深蓝学院
shenlanxueyuan.com

第六章作业思路分享

主讲人 刘哲铭

第六章作业思路分享

深蓝学院 shenlanxueyuan.com

●一、必作题

●1. 基于中值法的解算

首先，参照课件的内容，中值法和欧拉法的区别在于相邻两帧之间的更新用的是平均值还是上一时刻的值，中值法用的是均值

a. 欧拉法： $\phi = \omega_{k-1}(t_k - t_{k-1})$

b. 中值法： $\phi = \frac{\omega_{k-1}+\omega_k}{2}(t_k - t_{k-1})$

第六章作业思路分享

深蓝学院
shenlanxueyuan.com

●一、必作题

●1. 基于中值法的解算

中值法需要用到前后两相邻帧的信息，而程序里是用的消息队列，所以，设置两个索引号，0和1，索引imu_data_buffer_得到相邻的imu，用于更新位姿，然后pop掉最早的imu，如此循环更新位姿即可

- 一、必作题
- 1. 基于中值法的解算

```
Eigen::Vector3d angular_delta;
Eigen::Vector3d velocity_delta;
Eigen::Matrix3d R_curr;
Eigen::Matrix3d R_prev;
size_t current_idx = 1, prev_idx = 0;
double delta_time =
    imu_data_buff_.at(current_idx).time - imu_data_buff_.at(prev_idx).time;
GetAngularDelta(current_idx, prev_idx, angular_delta);
UpdateOrientation(angular_delta, R_curr, R_prev);
GetVelocityDelta(current_idx, prev_idx, R_curr, R_prev, delta_time,
                 velocity_delta);
UpdatePosition(delta_time, velocity_delta);
savePoseTUM(imu_data_buff_.at(current_idx).time, pose_, estimate_ofile_);
imu_data_buff_.pop_front();
```

更新角度增量 --> 更新姿态 --> 更新速度 --> 更新位置

第六章作业思路分享

深蓝学院 shenlanxueyuan.com

●一、必作题

●1. 基于中值法的解算

角度增量：
```
angular_delta = 0.5 * delta_t * (angular_vel_curr + angular_vel_prev);
```

速度增量更新：
```
velocity_delta = 0.5 * delta_t * (linear_acc_curr + linear_acc_prev);
```

位置更新：
```
pose_.block<3, 1>(0, 3) += delta_t * vel_ + 0.5 * delta_t * velocity_delta;
vel_ += velocity_delta;
```

2. 基于欧拉法的解算，对比精度

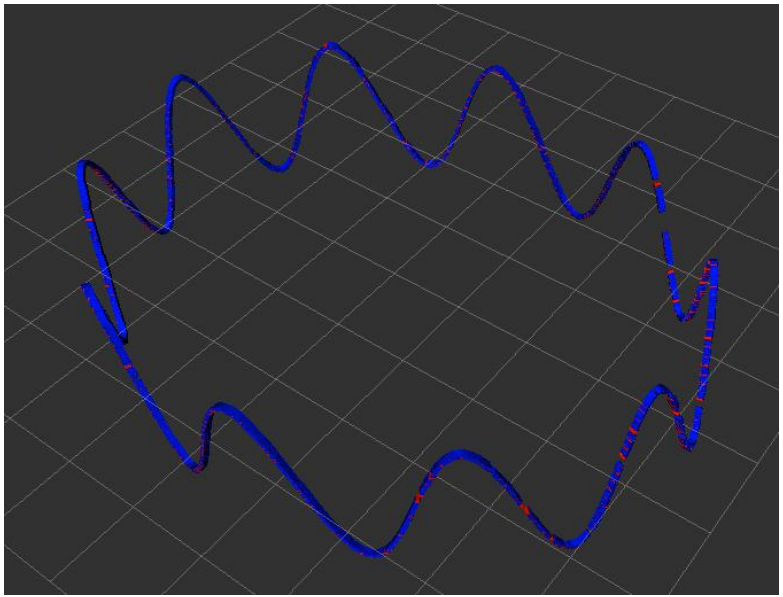
这道题目相对来说更简单，只需要将之前步骤里的更新换成欧拉法就可以，可以记录下数据，最后用EVO评测

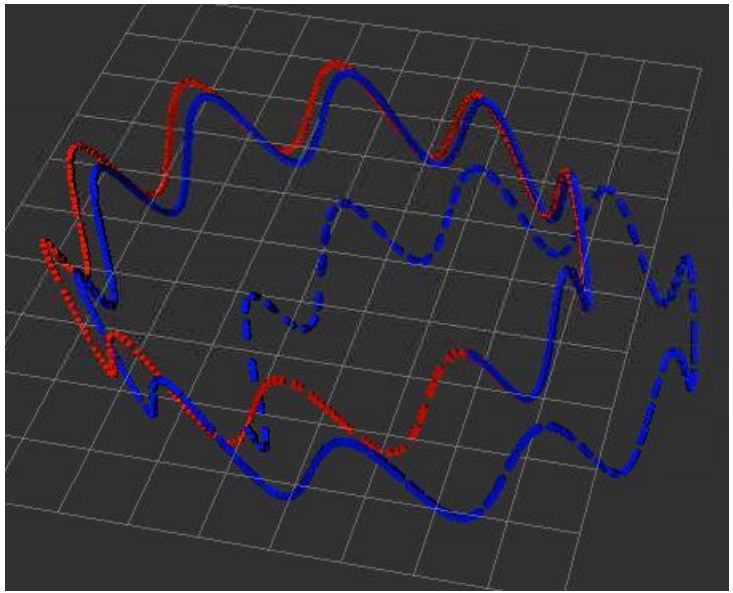
第六章作业思路分享

深蓝学院
shenlanxueyuan.com

●一、必作题

- 第二题结果：





第六章作业思路分享

深蓝学院
shenlanxueyuan.com

- **一、必作题**
- 3. 利用IMU仿真程序，自己生成不同运动状况(静止、匀速、加减速、快速转弯等)的仿真数据，对比两种解算方法精度差异与运动状况的关系，并给出原因分析

第三题需要花费一定的耐心和时间，建议先看gnss_ins_sim的README，弄懂数据生成的格式即各个坐标系的定义

https://github.com/Aceinna/gnss-ins-sim

第六章作业思路分享

深蓝学院 shenlanxueyuan.com

●一、必作题

a. 环境问题：

首先，如果使用的是docker镜像，那么可以直接使用作业中的launch文件生成对应的bag包；如果不是用的docker，则需要将gnss_ins_sim包安装到系统中去.

在 gnss-ins-sim 包中有个setup.py，可用于安装

b. 数据生成问题:

因为这个仿真软件的数据是表示在NED下的，和作业中的代码是一致的，所以只需要将对应的数据生成，并提供/pose/ground_truth即可

运动文件的生成和设置参考软件包的README

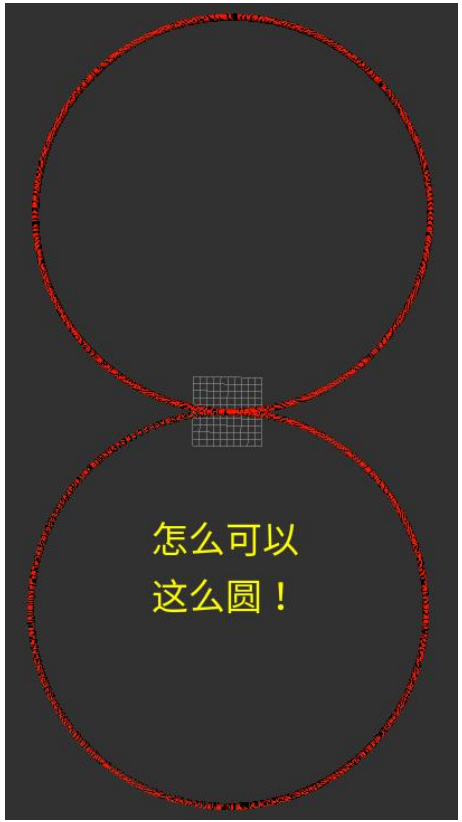
第六章作业思路分享

●一、必作题

b. 数据生成问题:

这里举个例子，比如绕飞8字，command = 1表示指令是角速度和加速度指令，每个的时间是36秒，gps可见，初始时刻只有x方向有速度

A	B	C	D	E	F	G	H	I
ini lat (de	ini lon (d	ini alt (m	ini vx_bo	ini vy_bo	ini vz_bo	ini yaw (d	ini pitch	ini roll (deg)
32	120	0	5	0	0	0	0	0
comman	yaw (deg	pitch (de	roll (deg)	vx_body	vy_body	vz_body	comman	GPS visibility
1	10	0	0	0	0	0	36	1
1	-10	0	0	0	0	0	36	1
1	10	0	0	0	0	0	36	1
1	-10	0	0	0	0	0	36	1



第六章作业思路分享

深蓝学院 shenlanxueyuan.com

●一、必作题

c. 真实值:

在生成bag包的脚本文件里，同样可以记录下gt值，gt值即仿真软件中的ref值，它与imu是同样的长度的，记录位置，速度以及姿态，为了区分，这里重新写了一下。然后，将数据保存到bag中，就可以用于对比结果啦，相信大家可以完成后续操作！

```
# ref_pose measurements:
full_ref_pose_mesurement = list()
step_gnss_size = 1.0 / fs_imu
print("step_gnss_size is: "+ str(step_gnss_size))
for j, (ref_pos, ref_pos_quad, ref_vel) in enumerate(
    zip(
        # a. pos
        sim.dmgr.get_data_all('ref_pos').data,
        # b. pos_quad
        sim.dmgr.get_data_all('ref_att_quat').data,
        # c. ref_vel
        sim.dmgr.get_data_all('ref_vel').data
    )
):
    ref_pose_mesurement = dict()
    ref_pose_mesurement['data'] = dict()
    ref_pose_mesurement['stamp'] = j*step_gnss_size
    ref_pose_mesurement['data']['ref_pos_x'] = ref_pos[0]
    ref_pose_mesurement['data']['ref_pos_y'] = ref_pos[1]
    ref_pose_mesurement['data']['ref_pos_z'] = ref_pos[2]
    ref_pose_mesurement['data']['ref_pos_quad_w'] = ref_pos_quad[0]
    ref_pose_mesurement['data']['ref_pos_quad_x'] = ref_pos_quad[1]
    ref_pose_mesurement['data']['ref_pos_quad_y'] = ref_pos_quad[2]
    ref_pose_mesurement['data']['ref_pos_quad_z'] = ref_pos_quad[3]
    ref_pose_mesurement['data']['ref_vel_x'] = ref_vel[0]
    ref_pose_mesurement['data']['ref_vel_y'] = ref_vel[1]
    ref_pose_mesurement['data']['ref_vel_z'] = ref_vel[2]
    full_ref_pose_mesurement.append(ref_pose_mesurement)
return full_imu_mesurement,full_ref_pose_mesurement
```

第六章作业思路分享

●二、附加题

这里讲一下RK4积分(4阶龙格库塔法)，RK4相当于将区间用4个导数组合，来代替欧拉法的斜率，分别是区间起点，中间点两种斜率以及区间结束点斜率

参考：

《Quaternion kinematics for the error-state Kalman filter》

A.3 The RK4 method

This is usually referred to as simply the Runge-Kutta method. It assumes evaluation values for $f()$ at the start, midpoint and end of the interval. And it uses four stages or iterations to compute the integral, with four derivatives, $k_1 \dots k_4$, that are obtained sequentially. These derivatives, or *slopes*, are then weight-averaged to obtain the 4th-order estimate of the derivative in the interval.

The RK4 method is better specified as a small algorithm than a one-step formula like the two methods above. The RK4 integration step is,

$$\boxed{\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{\Delta t}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)}\,, \tag{334}$$

that is, the increment is computed by assuming a slope which is the weighted average of the slopes k_1, k_2, k_3, k_4, with

$$k_1 = f(t_n, \mathbf{x}_n) \tag{335}$$

$$k_2 = f\left(t_n + \frac{1}{2}\Delta t\ ,\ \mathbf{x}_n + \frac{\Delta t}{2}k_1\right) \tag{336}$$

$$k_3 = f\left(t_n + \frac{1}{2}\Delta t\ ,\ \mathbf{x}_n + \frac{\Delta t}{2}k_2\right) \tag{337}$$

$$k_4 = f\left(t_n + \Delta t\ ,\ \mathbf{x}_n + \Delta t \cdot k_3\right)\,. \tag{338}$$

The different slopes have the following interpretation:

- k_1 is the slope at the beginning of the interval, using $\mathbf{x}_n$, (Euler's method);
- k_2 is the slope at the midpoint of the interval, using $\mathbf{x}_n + \frac{1}{2}\Delta t \cdot k_1$, (midpoint method);
- k_3 is again the slope at the midpoint, but now using $\mathbf{x}_n + \frac{1}{2}\Delta t \cdot k_2$;
- k_4 is the slope at the end of the interval, using $\mathbf{x}_n + \Delta t \cdot k_3$.

●二、附加题

对应到程序中，我们知道：dp/dt = f(t,p) = v(t)， dv/dt = f(t,v) = a(t)

所以代入到RK4的方程中就有：

pk1 = v;　　vk1 = a0;

pk2 = v + 0.5*dt*vk1;　　vk2 = a_mid;

pk3 = v + 0.5*dt*vk2;　　vk3 = vk2 = a_mid;

pk4 = v + dt*vk3;　　vk4 = a1;

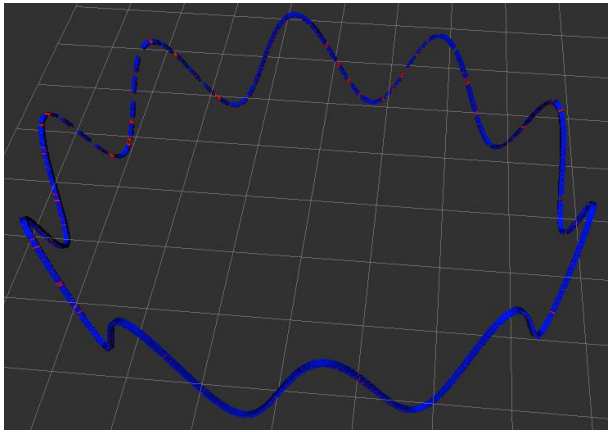
第六章作业思路分享

深蓝学院 shenlanxueyuan.com

●二、附加题

```
} else if (integration_method_ == IntegrationMethod::RK4) {
  Eigen::Quaterniond prev_q(R_prev);
  Eigen::Quaterniond current_q(R_curr);
  Eigen::Quaterniond half_q = prev_q.slerp(0.5, current_q);

  Eigen::Vector3d mid_acc = 0.5 * (imu_data_curr.linear_acceleration +
                                   imu_data_prev.linear_acceleration);
  vk1_ = linear_acc_prev;
  vk2_ = GetUnbiasedLinearAcc(mid_acc, half_q.toRotationMatrix());
  vk3_ = vk2_;
  vk4_ = linear_acc_curr;
  velocity_delta = 1.0 * delta_t / 6 * (vk1_ + 2 * vk2_ + 2 * vk3_ + vk4_);
}
```



```
if (integration_method_ == IntegrationMethod::RK4) {
  Eigen::Vector3d pk1 = vel_;
  Eigen::Vector3d pk2 = vel_ + 0.5 * delta_t * vk1_;
  Eigen::Vector3d pk3 = vel_ + 0.5 * delta_t * vk2_;
  Eigen::Vector3d pk4 = vel_ + delta_t * vk3_;
  pose_.block<3, 1>(0, 3) +=
      1.0 * delta_t / 6 * (pk1 + 2 * pk2 + 2 * pk3 + pk4);
  vel_ += velocity_delta;
} else {
```