《空中机器人》实验报告

第五组：毛永奇、窦泽鑫、娄开杨、薛宇航、李子曦、孙上为

1 设备搭建与环境建立

1.1 组装步骤

1. **硬件焊接**：包括焊接 XT60 电源接头、电调和电机，注意焊接时正负极连接正确并避免虚焊。
2. **配件安装与连接**：包括固定机架、布线、安装电调、粘贴飞控、连接信号线、飞机与遥控器配对、安装起落架等。
3. **安全检查**：确认主电源无短路、电机转向正确，同时进行理线，确保桨叶不会打到电线。

1.2 参数设置

1. 使用 **QGroundControl** 软件连接飞控，更新固件。
2. 校准传感器及遥控器，配置飞行模式及电源设置。实际操作中需要将无人机紧贴地面或墙壁以保证水平和竖直。
3. 调整飞控参数（如安全模式和通信协议）及电机转向，对需要反转的电机输入反转指令。

2 控制程序设计

2.1 线性控制器

线性控制器基于平衡悬停态下对于非线性方程的线性近似。经过牛顿方程和欧拉方程线性化近似可得如下结论：

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} g(\theta cos\psi + \phi sin\psi) \\ g(\theta sin\psi - \phi cos\psi) \\ -g + \frac{U_1}{m} \end{bmatrix}$$

$$I \times \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} + \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times I \cdot \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} l(F_2 - F_4) \\ l(F_3 - F_4) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix}$$

此外，世界坐标系下三个方向上的角速度近似为无人机坐标系相对于世界坐标系在三个轴上的旋转角度的一阶导数，可将这一种控制可以通过 PID 控制原理应用于实际问题中。具体实现分为位置控制和姿态控制两个部分，实现的 PID 控制如下：

1. 位置控制：

   PID：$\ddot{p}_{i,c} = \ddot{p}_i^{des} + K_{d,i}(\dot{p}_i^{des} - \dot{p}) + K_{p,i}(p_i^{des} - p_i)$

   模型：$u_1 = m(g + \ddot{p}_{3,c})$

   $$\phi_c = \frac{1}{g}(\ddot{p}_{1,c}\sin\psi - \ddot{p}_{2,c}\cos\psi)$$

   $$\theta_c = \frac{1}{g}(\ddot{p}_{1,c}\cos\psi + \ddot{p}_{2,c}\sin\psi)$$

2. 姿态控制：

   PID：$\begin{bmatrix} \ddot{\phi}_c \\ \ddot{\theta}_c \\ \ddot{\psi}_c \end{bmatrix} = \begin{bmatrix} K_{p,\phi}(\phi_c - \phi) + K_{d,\phi}(\dot{\phi}_c - \dot{\phi}) \\ K_{p,\theta}(\theta_c - \theta) + K_{d,\theta}(\dot{\theta}_c - \dot{\theta}) \\ K_{p,\psi}(\psi_c - \psi) + K_{d,\psi}(\dot{\psi}_c - \dot{\psi}) \end{bmatrix}$

   模型：线性近似的欧拉方程

2.2 坐标系转换

实验中无人机相关的运动信息存在两种坐标系：

1. 世界坐标系：包括期望设定值。
2. 参数 u_1 处于动捕世界坐标系下。
3. 机体坐标系：包括机体的运动角速度等。

实机中我们能够获得的姿态有里程计与 IMU 进行 EKF 融合后的里程计姿态表示 ${}^{W}R_B$、北东地坐标系下的 IMU 姿态表示 ${}^{E}R_B$，以及经过线性化公式得出的世界坐标系下的期望姿态表示 ${}^{W}R_{B'}$。结合以上，得到的最终姿态转换公式如下：

$${}^{E}R_{B'} = {}^{E}R_B({}^{W}R_B)^{-1}{}^{W}R_{B'}$$

2.3 代码解释

具体算法流程如下：

1. 计算期望的加速度：

   首先定义位置误差 $err.p = des.p - odom.p$，速度误差 $err.v = des.v - odom.v$，`des` 为期望值，`odom` 为动捕得到的速度或位置值，我们设定期望速度为 $[0,0,0]$, 期望加速度 `des_acc` =[0,0,0]。

   随后根据 PID 控制，得到期望的加速度。

```
// compute desired acceleration
Eigen::Vector3d des_acc(0.0, 0.0, 0.0);

double p_a_z_c = param_.gain.Kv2 * (0 - odom.v(2)) + param_.gain.
    Kp2 * (des.p(2) - odom.p(2)) + param_.gra;
double p_a_y_c = param_.gain.Kv1 * (0 - odom.v(1)) + param_.gain.
    Kp1 * (des.p(1) - odom.p(1));
double p_a_x_c = param_.gain.Kv0 * (0 - odom.v(0)) + param_.gain.
    Kp0 * (des.p(0) - odom.p(0));
```

2. 获得偏转角：
   从 imu 获得四元数并转换为旋转矩阵 `rotationMatrix` ，并获得 ZYX(yaw,pitch,roll) 欧拉角。
   计算 `sy` 以检查旋转矩阵近奇异性，若 `sy` 非常小，则表示出现万向节死锁，将滚动角度设置为 0 回退。其中 yaw 角用 `atan2` 来求得。

```
Eigen::Matrix3d rotationMatrix = imu.q.toRotationMatrix();
Eigen::Vector3d eulerAngles = rotationMatrix.eulerAngles(2, 1, 0);
float sy = sqrt(rotationMatrix(0,0)*rotationMatrix(0,0)+
    rotationMatrix(1,0)*rotationMatrix(1,0));
bool sing = sy < 1e-6;
if(!sing) eulerAngles(0) = atan2(rotationMatrix(1,0),rotationMatrix
    (0,0));
else eulerAngles(0) = 0;
```

3. 更新偏转角：
   将偏转角更新为从动捕获得，流程与上一步骤一样，此时alpha为测量得到的偏转角。

```
double alpha = eulerAngles(0);
Eigen::Matrix3d rotationMatrix_odom = odom.q.toRotationMatrix();
Eigen::Vector3d eulerAngles_odom = rotationMatrix_odom.eulerAngles
    (2, 1, 0);
alpha = eulerAngles_odom(0);
```

4. 计算姿态：
   通过此前的公式可得 `pose(2)` (roll 角)，`pose(1)` (pitch 角)， `pose(0)` 即为 yaw 角 alpha。

```
Eigen::Vector3d pose;
pose(2) = (p_a_x_c*sin(alpha) - p_a_y_c*cos(alpha)) / param_.gra;
```

```
pose(1) = (p_a_x_c*cos(alpha) + p_a_y_c*sin(alpha)) / param_.gra;
pose(0) = eulerAngles_odom(0);
```

5. 计算所需推力:
   将所需的加速度合并到矢量 des_acc 中,调用函数 computeDesiredCollectiveThrustSignal 来计算实现此加速所需的推力并存储在 u.thrust 中。

```
des_acc << p_a_x_c, p_a_y_c, p_a_z_c;
// supposed to be readonly, compute thrust by acc
u.thrust = computeDesiredCollectiveThrustSignal(des_acc);
```

6. 得到姿态四元数:
   将无人机姿态的欧拉角转变为四元数储存在 u.q 中，随后将姿态转变为北东地坐标系下。

```
u.q = Eigen::Quaterniond(1.0, 0.0, 0.0, 0.0);
u.q = Eigen::AngleAxisd(pose(0), Eigen::Vector3d::UnitZ())
    * Eigen::AngleAxisd(pose(1), Eigen::Vector3d::UnitY())
    * Eigen::AngleAxisd(pose(2), Eigen::Vector3d::UnitX());
u.q = imu.q * odom.q.inverse() * u.q;
```

3 ROS 模拟仿真

编写程序后我们先进行了一系列仿真，用以检验结果并探求较好的 PID 控制系数。

1. 将获取的包解压编译，在 ROS 中运行已完成的程序。
2. ROS 仿真通过控制器的控制输入与控制指令，在线上平台中实现无人机的飞行模拟。
3. 通过录 rosbag 记录里程计数据，并通过 plotjuggler 进行数据分析。
4. 通过多次重复测试，以获得更加的控制参数。

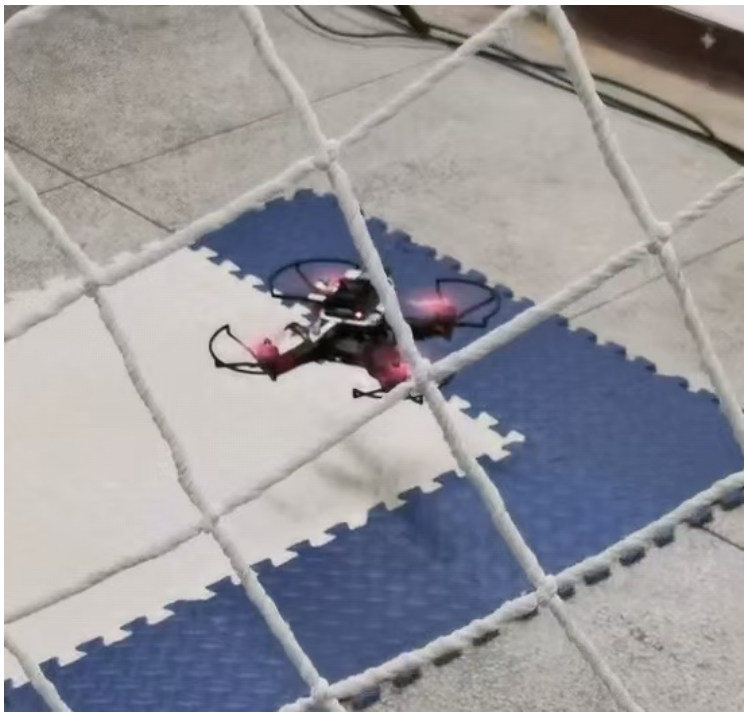
通过 ROS 仿真与模拟，我们的无人机控制程序得到了较好的仿真结果，并将测试得到的参数用于实机测试。

4 实现效果

在本次实验中，我们组装的无人机实现了基本的结构并能够完成飞行动作。在此后的无人机悬停控制程序实现了无人机在环境场景中的稳定悬浮。

整体效果上，由于手动设置的一定误差，无人机在手动操作时受到了操作手的较大影响，运动较为不稳定。而在编写控制程序后，无人机实现了良好的控制效果，基本实

现了稳定动作，整体的飞行也更为平滑流畅。在悬停测试时，无人机在悬停的最初阶段时常发生一次较大的偏航后立即稳定，我们推测为程序中在使用欧拉角转化的过程中的结果缺少范围限制所致。但这并不影响无人机的稳定性，且整体垂直方向上的超调与波动均较小。



无人机悬停最终效果图

在实机测试中，由于我们预先通过仿真过程得到了较好的控制参数，不仅大大缩短了实机测试的优化时间，而且在整体的无人机实机也展现了较好的性能，基本与仿真结果一致。

5 小组分工

- 毛永奇：搭建硬件、配置飞控（试飞）、编写和测试控制代码；
- 窦泽鑫：搭建硬件、配置树莓派环境、参与调试代码；
- 娄开杨：搭建硬件、参与调试代码
- 薛宇航：搭建硬件、编写实验报告
- 李子曦：搭建硬件、编写实验报告
- 孙上为：搭建硬件、配置飞控（试飞）、编写实验报告

6 总结与体会

在本次实验中我们从硬件入手搭建了一架无人机，完成了无人机的线性控制器代码编写，实现仿真环境下的自主悬停并通过树莓派最终完成了无人机实机自主悬停。在焊接，组装，调试的过程中，我们发现问题并将其解决，这加深了我们对无人机软硬件、姿态转换、坐标系转换、PID 控制原理的理解，最终实现了较好的效果。对在这过程中帮助我们编写、调试代码的队友以及指导我们的助教学长们致以最诚挚的感谢。