollo UID、M2 FCU 的 UID、M2 FCU 的基本配置参数、电机加解锁、自动起飞、自动返航、自动降落、即点即飞等。

3.1.1.1 获取同飞控的链接状态

方法名称	connection
所属模块	drone_proxy
所属类	Drone
功能	获取 Apollo 同 M2 飞控的链接状态
输入	无
输出	true 表示链接正常； fasle 表示链接断开

具体使用参考例程 1。

3.1.1.2 获取 Apollo UID

方法名称	getApolloUID
所属模块	drone_proxy
所属类	Drone
功能	获取 Apollo 的 UID
输入	无
输出	uid： string 类型，若返回值 success 为 true，该字段为获取的的 UID，6 个字节，以 12 个十六进制字符串显示

具体使用参考例程 1。

3.1.1.3 获取飞控 UID

方法名称	getM2FcuUID
所属模块	drone_proxy
所属类	Drone

功能	获取 M2 FCU 的 UID
输入	无
输出	success：bool 类型，true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因
	uid： string 类型，若返回值 success 为 true，该字段为获取的的 UID，UID 为 12 字节，24 个以十六进制字符串显示

具体使用参考例程 1。

3.1.1.4 获取飞控基本配置信息

方法名称	getFCUBasicParams
所属模块	drone_proxy
所属类	Drone
功能	获取飞控的基本配置参数
输入	无
输出	success：bool 类型，true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因
	params：FCUBasicParams 类型，若返回值 success 为 true，该字段为获取的飞控基本参数

FCUBasicParams 类型定义如下：

No	内容	物理值					备注
		意义	量纲	类型	精度	取值	
1.	type	飞行器类型	/	enum	/	0 表示十型四旋翼 1 表示×型四旋翼 2 表示 Y 型六旋翼 3 表示倒 Y 型六旋翼 4 表示米型六旋翼 5 表示×型六旋翼 6 表示十型八旋翼 7 表示×型八旋翼 8 表示×型上下共轴八旋翼 9 表示×型六轴八桨 10 表示十型四轴六桨 11 表示固定翼 12 表示单旋翼直升机	
2.	maxHDipAngle	最大水平倾角	deg	int	1	0~90	
3.	maxHAngleVelo	最大水平角速度	deg/s	int	1	0~255	
4.	maxYawRate	最大偏航角速度	deg/s	int	1	0~255	
5.	maxFlyHeight	最大高度限制	m	float	0.1	0~1000	

6.	maxVVelo	最大垂直速度	m/s	float	0.01	0～300	
7.	maxHVelo	最大水平速度	m/s	float	0.01	0～300	

具体使用参考例程 1。

3.1.1.5 电机加锁

方法名称	lock
所属模块	drone_proxy
所属类	Drone
功能	对电机进行加锁
输入	无
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

具体使用参考例程 2。

3.1.1.6 电机解锁

方法名称	unlock
所属模块	drone_proxy
所属类	Drone
功能	对电机进行解锁
输入	无
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

具体使用参考例程 2。

3.1.1.7 获取自动返航高度

方法名称	getReturnHeight
所属模块	drone_proxy
所属类	Drone
功能	返回当前的自动返航高度
输入	无
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr：string 类型，若返回值 success 为 false，该字段描述失败的原因
	height: float 类型，当前的自动返航高度

具体使用参考例程 3。

3.1.1.8 设置自动返航高度

方法名称	setReturnHeight
所属模块	drone_proxy

所属类	Drone
功能	设置自动返航高度
输入	height：float 类型，指定起飞高度，取值-300-300m，精度到 0.1m
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr：string 类型，若返回值 success 为 false，该字段描述失败的原因

具体使用参考例程 3。

3.1.1.9 自动起飞

方法名称	takeoff
所属模块	drone_proxy
所属类	Drone
功能	按用户指定高度进行自动起飞，起飞后转入悬停模式
输入	height：float 类型，指定起飞高度，取值 3-100m，精度到 0.01m
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr：string 类型，若返回值 success 为 false，该字段描述失败的原因

具体使用参考例程 3。

3.1.1.10 自动降落

方法名称	land
所属模块	drone_proxy
所属类	Drone
功能	自动降落，飞机着落后，电机自动加锁
输入	无
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

具体使用参考例程 3。

3.1.1.11 即点即飞

方法名称	flyto
所属模块	drone_proxy
所属类	Drone
功能	向用户指定目标点飞行，到达后处于悬停模式
输入	latitude: double 类型，目标点的纬度，取值-90~90deg，精度到 1e-7
	longtitude: double 类型，目标点的经度，取值-180~180deg，精度到 1e-7
	altitude: float 类型，高度-300~300m，精度到 0.01m
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr：string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.1.12 自动返航

方法名称	returnHome
所属模块	drone_proxy

所属类	Drone
功能	自动返回到起飞点并降落，着陆后电机加锁
输入	course: 航向，0 表示保持当前航向，1 表示复位为起飞航向
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr：string 类型，若返回值 success 为 false，该字段描述失败的原因

具体使用参考例程 3。

3.1.2 遥测数据

3.1.2.1 遥测数据类型

3.1.2.1.1 地面系经纬高

类型为 Position，包含以下字段。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	
1	latitude	纬度	deg	double	1e-7	-90-90	
2	longitude	经度	deg	double	1e-7	-180-180	
3	altitude	高度	m	float	1e-3	-500-8000	

3.1.2.1.2 地面系线速度

类型为 Velocity，包括以下字段。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	
1	vel_north	北向速度	m/s	float	1e-2	-300~300	
2	vel_east	东向速度	m/s	float	1e-2	-300~300	
3	vel_ground	地向速度	m/s	float	1e-2	-300~300	

3.1.2.1.3 欧拉角

类型为 EulerAngle，包括以下字段。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	
1	roll	滚转角	deg	float	1e-2	-180-180	
2	pitch	俯仰角	deg	float	1e-2	-90-90	
3	yaw	偏航角	deg	float	1e-2	0-360	

3.1.2.1.4 机体滤波线加速度、机体原始加速度

类型为 Acc，包含以下字段。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	
1	acc_x	x 轴加速度	m/s^2	float	1e-6	-50~50	
2	acc_y	y 轴加速度	m/s^2	float	1e-6	-50~50	

3	acc_z	z 轴加速度	m/s^2	float	1e-6	-50~50	

3.1.2.1.5 机体滤波角速度、机体原始角速度

类型为 Gyro，包含以下字段。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	
1	gyro_x	滚转角速率	rad/s	float	1e-6	-20~20	
2	gyro_y	俯仰角速率	rad/s	float	1e-6	-20~20	
3	gyro_z	偏航角速率	rad/s	float	1e-6	-20~20	

3.1.2.1.6 GPS 原始数据

类型为 Gps，包含以下字段。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	
1	latitude	纬度	deg	double	1e-7	-90-90	
2	longitude	经度	deg	double	1e-7	-180-180	
3	altitude	高度	m	float	1e-3	-500~3000	
4	northVelocity	北向速度	m/s	float	0.01	-100~100	
5	eastVelocity	东向速度	m/s	float	0.01	-100-100	
6	downVelocity	地向速度	m/s	float	0.01	-100-100	
7	sateNumber	卫星数	/	u8	1	0~32	
8	hAcc	水平精度因子	m	float	0.001	0-50	
9	vAcc	垂直精度因子	m	float	0.001	0-50	
10	sAcc	速度精度椅子	m/s	float	0.01	0-50	
11	fixType	定位状态	/			0:无 GPS 信号; 1:GPS 信号差; 2:GPS 信号一般，但可定位; 3:GPS 信号良好	

3.1.2.1.7 RTK 原始数据

类型为 Rtk，包含以下字段。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	
1	latitude	RTK 纬度	deg	double	1e-7	-90-90	
2	longitude	RTK 经度	deg	double	1e-7	-180-180	
3	altitude	RTK 高度	m	float	1e-3	-8000-8000	
4	northVelocity	RTK 北向速度	m/s	float	0.01	-100~100	
5	eastVelocity	RTK 东向速度	m/s	float	0.01	-100-100	
6	downVelocity	RTK 天向速度	m/s	float	0.01	-100-100	

7	locationStatus	RTK 定位状态	/	u8	1	0-255	
8	sateNumber	RTK 星数	/	u8	1	0-255	
9	baseHealthy	RTK 基站健康	/	u8	1	0-255	
10	rtkHeading	RTK 方向	deg	u16	0.01	0-360	
11	rtkHeadingStd	RTK 方向标准差	deg	u8	0.1	0-32	
12	frameLossRate	RTK 底板丢帧率	/	u8	1	0-100	

3.1.2.1.8 电机输出

类型为 MotorOutput，包含以下字段。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	
1	M1	电机 1	%	u8	1	0~100	
2	M2	电机 2	%	u8	1	0~100	
3	M3	电机 3	%	u8	1	0~100	
4	M4	电机 4	%	u8	1	0~100	
5	M5	电机 5	%	u8	1	0~100	
6	M6	电机 6	%	u8	1	0~100	
7	M7	电机 7	%	u8	1	0~100	
8	M8	电机 8	%	u8	1	0~100	

3.1.2.1.9 磁罗盘数据

类型为 Mag，包含以下字段。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	
1	mag_x	x 轴磁场值	mGauss	s16	1	-1000~1000	
2	mag_y	y 轴磁场值	mGauss	s16	1	-1000~1000	
3	mag_z	z 轴磁场值	mGauss	s16	1	-1000~1000	

3.1.2.1.10 气压计数据

类型为 Barometer，包含以下字段。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	
1	press	气压	Pa	u32	1	0-150000	
2	tempr	温度	℃	s8	1	-127-127	
3	altitude	高度	m	float	0.1	-3276-3276	

3.1.2.1.11 电压数据

类型为 Voltage，包含以下字段。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	

1	bat_volt	电源总压	V	u16	0.01V	0-655.35	
2	can_volt	CAN 总线电压	V	u16	0.01V	0-655.35	
3	pmu_tempr	PMU 温度	℃	s16	0.01℃	-327-327	

3.1.2.1.12 飞行状态

类型为 State，包含以下字段。

No	内容	物理值			备注
		意义	类型	取值	
1	armed	是否已解锁	bool	true/false，true 表示已解锁	
2	landed	是否已着陆	bool	true/false，true 表示已着落	
3	mode	当前飞行模式	string	"STABILIZE":姿态模式 "ACRO":手动模式 "LOITER":悬停模式 "HOTPOINT" :定点环绕模式 "GUIDED" :即点即飞模式 "WAYPOINT":航点模式 "RETURN":自动返航模式 "TAKEOFF":自动起飞模式 "LAND":自动着陆模式 "HYBRID":GPS 模式 "API":API 控制模式 "FOLLOWME":跟随模式	
4	stabilize	姿态模式、悬停模式详细状态信息	StabilizeStatus	status：int 类型，0 表示运动中，1 表示悬停中	当处于姿态模式、悬停模式时有效
5	hotpoint	定点环绕模式详细状态信息	HotPointStatus	status：int 类型，0 表示顺时针环绕，1 表示逆时针环绕，2 表示半径调整，3 表示速度调整 surroundTimes：int 类型，表示已环绕圈数	当处于定点环绕模式时有效
6	guid	即点即飞模式详细状态信息	GuidedStatus	status：int 类型，0 表示运动中，1 表示已到达目标点	当处于即点即飞模式时有效
7	waypoint	航点模式详细信息	WayPointStatus	status：int 类型，0 表示运动，1 表示已到达航点 trackId：int 类型，表示航线号 segmentId：int 类型，表示航段号 waypointId：int 类型，表示航点号	当处于航点模式时有效，一个航线有一到多个航段组成，一个航段有一到多个航点组成

8	rtl	自动返航详细状态信息	ReturnStatus	type：int 类型，0 表示沿预设高度返航，1 表示沿原航路返航，2 表示沿预设航路返航	当处于自动返航模式时有效
				status：int 类型，0 表示返航开始，1 表示爬升/下降至返航高度，2 表示正在返回至 home 点上空，3 表示正在 home 点上空悬停，4 表示正在下降，5 表示正在着陆，8 表示正在飞往目标点，9 表示已到达目标点	
				targetId：int 类型，正飞往该航点	
9	takeoff	自动起飞详细状态信息	TakeoffStatus	status：int 类型，0=正在起飞，1=已到达起飞高度	处于自动起飞模式时有效
10	land	自动降落详细状态信息	LandStatus	status：int 类型，0=正在飞往着陆点，1=正在着陆点上空悬停，2=正在下降	当处于自动降落模式时有效
11	api	API 控制模式详细信息	CtrlMode	hMode：int 类型，水平方向控制方式，0 表示水平姿态，1 表示水平速度，2 表示水平位置	当处于 API 控制模式时有效
				vMode：int 类型，垂直方向控制方式，0 表示油门输入，1 表示垂直速度，2 表示垂直位置	
				yawMode：int 类型，偏航方向控制方式，0 表示偏航角速率，1 表示偏航角度	
				coordinate：int 类型，控制模式采用的坐标系，0 表示地面系，1 表示机体系	
12	followme	跟随模式详细状态信息	FollowMeStatus	status：int 类型，0 表示正在跟踪，1 表示距离保持，2 表示航向调整	当处于跟随模式时有效
13	alarm	告警信息	AlarmStatus	magErr：bool 类型，ture 表示磁罗盘异常； 类似的 barometerErr 为 true 表示气压计异常，imuErr 为 true 表示 IMU 异常，gpsErr 为 true 表示 GPS 信号差，beyondHeight 为 true 表示超	告警信息

				出高度限制，beyondPosition 为 true 表示超出位置限制，RCloss 为 true 表示遥控消失，lowVoltage1 为 true 表示一级低压，lowVoltage2 为 true 表示二级低压	

3.1.2.2 主动查询遥测数据

方法名称	getMsg
所属模块	drone_proxy
所属类	Drone.telemetry
功能	查询当前最新的遥测数据，若超时或查询失败，返回字符串 none
输入	topic：遥测数据类型，string 类型，支持 "/UAV/Connection"：同 FCU 的链接状态 "/UAV/EulerAngle"：欧拉角 "/UAV/FilterAcc"：机体滤波加速度 "/UAV/FilterGyro"：机体滤波角速度 "/UAV/Mag"：磁罗盘数据 "/UAV/MotorOutput":电机输出 "/UAV/Position"：大地坐标系经纬高 "/UAV/RawAcc":机体原始加速度信息 "/UAV/RawGyro"：机体原始角速度 "/UAV/RawGps"：Gps 原始数据 "/UAV/Rtk"：RTK 原始数据 "/UAV/State":飞行状态 "/UAV/Velocity":大地系速度 "/UAV/Voltage"：电压
	timeout，超时时间，double 类型，单位 s
输出	msg：若为字符串“none”表示，timeout 时间内，没有收到消息；否则为相应的 topic 数据

详细使用参见例程 3。

3.1.2.3 订阅接收遥测数据

用户可订阅遥测数据，进行异步数据接收处理。Apollo 提供遥测数据订阅、启动接口。

3.1.2.3.1 订阅数据

方法名称	listen
所属模块	drone_proxy
所属类	Drone.telemetry
功能	订阅指定类型的遥测数据，并注册相关数据处理回调函数
输入	topic_handles：字典类型，key 为 string 类型，表示订阅的消息类型，value 为处理该消息的回调函数；可以取值如下 "/UAV/Connection"：同 FCU 的链接状态

	"/UAV/EulerAngle": 欧拉角 "/UAV/FilterAcc": 机体滤波加速度 "/UAV/FilterGyro": 机体滤波角速度 "/UAV/Mag": 磁罗盘数据 "/UAV/MotorOutput":电机输出 "/UAV/Position": 大地坐标系经纬高 "/UAV/RawAcc":机体原始加速度信息 "/UAV/RawGyro": 机体原始角速度 "/UAV/RawGps": Gps 原始数据 "/UAV/Rtk": RTK 原始数据 "/UAV/State":飞行状态 "/UAV/Velocity":大地系速度 "/UAV/Voltage": 电压
输出	无

3.1.2.3.2 取消订阅

方法名称	unlisten
所属模块	drone_proxy
所属类	Drone.telemetry
功能	停止订阅该主题的遥测数据
输入	string 类型，表示订阅的消息类型，可以取值如下 "/UAV/Connection": 同 FCU 的链接状态 "/UAV/EulerAngle": 欧拉角 "/UAV/FilterAcc": 机体滤波加速度 "/UAV/FilterGyro": 机体滤波角速度 "/UAV/Mag": 磁罗盘数据 "/UAV/MotorOutput":电机输出 "/UAV/Position": 大地坐标系经纬高 "/UAV/RawAcc":机体原始加速度信息 "/UAV/RawGyro": 机体原始角速度 "/UAV/RawGps": Gps 原始数据 "/UAV/Rtk": RTK 原始数据 "/UAV/State":飞行状态 "/UAV/Velocity":大地系速度 "/UAV/Voltage": 电压
输出	无

3.1.2.3.3 启动接收

方法名称	start
所属模块	drone_proxy
所属类	Drone.telemetry
功能	启动遥测数据接收，收到遥测数据后，根据类型调用相关的回调函数
输入	无
输出	无

3.1.3 高级控制

Apollo 提供高级飞行控制能力，用户可设定水平、垂直、偏航方向的控制模式对飞行器进行灵活控制。目前，水平方向支持水平姿态、水平速度、水平位置三种控制模式，垂直方向支持油门输入、垂直速度、垂直位置三种控制模式，偏航方向支持偏航角速率、偏航角度两种控制模式，同时支持用户指定采用的坐标系（地面系或机体系）。

为方便使用，提供以下三种常用控制模式：

- 位置偏航控制：坐标系采用大地系，水平方向采用位置控制，垂直方向采用位置控制，偏航方向采用偏航角度控制；
- 速度偏航速率控制：坐标系采用大地系，水平方向采用速度控制，垂直方向采用速度控制，偏航方向采用偏航角速度控制；
- 定高姿态控制：坐标系采用大地系，水平方向为姿态控制，垂直方向为位置控制，偏航方向为角度控制；

本节方法使用例程见历程 5。

3.1.3.1 设置失效保护策略

方法名称	setFailsafePolicy
所属模块	drone_proxy
所属类	Drone.flyCtrl
功能	设置高级控制失效时的保护措施，在 API 模式下，若飞控同 Apollo 断开链接，将触发此策略
输入	failsafe：指定失效保护措施，int 类型，取值： 1 表示原地悬停，等待其它有效指令，若无其它有效指令接入，则执行预设失控保护策略； 2 表示沿预设高度返航； 3 表示原地降落； 4 表示若执行航路，则航路结束后返航
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.3.2 进入用户自定义 API 控制模式

方法名称	enter
所属模块	drone_proxy
所属类	Drone.flyCtrl
输入	flag：CtrlMode 类型，指定采用的控制模式。 CtrlMode 类型包括以下字段：

	➢ hMode：int 类型，指定水平方向控制方式，0 表示水平姿态，1 表示水平速度，2 表示水平位置 ➢ vMode：int 类型，指定垂直方向控制方式，0 表示油门输入，1 表示垂直速度，2 表示垂直位置 ➢ yawMode：int 类型，指定偏航方向控制方式，0 表示航角速率，1 表示偏航角度 ➢ coordinate：int 类型，指定采用的坐标系，0 表示地面系，1 表示机体系
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.3.3 进入位置偏航 API 控制模式

方法名称	enterPositionAndYawCtrl
所属模块	drone_proxy
所属类	Drone.flyCtrl
输入	无
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.3.4 进入速度角速率 API 控制模式

方法名称	enterVelocityAndYawRateCtrl
所属模块	drone_proxy
所属类	Drone. flyCtrl
输入	无
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.3.5 进入姿态定高 API 控制模式

方法名称	enterAttitudeAndVerPosCtrl
所属模块	drone_proxy
所属类	Drone. flyCtrl
输入	无
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.3.6 用户自定义 API 控制

方法名称	control
所属模块	drone_proxy
所属类	Drone. flyCtrl
输入	flag：CtrlMode 类型，需同 enter 方法设置的模式保持一致

	x：double 类型，x 轴方向的控制量输入 ➢ 若 flag 中水平方向为位置控制，x 表示纬度，取值-90-90deg，精度到 1e-7deg ➢ 若 flag 中水平方向为速度控制，x 表示北向速度，取值为-300~300 m/s，精度到 0.01m/s ➢ 若 flag 中水平方向为姿态控制，x 表示滚转角度，取值为-180-180 deg，精度到 0.01deg
	y：double 类型，y 轴方向的控制量输入 ➢ 若 flag 中水平方向为位置控制，y 表示经度，取值-180-180deg，精度到 1e-7deg ➢ 若 flag 中水平方向为速度控制，y 表示东向速度，取值为-300~300 m/s, 精度到 0.01m/s ➢ 若 flag 中水平方向为姿态控制，y 表示俯仰角度，取值为-90-90deg, 精度到 0.01deg
	z：double 类型，z 轴方向控制量输入 ➢ 若 flag 中垂直方向为油门控制，z 表示油门百分比，取值 10.0-100.0，精度到 0.1 ➢ 若 flag 为位置控制，z 表示高度，取值-500-8000m，精度到 0.01m ➢ 若 flag 为速度控制，表示地向速度（上升为正），取值为-300~300 m/s, 精度到 0.01m/s
	yaw：double 类型，偏航方向控制量输入 ➢ 若 flag 中偏航控制为角度控制，yaw 表示偏航角度，取值 0-360deg，精度到 0.01deg ➢ 若 flag 为偏航控制为角速度控制，yaw 表示偏航角速率，取值为-300~300 deg/s，精度为 0.01deg/s
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.3.7 位置偏航控制

方法名称	positionAndYawCtrl
所属模块	drone_proxy
所属类	Drone. flyCtrl
输入	x：double 类型， 纬度，取值-90-90deg，精度到 1e-7deg
	y：double 类型，经度，取值-180-180deg，精度到 1e-7deg
	z：double 类型，高度，取值-500-8000m，精度到 0.01m
	yaw：double 类型，偏航角度，取值 0-360deg，精度到 0.01deg
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.3.8 速度偏航速率控制

方法名称	velocityAndYawRateCtrl
所属模块	drone_proxy
所属类	Drone. flyctrl

输入	x：double 类型，北向速度，取值为-300~300 m/s，精度到 0.01m/s
	y：double 类型，东向速度，取值为-300~300 m/s，精度到 0.01m/s
	z：double 类型，地向速度，取值为-300~300 m/s，精度到 0.01m/s
	yaw：double 类型，偏航角速度，取值为-300~300 deg/s，精度为 0.01deg/s
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.3.9 姿态定高控制

方法名称	attitudeAndVerPosCtrl
所属模块	drone_proxy
所属类	Drone. flyCtrl
输入	x：double 类型，滚转角度，取值为-180-180 deg，精度到 0.01deg
	y：double 类型，俯仰角度，取值为-90-90deg，精度到 0.01deg
	z：double 类型，高度，取值-500-8000m，精度到 0.01m
	yaw：double 类型，偏航角度，取值 0-360deg，精度到 0.01deg
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.3.10 退出 API 控制模式

方法名称	quit
所属模块	drone_proxy
所属类	Drone. flyCtrl
输入	无
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.4 航点任务

Apollo 支持航点任务，提供任务加载、启动、停止、暂停、恢复、继续、跳转命令，具体使用可参考例程 7。

3.1.4.1 航点任务描述

任务总体描述，结构体为 WPBasicParam。一条航线由 trackId 进行标识，一条航线可以划分为航段，每个航段由 segmentId 进行标识，每个航段可以最多 255 个航点组成，这样一个航线可支持 255 个航段*255 个航点。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	
1	trackId	航线序号	1	int	1	1-255	

2	segmentId	航段序号	1	int	1	1-255	
3	wpNumber	航点总数	1	int	1	1-255	
4	executiveTimes	执行次数	1	int	1	1-255	
5	traceMode	转弯方式	/	int	/	0: 定点转弯；1：协调转弯	
6	RCLostAction	遥控失控动作	/	int	/	0：执行失控保护 1：继续航路，结束后失控保护	

航点详细信息，由多个航点组成的队列，每个航点由结构体 WPItem 定义。

No	内容	物理值					备注
		意义	量纲	类型	精度	范围	
1	type	航点类型	/	int	/	0: 起飞点,在该点飞机执行起飞动作，可设置悬停时间 1：过路点，普通航迹过路点，在该点可设置悬停时间 2：返回点，到达该点后，飞机自动执行返航 3：着陆点，到达该点后，飞机将自动着陆	
2	latitude	航段纬度	deg	double	1e-7	-90～90	
3	longitude	航点经度	deg	double	1e-7	-180～180	
4	altitude	航点高度	m	float	0.1	-500~3000	
5	velocity	从该航点到下一航点飞行速度	m/s	float	0.01	0~600	
6	heading	从该航点到下一航点机头朝向	deg	float	0.01	机头相对于该条轨迹的朝向，0-360 度，向右为正。由于 0 和 360 表示同一方向，因此使用 0 表示保持当前航向，360 表示指向下一点	
7	hasAction	该航点是否有任务动作	/	bool	/	是否有动作，若为 true，飞机将在此悬停，直到收到“继续指令”后继续飞往下一个航点	
8	hovertime	该行点停留时间	s	int	1	若 hasAction 为 false，通过此字段设置飞机在该点悬停时间，最长时间为 3600s。	
9	returnHeading	从该航航点返航时机头朝向	deg	float	0.01	当 type 为返回点时，该字段可设置返航机头朝向，取值为 0-360，type 为其他类型是，该字段无效	

3.1.4.2 航点任务装订

方法名称	load
所属模块	drone_proxy
所属类	Drone.wpMission
输入	basicParam: WPBasicParam 类型，描述航点任务总体情况
	wpItems： WPItem[]类型，由一到多个航点组成的列表
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.4.3 启动航点任务

方法名称	start
所属模块	drone_proxy
所属类	Drone. wpMission
输入	无
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.4.4 停止航点任务

方法名称	stop
所属模块	drone_proxy
所属类	Drone. wpMission
输入	无
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.4.5 暂停航点任务

方法名称	pause
所属模块	drone_proxy
所属类	Drone. wpMission
输入	无
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.4.6 恢复航点任务

方法名称	resume
所属模块	drone_proxy
所属类	Drone. wpMission
输入	无
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.4.7 跳转航路点

方法名称	jump
所属模块	drone_proxy
所属类	Drone. wpMission
输入	wpId:int 类型，指定目的航点的编号
	immediately： bool 类型，true 表示立即跳转，false 表示飞到下一个航点后跳转
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.1.4.8 继续飞行

当航点属性中 hasAction 设置为 true 时 ，飞机会在此航点进行悬停，直到收到 next 指令后，继续飞行。

方法名称	next
所属模块	drone_proxy
所属类	Drone. wpMission
输入	nowWpId： int 类型，当前正在悬停的航点号
输出	success：bool 类型， true 表示命令执行成功，false 表示命令执行失败
	errstr： string 类型，若返回值 success 为 false，该字段描述失败的原因

3.2 远程通信

Apollo 预装了 mqtt（paho-mqtt），并在拓攻云配置了测试专用的云中心，用户可以使用 Apollo4G 远程通信能力，将数据发往云中心。关于 mqtt 的详细使用，请参考 https://pypi.python.org/pypi/paho-mqtt/1.1 。

为方便用户测试，拓攻云上提供了 mqtt 服务端，IP 地址：47.93.190.35，端口为 1883。用户也可以参考建立自己的云端（详细可参见 http://www.emqtt.com/ ）。客户可以将 Apollo 状态信息发布到云端，客户计算机可以从云端订阅相关信息（需在客户计算机安装 mqtt 客户端，例如 mosquitto，详细使用见 http://mosquitto.org/download/ ），同时客户计算机可以通过 mqtt 下达飞行命令给 Apollo，触发 Apollo 执行飞行任务。

详细使用参考例程 8 和例程 9。

3.3 外设控制

Apollo 提供了丰富的外部接口，用户可根据需求对 M1-M8 的工作模式进行灵活设置，目前支持 GPIO 输出和 PWM 输出两种模式，后续将支持 GPIO 输入和 PWM 输入模式。IO

的控制由内置的 COMCU 负责，控制精度可达微妙级。同时，Apollo 为用户提供了方便的 API 接口进行 IO 控制，详见例程 10 和 11。

3.3.1.1 设置工作模式

通过给接口用户可设定 M1-M8 口的工作模式。

方法名称	setMode
所属模块	iocu_proxy
所属类	IOCU
输入	pinId： int 类型，指定的 M1-M8 接口编号
	ioMode: int 类型，指定的工作模式，0 表示 GPIO 输出，1 表示 GPIO 输入，2 表示 PWM 输出，3 表示 PWM 输入，目前
输出	res：bool 类型， true 表示命令执行成功，false 表示命令执行失败

3.3.1.2 GPIO 输出设置

通过该接口控制 GPIO 口的电平输出。

方法名称	gpioSet
所属模块	iocu_proxy
所属类	IOCU
输入	pinId： int 类型，指定的 M1-M8 接口编号
	level: int 类型，指定的高低电平，0 表示低电平，1 表示高电平
输出	res：bool 类型， true 表示命令执行成功，false 表示命令执行失败

3.3.1.3 PWM 输出设置

通过该接口控制 PWM 口的输出。

方法名称	pwmSet
所属模块	iocu_proxy
所属类	IOCU
输入	pinId： int 类型，指定的 M1-M8 接口编号
	period: int 类型，PWM 信号周期，单位为 us，目前仅支持 20000us，后续支持频率可修改
	duty:int 类型，PWM 信号单周期内高电平持续时间，单位为 us，目前支持的最小单位为 10us
输出	res：bool 类型， true 表示命令执行成功，false 表示命令执行失败

4 快速上手

通过本章，用户可快速上手 Apollo，对 Apollo 飞行控制功能建立直观的认识。**请确保飞机处于空旷环境，保证足够的飞行空间，确保用户同飞机保持安全距离。**

参照 2.2 节，建立飞行态环境。飞机加电后，Apollo 自动启动，此时在用户计算机串口终端中能够看到 Apollo 启动信息。Apollo 启动完成后约 1 分钟（后续优化启动时间），系统会自动启动 rosout, fcu_access_node、iocu_access_server 三个服务，其中 fcu_access_node 服务负责接入 M2 FCU。

用户在计算机串口终端中进行如下操作：

a) 查看服务是否运行

输入：rosnode list

可以看到系统默认启动了三个 ROS 服务，即 rosout、fcu_access_server、iocu_access_server。其中，rosout 负责日志输出，fcu_access_server 负责接入 M2 飞控，iocu_access_server 负责外部接口管理。

b) 进入 demo 目录

输入：cd /opt/app/demo/

c) 进入 python 命令行模式

输入：python

d) 导入飞控接入代理

在 python 交互环境中输入：from drone_proxy import *

e) 创建飞控代理

在 python 交互环境中输入：drone=Drone()

f) 查看同飞控的链接状态

在 python 交互环境中输入：drone.connection()

输出 True 代表已接入飞控，False 代表尚未接入飞控。若未接入飞控请确保同飞控的串口链接正常。

g) 查看 Apollo 的 UID

在 python 交互环境中输入：drone.getApolloUID()

输出为 12 位的字符串，是该 Apollo 的唯一标识。

h) 查看电机是否解锁

在 python 交互环境中输入：drone.telemetry.getMsg("/UAV/State",1).armed

输出 False 表示电机未解锁。

i) 电机解锁

在 python 交互环境中输入：drone.unlock()

输出 success:True 表示执行成功，发现飞机电机解锁

j)　自动起飞

在 python 交互环境中输入：drone.takeoff(4)

飞机将自动起飞，起飞高度为 4 米。若命名执行报错，请查看 6.1 节。

k)　查看位置信息

在 python 交互环境中输入：drone.telemetry.getMsg("/UAV/Position",1)

将获得飞机当前位置信息，高度在 3.5m—4.5m 范围内。

l)　自动降落

在 python 交互环境中输入：drone.land()

飞机将自动降落。

5　例程

请确保飞机处于空旷环境，保证足够的飞行空间，确保用户同飞机保持安全距离。

5.1　例程 1—参数查询

5.1.1　概述

通过本例程，掌握获取飞控 UID、飞控基本参数的方法。参照 2.2 节，建立飞行态环境。飞机加电后，使用串口终端登录 Apollo，进行如下操作：

1）进入 demo 目录: cd　/opt/app/demo

2）执行用例：python test_basic.py

输出如下：

链接状态：　True
ApolloUID= aabee0e68251
M2FcuUID= 0030001d3136510c30373736
FCUBasicParams= type: 1
maxHDipAngle: 35
maxHAngleVelo: 150
maxYawRate: 120
maxFlyHeight: 500.0
maxVVelo: 655.349975586
maxHVelo: 6.0

5.1.2 代码解读

1）导入 python 模块，并创建飞控接入代理

```
#导入 m2fcu 接入代理
from drone_proxy import *
#创建 Drone
drone=Drone()
```

2） 查看同飞控的链接状态

```
print "链接状态：",drone.connection()
```

3）调用 getApolloUID 方法获取 Apollo 的 UID

```
#获取 Apollo 的 UID
apollo_uid=drone.getApolloUID()
print "ApolloUID=",apollo_uid
```

4）调用 getM2FcuUID 方法获取飞控 UID

```
ret=drone.getM2FcuUID()
print "M2FcuUID=", str(ret.uid)
```

5）调用 getFCUBasicParams 方法，获取飞控基本配置

```
ret=drone.getFCUBasicParams()
print "FCUBasicParams=",ret.params
```

5.2 例程 2—电机加解锁

5.2.1 概述

通过本例程，熟悉电机加解锁操作。参照 2.2 节，建立飞行态环境。 飞机加电后，使用串口终端登录 Apollo，进行如下操作：

1） 进入 demo 目录: cd /opt/app/demo

2） 执行用例：python test_lock.py

5.2.2 代码解读

1）导入 python 模块，并创建飞控接入代理

```
#导入 m2fcu 接入代理
from drone_proxy import *
#创建 Drone
drone=Drone()
```

2）查看当前电机是否解锁

使用 getMsg 方法（见 4.1.2.2），返回值为 none 表示获取失败。若获取成功，打印飞行

状态（见 4.1.2.1.12）中的 armed 字段。

```
state = drone.telemetry.getMsg("/UAV/State",5)
if state == "none":
    print "获取状态失败"
else:
    print "解锁状态:",state.armed
```

3）调用 unlock 方法进行解锁

```
ret = drone.unlock()
print "进行解锁: ",ret.success
time.sleep(5)
```

4）调用 lock 方法进行加锁

```
ret = drone.lock()
print "进行加锁: ",ret.success
```

5.3 例程 3—即点即飞与自动返航

5.3.1 概述

通过本例程，用户可熟悉基本的飞行控制和遥测数据主动获取方法。本例程对飞机解锁后，进行自动起飞，然后进行即点即飞，最后进行自动返航，整个过程周期性查询飞机飞行状态。

参照 2.2 节，建立飞行态环境。飞机加电后，使用串口终端登录 Apollo，进行如下操作：

1） 进入 demo 目录: cd　/opt/app/demo

2） 执行用例：python test_flyto.py

现象：飞机自动起飞，起飞高度为 4 米；向北飞行 0.0003deg，大约 17 米；然后进行自动返航。

5.3.2 代码解读

1） 导入模块，创建飞控接入代理，电机解锁

```
# -*- coding: utf-8 -*-
import time
#导入 m2fcu 接入代理
from drone_proxy import *
#创建 Drone
drone=Drone()
#电机解锁
ret = drone.unlock()
```

2) 重新设置返航高度

```
#获取返航高度
ret = drone.getReturnHeight()
print "返航高度：",ret.height
#设置返航高度 4m
drone.setReturnHeight(4)
ret = drone.getReturnHeight()
print "返航高度变为：",ret.height
```

3）飞机解锁

```
#电机解锁
ret = drone.unlock()
print "进行解锁:",ret.success
#等待 5s
time.sleep(5)
```

4）自动起飞

```
#自动起飞
ret = drone.takeoff(4)
print "下达起飞命令：",ret.success
```

5）自动起飞并每隔两秒检查飞行状态，当自动起飞完成后，自动进入 LOITER 模式。

```
#每隔 2s 查询下飞行状态,最多检测 300s
has_enter_takeoff = False #尚未进入 TAKEOFF 模式
for i in range(150):
    state = drone.telemetry.getMsg("/UAV/State",1) #主动查询飞行状态
    if  isinstance(state,State): #如果获取到飞行状态
        print "当前飞行状态为：", state.mode
        if state.mode == "TAKEOFF": #处于 TAKEOFF 状态
            has_enter_takeoff = True
            if state.takeoff.status is 0:
                print "当前正在起飞..."
            else:
                print "已到达指定高度"
        if state.mode == "LOITER" and has_enter_takeoff:
            print "自动起飞完成，进入 LOITER 模式"
            break
    else: #无法获取飞行状态
        print "获取飞行状态失败"
    time.sleep(2)
```

6） 查询当前位置，如果成功则执行即点即飞，如果失败则进行降落

```
#获取当前点位
pos = drone.getMsg("/UAV/Position",2) #获取当前经纬高
if isinstance(pos,Position) #成功获取位置信息
   执行即点即飞
```

```
else:
    执行降落
```

7) 执行即点即飞，北向前进 0.0003 度，并每隔 2s 查询飞行状态，完成即点即飞任务后自动进入 LOITER 模式

```
print "当前飞机位置,纬度=",pos.latitude,"经度=",pos.longitude,"高度
=",pos.altitude
    #向北飞行.0003 度
    ret = drone.flyto(pos.latitude+0.0003,pos.longitude,pos.altitude)
    print "开始进入即点即飞模式，飞往
(",pos.latitude+0.0003,pos.longitude,pos.altitude,"): ",ret.success

    #每隔 2s 查询下飞行状态,最多检测 120s
    has_enter_guided = False
    for i in range(60):
        state = drone.telemetry.getMsg("/UAV/State",1)
        if  isinstance(state,State):
            print "当前飞行状态为: ", state.mode
            if state.mode == "GUIDED":
                has_enter_guided = True
                if state.guid.status is 0:
                    print "当前正在飞往目标点..."
                else:
                    print "已到达目标点"
            if state.mode == "LOITER" and has_enter_guided:
                print "已到达目标点，自动进入 LOITER 模式"
                break
        else:
            print "获取飞行状态失败"
        time.sleep(2)
```

8) 完成即点即飞后，执行自动返航，每隔 2s 查询飞行状态，完成着落后，state.landed 状态为 true

```
ret = drone.returnHome(0)
    print "开始自动返航:",ret.success

    #每隔 2s 查询下飞行状态,最多检测 300s
    for i in range(150):
        state = drone.telemetry.getMsg("/UAV/State",1)
        if  isinstance(state,State):
            print "当前飞行状态为: ", state.mode
            if state.mode == "RETURN":
                print "自动返航详细状态为",state.rtl.status
            if state.landed:
```

```
                print "已成功着陆"
                break
        else:
            print "获取飞行状态失败"
        time.sleep(2)
```

9) 如果无法获取位置，直接进行降落

```
    print "获取飞机位置失败，直接降落"
    ret = drone.land()
    print "下达降落指令：",ret.success
```

5.4 例程 4—订阅遥测数据

5.4.1 概述

本例程主要演示通过异步方式订阅接收遥测数据，并根据飞行状态，发送飞行指令，调度飞机进行解锁、起飞、即点即飞、自动返航等操作。

首先参照 2.2 节，建立飞行态环境。飞机加电后，使用串口终端登录 Apollo，进行如下操作：

1） 进入 demo 目录: cd /opt/app/demo

2） 执行用例：python test_flyto2.py

现象：飞机自动起飞，起飞高度为 4 米；向北飞行 0.0003deg，大约 17 米；然后进行自动返航。

5.4.2 代码解读

1） 定义位置回调函数，专门处理遥测数据中的位置信息。获得位置信息后，将位置记录到 current_pos 中。

```
current_pos  = Position()
def process_position(data):
    if isinstance(data,Position):
        global current_pos
        current_pos = data
```

2） 定义飞行状态回调函数，根据当前的飞行状态，执行相应动作。在回调函数中主要进行如下工作：

a) 如果飞机未解锁，进行解锁

b) 如果飞机已解锁未起飞，进行起飞

c) 如果飞机起飞结束，进行即点即飞

d) 如果完成即点即飞，则返航

e) 如果降落了，则退出监听

```
hasflyto = False
lastmode = ""
def process_flystate(data):
    if isinstance(data, State):
        global hasflyto
        global current_pos
        global home_pos
        global lastmode
        if not data.armed and not hasflyto: #飞机未解锁
                ret = drone.unlock()
            print "进行解锁:",ret.success
        if data.armed and data.landed: #飞机已解锁，未起飞
            ret = drone.takeoff(4)
            print "下达起飞命令，起飞高度 4m:", ret.success
        if data.mode == "TAKEOFF":  #飞机起飞中
            print "当前处于自动起飞状态，状态为", data.takeoff.status
        if data.mode == "LOITER" and lastmode == "TAKEOFF": #从起飞模式进
入到 LOITER 模式
            if not hasflyto: #尚未执行即点即飞
                    print "完成启动起飞，开始进行即点即飞"
                home_pos = current_pos
                ret = drone.flyto(home_pos.latitude + .0003,
home_pos.longitude, home_pos.altitude)
                print "开始即点即飞，飞往(",home_pos.latitude + .0003,
home_pos.longitude,"):",ret.success
                hasflyto = True
        if data.mode == "GUIDED": #即点即飞中
            print "飞往目标点..."
        if data.mode == "LOITER" and lastmode == "GUIDED":
                print "已到达目标点"
            ret = drone.returnHome(0)
            print "开始自动返航：",ret.success
        if lastmode == "RETURN" and data.landed:
                print "完成自动返航，已成功着陆"
            drone.telemetry.stop()
        lastmode = data.mode
```

3） 创建 drone，设置返航高度，注册遥测数据处理函数并启动数据接收

```
drone=Drone()
#获取返航高度
ret = drone.getReturnHeight()
print "返航高度：",ret.height
```

```
#设置返航高度 4m
drone.setReturnHeight(4)
ret = drone.getReturnHeight()
print "返航高度变为: ",ret.height
tm = {"/UAV/State": process_flystate,
    "/UAV/Position":process_position}
drone.telemetry.listen(tm)
drone.telemetry.start()
```

5.5 例程 5—速度控制—螺旋上升

5.5.1 概述

本例程展示 Apollo 高级飞行控制中的速度控制能力，实现飞机螺旋上升，总飞行高度在 20 米左右，飞行半径约 10 米。

首先参照 2.2 节，建立飞行态环境。飞机加电后，使用串口终端登录 Apollo，进行如下操作：

1） 进入 demo 目录: cd /opt/app/demo

2） 执行用例：python test_veloctrl.py

现象：飞机解锁后，以 1m/s 速度上升，10 秒后以 3m/s 水平速度，0.2m/s 垂直速度进行螺旋上升，上升三圈后，悬停 30 秒，然后自动着落，飞机飞行半径约 10m。

5.5.2 代码解读

1） 创建飞控接入代理，并进入速度控制模式

```
from drone_proxy import *
from math import cos, sin,pi

drone = Drone()
ret = drone.unlock()
print "解锁",ret
time.sleep(1)
ret = drone.flyCtrl.enterVelocityAndYawRateCtrl()
print "进入速度控制模式",ret
```

2） 通过速度控制，以垂直方向 1m/s 的速度进行起飞，保持 10s

```
#起飞，给定垂直速度 1m/s，持续 10s
drone.flyCtrl.velocityAndYawRateCtrl(0,0,1,0)
time.sleep(10)
```

3） 进行盘旋上升，垂直速度 1m/s，水平速度 3m/s，总共盘旋 3 圈，每个速度保持 0.1s，

环绕半径为 10m。

```
hv = 3.0 #水平速度 3m/s
r = 10.0 #环绕半径 10m
interval = 0.1 #每 0.1 秒调整一次速度
num = int(2*pi*r/(interval*hv)) #每圈的控制点
pa = 2*pi/num #每个控制点的角度

for i in range(3): #盘旋上升三圈
    print "当前盘旋圈数:", i
    for j in range(num):
        angle = pa*j
        hv_x = hv*cos(angle)
        hv_y = hv*sin(angle)
        print "水平速度控制量为：",hv_x,hv_y
        drone.flyCtrl.velocityAndYawRateCtrl(hv_x,hv_y,0.2,0)
        time.sleep(0.1)
```

4） 保持悬停 30s 后，退出速度控制模式，并进行降落

```
drone.flyCtrl.velocityAndYawRateCtrl(0,0,0,0)
print "悬停 30s"
time.sleep(30)
drone.flyCtrl.quit()
drone.land()
```

5.6 例程 6—速度控制—8 字飞行

5.6.1 概述

本例程展示 Apollo 高级飞行控制中的速度控制能力，实现飞机 8 字飞行，飞行高度在 10 米左右，东西长约 40 米。

首先参照 2.2 节，建立飞行态环境。飞机加电后，使用串口终端登录 Apollo，进行如下操作：

3） 进入 demo 目录: cd /opt/app/demo

4） 执行用例：python test_veloctrl2.py

现象：飞机解锁后，上升到 10m 左右，然后从西向东 8 字飞行，并自动着陆。

5.6.2 代码解读

1） 创建飞控接入代理，并进入速度控制模式

```
from drone_proxy import *
```

```
from math import cos, sin,pi

drone = Drone()
ret = drone.unlock()
print "解锁",ret
time.sleep(1)
ret = drone.flyCtrl.enterVelocityAndYawRateCtrl()
print "进入速度控制模式",ret
```

2） 通过速度控制，以垂直方向 1m/s 的速度进行起飞，保持 15s

```
#起飞，给定垂直速度 1m/s，持续 10s
drone.flyCtrl.velocityAndYawRateCtrl(0,0,1,0)
time.sleep(15)
```

3） 水平速度 3m/s，从西向东绕圈飞行，每个速度保持 0.1s，环绕半径为 10m。

```

hv = 3.0 #水平速度 3m/s
r = 10.0 #环绕半径 10m
interval = 0.1 #每 0.1 秒调整一次速度
num = int(2*pi*r/(interval*hv)) #每圈的控制点
pa = 2*pi/num #每个控制点的角度

for j in range(num):
    angle = pa*j
    hv_x = hv*cos(angle)
    hv_y = hv*sin(angle)
    if  j > num/2:
        hv_y = -hv_y
    print "水平速度控制量为：",hv_x,hv_y
    drone.flyCtrl.velocityAndYawRateCtrl(hv_x,hv_y,0,0)
    time.sleep(0.1)

for j in range(num):
    angle = pa*j
    hv_x = hv*cos(angle)
    hv_y = -hv*sin(angle)
    if  j > num/2:
        hv_y = -hv_y
    print "水平速度控制量为：",hv_x,hv_y
    drone.flyCtrl.velocityAndYawRateCtrl(hv_x,hv_y,0,0)
    time.sleep(0.1)
```

4） 保持悬停 20s 后，退出速度控制模式，并进行降落

```
drone.flyCtrl.velocityAndYawRateCtrl(0,0,0,0)
print "悬停 20s"
time.sleep(20)
```

```
drone.flyCtrl.quit()
drone.land()
```

5.7 例程 7—航点任务

5.7.1 概述

本例程展示 Apollo 航点任务飞行能力。参照 2.2 节，建立飞行态环境。飞机加电后，使用串口终端登录 Apollo，进行如下操作：

1） 进入 demo 目录: cd /opt/app/demo

2） 执行用例：python test_waypoint.py

现象：飞机解锁后，自动起飞，起飞高度 4m；向北方前进 0.0003deg，约 17m；向东飞行 0.0001deg，约 11m；然后向南飞行 0.0003deg，约 34m；然后自动返航。

5.7.2 代码解读

1） 解锁并获取当前飞机位置

```
drone = Drone()
ret = drone.unlock()
print "解锁:", ret.success
#获取飞机初始位置
ori_pos = drone.telemetry.getMsg("/UAV/Position",2)
```

2） 若成功解锁并获取飞机位置，可以进入航点任务

```
#解锁成功并成功获取当前位置信息，才能执行航点任务
if ret.success and  isinstance(ori_pos,Position):
```

3） 航点总体描述

```
    #航点任务基本描述
    bp = WPBasicParam()
    bp.trackId = 1 #航线编号
    bp.segmentId = 1 #航段编号
    bp.wpNumber = 4 #总共 3 个航点
    bp.executiveTimes = 1 #仅执行一次
    bp.traceMode = 0 #转弯模式为定点转弯
    bp.RCLostAction = 0 #RC 失联后执行失联保护
```

4） 添加第一个航点，为起飞点，起飞高度为 4，悬停 20s，起飞速度为 2m/s

```
#添加航点
    wps = []
    #第一个航点为起飞点，起飞悬停 20s
    wp1 = WPItem()
```

```
    wp1.latitude = ori_pos.latitude
    wp1.longitude = ori_pos.longitude
    wp1.altitude = ori_pos.altitude + 4
    wp1.heading = 360 #离开该航点时机头指向下一航点
    wp1.velocity = 2  #离开该航点时速度为 2m/s
    wp1.type = 0  #起飞点
    wp1.hovertime = 20 #悬停 20s
    wps.append(wp1)
```

5) 添加第二个航点，为过路点，北向 0.0003 度，速度为 2m/s，不悬停

```
    #第二个航点为 PASS，在北向 0.0003 方向，不悬停
    wp2 = WPItem()
    wp2.latitude = ori_pos.latitude + 0.0003
    wp2.longitude = ori_pos.longitude
    wp2.altitude = ori_pos.altitude + 4
    wp2.heading = 360 #离开该航点时机头指向下一航点
    wp2.velocity = 2  #离开该航点时速度为 2m/s
    wp2.type = 1  # PASS 点
    wp2.hovertime = 0 #不悬停
    wps.append(wp2)
```

5） 添加第三个航点，为过路点，北向.0003 度，东向 0.0001 度，悬停 1 分钟

```
    #第三个航点为 PASS，在北向 0.0003,东向 0.0002,悬停 60s
    wp3 = WPItem()
    wp3.latitude = ori_pos.latitude + 0.0003
    wp3.longitude = ori_pos.longitude + 0.0002
    wp3.altitude = ori_pos.altitude + 4
    wp3.heading = 360 #离开该航点时机头指向下一航点
    wp3.velocity = 2  #离开该航点时速度为 2m/s
    wp3.type = 1  # PASS 点
    wp3.hovertime = 20 #悬停
    wps.append(wp3)
```

6） 添加第四个航点，为返航点，南向 0.0003 度，东向 0.0001 度

```
    #第四个点为返航点，南向 0.0003,东向 0.0002
    wp4 = WPItem()
    wp4.latitude = ori_pos.latitude - 0.0003
    wp4.longitude = ori_pos.longitude + 0.0002
    wp4.altitude = ori_pos.altitude + 4
    wp4.heading = 360 #离开该航点时机头指向下一航点
    wp4.velocity = 2  #离开该航点时速度为 2m/s
    wp4.type = 2 # 自动返航点
    wp4.returnHeading = 360
    wps.append(wp4)
```

7） 加载并启动航点任务

```
#装订航点
    ret = drone.wpMission.load(bp, wps)
    print "航点装订:",ret.success
    if ret.success: #装订成功
        ret = drone.wpMission.start()
        print "启动航点:",ret.success
```

8） 查看飞行状态执行情况

```
while True:
        #查询飞行状态
        state = drone.telemetry.getMsg("/UAV/State",1)
        if state.landed:
            print "飞机已着陆"
            break
        if state.mode == "WAYPOINT":
            if state.waypoint.status is 0:
                print "执行航点任务中，正飞往",
state.waypoint.waypointId
            else:
                print "执行航点任务中，已到达",
state.waypoint.waypointId
        if state.mode == "RETURN":
            print "返航中..."
        time.sleep(1)
```

5.8 例程 8—基于 4G 的位置上报

5.8.1 概述

本例程展示 Apollo 远程通信能力，可以将当前位置通过 4G 链路上报云端。其中 mqtt 的服务端 IP 地址 47.93.190.35，端口 1883；为方便客户观察效果，我们提供了 web 客户端去订阅 Apollo 发布的消息。

云端：

1） 登录 mqtt　web 客户端。http://47.93.190.35:9900，并输入您 Apollo 的 UID

2） 登陆后，客户端自动订阅该 Apollo 的位置信息。

Apollo：

参照 2.2 节，建立飞行态环境。飞机加电后，使用串口终端登录 Apollo，进行如下操作：

1） 进入 demo 目录: cd /opt/app/demo

2） 执行用例：python test_reportposition.py

现象：飞机将上报当前位置，mqtt web 客户端显示上报的位置信息。

5.8.2 代码解读

1） 获取飞机位置，构建位置上报字符串

```
# -*- coding: utf-8 -*-
#导入 m2fcu 接入代理
from drone_proxy import *
import paho.mqtt.publish as publish

#创建 Drone
drone=Drone()
#获取唯一标识 UID
uid = drone.getApolloUID()
#位置主题
topic = "/Apollo/"+str(uid)+"/position"
```

2) 使用 mqtt 进行位置上报，消息主题为"Apollo/uid/position"

```
#发布消息
#获取位置信息
pos = drone.telemetry.getMsg("/UAV/Position",5)
if pos == "none":
    print "get position err"
else:
    #发布消息
    msg = str(pos.latitude) + "," + str(pos.longitude) + "," +
str(pos.altitude)
    publish.single(topic, msg, hostname="47.93.190.35", port=1883,qos=0)
```

5.9 例程 9—基于 4G 的远程控制

5.9.1 概述

本例展示 Apollo 的远程控制能力，用户可以在云端通过 4G 对飞机进行远程控制。

Apollo：

参照 2.2 节，建立飞行态环境。飞机加电后，使用串口终端登录 Apollo，进行如下操作：

1） 进入 demo 目录: cd /opt/app/demo

2） 执行用例：python test_remotctrl.py

显示“成功连接云端 0”，表明您的 Apollo 已接入云，可以进行远程控制。

云端：

1） 登录 mqtt　web 客户端。http://47.93.190.35:9900，并输入您 Apollo 的 UID

2） 在 web 页面上用户可以点击解锁按钮对飞机进行解锁

3） 点击起飞按钮，远程控制飞机起飞

4） 点击降落按钮，远程控制飞机降落

5） 点击结束按钮，让 Apollo 退出本例程。

5.9.2 代码解读

1) 创建飞控接入代理，并查询 UID

```
#-*-coding:utf-8-*-
#创建 Drone
from drone_proxy import *
drone=Drone()
#获取唯一标识 UID
uid = drone.getApolloUID()
import paho.mqtt.client as mqtt
client = mqtt.Client(client_id=str(uid))
```

2) 定义连接回调函数，当成功连接云后，自动调用该函数，订阅主题为 Apollo/uid/cmd 的消息

```
# 当连接上服务器后回调此函数
def on_connect(client, userdata, flags, rc):
    print("成功连接云端 "+str(rc))
    # 放在 on_connect 函数里意味着
    # 重新连接时订阅主题将会被更新
    topic = "/Apollo/"+str(uid)+"/cmd"
    client.subscribe(topic)
```

2） 定义消息处理函数，收到云端发来的命令后进行执行，支持的命令包括电机解锁 unlock、起飞 takeoff(默认起飞高度为 4m)、降落 land、退出 exit。

```
# 从服务器接受到消息后回调此函数
def on_message(client, userdata, msg):
    print("主题:"+str(msg.topic)+" 消息:"+str(msg.payload))
    if msg.payload == "unlock":
        ret = drone.unlock()
        print "飞机解锁:",ret.success
    if msg.payload == "takeoff":
        ret = drone.takeoff(4)
        print "飞机起飞:",ret.success
    if msg.payload == "land":
        ret = drone.land()
        print "飞机降落:",ret.success
```

```
    if msg.payload == "exit":
        client.disconnect()
        print "exit....."
```

3） 创建客户端并连接云端

```
client.on_connect = on_connect #设置连接上服务器回调函数
client.on_message = on_message  #设置接收到服务器消息回调函数
client.connect("47.93.190.35", 1883, 60)  #连接服务器,端口为 1883,维持心跳为 60 秒
client.loop_forever()
```

5.10 例程 10—控制 GPIO 输出

5.10.1概述

本例程可在开发态进行，用户登录 Apollo 后，进入 demo 目录执行 test_gpio.py 即可控制 GPIO。

5.10.2代码解读

```
# -*- coding: utf-8 -*-
from iocu_proxy import *
import time
#创建 io 控制类
io = IOCU()
#1 口工作模式设置为 GPIO 输出
io.setMode(1,0)
for i in range(10):
    #1 口输出高电平
    io.gpioSet(1,1)
    time.sleep(1)
    #1 口输出低电平
    io.gpioSet(1,0)
    time.sleep(1)
```

5.11 例程 11—控制 PWM 输出

5.11.1概述

本例程可在开发态进行，用户登录 Apollo 后，进入 demo 目录执行 test_pwm.py 即可控制 PWM 输出。

5.11.2代码解读

```
# -*- coding: utf-8 -*-
from iocu_proxy import *
import time
#创建 io 控制类
io = IOCU()
#1 口工作模式设置为 PWM 输出
io.setMode(1,2)
#输出 PWM，周期 20000us，高电平持续时间 1000us
io.pwmSet(1,200000,1000)
time.sleep(10)
#停止输出
io.pwmSet(1,0,0)
```

6 常见问题

6.1 命令执行错误提示

执行命令的返回值 errstr 有以下几种情况：

errstr	意义	原因
cmd not support	指令不支持	该指令尚不支持
crc err	校验失败	飞控收到报文后发现数据校验失败，表明传输过程中出错
cmd wrong range	指令内容超范围	
gps critical	GPS 信号差	由于 GPS 信号差无法执行航点任务、自动起飞、自动悬停、自动返航、即点即飞
altitude too low	高度过低	飞行高度超过高度限制的最小值
altitude too high	高度过高	飞行高度超过高度限制的最大值
distance too far	距离过远	——
imu critical	IMU 状态差	——
barometer critical	气压计状态差	——
magnetic compass critical	磁罗盘状态差	——
rtk critial	RTK 位置精度差	——
rtk head critical	RTK 航向精度低	——
voltage too low	电池电压低	——
optical flow critical	光流信号质量差	——
drone in air	飞行器已在空中	飞机已在空中，无法执行电机加解锁和自动起飞指令
drone on ground	飞行器已着陆	飞机已着陆，无法执行自动返航、自动降落指令
not in gps mode	未在 GPS 模式	——

drone above home	飞行器在 Home 点上空	已在 Home 点上空时，无法执行自动返航指令
wap point mission has not upload	航点未上传	执行航点任务时，航点没有上传
way point too close	航点距离太近	上传的航点之间距离太近
not in navigation mode	未在航点跟踪模式	未在航点模式下，无法执行航点暂停、恢复、停止等指令
not in way point pause	未在航线暂停状态	航点任务未暂停，不支持航点任务继续指令
this way point not in the mission	航点未在队列中	跳转航路点不在航点任务中
not in target mode	未在目标飞行模式	——
way point speed too fast	航点速度过大	——
motor has already armed	电机已解锁	电机已解锁，不支持再次解锁
motor not armed	电机未解锁	电机未解锁，不支持其他飞行控制指令
throttle input too high	油门输入未达最低	解锁、自动起飞时需保证油门至于最低位
can not send this cmd to fcu	无法发送指令到飞控	Apollo 同飞控的串口链路尚未联通
cmd param is invalid	参数非法	命令参数非法

6.2 飞行态下使用串口终端执行命令返回值输出不全

飞行态下，用户计算机同 Apollo 之间采用数传进行通信，数传支持的单个数据包大小在 400 字节左右，当命令返回值大于 400 字节时，返回值字符串会被截断。

6.3 飞控接入服务运行管理及日志查看

查看飞控接入服务运行状态：在终端中输入 service drone_access status

启动飞控接入服务：在终端中输入 service drone_access start

停止飞控接入服务：在终端中输入 service drone_access stop

重启飞控接入服务：在终端中输入 service drone_access restart

飞控接入服务日志文件放在/var/log/upstart/drone_access.log 下，可以使用 tail -f 文件名进行查看。