```
+void MavlinkReceiver::handle_message_ca_trajectory_msg(mavlink_message_t *msg)
+{
+       /* ca_trajectory */
+       mavlink_ca_trajectory_t traj;
+       mavlink_msg_ca_trajectory_decode(msg, &traj);
+
+       struct ca_traj_struct_s f;
+       memset(&f, 0, sizeof(f));
+
+       f.timestamp = hrt_absolute_time();
+       f.seq_id = traj.seq_id;
+       f.time_start_usec = traj.time_start_usec;
+       f.time_stop_usec = traj.time_stop_usec;
+       for(int i=0;i<28;i++)
+               f.coefficients[i] = traj.coefficients[i];
+
+       if (_ca_traj_msg_pub<= 0) {
+               _ca_traj_msg_pub = orb_advertise(ORB_ID(ca_trajectory_msg), &f);
+       } else {
+               orb_publish(ORB_ID(ca_trajectory_msg), _ca_traj_msg_pub, &f);
+       }
+}

MavlinkReceiver::handle_message(mavlink_message_t *msg)
 {
        switch (msg->msgid) {
        case MAVLINK_MSG_ID_COMMAND_LONG:
               handle_message_command_long(msg);
            .
            .
            .
            .

       case MAVLINK_MSG_ID_MANUAL_CONTROL:
              handle_message_manual_control(msg);
              break;

+      case MAVLINK_MSG_ID_CA_TRAJECTORY:
+             handle_message_ca_trajectory_msg(msg);
+             break;
       default:
              break;
       }
```

系统启动控制

参考链接：http://pixhawk.org/dev/system_startup

PX4 的启动是由位于“/etc/init.d/rcS”下的启动脚本(startup script)来控制的，这个脚本位于被编译到固件中的 ROM 文件系统中。这个脚本检测可用的硬件，加载硬件驱动，并且根据你的设置启动系统正常运行所需的所有 app(任务软件，包括位置和姿态估计，位置和姿态控制，遥测等)。

所有属于自动启动程序的脚本文件可以在固件源程序的如下目录中找到：ROMFS/px4fmu_common/init.d .

这个 rcS 脚本执行如下步骤：

1. 挂载 SD 卡；
2. 如果 SD 中有 etc/rc.txt 文件，则按照该启动脚本文件进行启动；
3. 如果 SD 卡中没有 etc/rc.txt 文件，下面的自动启动程序将会被执行：
    a) 如果 SYS_AUTOSTART 变量被设置为一个有效的配置，预定义的模型配置（ predefined model configuration ）之一将会被加载。模型设置包含一系列的环境变量，后面将会详细列出。
    b) 如果 SD 卡中有 etc/config.txt 文件，它将被执行。这个文件允许对部分或者整个预定义的模型配置参数进行覆盖。
    c) 实际的启动 app 将会根据模型配置定义的环境变量被执行。
    d) 如果 SD 卡中 etc/extras.txt 文件，它将被执行。这个文件允许在自动启动程序被执行完成后启动用户的 app（ 如遥测)。

注意 :在 SD 卡中创建 etc/rc/txt 文件将会使得被编译在固件内部的启动文件不再执行。由于创建这个文件后你必须亲自启动所有的基本系统进程，所以除非你是知道自己在做什么的高级用户，否则请不要创建这个文件。在大多数情况下，你只需要通过创建 etc/config.txt 和（或者）etc/extras.txt 文件来改变自动启动程

序（这样就不需要担心会影响到系统必备程序的启动）。

注意：启动脚本文件中每行最多允许有 80 个字符。任何超过 80 个字符的行都有可能造成启动出错。

预编译模型配置（Predefined Model Configurations）：下面是针对一些可以买到的 ARF/RTF 模型的预定义的模型配置。要使用这些模型配置中的一个，只需要设置环境变量 SYS_AUTOSTAET = <AUTOSTART_ID>。如果要将与机架相关的参数（如姿态控制参数：MC_XXX）设置为该模型配置默认的参数，将参数 SYS_AUTOCONFIG = 1。保存参数并重新启动飞控。这些配置参数将生效，然后 SYS_AUTOCONFIG 将自动变回 0。选择的模型配置可以通过 etc/config.txt 来进行覆盖。

下表是 Autostart ID 与模型的对应关系：

Autostart ID	Model
1001	HIL Quadcopter X
1003	HIL Quadcopter +
3031	Phantom FPV Flying Wing
3033	Wing Wing (aka Z-84) Flying Wing
3034	FX-79 Buffalo Flying Wing
4008	AR.Drone mechanical frame
4010	DJI Flamewheel F330 Quadcopter
4011	DJI Flamewheel F450 Quadcopter
10015	Team Blacksheep Discovery Quadcopter
10016	3DR Iris Quadcopter

配置变量（Configuration Variables）：下表中列出的变量定义哪个 app 会被启

动 ,哪个硬件将被使用 ,以及一些别的参数。它们可以通过 SD 卡中的 etc/config.txt 文件进行设置。如果 SYS_AUTOSTART 变量被设置为一个有效的配置 ,etc/config.txt 可以用来调整这些配置，比如设置 PWM 范围。

Environment variable	Possible values (default is underlined)	Description
`VEHICLE_TYPE`	`mc` = multicopter, `fw` = fixed wing, `none` = non-flying configuration without controllers running	Type of vehicle
`MIXER`	For multicopters: `FMU_quad_x`,`FMU_quad_+`,`FMU_quad_w`,`FMU_hex_x`,`FMU_hex_+`, …	Mixer to use. See section 'Output mixers' below.
	For fixed wings: `FMU_AERT`, `FMU_AET`,`FMU_Q`	
`USE_IO`	`yes`, `no`	If the PX4IO should be used. Set to `no` if using standalone PX4FMUv1 setup.
`OUTPUT_MODE`	`io`, `fmu`, `mkblctrl`	Control output mode, default value depends on `USE_IO` variable.
`PWM_OUTPUTS`	List of channels, e.g. `1234`, default is `none`	Channels the following PWM parameters get applied to. Please note that certain channels are grouped and need to be changed together as documented here!
`PWM_RATE`	Frequency in Hz, default is `50`	PWM output rate. To improve ESC latency on multicopter setups, it's recommended to use 400 Hz if ESCs support it.
`PWM_DISARMED`	Signal length in microseconds, default is `0`	PWM output value for disarmed state. `0` (default value) means no

		PWM signal, it's safe, but many ESC models will beep “no signal” after disarm, so it can be set to lower limit of PWM range
`PWM_MIN`	Signal length in microseconds, default is `1000`	PWM value for minimum throttle. Multicopter props **MUST** spin at low idle speed; this is necessary to prevent full stop of props during flight and warn that the system is armed!
`PWM_MAX`	Signal length in microseconds, default is `2000`	PWM value for maximum throttle.
`FMU_MODE`	`pwm`, `gpio` For FMUv1 also: `serial`, `gpio_serial`,`pwm_serial`, `pwm_gpio`	FMU output mode, e.g. set to `serial` to get debug console on PX4FMUv1 USART2.
`HIL`	`yes`, `no`	HIL (Hardware In the Loop) mode, this will force OUTPUT_MODE to fake HIL output, disable logging and GPS

输出混控（Output Mixers）:输出混控定义了控制器的输出如何映射为马达和舵机控制输出。所有被编译在固件中的混控文件位于 ROM 文件系统的 /etc/mixers 文件夹中。

可以使用一个自己修改的混控。只需要将你自己写的混控文件（mixer file）放到 SD 卡的 etc/mixers 文件夹下即可。比如，使用用户自己设置的混控文件来控制一架已经被定义为 MIXER = FMU_Q 的固定翼飞机，那么 SD 卡中的混控文件必须被命名为 FMU_Q.mix。之后 SD 卡中的这个文件将会覆盖掉被编译到固件中的那个混控文件。

用户模型配置（Custom Model）例子:

配置一个如下型号的四轴：“+”型、电调接 PX4IO 板，PWM 范围为 1100~1900，那么 config.txt 文件包含如下内容：

```
# Generic Quadcopter +
set VEHICLE_TYPE mc
set MIXER FMU_quad_+
# PX4IO PWM output will be used by default
set PWM_OUTPUTS 1234    # Change parameters for the first 4 outputs
set PWM_RATE 400        # Set PWM rate to 400 Hz for better performance
set PWM_DISARMED 900    # Motors should stop at this PWM value
set PWM_MIN 1100    # Motors should spin at low, idle speed at this PWM value
set PWM_MAX 1900        # Motors should spin at max speed at this PWM value
```

系统启动控制调试（**Start Script Debugging**）

参考链接：http://pixhawk.org/dev/start_script_debugging

PX4 的软件是在不断进化的，因此有时系统启动方式需要进行修改。我们做到使得这些更改最小。如果启动脚本不执行，系统将直接进入调试（debug）状态。

自动启动脚本是由开源社区共同维护的，但是，如果的 microSD 卡中有自定义的启动脚本，你需要在每次软件进化后更新它们。

基本调试技巧：

固件中有名为“test”的测试命令，它可以用来帮助列出所有测试项目，并且很有利于隔离问题。比如 test sensor 用来测试传感器。单独的驱动（driver）也有测试，大多数情况下是一个状态指令（status command）比如：lsm303d status）。

microSD 中的用户脚本

- 如果你想要添加单独的 app，使用 EXTRAS_FILE，并将其放入 SD 卡中的/etc/extras.txt 文件中。它能覆盖掉原有平台参数。
- 如果你确实需要进行一个完全的自定义启动，在 SD 卡的/etc/rc.txt 中放置 RC_FILE。只有在你知道自己在干什么的情况下才可以做。

调试（Debugging）

下面的调试例子是针对 rc.txt 的，但是它也适用于其他类似的文件。

1. 如果 SD 卡中有/etc/rc.txt，将它重命名为/etc/rc_test.txt；
2. 将 SD 卡放回飞控板并启动；
3. 使用 USB 连接终端；
4. 手动执行测试脚本：sh /fs/microsd/etc/rc_test.txt；
5. 这一步是关键：观察脚本的启动，找到它是在哪里运行失败并停止的。检查失败处相关代码的语法错误和逻辑错误。修改错误，重新运行直至成功。
6. 小技巧：为了发现是哪一行错了，在启动脚本中添加一些 echo 命令来将程序运行状态输出，从而可以便捷地发现是哪里出了问题。
   - echo “here1”
   - echo “here2”
   - 等等

混控（MIX）

参考链接：http://pixhawk.org/dev/mixing

执行器控制组（Actuator Control Groups）：一个执行器控制组是由 8 个涉及