PX4二次开发

PX4外环PID算法

一、外环PID算法

在上一次作业中，我们通过向PX4的位置控制器发送一条轨迹来实现飞行控制，现在我们知道：我们最终实际上是向`trajectory_setpoint`话题发布了一条轨迹，这条轨迹最终被`MultiCopterPositionControl`模块接收并执行。此次将通过书写代码来替换PX4位置控制器的关键部分：位置环PID算法。

我们期望通过控制飞机执行加速度指令来实现轨迹跟踪，以加速度为控制量，我们可以得到如下：

$$A_{control} = A_r + K_p(P_d - P_c) + K_v(V_d - V_c)$$

其中，A_r 为期望加速度，P_d 为期望位置，P_c 为当前位置。无人机的真实状态可以通过状态估计器EKF2获取，因此通过上述方法我们便可以通过控制飞机执行加速度指令实现轨迹跟踪。

二、代码实现

1、轨迹发布

在上次作业中，我们实现了轨迹发布的代码：

```
void OffboardCtrl::PublishTrajectorySetpoint()
{
    if(!_flag)
    {
        trajectory_setpoint_s trajectory_setpoint;
        memset(&trajectory_setpoint, 0, sizeof(trajectory_setpoint_s));
        trajectory_setpoint.timestamp = hrt_absolute_time();

        _current_time = hrt_absolute_time();
        const float dt = (_current_time - _switched_time) * 1e-6;


        trajectory_setpoint.position[0] =  _switched_pos.x + 1.0f * sinf(dt*0.5f);
        trajectory_setpoint.position[1] =  _switched_pos.y + 1.0f - 1.0f * cosf(dt*0.5f);
        trajectory_setpoint.position[2] =  _switched_pos.z;

        _trajectory_setpoint_pub.publish(trajectory_setpoint);
        PublishOffboardMode();
    }
}
```

为了实现此次作业要求，我们需要将该轨迹命令发布给我们的自定义位置控制器，经计算出加速度命令，从而传给我们的PX4位置控制器。结合前面的学习，我们知道uORB通信具有多话题发布订阅机制，因此我们在`msg/TrajectorySetpoint.msg`文件中添加如下代码：`# TOPICS trajectory_setpoint trif_trajectory_setpoint`，那么通过这样的方式我们便可以使用自定义的控制器来订阅轨迹命令。

2、计算加速度指令

核心代码如下：

```
void TrifPosCtrl::ComputeAccCommand()
{
        uint64_t now = hrt_absolute_time();
        float deltat = (_last_time > 0) ? ((now - _last_time) * 1e-6f) : 0.02f;
        _last_time = now;
        if (deltat > 0.1f) deltat = 0.02f;

        _des_pos = Vector3f(_trif_trajectory.position);
        _des_vel = Vector3f(_trif_trajectory.velocity);
        _des_acc = Vector3f(_trif_trajectory.acceleration);
        _pos_error = _des_pos - Vector3f(_vehicle_local_position.x, _vehicle_local_position.y, _vehicle_local_position.z);
        _vel_error = _des_vel - Vector3f(_vehicle_local_position.vx, _vehicle_local_position.vy, _vehicle_local_position.vz);

        setZeroIfNanVec3(_pos_error);
        setZeroIfNanVec3(_vel_error);
        setZeroIfNanVec3(_des_acc);

        _acc_command = _des_acc + _pos_error.emult(_pos_kp) + _vel_error.emult(_vel_kp) + _vel_err_int.emult(_vel_ki);

        setZeroIfNanVec3(_acc_command);
        _vel_err_int = _vel_err_int + _vel_error * deltat ;
        setZeroIfNanVec3(_vel_err_int);

        _trif_status.timestamp = hrt_absolute_time();
        _pos_error.copyTo(_trif_status.pos_err);
        _vel_error.copyTo(_trif_status.vel_err);
        _vel_err_int.copyTo(_trif_status.vel_err_int);
        _trif_status.dt = deltat;
        _trif_status_pub.publish(_trif_status);
}
```

3、PX4参数配置机制

在PX4中，我们可以自定义参数，并在QGC中对其进行配置， 这样对于调参或者修改配置参数十分方便。下面我以自定义PID控制参数为例：

（1）书写YAML声明参数：告诉PX4需要定义哪些参数以及基本的类型信息

我们在`trif_posctrl`模块路径下创建一个叫做`module.yaml`的文件，并在文件中输入下面的参数定义声明:

```yaml
moudle_name: trif_posctrl
parameters:
    - group: TRIF_POSCTRL Settings
      definitions:
        TRIF_POS_KP_XY:
            description:
                short: The KP parameter for the position control in the horizontal direction
            type: float
            default: 5.0
            min: 0.0

        TRIF_POS_KP_Z:
            description:
                short: The KP parameter for the position control in the vertical direction
            type: float
            default: 7.0
            min: 0.0

        TRIF_VEL_KP_XY:
            description:
                short: The KP parameter for the velocity control in the horizontal direction
            type: float
            default: 3.0
            min: 0.0

        TRIF_VEL_KP_Z:
            description:
                short: The KP parameter for the velocity control in the vertical direction
            type: float
            default: 5.0
            min: 0.0
```

(2) 书写C++监听参数变化

这一步有很多工作需要做。首先我们需要在主类中继承 ModuleParams 基类：

```cpp
class TrifPosCtrl : public ModuleBase<TrifPosCtrl>, public ModuleParams , public px4::ScheduledWorkItem
{
public:
        TrifPosCtrl();
        ~TrifPosCtrl() override;
```

然后在类里面利用 `DEFINE_PARAMETERS` 宏声明此类可能需要访问到的参数：

```cpp
 DEFINE_PARAMETERS(
        (ParamFloat<px4::params::TRIF_POS_KP_XY>) trif_pos_kp_xy,
        (ParamFloat<px4::params::TRIF_POS_KP_Z>) trif_pos_kp_z,
        (ParamFloat<px4::params::TRIF_VEL_KP_XY>) trif_vel_kp_xy,
        (ParamFloat<px4::params::TRIF_VEL_KP_Z>) trif_vel_kp_z,
        (ParamFloat<px4::params::TRIF_VEL_KI_XY>) trif_vel_ki_xy,
        (ParamFloat<px4::params::TRIF_VEL_KI_Z>) trif_vel_ki_z
 )
```

接着我们需要监听`parameter_update`话题检测参数更新,并在回调中处理参数更新：

```
if (_parameter_update_sub.updated()) {
        parameter_update_s param_update;
        _parameter_update_sub.copy(&param_update);
        updateParams(); // update module parameters (in DEFINE_PARAMETERS)
        getParameters();
}
```

最后通过`get()`接口获取参数值：

```
void TrifPosCtrl::getParameters()
{
        _pos_kp(0) = _pos_kp(1) = trif_pos_kp_xy.get();
        _pos_kp(2) = trif_pos_kp_z.get();

        _vel_kp(0) = _vel_kp(1) = trif_vel_kp_xy.get();
        _vel_kp(2) = trif_vel_kp_z.get();

        _vel_ki(0) = _vel_ki(1) = trif_vel_ki_xy.get();
        _vel_ki(2) = trif_vel_ki_z.get();

        printf("PosCtrl parameter updated:\n"
               "\tPOS_KP: [%f, %f, %f]\n"
               "\tVEL_KP: [%f, %f, %f]\n"
               "\tVEL_KI: [%f, %f, %f]\n" ,
               (double) _pos_kp(0), (double) _pos_kp(1), (double) _pos_kp(2),
               (double) _vel_kp(0), (double) _vel_kp(1), (double) _vel_kp(2),
               (double) _vel_ki(0), (double) _vel_ki(1), (double) _vel_ki(2)
            );

}
```

(3) 书写CMake声明此模块需要用到的数据信息

最后我们只需要在`px4_add_module`中添加一个`MODULE_CONFIG`即可，并将对应的数据定义文件传给此参数。

```
px4_add_module(
        MODULE examples__trif_posctrl
        MAIN trif_posctrl
        STACK_MAIN 2000
        MODULE_CONFIG
                module.yaml
        SRCS
                trif_posctrl.cpp
        DEPENDS
)
```

(4) QGC中进行参数设置

在使用QGC进行参数设置之前，一定要保证使用对应参数的模块已经被启动，否则在QGC里参数不可见。

先启动模块，再启动QGC，然后进入参数设置模块：

```
pxh> offboard_ctrl start
pxh> trif_posctrl start
pxh> PosCtrl parameter updated:
        POS_KP: [2.000000, 2.000000, 3.000000]
        VEL_KP: [2.000000, 2.000000, 2.000000]
        VEL_KI: [1.500000, 1.500000, 1.500000]
```

我们可以在QGC里看见我们自定义的参数了。然后在QGC中对参数进行设置看一下有没有效果。

4、自定义飞行日志

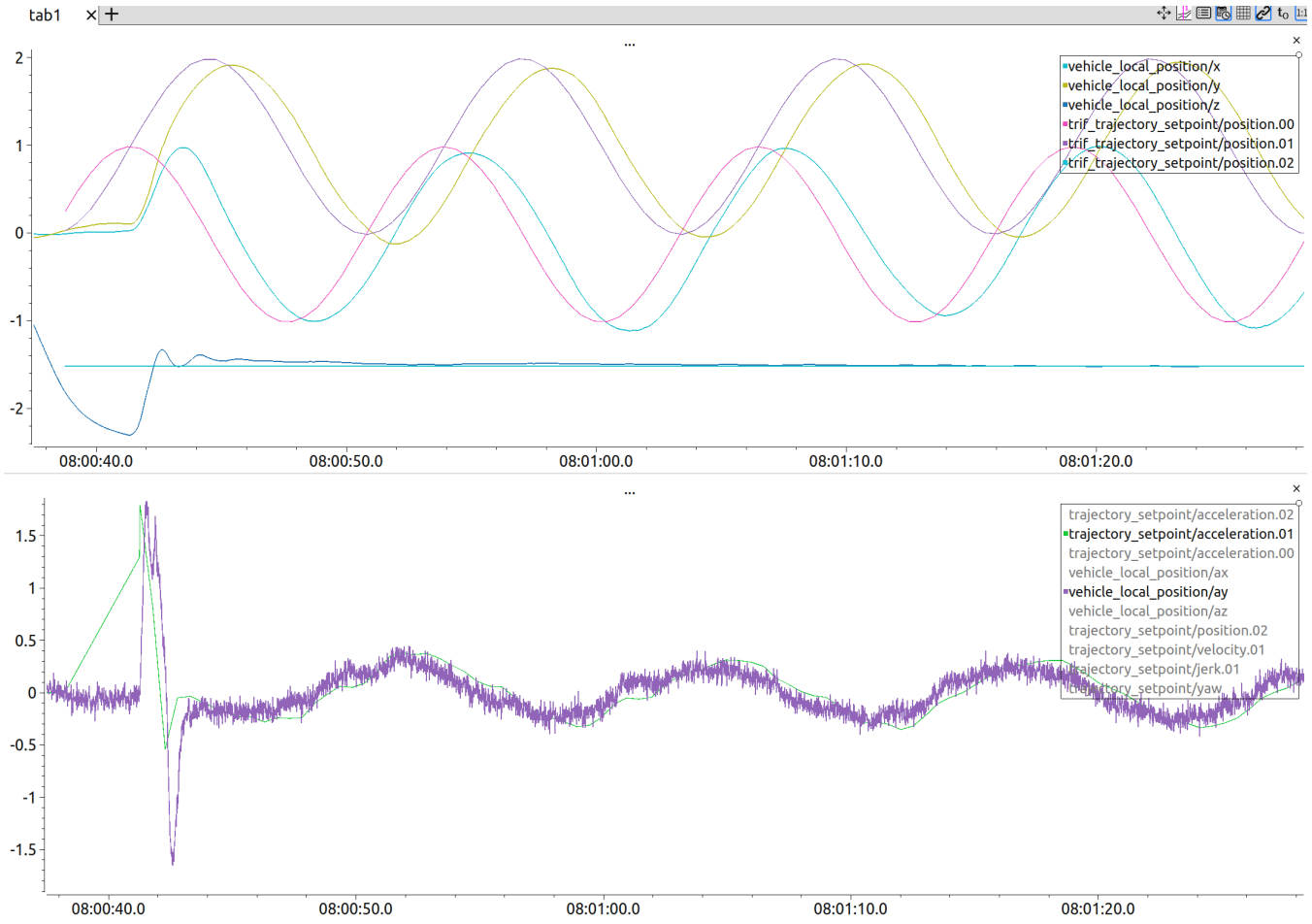
我们在进行PX4二次开发的过程中，不可避免地需要进行uorb消息的自定义和发布，当我们想要对这些数据进行分析的时候，飞行日志就是一个非常重要的分析工具。当我们需要将自定义的消息写入飞行日志的时候应该怎么做呢？事实上，飞行日志中写入的内容全部都是uORB消息，我们只需要声明将我们关心的话题消息一并写入飞行日志即可。找到 `src/modules/logger/log_topics.cpp` 有一个 `add_default_topics` 函数，我们只需要在这里添加我们感兴趣的话题即可将其写入飞行日志：

```
    add_topic("vehicle_roi", 1000);
    add_topic("vehicle_status");
    add_optional_topic("vtol_vehicle_status", 200);
    add_topic("wind", 1000);

    //Added by Chan
    add_topic("trif_trajectory_setpoint", 200);
    add_topic("trif_status", 200);
    add_topic("trif_thrust_estimate", 100);
    add_topic("task4_trajectory_setpoint", 200);

    // multi topics
    add_optional_topic_multi("actuator_outputs", 100, 3);
    add_optional_topic_multi("airspeed_wind", 1000, 4);
    add_optional_topic_multi("control_allocator_status", 200, 2);
    add_optional_topic_multi("rate_ctrl_status", 200, 2);
    add_optional_topic_multi("sensor_hygrometer", 500, 4);
    add_optional_topic_multi("rpm", 200);
```

最后我们可在 `plotjuggler` 中看到各个话题的数据，并能清晰的进行期望值和实际值的对比分析。



悬停油门量估计-衰减记忆递推最小二乘法

一、物理模型构建

在PX4中，油门量与推力之间建模为一个线性关系，然后用最小二乘法估计线性模型的参数。

假设推力F与油门量T的关系为：

$$F = KT$$

同时定义悬停时油门为 T_h ,则在悬停时我们可以得到

$$F = KT_h = mg$$

$$K = mg/T_h$$

在任意时刻，假设油门量为T，推力为F，加速度为a，我们可以得到方程

$$F - mg = am$$

$$KT - mg = mg * T/T_h - mg = am$$

最终可以解算得到

$$T = T_h(a/g + 1)$$

我们可以发现在应用场景中，随着时间的推移，我们能逐渐积累许多油门量T和加速度a的数据，因此通过最小二乘法我们可以估计出悬停时油门T_h。

二、最小二乘法

设有一系列数据 $X = [x_1, x_2, \cdots x_n] \in \mathbb{R}^{d\times n}$ $Y = [y_1, y_2, \cdots y_n]^T \in \mathbb{R}^n$,我们期望 x_i, y_i 呈线性关系：

$$y_i = w^T \cdot x_i$$

为了估计最优的w ，我们可以解下面的最小二乘问题：

$$\min_{\omega} \|X^T\omega - Y\|_2^2$$

我们令$L(w) = \|X^T\omega - Y\|_2^2$ ，根据向量范数的定义$\|v\|^2 = v^Tv$ ，可展开为

$$\begin{aligned} L(\omega) &= (X^T\omega - Y)^T(X^T\omega - Y) \\ &= (\omega^TX - Y^T)(X^T\omega - Y) \\ &= \omega^TXX^T\omega - \omega^TXY - Y^TX^T\omega + Y^TY \end{aligned}$$

其中，w^TXY 是一个标量，标量的转置等于本身。所以 $（w^TXY）^T = Y^TX^Tw$ ，最后合并同类项可得：

$$L(\omega) = \omega^T(XX^T)\omega - 2\omega^T(XY) + Y^TY$$

接着需要对w 求梯度，并令其为0。这里需要用到两个矩阵求导公式：

$$\frac{\partial(\omega^TA\omega)}{\partial\omega} = 2A\omega（当A是对称矩阵时，这里XX^T必然对称）$$

$$\frac{\partial(\omega^Tb)}{\partial\omega} = b$$

代入公式求导：

$$\frac{\partial L}{\partial\omega} = 2(XX^T)\omega - 2(XY)$$

令梯度为0，可解得：

$$\omega = (XX^T)^{-1}XY$$

三、递推最小二乘法

我们在上面提到过通过最小二乘法估计悬停时油门T_h ，然而随着时间的累积，数据量越来越多，势必影响计算效率。为了解决这个问题，递推最小二乘法应运而生。

递推最小二乘法的基本原理是：建立具有n-1组数据的最小二乘解和具有n组数据的最小二乘解之间的递推关系，从而在第n组数据到来时，可以直接根据第n-1组数据的解和新来的数据之间的简单递推，得到当前最小二乘问题的解。这个过程可以形式化表达如下：

$$\omega_{n+1} = \omega_n + \epsilon_{n+1}$$

根据上面最小二乘法我们已经得到了w 的解:

$$\omega_k = (X_kX_k^T)^{-1}X_kY_k$$

为了后面计算的方便，我们令$X_kX_k^T=P_k^{-1}$，$X_kY_k=U_k$,得到：

$$w_k=P_kU_k$$

关于P_k、U_k 我们不难得到下面的递推关系:

$$P_{k+1}^{-1}=P_k^{-1}+x_{k+1}x_{k+1}^T$$
$$U_{k+1}=U_k+x_{k+1}y_{k+1}$$
$$w_{k+1}=P_{k+1}U_{k+1}$$

接下来开始递推：由公式（3)、(4）可得

$$P_{k+1}=(P_k^{-1}+x_{k+1}x_{k+1}^T)^{-1}$$
$$=[I-(P_k^{-1}+x_{k+1}Ix_{k+1}^T)^{-1}x_{k+1}Ix_{k+1}^T]P_k$$
$$令K=(P_k^{-1}+x_{k+1}Ix_{k+1}^T)^{-1}x_{k+1}I$$
$$=P_kx_{k+1}(I+x_{k+1}^TP_kx_{k+1})^{-1}=P_{k+1}x_{k+1}$$
$$则P_{k+1}=(I-Kx_{k+1}^T)P_k$$

最终得到：

$$w_{k+1}=P_{k+1}U_{k+1}=P_{k+1}(U_k+x_{k+1}y_{k+1})$$
$$=P_{k+1}U_k+Ky_{k+1}=(I-Kx_{k+1}^T)P_kU_k+Ky_{k+1}$$
$$=w_k+K(y_{k+1}-x_{k+1}^Tw_k)$$

补充公式：

$$(I+A)^{-1}(I+A)=I=(I+A)-A$$
$$(I+A)^{-1}=I-(I+A)^{-1}A \tag{1}$$

$$(A+B)^{-1}=[A(I+A^{-1}B)]^{-1}=(I+A^{-1}B)^{-1}A^{-1} \tag{2}$$

$$(A+B)^{-1}=(I+A^{-1}B)^{-1}A^{-1}=[I-(I+A^{-1}B)^{-1}A^{-1}B]A^{-1}$$
$$=(I-(A+B)^{-1}B)A^{-1} \tag{3}$$

$$BA^{-1}CDB+B=BA^{-1}(CDB+A)=(BA^{-1}C+D^{-1})DB$$
$$(BA^{-1}C+D^{-1})^{-1}BA^{-1}=DB(CDB+A)^{-1} \tag{4}$$

四、衰减记忆递推最小二乘法

递推最小二乘法成功解决了当数据连续到来时问题求解的效率问题，但随着时间的推移，之间的关系可能发生缓慢的变化，而我们在递推最小二乘法中，将所有这些数据都同等对待，这显然是不合适的，为此，我们需要解决下面一个最小二乘问题：

$$\min_{\omega}\sum_{k=1}^{n}\|\gamma^{n-k}(\omega^Tx_k-y_k)\|_2^2$$

定义：$\Lambda_n=diag([\gamma^n,\gamma^{n-1},\cdots 1])$，则上述问题可以被重新描述为：

$$\min_{\omega}\|\Lambda_n(X^T\omega-Y)\|_2^2$$

上述问题的最优解是：

$$\omega = (X\Lambda_n^2 X^T)^{-1} X\Lambda_n^2 Y$$

令$P_n = (X\Lambda_n^2 X^T)^{-1}$,$U_n = X\Lambda_n^2 Y$

可得到以下递推关系：

$$P_{n+1}^{-1} = \gamma^2 P_n^{-1} + x_{n+1}x_{n+1}^T$$
$$U_{n+1} = \gamma^2 U_n + x_{n+1}y_{n+1}$$

关于P矩阵，我们有：

$$\begin{aligned} P_{n+1} &= (\gamma^2 P_n^{-1} + x_{n+1}x_{n+1}^T)^{-1} \\ &= \frac{1}{\gamma^2}(I + \frac{1}{\gamma^2}P_n x_{n+1}x_{n+1}^T)^{-1}P_n \\ &= \frac{1}{\gamma^2}[I - \frac{1}{\gamma^2}(I + \frac{1}{\gamma^2}P_n x_{n+1}x_{n+1}^T)^{-1}P_n x_{n+1}x_{n+1}^T)]P_n \\ &= \frac{1}{\gamma^2}(I - Kx_{n+1}^T)P_n \end{aligned}$$

其中：

$$\begin{aligned} \mathrm{K} &= \frac{1}{\gamma^2}(I + \frac{1}{\gamma^2}P_n x_{n+1}x_{n+1}^T)^{-1}P_n x_{n+1} \\ &= (\gamma^2 P_n^{-1} + x_{n+1}x_{n+1}^T)^{-1}x_{n+1} \\ &= \frac{1}{\gamma^2}P_n x_{n+1}(\frac{1}{\gamma^2}x_{n+1}^T P_n x_{n+1} + I)^{-1} \\ &= P_n x_{n+1}(x_{n+1}^T P_n x_{n+1} + \gamma^2)^{-1} \\ &= P_{n+1}x_{n+1} \end{aligned}$$

最终可以得到：

$$\begin{aligned} \omega_{n+1} &= P_{n+1}U_{n+1} \\ &= P_{n+1}(\gamma^2 U_n + x_{n+1}y_{n+1}) \\ &= \gamma^2 P_{n+1}U_n + Ky_{n+1} \\ &= (I - Kx_{n+1}^T)P_n U_n + Ky_{n+1} \\ &= \omega_n + K(y_{n+1} - \omega_n x_{n+1}^T) \end{aligned}$$

五、代码实现

下面贴出核心代码：

```cpp
void TrifPosCtrl::HoverThrottleEstimate()
{
        float thrust_z = _vehicle_thrust_setpoint.xyz[2];
        float y = -thrust_z;
        if (y < 0.1f) return;
        float g = 9.81f;
        //float a_z = _sensor_accel.z;
        float a_z = _vehicle_local_position.az;
        float x = (g - a_z) / g;

        float gamma_sq = _gamma * _gamma;
        float P_x = _P * x;
        float K = P_x / (x * P_x + gamma_sq);  //K计算
        _P = (1.0f - K * x) * _P / gamma_sq;   //P更新

        float y_e = y - _hover_thr * x;
        _hover_thr = _hover_thr + K * y_e;  //悬停油门更新

        if (_hover_thr > 0.9f) _hover_thr = 0.9f;
        else if (_hover_thr < 0.1f) _hover_thr = 0.1f;

        trif_thrust_estimate_s trif_thrust_estimate;
        trif_thrust_estimate.timestamp = hrt_absolute_time();
        trif_thrust_estimate.hover_thrust = _hover_thr;
        _trif_thrust_estimate_pub.publish(trif_thrust_estimate);

}
```

其中我们的实时推力可以从话题 `vehicle_thrust_setpoint` 中读取，由于PX4的坐标系为NED，因此为了方便后面统一计算，我们统一将所有数据设为标量。同时，实时加速度从话题 `vehicle_local_position` 中读取。

最终根据上述公式，代入计算，得到如下图的悬停油门量估计，我们可以发现我们估计出来的 `trif_thrust_estimate/hover_thrust` 与 PX4原生悬停油门量 `hover_thrust_estimate/hover_thrust` 相差0.005左右。