# B样条插值估计轨迹

控制点的采样时间

/// Uniform sampling time for our control points 控制点的采样时间！

double dt;

系统的起始时间

/// Start time of the system

double timestamp\_start;

存放控制点

/// Our control SE3 control poses (R\_ItoG, p\_IinG)

std::map<double,Eigen::MatrixXd> control\_points;

提取系统计算的起始时间

/// Returns the simulation start time that we should start simulating from

double get\_start\_time() {

   return timestamp\_start;

}

找给定时间下位姿的上下边界

bool find\_bounding\_poses(double timestamp, std::map<double,Eigen::MatrixXd> &poses,

double &t0, Eigen::Matrix4d &pose0, double &t1, Eigen::Matrix4d &pose1);

取时间靠前的一个位姿

auto lower\_bound = poses.lower\_bound(timestamp);

取时间靠后的一个位姿

auto upper\_bound = poses.upper\_bound(timestamp);

保存前一帧的时间信息和位姿信息

// If we found the older one, set it

if (found\_older) {

    t0 = lower\_bound->first;

    pose0 = lower\_bound->second;

}

保存后一帧的时间与位姿信息

// If we found the newest one, set it

if (found\_newer) {

    t1 = upper\_bound->first;

    pose1 = upper\_bound->second;

}

四个点确定样条曲线

bool find\_bounding\_control\_points(double timestamp, std::map<double,Eigen::MatrixXd> &poses,

double &t0, Eigen::Matrix4d &pose0, double &t1, Eigen::Matrix4d &pose1,

double &t2, Eigen::Matrix4d &pose2, double &t3, Eigen::Matrix4d &pose3);};

提供给定时间下的姿态和位置

bool get\_pose(double timestamp, Eigen::Matrix3d &R\_GtoI, Eigen::Vector3d &p\_IinG);

全局坐标系下的姿态

R\_GtoI

全局坐标系下的位置

p\_IinG

接下来就是套公式了！

求

double DT = (t2-t1);



double u = (timestamp-t1)/DT;

，，

double b0 = 1.0/6.0\*(5+3\*u-3\*u\*u+u\*u\*u);

double b1 = 1.0/6.0\*(1+3\*u+3\*u\*u-2\*u\*u\*u);

double b2 = 1.0/6.0\*(u\*u\*u);

，，

// Calculate interpolated poses

Eigen::Matrix4d A0 = exp\_se3(b0\*log\_se3(Inv\_se3(pose0)\*pose1));

Eigen::Matrix4d A1 = exp\_se3(b1\*log\_se3(Inv\_se3(pose1)\*pose2));

Eigen::Matrix4d A2 = exp\_se3(b2\*log\_se3(Inv\_se3(pose2)\*pose3));

获取插值点的位姿，

Eigen::Matrix4d pose\_interp = pose0\*A0\*A1\*A2;

R\_GtoI = pose\_interp.block(0,0,3,3).transpose();

p\_IinG = pose\_interp.block(0,3,3,1);

获取指定时间下的角速度与线速度

bool get\_velocity(double timestamp, Eigen::Matrix3d &R\_GtoI, Eigen::Vector3d &p\_IinG, Eigen::Vector3d &w\_IinI, Eigen::Vector3d &v\_IinG);

全局坐标系下的姿态

R\_GtoI

全局坐标系下的位置

p\_IinG

惯性系下的角速度

w\_IinI

全局坐标下的线速度

v\_IinG

接下来就是套公式了！

求

double DT = (t2-t1);



double u = (timestamp-t1)/DT;

，，

double b0 = 1.0/6.0\*(5+3\*u-3\*u\*u+u\*u\*u);

double b1 = 1.0/6.0\*(1+3\*u+3\*u\*u-2\*u\*u\*u);

double b2 = 1.0/6.0\*(u\*u\*u);

，，

double b0dot = 1.0/(6.0\*DT)\*(3-6\*u+3\*u\*u);

double b1dot = 1.0/(6.0\*DT)\*(3+6\*u-6\*u\*u);

double b2dot = 1.0/(6.0\*DT)\*(3\*u\*u);

，，

Eigen::Matrix<double,6,1> omega\_10 = log\_se3(Inv\_se3(pose0)\*pose1);

Eigen::Matrix<double,6,1> omega\_21 = log\_se3(Inv\_se3(pose1)\*pose2);

Eigen::Matrix<double,6,1> omega\_32 = log\_se3(Inv\_se3(pose2)\*pose3);

Eigen::Matrix4d A0 = exp\_se3(b0\*omega\_10);

Eigen::Matrix4d A1 = exp\_se3(b1\*omega\_21);

Eigen::Matrix4d A2 = exp\_se3(b2\*omega\_32);



Eigen::Matrix4d A0dot = b0dot\*hat\_se3(omega\_10)\*A0;

Eigen::Matrix4d A1dot = b1dot\*hat\_se3(omega\_21)\*A1;

Eigen::Matrix4d A2dot = b2dot\*hat\_se3(omega\_32)\*A2;

计算姿态和位置

// Get the interpolated pose

Eigen::Matrix4d pose\_interp = pose0\*A0\*A1\*A2;

R\_GtoI = pose\_interp.block(0,0,3,3).transpose();

p\_IinG = pose\_interp.block(0,3,3,1);



Eigen::Matrix4d vel\_interp = pose0\*(A0dot\*A1\*A2+A0\*A1dot\*A2+A0\*A1\*A2dot);

求局部坐标系下的角速度

// NOTE: Rdot = R\*skew(omega) => R^T\*Rdot = skew(omega)

w\_IinI = vee(pose\_interp.block(0,0,3,3).transpose()\*vel\_interp.block(0,0,3,3));

求全局坐标系下的线速度

v\_IinG = vel\_interp.block(0,3,3,1);

获取指定时间下的角加速度和线加速度

bool get\_acceleration(double timestamp, Eigen::Matrix3d &R\_GtoI, Eigen::Vector3d &p\_IinG,

Eigen::Vector3d &w\_IinI, Eigen::Vector3d &v\_IinG,

Eigen::Vector3d &alpha\_IinI, Eigen::Vector3d &a\_IinG);

接下来就是套公式了！

求

double DT = (t2-t1);



double u = (timestamp-t1)/DT;

，，

double b0 = 1.0/6.0\*(5+3\*u-3\*u\*u+u\*u\*u);

double b1 = 1.0/6.0\*(1+3\*u+3\*u\*u-2\*u\*u\*u);

double b2 = 1.0/6.0\*(u\*u\*u);

，，

double b0dot = 1.0/(6.0\*DT)\*(3-6\*u+3\*u\*u);

double b1dot = 1.0/(6.0\*DT)\*(3+6\*u-6\*u\*u);

double b2dot = 1.0/(6.0\*DT)\*(3\*u\*u);

，，

double b0dotdot = 1.0/(6.0\*DT\*DT)\*(-6+6\*u);

double b1dotdot = 1.0/(6.0\*DT\*DT)\*(6-12\*u);

double b2dotdot = 1.0/(6.0\*DT\*DT)\*(6\*u);

，，

Eigen::Matrix<double,6,1> omega\_10 = log\_se3(Inv\_se3(pose0)\*pose1);

Eigen::Matrix<double,6,1> omega\_21 = log\_se3(Inv\_se3(pose1)\*pose2);

Eigen::Matrix<double,6,1> omega\_32 = log\_se3(Inv\_se3(pose2)\*pose3);

Eigen::Matrix4d A0 = exp\_se3(b0\*omega\_10);

Eigen::Matrix4d A1 = exp\_se3(b1\*omega\_21);

Eigen::Matrix4d A2 = exp\_se3(b2\*omega\_32);



Eigen::Matrix4d A0dot = b0dot\*hat\_se3(omega\_10)\*A0;

Eigen::Matrix4d A1dot = b1dot\*hat\_se3(omega\_21)\*A1;

Eigen::Matrix4d A2dot = b2dot\*hat\_se3(omega\_32)\*A2;



Eigen::Matrix4d A0dotdot = b0dot\*hat\_se3(omega\_10)\*A0dot+b0dotdot\*hat\_se3(omega\_10)\*A0;

Eigen::Matrix4d A1dotdot = b1dot\*hat\_se3(omega\_21)\*A1dot+b1dotdot\*hat\_se3(omega\_21)\*A1;

Eigen::Matrix4d A2dotdot = b2dot\*hat\_se3(omega\_32)\*A2dot+b2dotdot\*hat\_se3(omega\_32)\*A2;

计算姿态和位置

// Get the interpolated pose

Eigen::Matrix4d pose\_interp = pose0\*A0\*A1\*A2;

R\_GtoI = pose\_interp.block(0,0,3,3).transpose();

p\_IinG = pose\_interp.block(0,3,3,1);



Eigen::Matrix4d vel\_interp = pose0\*(A0dot\*A1\*A2+A0\*A1dot\*A2+A0\*A1\*A2dot);

求局部坐标系下的角速度

w\_IinI = vee(pose\_interp.block(0,0,3,3).transpose()\*vel\_interp.block(0,0,3,3));

求全局坐标系下的线速度

v\_IinG = vel\_interp.block(0,3,3,1);



Eigen::Matrix4d acc\_interp = pose0\*(A0dotdot\*A1\*A2+A0\*A1dotdot\*A2+A0\*A1\*A2dotdot+2\*A0dot\*A1dot\*A2+2\*A0\*A1dot\*A2dot+2\*A0dot\*A1\*A2dot);

计算惯性系下的角加速度

// NOTE: Rdot = R\*skew(omega)

// NOTE: Rdotdot = Rdot\*skew(omega) + R\*skew(alpha) => R^T\*(Rdotdot-Rdot\*skew(omega))=skew(alpha)

Eigen::Matrix3d omegaskew = pose\_interp.block(0,0,3,3).transpose()\*vel\_interp.block(0,0,3,3);

alpha\_IinI = vee(pose\_interp.block(0,0,3,3).transpose()\*(acc\_interp.block(0,0,3,3)-vel\_interp.block(0,0,3,3)\*omegaskew));

计算全局系下的线加速度

a\_IinG = acc\_interp.block(0,3,3,1);

保存一系列的点信息，并产生控制点

void feed\_trajectory(std::vector<Eigen::VectorXd> traj\_points);

按要求存放候选点，包括时间戳和位姿矩阵

// convert all our trajectory points into SE(3) matrices

// we are given [timestamp, p\_IinG, q\_GtoI]

std::map<double,Eigen::MatrixXd> trajectory\_points;

for(size\_t i=0; i<traj\_points.size()-1; i++) {

     Eigen::Matrix4d T\_IinG = Eigen::Matrix4d::Identity();

     T\_IinG.block(0,0,3,3) = quat\_2\_Rot(traj\_points.at(i).block(4,0,4,1)).transpose();

T\_IinG.block(0,3,3,1) = traj\_points.at(i).block(1,0,3,1);

    trajectory\_points.insert({traj\_points.at(i)(0),T\_IinG});

}

得到最前和最后的时间戳

// Get the oldest timestamp

double timestamp\_min = INFINITY;

double timestamp\_max = -INFINITY;

for(std::pair<const double, Eigen::MatrixXd> &pose : trajectory\_points){

    if(pose.first <= timestamp\_min) {

        timestamp\_min = pose.first;

    }

    if(pose.first >= timestamp\_min) {

        timestamp\_max = pose.first;

    }

}

创建控制点

// then create spline control points

double timestamp\_curr = timestamp\_min;

while(true) {

// Get bounding posed for the current time

    double t0, t1;

    Eigen::Matrix4d pose0, pose1;

    bool success = find\_bounding\_poses(timestamp\_curr, trajectory\_points, t0, pose0, t1, pose1);

//printf("[SIM]: time curr = %.6f | lambda = %.3f | dt = %.3f | dtmeas = %.3f\n",timestamp\_curr,(timestamp\_curr-t0)/(t1-t0),dt,(t1-t0));

// If we didn't find a bounding pose, then that means we are at the end of the dataset

// Thus break out of this loop since we have created our max number of control points

    if(!success)

        break;

// Linear interpolation and append to our control points

    double lambda = (timestamp\_curr-t0)/(t1-t0);

    Eigen::Matrix4d pose\_interp = exp\_se3(lambda\*log\_se3(pose1\*Inv\_se3(pose0)))\*pose0;

    control\_points.insert({timestamp\_curr, pose\_interp});

    timestamp\_curr += dt;

//std::cout << pose\_interp(0,3) << "," << pose\_interp(1,3) << "," << pose\_interp(2,3) << std::endl;

}