

目 录

目 录	IV
一 PX4	1
1.1 编译	1
1.2 软件在环仿真 SIL	4
1.2.1 启动流程	4
1.2.2 jMAVSIM 初始化	9
1.2.3 PX4 SITL 初始化	9
1.2.3.1 simulator 模块流程	12
1.2.3.2 gyrosim 等虚拟传感器模块流程	12
1.2.3.3 pwm_out_sim 与 mixer 模块流程	13
1.2.3.4 MAVLink 解析	15
1.2.3.5 其他 MAVLink 问题	15
1.2.4 仿真通信流程	15
1.2.5 Simulink 与 PX4 进行 SIL 仿真	16
1.3 硬件在环仿真 HIL	16
1.3.1 启动流程	16
1.3.2 jMAVSIM 启动时间的讨论	21
二 ROS	25
2.1 基础	25
2.1.1 创建工作空间	25
2.1.2 创建一个 pkg	25
2.1.3 编译 git 上的 pkg	25
2.1.4 问题及解决方案	25
三 Python3 学习使用	26
3.1 Python3 生成 PDF 文档	26
3.1.1 读取测试数据	26
3.1.2 数据分析及处理	27
3.1.3 测试结果及绘图	27
3.1.4 生成 PDF 文档	28
3.2 数学运算	30

四 状态估计	31
4.1 基础理论知识	31
4.1.1 低通滤波器	31
4.1.2 动态观测器	31
4.1.3 连续-离散卡尔曼滤波器	32
4.1.4 从概率角度的理解	34
4.1.5 更多讨论	35
4.2 四元数	36
4.2.1 四元数基本计算	36
4.2.2 四元数微分方程	36
4.3 PX4 状态估计源码分析	36
4.3.1 attitude_estimator_q	36
五 轨迹跟踪	37
5.1 基础理论知识	37
5.1.1 L1 轨迹跟踪	37
5.1.2 进一步讨论	38
5.1.3 其他的几种轨迹跟踪	38
5.2 MPC 控制及动态路径跟踪	39
六 规划	41
6.1 最短路径算法	41
6.1.1 Floyd 算法	41
6.1.2 Dijkstra 算法	41
6.1.3 Bellman-Ford 算法	42
6.1.4 A* 算法	42
6.2 路径搜索算法	45
6.2.1 PRM 算法	45
6.2.2 RRT 算法	45
6.2.3 RRT* 算法	47
6.3 路径曲线生成算法	47
6.3.1 Dubins path	47
6.3.2 Bezier 曲线和 B-spline 曲线	48
6.3.3 其他形式曲线	49
6.3.4 多项式轨迹	49

七 C++	51
7.1 基础算法	51
7.1.1 快速排序	51
7.1.2 堆排序	51
7.1.3 计数排序	52
7.2 语言使用	53
7.2.1 const 关键字的使用	53
7.2.2 用标准库构造堆	53
八 机器学习	54
8.1 准备知识	54
8.1.1 概述	54
8.1.2 统计学习三要素	54
8.1.3 模型评估	55
8.1.4 归一化与标准化	55
8.1.5 交叉熵代价函数	55
8.1.6 对数损失函数	56
8.2 梯度下降及线性回归	56
8.2.1 基本算法	56
8.2.2 线性回归	57
8.3 Logistic Regression, LR	57
8.3.1 Sigmoid 函数与 Softmax 函数的区别	58
8.3.2 Sigmoid 函数多分类与 Softmax 函数多分类的区别	58
8.3.3 二分类逻辑回归求解	58
8.3.4 softmax 多分类求解	59
九 强化学习	60
9.1 强化学习里的对比思考	60
9.1.1 策略迭代与值迭代	60
9.2 深度强化学习准备知识	61
9.2.1 Q-Learning 和 Policy Gradient 的对比	62
9.2.2 策略分类	62
9.3 DQN	62
9.4 策略优化的基础知识	63
9.4.1 回报函数	65
9.4.2 Generalized Advantage Estimation, GAE	65

9.5 DDPG 算法	66
9.5.1 另一种目标函数的形式	66
9.5.2 确定性策略梯度 DPG	67
十 一些有趣的概率问题	68
10.1 三门问题	68
10.2 吸收马尔可夫过程	68

— PX4

1.1 编译

对于简单的工程，可以直接手动用 gcc/g++ 将.c/.cpp 文件编译链接为可执行文件。若工程包含多个文件夹多个文件，可以通过编写 makefile 文件并利用 make 工具，自动化地对所有文件进行编译。而对于更为复杂的工程，可以通过编写 CMakeLists 文件，并利用 cmake 工具，自动化地生成 makefile 文件。

PX4 代码编译的入口是根目录下的 Makefile 文件。该 Makefile 文件首先执行一些基本的检测、设置，然后利用 cmake 工具，由根目录下的 CMakeLists 文件和 PX4 代码结构中的其他配置文件，自动生成 makefile 文件并进行 make 编译。

make 命令的基本教程可以参考[这里](#)。

首先对 git 环境、编译工具、系统目录等进行了简单的检测和配置：

```

38 ifeq ($(wildcard .git),)
39   $(error YOU HAVE TO USE GIT TO DOWNLOAD THIS REPOSITORY. ABORTING
40   .)
41 endif
42
43 CMAKE_VER := $(shell Tools/check_cmake.sh; echo $$?)
44 ifneq ($(CMAKE_VER),0)
45   $(warning Not a valid CMake version or CMake not installed.)
46   $(warning On Ubuntu 16.04, install or upgrade via:)
47   $(warning )
48   $(warning 3rd party PPA:)
49   $(warning sudo add-apt-repository ppa:george-edison55/cmake-3.x -y
50   )
51   $(warning sudo apt-get update)
52   $(warning sudo apt-get install cmake)
53   $(warning )
54   $(warning Official website:)
55   $(warning wget https://cmake.org/files/v3.4/cmake-3.4.3-Linux-
56   x86_64.sh)
57   $(warning chmod +x cmake-3.4.3-Linux-x86_64.sh)
58   $(warning sudo mkdir /opt/cmake-3.4.3)
59   $(warning sudo ./cmake-3.4.3-Linux-x86_64.sh --prefix=/opt/cmake
       -3.4.3 --exclude-subdir)
60   $(warning export PATH=/opt/cmake-3.4.3/bin:$PATH)
61   $(warning )
62   $(error Fatal)
```

```

60 endif

79 # explicity set default build target
80 all: posix_sitl_default

81

82 # Parsing
83 #

-----  

84 # assume 1st argument passed is the main target, the
85 # rest are arguments to pass to the makefile generated
86 # by cmake in the subdirectory
87 FIRST_ARG := $(firstword $(MAKECMDGOALS))
88 ARGS := $(wordlist 2,$(words $(MAKECMDGOALS)),$(MAKECMDGOALS))
89 j ?= 4

90

91 ifndef NO_NINJA_BUILD
92 NINJA_BUILD := $(shell ninja --version 2>/dev/null)
93 endif
94 ifdef NINJA_BUILD
95   PX4_CMAKE_GENERATOR ?= "Ninja"
96   PX4_MAKE = ninja
97   PX4_MAKE_ARGS =
98 else
99
100 ifdef SYSTEMROOT
101   # Windows
102   PX4_CMAKE_GENERATOR ?= "MSYS Makefiles"
103 else
104   PX4_CMAKE_GENERATOR ?= "Unix Makefiles"
105 endif
106   PX4_MAKE = $(MAKE)
107   PX4_MAKE_ARGS = -j$(j) --no-print-directory
108 endif
109

110 # check if replay env variable is set & set build dir accordingly
111 ifdef replay
112   BUILD_DIR_SUFFIX := _replay
113 else
114   BUILD_DIR_SUFFIX :=

```

```

115 endif
116
117 # additional config parameters passed to cmake
118 CMAKE_ARGS :=
119 ifdef EXTERNAL_MODULES_LOCATION
120   CMAKE_ARGS := -DEXTERNAL_MODULES_LOCATION:STRING=$(
121     EXTERNAL_MODULES_LOCATION)
122 endif
123 SRC_DIR := $(shell dirname $(realpath $(lastword $(MAKEFILE_LIST))))
```

以下是/cmake/configs/目录下获取所有可编译的目标，其中 ALL_CONFIG_TARGETS 是所有目标，NUTTX_CONFIG_TARGETS 是 NUTTX_开头的目标。

```

149 # Get a list of all config targets.
150 ALL_CONFIG_TARGETS := $(basename $(shell find "$${SRC_DIR}/cmake/
151   configs" ! -name '*_common*' ! -name '*_sdflight_*' -name '*.cmake'
152   ' -print | sed -e 's:^.*/::' | sort))
153 # Strip off leading nuttx_
154 NUTTX_CONFIG_TARGETS := $(patsubst nuttx_%,%,$(filter nuttx_%,$(
155   ALL_CONFIG_TARGETS)))
```

然后利用 call-build 对当前目标 (\$@) 进行编译。再往下基本都是各种伪目标的说明。

```

158 # All targets.
159 $ (ALL_CONFIG_TARGETS):
160   $(call cmake-build,$@)
161
162 # Abbreviated config targets.
163
164 # nuttx_ is left off by default; provide a rule to allow that.
165 $ (NUTTX_CONFIG_TARGETS):
166   $(call cmake-build,nuttx_$$@)
```

对 cmake-build 的宏定义是 Makefile 文件中的核心部分：

```

125 # Functions
126 #
127 # describe how to build a cmake config
128 define cmake-build
```

```

129 +@$(eval BUILD_DIR = $(SRC_DIR)/build_$$BUILD_SUFFIX)
130 +@if [ $(PX4_CMAKE_GENERATOR) = "Ninja" ] && [ -e $(BUILD_DIR)/
     Makefile ]; then rm -rf $(BUILD_DIR); fi
131 +@if [ ! -e $(BUILD_DIR)/CMakeCache.txt ]; then mkdir -p $(BUILD_DIR)
     && cd $(BUILD_DIR) && cmake .. -G$(PX4_CMAKE_GENERATOR) -DCONFIG=
     $1 $(CMAKE_ARGS) || (cd .. && rm -rf $(BUILD_DIR)); fi
132 +@echo "PX4 CONFIG: $(BUILD_DIR)"
133 +@$(PX4_MAKE) -C "$(BUILD_DIR)" $(PX4_MAKE_ARGS) $(ARGS)
134 endif

```

129 行定义 BUILD_DIR = /build_@\$ 变量，也就是之后所有编译文件生成的目录。130 行判断 Ninja 和 cmake 之间的冲突。

131 行首先根据 BUILD_DIR/CMakeCache.txt 文件来判断是否需要创建 BUILD_DIR 文件夹并调用 cmake。cmake .. 是指根据根目录下的 CMakeLists 进行 makefile 文件生成，-G\$(PX4_CMAKE_GENERATOR) 则是指定使用 Ninja 或是 cmake，-DCONFIG=\$1 是将第一个参数赋值给 cmake 的变量 CONFIG，该变量在 CMakeLists 中指定了生成 makefile 文件过程中会包含哪个 cmake/configs/.cmake 文件。

133 行，根据目标目录下的 makefile 文件（或是 Ninja 的 rule 文件）编译。

如果想屏蔽掉 Ninja，可以在 91 行之前添加宏定义：

```

0 define NO_NINJA_BUILD
1 endif

```

1.2 软件在环仿真 SIL

1.2.1 启动流程

以下用 make posix_sitl_default jmavsim 为例，分析软件在环仿真（SITL）启动流程。

编译完 posix_sitl_default 目标后，会调用/Tools/sitl_run.sh 脚本（通过什么调用的还不知道）。

```

1 #!/bin/bash
2
3 set -e
4
5 echo args: $@
6
7 sitl_bin=$1
8 rcs_dir=$2
9 debugger=$3

```

```
10 program=$4
11 model=$5
12 src_path=$6
13 build_path=$7
14
15 echo SITL ARGS
16
17 echo sitl_bin: $sitl_bin
18 echo rcs_dir: $rcs_dir
19 echo debugger: $debugger
20 echo program: $program
21 echo model: $model
22 echo src_path: $src_path
23 echo build_path: $build_path
24
25 working_dir=`pwd`
26 sitl_bin=$build_path/src/firmware posix/px4
27 rootfs=$build_path/tmp/rootfs
28
29 if [ "$chroot" == "1" ]
30 then
31     chroot_enabled=-c
32     sudo_enabled=sudo
33 else
34     chroot_enabled=""
35     sudo_enabled=""
36 fi
37
38 # To disable user input
39 if [[ -n "$NO_PXH" ]]; then
40     no_pxh=-d
41 else
42     no_pxh=""
43 fi
44
45 if [ "$model" == "" ] || [ "$model" == "none" ]
46 then
47     echo "empty model, setting iris as default"
48     model="iris"
49 fi
```

```

50
51 if [ "$#" -lt 7 ]
52 then
53   echo usage: sitl_run.sh rc_script rcs_dir debugger program model
54   echo
55   echo "src_path build_path"
56   exit 1
57 fi
58
59 # kill process names that might still
60 # be running from last time
61 pkill gazebo && pgrep gazebo
62 pkill px4 && pgrep px4
63 jmavsim_pid=`ps aux | grep java | grep Simulator | cut -d" " -f1`
64 if [ -n "$jmavsim_pid" ]
65 then
66   kill $jmavsim_pid
67 fi
68
69 cp $src_path/Tools posix_lldbinit $working_dir/.lldbinit
70 cp $src_path/Tools posix_gdbinit $working_dir/.gdbinit
71
72 SIM_PID=0
73
74 if [ "$program" == "jmavsim" ] && [ ! -n "$no_sim" ]
75 then
76   $src_path/Tools/jmavsim_run.sh &
77   SIM_PID=`echo $!`>>>
78   cd ../..
79 elif [ "$program" == "gazebo" ] && [ ! -n "$no_sim" ]
80 then
81   if [ -x "$(command -v gazebo)" ]
82   then
83     # Set the plugin path so Gazebo finds our model and sim
84     source $src_path/Tools/setup_gazebo.bash ${src_path} ${build_path}
85
86     gzserver --verbose ${src_path}/Tools/sitl_gazebo/worlds/${model}.
87     world &
88   SIM_PID=`echo $!`>>>

```

```

88 if [ [ -n "$HEADLESS" ]]; then
89   echo "not running gazebo gui"
90 else
91   gzclient --verbose &
92   GUI_PID=`echo $!`  

93 fi
94 else
95   echo "You need to have gazebo simulator installed!"
96   exit 1
97 fi
98 elif [ "$program" == "replay" ] && [ ! -n "$no_sim" ]
99 then
100   echo "Replaying logfile: $logfile"
101   # This is not a simulator, but a log file to replay
102
103   # Check if we need to creat a param file to allow user to change
104   # parameters
105   if ! [ -f "$rootfs/replay_params.txt" ]
106     then
107       mkdir -p $rootfs
108       touch $rootfs/replay_params.txt
109     fi
110 fi
111 cd $working_dir
112
113 if [ "$logfile" != "" ]
114 then
115   cp $logfile $rootfs/replay.px4log
116 fi
117
118 # Do not exit on failure now from here on because we want the
119 # complete cleanup
120 set +e
121
122 sitl_command="$sudo_enabled $sitl_bin $no_pjh $chroot_enabled
123   $src_path $src_path/${rcS_dir}/${model}"
124
125 echo SITL COMMAND: $sitl_command

```

```

125 # Start Java simulator
126 if [ "$debugger" == "lldb" ]
127 then
128   lldb -- $sitl_command
129 elif [ "$debugger" == "gdb" ]
130 then
131   gdb --args $sitl_command
132 elif [ "$debugger" == "ddd" ]
133 then
134   ddd --debugger gdb --args $sitl_command
135 elif [ "$debugger" == "valgrind" ]
136 then
137   valgrind $sitl_command
138 else
139   $sitl_command
140 fi
141
142 if [ "$program" == "jmavsim" ]
143 then
144   pkill -9 -P $SIM_PID
145   kill -9 $SIM_PID
146 elif [ "$program" == "gazebo" ]
147 then
148   kill -9 $SIM_PID
149   if [[ ! -n "$HEADLESS" ]]; then
150     kill -9 $GUI_PID
151   fi
152 fi

```

其中，73 行判断仿真器，如果是 jmavsim，则在 75 行调用/Tools/jmavsim_run.sh，启动仿真器。在 jmavsim_run.sh 中，主要是在最后先进行编译，然后执行仿真器：

```

37 ant create_run_jar copy_res
38 cd out/production
39 java -Djava.ext.dirs= -jar jmavsim_run.jar $device $extra_args

```

如果是手动启动的话，按照[官网说明](#)，采用如下命令：

```

1 cd ~/jMAVSIM
2 java -Djava.ext.dirs= -cp lib/*:out/production/jmavsim.jar me.drton.
   jmavsim.Simulator -udp 127.0.0.1:14560

```

可以看出，对应调用的入口是 Simulator.java。

继续在 sitl_run.sh 中的 139 行，调用编译出的 px4 可执行文件，对应在终端中显示的信息示例为：

```
0 SITL COMMAND: /home/zhangbo39/firmware/build_posix_sitl_default/src/
    firmware posix/px4 /home/zhangbo39/firmware /home/zhangbo39/
    firmware posix-configs/SITL/init/lpe/iris
1 data path: /home/zhangbo39/firmware
2 commands file: /home/zhangbo39/firmware posix-configs/SITL/init/lpe/
    iris
```

其中，SITL COMMAND 所显示的也就是手动启动时的命令。

1.2.2 jMAVSim 初始化

jMAVSim 中所有的类都包含在 World 这个类中，其中实例化了的类：

1. SimpleEnvironment, 环境类，包含了对环境风的简单建模;
2. connHIL，通信类，负责仿真系统与 PX4 之间的通信;
3. connCommon，通信类，负责地面站与 PX4 之间的通信;
4. vehicle，飞行器类，对仿真对象的简单建模;
5. 视景类（待定）

除此之外，还有 1 个仿真系统类和 2 个端口类被实例化，分别是：

1. hilSystem，仿真系统
2. autopilotMAVLinkPort，飞控通信端口
3. udpGCMAVLinkPort，地面站通信端口

上述三个端口（系统），1 和 2 被加入到 connHIL 中，2 和 3 被加入到 connCommon 中。从这一点可以看出，从飞控端发来的信号除了被送入仿真系统中外，还会被送入地面站。

1.2.3 PX4 SITL 初始化

执行 SITL 仿真时，入口的启动文件就是命令行的第二个参数，”/home/zhangbo39/firmware posix-configs/SITL/init/lpe/iris”，这个类似与常规的 rcS 文件。

```
1 uorb start
2 param load
3 param set MAV_TYPE 2
4 param set SYS_AUTOSTART 4010
5 param set SYS_RESTART_TYPE 2
6 dataman start
7 param set BAT_N_CELLS 3
8 param set CAL_GYRO0_ID 2293768
```

```
9 param set CAL_ACC0_ID 1376264
10 param set CAL_ACC1_ID 1310728
11 param set CAL_MAG0_ID 196616
12 param set CAL_GYRO0_XOFF 0.01
13 param set CAL_ACC0_XOFF 0.01
14 param set CAL_ACC0_YOFF -0.01
15 param set CAL_ACC0_ZOFF 0.01
16 param set CAL_ACC0_XSCALE 1.01
17 param set CAL_ACC0_YSCALE 1.01
18 param set CAL_ACC0_ZSCALE 1.01
19 param set CAL_ACC1_XOFF 0.01
20 param set CAL_MAG0_XOFF 0.01
21 param set SENS_BOARD_ROT 0
22 param set SENS_BOARD_X_OFF 0.000001
23 param set COM_RC_IN_MODE 1
24 param set NAV_DLL_ACT 2
25 param set COM_DISARM_LAND 3
26 param set NAV_ACC_RAD 2.0
27 param set COM_OF_LOSS_T 5
28 param set COM_OBL_ACT 2
29 param set COM_OBL_RC_ACT 0
30 param set RTL_RETURN_ALT 30.0
31 param set RTL_DESCEND_ALT 5.0
32 param set RTL_LAND_DELAY 5
33 param set MIS_TAKEOFF_ALT 2.5
34 param set MC_ROLLRATE_P 0.2
35 param set MC_PITCHRATE_P 0.2
36 param set MC_PITCH_P 6
37 param set MC_ROLL_P 6
38 param set MPC_HOLD_MAX_Z 2.0
39 param set MPC_HOLD_XY_DZ 0.1
40 param set MPC_Z_VEL_P 0.6
41 param set MPC_Z_VEL_I 0.15
42 param set EKF2_GBIAS_INIT 0.01
43 param set EKF2_ANGERR_INIT 0.01
44 replay tryapplyparams
45 simulator start -s
46 rgbledsim start
47 tone_alarm start
48 gyrosim start
```

```

49 accelsim start
50 barosim start
51 adcsim start
52 gpssim start
53 pwm_out_sim mode_pwm
54 sleep 1
55 sensors start
56 commander start
57 land_detector start multicopter
58 navigator start
59 attitude_estimator_q start
60 local_position_estimator start
61 mc_pos_control start
62 mc_att_control start
63 mixer load /dev/pwm_output0 ROMFS/px4fmu_common/mixers/quad_w.main.
    mix
64 mavlink start -u 14556 -r 4000000
65 mavlink start -u 14557 -r 4000000 -m onboard -o 14540
66 mavlink stream -r 50 -s POSITION_TARGET_LOCAL_NED -u 14556
67 mavlink stream -r 50 -s LOCAL_POSITION_NED -u 14556
68 mavlink stream -r 50 -s GLOBAL_POSITION_INT -u 14556
69 mavlink stream -r 50 -s ATTITUDE -u 14556
70 mavlink stream -r 50 -s ATTITUDE_QUATERNION -u 14556
71 mavlink stream -r 50 -s ATTITUDE_TARGET -u 14556
72 mavlink stream -r 50 -s SERVO_OUTPUT_RAW_0 -u 14556
73 mavlink stream -r 20 -s RC_CHANNELS -u 14556
74 mavlink stream -r 250 -s HIGHRES_IMU -u 14556
75 mavlink stream -r 10 -s OPTICAL_FLOW_RAD -u 14556
76 logger start -e -t
77 mavlink boot_complete
78 replay trystart

```

上述代码中，45–53 行、63 行都是与仿真的几个文件。simulator 应用相当于仿真中的一个底层接口，主要作用是负责接收解析仿真器发来的 msg，并将 PX4 的飞行控制量 PWM 发送出去。gyrosim、accelsim、barosim、adcsim、gpssim 是虚拟的底层传感器，但是数据本质上是从 simulator 的数据基础上进行了一次转发。pwm_out_sim 则是调用混控将飞控控制量计算为 PWM 值。mixer 与仿真的地方在于 load 后的设计参数。

1.2.3.1 simulator 模块流程

/firmware/build_posix_sitl_default/src/firmware posix/apps.cpp 中对应入口函数为

```
1 int simulator_main(int argc, char *argv[])
```

启动 Simulator::start 线程:

```
1 int Simulator::start(int argc, char *argv[]){
2     _instance = new Simulator(); //实例化PX4仿真层
3     if (_instance) {
4         ...
5         if (argv[2][1] == 's') {
6             _instance->initializeSensorData();
7 #ifndef __PX4_QURT
8             // Update sensor data
9             _instance->pollForMAVLinkMessages(false, udp_port);
10 #endif
11         ...
12     }
```

其中 _instance->pollForMAVLinkMessages(false, udp_port) 负责仿真阶段的数据收发。具体分为三个阶段:

- (1) 阻塞等待 UDP 数据，每当收到仿真器数据时，回复心跳包。如果收到的数据是非 #0 的 msg，则进入第二阶段。
- (2) 订阅 PWM 输出、飞机状态消息，创建 Simulator::sending_trampoline 线程。向仿真器发送 #76 消息。进入第三阶段
- (3) 阻塞等待 UDP 数据，每当收到仿真器数据时进行解包，若是特定消息 (HIL_SENSOR、HIL_OPTICAL_FLOW、DISTANCE_SENSOR、HIL_GPS、RC_CHANNELS、HIL_STATE_QUATERNION) 则进一步更新虚拟传感器数据，将数据写到指定地址。
特别需要注意的是，此阶段中 publish 形参传入的参数是 false，所以 simulator 收到的数据并不会直接 publish 出去，还需要虚拟传感器的中间环节。

Simulator::sending_trampoline 的主要流程是阻塞等待 _actuator_outputs_sub[0] 的更新，然后调用 poll_topics() 拷贝数据，调用 send_controls() 函数将数据用 #93 消息 (HIL_ACTUATOR_CONTROLS) 发送出去。

1.2.3.2 gyrosim 等虚拟传感器模块流程

类似地，入口函数为

```
1 int gyrosim_main(int argc, char *argv[])
```

启动 int start(enum Rotation rotation):

```

1 ...
2 *g_dev_ptr = new GYROSIM(path_accel, path_gyro, rotation);
3 ...
4 if (OK != (*g_dev_ptr)->init()) {
5     goto fail;
6 }
7 ...

```

之后都会调用 _measure() 函数，在其中通过执行如下函数读取 simulator 的数据。

```

1 if (OK != transfer((uint8_t *)&mpu_report, ((uint8_t *)&mpu_report),
2                     sizeof(mpu_report))) {
3     return;
4 }

```

在 transfer() 函数中，实例化一个临时的 Simulator 并调用 getMPUReport() 读取数据，然后回传数据。

从代码中可以看出，gyrosim、accelsim 会读取并发布多个传感器的数据，具体如下：

gyrosim: gyro (优先级 HIGH, 100), accel (优先级 HIGH, 100)

accelsim: accel (优先级 DEFAULT, 75), mag (优先级 LOW, 50)

另外，如果将 gyrosim 应用屏蔽掉，则仿真中 sensors status 的四个传感器通道数据都是-1。这时因为在 sensors 模块中，更新数据是阻塞等待 gyro 数据，所以如果改为阻塞等待 accel 数据，则另外三个通道数据都会出现，但是 accel 只有 75 优先级的数据。

1.2.3.3 pwm_out_sim 与 mixer 模块流程

pwm_out_sim 的主体 PWMSIM 类继承自虚拟设备类，包含有一个设备 io 接口。

```

1 #ifdef __PX4_NUTTX
2 class PWMSim : public device::CDev
3 #else
4 class PWMSim : public device::VDev
5 #endif
6 {
7 public:
8 ...
9 virtual int ioctl(device::file_t *filp, int cmd, unsigned long arg);

```

在 PWMSIM 构造函数中，就将 pwm_out_sim 的设备地址绑定为 "/dev/pwm_output0" (宏定义变量为 PWM_OUTPUT0_DEVICE_PATH)，因此在 mixer 中 load 后的地址也是指向这个设备，从而在 mixer 的 load 函数中，进行 MIXERIOCRESET、

```

zhangbo39@x4650: ~/firmware
px4 starting.

INFO [dataman] Unknown restart, data manager file 'rootfs/fs/microsd/dataman' size is 47640 bytes
INFO [platforms_posix_drivers_ledsim] LED::init
INFO [platforms_posix_drivers_ledsim] LED::init
INFO [simulator] Waiting for initial data on UDP port 14666. Please start the flight simulator to proceed..
WARN [simulator] msg id : 0
WARN [simulator] msg id : 0
WARN [simulator] msg id : 11
INFO [simulator] Got initial simulation data, running sim..
INFO [pwm_out_sim] MODE_8PWM
Sleeping for 1 s; (1000000 us).
WARN [sensors] FATAL: no gyro found: /dev/gyro0 (9)
ERROR [sensors] sensor initialization failed
INFO [platforms_posix_px4_layer] simulator is slow. Delay added: 1111894 us
WARN [simulator] msg id : 107
INFO [mavlink] mode: Normal, data rate: 4000000 B/s on udp port 14556 remote port 14550
INFO [mavlink] mode: Onboard, data rate: 4000000 B/s on udp port 14557 remote port 14540

INFO [platforms_posix_px4_layer] simulator is slow. Delay added: 5850903 us
WARN [simulator] msg id : 107
INFO [tone_alarm] startup
INFO [mavlink] MAVLink only on localhost (set param MAV_BROADCAST = 1 to enable network)
INFO [logger] logger started (mode=all)
pxh>
pxh> WARN [attitude_estimator_q] Q POLL TIMEOUT
pxh> sensors status
INFO [sensors] gyro status:
INFO [lib_ecl] validator: best: -1, prev best: -1, failsafe: NO (0 events)
INFO [sensors] accel status:
INFO [lib_ecl] validator: best: -1, prev best: -1, failsafe: NO (0 events)
INFO [sensors] mag status:
INFO [lib_ecl] validator: best: -1, prev best: -1, failsafe: NO (0 events)
INFO [sensors] baro status:
INFO [lib_ecl] validator: best: -1, prev best: -1, failsafe: NO (0 events)
pxh> 

```

图 1 阻塞等待 gyro 数据时屏蔽 gyrosim

```

zhangbo39@x4650: ~/firmware
INFO [simulator] Got initial simulation data, running sim..
INFO [pwm_out_sim] MODE_8PWM
Sleeping for 1 s; (1000000 us).
WARN [sensors] FATAL: no gyro found: /dev/gyro0 (9)
ERROR [sensors] sensor initialization failed
INFO [platforms_posix_px4_layer] simulator is slow. Delay added: 1206978 us
WARN [simulator] msg id : 107
INFO [mavlink] mode: Normal, data rate: 4000000 B/s on udp port 14556 remote port 14550
INFO [mavlink] mode: Onboard, data rate: 4000000 B/s on udp port 14557 remote port 14540
INFO [platforms_posix_px4_layer] simulator is slow. Delay added: 1216812 us
WARN [simulator] msg id : 107
INFO [tone_alarm] startup
INFO [mavlink] MAVLink only on localhost (set param MAV_BROADCAST = 1 to enable network)
pxh> INFO [logger] logger started (mode=all)
WARN [attitude_estimator_q] Q POLL TIMEOUT
pxh> sensors status
INFO [sensors] gyro status:
INFO [lib_ecl] validator: best: -1, prev best: -1, failsafe: NO (0 events)
INFO [sensors] accel status:
INFO [lib_ecl] validator: best: 0, prev best: 0, failsafe: NO (0 events)
INFO [lib_ecl] sensor #0, prio: 75, state: STALE_DATA
INFO [lib_ecl]     val: 1.5000, lp: 1.5000 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
INFO [lib_ecl]     val: 0.0000, lp: 0.0000 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
INFO [lib_ecl]     val: 0.0000, lp: 0.0000 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
INFO [sensors] mag status:
INFO [lib_ecl] validator: best: 0, prev best: 0, failsafe: NO (0 events)
INFO [lib_ecl] sensor #0, prio: 50, state: STALE_DATA
INFO [lib_ecl]     val: 0.8986, lp: 0.8986 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
INFO [lib_ecl]     val: 0.0331, lp: 0.0331 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
INFO [lib_ecl]     val: 1.7829, lp: 1.7829 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
INFO [sensors] baro status:
INFO [lib_ecl] validator: best: 0, prev best: 0, failsafe: NO (0 events)
INFO [lib_ecl] sensor #0, prio: 75, state: STALE_DATA
INFO [lib_ecl]     val: 487.9141, lp: 487.9141 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
INFO [lib_ecl]     val: 0.0000, lp: 0.0000 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
INFO [lib_ecl]     val: 0.0000, lp: 0.0000 mean dev: 0.0000 RMS: 0.0000 conf: 0.0000
pxh> 

```

图 2 阻塞等待 accel 数据时屏蔽 gyrosim

MIXERIOCLOADBUF 两个指令时都是调用了 `PWMSim::ioctl()` 函数。通过这个 load 函数完成了混控器的初始化（构造混控列表等）。

`pwm_out_sim` 的基本流程就是先设置模式，包含输出通道数量（4）、通信频率（50Hz）等。随后再启动 `PWMSim::task_main_trampoline` 线程，负责订阅控制器的控制量，经过混控后再发布出去。

1.2.3.4 MAVLink 解析

SIL 中 UDP 发送与接收的数据都是按照 MAVLink 协议打包的。对于 PX4 端向 UDP 发送数据，调用的是 `simulator_mavlink.cpp` 中的函数

```
1 void Simulator::send_mavlink_message(const uint8_t msgid, const
                                         void *msg, uint8_t component_ID)
```

而对于从 UDP 取数据到 PX4，调用的入口是 `mavlink_parse_char()` 函数，这些函数与 HIL/真实飞行模式是共用的，因此在 SIL 中的接收数据并解包的过程与常规模式相同。

另外，如果在常规模式中修改了通信协议，那么 SIL 中共需要修改两个地方，一个是发送的部分（即 `Simulator::send_mavlink_message()` 函数），另一个是将 `simulator.h` 中包含的头文件修改与常规模式一致。

1.2.3.5 其他 MAVLink 问题

1. Payload 中每个字段都使用小端模式，低位在前、高位在后。
2. Payload 中字段的顺序是按照每个字段 type size 降序排列的，若有拓展字段，则在排序后追加。
3. CRC 校验中，包头 FE 并不校验，在包尾还会增加两个额外校验位 (extraCRC)，该校验位的计算与每一条消息的消息名称、payload 中字段类型、字段名、字段数组长度等计算。心跳包计算结果为 50，所以增加的校验位为'0x32'。具体的计算方法可以参考 jMAVSIM，或者可以参考 mavlink 库中的宏定义 MAVLINK_MESSAGE_CRC32。需要注意的是该宏定义的实现在 mavlink1.0 和 mavlink2.0 是不同的。

1.2.4 仿真通信流程

在基本的仿真准备完成后，一个相对完整的仿真通信流程是这样的：

1. hilSystem 会按照固定的时间间隔 (1000ms) 向端口 14560 发心跳包，msgID=0。SIM->PX4
2. autopilotMAVLinkPort 监听到 PX4->SIM 的心跳包，转发给 hilSystem 做收到心跳后的第一次处理。
3. 当 hilSystem 第二次收到 PX4->SIM 的心跳包后，初始化 MAVLink：向 PX4 发送 msgID=11 的消息，标示出自己的 sysID，并将 PX4 置为 disarmed。此时初始化

完成，在 hilSystem 发送心跳的过程时，也会发送仿真系统状态，分别用 107、115、113 号消息。

4. PX4->SIM 发送 76 号消息，其中指令为 511 (设置消息间隔), param1=115 (指定为 115 号消息, HIL_STATE_QUATERNION) ,param2=5000 (该消息间隔为 5000us)。

5. PX4->SIM 按照一定的间隔发送 93 号消息，即飞控算出的各个电机输出值。按照 MAVLink 协议，#93 消息中的控制量取值-1 .. 1，然而在 PX4 源码和 jMAVSIM 中，都将该值视为 0 .. 1 之间的值。jMAVSIM 端见 Rotor.java 文件，PX4 源码见 simulator_mavlink.cpp 中的函数：

```
1 void Simulator::pack_actuator_message(mavlink_hil_actuator_controls_t
    &msg, unsigned index);
```

1.2.5 Simulink 与 PX4 进行 SIL 仿真

1.3 硬件在环仿真 HIL

1.3.1 启动流程

jMAVSIM 在 HIL 模式下启动指令为：

```
1 java -Djava.ext.dirs= -cp lib/*:out/production/jmavsim.jar me.drton.
jmavsim.Simulator -serial /dev/ttyACM0 921600 -qgc
```

PX4 在 HIL 模式下，使用的固件仍然是正常飞行时的 px4fmu-v2_default 这个，变化的地方在于机型选择，需要使用仿真机型： 对应的系统参数应该是 SYS_-



图 3 HIL 模式时需要选择的机型

AUTOSTART，选择 HIL Quadcopter X 时对应于 1001：

HIL 在初始化时的启动文件也是 /firmware/ROMFS/px4fmu_common/init.d/rcs，其中会调用编译时生成的 /etc/init.d/rc.autostart 的文件。在这个

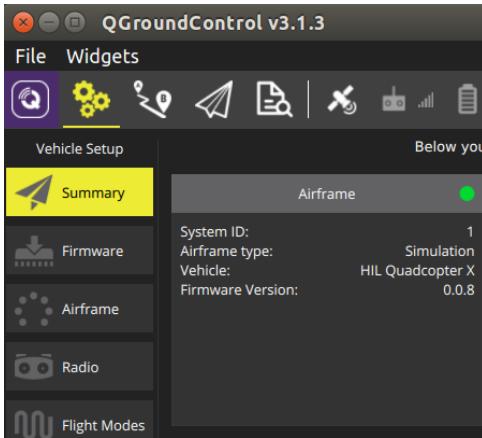


图 4 HIL 模式下的机型

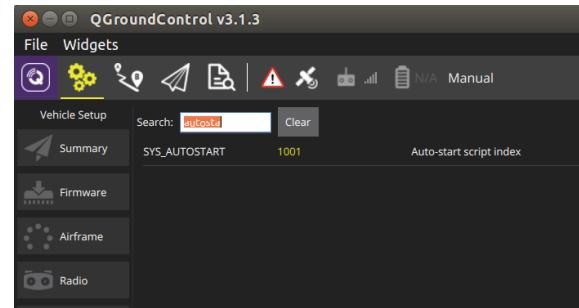


图 5 HIL 模式下对应的参数

rc.autostart 文件中，通过比较 `SYS_AUTOSTART` 来确定执行`/etc/init.d/`文件夹下的机型脚本。1001 对应于 `1001_rc_quad_x.hil`，内容为：

```

1  #!nsh
2
3  #
4  # @name HIL Quadcopter X
5
6  #
7  # @type Simulation
8
9
10 sh /etc/init.d/rc.mc_defaults
11
12 set MIXER quad_x
13
14 set HIL yes

```

主要参数是将 `HIL` 变量置为 `yes`，这个变量在 rcS 中的作用：(1) commander 的启动方式为"commander start -hil"；(2) 启动 `pwm_out_sim`，且参数为 `mode_pwm16`。`pwm_out_sim` 的启动及其参数与 SIL 中的作用基本一致。commander 采用 hil 模式时，会在线程中设置变量 `startup_in_hil`

```

0  if (argc > 2) {
1      if (!strcmp(argv[2], "-hil")) {
2          startup_in_hil = true;
3      } else {
4          PX4_ERR("Argument %s not supported, abort.", argv[2]);
5          thread_should_exit = true;
6      }

```

7 }

接着设置变量 `status.hil_state`

```
0 if (startup_in_hil) {
1     status.hil_state = vehicle_status_s::HIL_STATE_ON;
2 } else {
3     status.hil_state = vehicle_status_s::HIL_STATE_OFF;
4 }
```

并在函数 `set_control_mode` 中：

```
0 control_mode.flag_system_hil_enabled = status.hil_state ==
1     vehicle_status_s::HIL_STATE_ON;
```

同时也会发布出去：

```
0 if (counter % (200000 / COMMANDER_MONITORING_INTERVAL) == 0 ||
1     status_changed) {
2     set_control_mode();
3     control_mode.timestamp = now;
4     orb_publish(ORB_ID(vehicle_control_mode), control_mode_pub, &
5     control_mode);
```

这个 `vehicle_control_mode` 会在 `sensors.cpp` 中的 `vehicle_control_mode_poll()` 函数中被订阅：

```
1 void Sensors::vehicle_control_mode_poll()
2 {
3     struct vehicle_control_mode_s vcontrol_mode;
4     bool vcontrol_mode_updated;
5
6     /* Check HIL state if vehicle control mode has changed */
7     orb_check(_vcontrol_mode_sub, &vcontrol_mode_updated);
8
9     if (vcontrol_mode_updated) {
10
11         orb_copy(ORB_ID(vehicle_control_mode), _vcontrol_mode_sub, &
12             vcontrol_mode);
13         _armed = vcontrol_mode.flag_armed;
14
15         /* switching from non-HIL to HIL mode */
16         if (vcontrol_mode.flag_system_hil_enabled && !_hil_enabled) {
17             _hil_enabled = true;
```

```

17     _publishing = false;
18     /* switching from HIL to non-HIL mode */
19
20 } else if (!_publishing && !_hil_enabled) {
21     _hil_enabled = false;
22     _publishing = true;
23 }
24 }
25 }
```

其作用就是给变量 _publishing 赋值，而这个值在 Sensors::task_main() 中会决定是否将 sensors 中综合得到的传感器数据发布除去：

```

1 if (_publishing && raw.timestamp > 0) {
2
3     /* construct relative timestamps */
4     if (_last_accel_timestamp[_accel.last_best_vote]) {
5         raw.accelerometer_timestamp_relative = (int32_t)(
6             _last_accel_timestamp[_accel.last_best_vote] - raw.
7             timestamp);
8     }
9
10    if (_last_mag_timestamp[_mag.last_best_vote]) {
11        raw.magnetometer_timestamp_relative = (int32_t)(
12            _last_mag_timestamp[_mag.last_best_vote] - raw.timestamp);
13    }
14
15    orb_publish(ORB_ID(sensor_combined), _sensor_pub, &raw);
16 }
```

综合来看，如果是 HIL 模式，sensors 中的传感器数据不会发布到 sensor_combined 这个 topic 上，导航模块中订阅的传感器信息也就不是 sensors 发出的了。

实际在 HIL 模式中，从串口获取仿真数据并发布给导航的流程为：rcS 启动串口的 mavlink 之后，也会订阅 vehicle_control_mode 这个 topic，并设置 _hil_enabled = true，这样当 MavlinkReceiver::handle_message() 函数收到 msg 后，会在解析 hil 消息后并进行发布。在 MavlinkReceiver::handle_message_hil_sensor() 函数中，会将数据发布到 sensor_gyro 等传感器数据以

及 sensor_combined 这个导航使用的 topic 中。

而在 HIL 模式中, 将飞控数据用 mavlink 协议从串口发送的流程为: mavlink 启动后会生成一个 MavLinkStream 链表, 在 mavlink_main.cpp 中的 int Mavlink::task_main() 函数中, 会执行:

```

1 /* update streams */
2 MavlinkStream *stream;
3 LL_FOREACH(_streams, stream) {
4     stream->update(t);
5 }
```

这样就会在链表中调用每一个 stream 节点的 update() 成员函数。在 update() 函数中, 会判断当前时间是否超过了上次发送后的间隔, 然后调用 send(t) 进行数据发送。

综合来看, 在 HIL 模式下数据的接收与发送都是通过 mavlink 相关的模块来实现的, 而 SIL 中则是用 simulator 实现数据中转的。

另外要注意的是, 虽然 sensors 不发布 sensor_combined 这个 topic, 但是 sensors 以及其他传感器模块 (mpu6000 等) 的线程仍然在进行, 所以传感器数据 (HIL 时飞控板的真实传感器数据) 仍然会读取并在 sensors 中订阅, 所以如果在 nsh 中用 sensors status 命令查看状态, 依然会出现多个传感器的数据:

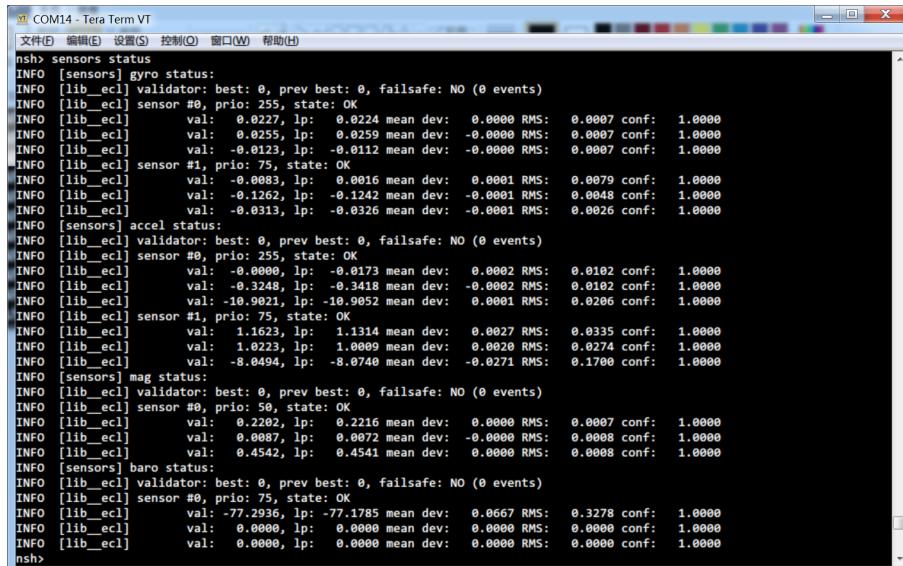


图 6 HIL 模式下飞控通电, 未连接 jMAVSIM 时的 sensors status

从上面的图6和图7可知, 确实能够读到多个传感器数据, 并且连接 jMAVSIM 之后默认的传感器数据都是仿真数据。而 baro 气压计状态更清晰的说明, 未连接 jMAVSIM 或 jMAVSIM 没发数据的时候, sensors status 的读数是飞控的真实数据。

进一步分析可知, 由于真实传感器以及 MavlinkReceiver 中发布数据都是按照单传感器模式 (orb_advertise()), 则优先级为默认值 75) 发布, 所以两者的数据会发生重叠, 这样即使真实气压计和 jMAVSIM 都有 baro 数据, 但 sensors status 中 baro

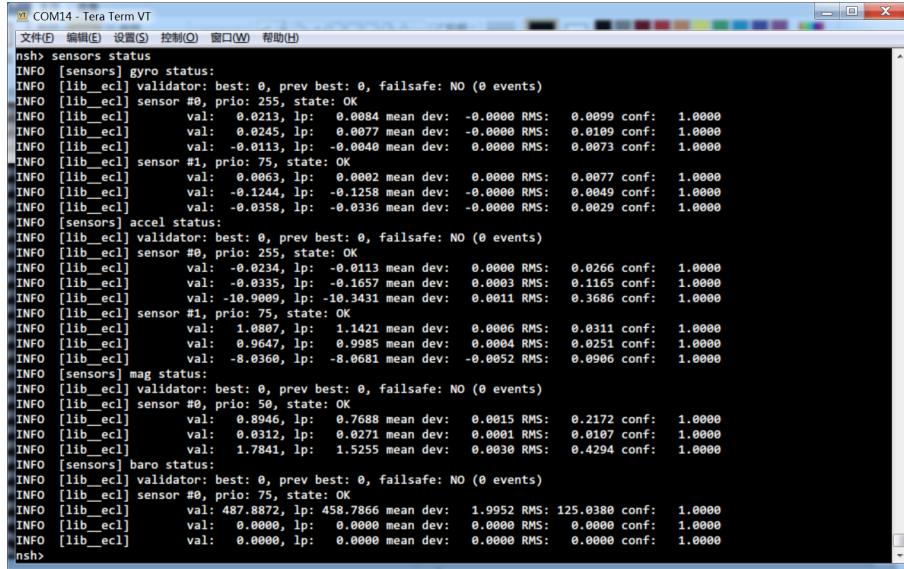


图 7 HIL 模式下飞控通电，连接 jMAVSim 之后的 sensors status

只有一个数据。所以如果在 `void MavlinkReceiver::handle_message_hil_sensor(mavlink_message_t *msg)` 函数中使用 `orb_advertise_multi()` 进行声明优先级为 100(`ORB_PRIO_HIGH`)，就能够显示多个 baro 数据了：

```

1  /* baro */
2  {
3      struct baro_report baro = {};
4
5      baro.timestamp = timestamp;
6      baro.pressure = imu.abs_pressure;
7      baro.altitude = imu.pressure_alt;
8      baro.temperature = imu.temperature;
9      int a = -1;
10     if (_baro_pub == nullptr) {
11         // _baro_pub = orb_advertise(ORB_ID(sensor_baro), &baro);
12         _baro_pub = orb_advertise_multi(ORB_ID(sensor_baro), &baro, &a,
13                                         ORB_PRIO_HIGH);
14     } else {
15         orb_publish(ORB_ID(sensor_baro), _baro_pub, &baro);
16     }
17 }

```

1.3.2 jMAVSim 启动时间的讨论

虽然官网教程中介绍了 HIL 模式的连接方式，但是对于飞控上电和 jMAVSim 开启的先后并没有说明。按照字面正常推断，应该是飞控首先上电，然后在 PC 端打开

```

INFO [sensors] baro status:
INFO [lib_ecl] validator: best: 0, prev best: 0, failsafe: NO (0 events)
INFO [lib_ecl] sensor #0, prio: 75, state: OK
INFO [lib_ecl]     val: -77.4588, lp: -77.5504 mean dev:  0.0068 RMS:  0.2968 conf:  1.0000
INFO [lib_ecl]     val:  0.0000, lp:  0.0000 mean dev:  0.0000 RMS:  0.0000 conf:  1.0000
INFO [lib_ecl]     val:  0.0000, lp:  0.0000 mean dev:  0.0000 RMS:  0.0000 conf:  1.0000
nsh>

```

图 8 HIL 模式下飞控通电，未连接 jMAVSIM 时的 baro

```

INFO [sensors] baro status:
INFO [lib_ecl] validator: best: 0, prev best: 0, failsafe: NO (0 events)
INFO [lib_ecl] sensor #0, prio: 75, state: OK
INFO [lib_ecl]     val: 487.8872, lp: 458.7866 mean dev:  1.9952 RMS: 125.0380 conf:  1.0000
INFO [lib_ecl]     val:  0.0000, lp:  0.0000 mean dev:  0.0000 RMS:  0.0000 conf:  1.0000
INFO [lib_ecl]     val:  0.0000, lp:  0.0000 mean dev:  0.0000 RMS:  0.0000 conf:  1.0000

```

图 9 HIL 模式下飞控通电，连接 jMAVSIM 之后的 baro

jMAVSIM 软件（如果先开启 jMAVSIM 的话，由于串口没有数据通信，jMAVSIM 会自动关闭）。

这里就存在一个飞控上电与 jMAVSIM 启动间隔时间的问题。经过测试，采用 INAV 导航算法时，如果在一定时间之后才启动 jMAVSIM（向 PX4 发送数据）的话，飞控将不能锁定 HOME 点坐标。对应的现象是能够获取初始 GPS，但无法定位当前飞控坐标，也无法响应 mission 等指令。原因如下：

在 `position_estimator_inav_main.cpp` 中，飞控上电后会启动导航主线程 `position_estimator_inav_thread_main`。与其他的常规主线程类似，在阻塞等待数据更新这个主循环之前，是对导航状态量的初始化过程。而这个初始化流程的最后一步是等待并采集一定数量的气压高数据，最后确定气压高偏移值，并将 `local_pos.z_valid` 置为真。那如果 jMAVSIM 一直为发送气压数据，经过一定的等待时间 (`MAX_WAIT_FOR_BARO_SAMPLE=3s`) 后，也会结束等待气压数据，而这时 `local_pos.z_valid` 仍为假。具体代码如下。

```

1 while (wait_baro && !thread_should_exit) {
2     int ret = px4_poll(&fds_init[0], 1, 1000);
3
4     if (ret < 0) {
5         /* poll error */
6         mavlink_log_info(&mavlink_log_pub, "[inav] poll error on init");
7
8     } else if (hrt_absolute_time() - baro_wait_for_sample_time >
9             MAX_WAIT_FOR_BARO_SAMPLE) {

```

```

INFO [sensors] baro status:
INFO [lib_ecl] validator: best: 1, prev best: 0, failsafe: YES (1 events)
INFO [lib_ecl] sensor #0, prio: 75, state: OK
INFO [lib_ecl]     val: -77.2936, lp: -77.5044 mean dev:  0.0287 RMS:  0.2875 conf:  1.0000
INFO [lib_ecl]     val:  0.0000, lp:  0.0000 mean dev:  0.0000 RMS:  0.0000 conf:  1.0000
INFO [lib_ecl]     val:  0.0000, lp:  0.0000 mean dev:  0.0000 RMS:  0.0000 conf:  1.0000
INFO [lib_ecl] sensor #1, prio: 100, state: OK
INFO [lib_ecl]     val: 488.0295, lp: 487.9974 mean dev:  0.6643 RMS: 13.6708 conf:  1.0000
INFO [lib_ecl]     val:  0.0000, lp:  0.0000 mean dev:  0.0000 RMS:  0.0000 conf:  1.0000
INFO [lib_ecl]     val:  0.0000, lp:  0.0000 mean dev:  0.0000 RMS:  0.0000 conf:  1.0000
nsh>

```

图 10 增加优先级后，连接 jMAVSIM 之后的 baro

```

9  wait_baro = false;
10 mavlink_log_info(&mavlink_log_pub, "[inav] timed out waiting for a
11     baro sample");
12 } else if (ret > 0) {
13     if (fds_init[0].revents & POLLIN) {
14         orb_copy(ORB_ID(sensor_combined), sensor_combined_sub, &sensor);
15
16         if (wait_baro && sensor.timestamp + sensor.
17             baro_timestamp_relative != baro_timestamp) {
18             baro_timestamp = sensor.timestamp + sensor.
19                 baro_timestamp_relative;
20             baro_wait_for_sample_time = hrt_absolute_time();
21
22             /* mean calculation over several measurements */
23             if (baro_init_cnt < baro_init_num) {
24                 if (PX4_ISFINITE(sensor.baro_alt_meter)) {
25                     baro_offset += sensor.baro_alt_meter;
26                     baro_init_cnt++;
27                 }
28             } else {
29                 wait_baro = false;
30                 baro_offset /= (float) baro_init_cnt;
31                 local_pos.z_valid = true;
32                 local_pos.v_z_valid = true;
33             }
34         }
35     }
36 } else {
37     PX4_WARN("INAV poll timeout");
38 }
39 }
```

最后发布 vehicle_global_position 消息时的判别条件是 if (local_pos.xy_global && local_pos.z_global)。显然 jMAVSim 启动时间会影响到这里的消息发布。所以即便之后飞控收到了 GPS 数据，但 vehicle_global_position 也始终无法发布，之后也就会遇到前文中出现的现象了。

二 ROS

2.1 基础

2.1.1 创建工作空间

```
1 mkdir -p ~/workspace/src  
2 cd ~/workspace  
3 catkin_make  
4 source devel/setup.bash
```

2.1.2 创建一个 pkg

```
1 cd ~/workspace/src  
2 catkin_create_pkg pkgname std_msgs rospy roscpp  
3 cd ~/workspace  
4 catkin_make
```

2.1.3 编译 git 上的 pkg

1. 创建一个空的文件夹作为工作空间目录
2. git clone 待使用的目标 package
3. 在工作空间目录下，使用 `catkin_make --source pkgname` 进行编译
4. (似乎可以忽略) 使用 `catkin_make install --source pkgname` 安装。
5. 使用 `source devel/setup.bash` 读取当前工作空间的环境变量。

2.1.4 问题及解决方案

编译报错: `ModuleNotFoundError: No module named 'em'` 使用“`~$ pip install empy`”

编译继续报错: `AttributeError: module 'em' has no attribute 'Interpreter'` 解决:

`pip uninstall em` `pip install empy`

三 Python3 学习使用

3.1 Python3 生成 PDF 文档

初始的需求是将测试设备得到的测量数据，自动生成为测试报告。其中测量数据形式为 $[t, \mathbf{D}_1, \mathbf{D}_2, \dots, \mathbf{D}_n]$ 这样的数据集，第一列为时间戳，后续为特定数据。考虑到通用性和开发便捷性，以及对数据处理和绘图的要求，考虑采用 Python3 生成 PDF 文档。特别标明用的 Py3 版本是因为在某些环节中 Py2 的处理会不适用。

整体流程以及主要使用的包如图11所示：

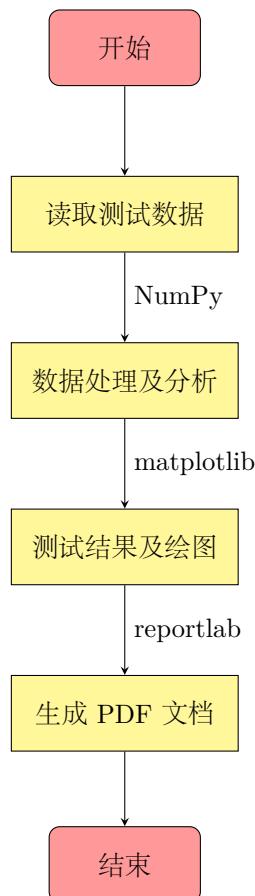


图 11 自动报告 python 生成流程

3.1.1 读取测试数据

这里默认下位机数据已经回传到上位机中，协议使用 csv 格式且无第一行变量名称，那么可以直接调用

```
1 import numpy, sys
2 st = sys.argv
3 if (len(st) > 1):
4     filename = st[1].split('.')[0]
```

```

5 else:
6     print("Please input log file name!")
7     return
8 data = numpy.loadtxt(open(filename+'.csv','rb'),delimiter=",",
9 skiprows=0)

```

3.1.2 数据分析及处理

待定

3.1.3 测试结果及绘图

值得注意的是绘图中 1 维/n 维数据的处理、中文处理等。具体代码为：

```

1 import matplotlib
2 import matplotlib.pyplot as plt
3
4 #字体路径、尺寸设置
5 TTFontpath = {'zen':'./wqy-zenhei.ttc','micro':'./wqy-microhei.ttc'}
6 zhfont2 = matplotlib.font_manager.FontProperties(fname=TTFontpath['
    micro'])
7 zhfont1 = matplotlib.font_manager.FontProperties(fname=TTFontpath['
    zen'])
8 zhfontlegend = zhfont1
9 zhfontlegend.set_size(22)
10
11 #画图
12 def getMyImage(x, y, xlabel, ylabel, title, legendstr = ''):
13     fig = plt.figure(figsize = (16,16))
14     if y.size == y.shape[0]:
15         plt.plot(x,y, label = legendstr)
16     else:
17         for i in range(y.shape[1]):
18             plt.plot(x, y[:,i], label = legendstr[i])
19     plt.xlabel(xlabel,fontsize = 24, fontproperties=zhfont1)
20     plt.ylabel(ylabel,fontsize = 24, fontproperties=zhfont1)
21     plt.title(title,fontsize = 28, fontproperties=zhfont1)
22     if legendstr != '':
23         plt.legend(loc='best', ncol=y.size//y.shape[0], prop =
24             zhfontlegend, shadow=True, fancybox=True).get_frame().
25             set_alpha(0.4)

```

```

25 #主函数中截选
26
27 simt = (data[:,0] - data[0,0]) / 1000.0
28 curr = data[:,[3,4,7]]
29 getMyImage(simt, data[:,5], '时间(s)', '温度(度)', '温度曲线')
30 getMyImage(simt, curr, '时间(s)', '电压(V)', '电调电压曲线', ['上电机电调电
    流', '下电机电调电流', '总电流'])

```

NumPy 的 array 矩阵，如果抽取一列的话，结果是一维向量，这时 shape 属性的为 ‘(len,)’，没有第二个值。所以在 14 行做了特殊的判断。

ubuntu 下系统的字体可以在 ‘/usr/share/fonts/truetype/’ 下查找，这里将字体放到项目目录下了。图例中的属性使用 ‘prop’ 关键字，而标签、标题则是 ‘fontsize/ fontproperties’ 等。

3.1.4 生成 PDF 文档

生成 PDF 使用 reportlab 这个包，并且是基于 test_charts_textlabels.py 这个模板改的。主要的细节有两点，一是中文格式等，二是 matplotlib 图片怎么直接插入到 PDF 中（而不用先临时保存为文件）。reportlab 包的使用具体可以参考 reportlab 的手册，或者直接从 [reportlab3.4 源码地址](#) 下载源码并使用 ‘python3 setup.py install’ 安装，然后生成测试代码、demo、手册等 PDF。

中文格式采用以下的办法：

```

1 from reportlab.pdfbase import pdfmetrics
2 from reportlab.pdfbase.ttfonts import TTFont
3
4 TTFontpath = {'zen':'./wqy-zenhei.ttc','micro':'./wqy-microhei.ttc'}
5 pdfmetrics.registerFont(TTFont('micro', TTFontpath['micro']))
6 pdfmetrics.registerFont(TTFont('zen', TTFontpath['zen']))

```

在使用时：

```

1 story = []
2 styleSheet = getSampleStyleSheet()
3 bt = styleSheet['BodyText']
4 cbt = styleSheet['BodyText']
5 h1 = styleSheet['Heading1']
6 ch1 = styleSheet['Heading1']
7
8 cbt.fontSize = 'micro'
9 ch1.fontSize = 'zen'

```

```

11 story.append(Paragraph('测试标题测试标题', ch1))
12 story.append(Paragraph('测试文本测试文本', cbt))

```

将 matplotlib 的图片插入到 reportlab 的 PDF 中，有 2 个中间步骤：

(1) 将 matplotlib 的图片对象转为 reportlab 中的图片对象

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 from PIL import Image
5 import matplotlib.pyplot as plt
6 from reportlab.pdfgen import canvas
7 from reportlab.lib.units import inch, cm
8
9 from reportlab.lib.utils import ImageReader
10
11 fig = plt.figure(figsize=(4, 3))
12 plt.plot([1, 2, 3, 4])
13 plt.ylabel('some numbers')
14
15 imgdata = Image.io.BytesIO() #
16 fig.savefig(imgdata, format='png')
17 imgdata.seek(0) # rewind the data
18
19 Image = ImageReader(imgdata) #
20
21 c = canvas.Canvas('test.pdf')
22 c.drawImage(Image, cm, cm, inch, inch)
23 c.save()

```

以上是能够在 Python3 中运行的 demo，具体参考的是 stackoverflow 上[How to draw Image a matplotlib figure in a reportlab canvas?](#)的回答和[how to save a pylab figure into in-memory file which can be read into PIL image?](#)的回答。前一个回答中的代码在 Python2 上没问题，当在 Python3 中无法使用。

(2) 在 reportlab 的模板中追加图片

```

1 class flowable_fig(reportlab.platypus.Flowable):
2     def __init__(self, imgdata):
3         reportlab.platypus.Flowable.__init__(self)
4         self.img = reportlab.lib.utils.ImageReader(imgdata)
5     def draw(self):
6         self.canv.drawImage(self.img, 0.5 * A4[0]-3.5*inch, 0*inch,

```

```

height = -3*inch, width = 5*inch)

7
8 def getMyImage(x, y, xlabel, ylabel, title, legendstr = ''):
9     ...
10    imgdata = Image.io.BytesIO()
11    fig.savefig(imgdata, format='png')
12    imgdata.seek(0)
13    return flowable_fig(imgdata)
14
15 #主函数中截选
16 ...
17 story.append(getMyImage(simt, data[:,5], '时间(s)', '湿度(%)', '湿度曲
线'))

```

参考的是Draw images with canvas and use SimpleDocTemplate的回答。

3.2 数学运算

16 进制与浮点数相互转化:

```

1 import struct
2 struct.unpack('!f', bytes.fromhex('3f660ab9'))[0]
3 hex(struct.unpack('<I', struct.pack('<f', 2.0000))[0])

```

abcdefg

1 hello world!

四 状态估计

4.1 基础理论知识

本节主要参考了文献 State Estimation for Micro Air Vehicles 的第 4、5 节内容。

4.1.1 低通滤波器

低通滤波器可以认为是一种较为简单的估计器。常见的一阶低通滤波器可以表示为：

$$Y(s) = \frac{a}{s+a} U(s)$$

通过 Laplace 逆变换可以得到时域形式为：

$$\dot{y} = -ay + au \quad (4-1)$$

进一步地，求解该微分方程可得：

$$y(t+T) = e^{-aT}y(t) + a \int_0^T e^{-a(T-\tau)} u(\tau) d\tau$$

如果用 T_s 表示采样周期，且假设 $u(t)$ 在采样周期内保持常值，那么有：

$$\begin{aligned} y(k+1) &= e^{-aT_s}y(k) + a \int_0^{T_s} e^{-a(T_s-\tau)} d\tau \cdot u(k) \\ &= e^{-aT_s}y(k) + (1 - e^{-aT_s})u(k) \end{aligned} \quad (4-2)$$

定义 $\alpha_{LPF} = e^{-aT_s}$ ，则可以进一步简化为：

$$y(k+1) = \alpha_{LPF}y(k) + (1 - \alpha_{LPF})u(k) \quad (4-3)$$

如果用符号 $LPF(\cdot)$ 表示上述的低通滤波运算，那么 $\hat{x} = LPF(x)$ 就是 x 在低通滤波下的估计值。

4.1.2 动态观测器

考虑一个 LTI 系统并表示为：

$$\dot{x} = Ax + Bu$$

$$y = Cx$$

对于该系统的连续时间观测器可以表示为：

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - C\hat{x}) \quad (4-4)$$

其中， \hat{x} 为 x 的估计值。上式中等号右端前两项可以认为是对原系统的预估，最后一项可以认为是利用观测数据对状态的校正。定义估计误差为 $\tilde{x} = x - \hat{x}$ ，那么可以得到：

$$\dot{\tilde{x}} = (A - LC)\tilde{x} \quad (4-5)$$

这表明如果选择合适的 L 值使得矩阵 $A - LC$ 是 Hurwitz 稳定的，那么估计误差将以指数方式收敛到零。

如果观测数据并不是在每一拍都可以获得（传感器数据更新较慢），那么可以采用如下的两步方式进行处理，如图12所示。

在没有得到传感器数据的时候，只使用预估方式进行估计：

$$\dot{\hat{x}} = A\hat{x} + Bu \quad (4-6)$$

当得到传感器数据后，则进行校正：

$$\hat{x}^+ = \hat{x}^- + L(y(t_k) - C\hat{x}^-) \quad (4-7)$$

其中， \hat{x}^- 表示预估值， \hat{x}^+ 表示校正值。注意到，在这种处理方式下，并不要求传感器数据以固定频率更新。

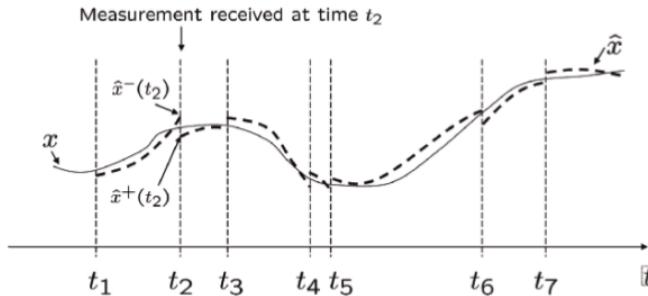


图 12 用动态观测器进行预估-校正方式的状态估计

4.1.3 连续-离散卡尔曼滤波器

考虑如下的系统：

$$\begin{aligned} \dot{x} &= Ax + Bu + \xi \\ y_k &= Cx_k + \eta_k \end{aligned} \quad (4-8)$$

其中，状态方程用连续形式表示，观测方程则用离散的形式表示，下标 k 表示第 k 次采样。 ξ 表示过程噪声，假设均值为 0、协方差为 \mathbf{Q} ，实际中 \mathbf{Q} 往往是未知的。 η_k 表示测量噪声，与传感器特性相关，假设其均值为 0、协方差为 \mathbf{R} ，实际中可以通过传感器标定估计出 \mathbf{R} 。

那么针对该系统，可以构造与动态观测器形式类似的连续-离散卡尔曼滤波器：

$$\dot{\hat{x}} = A\hat{x} + Bu$$

$$\hat{x}_k^+ = \hat{x}_k^- + L(y_k - C\hat{x}_k^-) \quad (4-9)$$

定义滤波器的估计误差 $\tilde{x} = x - \hat{x}$, 那么估计误差的协方差可以表示为:

$$P(t) = E \left\{ \tilde{x}(t)\tilde{x}(t)^T \right\} \quad (4-10)$$

显然 $P(t)$ 是一个对称、半正定的, 所以它的特征值均非负。当 $P(t)$ 具有较小的特征值时, 滤波器估计误差的方差较小, 所以这时估计误差也较小。考虑到

$$\text{tr}(P) = \sum \lambda_i$$

所以卡尔曼滤波器就是求解 L 使 $\text{tr}(P)$ 取得最小值, 从而获得在均方误差最小指标下的最优状态估计。

■ 传感器数据更新前

对估计误差求导可得:

$$\begin{aligned} \dot{\tilde{x}} &= \dot{x} - \dot{\hat{x}} \\ &= (Ax + Bu + \xi) - (A\hat{x} + Bu) \\ &= A\tilde{x} + \xi \end{aligned}$$

该微分方程的解为:

$$\tilde{x}(t) = e^{At}\tilde{x}_0 + \int_0^t e^{A(t-\tau)}\xi(\tau) d\tau \quad (4-11)$$

对 P 求导可得:

$$\begin{aligned} \dot{P} &= \frac{d}{dt} E\{\tilde{x}\tilde{x}^T\} \\ &= E\{\dot{\tilde{x}}\tilde{x}^T + \tilde{x}\dot{\tilde{x}}^T\} \\ &= E\{A\tilde{x}\tilde{x}^T + \xi\tilde{x}^T + \tilde{x}\tilde{x}^T A^T + \tilde{x}\xi^T\} \\ &= AP + PA^T + E\{\xi\tilde{x}^T\} + E\{\tilde{x}\xi^T\} \end{aligned}$$

其中,

$$\begin{aligned} E\{\xi\tilde{x}^T\} &= E \left\{ \xi(t)\tilde{x}_0 e^{A^T t} + \int_0^t \xi(\tau)\xi^T(\tau) e^{A^T(t-\tau)} d\tau \right\} \\ &= \frac{1}{2}Q \end{aligned}$$

代入上式可得:

$$\dot{P} = AP + PA^T + Q \quad (4-12)$$

■ 传感器数据更新时

在传感器数据更新时, 有:

$$\hat{x}_k^+ = \hat{x}_k^- + L(y_k - C\hat{x}_k^-)$$

假定 $\boldsymbol{\eta}$ 和 \mathbf{x} 是独立的，也就是 $E\{\tilde{\mathbf{x}}^-\boldsymbol{\eta}^T L^T\} = E\{L\boldsymbol{\eta}\tilde{\mathbf{x}}^{-T}\} = \mathbf{0}$ ，那么可以计算 \mathbf{P}^+ 为：

$$\begin{aligned}
 \mathbf{P}^+ &= E\{\tilde{\mathbf{x}}^+\tilde{\mathbf{x}}^{+T}\} \\
 &= E\left\{(\tilde{\mathbf{x}}^- - LC\tilde{\mathbf{x}}^- - L\boldsymbol{\eta})(\tilde{\mathbf{x}}^- - LC\tilde{\mathbf{x}}^- - L\boldsymbol{\eta})^T\right\} \\
 &= E\left\{\tilde{\mathbf{x}}^-\tilde{\mathbf{x}}^{-T} - \tilde{\mathbf{x}}^-\tilde{\mathbf{x}}^{-T}C^T L^T - \tilde{\mathbf{x}}^-\boldsymbol{\eta}^T L^T \right. \\
 &\quad \left.- LC\tilde{\mathbf{x}}^-\tilde{\mathbf{x}}^{-T} + LC\tilde{\mathbf{x}}^-\tilde{\mathbf{x}}^{-T}C^T L^T + LC\tilde{\mathbf{x}}^-\boldsymbol{\eta}^T L^T \right. \\
 &\quad \left.- L\boldsymbol{\eta}\tilde{\mathbf{x}}^{-T} + L\boldsymbol{\eta}\tilde{\mathbf{x}}^{-T}C^T L^T + L\boldsymbol{\eta}\boldsymbol{\eta}^T L^T\right\} \\
 &= \mathbf{P}^- - \mathbf{P}^- C^T L^T - L C \mathbf{P}^- + L C \mathbf{P}^- C^T L^T + L R L^T
 \end{aligned} \tag{4-13}$$

为了求解使 $\text{tr}(\mathbf{P}^+)$ 最小化时的 L 值，一个必要的条件为：

$$\begin{aligned}
 \frac{\partial}{\partial L} \text{tr}(\mathbf{P}^+) &= -\mathbf{P}^- C^T - \mathbf{P}^- C^T + 2L C \mathbf{P}^- C^T + 2L R = 0 \\
 \Rightarrow 2L(R + C \mathbf{P}^- C^T) &= 2\mathbf{P}^- C^T \\
 \Rightarrow L &= \mathbf{P}^- C^T (R + C \mathbf{P}^- C^T)^{-1}
 \end{aligned}$$

推导中用到以下矩阵关系：

$$\begin{aligned}
 \frac{\partial}{\partial A} \text{tr}(BAD) &= B^T D^T \\
 \frac{\partial}{\partial A} \text{tr}(ABA^T) &= 2AB, B = B^T
 \end{aligned}$$

将 L 的计算式代入式 (4-13) 中可得：

$$\mathbf{P}^+ = (\mathbf{I} - LC)\mathbf{P}^- \tag{4-14}$$

■ 总结

综合上述推导可以得到如下的卡尔曼滤波器迭代计算公式：

$$\begin{aligned}
 \dot{\hat{\mathbf{x}}} &= A\hat{\mathbf{x}} + Bu \\
 \dot{\mathbf{P}} &= AP + PA^T + Q \\
 L_k &= \mathbf{P}_k^- C^T (R + C \mathbf{P}_k^- C^T)^{-1} \\
 \mathbf{P}_k^+ &= (\mathbf{I} - L_k C) \mathbf{P}_k^- \\
 \hat{\mathbf{x}}_k^+ &= \hat{\mathbf{x}}_k^- + L_k (y_k - C\hat{\mathbf{x}}_k^-)
 \end{aligned}$$

4.1.4 从概率角度的理解

本节参考自 [How a Kalman filter works, in pictures](#) 这篇文章。

正态分布的概率密度函数可以表示为：

$$N(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

对于两个正态分布的随机变量相乘，则有：

$$N(x, \mu_0, \sigma_0) \cdot N(x, \mu_1, \sigma_1) = N(x, \mu', \sigma')$$

其中，相乘后概率分布的参数为：

$$\begin{aligned}\mu' &= \frac{\mu_1\sigma_0^2 + \mu_0\sigma_1^2}{\sigma_0^2 + \sigma_1^2} = \mu_0 + \frac{\sigma_0^2(\mu_1 - \mu_0)}{\sigma_0^2 + \sigma_1^2} \\ \sigma' &= \frac{\sigma_0^2\sigma_1^2}{\sigma_0^2 + \sigma_1^2} = \sigma_0^2 - \frac{\sigma_0^4}{\sigma_0^2 + \sigma_1^2}\end{aligned}$$

若定义参数 k 为

$$k = \frac{\sigma_0^2}{\sigma_0^2 + \sigma_1^2}$$

则有：

$$\begin{aligned}\mu' &= \mu_0 + k(\mu_1 - \mu_0) \\ \sigma' &= \sigma_0^2 - k\sigma_0^2\end{aligned}$$

4.1.5 更多讨论

以下再讨论几点不一样的地方。

■ 非线性模型和 EKF

若系统动态及测量模型都是非线性的：

$$\begin{aligned}\dot{x} &= f(x, u) + \xi \\ y_k &= h(x_k) + \eta_k\end{aligned}$$

仍可以通过以下的处理方式，求取非线性函数的雅可比矩阵作为 A 、 C ：

$$\begin{aligned}A &= \frac{\partial f}{\partial x}(\hat{x}, u) \\ C &= \frac{\partial h}{\partial x}(\hat{x}, u)\end{aligned}$$

■ 状态转移矩阵形式的 KF

若系统动态方程用状态转移矩阵的形式表示：

$$x_{k+1} = \Phi(T_s)x_k + G(T_s)u_k$$

其中，

$$\begin{aligned}\Phi(T_s) &= \Phi(t)|_{t=kT_s} = e^{At}|_{t=kT_s} \\ G(T_s) &= \int_0^{T_s} \Phi(\tau)B d\tau\end{aligned}$$

那么这种情况下的卡尔曼滤波器的预测方程应表示为：

$$\begin{aligned}\mathbf{x}_{k+1} &= \Phi(T_s)\mathbf{x}_k + \mathbf{G}(T_s)\mathbf{u}_k \\ \mathbf{P}_{k+1} &= \Phi(T_s)\mathbf{P}_k^+\Phi^T(T_s) + \mathbf{Q}\end{aligned}$$

4.2 四元数

本节内容主要参考文献[Quaternion and Rotation](#)

4.2.1 四元数基本计算

4.2.2 四元数微分方程

定理 1 假设 $q(t)$ 表示一单位四元数函数， $\omega(t)$ 表示由 $q(t)$ 确定的角速度。那么 $q(t)$ 的导数为：

$$\dot{q} = \frac{1}{2}\omega q$$

具体证明可以看参考文献。在实际运动过程中，获得的角速度矢量通常时在旋转坐标系下描述的（如在无人机转动时的机体系下，由三轴陀螺仪获得的角速度）。假设该角速度为 ω' ，则有 $\omega' = q^*\omega q$ 。将 $\omega = q\omega'q^*$ 待如到上述定理中有：

$$\dot{q} = \frac{1}{2}q\omega'$$

4.3 PX4 状态估计源码分析

4.3.1 attitude_estimator_q

通过阅读源码，可以将该状态估计的核心算法归纳如下：

$$\begin{aligned}\boldsymbol{\delta} &= K_P \mathbf{e} + K_I \int \mathbf{e} \\ \boldsymbol{\omega}^+ &= \boldsymbol{\omega}^- + \boldsymbol{\delta} \\ \dot{\hat{\mathbf{q}}} &= \frac{1}{2} \hat{\mathbf{q}} \times \boldsymbol{\omega}^+\end{aligned}$$

其中， \mathbf{e} 表示对姿态估计的角度偏差， $\boldsymbol{\delta}$ 表示对角速度的零偏修正量， $\boldsymbol{\omega}^-$ 表示陀螺读数， $\boldsymbol{\omega}^+$ 表示修正后的角速度。

可以看出，该算法的核心思想就是将状态估计的**所有误差**均视为一个**动态变化的角速度（陀螺读数）零偏值**，然后利用 PI 的方式计算该动态零偏值，并在**角速度环**进行修正¹，最后通过积分获得对四元数的状态估计值。

角度偏差 \mathbf{e} 主要是通过加速度计和磁罗盘进行计算的：利用磁罗盘读数和当前姿态计算地磁偏差，从而获得姿态的偏航误差；利用加速度计及计算的运动加速度，求得重力加速度并计算姿态偏差；最后将两个偏差向量（三轴角度偏差向量）按权重求和。

¹ 若单纯从角度、角速度、PI 控制来理解，会觉得修正量因该补偿在角度上。但是如果联想到 ADRC 的**动态补偿**思想，该算法也有异曲同工的地方。

五 轨迹跟踪

5.1 基础理论知识

5.1.1 L1 轨迹跟踪

本节主要参考文献A New Nonlinear Guidance Logic for Trajectory Tracking。

L1 轨迹跟踪控制算法的核心包括两个部分：

- (1) 在期望轨迹上寻找与飞机相距 L_1 的参考点，并计算速度矢量与目标向量间夹角 η
- (2) 计算侧向加速度指令为：

$$a_{s\text{cmd}} = 2 \frac{V^2}{L_1} \sin \eta \quad (5-1)$$

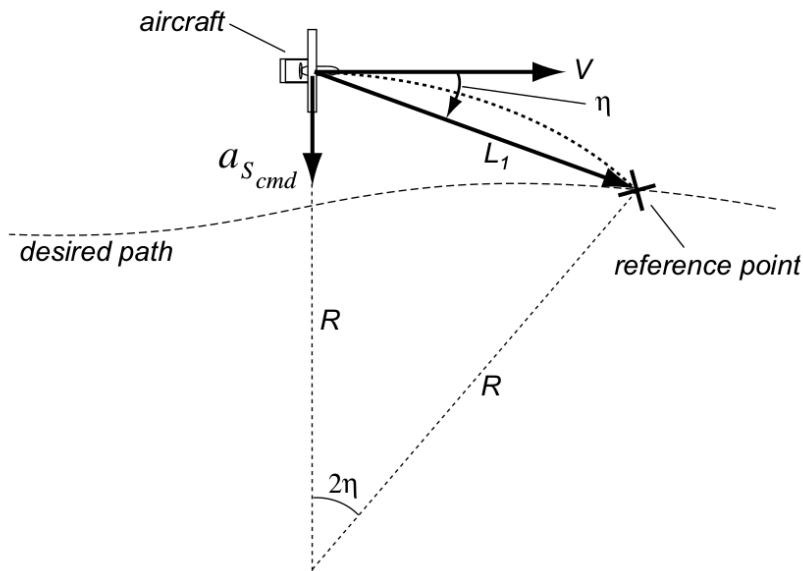


图 13 L1 轨迹跟踪计算示意图

通常轨迹跟踪都会针对直线轨迹和定点盘旋轨迹分别讨论。在直线轨迹跟踪中，侧偏距误差 d 可以用飞机质心与期望航线的垂直距离表示；在定点盘旋轨迹跟踪中，侧偏距误差 d 用飞机质心与圆的距离表示（质心到圆心距离 - 圆半径）。所以在应用 L1 轨迹跟踪算法时，需要考虑 $d > L_1$ 的情况，也就是飞机距离轨迹较远的情况。

对于直线轨迹跟踪，在图14中，可以对 η_1 进行限制，如 $-30^\circ \leq \text{sat}(\eta_1) \leq 30^\circ$ 。而对于盘旋轨迹，可以在 $d > L_1$ 时先进行直线轨迹跟踪，期望直线可以取为过质心的圆切线（圆半径可适当增加一个差值），在 $d < L_1$ 后再转入盘旋跟踪。另外在定点盘旋跟踪中需要利用速度向量与心心距向量判断旋转的方向（滚转的正负号）。

在Matlab 仿真中验证定点盘旋算法时，通过几何关系可以直接计算 η 角度，再按原始公式计算加速度指令。而论文中则采用 $\frac{V^2}{R} - \ddot{d}$ 进行近似，其中 \ddot{d} 在实现时似乎不

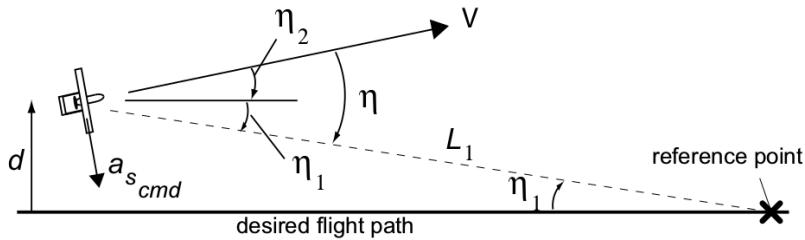


图 14 直线轨迹中 L1 轨迹跟踪计算示意图

方便计算。计算 η 角度时，涉及相交圆的求解，[参考这里的推导](#)。

最后，在 L1 算法中，给出了期望的加速度指令，如果假设飞机在进行高度不变的稳定盘旋时，侧向加速度指令等价于滚转角指令。而对于航向指令，论文中并没有提及，但是从 PX4 源码中可以看出，直线轨迹时航向指令指向 L1 跟踪点，盘旋轨迹时航向指令始终指向圆心点两种轨迹下虽然给出了偏航指令，但在内环均保持协调转弯的偏航控制策略，而未跟踪该偏航角。

5.1.2 进一步讨论

以下讨论只针对固定翼，且假设 1 具有高度保持系统，2 内环姿态控制的偏航通道按照协调转弯方式进行控制。多旋翼在做轨迹跟踪时，需要独立给出航向跟踪指令。

首先考虑偏盘旋轨迹跟踪问题，对于给定的飞行速度 V_a 和侧向加速度 a_s (等价滚转角)，在稳定盘旋时盘旋半径为 $R = V_a^2 / a_s$ 。显然对于给定的盘旋半径，可以计算盘旋时的侧向加速度。如果以跟踪盘旋轨迹的圆心点作为参考点，那么无人机相对该参考点的运动可以分解为两部分：1. 质心与圆心连线上的运动;2. 相对圆心的圆周运动。若以侧向加速度为控制量，那么有

$$a_{s\text{cmd}} = a_{\parallel} + a_{\perp} = f(d, V_a) + \frac{V_a^2}{R} \quad (5-2)$$

其中， d 表示侧向轨迹偏差， $f(d, V_a)$ 表示侧向偏差及飞行速度的函数，可以理解为对 d 的 PD 控制律。一种更形象的解释为： a_{\perp} 能够让无人机产生一个指定半径的圆轨迹， a_{\parallel} 能够让这个圆轨迹的圆心逐渐移动到指定的位置上。

对于直线轨迹跟踪，可以认为是半径为 ∞ 的圆轨迹，求解算法基本一致，只是需要在计算时限制 $f(d, V_a)$ 中的 d 值，以保证飞机能始终向前飞行，而不会对着航线飞行。

5.1.3 其他的几种轨迹跟踪

本节主要参考文献[Unmanned Aerial Vehicle Path Following](#)。

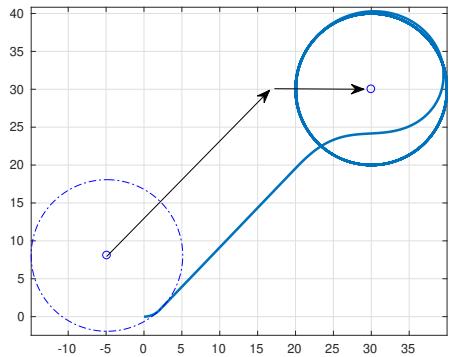
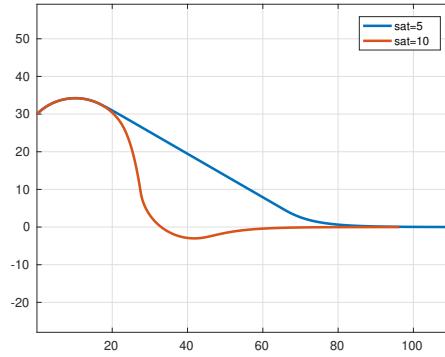


图 15 一种更形象的解释轨迹跟踪

图 16 直线轨迹跟踪, 对 d 取不同约束值的轨迹

5.2 MPC 控制及动态路径跟踪

模型预测控制 (MPC) 可以用于外环的路径跟踪控制上。若假设纵向控制中已经设计好了速度、高度控制器 (比如 TECS)，且假设偏航控制能够按照协调转弯的方式保持飞机无侧滑，那么二维路径跟踪问题就可以转换成以侧向加速度/滚转角为输入量、以姿态闭环为被控系统的 SISO 控制问题。若内环控制器具有较好的控制性能，那么姿态闭环可以动态线性化为一个二阶串级线性系统，这时可以应用 MPC 进行控制。

MPC 的核心问题就是在于对优化目标函数的设计及求解，若以师姐论文中的动态路径跟踪（避障）问题为例，优化目标函数可以设计为

$$J = \sum_i^N J^{(i)} = \sum_i^N \left(J_{path}^{(i)} + J_{angle}^{(i)} + J_{obs}^{(i)} + J_{final}^{(i)} \right)$$

其中， N 为滚动优化的长度， $J^{(i)}$ 为滚动优化中每一步的罚函数， $J_{path}^{(i)}$ 为路径跟踪误差， $J_{angle}^{(i)}$ 飞行航向跟踪误差， $J_{obs}^{(i)}$ 为避障罚函数 + $J_{final}^{(i)}$ 为终点吸引罚函数。具体代码 [在这里](#)。运行的结果为：

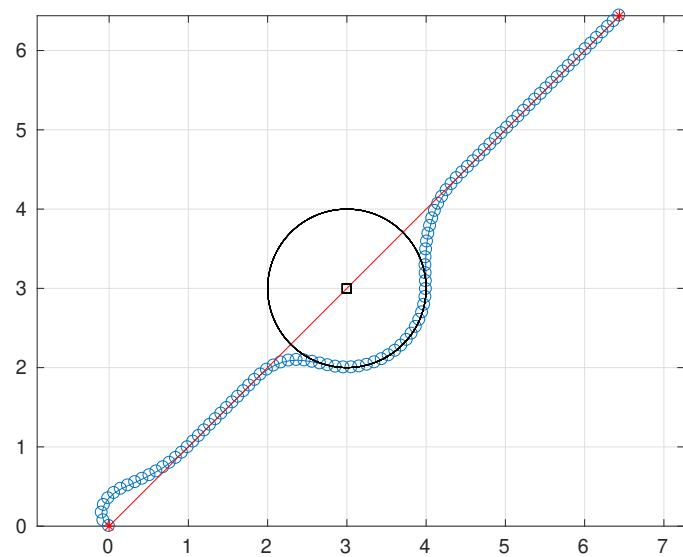


图 17 一个简单的基于 MPC 的动态路径跟踪

六 规划

6.1 最短路径算法

6.1.1 Floyd 算法

Floyd 算法采用动态规划思想， $f[k][i][j]$ 表示 i 和 j 之间可以通过编号为 $1 \dots k$ 的节点的最短路径。初值 $f[0][i][j]$ 为原图的邻接矩阵。

则 $f[k][i][j]$ 可以从 $f[k-1][i][j]$ 转移来，表示 i 到 j 不经过 k 这个节点。也可以从 $f[k-1][i][k] + f[k-1][k][j]$ 转移过来，表示经过 k 这个点。意思即

$$f[k][i][j] = \min(f[k-1][i][j], f[k-1][i][k] + f[k-1][k][j])$$

核心代码为

```

1   for (k=1; k<=n; k++)
2       for (i=1; i<=n; i++)
3           for (j=1; j<=n; j++)
4               if (e[i][j] > e[i][k] + e[k][j])
5                   e[i][j] = e[i][k] + e[k][j];

```

Floyd 算法能够求解有向图中任意两个点之间的最短路径，且只能在不存在负权环的情况下使用，时间复杂度 $O(N^3)$ ，空间复杂度为 $O(N^2)$ 。

6.1.2 Dijkstra 算法

Dijkstra 算法采用贪心思想，维护两个点集 A, B 。 A 点集代表已经求出源点到该点的最短路的点的集合， B 代表未求出源点到该点的最短路径的点的集合。维护一个向量 $d, d[i]$ 代表源点到点 i 的最短路径长度。不断进行以下操作：找出点集 B 中 $d[i]$ 最小的点，将这个点加入点集 A 中，然后用这个点到其邻接点的距离来更新向量 d ，直到点集 B 为空。

具体步骤为：

- (1) 将所有的顶点分为两部分：已知最短路程的顶点集合 P 和未知最短路径的顶点集合 Q 。最开始，已知最短路径的顶点集合 P 中只有源点一个顶点。我们这里用一个 $book[i]$ 数组来记录哪些点在集合 P 中。例如对于某个顶点 i ，如果 $book[i]$ 为 1 则表示这个顶点在集合 P 中，如果 $book[i]$ 为 0 则表示这个顶点在集合 Q 中。
- (2) 设置源点 s 到自己的最短路径为 0 即 $dis=0$ 。若存在源点有能直接到达的顶点 i ，则把 $dis[i]$ 设为 $e[s][i]$ 。同时把所有其它（源点不能直接到达的）顶点的最短路径设为 ∞ 。
- (3) 在集合 Q 的所有顶点中选择一个离源点 s 最近的顶点 u （即 $dis[u]$ 最小）加入

到集合 P 。并考察所有以点 u 为起点的边，对每一条边进行松弛操作。例如存在一条从 u 到 v 的边，那么可以通过将边 $u \rightarrow v$ 添加到尾部来拓展一条从 s 到 v 的路径，这条路径的长度是 $dis[u] + e[u][v]$ 。如果这个值比目前已知的 $dis[v]$ 的值要小，我们可以用新值来替代当前 $dis[v]$ 中的值。

- (4) 重复第 3 步，如果集合 Q 为空，算法结束。最终 dis 数组中的值就是源点到所有顶点的最短路径。

Dijkstra 算法能够求解图中单源点到其他所有点的最短路径。时间复杂度为 $O(|V|^2)$, $|V|$ 表示顶点个数。如果采用优先队列，时间复杂度为 $O(|E| + |V|\log|V|)$, $|E|$ 表示边数。

Algorithm 1 Dijkstra algorithm

```

1: for each vertex  $v \in V[G]$  do
2:    $d[v] := \text{infinity}$ 
3:    $\text{previous}[v] := \text{undefined}$ 
4: end for
5:  $d[s] := 0$ 
6:  $S := \text{empty set}$ 
7:  $Q := \text{set of all vertices}$ 
8: while  $Q$  is not empty do
9:    $u := \text{Extract\_Min}(Q)$ 
10:   $S.append(u)$ 
11:  for each edge outgoing from  $u$  as  $(u,v)$  do
12:    if  $d[v] > d[u] + w(u,v)$  then
13:       $d[v] := d[u] + w(u,v)$ 
14:       $\text{previous}[v] := u$ 
15:    end if
16:  end for
17: end while

```

6.1.3 Bellman-Ford 算法

具体步骤为：

- (1) 创建源顶点 v 到图中所有顶点的距离的集合 $distSet$, 为图中的所有顶点指定一个距离值，初始均为 Infinite，源顶点距离为 0；
- (2) 计算最短路径，执行 $V-1$ 次遍历：对于图中的每条边，如果起点 u 的距离 d 加上边的权值 w 小于终点 v 的距离 d ，则更新终点 v 的距离值 d ；
- (3) 检测图中是否有负权边形成了环：遍历图中的所有边，计算 u 至 v 的距离，如果对于 v 存在更小的距离，则说明存在环；

Bellman-Ford 算法也能解决单源最短路问题，且对边的情况没有要求，不仅可以处理负权边，还能处理负环。时间复杂度为 $O(|V| \cdot |E|)$

6.1.4 A* 算法

个人理解 A* 算法只是针对搜索方向的一种启发式处理方法，在寻找最短路径的搜索过程中，通常都是以 Dijkstra 算法为基础进行的。具体的伪代码为：

```

1 function A*(start, goal)
2     // The set of nodes already evaluated
3     closedSet := {}
4
5     // The set of currently discovered nodes that are not evaluated
6     // yet.
7     // Initially, only the start node is known.
8     openSet := {start}
9
10    // For each node, which node it can most efficiently be reached
11    // from.
12    // If a node can be reached from many nodes, cameFrom will
13    // eventually contain the
14    // most efficient previous step.
15    cameFrom := an empty map
16
17    // For each node, the cost of getting from the start node to that
18    // node.
19    gScore := map with default value of Infinity
20
21    // The cost of going from start to start is zero.
22    gScore[start] := 0
23
24    // For each node, the total cost of getting from the start node to
25    // the goal
26    // by passing by that node. That value is partly known, partly
27    // heuristic.
28    fScore := map with default value of Infinity
29
30    // For the first node, that value is completely heuristic.
31    fScore[start] := heuristic_cost_estimate(start, goal)
32
33    while openSet is not empty
34        current := the node in openSet having the lowest fScore[] value
35        if current = goal
36            return reconstruct_path(cameFrom, current)
37
38        openSet.Remove(current)
39        closedSet.Add(current)

```

```

34
35     for each neighbor of current
36         if neighbor in closedSet
37             continue // Ignore the neighbor which is already
38             evaluated.
39
40         if neighbor not in openSet // Discover a new node
41             openSet.Add(neighbor)
42
43             // The distance from start to a neighbor
44             //the "dist_between" function may vary as per the solution
45             // requirements.
46             tentative_gScore := gScore[current] + dist_between(current,
47             neighbor)
48             if tentative_gScore >= gScore[neighbor]
49                 continue // This is not a better path.
50
51             // This path is the best until now. Record it!
52             cameFrom[neighbor] := current
53             gScore[neighbor] := tentative_gScore
54             fScore[neighbor] := gScore[neighbor] +
55             heuristic_cost_estimate(neighbor, goal)
56
57             return failure
58
59
60     function reconstruct_path(cameFrom, current)
61         total_path := [current]
62         while current in cameFrom.Keys:
63             current := cameFrom[current]
64             total_path.append(current)
65         return total_path

```

上述伪代码可以用于移动机器人路径搜索。其中 closedSet 表示已经搜索过的位置，openSet 表示待搜索的位置。从 openSet 中按照 fScore 的值找到最近的待搜索点，从 openSet 中移入 closedSet 中。更新该点相邻可达点的 fScore 值和 cameFrom 表，同时将相邻点（不在 closedSet）加入 openSet 中。不断重复直到搜索到达目标点，然后根据 cameFrom 表逆推得到最短路径。

与 Dijkstra 算法相比，A* 算法最核心的地方就是对距离函数的取值：

$$f(n) = g(n) + h(n)$$

其中， $g(n)$ 就是 Dijkstra 算法中计算出的从源点到当前点的距离， $h(n)$ 则表示当前点到目标点的距离估计。常见的估计函数有：欧几里得距离、曼哈顿距离、切比雪夫距离；这个公式遵循以下特性：

- (1) 如果 $g(n)$ 为 0，则转化为使用贪心策略的深度优先搜索，速度最快，但可能得不出最优解；
- (2) 如果 $h(n)$ 为 0，则转化为单源最短路径问题，即 Dijkstra 算法；
- (3) **如果 $h(n)$ 不大于顶点 n 到目标点的真实距离，则一定可以求出最优解。**而且 $h(n)$ 越小，需要计算的节点越多，算法效率越低。

利用该估计值启发搜索方向。这是我用 Matlab 编写的[A* 代码](#)。为了便于 Matlab 搜索和定位，代码中将二维坐标转换为了一维坐标处理。

6.2 路径搜索算法

6.2.1 PRM 算法

概率路图 (Probabilistic RoadMap, PRM) 算法。基本步骤分为两步：先对地图进行预处理建图，生成可行路径无向图，然后查询源点、终点在图上对应的最短距离。

■ PRM 算法预处理步骤

- (1) 地图随机采样，生成中间路径点。
- (2) 按照不同的方法，在路径点之间生成路径
- (3) 对路径进行碰撞检测。

在 PRM 算法中，常规方式是逐点采样，然后按照半径距离 r 连接到已生成的图中；一种简化方法 (sPRM) 是一次生成 N 个采样点，然后再逐点生成路径。此外，对最近若干点的挑选方式也有不同处理。常规是按照给定的半径球内寻点。k-Nearest (s)PRM 始终搜寻最近的 k 个点生成路径。Bounded-degree (s)PRM 则是对常规方式的球内点添加了 k 个最近上限。

6.2.2 RRT 算法

快速扩展随机树 (Rapidly-exploring Random Trees, RRT) 算法。本节主要参考[Rahul Kala](#)的代码以及[这篇博客](#)。

■ RRT 算法步骤

- (1) 初始化随机树，确定初始点、目标点，地图尺寸、地图障碍物位置等
- (2) 随机采样。按照一定的概率在地图中挑选随机点或是用目标点作为当前采样点 q_{rand}
- (3) 选取最近节点。在当前随机树中找出与采样点最近的节点，作为扩展节点 $q_{nearest}$
- (4) 随机树扩展。沿 $q_{nearest}$ 和 q_{rand} 方向，按照一定步长生成新节点 q_{new}
- (5) 碰撞检测。若 $\overrightarrow{q_{nearest}q_{new}}$ 上的点都与障碍物无碰撞，则将 q_{new} 加入随机树中。

(6) 若 q_{new} 距离目标点足够近则停止搜索，否则跳转步骤 2。

在实际使用是，还需要考虑：如何进行随机采样？如何定义“最近”以及如何快速搜索到最近节点 (Kd-Tree)？如何进行扩展等。博客中举了一个车型机器人的例子，显然车的前后移动要比旋转、航向移动要方便，对应与更“近”。

■ Bidirectional RRT/RRT Connect

单源点开始搜索的 RRT 算法在搜索效率上仍显的较慢，因此有人提出如下的双向 RRT 搜索算法。基本思想可以理解为：先对源点做一次 RRT 搜索，然后用扩展点对终

```

1.  $V_1 \leftarrow \{q_{init}\}; E_1 \leftarrow \emptyset; G_1 \leftarrow (V_1, E_1);$ 
2.  $V_2 \leftarrow \{q_{goal}\}; E_2 \leftarrow \emptyset; G_2 \leftarrow (V_2, E_2); i \leftarrow 0;$ 
3. while  $i < N$  do
4.    $q_{rand} \leftarrow \text{Sample}(i); i \leftarrow i + 1;$ 
5.    $q_{nearest} \leftarrow \text{Nearst}(G_1, q_{rand});$ 
6.    $q_{new} \leftarrow \text{Steer}(q_{nearest}, q_{rand});$ 
7.   if  $\text{ObstacleFree}(q_{nearest}, q_{new})$  then
8.      $V_1 \leftarrow V_1 \cup \{q_{new}\};$ 
9.      $E_1 \leftarrow E_1 \cup \{(q_{nearest}, q_{new})\};$ 
10.     $q'_{nearest} \leftarrow \text{Nearst}(G_2, q_{new});$ 
11.     $q'_{new} \leftarrow \text{Steer}(q'_{nearest}, q_{new});$ 
12.    if  $\text{ObstacleFree}(q'_{nearest}, q'_{new})$  then
13.       $V_2 \leftarrow V_2 \cup \{q'_{new}\};$ 
14.       $E_2 \leftarrow E_2 \cup \{(q'_{nearest}, q'_{new})\};$ 
15.      do
16.         $q''_{new} \leftarrow \text{Steer}(q'_{new}, q_{new});$ 
17.        if  $\text{ObstacleFree}(q''_{new}, q'_{new})$  then
18.           $V_2 \leftarrow V_2 \cup \{q''_{new}\};$ 
19.           $E_2 \leftarrow E_2 \cup \{(q''_{new}, q'_{new})\};$ 
20.           $q'_{new} \leftarrow q''_{new};$ 
21.        else break;
22.        while  $not q'_{new} = q_{new}$ 
23.        if  $q'_{new} = q_{new}$  then return  $(V_1, E_1);$ 
24.      if  $|V_2| < |V_1|$  then Swap $(V_1, V_2);$ 
```

图 18 RRT-Connect 算法步骤

点做一次 RRT 搜索，然后用源点 RRT 的扩展点与终点 RRT 的扩展点再做连续 RRT 拓展，直到遇到障碍物。然后重复上述步骤。为了平衡两棵随机树的节点，每一步搜索中的第一次的 RRT 可以对节点数多的进行。

6.2.3 RRT* 算法

最优 RRT(RRT*) 算法，主要参考[这篇论文](#)。简单来说，RRT* 算法的“最优”性体现在对随机树的动态优化上，基本思想为：

- (1) 随机树扩展。对于待扩展的新节点 q_{new} ，不是简单的生成从 q_{nearest} 到 q_{new} 的路径，而是在 q_{new} 的一个邻域内搜索所有节点 q_{near} ，使得源点经过 q_{near} 到 q_{new} 的路径最短，这时再添加 q_{near} 到 q_{new} 路径。
- (2) 随机树修正。对 q_{new} 邻域内所有节点 q_{near} 再做一次搜索，若源点经过新节点 q_{new} 到 q_{near} 的距离更短，则变更 q_{near} 的父节点为 q_{new} ，且替换原路径。

```
Algorithm 6: RRT*
1  $V \leftarrow \{x_{\text{init}}\}; E \leftarrow \emptyset;$ 
2 for  $i = 1, \dots, n$  do
3    $x_{\text{rand}} \leftarrow \text{SampleFree}_i;$ 
4    $x_{\text{nearest}} \leftarrow \text{Nearest}(G = (V, E), x_{\text{rand}});$ 
5    $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}});$ 
6   if  $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$  then
7      $X_{\text{near}} \leftarrow \text{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\});$ 
8      $V \leftarrow V \cup \{x_{\text{new}}\};$ 
9      $x_{\text{min}} \leftarrow x_{\text{nearest}}; c_{\text{min}} \leftarrow \text{Cost}(x_{\text{nearest}}) + c(\text{Line}(x_{\text{nearest}}, x_{\text{new}}));$ 
10    foreach  $x_{\text{near}} \in X_{\text{near}}$  do // Connect along a minimum-cost path
11      if  $\text{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \text{Cost}(x_{\text{near}}) + c(\text{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$  then
12         $x_{\text{min}} \leftarrow x_{\text{near}}; c_{\text{min}} \leftarrow \text{Cost}(x_{\text{near}}) + c(\text{Line}(x_{\text{near}}, x_{\text{new}}))$ 
13     $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\};$ 
14    foreach  $x_{\text{near}} \in X_{\text{near}}$  do // Rewire the tree
15      if  $\text{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \text{Cost}(x_{\text{new}}) + c(\text{Line}(x_{\text{new}}, x_{\text{near}})) < \text{Cost}(x_{\text{near}})$ 
16      then  $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}});$ 
            $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$ 
17 return  $G = (V, E);$ 
```

图 19 RRT* 算法伪代码

■ 疑问

RRT* 算法在每一步搜索中都需要寻找 q_{new} 的邻域节点，这个搜索的代价是不是太大？如果按照常规 RRT 算法生成节点和路径，最后再做一次全局的最短路径搜索，性能对比如何？

6.3 路径曲线生成算法

6.3.1 Dubins path

对于平面路径生成，可以用 $(x, y, \psi)'$ 表示状态，其中 x, y 为轨迹点的平面坐标位置， ψ 表示轨迹点运动的轨迹偏角（对无人机，假设无侧滑角飞行）。Dubins 曲线采用圆弧和直线连接源点和终点，对于包含直线的 Dubins 曲线，按照源点转弯方向和终点转弯方向，可以分为 RSR, RSL, LSR, LSL 共四种曲线形式。

具体算法思想可以参考[这篇论文](#)，具体的 MATLAB 实现可以看[这里](#)，主要注意下坐标系 (x 轴指向北) 与论文中不一样，实现过程比较简单，都是几何运算。

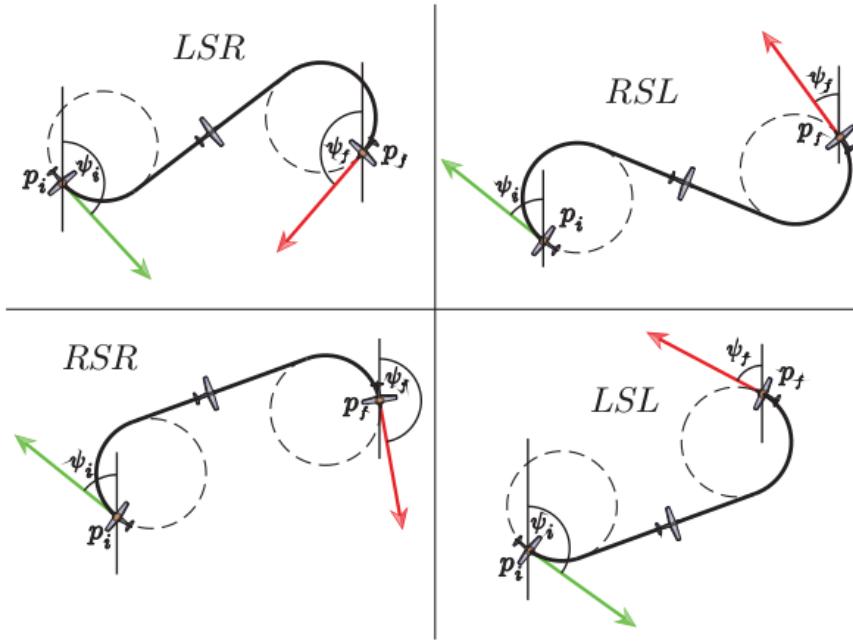


图 20 四种 Dubins 曲线形式

基本思路都是：

- (1) 按照路径形式确定源点、终点的转弯圆心
- (2) 按照路径形式确定两圆的切线方向、航迹偏角
- (3) 计算圆上切点
- (4) 计算转弯旋转角度的变化增量

关于如何根据条件状态直接选择路程最短的形式，可以参考王师姐论文。

6.3.2 Bezier 曲线和 B-spline 曲线

■ 贝塞尔曲线

参考[这篇博客](#)。给定源点 P_0 和终点 P_1 ，那么一阶 Bezier 曲线就是这两个点的连线，连线上的点可以描述为：

$$P = (1 - t)P_0 + tP_1$$

其中， $t: 0 \Rightarrow 1$ 。这个形式相当于一个一阶插值。类似的，二阶 Bezier 曲线中，一共需要 3 个控制点 P_0, P_1, P_2 ，先计算 P_0 和 P_1 的曲线点 P_0^1 ，再计算 P_1 和 P_2 的曲线点 P_1^1 ，最后计算 P_0^1 和 P_1^1 的曲线点就是二阶曲线上的点。

$$\begin{aligned} P &= (1 - t)[(1 - t)P_0 + tP_1] + t[(1 - t)P_1 + tP_2] \\ &= (1 - t)^2 P_0 + 2t(1 - t)P_1 + t^2 tP_2 \end{aligned}$$

贝塞尔曲线存在的问题：

- (1) 确定了多边形的顶点数 (m 个)，也就决定了所定义的 Bezier 曲线的阶次 ($m-1$ 次)，不够灵活。
- (2) 当顶点数 (m) 较大时，曲线的阶次将比较高。此时，多边形对曲线形状的控制将明显减弱。
- (3) Bezier 的调和函数的值，在开区间 $(0,1)$ 内均不为 0。因此，所定义的曲线在 $(0 < t < 1)$ 的区间内的任何一点均要受到全部顶点的影响，即改变其中任一个顶点的位置，都将对整条曲线产生影响，因此对曲线进行局部修改是不可能的。

■ B 样条曲线

可以参考[这篇博客](#)。讲的比较清楚，而且代码也可以运行通过。

一般使用其中的准均匀 B 样条曲线，需要特别注意其中求取节点的地方，包括数量及其分布。节点可以理解为在原控制点上的扩充，用于计算基函数。控制点就是给定的控制多边形的定点。

6.3.3 其他形式曲线

■ Reeds-Shepp 曲线

与 Dubins 曲线相比，RS 曲线中机器人既能够前向运动，也能够后向运动（类似具有倒车的性质）。共计 9 类 46 种曲线直线组合结果。

■ Balkcom-Mason 曲线

与 RS 曲线相比，Balkcom-Mason 曲线中机器人可以绕中心旋转（类似双轮机器人差动转向的性质）。

■ Pythagorean Hodograph 曲线

PH 曲线是一种具有有理特性的参数化多项式曲线。具体计算可以根据师姐论文中的方法，在复平面内进行计算。

6.3.4 多项式轨迹

通过路径搜索算法可以找到一系列的离散路径点，通过多项式曲线将这些点连接起来形成多项式轨迹。本节的主要算法参考[论文 A](#) 和[论文 B](#)。其中对于多项式轨迹的基本内容可以阅读教材 Trajectory Planning for Automatic Machines and Robots Springer 第二章中的相关部分。

对于本节的多项式轨迹生成问题，可以转化为求解一个二次规划 (QP, Quadratic Program) 问题。假设多项式的系数可以表示为 $\mathbf{p} = [p_0, p_1, \dots, p_n]^T$ ，多项式函数为 $P(t)$ 那么采用论文 A 中的目标函数：

$$J(T) = \int_0^T c_0 P(t)^2 + c_1 P'(t)^2 + c_2 P''(t)^2 + \dots + c_N P^{(N)}(t)^2 dt = \mathbf{p}^T Q(T) \mathbf{p} \quad (6-1)$$

其中， $Q(T)$ 为 Hessian 矩阵，可以通过简单的微积分运算求解。 T 表示每一段多项式

轨迹的总时间。对于 $M+1$ 个离散路径点，共需要生成 M 段多项式轨迹，这样总的目标函数可以表示为：

$$J_{total} = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \begin{bmatrix} Q_1(T_1) & & \\ & \ddots & \\ & & Q_M(T_M) \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix} \quad (6-2)$$

对于二次规划问题可以通过现成的工具箱进行求解，所以剩下的问题是如何将轨迹生成中的边界约束转化为二次规划的约束形式。此外需要注意的是上述目标函数中需要指定每一段的轨迹时间，这些时间量可以作为上一层优化问题的优化变量。

从无人机飞行动力学考虑，一种合理的假设是飞行过程中无人机位置、速度、加速度是连续变化的，每一段的边界条件也是位置、速度、加速度这三个变量。那么对于任意一段多项式轨迹，初始边界条件 3 个，终端边界条件 3 个，因此需要最少 5 次多项式（6 个系数）才能满足所有边界条件。对每一段多项式的边界条件可以描述为：

$$\mathbf{A}_i \mathbf{p}_i = \mathbf{d}_i \quad (6-3)$$

对于 5 次多项式，考虑 P、V、A 三项边界时，上述三项展开为：

$$\mathbf{A}_i = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_T & t_T^2 & t_T^3 & t_T^4 & t_T^5 \\ 0 & 1 & 2t_T & 3t_T^2 & 4t_T^3 & 5t_T^4 \\ 0 & 0 & 2 & 6t_T & 12t_T^2 & 20t_T^3 \end{bmatrix} \quad (6-4)$$

$$\mathbf{d}_i = [P_0, V_0, A_0, P_T, V_T, A_T]^T \quad (6-5)$$

在实际规划过程中，常常并不知道无人机到达中间离散路径点位置时的速度、加速度，因此需要使用连续边界条件，即

$$A_{T,i} \mathbf{p}_i = A_{0,i+1} \mathbf{p}_{i+1} \quad (6-6)$$

七 C++

7.1 基础算法

排序算法的稳定性：排序前后相同元素的相对位置不变，则称排序算法是稳定的；否则排序算法是不稳定的

7.1.1 快速排序

平均时间复杂度 $O(n \log n)$, 最快情况为 $O(n^2)$

```

0
1 template<typename T>
2 void quick_sort(vector<T> &nums, int left, int right){
3     if (left >= right) return;
4     int low = left;
5     int high = right;
6     T val = nums[left];
7     while(left < right) {
8         while(left < right && nums[right] >= val) right--;
9         nums[left] = nums[right];
10        while(left < right && nums[left] <= val) left++;
11        nums[right] = nums[left];
12    }
13    nums[left] = val;
14    quick_sort(nums, low, left-1);
15    quick_sort(nums, left+1, high);
16 }
```

7.1.2 堆排序

平均时间复杂度 $O(n \log n)$, 最快情况为 $O(n \log n)$

```

0
1 template<typename T>
2 void heap_max(vector<T> &nums, int start, int end) {
3     int dad = start;
4     int son = dad * 2 + 1;
5     while(son <= end) {
6         if (son+1 <= end && nums[son] < nums[son+1]) son++;
7         if (nums[dad] >= nums[son]) return;
8         else {
9             swap(nums[dad], nums[son]);
```

```

10         dad = son;
11         son = son * 2 + 1;
12     }
13 }
14 return;
15 }
16
17 template<typename T>
18 void heap_sort(vector<T> &nums, int start, int end) {
19     for(int i = end; i >= 0; --i) {
20         heap_max(nums, i, end);
21     }
22     for(int i=0; i <= end; ++i) {
23         swap(nums[0], nums[end-i]);
24         heap_max(nums, 0, end-1-i);
25     }
26 }
```

7.1.3 计数排序

平均时间复杂度和最坏时间复杂度都是 $O(n + k)$, 稳定排序。

```

0 void countsort(vector<int> &nums, int start, int end) {
1     int min = 0x7fffffff;
2     int max = 0x80000000;
3     for(auto n:nums) {
4         if (min>n) min=n;
5         if (max<n) max=n;
6     }
7     vector<int> counts(max-min+1, 0);
8     vector<int> ans(nums.size(), 0);
9     for(auto n:nums) {
10        counts[n-min]++;
11    }
12    for(int i=1; i<counts.size(); ++i) counts[i] += counts[i-1];
13    for(int i=0; i<nums.size(); ++i) {
14        ans[counts[nums[i]-min]-1] = nums[i]; //--对应于重复的数字
15    }
16    for(int i=0; i<nums.size(); ++i) nums[i] = ans[i];
17 }
```

7.2 语言使用

7.2.1 const 关键字的使用

参考 1 参考 2

1. const 修饰一个变量，表明其为一个常量，初始化赋值后不能被修改。
2. const 修饰指针，可以定义为一个指针常量 (`int * const p`)，其只能指向初始化的地址；可以定义为一个常量指针 (`const int * p`)，其只能指向常量；最后可以定义一个指向常量的指针常量 (`const int * const p`)
3. const 修饰函数的传入参数，表明其在函数体内部不允许被修改。一般只对非内部类型，使用常量引用的形式 `const &`
4. const 修饰类成员函数返回值，表明其不能作为左值使用
5. const 修饰类成员函数，表明该函数不允许修改类的成员变量。任何不会修改数据成员的函数都应该声明为 `const` 类型: `int fun(int a) const;`

7.2.2 用标准库构造堆

参考

1. 对于一个 `vector`，可以使用 `std::make_heap` 生成一个堆，默认是大顶堆 (`less<int>()`)，如果使用小顶堆，比较器使用 `greater<int>()`
2. `std::pop_heap`，将堆顶元素移到 `vector` 的末尾，同时进行调整。小顶堆使用 `greater<int>()`
3. `std::push_heap`，将 `vector` 末尾的元素视为新插入的元素，并进行调整。小顶堆使用 `greater<int>()`
4. 调整后的结果并不是对 `vector` 进行排序。`std::sort_heap` 则是进行排序。

八 机器学习

这一部分主要是对 JDAI 中级培训的内容总结，对于课程中重要但又未详细讲解的地方，也会添加一些其他教材中的内容作为补充。

8.1 准备知识

8.1.1 概述

机器学习的基本过程：

1. 监督学习（有已知样本）：信息获取与预处理-> 特征提取与特征选择-> 分类器设计（训练）-> 分类决策（识别）
2. 非监督学习（无已知样本）：信息获取与预处理-> 特征提取与特征选择-> 聚类（自学习）-> 结果解释

机器学习技术主要包括：

1. 分类：决策树、K 近邻、逻辑回归、支持向量机、神经网络等。
2. 聚类：C 均值、模糊 C 均值、改进的模糊 C 均值等。

机器学习的一些基本名词：

- (1) 特征 (Feature)：样本的任何可区分且可观测的方面。包括定量、定性特征，但通常都转化为定量特征。特征向量、特征空间。
- (2) 分类器：能够将每一个样本都分到某个类别中（或者拒绝）的算法。

8.1.2 统计学习三要素

统计学习方法 = 模型 + 策略 + 算法。以下对监督学习进行阐述。

模型就是所要学习到的条件概率分布或是决策函数。假设空间 (hypothesis space) 就是所有可能模型组成的集合。决策函数表示的模型为非概率模型，条件概率分布表示的模型为概率模型。

策略可以理解为确定优化目标函数，即按照什么样的准则来学习/选择最优模型。损失函数度量模型一次预测的好坏，风险函数度量平均意义下的模型预测好坏。学习的目标是选择期望风险最小的模型。

训练数据集上的平均损失为经验风险。经验风险最小化、结构风险最小化。结构风险最小化等价于正则化。

损失函数、代价函数、目标函数的区别

- (1) 损失函数 (loss fun.)，也可以理解为误差函数 (error fun.)，针对单个样本点。
- (2) 代价函数 (cost fun.)，针对整个样本集/训练集而言。可以认为是所有样本点损失函数的平均。
- (3) 目标函数 (object fun.)，一般是代价函数 + 正则项。

算法是指学习模型的具体计算方法。

8.1.3 模型评估

模型评估的必要性?

■ 过拟合问题

Mitchell 在 Machine Learning 中给了一个定义: Given a hypothesis space H , a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

在这个定义里并没有严格证明一定存在 h' , 所以可以理解过拟合也是一个相对而言的问题。简单来说, 过拟合指所选取的模型在已知数据集上的预测效果很好, 但对未知数据预测效果很差的现象。过拟合本质是一味追求在已知数据上的预测能力, 选取的模型复杂度往往比真实模型更高, 使得将样本数据本身所包含的噪声也作为真实数据进行学习。

什么是模型复杂度? 如何解决过拟合问题?

■ 混淆矩阵

针对实际问题时, 主要关注混淆矩阵中非对角线上的元素, 也就是分类器判断错误的样本。

多分类问题也可以生成混淆矩阵, 只不过不再是 2×2 的矩阵 (类别 True、False), 而是类别数的 $n \times n$ 的矩阵。

■ ROC 曲线

坐标横轴为 False Positive Rate(FPR, FP/P), 坐标纵轴为 True Positive Rate(TPR, TP/N)

8.1.4 归一化与标准化

8.1.5 交叉熵代价函数

■ 信息熵: 消除不确定性所需要的信息量的度量。

令事件 x 的信息量表示为 $I(x)$, 假设信息量与概率有关。考虑两个独立不相关事件 x 和 y , 显然有 $I(x,y) = I(x) + I(y)$, 而 $p(x,y) = p(x) * p(y)$ 。所以可以定义信息量为:

$$I(x) = -\log p(x)$$

取负号是为了保证信息量为正。对数函数可以任意选取, 信息论中以 2 为底, 机器学习通常以 e 为底。对于随机变量 X , 信息熵为:

$$H(X) = \mathbb{E}[-\log p(X)] = -\sum_i p(x_i) \log p(x_i)$$

■ 相对熵 (KL 散度): 描述两个概率分布的非对称性度量。定义为

$$D_{KL}(p||q) = -\sum_i p(x_i) \log \frac{q(x_i)}{p(x_i)}$$

尽管从直觉上 KL 散度是个度量或距离函数, 但是它实际上并不是一个真正的度量或距离。因为 KL 散度不具有对称性: 从分布 P 到 Q 的距离通常并不等于从 Q 到 P 的距离。 $D_{KL}(p||q) \neq D_{KL}(q||p)$

通常 $p(x)$ 表示真实 (样本) 概率分布, $q(x)$ 表示模型概率分布。显然可以使用相对熵作为代价函数, 使得模型生成的概率分布逼近真实样本的概率分布。所以进一步, 对 $D_{KL}(p||q)$ 变换可得

$$\begin{aligned} D_{KL}(p||q) &= -\sum_i p(x_i) \log q(x_i) + \sum_i p(x_i) \log p(x_i) \\ &= H(p, q) - H(p) \end{aligned}$$

其中 $H(p, q) = -\sum_i p(x_i) \log q(x_i)$ 即表示交叉熵。由于真实 (样本) 概率分布是固定的, 所以交叉熵也可以作为代价函数, 计算更简便。

对于二分类问题, 单个样本点上的损失函数可以表示为二元交叉熵的形式:

$$L = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

8.1.6 对数损失函数

对数损失函数为

$$L(Y, P(Y|X)) = -\log P(Y|X)$$

8.2 梯度下降及线性回归

8.2.1 基本算法

求解无约束最优化问题:

$$\min_{x \in R^n} f(x)$$

从初始值 $x^{(0)}$ 开始, 按照负梯度方向逐步迭代直到收敛。令 $g_k = \nabla f(x^{(k)})$ 表示 $f(x)$ 在 $x^{(k)}$ 处的梯度, 则迭代公式为:

$$x^{(k+1)} = x^{(k)} - \alpha_k g_k$$

当 $\|g_k\| < \epsilon$ 、 $\|f(x^{(k+1)}) - f(x^{(k)})\| < \epsilon$ 或者 $\|x^{(k+1)} - x^{(k)}\| < \epsilon$ 时停止迭代。

需要注意的是关于学习率或者步长 α , 通常选取为固定值, 但是在《统计学习方法》的附录中, α 是通过一维搜索确定的: $\alpha_k = \arg \min f(x^{(k)} - \alpha g_k)$

8.2.2 线性回归

假设共 m 个样本点，每个样本具有 n 维特征，则定义 $X \in R^{m \times (n+1)}$, $Y \in R^{m \times 1}$, $\theta \in R^{(n+1) \times 1}$, 其中 X 包含了常值特征， θ 包含了偏置系数。

$$\begin{aligned} L(\theta) &= \frac{1}{m}(Y - X\theta)^T(Y - X\theta) \\ &= \frac{1}{m} \sum_{i=1}^m (y_i - x_i\theta)^2 \end{aligned}$$

梯度为：

$$\nabla_{\theta} L(\theta) = -\frac{2}{m}X^T(Y - X\theta)$$

★ 思考与总结

- (1) 每一个样本点按照行向量的形式表示，这可以从代码的角度理解：新数据都是先填充完一行，在堆叠。例如二维 vector。
- (2) 可以预先增加对样本的标准化/归一化处理，这样求解的步长、收敛阈值等都可以固定，同时可以显著提高收敛速度。

8.3 Logistic Regression, LR

二分类 LR 模型为：

$$\begin{aligned} P(y=1|x) &= \frac{e^{w \cdot x + b}}{1 + e^{w \cdot x + b}} \\ P(y=0|x) &= \frac{1}{1 + e^{w \cdot x + b}} \end{aligned}$$

其中 $y \in \{0, 1\}$ 为输出。

假设一个事件的发生概率为 p ，定义事件发生的几率 (odds) 为发生概率与不发生概率的比值，则对数几率 (log odds) 或 logit 函数：

$$\text{logit} = \log \frac{p}{1-p}$$

那么对 LR 而言，有：

$$\text{logit}(x) = \log \frac{P(y=1|x)}{1 - P(y=1|x)} = w \cdot x + b$$

也就是说线性运算的结果对应表示该事件 $A(y=1|x)$ 的对数几率。

可以通过比较 $P(y=1|x)$ 和 $P(y=0|x)$ 的大小，将 x 进行分类。但 logistic 决策的本质是根据 $\text{logit}(x)$ 判断：若 $\text{logit}(x) > 0$ ，则 $x \in y_1$ ；若 $\text{logit}(x) < 0$ ，则 $x \in y_0$

可以将二分类 LR 模型推广到多分类 LR 模型，其实就是 Softmax 回归：

$$P(y=k|x) = \frac{\exp(w_k \cdot x + b_k)}{\sum_{j=1}^K \exp(w_j \cdot x + b_j)}$$

8.3.1 Sigmoid 函数与 Softmax 函数的区别

Sigmoid 函数形式为：

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

Softmax 函数形式为：

$$\text{Softmax}(x_j) = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}, j = 1, 2, \dots, K$$

Sigmoid 函数只考虑单变量，将其映射到 (0,1) 区间内，即使对向量做运算，向量每个元素间的结果无联系。Softmax 函数则考虑了向量中的每一个元素，虽然每个元素的值也是映射到 (0,1) 区间内，但各元素之和为 1。

对于二值分类问题，Sigmoid 函数和 Softmax 函数都输出两个值，假设输出 [0, 1] 代表“是”，输出 [1, 0] 代表“否”。

那么 Softmax 可能输出 [0.3, 0.7]，代表算法认为“是”的概率是 0.7，“否”的概率是 0.3，相加为 1。

Sigmoid 的输出可能是 [0.4, 0.8]，它们相加不为 1。解释来说就是 Sigmoid 认为输出第一位为 1 的概率是 0.4，第一位不为 1 的概率是 0.6 (1-p)，第二位为 1 的概率是 0.8，第二位不为 1 的概率是 0.2。

8.3.2 Sigmoid 函数多分类与 Softmax 函数多分类的区别

按照是否属于该类别的思路，可以设计多个使用 Sigmoid 函数的 LR 分类器。但是这样对每个分类器训练的时候，是/不是该类别的数据样本量会出现偏差。输出的结果向量中，每一个元素表示是/不是情况下，是该类别的概率，每个元素间不相关。

而使用 Softmax 多分类，则输出结果表示每一个类别的可能性。

8.3.3 二分类逻辑回归求解

二分类逻辑回归模型可以统一表示为：

$$P(y|x; \theta) = (\sigma_\theta(x))^y \cdot (1 - \sigma_\theta(x))^{(1-y)}$$

其中， $\sigma_\theta(x) = \sigma(\theta^T x)$ 。二分类逻辑回归就是在样本 X 上找到最大可能的概率分布 $P(Y|X; \theta)$ ，显然是一个求解极大似然的问题。似然函数：

$$L(\theta) = P(Y|X; \theta) = \prod_i P(y_i|x_i; \theta)$$

取对数并取反，即采用对数损失函数（交叉熵代价函数）

$$\begin{aligned} L &= -\log L(\theta) = -\log P(Y|X; \theta) = -\sum_i P(y_i|x_i; \theta) \\ &= -\sum_i [y_i \log(\sigma_\theta(x_i)) + (1 - y_i) \log(1 - \sigma_\theta(x_i))] \end{aligned}$$

由于 $\nabla_x \sigma(x) = \sigma(x)(1 - \sigma(x))$, 所以对 L 求梯度可得:

$$\nabla_{\theta} L = - \sum_i (y_i - \sigma_{\theta}(x_i)) x_i$$

★ 为什么逻辑回归要使用交叉熵代价函数?

1. 为什么可以使用交叉熵代价函数: 目的是为了让样本预测的概率可能性最大, 并且训练求解参数的速度是比较快的, 而且更新速度只和 x, y 有关, 比较的稳定。

2. 为什么不用平方损失函数: 梯度更新速度和 sigmod 函数的梯度相关, 训练速度会非常慢且有梯度消失的可能。而且平方损失会导致损失函数是 theta 的非凸函数, 不利于求解, 因为非凸函数存在很多局部最优解。

使用交叉熵代价 (对数损失) 函数的本质目的在于极大似然的原理。即使激活函数使用 ReLU, 仍然可以使用。

8.3.4 softmax 多分类求解

采用 softmax 的多分类逻辑回归模型可以统一表示为:

$$P(y|x; \theta) = \sum_{i=1}^k \mathbb{1}\{y_j = i\} \frac{e^{\theta_i^T x}}{\sum_k e^{\theta_k^T x}}$$

同样按照求解极大似然估计问题, 取对数损失函数为:

$$\begin{aligned} L &= -\log P(Y|X; \theta) = -\sum_j P(y_j|x_j; \theta) \\ &= -\sum_{j=1}^m \sum_{i=1}^k \mathbb{1}\{y_j = i\} \log \left[\frac{e^{\theta_i^T x_j}}{\sum_k e^{\theta_k^T x_j}} \right] \\ &= -\sum_{j=1}^m \sum_{i=1}^k \mathbb{1}\{y_j = i\} \left(\theta_i^T x_j - \log \sum_k e^{\theta_k^T x_j} \right) \end{aligned}$$

求梯度可得:

$$\begin{aligned} \frac{\partial L(\theta)}{\partial \theta_i} &= -\sum_{j=1}^m \mathbb{1}\{y_j = i\} \left(x_j - \frac{\partial \log \sum_k e^{\theta_k^T x_j}}{\partial \theta_i} \right) \\ &= -\sum_{j=1}^m \mathbb{1}\{y_j = i\} \left(x_j - x_j \frac{e^{\theta_i^T x_j}}{\sum_k e^{\theta_k^T x_j}} \right) \\ &= -\sum_{j=1}^m x_j [\mathbb{1}\{y_j = i\} - P(y_j = i|x_j; \theta)] \end{aligned}$$

九 强化学习

马尔可夫性质：当一个随机过程在给定现在状态及所有过去状态情况下，其未来状态的条件概率分布仅依赖于当前状态；换句话说，在给定现在状态时，它与过去状态（即该过程的历史路径）是条件独立的，那么此随机过程即具有马尔可夫性质。

9.1 强化学习里的对比思考

9.1.1 策略迭代与值迭代

先参考[这个回答](#)。策略迭代 (Policy iteration) 的伪代码如图21所示。策略迭代中，

```
Policy Iteration (using iterative policy evaluation) for estimating  $\pi \approx \pi_*$ 

1. Initialization
 $V(s) \in \mathbb{R}$  and  $\pi(s) \in \mathcal{A}(s)$  arbitrarily for all  $s \in \mathcal{S}$ 

2. Policy Evaluation
Loop:
 $\Delta \leftarrow 0$ 
Loop for each  $s \in \mathcal{S}$ :
 $v \leftarrow V(s)$ 
 $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$ 
 $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
until  $\Delta < \theta$  (a small positive number determining the accuracy of estimation)

3. Policy Improvement
 $policy-stable \leftarrow true$ 
For each  $s \in \mathcal{S}$ :
 $old-action \leftarrow \pi(s)$ 
 $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$ 
If  $old-action \neq \pi(s)$ , then  $policy-stable \leftarrow false$ 
If  $policy-stable$ , then stop and return  $V \approx v_*$  and  $\pi \approx \pi_*$ ; else go to 2
```

图 21 策略迭代算法

首先以一个固定的策略（具有概率分布），开始对值函数进行迭代，当值函数稳定后，进行策略提升（更新策略，比如按照 greedy 的形式）。然后重复上述步骤直到策略稳定。值迭代 (Value iteration) 的伪代码如图22所示。值迭代可以认为策略只选择 greedy 策略，每做一次值函数迭代，都进行了策略提升（表现在值函数更新时使用的 max 函数）。

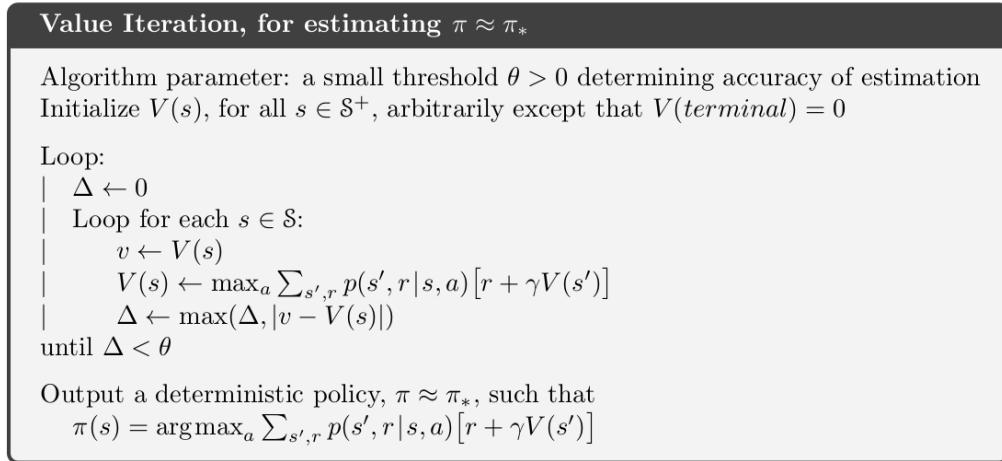


图 22 值迭代算法

9.2 深度强化学习准备知识

于 Model-Free RL，可以分为 Policy Optimization 和 Q-Learning 两类方法。Policy Optimization 优化方法包括：PolicyGradient、A2C/A3C、TRPO、PPO 等。Q-Learning 方法包括：DQN、C51、QR-DQN、HER 等。此外还有将两种方法融合使用的，包括 DDPG、TD3、SAC 等。

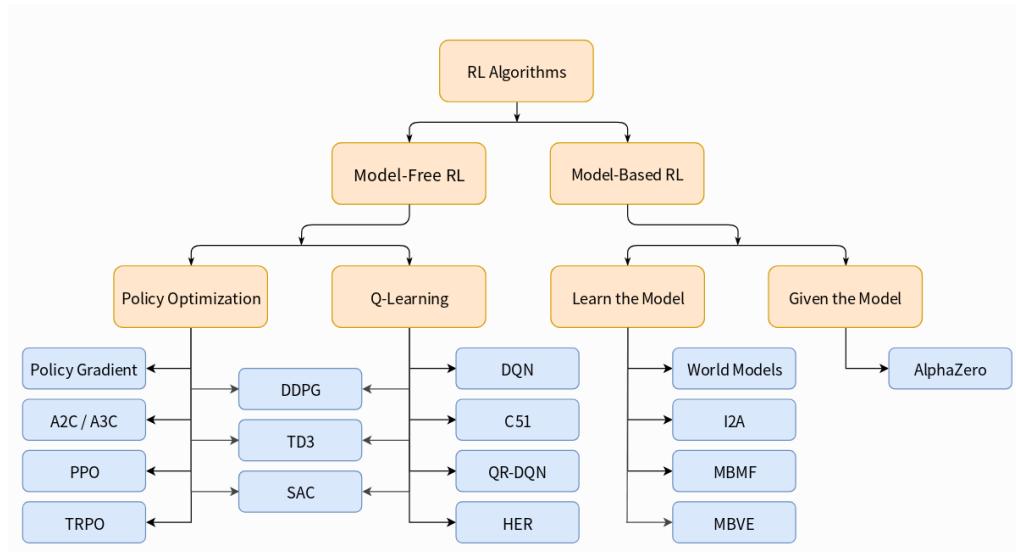


图 23 RL 算法的大致分类

Policy Optimization 基本思想是直接对策略 $\pi_\theta(a|s)$ 进行估计，并利用目标函数 $J(\pi_\theta)$ 最优化参数 θ 。所以这种优化方法绝大部分都是 on-policy 的方式。此外在策略优化方法的实现过程中，通常也会采用 Actor-Critic 的框架。只不过一般在 Policy Optimization 中的 Critic 部分，只进行 $V_\phi(s)$ 的估计。

Q-Learning 基本思想是用 $Q_\theta(s,a)$ 对最优动作值函数进行估计。通常表现为 off-policy 的方式。最终的策略可以表示为：

$$a(s) = \arg \max_a Q_\theta(s,a)$$

而将两种方法结合起来的算法，就是同时估计 $Q_\theta(s, a)$ 和 $\pi_\theta(a|s)$

9.2.1 Q-Learning 和 Policy Gradient 的对比

参考[这个问题](#)

9.2.2 策略分类

on-policy, off-policy

Deterministic Policies, Stochastic Policies

Categorical Policies, Diagonal Gaussian Policies

9.3 DQN

Q-learning 在每步迭代过程中，使用如下的损失函数：

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

DQN 算法的几个核心特点包括：

- (1) 使用 CNN 完成 end-to-end 的特征提取及对 Q 函数的估计
- (2) 采用 experience replay 技巧，削弱 episode 样本间的相关性，提高算法的收敛的稳定性
- (3) 降低 target 网络的更新频次，增加稳定性

具体算法伪代码为：

特点总结：

- (1) off-policy, 使用 greedy- ϵ 的策略做探索，更新 greedy 策略
- (2) 最终保留的是 target 值函数，即伪代码里的 $\hat{Q}(s, a; \theta)$
- (3) 离散动作空间，所以 a_{ph} 为 1 维。
- (4) 对于 $\max_a Q(s, a; \theta)$ 的实现，需要对根据 s 生成的所有 $Q(s, \cdot; \theta)$ 根据 a_{ph} 变量进行切片。使用了 `tf.gather_nd()` 函数
- (5) 也可以使用 `tf.one_hot()` 进行切片：`q_sa = tf.reduce_sum(tf.one_hot(a, act_dim) * qvalue, axis=1)`
- (6) 两个神经网络间的参数复制，使用 `tf.assign()` 函数。

具体代码为

```

0 def get_vars(scope):
1     return [x for x in tf.global_variables() if scope in x.name]
2
3 self.target_copy = tf.group([tf.assign(v_target, v_value) for v_value
, v_target in zip(get_vars('value'), get_vars('target'))])

```

Algorithm 1: deep Q-learning with experience replay.

```

Initialize replay memory  $D$  to capacity  $N$ 
Initialize action-value function  $Q$  with random weights  $\theta$ 
Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$ 
For episode = 1,  $M$  do
    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$ 
    For  $t = 1, T$  do
        With probability  $\varepsilon$  select a random action  $a_t$ 
        otherwise select  $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$ 
        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$ 
        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ 
        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $D$ 
        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$ 
        Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$ 
        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  with respect to the
        network parameters  $\theta$ 
        Every  $C$  steps reset  $\hat{Q} = Q$ 
    End For
End For

```

图 24 DQN 算法伪代码

9.4 策略优化的基础知识

考虑随机性参数化策略 π_θ , 策略的目的是将回报的期望最大化。因此目标函数为

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

显然可以通过梯度下降的方法对策略进行优化:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}$$

其中 $\nabla_\theta J(\pi_\theta)$ 称为策略梯度。为了使用这个算法, 需要显式给出策略梯度的表达式, 包括两个步骤: 1) 推导出策略性能的分析梯度, 结果应该表示成期望的形式。2) 形成该期望值的样本估计, 这个估计值可以从有限数量的样本中计算。

对于在策略 π_θ 下采样获得的一条轨迹 $\tau = (s_0, a_0, \dots, s_{T+1})$, 其概率为:

$$P(\tau|\theta) = \rho_0(s_0) \prod_{t=0}^T P(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$$

考虑对数微分技巧:

$$\nabla_\theta P(\tau|\theta) = P(\tau|\theta) \nabla_\theta \log P(\tau|\theta)$$

这样轨迹的对数概率为：

$$\log P(\tau|\theta) = \log \rho_0(s_0) + \sum_{t=0}^T (\log P(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t))$$

对 θ 求微分为：

$$\nabla_\theta \log P(\tau|\theta) = \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t|s_t)$$

计算策略梯度为：

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] \\ &= \nabla_\theta \int_\tau P(\tau|\theta) R(\tau) \\ &= \int_\tau \nabla_\theta P(\tau|\theta) R(\tau) \\ &= \int_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) R(\tau) \\ &= \mathbb{E}_{\tau \sim \pi_\theta} [\nabla_\theta \log P(\tau|\theta) R(\tau)]\end{aligned}$$

代入之前的计算结果可得：

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \right] \quad (9-1)$$

式 (9-1) 是一个期望表达式，那么可以通过采样平均的方式进行估计。假设我们获得了一个轨迹数据集： $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$ ，那么策略梯度就可以估计为：

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)$$

当使用神经网络优化的方式的时候，通常定义 $L(\theta) = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^T \log \pi_\theta(a_t|s_t) R(\tau)$ 作为损失函数，这样损失函数的梯度就是策略梯度。

几个注意的地方 虽然上述定义了一个“损失函数”，但并不意味着它与监督学习里的损失函数具有相同的意义。具体来说，有以下两点不同：

1. **数据分布依赖于参数。** 通常损失函数定义在一个固定的数据分布上，与我们待优化的参数是独立的。但是在这里，数据需要根据最新的策略进行采样。
2. **这里的“损失函数”无法衡量性能。** 损失函数通常是对我们所关心的性能指标的一种估计。在这里，我们关心期望回报 $J(\pi_\theta)$ ，但是定义的“损失函数”并不能对其进行估计。这里“损失函数”对我们的作用只是因为在基于当前待优化参数采样的数据集下，损失函数的梯度与性能梯度相同。而在完成一次梯度下降之后，二者就没有什么联系了。这意味着，在给定的 batch 上，最小化损失函数并不能保证一定能够提高期望回报。

9.4.1 回报函数

式 (9-1) 中在每一个采样状态下都以整条轨迹的回报 $R(\tau)$ 作为当前状态的回报，这显然是不合适的。通常是否增强当前状态的某个行为，应该考虑执行该行为后的回报，而不是执行之前的历史回报。所以可以定义如下的”reward-to-go”

$$\hat{R}_t \doteq \sum_{t'=t}^T \gamma^{t'-t} R(s_{t'}, a_{t'}, s_{t'+1})$$

其中 γ 为回报的折扣率。

考虑到可以证明：

$$\mathbb{E}_{a_t \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a_t | s_t) b(s_t)] = 0$$

所以可以在计算策略梯度的时候给回报部分增加一个只与状态相关的值函数 $b(s_t)$ ，称为 baseline。最常规的 baseline 选择就是当前策略的值函数 $V^\pi(s_t)$ 。那么这个时候相当于用优势函数作为回报。通常 $V^\pi(s - t)$ 无法直接得到，所以一般也是采用神经网络 $V_\phi(s_t)$ 对最优动作值函数进行估计。

所以一个更为一般形式的策略梯度为：

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) \Phi_t \right] \quad (9-2)$$

其中 Φ_t 可以取为不同形式的函数。

9.4.2 Generalized Advantage Estimation, GAE

假设 V 是 Value 值函数的一个近似，那么在折扣率 γ 下的 TD 残差可以表示为 $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$ 显然 δ_t^V 可以看成是优势函数在动作 a_t 下的一个估计。如果值函数是真实的 $V = V^{\pi, \gamma}$

$$\mathbb{E}_{s_{t+1}} [\delta_t^{V^{\pi, \gamma}}] = \mathbb{E}_{s_{t+1}} [r_t + \gamma V^{\pi, \gamma}(s_{t+1}) - V^{\pi, \gamma}(s_t)] = A^{\pi, \gamma}(s_t, a_t)$$

通过推导可以得到：

$$\hat{A}_t^{GAE(\gamma, \lambda)} \doteq \sum_{l=0}^{\infty} (\gamma \lambda)^l \delta_{t+l}^V$$

表示了一种广义的优势函数的估计。 λ 为调节参数：

$$\begin{aligned} \text{GAE}(\gamma, \lambda=0) : \hat{A}_t &= \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \\ \text{GAE}(\gamma, \lambda=1) : \hat{A}_t &= \sum_{l=0}^{\infty} \gamma^l \delta_{t+l} = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V(s_t) \end{aligned}$$

$\text{GAE}(\gamma, 0)$ 估计是无偏差的，但会引入较大的方差。 $\text{GAE}(\gamma, 1)$ 估计是有偏差的，但是方差较小。

计算 GAE 的代码：

```
0 def discount_cumsum(x, discount):
```

```

1  """
2 magic from rllab for computing discounted cumulative sums of vectors.
3
4 input:
5     vector x,
6     [x0,
7         x1,
8         x2]
9
10 output:
11     [x0 + discount * x1 + discount^2 * x2,
12         x1 + discount * x2,
13         x2]
14 """
15 return scipy.signal.lfilter([1], [1, float(-discount)], x[::-1], axis
16 =0) [::-1]
17
18 deltas = rews[:-1] + self.gamma * vals[1:] - vals[:-1]
self.adv_buf[path_slice] = discount_cumsum(deltas, self.gamma * self.
    lam)

```

9.5 DDPG 算法

9.5.1 另一种目标函数的形式

在9.4节中，定义了目标函数的形式为 $J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$ 。并推导了在该形式下的策略梯度。从这个目标函数的定义形式可以看出，其出发点在于考虑每一条轨迹回报，目的是使得在轨迹空间期望回报最大。

在 DPG 算法之前，首先换一个角度。考虑状态-动作空间中的每一个 (s, a) 的价值函数 $Q(s, a)$ ，如果目的是使得状态-动作空间中的价值函数期望最大，那么就可以定义如下形式的目标函数：

$$\begin{aligned} J(\pi_\theta) &= \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta}[Q(s, a)] \\ &= \int_S \rho^\pi(s) \int_A \pi_\theta(s, a) Q(s, a) da ds \end{aligned} \quad (9-3)$$

其中， $\rho^\pi(s)$ 表示在状态 s 在一条轨迹中被访问的概率。具体定义为：

$$\rho^\pi(s') \doteq \int_S \sum_{t=1}^{\infty} \gamma^{t-1} p_0(s) p(s \rightarrow s', t, \pi) ds$$

上式右端第三项表示：在策略 π 作用下，从状态 s 经过 t 步转移后到达 s' 的概率。

对式 (9-4) 求梯度有：

$$\begin{aligned}
 \nabla_{\theta} J(\pi_{\theta}) &= \nabla_{\theta} \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \pi_{\theta}(s, a) Q(s, a) da ds \\
 &= \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \nabla_{\theta} \pi_{\theta}(s, a) Q(s, a) da ds \\
 &= \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \pi_{\theta}(s, a) \nabla_{\theta} \log \pi_{\theta}(s, a) Q(s, a) da ds \\
 &= \mathbb{E}_{s \sim \rho^{\pi}, a \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) Q(s, a)]
 \end{aligned}$$

显然与式 (9-1) 是等价。

9.5.2 确定性策略梯度 DPG

进一步考虑确定性策略，定义为：

$$a = \mu_{\theta}(s)$$

即在任意状态 s 下都能得到惟一性的动作 a 。代入上一节的目标函数中有：

$$\begin{aligned}
 J(\theta) &= \mathbb{E}_{s \sim \rho^{\mu}} [Q(s, \mu_{\theta}(s))] \\
 &= \int_{\mathcal{S}} \rho^{\mu}(s) Q(s, \mu_{\theta}(s)) ds
 \end{aligned} \tag{9-4}$$

则策略梯度为：

$$\begin{aligned}
 \nabla_{\theta} J(\theta) &= \int_{\mathcal{S}} \rho^{\mu}(s) \nabla_{\theta} \mu_{\theta}(s) \nabla_a Q^{\mu}(s, a) |_{a=\mu_{\theta}(s)} ds \\
 &= \mathbb{E}_{s \sim \rho^{\mu}} [\nabla_{\theta} \mu_{\theta}(s) \nabla_a Q^{\mu}(s, a) |_{a=\mu_{\theta}(s)}]
 \end{aligned}$$

十 一些有趣的概率问题

10.1 三门问题

10.2 吸收马尔可夫过程

先参考[这篇文献](#)的 11.2 节。