# 软件源代码

## 运行环境

操作系统：Ubuntu18.04

所需依赖：

1. 机器人操作系统：ros melodic；
2. Eigen线性代数库：version >= 3.2.2；
3. GLPK线性规划求解器：GNU Linear Programming Kit，version=5.0；
4. Gurobi数学规划求解器：Gurobi 10.0版本；
5. 双线性规划库：cdd-lib，Version = 0.92a；

## 源代码

### ros自定义消息类型

#### FeetPosition.msg：

geometry\_msgs/Point[6] foot

#### hexapod\_RPY.msg：

geometry\_msgs/Point position

hit\_spider/hexapod\_RPY orientation

#### hexapod\_Base\_Pose.msg

float64 roll

float64 pitch

float64 yaw

#### hexapod\_State.msg：

#当前机器人状态

hit\_spider/hexapod\_Base\_Pose base\_Pose\_Now

int8[6] support\_State\_Now

int8[6] faultLeg\_State\_Now

hit\_spider/FeetPosition feetPositionNow

#下一步机器人状态

hit\_spider/hexapod\_Base\_Pose base\_Pose\_Next

int8[6] support\_State\_Next

int8[6] faultLeg\_State\_Next

hit\_spider/FeetPosition feetPositionNext

#移动方向

geometry\_msgs/Point move\_Direction

std\_msgs/String remarks

### 头文件

#### util.cpp

// 数据类型定义、一些常量、常用接口函数的声明

#ifndef HIT\_SPIDER\_UTIL\_HH

#define HIT\_SPIDER\_UTIL\_HH

#include <iostream>

#include <string>

#include <fstream> //文件操作

#include <sstream> //字符串操作,搭配fstream读取文件数据

#include <vector> //STL容器(可以尝试使用list保存备选节点)

#include <algorithm> //STL算法

#include <utility> //std::pair

#include <memory> //shared智能指针动态内存分配

#include <eigen3/Eigen/Dense>

#include <ros/ros.h>

#include <std\_msgs/String.h>

#include <geometry\_msgs/Point.h> //三维点

#include <geometry\_msgs/Transform.h> //坐标变换

#include <geometry\_msgs/TransformStamped.h> //用于广播的坐标变换(带时间戳)

#include <sensor\_msgs/JointState.h> //关节角度

#include <std\_msgs/ColorRGBA.h> //颜色,初始化Marker

#include <visualization\_msgs/Marker.h> //Marker可视化

#include <tf2\_ros/buffer.h> //动态坐标变换缓存

#include <tf2\_ros/transform\_listener.h> //坐标变换接收者

#include <tf2\_ros/transform\_broadcaster.h> //动态坐标变换发布

#include <tf2\_ros/static\_transform\_broadcaster.h> //静态坐标变换发布

#include <tf2\_geometry\_msgs/tf2\_geometry\_msgs.h> //tf2消息转换为geometry\_msgsg消息

#include <tf2/LinearMath/Quaternion.h> //欧拉角转四元数

#include <grid\_map\_ros/grid\_map\_ros.hpp> //grid\_map地图

#include <grid\_map\_msgs/GridMap.h>

#include "hit\_spider/hexapod\_RPY.h" //机体姿态

#include "hit\_spider/hexapod\_Base\_Pose.h" //机体位姿

#include "hit\_spider/FeetPosition.h" //六个腿落足点

#include "hit\_spider/hexapod\_State.h" //机器人状态State

#define PI 3.14159265358979323846

//------数据类型的定义------

typedef Eigen::Matrix<double, 2, 1> point\_Planar; // 二维平面上的点

typedef Eigen::Matrix<double, 3, 1> Vector3; // 三维点

typedef Eigen::Matrix<double, 6, 1> Vector6; // 六维状态

typedef Eigen::Matrix<double, Eigen::Dynamic, 1> VectorX; // 动态维数向量

typedef Eigen::Matrix<int, Eigen::Dynamic, 1> VectorXi;

typedef Eigen::Matrix<double, 3, 3> Matrix3; // 3×3矩阵

typedef Eigen::Matrix<double, 6, 3> Matrix63; // 6×3矩阵，六条腿的关节角度

typedef Eigen::Matrix<double, Eigen::Dynamic, 3> MatrixX3; // 三维向量按行排列的，可用于表示接触点、法向量序列（不知道有多少接触点）、雅克比矩阵序列

typedef Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic> MatrixXX;

Matrix3 crossMatrix(const Vector3 &x); // 求三维向量对应的反对称矩阵

//------机器人相关常量,全局共享变量声明------

#define FAULT\_LEG 1

#define NORMAL\_LEG 0

#define SUPPORT 1

#define SWING 0

extern const float body\_fixJi\_theta[6]; // 六条腿在Truck分布的角度

extern const std::vector<std::string> HEXAPOD\_JOINT\_STATE\_NAME; // 18个关节名称

extern const Eigen::Isometry3d TransMatrix\_FixJi\_Body[6]; // FixJi描述base的齐次变换矩阵(FixJi就是根关节默认位姿)

extern const Eigen::Isometry3d TransMatrix\_Body\_2\_fixJi[6]; // base描述FixGu的齐次变换矩阵(FixGu就是扇形坐标系)

extern const Vector3 Hexapod\_defaultFoothold[6]; // 默认落足位置(相对于base坐标系)

//------常用函数

/\*\*

\* @ : 生成区间内的一个随机整数

\* @description:

\* @param {int} minValue：区间下界

\* @param {int} maxValue：区间上界

\* @return {\*}：生成的随机整数

\*/

int rand\_customization(int minValue, int maxValue);

/\*\*

\* @ : 获取世界系下机器人Truck机体的齐次变换矩阵

\* @description:

\* @param {hit\_spider::hexapod\_Base\_Pose} &pose：机器人位姿，欧拉角和位置

\* @return {Eigen::Isometry3d}：EIgen的4×4齐次变换矩阵

\*/

Eigen::Isometry3d getTrans\_W\_B(const hit\_spider::hexapod\_Base\_Pose &pose);

geometry\_msgs::Transform getTransform\_W\_B(const hit\_spider::hexapod\_Base\_Pose &pose);

/\*\*

\* @ : 初始化树节点的状态，质心位姿、六条腿落足点均是默认位置、所有腿均是支撑腿、没有废腿、节点访问次数0

\* @description:

\* @param {MCTS\_State} &hexapodState

\* @return {\*}

\*/

void init\_hexapodState(hit\_spider::hexapod\_State &hexapodState);

#endif

#### static\_information.hh

// 静态数据：地图信息、静态坐标变换

#include "hit\_spider/util.hh"

// #define terrain\_file\_name "/home/ptw/dynamic\_MCTS/src/hit\_spider/config/terrain/随机大块.txt"

// #define terrain\_file\_name "/home/ptw/dynamic\_MCTS/src/hit\_spider/config/terrain/连续壕沟.txt"

#define terrain\_file\_name "/home/ptw/dynamic\_MCTS/src/hit\_spider/config/terrain/中间空.txt"

namespace Static\_Information

{

MatrixX3 get\_now\_Feasible\_foot\_position(const hit\_spider::hexapod\_Base\_Pose &pose);

extern const grid\_map::GridMap mapData;

extern const MatrixX3 Feasible\_foot\_position;

}

#### hexapod.hh

// 六足机器人对象

#ifndef HIT\_SPIDER\_HEXAPOD\_H\_

#define HIT\_SPIDER\_HEXAPOD\_H\_

#include "hit\_spider/util.hh"

struct Leg

{

Leg();

~Leg();

void set\_num(const int &num);

/\*\*

\* @ : 单腿逆运动学，由落足位置更新单腿的三个关节角度

\* @description:

\* @param {Point} foot\_Word：世界坐标系下的足端位置

\* @param {Transform} robotW：世界系下机器人Trunk机体的坐标变换T

\* @return {\*}：更新leg对象中的三个关节角度

\*/

void IK(const hit\_spider::hexapod\_Base\_Pose &Wolrd\_BasePose, const geometry\_msgs::Point &World\_foot);

int legNum;

// 单腿三个关节角度

float theta1;

float theta2;

float theta3;

};

class Hexapod

{

public:

Hexapod(ros::NodeHandle &nh); // 发布gu节相对于ji节的静态坐标变换

~Hexapod();

/\*\*

\* @ : 计算整机逆运动学，发布坐标变换、关节角度信息，同时绘图

\* @description:

\* @param {hexapod\_State} &p

\* @return {\*}

\*/

void IK\_Robot(const hit\_spider::hexapod\_State &p);

private:

Leg \*legs;

// 18个关节角度

ros::Publisher joint\_state\_pub\_; // 关节角度信息发布者

sensor\_msgs::JointState joint\_state;

// odom到机器人base\_link的坐标变换

tf2\_ros::TransformBroadcaster odom\_broadcaster; // 动态坐标变换发布者

geometry\_msgs::TransformStamped odom\_2\_trunck;

private:

};

/\*\*

\* @ : 根据机体位姿和落足点求解18个关节角度

\* @description:

\* @param {hexapod\_Base\_Pose} &base\_pose：机体位姿

\* @param {FeetPosition} &feet\_position：落足点

\* @return {\*}：六条腿的关节角度，每一行是一条腿的关节角度

\*/

Matrix63 whole\_body\_inverse\_kin(const hit\_spider::hexapod\_Base\_Pose &Wolrd\_BasePose, const hit\_spider::FeetPosition &Wolrd\_feet);

//返回指定腿(支撑腿)的关节角度

bool support\_leg\_inverse\_kin(const hit\_spider::hexapod\_Base\_Pose &Wolrd\_BasePose, const hit\_spider::FeetPosition &World\_feet, const std::vector<int> &support\_leg, MatrixX3 &joints\_output);

#endif

#### solve\_LP\_GLPK.hh

// 使用GLPK求解LP问题的接口头文件

#ifndef ROBOT\_STATE\_TRANSITION\_SOLVE\_LP\_GLPK\_HH

#define ROBOT\_STATE\_TRANSITION\_SOLVE\_LP\_GLPK\_HH

#include <glpk.h>

#include "hit\_spider/util.hh"

namespace Robot\_State\_Transition

{

/\*\*

\* min g^T \* x;

\* s.t.

\* Ax<=b;

\* Dx=d;

\* @description: 注意：只给Bretl.cpp使用的，其他问题别使用，因为处理无穷解情况有所不同!!!

\* @description: 注意：只给Bretl.cpp使用的，其他问题别使用，因为处理无穷解情况有所不同!!!

\* @description: 注意：只给Bretl.cpp使用的，其他问题别使用，因为处理无穷解情况有所不同!!!

\* @param A:线性不等式约束的系数矩阵

\* @param b:线性不等式约束的右边值

\* @param D:线性等式约束的系数矩阵

\* @param d:线性等式约束的右边值

\* @param g:线性目标值

\* @param vec\_opt:优化的最优决策变量

\* @return bool：优化是否成功

\*/

bool solve\_LP\_Bretl\_GLPK(const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d,

const VectorX &g,

VectorX &vec\_opt);

/\*\*

\* min(max) g^T \* x;

\* s.t.

\* Ax<=b;

\* Dx=d;

\* @description: 为GLPK设置的求解器接口，能输入Eigen矩阵形式的LP问题

\* @param A:线性不等式约束的系数矩阵

\* @param b:线性不等式约束的右边值

\* @param D:线性等式约束的系数矩阵

\* @param d:线性等式约束的右边值

\* @param g:线性目标值

\* @param minBounds:决策变量的上界

\* @param maxBounds:决策变量的下界

\* @param direction:优化的方向，默认是最小值

\* @param \*result\_x:接受变量的指针

\* @param \*direction:接受最优值的指针，默认是求解最小值

\* @return bool：求解器是否求解成功、目标函数值、决策变量值

\*/

bool solve\_LP\_GLPK(const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d,

const VectorX &g,

const VectorX &minBounds, const VectorX &maxBounds,

bool direction = false,

VectorX \*result\_x = nullptr,

double \*result\_optimization = nullptr);

}

#endif

#### Bretl.hh

// 满足线性摩擦锥约束的空间向平面的投影——类和函数的声明

#ifndef ROBOT\_STATE\_TRANSITION\_BRETL\_HH

#define ROBOT\_STATE\_TRANSITION\_BRETL\_HH

#include "hit\_spider/util.hh"

#include "solve\_LP\_GLPK.hh"

typedef Eigen::Matrix<double, 2, 1> point\_Planar; // 二维平面上的点

#define Threshold\_Expand 1e-2 // 如果想要更快更简单的集合图形就选择这个

// #define Threshold\_Expand 1e-4

#define Threshold\_Init 1e-5

namespace Robot\_State\_Transition

{

struct Vertex\_Planar; // 二维平面上的顶点

struct Polygon\_Planar; // 二维平面上的多边形

struct Vertex\_Planar

{

/\*\*

\* @ : Vertex\_Planar的默认构造函数

\* @description:Vertex\_Planar的构造函数

\* @return {\*}

\*/

Vertex\_Planar();

/\*\*

\* @ : 输入二维点坐标来初始化平面上的点

\* @description:Vertex\_Planar的构造函数

\* @param {point\_Planar} point\_input：输入二维点2×1

\* @return {\*}

\*/

Vertex\_Planar(point\_Planar point\_input);

/\*\*

\* @ : 拓展该顶点

\* @description: 该顶点与它的下一个连接点构成一条edge，沿着edge的垂直方向可以进行拓展

\* @return {\*}：返回拓展后的新Vertex顶点的指针

\*/

Vertex\_Planar \*expand(VectorX &g,

const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d);

point\_Planar point; // 二维平面上的Vertex顶点坐标

Vertex\_Planar \*next = nullptr; // 二维平面Vertex顶点连接的下一个Vertex顶点

bool expanded = false; // 该顶点（对应的线）是否拓展过

};

struct Polygon\_Planar

{

/\*\*

\* @ : 用三个顶点的指针来初始化平面多边形

\* @description: Polygon\_Planar的构造函数

\* @param {Vertex\_Planar} &v1：第一个顶点的指针，它连接的下一个顶点是v2

\* @param {Vertex\_Planar} &v2：第一个顶点的指针，它连接的下一个顶点是v3

\* @param {Vertex\_Planar} &v3：第一个顶点的指针，它连接的下一个顶点是v1

\* @return {\*}

\*/

Polygon\_Planar(Vertex\_Planar \*v1, Vertex\_Planar \*v2, Vertex\_Planar \*v3);

/\*\*

\* @ : 用三个顶点来初始化平面多边形

\* @description: Polygon\_Planar的构造函数

\* @param {Vertex\_Planar} \*v1：顶点1，连接顶点2

\* @param {Vertex\_Planar} \*v2：顶点2，连接顶点3

\* @param {Vertex\_Planar} \*v3：顶点3，连接顶点1

\* @return {\*}

\*/

Polygon\_Planar(const Vertex\_Planar &v1, const Vertex\_Planar &v2, const Vertex\_Planar &v3);

/\*\*

\* @ : delete堆区new出来的Vertex顶点对象

\* @description: Polygon\_Planar的析构函数

\* @return {\*}

\*/

~Polygon\_Planar();

/\*\*

\* @ : 多边形所有顶点是否全部拓展完

\* @description: 所有顶点全部多拓展完有两种情况：1：线性多边形和和高维凸包的投影完全重合；2：线性多边形边线和高维凸包投影边线很接近，误差满足一定范围

\* @return {\*}

\*/

bool all\_expanded() const;

/\*\*

\* @ :根据约束条件循环拓展多边形直至到达最大迭代次数

\* @description:

\* @param {VectorX} &g：优化目标值，输入的只是一个壳子

\* @param {MatrixXX} &A：不等式约束的约束矩阵

\* @param {VectorX} &b：不等式约束的右边值

\* @param {MatrixXX} &D：等式约束的约束矩阵

\* @param {VectorX} &d：等式约束的右边值

\* @param {int} &max\_iter：迭代的最大次数

\* @return {\*}

\*/

void iter\_expand(VectorX g,

const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d,

const int &max\_iter);

/\*\*

\* @ : 按照y值对多边形顶点重新排序

\* @description:

\* @return {\*}：排序后的顶点集合

\*/

void sort\_vertices();

/\*\*

\* @ : 输出间距大于min\_dist的顶点集合

\* @description:

\* @param {double} min\_dist：最小的顶点间距

\* @return {\*}：间距大于min\_dist的顶点集合

\*/

MatrixXX export\_vertices(double min\_dist = 1e-2) const;

std::vector<Vertex\_Planar \*> vertices; // 构成平面多边形的Vertex顶点指针集合

};

/\*\*

\* @ : 朝着一个方向进行优化，获取该方向上的边界点

\* @description:

\* @param {point\_Planar} &vdir：二维平面上优化的方向

\* @param {VectorX} &g：优化目标值，输入的只是一个壳子

\* @param {MatrixXX} &A：不等式约束的约束矩阵

\* @param {VectorX} &b：不等式约束的右边值

\* @param {MatrixXX} &D：等式约束的约束矩阵

\* @param {VectorX} &d：等式约束的右边值

\* @param {point\_Planar} &result\_opt：如果优化成功会返回平面上新扩展的点

\* @return {\*}：求解是否成功

\*/

bool optimize\_direction(const point\_Planar &vdir, VectorX &g,

const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d,

point\_Planar &result\_opt);

/\*\*

\* @ :朝着theta定义的方向进行拓展

\* @description:

\* @param {double} theta：与x轴的夹角定义拓展方向

\* @param {VectorX} &g：优化目标值，输入的只是一个壳子

\* @param {MatrixXX} &A：不等式约束的约束矩阵

\* @param {VectorX} &b：不等式约束的右边值

\* @param {MatrixXX} &D：等式约束的约束矩阵

\* @param {VectorX} &d：等式约束的右边值

\* @param {point\_Planar} &result\_opt：如果优化成功会返回平面上新扩展的点

\* @return {\*}：求解是否成功

\*/

bool optimize\_angle(const double theta, VectorX &g,

const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d,

point\_Planar &result\_opt);

/\*\*

\* @ :

\* @description:

\* @param {VectorX} &g：优化目标值，输入的只是一个壳子

\* @param {MatrixXX} &A：不等式约束的约束矩阵

\* @param {VectorX} &b：不等式约束的右边值

\* @param {MatrixXX} &D：等式约束的约束矩阵

\* @param {VectorX} &d：等式约束的右边值

\* @param {double} \*init\_angle：初始拓展角度（决定初始三个点）

\* @param {int} max\_iter：最大迭代次数（初始三个点算入其中）

\* @return {Polygon\_Planar}：

\*/

Polygon\_Planar compute\_polygon(VectorX &g,

const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d,

double \*init\_angle = nullptr,

int max\_iter = 1000);

/\*\*

\* @ : 高维凸包x投影为y=Ex+f，x定义为:Ax<=b;Dx=d;

\* @description:

\* @param {MatrixXX} &E：y = Ex + f

\* @param {VectorX} &f：

\* @param {MatrixXX} &A：Ax <= b

\* @param {VectorX} &b：

\* @param {MatrixXX} &D：Dx = d

\* @param {VectorX} &d：

\* @param {double} \*init\_angle：初始拓展方向

\* @param {int} max\_iter：最大拓展次数

\* @param {double} max\_radius：y的范围

\* @return {\*}：投影多边形的顶点集合

\*/

MatrixXX project\_polytope\_bretl(const MatrixXX &E, const VectorX &f,

const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d,

double \*init\_angle = nullptr,

int max\_iter = 1000, const double max\_radius = 1e5);

/\*\*

\* @ : 计算由CWC决定的质心平面XY可行域，顺时针排序的点！每一列是一个点！

\* @description: 用户接口

\* @param {MatrixX3} &contactPoints：接触点位置，每一行是一个位置

\* @param {MatrixX3} &contactNormals：接触点法向量，每一行是一个法向量

\* @param {double} frictionCoefficient：摩擦系数

\* @return {\*}：返回的点是按照顺时针排序的！

\*/

MatrixXX comput\_friction\_region(const MatrixX3 &contactPoints, const MatrixX3 &contactNormals\_input, const double frictionCoefficient, const double &m);

}

#endif

#### my\_cdd.hh

// 双线性变换库的用户接口

#include "hit\_spider/util.hh"

#include <cdd/setoper.h>

#include <cdd/cdd.h>

namespace Robot\_State\_Transition

{

// 顶点转换为不等式

bool vertices\_to\_H(const MatrixXX &vertices\_set, MatrixXX &H\_output, VectorX &h\_output);

}

#### kinematics\_constrain.hh

// 六足机器人运动学约束函数声明

#include "hit\_spider/util.hh"

#include "solve\_LP\_GLPK.hh"

namespace Robot\_State\_Transition

{

extern const MatrixXX A\_Ji\_foot; // 单腿foot坐标系下约束fixJi的约束矩阵

extern const VectorX b\_Ji\_foot; // 单腿fixJi坐标系下约束foot的右边值

extern const MatrixXX A\_foot\_Ji; // 单腿foot坐标系下约束fixJi的约束矩阵

extern const VectorX b\_foot\_Ji; // 单腿foot坐标系下约束fixJi的右边值

/\*\*

\* @ : 质心约束摆动腿足端

\* @description:

\* @param {hexapod\_Base\_Pose} &base\_pose：机体的位姿

\* @return {\*}：返回的是对六个足端的约束方程，7行为一组，摆动腿选择自己对应的约束方程即可

\*/

std::pair<MatrixXX, VectorX> get\_kinematics\_con\_cog\_foot(const hit\_spider::hexapod\_Base\_Pose &base\_pose);

// 足端约束机体的质心

std::pair<MatrixXX, VectorX> get\_kinematics\_con\_foot\_cog(const hit\_spider::hexapod\_Base\_Pose &base\_pose,const std::vector<int> &support\_leg, const MatrixX3 &contactPoints, const MatrixX3 &contactNormals);

}

#### dynamic\_constrain.hh

// 六足机器人动力学约束头文件——函数的声明

#include "hit\_spider/util.hh"

#include "hit\_spider/robot\_state\_transition/solve\_LP\_GLPK.hh"

namespace Robot\_State\_Transition

{

extern const double Mass; // 机体质量

extern const Vector3 MaxTorque; // 电机最大扭矩

// 获取质心动力学方程约束Df=d

std::pair<MatrixXX, VectorX> get\_dynamic\_eq\_constrain(const Vector3 &c, const int &num\_support\_leg, const MatrixX3 &contactPoints);

// 获取关节扭矩不等式约束Ax<=b

std::pair<MatrixXX, VectorX> get\_jointToque\_neq\_constrain(const int &num\_support\_leg, const MatrixX3 &Joints\_angles);

// 当前状态是否满足动力学约束

bool is\_meet\_dynamic\_con(const Vector3 &c, const int &num\_support\_leg, const MatrixX3 &contactPoints, const MatrixX3 &Joints\_angles);

}

#### MTree.hh

// 蒙特卡洛搜索树类头文件——声明类和函数

#ifndef HIT\_SPIDER\_DBM\_MTREE\_H

#define HIT\_SPIDER\_DBM\_MTREE\_H

#include "hit\_spider/util.hh"

#include "hit\_spider/MCTS/planning.hh"

struct MNode; // MTCS树节点声明

typedef std::shared\_ptr<MNode> MNode\_ptr; // MCTS树节点指针

struct BestChiledParameter

{

// UCB的权重分配并没有考虑静态稳定裕度

float C = 0.0; // UCB

float w1 = 0.0; // simLengthAverage，向后模拟仿真的平均拓展步长

float w2 = 0.0; // steplengthAverage，根节点到该节点的平均拓展步长

float w3 = 0.0; // disP2ME，父节点到该节点的距离

float w4 = 0.0;

float w5 = 0.0;

int simStepNum = 0;

};

struct MNode

{

// 构造函数：make\_shared创建共享指针时会调用构造函数

MNode() : Parent(nullptr), nodeDepth(0), visitTimes(0),

disParentToMe(0), simMaxDis(0), average\_step\_length(0)

{

}

MNode(const hit\_spider::hexapod\_State &state\_input) : Parent(nullptr), nodeDepth(0), visitTimes(0),

disParentToMe(0), simMaxDis(0), average\_step\_length(0)

{

this->element = state\_input;

}

hit\_spider::hexapod\_State element; // 节点对应的状态

std::vector<hit\_spider::hexapod\_State> alternativeStates; // 备选状态集，主要是下一步的支撑状态、质心位姿、摆动腿落足点

MNode\_ptr Parent; // 父节点指针

std::set<MNode\_ptr> children; // 节点的儿子节点构成的集合

int nodeDepth; // 节点的深度

int visitTimes; // 节点访问次数

double disParentToMe; // 到父节点的距离

double simMaxDis; // 向后仿真出来的最远位置x坐标

double average\_step\_length; // 从"最根节点"到该节点当前位置的平均步长

};

class MTree

{

public:

MTree(MNode\_ptr &root\_input); // 构造函数，设置根节点和它的depth

void Set\_PAR(const BestChiledParameter &PAR);

void Set\_PAR2(const BestChiledParameter &PAR2);

/\*\*

\* @ : 加入一个新的儿子节点

\* @description:

\* @param {MNode} \*node：儿子节点指针

\* @param {MNode} \*parent：对应的父节点指针

\* @return {\*}

\*/

void addNodeChild(MNode\_ptr &node, MNode\_ptr &parent);

/\*\*

\* @ : 获取当前根节点的最大拓展深度

\* @description:

\* @return {\*}

\*/

int getMaxDepth();

//------MCTS搜索树算法------

/\*\*

\* @ :寻找当前滑动根节点的下一层最优子节点

\* @description:

\* @param {BestChiledParameter} PAR

\* @return {\*}

\*/

MNode\_ptr findBestChild\_Expand(const MNode\_ptr &node\_findBestChild);

MNode\_ptr findBestChild\_Sliding\_Root();

/\*\*

\* @ : 拓展阶段

\* @description:

\* @param {MNode\_ptr} &node\_expand

\* @return {\*}

\*/

MNode\_ptr MCTS\_Expansion(MNode\_ptr &node\_expand);

/\*\*

\* @ : 选择和拓展阶段，选择一个节点，拓展它

\* @description:

\* @return {\*}

\*/

MNode\_ptr MCTS\_SelectionAndExpansion();

std::vector<hit\_spider::hexapod\_State> MCTS\_Simulation(MNode\_ptr &node\_sim);

/\*\*

\* @ : 对得分进行反向传播

\* @description:

\* @param {MNode\_ptr} &node\_back

\* @return {\*}

\*/

bool MCTS\_Backpropagation(MNode\_ptr &node\_back);

// private:

MNode\_ptr root; // “最”根节点

MNode\_ptr Sliding\_New\_root; // 滑动根节点

BestChiledParameter PAR; // selectionAndExpansion时使用的参数

BestChiledParameter PAR2; // sliding根节点时使用的参数

};

#endif

#### planning.hh

// 规划过程中使用的函数声明

#ifndef HIT\_SPIDER\_PLANNING\_H\_

#define HIT\_SPIDER\_PLANNING\_H\_

#include "hit\_spider/util.hh" //数据类型

#include "hit\_spider/hexapod.hh" //求解逆运动学关节角

#include "hit\_spider/static\_information.hh" //寻找环境中可落足点

#include "hit\_spider/robot\_state\_transition/Bretl.hh" //CWC

#include "hit\_spider/robot\_state\_transition/my\_cdd.hh" //双线性变换库

#include "hit\_spider/robot\_state\_transition/kinematics\_constrain.hh" //运动学约束

#include "hit\_spider/robot\_state\_transition/dynamic\_constrain.hh" //动力学约束

typedef Eigen::Matrix<int, 6, 1> hexapod\_SupportState; // 支撑状态（容错状态）

typedef Eigen::Matrix<int, Eigen::Dynamic, 6> hexapod\_SupportState\_List; // 支撑状态集合（容错腿状态集合）

// 备选落足点集合

struct Footholds

{

std::vector<Vector3, Eigen::aligned\_allocator<Vector3>> leg[6];

};

namespace planning

{

extern const hexapod\_SupportState\_List initialSupportList; // 42个支撑状态构成集合

//根据当前废腿信息确定下一步的支撑腿状态集合

hexapod\_SupportState\_List findAvailable\_SupportStates(const hit\_spider::hexapod\_State &hexapodState);

hexapod\_SupportState\_List findAvailable\_SupportStates\_and\_maxLength(const hit\_spider::hexapod\_State &hexapodState, std::vector<double> &max\_length);

// Now到Next的步长计算

// 考虑静态稳定性约束的步长

double get\_CWC\_base\_length(const hit\_spider::hexapod\_State &hexapodState);

// 考虑运动学约束的步长

double get\_kin\_length(const hit\_spider::hexapod\_State &hexapodState);

// 考虑动力学约束的步长

double get\_dynamic\_length(const hit\_spider::hexapod\_State &hexapodState);

// Next摆动腿可行落足点集合

Footholds getAvailableFootholds(const hit\_spider::hexapod\_State &hexapodState);

// 专家法选择摆动腿落足点

void swingLegFoot\_position\_Expert(hit\_spider::hexapod\_State &hexapodState);

// 专家法计算下一个state

void get\_nextState\_Expert(hit\_spider::hexapod\_State &hexapodState);

// 随机法计算下一个state

void get\_nextState\_Rondom(hit\_spider::hexapod\_State &hexapodState);

// 确定备选状态集——专家法（专家法确定步长、摆动腿落足点）

std::vector<hit\_spider::hexapod\_State> get\_NextState\_list(const hit\_spider::hexapod\_State &hexapodState);

}

#endif

#### trajectory.hh

// 接触序列规划完成后，规划足端轨迹、机体轨迹

#ifndef MCTS\_TRAJECTORY\_HH

#define MCTS\_TRAJECTORY\_HH

#include "hit\_spider/util.hh"

namespace trajectory

{

// 输入六足机器人初末状态，计算足端轨迹的六次多项式系数

Eigen::Matrix<float, 7, 18> solution\_leg(const hit\_spider::hexapod\_State &state1, const hit\_spider::hexapod\_State &state2);

// 输入六足机器人初末状态，计算机体质心轨迹的五次多项式系数

Eigen::Matrix<float, 6, 3> solution\_body(const hit\_spider::hexapod\_Base\_Pose &robotNow, const hit\_spider::hexapod\_Base\_Pose &robotNext);

// 输入六足机器人初末状态，计算机体姿态轨迹的五次多项式系数

Eigen::Matrix<float, 6, 3> solution\_body\_rotation(const hit\_spider::hexapod\_Base\_Pose &robotNow, const hit\_spider::hexapod\_Base\_Pose &robotNext);

// 根据时间计算足端具体位置

void leg\_assignment(const hit\_spider::hexapod\_State &state, hit\_spider::FeetPosition &feetNow, const Eigen::Matrix<float, 7, 18> &solution, const float t);

// 根据时间计算具体的机体位姿

void body\_assignment(hit\_spider::hexapod\_Base\_Pose &robotNext, const Eigen::Matrix<float, 6, 3> &solution, const Eigen::Matrix<float, 6, 3> &solutionbodyRotate, const float t);

}

#endif

### 源文件

#### util.cpp

// util中接口函数的实现、全局变量的定义

#include "hit\_spider/util.hh"

#include <tf2/LinearMath/Quaternion.h>

//------共享全局变量定义------

namespace

{

// 初始化固定的坐标变换FixJi

Eigen::Isometry3d init\_TransMatrix\_FixJi\_Body(const int &num)

{

Eigen::Isometry3d T;

T.matrix().row(0) << cos(body\_fixJi\_theta[num]), sin(body\_fixJi\_theta[num]), 0, -0.4; //-R

T.matrix().row(1) << -sin(body\_fixJi\_theta[num]), cos(body\_fixJi\_theta[num]), 0, 0;

T.matrix().row(2) << 0, 0, 1, 0;

T.matrix().row(3) << 0, 0, 0, 1;

return T;

}

// 初始化固定的坐标变换FixGu

Eigen::Isometry3d init\_TransMatrix\_Body\_2\_fixJi(const int &num)

{

Eigen::Isometry3d T = init\_TransMatrix\_FixJi\_Body(num);

return T.inverse();

}

// 机体base坐标系下六个默认落足点

Vector3 init\_defaultFoothold(const int &num)

{

Vector3 p;

p.x() = 1.08 \* cos(body\_fixJi\_theta[num]);

p.y() = 1.08 \* sin(body\_fixJi\_theta[num]);

p.z() = -0.5;

return p;

}

}

const float body\_fixJi\_theta[6] = {PI \* 2 / 3, PI, PI \* 4 / 3, PI \* 5 / 3, 0, PI \* 1 / 3};

const std::vector<std::string> HEXAPOD\_JOINT\_STATE\_NAME{"joint\_lf\_1", "joint\_lf\_2", "joint\_lf\_3",

"joint\_lm\_1", "joint\_lm\_2", "joint\_lm\_3",

"joint\_lh\_1", "joint\_lh\_2", "joint\_lh\_3",

"joint\_rf\_1", "joint\_rf\_2", "joint\_rf\_3",

"joint\_rm\_1", "joint\_rm\_2", "joint\_rm\_3",

"joint\_rh\_1", "joint\_rh\_2", "joint\_rh\_3"};

const Eigen::Isometry3d TransMatrix\_FixJi\_Body[6] = {init\_TransMatrix\_FixJi\_Body(0), init\_TransMatrix\_FixJi\_Body(1), init\_TransMatrix\_FixJi\_Body(2),

init\_TransMatrix\_FixJi\_Body(3), init\_TransMatrix\_FixJi\_Body(4), init\_TransMatrix\_FixJi\_Body(5)};

const Eigen::Isometry3d TransMatrix\_Body\_2\_fixJi[6] = {init\_TransMatrix\_Body\_2\_fixJi(0), init\_TransMatrix\_Body\_2\_fixJi(1), init\_TransMatrix\_Body\_2\_fixJi(2),

init\_TransMatrix\_Body\_2\_fixJi(3), init\_TransMatrix\_Body\_2\_fixJi(4), init\_TransMatrix\_Body\_2\_fixJi(5)};

const Vector3 Hexapod\_defaultFoothold[6] = {init\_defaultFoothold(0), init\_defaultFoothold(1), init\_defaultFoothold(2),

init\_defaultFoothold(3), init\_defaultFoothold(4), init\_defaultFoothold(5)};

//------常用函数定义------

Matrix3 crossMatrix(const Vector3 &x)

{

Matrix3 res;

res.setZero();

res(0, 1) = -x(2);

res(0, 2) = x(1);

res(1, 0) = x(2);

res(1, 2) = -x(0);

res(2, 0) = -x(1);

res(2, 1) = x(0);

return res;

}

int rand\_customization(int minValue, int maxValue)

{

maxValue = maxValue + 1;

return rand() % (maxValue - minValue) + minValue;

}

Eigen::Isometry3d getTrans\_W\_B(const hit\_spider::hexapod\_Base\_Pose &pose)

{

tf2::Quaternion tf2\_q;

tf2\_q.setEuler(pose.orientation.roll, pose.orientation.pitch, pose.orientation.yaw);

Eigen::Quaterniond q(tf2\_q.getW(), tf2\_q.getX(), tf2\_q.getY(), tf2\_q.getZ());

Vector3 t(pose.position.x, pose.position.y, pose.position.z);

Eigen::Isometry3d T = Eigen::Isometry3d::Identity();

T.rotate(q);

T.pretranslate(t);

return T;

}

geometry\_msgs::Transform getTransform\_W\_B(const hit\_spider::hexapod\_Base\_Pose &pose)

{

geometry\_msgs::Transform trans;

tf2::Quaternion tf2\_q;

tf2\_q.setEuler(pose.orientation.roll, pose.orientation.pitch, pose.orientation.yaw);

trans.rotation.w = tf2\_q.getW();

trans.rotation.x = tf2\_q.getX();

trans.rotation.y = tf2\_q.getY();

trans.rotation.z = tf2\_q.getZ();

trans.translation.x = pose.position.x;

trans.translation.y = pose.position.y;

trans.translation.z = pose.position.z;

return trans;

}

void init\_hexapodState(hit\_spider::hexapod\_State &hexapodState)

{

// 当前机体位姿 与 下一状态机体位姿

hexapodState.base\_Pose\_Now.position.x = hexapodState.base\_Pose\_Now.position.y = 0.0;

hexapodState.base\_Pose\_Now.position.z = 0.5;

hexapodState.base\_Pose\_Now.orientation.roll = hexapodState.base\_Pose\_Now.orientation.yaw = hexapodState.base\_Pose\_Now.orientation.pitch = 0.0;

hexapodState.base\_Pose\_Next = hexapodState.base\_Pose\_Now;

// 六条腿的默认落足点

for (int i = 0; i < 6; ++i)

{

// 当前落足点

hexapodState.feetPositionNow.foot[i].x = Hexapod\_defaultFoothold[i].x();

hexapodState.feetPositionNow.foot[i].y = Hexapod\_defaultFoothold[i].y();

hexapodState.feetPositionNow.foot[i].z = 0;

// 当前支撑状态

hexapodState.support\_State\_Now[i] = SUPPORT;

// 当前容错状态

hexapodState.faultLeg\_State\_Now[i] = NORMAL\_LEG;

}

// 下一步落足点

hexapodState.feetPositionNext = hexapodState.feetPositionNow;

// 下一步支撑状态和容错状态

hexapodState.support\_State\_Next = hexapodState.support\_State\_Now;

hexapodState.faultLeg\_State\_Next = hexapodState.faultLeg\_State\_Now;

hexapodState.move\_Direction.x = 1;

hexapodState.move\_Direction.y = 0;

hexapodState.move\_Direction.z = 0;

}

#### static\_information.cpp

// 静态消息定义

#include "hit\_spider/static\_information.hh"

namespace Static\_Information

{

namespace

{

grid\_map::GridMap init\_grid\_map(const std::string &file\_name)

{

setlocale(LC\_ALL, ""); // 可输出中文

// grid\_map对象,地图大小设置,坐标系设置

grid\_map::GridMap mapData;

mapData.setGeometry(grid\_map::Length(30, 20), 0.1);

mapData.setFrameId("odom");

// 添加图层

mapData.add("elevation"); // 高程图

mapData.add("3D\_feeling", grid\_map::Matrix::Zero(mapData.getSize()(0), mapData.getSize()(1))); // 显示3d感觉的图层

for (grid\_map::GridMapIterator iterator(mapData); !iterator.isPastEnd(); ++iterator)

{

mapData.at("3D\_feeling", \*iterator) = -0.6;

}

// 1.从文件读取地图数据,加入到elevation图层中

std::ifstream fin(file\_name);

std::string line\_info, input\_result;

std::vector<float> vectorData;

if (fin) // 有该文件

{

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;

std::cout << "Find map file: " << file\_name << std::endl;

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;

while (getline(fin, line\_info)) // line中不包括每行的换行符

{

std::stringstream inputDATA(line\_info);

while (inputDATA >> input\_result)

{

vectorData.push\_back(atof(input\_result.c\_str()));

}

}

for (int i = 0; i < (int)vectorData.size(); i += 2)

{

grid\_map::Position gm\_position;

gm\_position << vectorData[i], vectorData[i + 1];

grid\_map::Index gm\_index;

mapData.getIndex(gm\_position, gm\_index);

mapData.at("elevation", gm\_index) = 0;

mapData.at("3D\_feeling", gm\_index) = 0;

// debug

// std::cout << vectorData[i] << "," << vectorData[i + 1] << std::endl; // 输出地形坐标

}

}

else // 没有该文件

{

ROS\_FATAL("没有指定地图文件,请检查路径");

ros::shutdown();

exit(1);

}

// 2.把机器人初始(默认)落足点加入到elevation图层中

for (int i = 0; i < 6; ++i)

{

grid\_map::Index gm\_index;

grid\_map::Position gm\_position;

gm\_position.x() = Hexapod\_defaultFoothold[i].x();

gm\_position.y() = Hexapod\_defaultFoothold[i].y();

mapData.getIndex(gm\_position, gm\_index);

mapData.at("elevation", gm\_index) = 0;

}

// 3.打印消息

ROS\_INFO("创建的grid\_map地图大小为 %d x %d m (%d x %d cells).", (int)mapData.getLength().x(), (int)mapData.getLength().y(), (int)mapData.getSize()(0), (int)mapData.getSize()(1));

return mapData;

}

MatrixX3 init\_Feasible\_foot\_position()

{

MatrixX3 AA;

int num = 0;

for (grid\_map::GridMapIterator it(mapData); !it.isPastEnd(); ++it)

{

grid\_map::Position position;

mapData.getPosition(\*it, position);

if (mapData.at("elevation", \*it) == 0)

{

++num;

}

}

AA.resize(num, 3);

int row\_index = 0;

for (grid\_map::GridMapIterator it(mapData); !it.isPastEnd(); ++it)

{

grid\_map::Position position;

mapData.getPosition(\*it, position);

if (mapData.at("elevation", \*it) == 0)

{

AA.row(row\_index++) << position.x(), position.y(), 0;

}

}

return AA;

}

}

MatrixX3 get\_now\_Feasible\_foot\_position(const hit\_spider::hexapod\_Base\_Pose &pose)

{

MatrixX3 AA;

int num = 0;

grid\_map::Position position;

grid\_map::Position center(pose.position.x, pose.position.y);

double radius = 1.5;

for (grid\_map::CircleIterator it(mapData, center, radius); !it.isPastEnd(); ++it)

{

mapData.getPosition(\*it, position);

if (mapData.at("elevation", \*it) == 0)

{

++num;

}

}

AA.resize(num, 3);

int row\_index = 0;

for (grid\_map::CircleIterator it(mapData, center, radius); !it.isPastEnd(); ++it)

{

grid\_map::Position position;

mapData.getPosition(\*it, position);

if (mapData.at("elevation", \*it) == 0)

{

AA.row(row\_index++) << position.x(), position.y(), 0;

}

}

return AA;

}

const grid\_map::GridMap mapData = init\_grid\_map(terrain\_file\_name);

const MatrixX3 Feasible\_foot\_position = init\_Feasible\_foot\_position();

}

#### static\_information\_publisher.cpp

// 静态信息发布者对象

#include "hit\_spider/static\_information.hh"

int main(int argc, char \*argv[])

{

setlocale(LC\_ALL, "");

ros::init(argc, argv, "static\_information\_publisher");

ros::NodeHandle nh;

//------静态地图相关------

ros::Publisher gridMapPub = nh.advertise<grid\_map\_msgs::GridMap>("grid\_map", 1, true); // 地图锁存住

ROS\_INFO("话题发布的的grid\_map地图大小为 %d x %d m (%d x %d cells).",

(int)Static\_Information::mapData.getLength().x(), (int)Static\_Information::mapData.getLength().y(),

(int)Static\_Information::mapData.getSize()(0), (int)Static\_Information::mapData.getSize()(1));

grid\_map\_msgs::GridMap gm\_message; // 创建grid\_map消息

grid\_map::GridMapRosConverter::toMessage(Static\_Information::mapData, gm\_message); // 把grid\_map地图转换为ros消息格式

gridMapPub.publish(gm\_message);

//------ros回头------

ros::spin();

return 0;

}

#### hexapod.cpp

#include "hit\_spider/hexapod.hh"

#include <functional> //std::bind()函数

//------Class Leg------

Leg::Leg() : legNum(0), theta1(0), theta2(0), theta3(0)

{

}

Leg::~Leg()

{

}

void Leg::set\_num(const int &num)

{

this->legNum = num;

}

void Leg::IK(const hit\_spider::hexapod\_Base\_Pose &Wolrd\_BasePose, const geometry\_msgs::Point &World\_foot)

{

// 计算足端目标机器人坐标系到世界坐标系的旋转平移矩阵T

Eigen::Isometry3d T\_W\_B = getTrans\_W\_B(Wolrd\_BasePose);

// 计算足端到固定基节坐标系下的坐标

Eigen::Vector3d p1\_ = Eigen::Vector3d(World\_foot.x, World\_foot.y, World\_foot.z);

Eigen::Vector3d foot\_FixJi;

foot\_FixJi = TransMatrix\_FixJi\_Body[legNum - 1] \* T\_W\_B.inverse() \* p1\_;

float tmp\_theta1 = atan2(foot\_FixJi.y(), foot\_FixJi.x());

float N = foot\_FixJi.z();

float M = 0.0f;

if (fabs(foot\_FixJi.y()) < 0.00000000001f)

{

M = foot\_FixJi.x() - 0.18f;

}

else

{

M = foot\_FixJi.y() / sin(tmp\_theta1) - 0.18f;

}

if (sqrt(M \* M + N \* N) > 0.5 + 0.5)

{

// debug

std::cout << "foot\_FixJi = " << foot\_FixJi << std::endl;

ROS\_ERROR("out of range,M:%f,N:%f", M, N);

}

float tmp\_acos = acos((M \* M + N \* N) / sqrt(M \* M + N \* N));

float tmp\_theta2 = atan2(N, M) + tmp\_acos;

float tmp\_theta3 = atan2(N - 0.5 \* sin(tmp\_theta2), M - 0.5f \* cos(tmp\_theta2)) - tmp\_theta2;

tmp\_theta1 = -tmp\_theta1;

tmp\_theta2 = -tmp\_theta2;

tmp\_theta3 = tmp\_theta3;

// 更新单腿关节角度

this->theta1 = tmp\_theta1;

this->theta2 = tmp\_theta2;

this->theta3 = tmp\_theta3;

}

//------Class Leg------

//------Class Hexapod------

Hexapod::Hexapod(ros::NodeHandle &nh)

{

this->legs = new Leg[6];

for (int i = 0; i < 6; ++i)

{

this->legs[i].set\_num(i + 1);

}

// joint\_states话题初始化

this->joint\_state\_pub\_ = nh.advertise<sensor\_msgs::JointState>("/joint\_states", 50);

this->joint\_state.name.resize(18);

this->joint\_state.position.resize(18);

for (size\_t i = 0; i < 18; ++i) // 设置18个关节的名字

{

this->joint\_state.name[i] = HEXAPOD\_JOINT\_STATE\_NAME[i];

}

// 设置坐标变换的名字

this->odom\_2\_trunck.header.frame\_id = "odom"; // 基坐标系（世界坐标系）

this->odom\_2\_trunck.child\_frame\_id = "link\_base"; // 子坐标系

odom\_2\_trunck.transform.rotation.w = 1; // 初始化一下

//------发布fixGu相对于fixJi节的静态坐标变换，用于绘图使用，静态坐标变换也需要spin

tf2\_ros::StaticTransformBroadcaster stb;

std::string str\_header1, str\_header2;

str\_header1 = "fixJi\_"; // urdf提供了这个连杆坐标系

str\_header1 = "fixGu\_";

tf2::Quaternion q;

q.setEuler(-PI / 2, 0, 0);

for (int i = 0; i < 6; ++i)

{

geometry\_msgs::TransformStamped tfs;

tfs.header.stamp = ros::Time::now();

tfs.header.frame\_id = str\_header1 + std::to\_string(i + 1);

tfs.child\_frame\_id = str\_header2 + std::to\_string(i + 1);

tfs.transform.rotation.w = q.getW();

tfs.transform.rotation.x = q.getX();

tfs.transform.rotation.y = q.getY();

tfs.transform.rotation.z = q.getZ();

tfs.transform.translation.x = 0.18;

tfs.transform.translation.y = 0;

tfs.transform.translation.z = 0;

stb.sendTransform(tfs);

}

}

Hexapod::~Hexapod()

{

delete[] legs;

}

void Hexapod::IK\_Robot(const hit\_spider::hexapod\_State &p)

{

// 设置odom姿态

this->odom\_2\_trunck.transform = getTransform\_W\_B(p.base\_Pose\_Now);

// 求解单腿逆运动学，设置18个关节的关节角度

for (int i = 0; i < 6; ++i)

{

this->legs[i].IK(p.base\_Pose\_Now, p.feetPositionNow.foot[i]);

}

// 更新发布的关机角度信息

joint\_state.position[0] = -(this->legs[0].theta1); // 第一条腿 == lf

joint\_state.position[1] = -(this->legs[0].theta2);

joint\_state.position[2] = (this->legs[0].theta3) + PI / 2;

joint\_state.position[3] = -(this->legs[1].theta1); // 第二条腿 == lm

joint\_state.position[4] = -(this->legs[1].theta2);

joint\_state.position[5] = (this->legs[1].theta3) + PI / 2;

joint\_state.position[6] = -(this->legs[2].theta1); // 第三条腿 == lh

joint\_state.position[7] = -(this->legs[2].theta2);

joint\_state.position[8] = (this->legs[2].theta3) + PI / 2;

joint\_state.position[9] = -(this->legs[5].theta1); // 第六条腿 == rh

joint\_state.position[10] = -(this->legs[5].theta2);

joint\_state.position[11] = (this->legs[5].theta3) + PI / 2;

joint\_state.position[12] = -(this->legs[4].theta1); // 第五条腿 == rm

joint\_state.position[13] = -(this->legs[4].theta2);

joint\_state.position[14] = (this->legs[4].theta3) + PI / 2;

joint\_state.position[15] = -(this->legs[3].theta1); // 第四条腿 == rm

joint\_state.position[16] = -(this->legs[3].theta2);

joint\_state.position[17] = (this->legs[3].theta3) + PI / 2;

// 广播坐标变换，机器人Trunk新的位姿

this->odom\_2\_trunck.header.stamp = ros::Time::now();

this->odom\_broadcaster.sendTransform(this->odom\_2\_trunck);

// 广播18个关节角度

this->joint\_state.header.stamp = ros::Time::now();

this->joint\_state\_pub\_.publish(this->joint\_state);

}

//------Class Hexapod------

Matrix63 whole\_body\_inverse\_kin(const hit\_spider::hexapod\_Base\_Pose &Wolrd\_BasePose, const hit\_spider::FeetPosition &World\_feet)

{

Matrix63 Joint\_angles;

// 计算足端目标机器人坐标系到世界坐标系的旋转平移矩阵T

Eigen::Isometry3d T\_W\_B = getTrans\_W\_B(Wolrd\_BasePose);

// 六条腿一个一个求解

for (int i = 0; i < 6; ++i)

{

Eigen::Vector3d p1\_ = Eigen::Vector3d(World\_feet.foot[i].x, World\_feet.foot[i].y, World\_feet.foot[i].z);

Eigen::Vector3d foot\_FixJi;

foot\_FixJi = TransMatrix\_FixJi\_Body[i] \* T\_W\_B.inverse() \* p1\_; // fixGu坐标系下描述足端位置

float tmp\_theta1 = atan2(foot\_FixJi.y(), foot\_FixJi.x());

float N = foot\_FixJi.z();

float M = 0.0f;

if (fabs(foot\_FixJi.y()) < 0.00000000001f)

{

M = foot\_FixJi.x() - 0.18f;

}

else

{

M = foot\_FixJi.y() / sin(tmp\_theta1) - 0.18f;

}

if (sqrt(M \* M + N \* N) > 0.5 + 0.5)

{

ROS\_ERROR("foot out of kinematics range,M:%f,N:%f", M, N);

}

float tmp\_acos = acos((M \* M + N \* N) / sqrt(M \* M + N \* N));

float tmp\_theta2 = atan2(N, M) + tmp\_acos;

float tmp\_theta3 = atan2(N - 0.5 \* sin(tmp\_theta2), M - 0.5f \* cos(tmp\_theta2)) - tmp\_theta2;

Joint\_angles(i, 0) = tmp\_theta1;

Joint\_angles(i, 1) = tmp\_theta2;

Joint\_angles(i, 2) = tmp\_theta3 + PI / 2;

}

return Joint\_angles;

}

bool support\_leg\_inverse\_kin(const hit\_spider::hexapod\_Base\_Pose &Wolrd\_BasePose, const hit\_spider::FeetPosition &World\_feet, const std::vector<int> &support\_leg, MatrixX3 &joints\_output)

{

joints\_output.resize(support\_leg.size(), 3);

// 计算足端目标机器人坐标系到世界坐标系的旋转平移矩阵T

Eigen::Isometry3d T\_W\_B = getTrans\_W\_B(Wolrd\_BasePose);

int row\_index = 0;

// 一个腿一个腿求解

for (int i = 0; i < 6; ++i)

{

auto iter = std::find(support\_leg.begin(), support\_leg.end(), i + 1);

if (iter != support\_leg.end())

{

Eigen::Vector3d p1\_ = Eigen::Vector3d(World\_feet.foot[i].x, World\_feet.foot[i].y, World\_feet.foot[i].z);

Eigen::Vector3d foot\_FixJi;

foot\_FixJi = TransMatrix\_FixJi\_Body[i] \* T\_W\_B.inverse() \* p1\_; // fixGu坐标系下描述足端位置

float tmp\_theta1 = atan2(foot\_FixJi.y(), foot\_FixJi.x());

float N = foot\_FixJi.z();

float M = 0.0f;

if (fabs(foot\_FixJi.y()) < 0.00000000001f)

{

M = foot\_FixJi.x() - 0.18f;

}

else

{

M = foot\_FixJi.y() / sin(tmp\_theta1) - 0.18f;

}

if (sqrt(M \* M + N \* N) > 0.5 + 0.5)

{

// debug

// ROS\_WARN("foot out of kinematics range,foot\_FixJi.y = :%f,foot\_FixJi.z = :%f", foot\_FixJi.x(), foot\_FixJi.z());

return false;

}

float tmp\_acos = acos((M \* M + N \* N) / sqrt(M \* M + N \* N));

float tmp\_theta2 = atan2(N, M) + tmp\_acos;

float tmp\_theta3 = atan2(N - 0.5 \* sin(tmp\_theta2), M - 0.5f \* cos(tmp\_theta2)) - tmp\_theta2;

joints\_output(row\_index, 0) = tmp\_theta1;

joints\_output(row\_index, 1) = tmp\_theta2;

joints\_output(row\_index, 2) = tmp\_theta3 + PI / 2;

++row\_index;

}

}

return true;

}

#### solve\_LP\_GLPK.cpp

//GLPK求解LP问题实现

#include "hit\_spider/robot\_state\_transition/solve\_LP\_GLPK.hh"

namespace Robot\_State\_Transition

{

bool solve\_LP\_Bretl\_GLPK(const MatrixXX &A,

const VectorX &b,

const MatrixXX &D,

const VectorX &d,

const VectorX &g,

VectorX &vec\_opt)

{

assert(A.cols() != 0); // 一定要有不等式约束

const unsigned int n\_structural = A.cols(); // GLPK中structural variables（原决策变量）的个数

const unsigned int n\_auxiliary = A.rows() + D.rows(); // GLPK中auxiliary variables（辅助变量）的个数

const unsigned int n\_unequal = A.rows(); // 不等式约束数量

const unsigned int n\_equal = D.rows(); // 等式约束数量

// 判断输入的是否合理

assert(b.rows() == n\_unequal && d.rows() == n\_equal);

assert(D.cols() == n\_structural || D.cols() == 0);

assert(g.rows() == n\_structural || g.rows() == 0);

// ------GLPK问题对象准备------

glp\_prob \*lp; // LP问题对象

lp = glp\_create\_prob(); // 创建LP问题对象

// glp\_set\_prob\_name(lp, "sample"); // 为LP问题对象命名

// LP问题优化的方向

glp\_set\_obj\_dir(lp, GLP\_MIN);

glp\_add\_rows(lp, n\_auxiliary); // 等式约束的数量（辅助变量的数量）

// ------添加决策变量------

glp\_add\_cols(lp, n\_structural); // 添加决策变量

for (size\_t i = 1; i <= n\_structural; ++i)

{

glp\_set\_col\_bnds(lp, i, GLP\_FR, 0, 0); // 无约束

}

// ------创建GLPK的等式约束对应的矩阵------

int ia[1 + n\_auxiliary \* n\_structural]; // 约束矩阵的行元素（原决策变量）

int ja[1 + n\_auxiliary \* n\_structural]; // 约束矩阵的列元素（辅助变量）

double ar[1 + n\_auxiliary \* n\_structural]; // 约束矩阵的值

int id\_row = 1; // 当前处理的约束序号

int id\_col = 1; // 约束对应决策变量序号

int id\_temp = 1; // GLPK数组的总长度

// 原问题的不等式约束矩阵A

for (size\_t i = 0; i < n\_unequal; ++i, ++id\_row)

{

glp\_set\_row\_bnds(lp, id\_row, GLP\_UP, 0, b(i)); // 辅助变量小于等于向量b

id\_col = 1;

for (size\_t j = 0; j < n\_structural; ++j, ++id\_col)

{

if (A(i, j) != 0)

{

ia[id\_temp] = id\_row, ja[id\_temp] = id\_col, ar[id\_temp] = A(i, j);

++id\_temp;

}

}

}

// 原问题的等式约束矩阵D

for (size\_t i = 0; i < n\_equal; ++i, ++id\_row)

{

glp\_set\_row\_bnds(lp, id\_row, GLP\_FX, d(i), d(i)); // 辅助变量小于等于向量b

id\_col = 1;

for (size\_t j = 0; j < n\_structural; ++j, ++id\_col)

{

if (D(i, j) != 0)

{

ia[id\_temp] = id\_row, ja[id\_temp] = id\_col, ar[id\_temp] = D(i, j);

++id\_temp;

}

}

}

// 设置目标值

if (g.rows() != 0)

{

for (size\_t i = 1; i <= n\_structural; ++i)

{

glp\_set\_obj\_coef(lp, i, g(i - 1)); // 决策变量的目标值

}

}

else

{

for (size\_t i = 1; i <= n\_structural; ++i)

{

glp\_set\_obj\_coef(lp, i, 0.0); // 无目标

}

}

glp\_set\_obj\_coef(lp, 0, 0.0); // 设置目标函数的常量值

glp\_load\_matrix(lp, id\_temp - 1, ia, ja, ar); // 获取GLPK约束矩阵

//------求解LP问题------

glp\_smcp opts; // LP求解器控制参数

glp\_init\_smcp(&opts); // 初始化控制参数为默认值

opts.msg\_lev = GLP\_MSG\_OFF; // 控制参数赋值，在求解的时候把这个参数传进去，这里可以还选择GLP\_MSG\_OFF

// debug 写文件

// glp\_write\_lp(lp, NULL, "/home/ptw/now/robot\_state\_transition/haha.lp");

bool output\_flag = false;

glp\_simplex(lp, &opts); // 求解LP问题

int res = glp\_get\_status(lp);

if (res == GLP\_OPT)

{

output\_flag = true;

// 提取最优决策变量

vec\_opt.resize(n\_structural, 1);

for (size\_t i = 1; i <= n\_structural; ++i)

{

vec\_opt(i - 1) = glp\_get\_col\_prim(lp, i);

}

}

else if (res == GLP\_UNBND)

{

output\_flag = false; // 对于Bretl问题不可能出现无穷多解的。

std::cout << "\033[33m\t@@@模型存在无穷多最优解!\t\033[0m" << std::endl;

}

else

{

std::cout << "\033[33m\t@@@There is a problem when solving LP problem (GLPK)!\t\033[0m" << std::endl;

std::cout << "res(GLPK的status) = " << res << std::endl;

output\_flag = false;

}

// 释放GLPK使用的内存

glp\_delete\_prob(lp);

glp\_free\_env();

return output\_flag;

}

bool solve\_LP\_GLPK(const MatrixXX &A,

const VectorX &b,

const MatrixXX &D,

const VectorX &d,

const VectorX &g,

const VectorX &minBounds,

const VectorX &maxBounds,

bool direction\_max,

VectorX \*result\_x,

double \*result\_optimization)

{

assert(A.cols() != 0); // 一定要有不等式约束

const unsigned int n\_structural = A.cols(); // GLPK中structural variables（原决策变量）的个数

const unsigned int n\_auxiliary = A.rows() + D.rows(); // GLPK中auxiliary variables（辅助变量）的个数

const unsigned int n\_unequal = A.rows(); // 不等式约束数量

const unsigned int n\_equal = D.rows(); // 等式约束数量

// 判断输入的是否合理

assert(b.rows() == n\_unequal && d.rows() == n\_equal);

assert(D.cols() == n\_structural || D.cols() == 0);

assert(g.rows() == n\_structural || g.rows() == 0);

assert(minBounds.rows() == n\_structural || minBounds.rows() == 0);

assert(maxBounds.rows() == n\_structural || maxBounds.rows() == 0);

// ------GLPK问题对象准备------

glp\_prob \*lp; // LP问题对象

lp = glp\_create\_prob(); // 创建LP问题对象

// LP问题优化的方向

if (direction\_max)

{

glp\_set\_obj\_dir(lp, GLP\_MAX);

}

else

{

glp\_set\_obj\_dir(lp, GLP\_MIN);

}

glp\_add\_rows(lp, n\_auxiliary); // 等式约束的数量（辅助变量的数量）

// ------添加决策变量------

glp\_add\_cols(lp, n\_structural); // 添加决策变量

if (minBounds.rows() != 0)

{

if (maxBounds.rows() != 0)

{

// 双边约束

for (size\_t i = 1; i <= n\_structural; ++i)

{

glp\_set\_col\_bnds(lp, i, GLP\_DB, minBounds(i - 1), maxBounds(i - 1));

}

}

else

{

// 最小值约束

for (size\_t i = 1; i <= n\_structural; ++i)

{

glp\_set\_col\_bnds(lp, i, GLP\_LO, minBounds(i - 1), 0);

}

}

}

else

{

if (maxBounds.rows() != 0)

{

// 最大值约束

for (size\_t i = 1; i <= n\_structural; ++i)

{

glp\_set\_col\_bnds(lp, i, GLP\_UP, 0, maxBounds(i - 1));

}

}

else

{

// 无约束

for (size\_t i = 1; i <= n\_structural; ++i)

{

glp\_set\_col\_bnds(lp, i, GLP\_FR, 0, 0);

}

}

}

// ------创建GLPK的等式约束对应的矩阵------

int ia[1 + n\_auxiliary \* n\_structural]; // 约束矩阵的行元素（原决策变量）

int ja[1 + n\_auxiliary \* n\_structural]; // 约束矩阵的列元素（辅助变量）

double ar[1 + n\_auxiliary \* n\_structural]; // 约束矩阵的值

int id\_row = 1; // 当前处理的约束序号

int id\_col = 1; // 约束对应决策变量序号

int id\_temp = 1; // GLPK数组的总长度

// 原问题的不等式约束矩阵A

for (size\_t i = 0; i < n\_unequal; ++i, ++id\_row)

{

glp\_set\_row\_bnds(lp, id\_row, GLP\_UP, 0, b(i)); // 辅助变量小于等于向量b

id\_col = 1;

for (size\_t j = 0; j < n\_structural; ++j, ++id\_col)

{

if (A(i, j) != 0)

{

ia[id\_temp] = id\_row, ja[id\_temp] = id\_col, ar[id\_temp] = A(i, j);

++id\_temp;

}

}

}

// 原问题的等式约束矩阵D

for (size\_t i = 0; i < n\_equal; ++i, ++id\_row)

{

glp\_set\_row\_bnds(lp, id\_row, GLP\_FX, d(i), d(i)); // 辅助变量小于等于向量b

id\_col = 1;

for (size\_t j = 0; j < n\_structural; ++j, ++id\_col)

{

if (D(i, j) != 0)

{

ia[id\_temp] = id\_row, ja[id\_temp] = id\_col, ar[id\_temp] = D(i, j);

++id\_temp;

}

}

}

// 设置目标值

if (g.rows() != 0)

{

for (size\_t i = 1; i <= n\_structural; ++i)

{

glp\_set\_obj\_coef(lp, i, g(i - 1)); // 决策变量的目标值

}

}

else

{

for (size\_t i = 1; i <= n\_structural; ++i)

{

glp\_set\_obj\_coef(lp, i, 0.0); // 无目标

}

}

glp\_set\_obj\_coef(lp, 0, 0.0); // 设置目标函数的常量值

glp\_load\_matrix(lp, id\_temp - 1, ia, ja, ar); // 获取GLPK约束矩阵

//------求解LP问题------

glp\_smcp opts; // LP求解器控制参数

glp\_init\_smcp(&opts); // 初始化控制参数为默认值

opts.msg\_lev = GLP\_MSG\_OFF; // 控制参数赋值，在求解的时候把这个参数传进去，这里可以还选择GLP\_MSG\_OFF

glp\_simplex(lp, &opts); // 求解LP问题

bool output\_flag = false;

int res = glp\_get\_status(lp);

if (res == GLP\_OPT)

{

output\_flag = true;

if (result\_x != nullptr) // 提取最优决策变量

{

(\*result\_x).resize(n\_structural, 1);

for (size\_t i = 1; i <= n\_structural; ++i)

{

(\*result\_x)(i - 1) = glp\_get\_col\_prim(lp, i);

}

}

if (result\_optimization != nullptr) // 提取最优值

{

\*result\_optimization = glp\_get\_obj\_val(lp);

}

}

else if (res == GLP\_UNBND)

{

output\_flag = true;

//debug

// std::cout << "\033[33m\t模型有无穷最优解\t\033[0m" << std::endl;

if (result\_optimization != nullptr) // 提取最优值，无法提取最优决策变量

{

\*result\_optimization = glp\_get\_obj\_val(lp);

}

}

else

{

// std::cout << "\033[33m\t@@@ (GLPK) There is a problem when solving LP problem | LP模型无解 (GLPK)!\t\033[0m" << std::endl;

// std::cout << "res(GLPK的status) = " << res << std::endl;

// glp\_write\_lp(lp, NULL, "/home/ptw/dynamic\_MCTS/haha.lp");

output\_flag = false;

}

// 释放GLPK使用的内存

glp\_delete\_prob(lp);

glp\_free\_env();

return output\_flag;

}

}

#### Bretl.cpp

// 满足线性摩擦锥约束的空间向平面的投影的具体实现

#include "hit\_spider/robot\_state\_transition/Bretl.hh"

namespace Robot\_State\_Transition

{

// Vertex\_Planar部分

Vertex\_Planar::Vertex\_Planar()

{

this->point.setZero();

next = nullptr;

expanded = false;

}

Vertex\_Planar::Vertex\_Planar(point\_Planar point\_input)

{

this->point = point\_input;

next = nullptr;

expanded = false;

}

Vertex\_Planar \*Vertex\_Planar::expand(VectorX &g,

const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d)

{

// 优化方向

// v1 = this

Vertex\_Planar \*v2 = this->next;

point\_Planar direction;

direction(0) = (v2->point)(1) - (this->point)(1); // 拓展方向(y,-x)垂直于edge

direction(1) = this->point(0) - v2->point(0);

direction.normalize();

// 进行优化

point\_Planar point\_boundary;

bool flag = optimize\_direction(direction, g, A, b, D, d, point\_boundary);

if (!flag)

{

this->expanded = true;

return nullptr;

}

// 是否添加添加新得到的边界点

point\_Planar p1 = point\_boundary - this->point;

point\_Planar p2 = v2->point - this->point;

double dis\_expand = abs(p1(0) \* p2(1) - p1(1) \* p2(0));

if (dis\_expand < Threshold\_Expand)

{

this->expanded = true;

return nullptr;

}

else

{

Vertex\_Planar \*vnew = new Vertex\_Planar(point\_boundary);

// vnew连接原来的两个点

vnew->next = this->next;

this->next = vnew;

this->expanded = false;

return vnew;

}

}

// Polygon\_Planar部分

Polygon\_Planar::Polygon\_Planar(Vertex\_Planar \*v1, Vertex\_Planar \*v2, Vertex\_Planar \*v3)

{

v1->next = v2;

v2->next = v3;

v3->next = v1;

vertices.push\_back(v1);

vertices.push\_back(v2);

vertices.push\_back(v3);

}

Polygon\_Planar::Polygon\_Planar(const Vertex\_Planar &v1, const Vertex\_Planar &v2, const Vertex\_Planar &v3)

{

Vertex\_Planar \*pv1 = new Vertex\_Planar(v1);

Vertex\_Planar \*pv2 = new Vertex\_Planar(v2);

Vertex\_Planar \*pv3 = new Vertex\_Planar(v3);

pv1->next = pv2;

pv2->next = pv3;

pv3->next = pv1;

this->vertices.push\_back(pv1);

this->vertices.push\_back(pv2);

this->vertices.push\_back(pv3);

}

Polygon\_Planar::~Polygon\_Planar()

{

for (auto iter = this->vertices.begin(); iter != this->vertices.end(); ++iter)

{

delete \*iter;

}

}

bool Polygon\_Planar::all\_expanded() const

{

for (auto iter = vertices.begin(); iter != vertices.end(); ++iter)

{

if (!(\*iter)->expanded)

{

return false;

}

}

return true;

}

void Polygon\_Planar::iter\_expand(VectorX g,

const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d,

const int &max\_iter)

{

int nb\_iter = 0;

Vertex\_Planar \*v = this->vertices.at(0);

while (!(this->all\_expanded()) && nb\_iter < max\_iter)

{

if (v->expanded)

{

v = v->next;

continue;

}

Vertex\_Planar \*vnew = v->expand(g, A, b, D, d);

if (vnew == nullptr)

{

continue;

}

this->vertices.push\_back(vnew);

++nb\_iter;

}

}

void Polygon\_Planar::sort\_vertices()

{

double minsd = 1e10;

auto ibottom = this->vertices.cbegin();

Vertex\_Planar \*v;

for (auto iter = this->vertices.cbegin(); iter != this->vertices.cend(); ++iter)

{

v = \*iter;

if (v->point(1) + v->next->point(1) < minsd)

{

ibottom = iter;

minsd = v->point(1) + v->next->point(1);

}

}

std::vector<Vertex\_Planar \*> newVertices;

v = \*ibottom;

do

{

newVertices.push\_back(v);

v = v->next;

} while (v != \*ibottom);

std::reverse(newVertices.begin(), newVertices.end()); // 反序

newVertices.insert(newVertices.begin(), \*(newVertices.end() - 1));

newVertices.pop\_back();

this->vertices = newVertices;

}

MatrixXX Polygon\_Planar::export\_vertices(double min\_dist) const

{

std::vector<Vertex\_Planar \*> newVertices;

newVertices.push\_back(\*(this->vertices.begin()));

for (auto iter = (this->vertices.cbegin() + 1); iter != (this->vertices.cend() - 1); ++iter)

{

Vertex\_Planar \*vcur = \*iter;

Vertex\_Planar \*vlast = \*(newVertices.end() - 1);

if ((vcur->point - vlast->point).norm() > min\_dist)

{

newVertices.push\_back(vcur);

}

}

newVertices.push\_back(\*(this->vertices.end() - 1));

// 转换为矩阵

MatrixXX output\_matrix;

output\_matrix.resize(2, newVertices.size());

int num\_col = 0;

for (auto iter = newVertices.begin(); iter != newVertices.end(); ++iter, ++num\_col)

{

output\_matrix.col(num\_col) = (\*iter)->point;

}

return output\_matrix;

}

// 这里面会改变优化目标g

bool optimize\_direction(const point\_Planar &vdir, VectorX &g,

const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d,

point\_Planar &result\_opt)

{

int g\_rows = g.rows();

g(g\_rows - 2) = -vdir(0); // x

g(g\_rows - 1) = -vdir(1); // y

VectorX vec\_opt;

bool flag = solve\_LP\_Bretl\_GLPK(A, b, D, d, g, vec\_opt);

if (flag)

{

result\_opt << vec\_opt(g\_rows - 2), vec\_opt(g\_rows - 1);

}

return flag;

}

bool optimize\_angle(const double theta, VectorX &g,

const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d,

point\_Planar &result\_opt)

{

point\_Planar vdir;

vdir(0) = cos(theta);

vdir(1) = sin(theta);

bool flag = optimize\_direction(vdir, g, A, b, D, d, result\_opt);

return flag;

}

Polygon\_Planar compute\_polygon(VectorX &g,

const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d,

double \*init\_angle,

int max\_iter)

{

bool flag\_angle = false;

if (init\_angle == nullptr)

{

flag\_angle = true;

init\_angle = new double((double)rand() / RAND\_MAX \* PI);

}

double theta = \*init\_angle;

double step\_angle = (double)2 / 3 \* PI;

MatrixXX init\_vertices;

init\_vertices.resize(2, 3); // 三个初始点

int init\_num = 0;

point\_Planar point;

// 第一个初始点

while (true)

{

bool flag = optimize\_angle(theta, g, A, b, D, d, point);

if (flag)

{

init\_vertices.col(init\_num++) = point;

break;

}

}

// debug

// std::cout << init\_vertices << std::endl;

// 另外两个初始点

while (init\_num < 3 && max\_iter > 0)

{

theta += step\_angle;

if (theta >= 2 \* PI)

{

step\_angle \*= 0.25 + 0.5 \* (double)random() / RAND\_MAX;

theta += step\_angle - 2 \* PI;

}

bool flag = optimize\_angle(theta, g, A, b, D, d, point);

if (flag)

{

// debug

// std::cout << "优化出的点为:\n"

// << point << std::endl;

// 防止初始点之间距离过小

if (init\_num == 1)

{

if ((point - init\_vertices.col(0)).norm() < Threshold\_Init)

{

continue;

}

init\_vertices.col(init\_num++) = point;

// debug

// std::cout << init\_vertices << std::endl;

--max\_iter;

}

else if (init\_num == 2)

{

if ((point - init\_vertices.col(0)).norm() < Threshold\_Init || (point - init\_vertices.col(1)).norm() < Threshold\_Init)

{

continue;

}

init\_vertices.col(init\_num++) = point;

// debug

// std::cout << init\_vertices << std::endl;

--max\_iter;

}

}

}

assert(init\_num == 3);

Vertex\_Planar \*v1 = new Vertex\_Planar(init\_vertices.col(0));

Vertex\_Planar \*v2 = new Vertex\_Planar(init\_vertices.col(1));

Vertex\_Planar \*v3 = new Vertex\_Planar(init\_vertices.col(2));

Polygon\_Planar polygon(v1, v2, v3);

polygon.iter\_expand(g, A, b, D, d, max\_iter);

if (flag\_angle)

{

delete init\_angle;

}

return polygon;

}

//------

MatrixXX project\_polytope\_bretl(const MatrixXX &E, const VectorX &f,

const MatrixXX &A, const VectorX &b,

const MatrixXX &D, const VectorX &d,

double \*init\_angle,

int max\_iter, const double max\_radius)

{

// 投影到二维平面上

assert(E.rows() == 2 && f.rows() == 2);

// 加入范围不等式约束

// Inequality constraints: A\_ext \* [ x u v ] <= b\_ext iff

// (1) A \* x <= b and (2) |u|, |v| <= max\_radius

MatrixXX A\_ext;

const int rows\_A = A.rows(); // 需要是const才能在下面使用

const int cols\_A = A.cols();

A\_ext.resize(rows\_A + 4, cols\_A + 2);

A\_ext.setZero();

A\_ext.block(0, 0, rows\_A, cols\_A) = A;

A\_ext(rows\_A, cols\_A) = 1;

A\_ext(rows\_A + 1, cols\_A) = -1;

A\_ext(rows\_A + 2, cols\_A + 1) = 1;

A\_ext(rows\_A + 3, cols\_A + 1) = -1;

VectorX b\_ext;

b\_ext.resize(rows\_A + 4);

b\_ext.setZero();

b\_ext.block(0, 0, rows\_A, 1) = b;

b\_ext.block(rows\_A, 0, 4, 1).array() = max\_radius;

// 加入等式约束

// Equality constraints: C\_ext \* [ x u v ] == d\_ext iff

// (1) C \* x == d and (2) [ u v ] == E \* x + f

MatrixXX D\_ext;

const int rows\_D = D.rows();

const int cols\_D = D.cols();

D\_ext.resize(rows\_D + 2, cols\_D + 2);

D\_ext.setZero();

D\_ext.block(0, 0, rows\_D, cols\_D) = D;

D\_ext.block(rows\_D, 0, 2, cols\_D) = E;

D\_ext.block(rows\_D, cols\_D, 2, 2) = -1 \* Eigen::Matrix<double, 2, 2>::Identity();

VectorX d\_ext;

d\_ext.resize(rows\_D + 2, 1);

d\_ext.setZero();

d\_ext.block(0, 0, rows\_D, 1) = d;

d\_ext.block(rows\_D, 0, 2, 1) = -1 \* f;

VectorX g;

g.resize(cols\_A + 2, 1);

g.setZero();

// // debug

// std::cout << "g = \n"

// << g << std::endl;

// std::cout << "A\_ext = \n"

// << A\_ext << std::endl;

// std::cout << "b\_ext = \n"

// << b\_ext << std::endl;

// std::cout << "D\_ext = \n"

// << D\_ext << std::endl;

// std::cout << "d\_ext = \n"

// << d\_ext << std::endl;

Polygon\_Planar polygon = compute\_polygon(g, A\_ext, b\_ext, D\_ext, d\_ext, init\_angle, max\_iter);

polygon.sort\_vertices();

MatrixXX point\_set = polygon.export\_vertices();

return point\_set;

}

MatrixXX comput\_friction\_region(const MatrixX3 &contactPoints, const MatrixX3 &contactNormals\_input, const double frictionCoefficient, const double &m)

{

double g = 9.81;

// 接触点数量、法向量数量需要保持一致

assert(contactPoints.rows() == contactNormals\_input.rows());

MatrixX3 contactNormals = contactNormals\_input.rowwise().normalized(); // 支撑点法向量归一化

const int contact\_Num = contactPoints.rows();

const int cone\_edge\_number = 4;

const double delta\_theta = 2 \* (double)M\_PI / cone\_edge\_number; // 线性化摩擦锥需要的变化角度，默认是pi/2

// CWC计算

//------一个接触点一个接触点进行计算------

// 变量定义参考质心动力学方程

MatrixXX A1(6, 3 \* contact\_Num); // 质心动力学约束矩阵

A1.setZero();

VectorX u(6, 1);

u.setZero();

u(2, 0) = m \* g; // 重力

MatrixXX B(4 \* contact\_Num, 3 \* contact\_Num); // 线性摩擦锥约束矩阵

B.setZero();

MatrixXX A1\_temp(6, 3); // A = [I3;Skew\_symmetric\_Matrix(接触点p)]

A1\_temp.setZero();

A1\_temp.block<3, 3>(0, 0).setIdentity();

MatrixXX B\_temp(3, 4);

for (int i = 0; i < contact\_Num; ++i)

{

Vector3 N = contactNormals.row(i);

A1\_temp.block<3, 3>(3, 0) = crossMatrix(contactPoints.row(i).transpose());

// 计算与法向量垂直的切向量

Vector3 T1, T2;

T1 = N.cross(Vector3::UnitY());

if (T1.norm() < 1e-5)

{

T1 = N.cross(Vector3::UnitX());

}

T2 = N.transpose().cross(T1);

T1.normalize();

T2.normalize();

// 四棱锥的线性约束

B\_temp.col(0) = T1 - frictionCoefficient \* N;

B\_temp.col(1) = T2 - frictionCoefficient \* N;

B\_temp.col(2) = -1 \* T1 - frictionCoefficient \* N;

B\_temp.col(3) = -1 \* T2 - frictionCoefficient \* N;

A1.block(0, 3 \* i, 6, 3) = A1\_temp;

B.block(4 \* i, 3 \* i, 4, 3) = B\_temp.transpose();

}

// c\_xy = Ex + f

MatrixXX E(2, 3 \* contact\_Num);

VectorX f(2);

E.row(0) = A1.row(4) / (-1 \* m \* g); // x坐标

E.row(1) = A1.row(3) / (m \* g); // y坐标

f.setZero();

// Ax == b

MatrixXX A(4, 3 \* contact\_Num);

VectorX b(4);

A.block(0, 0, 3, 3 \* contact\_Num) = A1.block(0, 0, 3, 3 \* contact\_Num);

A.block(3, 0, 1, 3 \* contact\_Num) = A1.block(5, 0, 1, 3 \* contact\_Num);

b.setZero();

b(2) = m \* g;

// Dx <= d

// MatrixXX D = B

VectorX d(4 \* contact\_Num);

d.setZero();

srand(10); // 确保初始init\_angle的角度相同，可以复现实验

return (project\_polytope\_bretl(E, f, B, d, A, b)).transpose();

}

}

#### my\_cdd.cpp

//双线性变换接口实现

#include "hit\_spider/robot\_state\_transition/my\_cdd.hh"

namespace Robot\_State\_Transition

{

// 顶点转换为不等式

bool vertices\_to\_H(const MatrixXX &vertices\_set, MatrixXX &H\_output, VectorX &h\_output)

{

//------初始化------

dd\_set\_global\_constants();

dd\_debug = false;

//------把vertices转换为cdd矩阵------

dd\_MatrixPtr Matrix\_dd = dd\_CreateMatrix((vertices\_set.rows()), (vertices\_set.cols() + 1));

Matrix\_dd->representation = dd\_Generator; // V表示法

Matrix\_dd->numbtype = dd\_Real; // 数据类型为real

for (dd\_rowrange i = 0; i < vertices\_set.rows(); ++i)

{

dd\_set\_d(Matrix\_dd->matrix[i][0], 1); // 1表示V-representation中的控制点集合，0表示ray射线集合

for (dd\_rowrange j = 1; j < vertices\_set.cols() + 1; ++j)

{

dd\_set\_d(Matrix\_dd->matrix[i][j], vertices\_set(i, j - 1));

}

}

// 把V-representation（cdd矩阵表示）转换为H-representation

dd\_ErrorType err = dd\_NoError;

dd\_PolyhedraPtr poly = dd\_DDMatrix2Poly(Matrix\_dd, &err); // cdd矩阵转换为多面体

if (err != dd\_NoError)

{

//debug

// dd\_WriteErrorMessages(stdout, err);

dd\_free\_global\_constants(); // 释放内存

return false;

}

dd\_MatrixPtr H;

H = dd\_CopyInequalities(poly);

// 把cdd数据类型转换为Eigen矩阵

// 获取Hx<=h中哪行是“等于号”

std::vector<long> eq\_rows;

for (long elem = 1; elem <= (long)(H->linset[0]); ++elem)

{

if (set\_member(elem, H->linset))

eq\_rows.push\_back(elem);

}

int rowsize = (int)H->rowsize;

H\_output.resize(rowsize + eq\_rows.size(), (int)H->colsize - 1);

h\_output.resize(rowsize + eq\_rows.size());

for (int i = 0; i < rowsize; ++i)

{

h\_output(i) = (double)(\*(H->matrix[i][0]));

for (int j = 1; j < H->colsize; ++j)

H\_output(i, j - 1) = -(double)(\*(H->matrix[i][j]));

}

int i = 0;

for (std::vector<long int>::const\_iterator cit = eq\_rows.begin(); cit != eq\_rows.end(); ++cit, ++i)

{

h\_output(rowsize + i) = -h\_output((int)((\*cit) - 1));

H\_output.row(rowsize + i) = -H\_output.row((int)((\*cit) - 1));

}

// 释放内存

dd\_FreeMatrix(H);

dd\_free\_global\_constants();

return true;

}

}

#### kinematics\_constrain.cpp

//六足机器人运动学约束的实现

#include "hit\_spider/robot\_state\_transition/kinematics\_constrain.hh"

#include "hit\_spider/hexapod.hh"

namespace Robot\_State\_Transition

{

namespace

{

/\*\*

\* @ :默单腿fixJifoot约束foot的约束矩阵

\* @description:

\* @return {\*}

\*/

MatrixXX init\_A\_Ji\_foot()

{

MatrixXX A(7, 3);

A.setZero();

A(0, 2) = 1;

A(1, 0) = -1;

A(1, 1) = 1.428148006742115;

A(2, 0) = -1;

A(2, 1) = -1.428148006742115;

A(3, 0) = 3.171594802363213;

A(3, 1) = -1;

A(3, 2) = -25.769207769201120;

A(4, 0) = 3.171594802363213;

A(4, 1) = 1;

A(4, 2) = -25.769207769201120;

A(5, 0) = 3.171594802363213;

A(5, 1) = 1;

A(5, 2) = -2.634036361284702;

A(6, 0) = 3.171594802363213;

A(6, 1) = -1;

A(6, 2) = -2.634036361284702;

return A;

}

/\*\*

\* @ :默单腿fixJifoot约束foot的右边值

\* @description:

\* @return {\*}

\*/

VectorX init\_b\_Ji\_foot()

{

VectorX b(7);

b.setZero();

b(0) = -0.28;

b(3) = 24.480747380741064;

b(4) = 24.480747380741064;

b(5) = 4.353148255853780;

b(6) = 4.353148255853780;

return b;

}

/\*\*

\* @ :默单腿foot约束fixJi的约束矩阵

\* @description:

\* @return {\*}

\*/

MatrixXX init\_A\_foot\_Ji()

{

MatrixXX A(7, 3);

A.setZero();

A(0, 2) = 1;

A(1, 0) = 1.428148006742115;

A(1, 1) = 1;

A(2, 0) = -1.428148006742115;

A(2, 1) = 1;

A(3, 0) = 1;

A(3, 1) = -3.171594802363213;

A(3, 2) = -25.769207769201120;

A(4, 0) = -1;

A(4, 1) = -3.171594802363213;

A(4, 2) = -25.769207769201120;

A(5, 0) = 1;

A(5, 1) = -3.171594802363213;

A(5, 2) = -2.634036361284702;

A(6, 0) = -1;

A(6, 1) = -3.171594802363213;

A(6, 2) = -2.634036361284702;

return A;

}

/\*\*

\* @ : 单腿foot约束fixJi的右边值

\* @description:

\* @return {\*}

\*/

VectorX init\_b\_foot\_Ji()

{

VectorX b(7);

b.setZero();

b(0) = -0.28;

b(3) = 24.480747380741064;

b(4) = 24.480747380741064;

b(5) = 4.353148255853780;

b(6) = 4.353148255853780;

return b;

}

}

const MatrixXX A\_Ji\_foot = init\_A\_Ji\_foot();

const VectorX b\_Ji\_foot = init\_b\_Ji\_foot();

const MatrixXX A\_foot\_Ji = init\_A\_foot\_Ji();

const VectorX b\_foot\_Ji = init\_b\_foot\_Ji();

std::pair<MatrixXX, VectorX> get\_kinematics\_con\_cog\_foot(const hit\_spider::hexapod\_Base\_Pose &base\_pose)

{

MatrixXX A\_output(42, 3);

VectorX b\_output(42);

A\_output.setZero();

b\_output.setZero();

// base描述world的齐次变换矩阵

Eigen::Isometry3d T\_world\_base = getTrans\_W\_B(base\_pose);

Matrix3 base\_world\_R = T\_world\_base.matrix().block(0, 0, 3, 3).transpose();

Vector3 base\_world\_p = -base\_world\_R \* T\_world\_base.matrix().block(0, 3, 3, 1);

for (int i = 0; i < 6; ++i)

{

// fixJi坐标系描述base

Matrix3 Ji\_base\_R = TransMatrix\_FixJi\_Body[i].matrix().block(0, 0, 3, 3);

Vector3 Ji\_base\_p = TransMatrix\_FixJi\_Body[i].matrix().block(0, 3, 3, 1);

A\_output.block(7 \* i, 0, 7, 3) = A\_Ji\_foot \* Ji\_base\_R \* base\_world\_R;

b\_output.block(7 \* i, 0, 7, 1) = b\_Ji\_foot - A\_Ji\_foot \* (Ji\_base\_R \* base\_world\_p + Ji\_base\_p);

}

return std::make\_pair(A\_output, b\_output);

}

std::pair<MatrixXX, VectorX> get\_kinematics\_con\_foot\_cog(const hit\_spider::hexapod\_Base\_Pose &base\_pose,const std::vector<int> &support\_leg, const MatrixX3 &contactPoints, const MatrixX3 &contactNormals)

{

assert(((int)support\_leg.size() == (int)contactPoints.rows()) && ((int)support\_leg.size() == (int)contactNormals.rows()));

MatrixX3 A\_output(7 \* support\_leg.size(), 3);

VectorX b\_output(7 \* support\_leg.size());

A\_output.setZero();

b\_output.setZero();

// 世界系下的机器人下一步位姿，用于判断下一步fixJi的位置

Eigen::Isometry3d T\_W\_B = getTrans\_W\_B(base\_pose);

const double base\_yaw = base\_pose.orientation.yaw;

for (int i = 0; i < (int)support\_leg.size(); ++i)

{

// 计算foot固定坐标系

Matrix3 R\_W\_foot;

R\_W\_foot.col(2) = -1 \* contactNormals.row(i);

R\_W\_foot.col(0) << cos(body\_fixJi\_theta[support\_leg[i] - 1] + base\_yaw), sin(body\_fixJi\_theta[support\_leg[i] - 1] + base\_yaw), 0;

R\_W\_foot.col(0) << -1 \* R\_W\_foot.col(0).cross(Vector3(0, 0, 1));

R\_W\_foot.col(1) = R\_W\_foot.col(2).cross(R\_W\_foot.col(0));

R\_W\_foot.colwise().normalize(); // 归一化

Matrix3 R\_foot\_W = R\_W\_foot.transpose();

Vector3 foot\_W\_p = -1 \* R\_foot\_W \* contactPoints.row(i).transpose();

// 获取base坐标系下质心到fixJi的偏移

Vector3 base\_fixJi\_translation = TransMatrix\_Body\_2\_fixJi[support\_leg[i] - 1].matrix().block<3, 1>(0, 3);

// 转换为世界坐标系下质心到fixJi的偏移（只有旋转矩阵作用，没有平移！）

Vector3 W\_base\_fixJi\_translation = T\_W\_B.matrix().block<3, 3>(0, 0) \* base\_fixJi\_translation;

A\_output.block(7 \* i, 0, 7, 3) = A\_foot\_Ji \* R\_foot\_W;

b\_output.block(7 \* i, 0, 7, 1) = b\_foot\_Ji - A\_foot\_Ji \* R\_foot\_W \* W\_base\_fixJi\_translation - A\_foot\_Ji \* foot\_W\_p;

}

return std::make\_pair(A\_output, b\_output);

}

}

#### dynamic\_constrain.cpp

// 动力学约束

#include "hit\_spider/robot\_state\_transition/dynamic\_constrain.hh"

namespace Robot\_State\_Transition

{

namespace

{

Vector3 init\_max\_torque()

{

Vector3 max\_T;

max\_T(0) = 0.286 \* 2240; // 根关节扭矩：电机额定转矩 \* 减速比

max\_T(1) = 0.747 \* 2240; // 根关节扭矩：电机额定转矩 \* 减速比

max\_T(2) = 0.286 \* 4640; // 根关节扭矩：电机额定转矩 \* 减速比

return max\_T;

}

/\*\*

\* @ : 获取雅克比矩阵

\* @description:

\* @param {double} &t：关节角度构成的向量

\* @return {\*}

\*/

Eigen::Matrix<double, 3, 3> get\_Jacobian(const Vector3 &t)

{

Eigen::Matrix<double, 3, 3> J;

const double t1 = t(0);

const double t2 = t(1);

const double t3 = t(2);

J << -sin(t1) \* (sin(t2 + t3) / (double)2.0 + cos(t2) / (double)2.0 + 0.18),

cos(t1) \* (cos(t2 + t3) / (double)2.0 - sin(t2) / (double)2.0), (cos(t2 + t3) \* cos(t1)) / (double)2.0,

cos(t1) \* (sin(t2 + t3) / (double)2.0 + cos(t2) / (double)2.0 + 0.18), sin(t1) \* (cos(t2 + t3) / (double)2.0 - sin(t2) / (double)2.0), (cos(t2 + t3) \* sin(t1)) / (double)2.0,

0, sin(t2 + t3) / (double)2.0 + cos(t2) / (double)2.0, sin(t2 + t3) / (double)2.0;

return J;

}

/\*\*

\* @ : 单腿动力学中的重力加速度项

\* @description:

\* @param {Vector3} &t：关节角度

\* @return {\*}

\*/

Vector3 get\_legMss\_consume\_Torque(const Vector3 &t)

{

Vector3 Leg\_T;

// const double theta1 = t(0);

const double theta2 = t(1);

const double theta3 = t(2);

Leg\_T(0) = 0;

Leg\_T(1) = (629153.0 \* 9.81 \* cos(theta2 + theta3)) / (double)62500.0 + (37009.0 \* 9.81 \* sin(theta2 + theta3)) / (double)20000.0 - (869173.0 \* 9.81 \* cos(theta2)) / (double)1000000.0;

Leg\_T(2) = (37009.0 \* 9.81 \* sin(theta2 + theta3)) / (double)20000.0;

return Leg\_T;

}

}

const double Mass = 121.8957;

const Vector3 MaxTorque = init\_max\_torque();

std::pair<MatrixXX, VectorX> get\_dynamic\_eq\_constrain(const Vector3 &c, const int &num\_support\_leg, const MatrixX3 &contactPoints)

{

assert(num\_support\_leg == contactPoints.rows());

MatrixXX D(6, 3 \* num\_support\_leg);

VectorX d(6);

// 对D赋值

for (int i = 0; i < num\_support\_leg; ++i)

{

D.block(0, 3 \* i, 3, 3).setIdentity(); // 单位矩阵

D.block(3, 3 \* i, 3, 3) = crossMatrix(contactPoints.row(i).transpose()); // 足端位置构成的反对称矩阵

}

// 对d赋值

d.setZero();

d(2) = 9.81 \* Mass; // 重力平衡

d(3) = c(0) \* 9.81 \* Mass; // 重力构成的倾覆力矩平衡

d(4) = c(1) \* 9.81 \* Mass;

return std::make\_pair(D, d);

}

std::pair<MatrixXX, VectorX> get\_jointToque\_neq\_constrain(const int &num\_support\_leg, const MatrixX3 &Joints\_angles)

{

assert(num\_support\_leg == Joints\_angles.rows());

MatrixXX A;

VectorX b;

A.resize(6 \* num\_support\_leg, 3 \* num\_support\_leg);

b.resize(6 \* num\_support\_leg);

A.setZero();

b.setZero();

for (int i = 0; i < num\_support\_leg; ++i)

{

Eigen::Matrix<double, 3, 3> J = get\_Jacobian(Joints\_angles.row(i).transpose());

Vector3 Leg\_T = get\_legMss\_consume\_Torque(Joints\_angles.row(i).transpose());

A.block(6 \* i, 3 \* i, 3, 3) = J.transpose(); // J^T <= maxT - Leg\_T

A.block(6 \* i + 3, 3 \* i, 3, 3) = -J.transpose(); //-J^T <= maxT + Leg\_T

b.block(6 \* i, 0, 3, 1) = MaxTorque - Leg\_T; // J^T <= maxT - Leg\_T

b.block(6 \* i + 3, 0, 3, 1) = MaxTorque + Leg\_T; //-J^T <= maxT + Leg\_T

}

return std::make\_pair(A, b);

}

bool is\_meet\_dynamic\_con(const Vector3 &c, const int &num\_support\_leg, const MatrixX3 &contactPoints, const MatrixX3 &Joints\_angles)

{

auto Dd = get\_dynamic\_eq\_constrain(c, num\_support\_leg, contactPoints);

auto Ab = get\_jointToque\_neq\_constrain(num\_support\_leg, Joints\_angles);

VectorX g, minBound, maxBound;

bool flag = solve\_LP\_GLPK(Ab.first, Ab.second, Dd.first, Dd.second, g, minBound, maxBound);

return flag;

}

}

#### planning.cpp

// 规划过程中使用的函数定义

#include "hit\_spider/MCTS/planning.hh"

namespace planning

{

namespace

{

/\*\*

\* @ : 0.初始化支撑状态列表，2^6 = 64种，减去少于三条腿支撑的情况一共：42种可能支撑状态

\* @description:应该放到private中的，因为只是用一次来初始化“已知的”支撑状态集合

\* @return {\*}

\*/

hexapod\_SupportState\_List createInitialSupportList()

{

hexapod\_SupportState\_List support\_state\_list(42, 6);

hexapod\_SupportState support\_state;

int ros\_index = 0;

for (int leg1 = 0; leg1 < 2; leg1++)

{

for (int leg2 = 0; leg2 < 2; leg2++)

{

for (int leg3 = 0; leg3 < 2; leg3++)

{

for (int leg4 = 0; leg4 < 2; leg4++)

{

for (int leg5 = 0; leg5 < 2; leg5++)

{

for (int leg6 = 0; leg6 < 2; leg6++)

{

if (leg1 + leg2 + leg3 + leg4 + leg5 + leg6 > 2)

{

support\_state(0) = leg1;

support\_state(1) = leg2;

support\_state(2) = leg3;

support\_state(3) = leg4;

support\_state(4) = leg5;

support\_state(5) = leg6;

support\_state\_list.row(ros\_index++) = support\_state;

}

}

}

}

}

}

}

// debug

assert(ros\_index == 42);

return support\_state\_list;

}

bool in\_polygon(const hit\_spider::hexapod\_Base\_Pose &pose\_now, const MatrixXX &CWC\_inputs)

{

MatrixXX A;

VectorX b;

if (!Robot\_State\_Transition::vertices\_to\_H(CWC\_inputs, A, b))

{

return false;

}

point\_Planar p;

p << pose\_now.position.x, pose\_now.position.y;

VectorX result = A \* p;

if ((result.array() <= b.array()).all())

{

return true;

}

return false;

}

void my\_round\_4(MatrixXX &points)

{

double num;

for (int i = 0; i < points.rows(); ++i)

{

for (int j = 0; j < points.cols(); ++j)

{

num = round(points(i, j) \* 1e4);

points(i, j) = num / (double)1e4;

}

}

}

}

const hexapod\_SupportState\_List initialSupportList = createInitialSupportList(); // 42个支撑状态构成集合

hexapod\_SupportState\_List findAvailable\_SupportStates(const hit\_spider::hexapod\_State &hexapodState)

{

//------支撑腿数量大于3，容错腿不能是支撑腿------

hexapod\_SupportState faultLeg;

for (int i = 0; i < 6; ++i)

{

faultLeg(i) = hexapodState.faultLeg\_State\_Now[i]; // 当前的错误退信息决定了Next的支撑可行状态

}

VectorXi can\_support(initialSupportList.rows());

can\_support.setZero();

for (int i = 0; i < initialSupportList.rows(); ++i)

{

if (initialSupportList.row(i) \* faultLeg == 0) // 容错腿不能当做支撑腿

{

can\_support(i) = 1;

}

}

hexapod\_SupportState\_List SupportList(can\_support.sum(), 6);

int row\_index = 0;

for (int i = 0; i < initialSupportList.rows(); ++i)

{

if (can\_support(i) == 1)

{

SupportList.row(row\_index++) = initialSupportList.row(i);

}

}

// debug

assert(row\_index == can\_support.sum());

const hexapod\_SupportState\_List SupportList\_temp = SupportList;

can\_support.resize(SupportList\_temp.rows(), 1);

can\_support.setZero();

//------判断哪些支撑状态是稳定的(运动学约束不用考虑，因为移动过程中考虑了运动学约束)------

for (int i = 0; i < SupportList\_temp.rows(); ++i)

{

int support\_leg\_num = 0; // 支撑腿数量

for (int j = 0; j < 6; ++j)

{

support\_leg\_num += SupportList\_temp(i, j);

}

std::vector<int> support\_leg; // 支撑腿标号

MatrixX3 contact\_Points(support\_leg\_num, 3); // 支撑腿接触点

MatrixX3 contact\_normals(support\_leg\_num, 3); // 接触点法向量

contact\_normals.setZero();

contact\_normals.col(2).setOnes(); // 先考虑简单情况

row\_index = 0;

for (int j = 0; j < 6; ++j)

{

if (SupportList\_temp(i, j) == 1)

{

contact\_Points.row(row\_index++) << hexapodState.feetPositionNow.foot[j].x,

hexapodState.feetPositionNow.foot[j].y,

hexapodState.feetPositionNow.foot[j].z;

support\_leg.push\_back(j + 1);

}

}

// 判断当前质心位置是否在下一步CWC内

if (support\_leg\_num == 3)

{

// 判断三个支撑点是否共线

MatrixX3 temp\_contact\_Points = contact\_Points;

temp\_contact\_Points.col(2).setOnes();

double S = temp\_contact\_Points.determinant();

if (S < 1e-5)

{

// debug

// ROS\_ERROR("三个支撑点共线");

continue;

}

}

MatrixXX CWC\_A = Robot\_State\_Transition::comput\_friction\_region(contact\_Points, contact\_normals, 0.5, Robot\_State\_Transition::Mass);

if (!in\_polygon(hexapodState.base\_Pose\_Now, CWC\_A))

{

continue;

}

// 判断当前质心位置是否满足关节力矩约束

Vector3 c;

c << hexapodState.base\_Pose\_Now.position.x, hexapodState.base\_Pose\_Now.position.y, hexapodState.base\_Pose\_Now.position.z;

MatrixX3 joints(support\_leg\_num, 3);

if ((support\_leg\_inverse\_kin(hexapodState.base\_Pose\_Now, hexapodState.feetPositionNow, support\_leg, joints)) && (Robot\_State\_Transition::is\_meet\_dynamic\_con(c, support\_leg\_num, contact\_Points, joints)))

{

can\_support(i) = 1;

}

}

// 更新可行支撑状态集合

SupportList.resize(can\_support.sum(), 6);

row\_index = 0;

for (int i = 0; i < SupportList\_temp.rows(); ++i)

{

if (can\_support(i) == 1)

{

SupportList.row(row\_index++) = SupportList\_temp.row(i);

}

}

// debug

assert(row\_index == can\_support.sum());

assert(SupportList.rows() == can\_support.sum());

return SupportList;

}

double get\_CWC\_base\_length(const hit\_spider::hexapod\_State &hexapodState)

{

int support\_leg\_num = 0;

for (int i = 0; i < 6; ++i)

{

support\_leg\_num += hexapodState.support\_State\_Next[i];

}

MatrixX3 contact\_Points(support\_leg\_num, 3);

MatrixX3 contact\_normals(support\_leg\_num, 3);

contact\_normals.setZero();

contact\_normals.col(2).setOnes(); // 先考虑简单情况

int row\_index = 0;

for (int i = 0; i < 6; ++i)

{

if (hexapodState.support\_State\_Next[i] == 1)

{

contact\_Points.row(row\_index++) << hexapodState.feetPositionNow.foot[i].x,

hexapodState.feetPositionNow.foot[i].y,

hexapodState.feetPositionNow.foot[i].z;

}

}

MatrixXX CWC\_A = Robot\_State\_Transition::comput\_friction\_region(contact\_Points, contact\_normals, 0.5, Robot\_State\_Transition::Mass);

// 求解多边形对应的约束

MatrixXX A;

VectorX b;

if (!Robot\_State\_Transition::vertices\_to\_H(CWC\_A, A, b))

{

my\_round\_4(CWC\_A);

if (!Robot\_State\_Transition::vertices\_to\_H(CWC\_A, A, b))

{

// debug

ROS\_ERROR("cdd不稳定!给它一次机会还是不稳定,请输入一个字符继续");

std::cout << contact\_Points << std::endl;

std::cout << CWC\_A << std::endl;

getchar();

return 0;

}

}

// 优化求解最大CWC约束的最大步长

MatrixXX D(1, 2);

VectorX d(1);

D << 0, 1;

d << 0;

VectorX g(2), minBound, maxBound;

g << 1, 0;

double CWC\_reslut\_length = 0.0;

Robot\_State\_Transition::solve\_LP\_GLPK(A, b, D, d, g, minBound, maxBound, true, nullptr, &CWC\_reslut\_length);

return (CWC\_reslut\_length - hexapodState.base\_Pose\_Now.position.x);

}

double get\_kin\_length(const hit\_spider::hexapod\_State &hexapodState)

{

int support\_leg\_num = 0;

for (int i = 0; i < 6; ++i)

{

support\_leg\_num += hexapodState.support\_State\_Next[i];

}

std::vector<int> support\_leg;

MatrixX3 contact\_Points(support\_leg\_num, 3);

MatrixX3 contact\_normals(support\_leg\_num, 3);

contact\_normals.setZero();

contact\_normals.col(2).setOnes(); // 先考虑简单情况

int row\_index = 0;

for (int i = 0; i < 6; ++i)

{

if (hexapodState.support\_State\_Next[i] == SUPPORT)

{

contact\_Points.row(row\_index++) << hexapodState.feetPositionNow.foot[i].x,

hexapodState.feetPositionNow.foot[i].y,

hexapodState.feetPositionNow.foot[i].z;

support\_leg.push\_back(i + 1);

}

}

// 当前支撑腿对质心的运动约束

auto feet\_con\_cog\_Ab = Robot\_State\_Transition::get\_kinematics\_con\_foot\_cog(hexapodState.base\_Pose\_Now, support\_leg, contact\_Points, contact\_normals);

MatrixXX A(7 \* support\_leg\_num, 3);

VectorX b(7 \* support\_leg\_num);

A.setZero();

b.setZero();

A.block(0, 0, 7 \* support\_leg\_num, 3) = feet\_con\_cog\_Ab.first;

b.block(0, 0, 7 \* support\_leg\_num, 1) = feet\_con\_cog\_Ab.second;

// 优化求解最大kin约束的最大步长

MatrixXX D(2, 3);

VectorX d(2);

D << 0, 1, 0, // y =0

0, 0, 1; // z=0.5

d << 0, 0.5;

VectorX g(3), minBound, maxBound;

g << 1, 0, 0;

double kin\_reslut\_length = 0.0;

Robot\_State\_Transition::solve\_LP\_GLPK(A, b, D, d, g, minBound, maxBound, true, nullptr, &kin\_reslut\_length);

return (kin\_reslut\_length - hexapodState.base\_Pose\_Now.position.x);

}

double get\_dynamic\_length(const hit\_spider::hexapod\_State &hexapodState)

{

// 不考虑joint扭矩约束的最大步长

double length\_ = 0.9 \* std::min(get\_CWC\_base\_length(hexapodState), get\_kin\_length(hexapodState));

if (length\_ < 1e-3)

{

length\_ = 0;

return length\_;

}

int support\_leg\_num = 0;

for (int i = 0; i < 6; ++i)

{

support\_leg\_num += hexapodState.support\_State\_Next[i];

}

MatrixX3 contact\_Points(support\_leg\_num, 3);

std::vector<int> support\_leg;

int row\_index = 0;

for (int i = 0; i < 6; ++i)

{

if (hexapodState.support\_State\_Next[i] == SUPPORT)

{

contact\_Points.row(row\_index++) << hexapodState.feetPositionNow.foot[i].x,

hexapodState.feetPositionNow.foot[i].y,

hexapodState.feetPositionNow.foot[i].z;

support\_leg.push\_back(i + 1);

}

}

Vector3 c(0, 0, 0.5);

hit\_spider::hexapod\_Base\_Pose base\_pose = hexapodState.base\_Pose\_Now;

MatrixX3 joints(support\_leg\_num, 3);

// 离散为10个点

int i = 0;

while (1)

{

for (i = 0; i < 10; ++i)

{

c(0) = hexapodState.base\_Pose\_Now.position.x + (double)(i + 1) \* length\_ / (double)10;

base\_pose.position.x = c(0);

if ((!support\_leg\_inverse\_kin(base\_pose, hexapodState.feetPositionNow, support\_leg, joints)) || (!Robot\_State\_Transition::is\_meet\_dynamic\_con(c, support\_leg\_num, contact\_Points, joints)))

{

length\_ = (c(0) - hexapodState.base\_Pose\_Now.position.x) \* 0.9;

break;

}

}

if (i == 10)

{

break;

}

}

return length\_;

}

hexapod\_SupportState\_List findAvailable\_SupportStates\_and\_maxLength(const hit\_spider::hexapod\_State &hexapodState, std::vector<double> &max\_length)

{

std::vector<double> temp\_vec\_destroy;

max\_length.swap(temp\_vec\_destroy); // 清空内存

//------支撑腿数量大于3，容错腿不能是支撑腿------

hexapod\_SupportState faultLeg;

for (int i = 0; i < 6; ++i)

{

faultLeg(i) = hexapodState.faultLeg\_State\_Now[i]; // 当前的错误退信息决定了Next的支撑可行状态

}

VectorXi can\_support(initialSupportList.rows());

can\_support.setZero();

for (int i = 0; i < initialSupportList.rows(); ++i)

{

if (initialSupportList.row(i) \* faultLeg == 0) // 容错腿不能当做支撑腿

{

can\_support(i) = 1;

}

}

hexapod\_SupportState\_List SupportList(can\_support.sum(), 6);

int row\_index = 0;

for (int i = 0; i < initialSupportList.rows(); ++i)

{

if (can\_support(i) == 1)

{

SupportList.row(row\_index++) = initialSupportList.row(i);

}

}

const hexapod\_SupportState\_List SupportList\_temp = SupportList;

can\_support.resize(SupportList\_temp.rows(), 1);

can\_support.setZero();

//------判断哪些支撑状态是稳定的(运动学约束不用考虑，因为移动过程中考虑了运动学约束)------

for (int i = 0; i < SupportList\_temp.rows(); ++i)

{

// 数据准备

int support\_leg\_num = 0; // 支撑腿数量

for (int j = 0; j < 6; ++j)

{

support\_leg\_num += SupportList\_temp(i, j);

}

std::vector<int> support\_leg; // 支撑腿标号

MatrixX3 contact\_Points(support\_leg\_num, 3); // 支撑腿接触点

MatrixX3 contact\_normals(support\_leg\_num, 3); // 接触点法向量

contact\_normals.setZero();

contact\_normals.col(2).setOnes(); // 先考虑简单情况

row\_index = 0;

for (int j = 0; j < 6; ++j)

{

if (SupportList\_temp(i, j) == 1)

{

contact\_Points.row(row\_index++) << hexapodState.feetPositionNow.foot[j].x,

hexapodState.feetPositionNow.foot[j].y,

hexapodState.feetPositionNow.foot[j].z;

support\_leg.push\_back(j + 1);

}

}

// debug

// std::cout << contact\_Points << std::endl;

// 1.判断当前质心位置是否在下一步CWC内

if (support\_leg\_num == 3)

{

// 判断三个支撑点是否共线

MatrixX3 temp\_contact\_Points = contact\_Points;

temp\_contact\_Points.col(2).setOnes();

double S = temp\_contact\_Points.determinant();

if (S < 1e-5)

{

// debug

// ROS\_ERROR("三个支撑点共线");

continue;

}

}

MatrixXX CWC\_A = Robot\_State\_Transition::comput\_friction\_region(contact\_Points, contact\_normals, 0.5, Robot\_State\_Transition::Mass);

MatrixXX A;

VectorX b;

// cdd求解约束方程

if (!Robot\_State\_Transition::vertices\_to\_H(CWC\_A, A, b))

{

my\_round\_4(CWC\_A);

if (!Robot\_State\_Transition::vertices\_to\_H(CWC\_A, A, b))

{

// debug

// ROS\_ERROR("cdd不稳定!给它一次机会还是不稳定"); // 认为此支撑状态是不行的

continue;

}

// debug

// std::cout << "cdd虽然有数值问题,但是减小精度就行了" << std::endl;

}

// 判断是否在多边形内部

point\_Planar c\_xy;

c\_xy << hexapodState.base\_Pose\_Now.position.x, hexapodState.base\_Pose\_Now.position.y;

if (!((A \* c\_xy).array() <= b.array()).all()) // 当前质心不在CWC多边形内部

{

continue;

}

// 2.判断当前质心位置是否满足关节力矩约束，

Vector3 c;

c << hexapodState.base\_Pose\_Now.position.x, hexapodState.base\_Pose\_Now.position.y, hexapodState.base\_Pose\_Now.position.z;

MatrixX3 joints(support\_leg\_num, 3);

// 当前质心位置是否满足支撑腿的关节力矩约束

if ((support\_leg\_inverse\_kin(hexapodState.base\_Pose\_Now, hexapodState.feetPositionNow, support\_leg, joints)) && (Robot\_State\_Transition::is\_meet\_dynamic\_con(c, support\_leg\_num, contact\_Points, joints)))

{

can\_support(i) = 1;

}

else

{

continue;

}

// 3.求解CWC最大步长

MatrixXX D(1, 2);

VectorX d(1);

D << 0, 1;

d << 0;

VectorX g(2), minBound, maxBound;

g << 1, 0;

double CWC\_reslut\_length = 0.0;

Robot\_State\_Transition::solve\_LP\_GLPK(A, b, D, d, g, minBound, maxBound, true, nullptr, &CWC\_reslut\_length);

CWC\_reslut\_length -= hexapodState.base\_Pose\_Now.position.x;

// 4.求解kin最大步长

// 当前支撑腿对质心的运动约束

auto feet\_con\_cog\_Ab = Robot\_State\_Transition::get\_kinematics\_con\_foot\_cog(hexapodState.base\_Pose\_Now, support\_leg, contact\_Points, contact\_normals);

A.resize(7 \* support\_leg\_num, 3);

b.resize(7 \* support\_leg\_num);

A.setZero();

b.setZero();

A.block(0, 0, 7 \* support\_leg\_num, 3) = feet\_con\_cog\_Ab.first;

b.block(0, 0, 7 \* support\_leg\_num, 1) = feet\_con\_cog\_Ab.second;

// 优化求解最大kin约束的最大步长

D.resize(2, 3);

d.resize(2, 1);

D << 0, 1, 0,

0, 0, 1;

d << 0, 0.5;

g.resize(3, 1);

g << 1, 0, 0;

double kin\_reslut\_length = 0.0;

Robot\_State\_Transition::solve\_LP\_GLPK(A, b, D, d, g, minBound, maxBound, true, nullptr, &kin\_reslut\_length);

kin\_reslut\_length -= hexapodState.base\_Pose\_Now.position.x;

// 5.求解dynamic最大步长

double length\_ = 0.9 \* std::min(CWC\_reslut\_length, kin\_reslut\_length);

if (length\_ < 1e-5)

{

length\_ = 0;

max\_length.push\_back(length\_);

continue; // 继续判断下一个支撑状态

}

c << 0, 0, 0.5;

hit\_spider::hexapod\_Base\_Pose base\_pose = hexapodState.base\_Pose\_Now;

joints.setZero();

// 离散为10个点

int ii = 0;

while (1)

{

for (ii = 0; ii < 10; ++ii)

{

c(0) = hexapodState.base\_Pose\_Now.position.x + (double)(ii + 1) \* length\_ / (double)10;

base\_pose.position.x = c(0);

if ((!support\_leg\_inverse\_kin(base\_pose, hexapodState.feetPositionNow, support\_leg, joints)) || (!Robot\_State\_Transition::is\_meet\_dynamic\_con(c, support\_leg\_num, contact\_Points, joints)))

{

length\_ = (c(0) - hexapodState.base\_Pose\_Now.position.x) \* 0.9;

break;

}

}

if (ii == 10)

{

break;

}

}

max\_length.push\_back(length\_);

}

// 更新可行支撑状态集合

SupportList.resize(can\_support.sum(), 6);

row\_index = 0;

for (int i = 0; i < SupportList\_temp.rows(); ++i)

{

if (can\_support(i) == 1)

{

SupportList.row(row\_index++) = SupportList\_temp.row(i);

}

}

// debug

assert(row\_index == can\_support.sum());

assert(SupportList.rows() == can\_support.sum());

assert((int)max\_length.size() == row\_index);

return SupportList;

}

Footholds getAvailableFootholds(const hit\_spider::hexapod\_State &hexapodState)

{

Footholds output\_feet\_position;

// 环境可落足点集合

MatrixX3 map\_feasible\_position = Static\_Information::get\_now\_Feasible\_foot\_position(hexapodState.base\_Pose\_Next);

// base约束foot

auto Ab = Robot\_State\_Transition::get\_kinematics\_con\_cog\_foot(hexapodState.base\_Pose\_Next);

for (int i = 0; i < 6; ++i)

{

if (hexapodState.support\_State\_Next[i] == SWING)

{

MatrixX3 A = Ab.first.block(7 \* i, 0, 7, 3);

VectorX b = Ab.second.block(7 \* i, 0, 7, 1);

// 判断环境中的点是否在kin质心约束foot的约束范围内

// for (int j = 0; j < (int)map\_feasible\_position.rows(); ++j)

// {

// VectorX temp = A \* map\_feasible\_position.row(j).transpose();

// if ((temp.array() <= b.array()).all())

// {

// output\_feet\_position.leg[i].push\_back(map\_feasible\_position.row(j).transpose());

// }

// }

MatrixXX temp = A \* map\_feasible\_position.transpose();

for (int j = 0; j < temp.cols(); ++j)

{

if ((temp.col(j).array() <= b.array()).all())

{

output\_feet\_position.leg[i].push\_back(map\_feasible\_position.row(j).transpose());

}

}

}

}

return output\_feet\_position;

}

void swingLegFoot\_position\_Expert(hit\_spider::hexapod\_State &hexapodState)

{

Footholds feasible\_positions = getAvailableFootholds(hexapodState);

for (int i = 0; i < 6; ++i)

{

if (hexapodState.support\_State\_Next[i] == SWING)

{

if (feasible\_positions.leg[i].size() != 0)

{

// 寻找x最大的落足点

auto max\_iter = std::max\_element(feasible\_positions.leg[i].begin(), feasible\_positions.leg[i].end(),

[](const Vector3 &v1, const Vector3 &v2)

{ return v1(0) < v2(0); });

hexapodState.feetPositionNext.foot[i].x = max\_iter->x();

hexapodState.feetPositionNext.foot[i].y = max\_iter->y();

hexapodState.feetPositionNext.foot[i].z = max\_iter->z();

}

else

{

hexapodState.faultLeg\_State\_Next[i] = FAULT\_LEG;

hexapodState.feetPositionNext.foot[i].x = hexapodState.base\_Pose\_Next.position.x + Hexapod\_defaultFoothold[i].x();

hexapodState.feetPositionNext.foot[i].y = hexapodState.base\_Pose\_Next.position.y + Hexapod\_defaultFoothold[i].y();

hexapodState.feetPositionNext.foot[i].z = hexapodState.base\_Pose\_Next.position.z + Hexapod\_defaultFoothold[i].z() + 0.3f;

}

}

else

{

hexapodState.feetPositionNext.foot[i] = hexapodState.feetPositionNow.foot[i]; // 支撑腿的位置不变

}

}

}

void get\_nextState\_Expert(hit\_spider::hexapod\_State &hexapodState)

{

hit\_spider::hexapod\_State temp\_hexapodState = hexapodState;

// 获取可行支撑状态

std::vector<double> max\_length;

const hexapod\_SupportState\_List feasible\_support\_state = findAvailable\_SupportStates\_and\_maxLength(hexapodState, max\_length);

// debug

// std::cout << "支撑状态有:" << std::endl

// << feasible\_support\_state << std::endl;

// std::cout << "前进距离为:" << std::endl;

// for (int i = 0; i < (int)max\_length.size(); ++i)

// {

// std::cout << max\_length[i] << '\n';

// }

// std::cout << std::endl;

auto iter = std::max\_element(max\_length.begin(), max\_length.end());

int support\_index = iter - max\_length.begin();

// 专家法选择最大前进距离对应的支撑状态作为下一步的支撑状态

hexapodState.base\_Pose\_Next = hexapodState.base\_Pose\_Now;

hexapodState.base\_Pose\_Next.position.x += \*iter;

for (int j = 0; j < 6; ++j)

{

hexapodState.support\_State\_Next[j] = feasible\_support\_state(support\_index, j);

}

// 专家法为摆动腿选择落足点

swingLegFoot\_position\_Expert(hexapodState);

}

void get\_nextState\_Rondom(hit\_spider::hexapod\_State &hexapodState)

{

hit\_spider::hexapod\_State temp\_hexapodState = hexapodState;

// 获取可行支撑状态

std::vector<double> max\_length;

const hexapod\_SupportState\_List feasible\_support\_state = findAvailable\_SupportStates\_and\_maxLength(hexapodState, max\_length);

// debug

// std::cout << "支撑状态有:" << std::endl

// << feasible\_support\_state << std::endl;

// std::cout << "前进距离为:" << std::endl;

// for (int i = 0; i < (int)max\_length.size(); ++i)

// {

// std::cout << max\_length[i] << '\n';

// }

// std::cout << std::endl;

int temp\_num = rand\_customization(0, max\_length.size() - 1);

// 随机选择支撑状态

hexapodState.base\_Pose\_Next = hexapodState.base\_Pose\_Now;

hexapodState.base\_Pose\_Next.position.x += max\_length[temp\_num];

for (int j = 0; j < 6; ++j)

{

hexapodState.support\_State\_Next[j] = feasible\_support\_state(temp\_num, j);

}

// 专家法为摆动腿选择落足点

swingLegFoot\_position\_Expert(hexapodState);

}

std::vector<hit\_spider::hexapod\_State> get\_NextState\_list(const hit\_spider::hexapod\_State &hexapodState)

{

hit\_spider::hexapod\_State temp\_hexapodState = hexapodState;

// 确定可行支撑状态

std::vector<double> max\_length;

const hexapod\_SupportState\_List feasible\_support\_state = findAvailable\_SupportStates\_and\_maxLength(hexapodState, max\_length);

// 输出的备选状态集

std::vector<hit\_spider::hexapod\_State> state\_list;

state\_list.reserve(feasible\_support\_state.rows());

for (int i = 0; i < (int)feasible\_support\_state.rows(); ++i)

{

// debug

// ROS\_INFO("第%d个支撑状态为:", i + 1);

// std::cout << feasible\_support\_state.row(i) << std::endl;

// 确定每一个支撑状态的前进距离

for (int j = 0; j < 6; ++j)

{

temp\_hexapodState.support\_State\_Next[j] = feasible\_support\_state(i, j);

}

double temp\_length = max\_length[i];

temp\_hexapodState.base\_Pose\_Next = hexapodState.base\_Pose\_Now;

temp\_hexapodState.base\_Pose\_Next.position.x += temp\_length;

// 确定摆动腿落足点

swingLegFoot\_position\_Expert(temp\_hexapodState);

temp\_hexapodState.base\_Pose\_Now = temp\_hexapodState.base\_Pose\_Next; // 机体位置

temp\_hexapodState.feetPositionNow = temp\_hexapodState.feetPositionNext; // 落足点位置

temp\_hexapodState.support\_State\_Now = temp\_hexapodState.support\_State\_Next; // 支撑状态

temp\_hexapodState.faultLeg\_State\_Now = temp\_hexapodState.faultLeg\_State\_Next; // 容错腿状态

state\_list.push\_back(temp\_hexapodState);

}

return state\_list;

}

}

#### MTree.cpp

// MCTS搜索树类具体定义

#include "hit\_spider/MCTS/MTree.hh"

//---构造与初始化

MTree::MTree(MNode\_ptr &root\_input) : root(root\_input), Sliding\_New\_root(root\_input)

{

this->root->visitTimes = 0;

this->root->nodeDepth = 1;

this->root->Parent = nullptr;

this->root->disParentToMe = 0;

this->root->simMaxDis = 0;

this->root->average\_step\_length = 0;

}

void MTree::Set\_PAR(const BestChiledParameter &PAR)

{

this->PAR = PAR;

}

void MTree::Set\_PAR2(const BestChiledParameter &PAR2)

{

this->PAR2 = PAR2;

}

// 拓展树的操作

void MTree::addNodeChild(MNode\_ptr &node, MNode\_ptr &parent)

{

node->Parent = parent;

node->nodeDepth = parent->nodeDepth + 1;

parent->children.insert(node);

}

// 获取树的深度

int getDepth(const MNode\_ptr &node)

{

if (node == nullptr) // 树还没有被使用过

{

return 0;

}

int res = 1;

std::set<MNode\_ptr> sons = node->children;

for (auto iter = sons.cbegin(); iter != sons.cend(); ++iter)

{

res = std::max(res, getDepth(\*iter) + 1);

}

return res;

}

int MTree::getMaxDepth()

{

return getDepth(this->root);

}

//------MCTS搜索树核心------

MNode\_ptr MTree::findBestChild\_Expand(const MNode\_ptr &node\_findBestChild)

{

auto best\_iter = node\_findBestChild->children.end(); // 指向一个set集合最后一个元素的下一个位置

float best\_Value = -1; // 最优儿子对应的UCB值

for (auto iter = node\_findBestChild->children.begin(); iter != node\_findBestChild->children.end(); ++iter)

{

// UCB计算：为什么要开三次方根呢? -> 为了归一化

float tmpValue = this->PAR.w1 \* pow(((\*iter)->simMaxDis) / (double)this->PAR.simStepNum, (double)1 / 3.0f) +

this->PAR.w2 \* pow(((\*iter)->average\_step\_length), (double)1 / 3.0f) +

this->PAR.C \* sqrt(log(float(this->Sliding\_New\_root->visitTimes)) / float((\*iter)->visitTimes));

if (tmpValue > best\_Value) // 寻找UCB的最大值

{

best\_Value = tmpValue;

best\_iter = iter;

}

}

// 代表返回的值为正常节点

if (best\_iter != node\_findBestChild->children.end())

{

return \*best\_iter;

}

else // 代表返回为被忽略的节点,所有的儿子节点全被忽略了

{

return node\_findBestChild;

}

}

MNode\_ptr MTree::findBestChild\_Sliding\_Root()

{

auto best\_iter = this->Sliding\_New\_root->children.end(); // 指向一个set集合最后一个元素的下一个位置

float best\_Value = -1; // 最优儿子对应的UCB值

for (auto iter = this->Sliding\_New\_root->children.begin(); iter != this->Sliding\_New\_root->children.end(); ++iter)

{

// UCB计算：为什么要开三次方根呢? -> 为了归一化

float tmpValue = this->PAR2.w1 \* pow(((\*iter)->simMaxDis) / (float)this->PAR.simStepNum, (double)1 / 3.0f) +

this->PAR2.w2 \* pow(((\*iter)->average\_step\_length), (double)1 / 3.0f);

if (tmpValue > best\_Value) // 寻找UCB的最大值

{

best\_Value = tmpValue;

best\_iter = iter;

}

}

// 代表返回的值为正常节点

if (best\_iter != this->Sliding\_New\_root->children.end())

{

return \*best\_iter;

}

else // 代表返回为被忽略的节点,所有的儿子节点全被忽略了

{

return this->Sliding\_New\_root;

}

}

MNode\_ptr MTree::MCTS\_SelectionAndExpansion()

{

MNode\_ptr root\_ = this->Sliding\_New\_root; // 从搜索树的滑动根节点开始搜索

// 死循环，一定要找到一个拓展节点出来！

while (1)

{

// 节点未全部展开，那UCB最大值一定是未展开节点上出现

if (root\_->alternativeStates.size() != 0)

{

// 拓展一次该节点，返回拓展节点的指针

MNode\_ptr returnNode = this->MCTS\_Expansion(root\_);

return returnNode; // 返回新拓展出的节点

}

// ！！！滑动后的序列并不好，一定要避免，出现这种情况就是UCB参数不合理！！！

// 希望这个不要发生，否则滑动根节点就没有任何意义了!!!

// 当前节点已经全部展开，但是所有备选儿子都是卡死的，没有一个真正的被加入到树中，所以该节点也是卡死节点

if (root\_->children.size() == 0)

{

// 将该节点从它的父节点中的子节点list中删除

if (root\_->Parent == nullptr) // 该节点就是根节点

{

ROS\_ERROR("!!!蒙特卡洛搜索树初始就是卡死的!!!\n 请输入一个字符以继续:");

getchar(); // 等到输入一个字符

}

// 并不是最根节点，还有希望求解，把该节点从它的父节点的children集合中删除

root\_->Parent->children.erase(root\_);

ROS\_ERROR("滑动根节点并不好\n请输入一个字符以继续:"); // 期望这个不要发生，否则滑动根节点就没有任何意义了!!!

root\_ = root\_->Parent; // 蒙特卡洛搜索树向前回溯一个节点

getchar();

continue;

}

else

{

//------当前滑动根节点节点已经全部展开，寻找它的最佳儿子节点，对最佳儿子节点再进行expandsion

root\_ = this->findBestChild\_Expand(root\_);

}

}

}

MNode\_ptr MTree::MCTS\_Expansion(MNode\_ptr &node\_expand)

{

// 在备选状态中随机选择一个状态

int index = rand\_customization(0, node\_expand->alternativeStates.size() - 1);

MNode\_ptr node = std::make\_shared<MNode>(node\_expand->alternativeStates[index]);

node\_expand->alternativeStates.erase(node\_expand->alternativeStates.begin() + index); // 从父节点的备选状态中删除

node->alternativeStates = planning::get\_NextState\_list(node->element); // 新拓展节点的备选状态集合

this->addNodeChild(node, node\_expand); // 把新拓展的节点添加到树中

// 当前节点距离它的父节点距离

node->disParentToMe = node->element.base\_Pose\_Now.position.x - node\_expand->element.base\_Pose\_Now.position.x;

node->average\_step\_length = (node->element.base\_Pose\_Now.position.x - this->Sliding\_New\_root->element.base\_Pose\_Now.position.x) / (double)(node->nodeDepth - this->Sliding\_New\_root->nodeDepth);

return node;

}

std::vector<hit\_spider::hexapod\_State> MTree::MCTS\_Simulation(MNode\_ptr &node\_sim)

{

std::vector<hit\_spider::hexapod\_State> smi\_state\_squence; // 仿真序列

hit\_spider::hexapod\_State hexapodState = node\_sim->element; // 向后仿真的state

int stepCount = 0; // 当前仿真步数

while (1)

{

++stepCount;

planning::get\_nextState\_Rondom(hexapodState); // 随机获取下一个支撑状态，专家法获取落足点

// 机器人向后走

hexapodState.base\_Pose\_Now = hexapodState.base\_Pose\_Next;

hexapodState.feetPositionNow = hexapodState.feetPositionNext;

hexapodState.faultLeg\_State\_Now = hexapodState.faultLeg\_State\_Next;

hexapodState.support\_State\_Now = hexapodState.support\_State\_Next;

smi\_state\_squence.push\_back(hexapodState); // 存入仿真序列中

if (stepCount >= this->PAR.simStepNum) // 仿真步数到了

{

// 仿真出来的最远距离

node\_sim->simMaxDis = hexapodState.base\_Pose\_Now.position.x;

return smi\_state\_squence;

}

}

}

bool MTree::MCTS\_Backpropagation(MNode\_ptr &node\_back)

{

node\_back->visitTimes += 1; // simulation后的叶节点访问次数加一

// 传播其余分值信息

// 传播访问次数信息

MNode\_ptr node\_ = node\_back;

while (1)

{

node\_ = node\_->Parent; // 不断向根节点方向传播

if (node\_ == nullptr) // 已经全部传播完了

{

break;

}

node\_->visitTimes += 1; // 父节点的访问次数增加1

}

// 新拓展出的节点是否卡死

if (node\_back->alternativeStates.size() == 0)

{

// debug

ROS\_WARN("新拓展出来的节点是卡死的");

node\_back->Parent->children.erase(node\_back);

return false;

}

// 传播最远仿真距离

node\_ = node\_back;

while (1)

{

node\_ = node\_->Parent; // 不断向根节点方向传播

if (node\_ == nullptr) // 已经全部传播完了

{

break;

}

double yuanlai = this->PAR2.w1 \* pow((node\_->simMaxDis) / (double)this->PAR.simStepNum, (double)1 / 3.0f) + this->PAR2.w2 \* pow(node\_back->average\_step\_length, (double)1 / 3.0f);

double tmpValue = this->PAR2.w1 \* pow((node\_back->simMaxDis) / (double)this->PAR.simStepNum, (double)1 / 3.0f) + this->PAR2.w2 \* pow(node\_back->average\_step\_length, (double)1 / 3.0f);

if (tmpValue > yuanlai)

{

node\_->simMaxDis = node\_back->simMaxDis;

node\_->average\_step\_length = node\_back->average\_step\_length;

}

}

return true;

}

#### trajectory.cpp

// 接触序列规划完成后，规划足端轨迹、机体轨迹具体实现

#include "hit\_spider/MCTS/trajectory.hh"

using namespace std;

namespace trajectory

{

// 腿部运动参数计算 7：每条腿每个方向的7个解析参数 18：每条腿三个方向\*6条腿，还能修改摆动腿的步高

Eigen::Matrix<float, 7, 18> solution\_leg(hit\_spider::hexapod\_State state1, hit\_spider::hexapod\_State state2)

{

Eigen::Matrix<float, 7, 7> mat\_leg, inversemat\_leg;

mat\_leg << 1, 0, 0, 0, 0, 0, 0,

1, 0.5, pow(0.5, 2), pow(0.5, 3), pow(0.5, 4), pow(0.5, 5), pow(0.5, 6),

1, 1, 1, 1, 1, 1, 1,

0, 1, 0, 0, 0, 0, 0,

0, 1, 2, 3, 4, 5, 6,

0, 0, 2, 0, 0, 0, 0,

0, 0, 2, 6, 12, 20, 30;

inversemat\_leg = mat\_leg.inverse(); // 矩阵求逆 Eigen库的用法

// vector<float> solution\_leg(3, 0);

Eigen::Matrix<float, 7, 18> solutionxyz = Eigen::MatrixXf::Zero(7, 18); // 矩阵初始化

Eigen::Matrix<float, 7, 18> knownquantity = Eigen::MatrixXf::Zero(7, 18);

for (int i = 0; i < 6; i++)

{

if (state2.support\_State\_Now[i] == 0) // 判断摆动腿 0：Swing

{

knownquantity(0, 3 \* i) = state1.feetPositionNow.foot[i].x;

knownquantity(0, 3 \* i + 1) = state1.feetPositionNow.foot[i].y;

knownquantity(0, 3 \* i + 2) = state1.feetPositionNow.foot[i].z;

knownquantity(2, 3 \* i) = state2.feetPositionNow.foot[i].x;

knownquantity(2, 3 \* i + 1) = state2.feetPositionNow.foot[i].y;

knownquantity(2, 3 \* i + 2) = state2.feetPositionNow.foot[i].z;

float h = state2.feetPositionNow.foot[i].z + 0.3; // 摆动腿步高 ////////////////////////////////

knownquantity(1, 3 \* i) = (state1.feetPositionNow.foot[i].x + state2.feetPositionNow.foot[i].x) / 2.0f;

knownquantity(1, 3 \* i + 1) = (state1.feetPositionNow.foot[i].y + state2.feetPositionNow.foot[i].y) / 2.0f;

knownquantity(1, 3 \* i + 2) = h;

}

}

solutionxyz = inversemat\_leg \* knownquantity;

return solutionxyz;

}

// 腿部数据赋值

void leg\_assignment(hit\_spider::hexapod\_State state, hit\_spider::FeetPosition &feetNow, Eigen::Matrix<float, 7, 18> solution, float t)

{

for (int j = 0; j < 6; j++)

{

if (state.support\_State\_Now[j] == 0)

{

feetNow.foot[j].x = solution(0, 3 \* j) + solution(1, 3 \* j) \* t + solution(2, 3 \* j) \* pow(t, 2) + solution(3, 3 \* j) \* pow(t, 3) + solution(4, 3 \* j) \* pow(t, 4) + solution(5, 3 \* j) \* pow(t, 5) + solution(6, 3 \* j) \* pow(t, 6);

feetNow.foot[j].y = solution(0, 3 \* j + 1) + solution(1, 3 \* j + 1) \* t + solution(2, 3 \* j + 1) \* pow(t, 2) + solution(3, 3 \* j + 1) \* pow(t, 3) + solution(4, 3 \* j + 1) \* pow(t, 4) + solution(5, 3 \* j + 1) \* pow(t, 5) + solution(6, 3 \* j + 1) \* pow(t, 6);

feetNow.foot[j].z = solution(0, 3 \* j + 2) + solution(1, 3 \* j + 2) \* t + solution(2, 3 \* j + 2) \* pow(t, 2) + solution(3, 3 \* j + 2) \* pow(t, 3) + solution(4, 3 \* j + 2) \* pow(t, 4) + solution(5, 3 \* j + 2) \* pow(t, 5) + solution(6, 3 \* j + 2) \* pow(t, 6);

}

}

}

// 机体参数计算 6：机体每个方向的6个解析参数 3：xyz三个方向

Eigen::Matrix<float, 6, 3> solution\_body(hit\_spider::hexapod\_Base\_Pose robotNow, hit\_spider::hexapod\_Base\_Pose robotNext)

{

Eigen::Matrix<float, 6, 6> mat\_body, inversemat\_body;

mat\_body << 1, 0, 0, 0, 0, 0,

1, 1, 1, 1, 1, 1,

0, 1, 0, 0, 0, 0,

0, 1, 2, 3, 4, 5,

0, 0, 2, 0, 0, 0,

0, 0, 2, 6, 12, 20;

inversemat\_body = mat\_body.inverse(); // 矩阵求逆 Eigen库的用法

Eigen::Matrix<float, 6, 3> solutionxyz\_body;

Eigen::Matrix<float, 6, 3> knownquantity\_body = Eigen::MatrixXf::Zero(6, 3); // 矩阵初始化

knownquantity\_body(0, 0) = robotNow.position.x;

knownquantity\_body(0, 1) = robotNow.position.y;

knownquantity\_body(0, 2) = robotNow.position.z;

knownquantity\_body(1, 0) = robotNext.position.x;

knownquantity\_body(1, 1) = robotNext.position.y;

knownquantity\_body(1, 2) = robotNext.position.z;

solutionxyz\_body = inversemat\_body \* knownquantity\_body;

return solutionxyz\_body;

}

Eigen::Matrix<float, 6, 3> solution\_body\_rotation(hit\_spider::hexapod\_Base\_Pose robotNow, hit\_spider::hexapod\_Base\_Pose robotNext)

{

Eigen::Matrix<float, 6, 6> mat\_body, inversemat\_body;

mat\_body << 1, 0, 0, 0, 0, 0,

1, 1, 1, 1, 1, 1,

0, 1, 0, 0, 0, 0,

0, 1, 2, 3, 4, 5,

0, 0, 2, 0, 0, 0,

0, 0, 2, 6, 12, 20;

inversemat\_body = mat\_body.inverse(); // 矩阵求逆 Eigen库的用法

Eigen::Matrix<float, 6, 3> solutionxyz\_body;

Eigen::Matrix<float, 6, 3> knownquantity\_body = Eigen::MatrixXf::Zero(6, 3); // 矩阵初始化

knownquantity\_body(0, 0) = robotNow.orientation.roll;

knownquantity\_body(0, 1) = robotNow.orientation.pitch;

knownquantity\_body(0, 2) = robotNow.orientation.yaw;

knownquantity\_body(1, 0) = robotNext.orientation.roll;

knownquantity\_body(1, 1) = robotNext.orientation.pitch;

knownquantity\_body(1, 2) = robotNext.orientation.yaw;

solutionxyz\_body = inversemat\_body \* knownquantity\_body;

return solutionxyz\_body;

}

// 机体数据赋值

void body\_assignment(hit\_spider::hexapod\_Base\_Pose &robotNext, Eigen::Matrix<float, 6, 3> solution, Eigen::Matrix<float, 6, 3> solutionbodyRotate, float t)

{

robotNext.position.x = solution(0, 0) + solution(1, 0) \* t + solution(2, 0) \* pow(t, 2) + solution(3, 0) \* pow(t, 3) + solution(4, 0) \* pow(t, 4) + solution(5, 0) \* pow(t, 5);

robotNext.position.y = solution(0, 1) + solution(1, 1) \* t + solution(2, 1) \* pow(t, 2) + solution(3, 1) \* pow(t, 3) + solution(4, 1) \* pow(t, 4) + solution(5, 1) \* pow(t, 5);

robotNext.position.z = solution(0, 2) + solution(1, 2) \* t + solution(2, 2) \* pow(t, 2) + solution(3, 2) \* pow(t, 3) + solution(4, 2) \* pow(t, 4) + solution(5, 2) \* pow(t, 5);

robotNext.orientation.roll = solutionbodyRotate(0, 0) + solutionbodyRotate(1, 0) \* t + solutionbodyRotate(2, 0) \* pow(t, 2) + solutionbodyRotate(3, 0) \* pow(t, 3) + solutionbodyRotate(4, 0) \* pow(t, 4) + solutionbodyRotate(5, 0) \* pow(t, 5);

robotNext.orientation.pitch = solutionbodyRotate(0, 1) + solutionbodyRotate(1, 1) \* t + solutionbodyRotate(2, 1) \* pow(t, 2) + solutionbodyRotate(3, 1) \* pow(t, 3) + solutionbodyRotate(4, 1) \* pow(t, 4) + solutionbodyRotate(5, 1) \* pow(t, 5);

robotNext.orientation.yaw = solutionbodyRotate(0, 2) + solutionbodyRotate(1, 2) \* t + solutionbodyRotate(2, 2) \* pow(t, 2) + solutionbodyRotate(3, 2) \* pow(t, 3) + solutionbodyRotate(4, 2) \* pow(t, 4) + solutionbodyRotate(5, 2) \* pow(t, 5);

}

}

#### walkingNode.cpp

// 接收规划出的接触状态序列，发布机器人行走中的信息

#include <ros/ros.h>

#include <Eigen/Core>

#include "hit\_spider/hexapod\_RPY.h" //机体姿态

#include "hit\_spider/hexapod\_Base\_Pose.h" //机体位姿

#include "hit\_spider/FeetPosition.h" //六个腿落足点

#include "hit\_spider/hexapod\_State.h" //机器人状态State

#include "hit\_spider/MCTS/trajectory.hh"

#include "hit\_spider/hexapod.hh"

#include <Eigen/Dense>

bool msgFlag = false;

std::vector<hit\_spider::hexapod\_State> statesList;

void supportStateCallback(const hit\_spider::hexapod\_State::ConstPtr &msg)

{

if (msgFlag == false)

{

statesList.push\_back(\*msg);

hit\_spider::hexapod\_State tmp((\*msg));

std::cout << "Received" << std::endl;

ROS\_INFO("当前statesList的存储值为:%d", (int)statesList.size());

ROS\_INFO("接收到消息,质心位置为:%.4f,%.4f,%.4f", msg->base\_Pose\_Now.position.x, msg->base\_Pose\_Now.position.y, msg->base\_Pose\_Now.position.z);

ROS\_INFO("接收到消息,第一条腿的位置:%.4f,%.4f,%.4f", msg->feetPositionNow.foot[0].x, msg->feetPositionNow.foot[0].y, msg->feetPositionNow.foot[0].z);

ROS\_INFO("接收到消息,当前腿的支撑状态:%d,%d,%d,%d,%d,%d", msg->support\_State\_Now[0], msg->support\_State\_Now[1], msg->support\_State\_Now[2], msg->support\_State\_Now[3], msg->support\_State\_Now[4], msg->support\_State\_Now[5]);

if ((\*msg).remarks.data == "end\_flag")

{

msgFlag = true;

std::cout << std::endl

<< std::endl;

}

}

}

// 生成行走轨迹,且发送出去

void publishWalkingTrajctory(Hexapod &hexapod, std::vector<hit\_spider::hexapod\_State> statesL\_, visualization\_msgs::Marker \*footTrajectory, ros::Publisher &footTrajectory\_marker\_pub)

{

for (int j = 0; j < (int)statesL\_.size() - 1; ++j)

{

// 计算插值曲线的参数

Eigen::Matrix<float, 7, 18> solutionleg;

solutionleg = trajectory::solution\_leg(statesL\_[j], statesL\_[j + 1]);

Eigen::Matrix<float, 6, 3> solutionbody;

solutionbody = trajectory::solution\_body(statesL\_[j].base\_Pose\_Now, statesL\_[j + 1].base\_Pose\_Now);

Eigen::Matrix<float, 6, 3> solutionbodyRotate;

solutionbodyRotate = trajectory::solution\_body\_rotation(statesL\_[j].base\_Pose\_Now, statesL\_[j + 1].base\_Pose\_Now);

float framNum = 100;

for (float i = 0.1; i < framNum + 1; i++) // 一次运动中插入多少帧 此处选取100

{

hit\_spider::hexapod\_State hexapodState = statesL\_[j + 1];

trajectory::body\_assignment(hexapodState.base\_Pose\_Now, solutionbody, solutionbodyRotate, i / framNum); // 用i来充当时间

trajectory::leg\_assignment(hexapodState, hexapodState.feetPositionNow, solutionleg, i / framNum);

for (int num\_leg = 0; num\_leg < 6; ++num\_leg)

{

footTrajectory[num\_leg].points.push\_back(hexapodState.feetPositionNow.foot[num\_leg]);

footTrajectory\_marker\_pub.publish(footTrajectory[num\_leg]);

}

hexapod.IK\_Robot(hexapodState);

ros::Duration(0.015).sleep();

}

ros::Duration(0.5).sleep();

}

}

int main(int argc, char \*\*argv)

{

ros::init(argc, argv, "walkingNode");

ros::NodeHandle node;

ros::Subscriber stateSub = node.subscribe("supportStateTopic", 100, supportStateCallback);

ros::Publisher footTrajectory\_marker\_pub = node.advertise<visualization\_msgs::Marker>("footTrajectory", 30);

// 创建足端轨迹Marker

visualization\_msgs::Marker footTrajectory[6];

for (int i = 0; i < 6; ++i)

{

footTrajectory[i].header.frame\_id = "/odom";

footTrajectory[i].ns = std::string("footTrajectory") + std::to\_string(i + 1);

footTrajectory[i].action = visualization\_msgs::Marker::ADD;

footTrajectory[i].pose.orientation.w = 1.0;

footTrajectory[i].type = visualization\_msgs::Marker::LINE\_STRIP;

footTrajectory[i].lifetime = ros::Duration(2.5);

footTrajectory[i].scale.x = 0.02;

footTrajectory[i].color.r = 0.99;

footTrajectory[i].color.g = 0.0;

footTrajectory[i].color.b = 0.0;

footTrajectory[i].color.a = 1.0;

}

int rosRate = 100; // 这个得快一点，才能及时更新作动器是否在运动

ros::Rate rate(rosRate);

Hexapod hexapod(node);

while (node.ok())

{

// 步态运动

if (msgFlag == true)

{

publishWalkingTrajctory(hexapod, statesList, footTrajectory, footTrajectory\_marker\_pub);

statesList.clear();

msgFlag = false;

}

ros::spinOnce();

rate.sleep();

}

return 0;

};

#### main.cpp

// 蒙特卡洛搜索树主程序

#include "hit\_spider/MCTS/MTree.hh"

#define num\_Search\_One\_Step 500

static void print\_once()

{

static bool has\_done = false;

if (!has\_done)

{

std::cout << std::endl;

std::cout << " N:"

<< "当前根节点访问的次数 / 滑动根节点前的剩余访问次数" << std::endl;

std::cout << " Dx:"

<< "滑动根节点的位置" << std::endl;

std::cout << " |->Ex:"

<< "当前根节点最远拓展距离（相对于根节点）" << std::endl;

std::cout << " MaxN:"

<< "搜索树最远拓展位置" << std::endl;

std::cout << " MaxS:"

<< "搜索树最远仿真距离" << std::endl;

std::cout << " x:"

<< "当前模拟节点的位置" << std::endl;

std::cout << " sim:"

<< "当前节点向后仿真距离" << std::endl;

has\_done = true;

}

}

int main(int argc, char \*argv[])

{

setlocale(LC\_ALL, "");

ros::init(argc, argv, "main\_planning\_node");

ros::NodeHandle nh\_;

ros::Duration(0.5).sleep(); // 等待其他的程序启动

// 发布消息以绘制支撑状态

ros::Publisher supportStatePub = nh\_.advertise<hit\_spider::hexapod\_State>("supportStateTopic", 500, true); // supportStateTopic

// ------可调参赛数

// rand函数的随机发生器 ---> 随机选择初始state的可复现性

srand((unsigned)time(NULL));

// srand(10);

static int depth\_count = num\_Search\_One\_Step;

// ------可调参赛数

// 初始化一个state

hit\_spider::hexapod\_State hexapodState;

init\_hexapodState(hexapodState);

// 初始化蒙特卡洛搜索树对象

MNode\_ptr nodeStart = std::make\_shared<MNode>();

nodeStart->element = hexapodState;

nodeStart->alternativeStates = planning::get\_NextState\_list(nodeStart->element); // 为初始节点添加备选状态

// 初始化蒙特卡洛搜索树

MTree tree(nodeStart);

BestChiledParameter PAR;

PAR.C = 1.0f;

PAR.w1 = 3.0f; // 向后模拟仿真距离的权重

PAR.w2 = 1.0f; // 平均步长的权重

PAR.simStepNum = 20;

BestChiledParameter PAR2;

PAR2.C = 0.0f;

PAR2.w1 = PAR.w1;

PAR2.w2 = PAR.w2;

tree.Set\_PAR(PAR);

tree.Set\_PAR2(PAR2);

while (1)

{

// 1.MCTS蒙特卡洛搜索树选择阶段，选择一个节点并进行拓展

MNode\_ptr tmpNode = tree.MCTS\_SelectionAndExpansion();

// 没有有效节点可以扩展，机器人已经卡死了

if (tmpNode == nodeStart) // nodeStart是根节点

{

ROS\_ERROR("机器人初始位置就是卡死的,输入一个字符结束吧");

getchar();

}

// 2.对选择的节点进行模拟仿真，获得节点得分，获得最大仿真距离

std::vector<hit\_spider::hexapod\_State> sequenceState = tree.MCTS\_Simulation(tmpNode);

// 3.反向传播得分情况、节点访问次数情况

bool Flag\_ = tree.MCTS\_Backpropagation(tmpNode);

// 更新最远位置信息（在外部就存放了最远仿真距离，那在节点内部还需要存放这个值吗）

static float maxSimX = 0.0f; // MCTS搜索到的最远位置

static float maxNodeX = 0.0f; // MCTS搜索树当前拓展到的最远位置

if (Flag\_)

{

if (maxSimX < tmpNode->simMaxDis)

{

maxSimX = tmpNode->simMaxDis;

}

if (maxNodeX < tmpNode->element.base\_Pose\_Now.position.x)

{

maxNodeX = tmpNode->element.base\_Pose\_Now.position.x;

}

}

print\_once();

std::cout << "\r"

<< "N:" << std::setw(5) << tree.root->visitTimes << "/" << --depth\_count << " Dx:" << std::setw(5) << tree.Sliding\_New\_root->element.base\_Pose\_Now.position.x << " |->Ex:" << std::setw(10) << maxNodeX - tree.Sliding\_New\_root->element.base\_Pose\_Now.position.x

<< " MaxN:" << std::setw(10) << maxNodeX << " MaxS:" << std::setw(10) << maxSimX << " x:" << std::setw(10) << tmpNode->element.base\_Pose\_Now.position.x << " sim:" << std::setw(10) << tmpNode->simMaxDis << std::flush;

// 移动根节点：当前根节点访问次数超过了depth\_count 或者 搜索树当前拓展的及最远节点距离滑动根节点超过0.8m

if ((depth\_count <= 0) || (maxNodeX - tree.Sliding\_New\_root->element.base\_Pose\_Now.position.x > 0.8f))

{

// 找出最佳子节点，然后移动根节点到该子节点

MNode\_ptr sliding\_node = tree.findBestChild\_Sliding\_Root();

// 剪枝，最优子节点的父节点其他的儿子节点全部舍弃

sliding\_node->Parent->children.clear();

sliding\_node->Parent->children.insert(sliding\_node);

tree.Sliding\_New\_root = sliding\_node; // 移动根节点

depth\_count = num\_Search\_One\_Step;

}

// 结束的终止条件：当前根节点最大访问次数超过，不用终止，一直向后寻找就行

// if (((tree.root->element.visitTimes > 100) && (maxSimX - maxNodeX < 0.2)) || (stopFlag\_ == false) || maxNodeX > 1)

if (maxNodeX > 7.2) // 10.2m 6.99m 2.99m

{

ROS\_WARN("找到结果了!!!\n输入一个字符继续");

getchar();

// 记录解序列

std::vector<hit\_spider::hexapod\_State> solutionStates;

while (1)

{

// 向前不断寻找父节点

solutionStates.push\_back(tmpNode->element);

std::cout << tmpNode << std::endl;

tmpNode = tmpNode->Parent;

if (tmpNode == nullptr) // 到达最根节点

{

break;

}

}

std::cout << "求解出来的简化前的序列长度:\t" << solutionStates.size() << std::endl;

std::reverse(solutionStates.begin(), solutionStates.end());

(solutionStates.rbegin())->remarks.data = "end\_flag";

while (1)

{

std::cout << "press button to pubulish:" << std::endl;

getchar();

for (int i = 0; i < (int)solutionStates.size(); i++)

{

supportStatePub.publish(solutionStates[i]);

std::cout << "x:" << solutionStates[i].base\_Pose\_Now.position.x << std::endl;

ros::Duration(0.05).sleep();

}

std::cout << "是否还想再运行一次?(NO退出)" << std::endl;

std::string str;

std::cin >> str;

if (str == "NO")

{

std::cout << "结束了，只能重新运行了" << std::endl;

break;

}

}

return 0;

}

}

return 1;

}

### 可视化文件

#### base.xacro

<?xml version="1.0"?>

<robot xmlns:xacro="http://ros.org/wiki/xacro">

<!-- ================================ Robot Base ================================ -->

<!-- The xacro macro xacro: EISpider\_base contains: base and trunk -->

<xacro:macro name="EISpider\_base" >

<!-- Floating-base Joint -->

<joint name="jnt\_floating\_base" type="fixed">

<origin xyz="0 0 0" rpy="0 0 ${PI/2}"/>

<parent link="link\_base"/>

<child link="link\_trunk"/>

</joint>

<!-- Links -->

<!-- Footprint link -->

<link name="link\_base">

<visual>

<geometry>

<cylinder length="0.01" radius="0.01"/>

</geometry>

<material name="bbbb">

<color rgba="0.8 0.8 0.8 1"/>

</material>

</visual>

</link>

<!-- Base link -->

<link name="link\_trunk">

<visual>

<origin xyz="-0.4264 -0.48 -0.07" rpy="${PI/2} 0 ${PI/2}"/>

<geometry>

<mesh filename="package://hit\_spider/model/meshes/base\_1.stl" scale="0.001 0.001 0.001"/>

</geometry>

<material name="bbbb"/>

</visual>

<visual>

<origin xyz=" -0.350 -0.315 0.22" rpy="${PI/2} 0 ${PI/2}"/>

<geometry>

<mesh filename="package://hit\_spider/model/meshes/base\_2.stl" scale="0.001 0.001 0.001"/>

</geometry>

<material name="bbbb"/>

</visual>

<collision>

<origin xyz="0 0 0.1" rpy="0 0 0"/>

<geometry>

<cylinder length="0.3" radius="0.3"/>

</geometry>

</collision>

<inertial>

<origin xyz="0 0 0.1" rpy="0 0 0"/>

<mass value="${base\_mass}"/>

<inertia

ixx="${base\_ixx}" ixy="0.0" ixz="0.0"

iyy="${base\_iyy}" iyz="0.0"

izz="${base\_izz}"/>

</inertial>

</link>

<!-- <link name="camera\_link">

<visual>

<origin xyz=" 0 0 0 " rpy="0 0 ${PI/2}" />

<geometry>

<mesh filename="/home/lzy/t\_p/src/hitSpider/meshs/sensors/kinect.dae"/>

</geometry>

<material name="black">

<color rgba="0 0 0 0.95"/>

</material>

</visual>

</link>

<joint name="camera\_joint" type="fixed">

<origin xyz="0.0 0.25 0.32" rpy="0 0 ${PI/2}"/>

<parent link="link\_base"/>

<child link="camera\_link"/>

</joint> -->

<!-- <link name="laser\_link">

<visual>

<origin xyz=" 0 0 0 " rpy="0 0 0" />

<geometry>

<mesh filename="/home/lzy/t\_p/src/hitSpider/meshs/sensors/kinect.dae"/>

</geometry>

<material name="black">

<color rgba="0 0 0 0.95"/>

</material>

</visual>

</link>

<joint name="camera\_joint" type="fixed">

<origin xyz="0.0 0.25 0.32" rpy="0 0 ${PI/2}"/>

<parent link="link\_base"/>

<child link="camera\_link"/>

</joint> -->

<!--/IMU sensor gazebo plugin-->

<xacro:include filename="$(find hit\_spider)/model/xacro/base/base.gazebo.xacro"/>

<xacro:IMUsensor load\_imu\_sensor="true" link\_name="link\_base" x="0" y="0" z="0"/>

</xacro:macro>

</robot>

#### leg.xacro

<?xml version="1.0"?>

<robot xmlns:xacro="http://ros.org/wiki/xacro">

<!--leg xacro-->

<xacro:macro name="EISpider\_Leg" params="LegName leg\_x leg\_y leg\_yaw">

<link name="link\_${LegName}\_1">

<visual>

<origin xyz="-0.065 0.080 -0.080" rpy="${PI/2} 0 0"/>

<geometry>

<mesh filename="package://hit\_spider/model/meshes/leg1.stl" scale="0.001 0.001 0.001"/>

</geometry>

<material name="xccc">

<color rgba="0.3 0.3 0.3 1"/>

</material>

</visual>

<inertial>

<origin xyz="${link1\_length/2} 0 0" rpy="0 0 0"/>

<mass value="${link1\_mass}"/>

<inertia

ixx="${link1\_ixx}" ixy="0.0" ixz="0.0"

iyy="${link1\_iyy}" iyz="0.0"

izz="${link1\_izz}"/>

</inertial>

</link>

<link name="link\_${LegName}\_2">

<visual>

<origin xyz="-0.086 0.086 0.065" rpy="0 ${PI/2} ${-PI/2}"/>

<geometry>

<mesh filename="package://hit\_spider/model/meshes/leg2.stl" scale="0.001 0.001 0.001"/>

</geometry>

<material name="sssss">

<color rgba="0.914 0.914 0.847 1"/>

</material>

</visual>

<inertial>

<origin xyz="${link2\_length/2} 0 0" rpy="0 0 0"/>

<mass value="${link2\_mass}"/>

<inertia

ixx="${link2\_ixx}" ixy="0.0" ixz="0.0"

iyy="${link2\_iyy}" iyz="0.0"

izz="${link2\_izz}"/>

</inertial>

</link>

<link name="link\_${LegName}\_3">

<visual>

<origin xyz="0.373 -0.065 -0.085" rpy="0 0 ${PI/2}"/>

<geometry>

<mesh filename="package://hit\_spider/model/meshes/leg3.stl" scale="0.001 0.001 0.001"/>

</geometry>

<material name="sssss"/>

</visual>

<visual>

<origin xyz="0.40 -0.036 -0.036" rpy="0 0 0"/>

<geometry>

<mesh filename="package://hit\_spider/model/meshes/foot.stl" scale="0.001 0.001 0.001"/>

</geometry>

</visual>

<collision>

<origin xyz="${link3\_length/2} 0 0" rpy="0 0 0"/>

<geometry>

<box size="${link3\_length} 0.05 0.05"/>

</geometry>

</collision>

<inertial>

<origin xyz="${link3\_length/2} 0 0" rpy="0 0 0"/>

<mass value="${link3\_mass}"/>

<inertia

ixx="${link3\_ixx}" ixy="0.0" ixz="0.0"

iyy="${link3\_iyy}" iyz="0.0"

izz="${link3\_izz}"/>

</inertial>

</link>

<link name="link\_${LegName}\_foot">

<visual>

<origin xyz="0 0 0" rpy="0 0 0"/>

<geometry>

<!--cylinder length="0.1" radius="0.05"/-->

<sphere radius="0.03"/>

</geometry>

</visual>

<collision>

<origin xyz="0 0 0" rpy="0 0 0"/>

<geometry>

<!--cylinder length="0.1" radius="0.05"/-->

<sphere radius="0.03"/>

</geometry>

</collision>

<inertial>

<origin xyz="0 0 0" rpy="0 0 0"/>

<mass value="0.01"/>

<inertia

ixx="0.01" ixy="0.0" ixz="0.0"

iyy="0.01" iyz="0.0"

izz="0.01"/>

</inertial>

</link>

<!--joints-->

<joint name="joint\_${LegName}\_1" type="revolute">

<parent link="link\_trunk"/>

<child link="link\_${LegName}\_1"/>

<origin xyz="${leg\_x} ${leg\_y} 0" rpy="0 0 ${leg\_yaw}"/>

<axis xyz="0 0 1"/>

<limit effort="350.0" lower="-${PI/2}" upper="${PI/2}" velocity="4.0"/>

<dynamics damping="30"/>

</joint>

<joint name="joint\_${LegName}\_2" type="revolute">

<parent link="link\_${LegName}\_1"/>

<child link="link\_${LegName}\_2"/>

<origin xyz="${link1\_length} 0 0" rpy="${PI/2} 0 0"/>

<axis xyz="0 0 1"/>

<limit effort="500.0" lower="-${PI/3}" upper="${PI/3}" velocity="4.0"/>

<dynamics damping="30"/>

</joint>

<joint name="joint\_${LegName}\_3" type="revolute">

<parent link="link\_${LegName}\_2"/>

<child link="link\_${LegName}\_3"/>

<origin xyz="${link2\_length} 0 0" rpy="0 0 ${-PI/2}"/>

<axis xyz="0 0 1"/>

<limit effort="500.0" lower="-${PI/2}" upper="${PI/2}" velocity="4.0"/>

<dynamics damping="30"/>

</joint>

<joint name="joint\_${LegName}\_ft\_sensor" type="fixed">

<parent link="link\_${LegName}\_3"/>

<child link="link\_${LegName}\_foot"/>

<origin xyz="${link3\_length} 0 0" rpy="0 ${PI/2} 0"/>

<axis xyz="0 0 1"/>

</joint>

<xacro:include filename="$(find hit\_spider)/model/xacro/leg/leg.transmission.xacro"/>

<xacro:leg\_transmission LegName="${LegName}"/>

<xacro:include filename="$(find hit\_spider)/model/xacro/leg/leg.gazebo.sensor.xacro"/>

<xacro:legFTsensor LegName="${LegName}"/>

</xacro:macro>

</robot>

#### common.xacro

<?xml version="1.0"?>

<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="robot param">

<!--pi-->

<xacro:property name="PI" value="3.1415926535897931"/>

<!--leg roots' position-->

<xacro:property name="leg\_lf\_x" value="0.3464" />

<xacro:property name="leg\_lm\_x" value="0.0" />

<xacro:property name="leg\_lh\_x" value="-0.3464" />

<xacro:property name="leg\_rh\_x" value="-0.3464" />

<xacro:property name="leg\_rm\_x" value="0" />

<xacro:property name="leg\_rf\_x" value="0.3464" />

<xacro:property name="leg\_lf\_y" value="0.2" />

<xacro:property name="leg\_lm\_y" value="0.4" />

<xacro:property name="leg\_lh\_y" value="0.2" />

<xacro:property name="leg\_rh\_y" value="-0.2" />

<xacro:property name="leg\_rm\_y" value="-0.4" />

<xacro:property name="leg\_rf\_y" value="-0.2" />

<xacro:property name="leg\_lf\_yaw" value="${PI/6}" />

<xacro:property name="leg\_lm\_yaw" value="${PI/2}" />

<xacro:property name="leg\_lh\_yaw" value="${5\*PI/6}" />

<xacro:property name="leg\_rh\_yaw" value="${-5\*PI/6}" />

<xacro:property name="leg\_rm\_yaw" value="${-PI/2}" />

<xacro:property name="leg\_rf\_yaw" value="${-PI/6}" />

<!--leg links' length-->

<xacro:property name="link1\_length" value="0.18" />

<xacro:property name="link2\_length" value="0.5" />

<xacro:property name="link3\_length" value="0.5" />

<!--mass-->

<xacro:property name="base\_mass" value="121.8957" />

<xacro:property name="link1\_mass" value="3.6031" />

<xacro:property name="link2\_mass" value="21.9855" />

<xacro:property name="link3\_mass" value="7.4018" />

<!--inerials-->

<xacro:property name="base\_ixx" value="5.0930" />

<xacro:property name="base\_iyy" value="5.0930" />

<xacro:property name="base\_izz" value="8.2937" />

<xacro:property name="link1\_ixx" value="0.011212" />

<xacro:property name="link1\_iyy" value="0.03169" />

<xacro:property name="link1\_izz" value="0.03169" />

<xacro:property name="link2\_ixx" value="0.07020" />

<xacro:property name="link2\_iyy" value="0.7228" />

<xacro:property name="link2\_izz" value="0.7228" />

<xacro:property name="link3\_ixx" value="0.02290" />

<xacro:property name="link3\_iyy" value="0.14949" />

<xacro:property name="link3\_izz" value="0.14949" />

</robot>

#### EISpider.xacro

<?xml version="1.0"?>

<robot name="EISpider" xmlns:xacro="http://www.ros.org/wiki/xacro">

<!-- The following included files set up definitions of parts of the robot body -->

<xacro:include filename="$(find hit\_spider)/model/xacro/common.xacro"/>

<!-- EISpider trunk -->

<xacro:include filename="$(find hit\_spider)/model/xacro/base/base.xacro"/>

<xacro:EISpider\_base/>

<!-- EISpider legs -->

<xacro:include filename="$(find hit\_spider)/model/xacro/leg/leg.xacro"/>

<xacro:EISpider\_Leg LegName="lf" leg\_x="${leg\_lf\_x}" leg\_y="${leg\_lf\_y}" leg\_yaw="${leg\_lf\_yaw}"/>

<xacro:EISpider\_Leg LegName="lm" leg\_x="${leg\_lm\_x}" leg\_y="${leg\_lm\_y}" leg\_yaw="${leg\_lm\_yaw}"/>

<xacro:EISpider\_Leg LegName="lh" leg\_x="${leg\_lh\_x}" leg\_y="${leg\_lh\_y}" leg\_yaw="${leg\_lh\_yaw}"/>

<xacro:EISpider\_Leg LegName="rf" leg\_x="${leg\_rf\_x}" leg\_y="${leg\_rf\_y}" leg\_yaw="${leg\_rf\_yaw}"/>

<xacro:EISpider\_Leg LegName="rm" leg\_x="${leg\_rm\_x}" leg\_y="${leg\_rm\_y}" leg\_yaw="${leg\_rm\_yaw}"/>

<xacro:EISpider\_Leg LegName="rh" leg\_x="${leg\_rh\_x}" leg\_y="${leg\_rh\_y}" leg\_yaw="${leg\_rh\_yaw}"/>

<xacro:include filename="$(find hit\_spider)/model/gazebo/gazebo.urdf.xacro"/>

</robot>

### 编译配置文件

#### CMakeLists.txt

cmake\_minimum\_required(VERSION 3.0.2)

project(hit\_spider)

SET(CMAKE\_BUILD\_TYPE Debug)

add\_definitions("-Wall") #启用g++检测代码警告

find\_package(catkin REQUIRED COMPONENTS

roscpp

rospy

std\_msgs

geometry\_msgs

sensor\_msgs

visualization\_msgs

tf2

tf2\_ros

tf2\_geometry\_msgs

message\_generation

grid\_map\_core

grid\_map\_ros

grid\_map\_cv

grid\_map\_filters

grid\_map\_loader

grid\_map\_msgs

grid\_map\_octomap

grid\_map\_rviz\_plugin

grid\_map\_visualization

)

## System dependencies are found with CMake's conventions

#Eigen3路径

find\_package(Eigen3 REQUIRED)

if (EIGEN3\_INCLUDE\_DIR)

message(STATUS "Found Eigen3")

include\_directories("${EIGEN3\_INCLUDE\_DIR}") #包含Eigen库头文件

endif()

#GLPK路径

set(GLPK\_INCLUDE\_DIR "/usr/local/include")

set(GLPK\_LIBRARY\_DIR "/usr/local/lib")

include\_directories("${GLPK\_INCLUDE\_DIR}") #包含GLPK头文件

link\_directories("${GLPK\_LIBRARY\_DIR}") #GLPK的链接库

#cdd库

set(CDD\_INCLUDE\_DIR "/usr/include/cdd")

set(CDD\_LIBRARY\_DIR "/usr/lib/x86\_64-linux-gnu/libcdd.so")

include\_directories("${CDD\_INCLUDE\_DIR}") #包含cdd的头文件

link\_directories("${CDD\_LIBRARY\_DIR}") #cdd库的链接库

add\_message\_files(FILES hexapod\_RPY.msg hexapod\_Base\_Pose.msg FeetPosition.msg hexapod\_State.msg)

generate\_messages(

DEPENDENCIES

geometry\_msgs std\_msgs

)

catkin\_package(

INCLUDE\_DIRS include

# LIBRARIES hit\_spider

CATKIN\_DEPENDS message\_runtime

DEPENDS EIGEN3

)

include\_directories(include ${catkin\_INCLUDE\_DIRS})

##---规划库的构建---##

set(INCLUDE\_DIR "${PROJECT\_SOURCE\_DIR}/include/${PROJECT\_NAME}") #工程的include文件夹

set(SRC\_DIR "${PROJECT\_SOURCE\_DIR}/src") #工程的src文件夹

set(lib\_Bisic bisic\_library)

set(lib\_MCTS dynamic\_MCTS\_library)

set(lib\_RST library\_robot\_state\_transition)

#头文件

set(lib\_Bisic\_headers

${INCLUDE\_DIR}/util.hh

${INCLUDE\_DIR}/hexapod.hh

${INCLUDE\_DIR}/static\_information.hh

)

set(lib\_RST\_headers

${INCLUDE\_DIR}/robot\_state\_transition/solve\_LP\_GLPK.hh

${INCLUDE\_DIR}/robot\_state\_transition/Bretl.hh

${INCLUDE\_DIR}/robot\_state\_transition/my\_cdd.hh

${INCLUDE\_DIR}/robot\_state\_transition/kinematics\_constrain.hh

${INCLUDE\_DIR}/robot\_state\_transition/dynamic\_constrain.hh

)

set(lib\_MCTS\_headers

${INCLUDE\_DIR}/MCTS/planning.hh

${INCLUDE\_DIR}/MCTS/MTree.hh

)

#源文件

set(lib\_Bisic\_sources

${lib\_Bisic\_headers}

${SRC\_DIR}/util.cpp

${SRC\_DIR}/hexapod.cpp

${SRC\_DIR}/static\_information.cpp

)

set(lib\_RST\_sources

${lib\_RST\_headers}

${SRC\_DIR}/robot\_state\_transition/solve\_LP\_GLPK.cpp

${SRC\_DIR}/robot\_state\_transition/Bretl.cpp

${SRC\_DIR}/robot\_state\_transition/my\_cdd.cpp

${SRC\_DIR}/robot\_state\_transition/kinematics\_constrain.cpp

${SRC\_DIR}/robot\_state\_transition/dynamic\_constrain.cpp

)

set(lib\_MCTS\_sources

${lib\_MCTS\_headers}

${SRC\_DIR}/MCTS/planning.cpp

${SRC\_DIR}/MCTS/MTree.cpp

)

add\_library(${lib\_Bisic} SHARED ${lib\_Bisic\_sources}) #构建 基本链接库 整机逆运动学、地图环境落足点

add\_dependencies(${lib\_Bisic} ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

target\_link\_libraries(${lib\_Bisic} ${catkin\_LIBRARIES}) #链接外部库

add\_library(${lib\_RST} SHARED ${lib\_RST\_sources}) #构建 机器人状态转移 动态链接库

add\_dependencies(${lib\_RST} ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

target\_link\_libraries(${lib\_RST} ${lib\_Bisic} libglpk.so ${CDD\_LIBRARY\_DIR} ${catkin\_LIBRARIES}) #链接外部库

add\_library(${lib\_MCTS} SHARED ${lib\_MCTS\_sources}) #构建 蒙特卡洛搜索树算法 动态链接库

add\_dependencies(${lib\_MCTS} ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

target\_link\_libraries(${lib\_MCTS} ${lib\_Bisic} ${lib\_RST} ${catkin\_LIBRARIES}) #链接外部库

##---规划库的构建---##

#------可执行程序------

add\_executable(static\_information\_publisher src/static\_information\_publisher.cpp)

target\_link\_libraries(static\_information\_publisher ${lib\_Bisic})

add\_executable(main src/main.cpp)

target\_link\_libraries(main ${lib\_MCTS})

add\_compile\_options(-fPIC)

#### package.xml

<?xml version="1.0"?>

<package format="2">

<name>hit\_spider</name>

<version>0.0.0</version>

<description>The hit\_spider package</description>

<maintainer email="ptw@todo.todo">ptw</maintainer>

<license>TODO</license>

<buildtool\_depend>catkin</buildtool\_depend>

<build\_depend>roscpp</build\_depend>

<build\_depend>rospy</build\_depend>

<build\_depend>std\_msgs</build\_depend>

<build\_depend>geometry\_msgs</build\_depend>

<build\_depend>sensor\_msgs</build\_depend>

<build\_depend>visualization\_msgs</build\_depend>

<build\_depend>tf2</build\_depend>

<build\_depend>tf2\_ros</build\_depend>

<build\_depend>tf2\_geometry\_msgs</build\_depend>

<build\_depend>message\_generation</build\_depend>

<build\_export\_depend>roscpp</build\_export\_depend>

<build\_export\_depend>rospy</build\_export\_depend>

<build\_export\_depend>std\_msgs</build\_export\_depend>

<build\_export\_depend>geometry\_msgs</build\_export\_depend>

<build\_export\_depend>tf2</build\_export\_depend>

<build\_export\_depend>tf2\_ros</build\_export\_depend>

<build\_export\_depend>tf2\_geometry\_msgs</build\_export\_depend>

<exec\_depend>roscpp</exec\_depend>

<exec\_depend>rospy</exec\_depend>

<exec\_depend>std\_msgs</exec\_depend>

<exec\_depend>geometry\_msgs</exec\_depend>

<exec\_depend>sensor\_msgs</exec\_depend>

<exec\_depend>visualization\_msgs</exec\_depend>

<exec\_depend>tf2</exec\_depend>

<exec\_depend>tf2\_ros</exec\_depend>

<exec\_depend>tf2\_geometry\_msgs</exec\_depend>

<exec\_depend>message\_runtime</exec\_depend>

<!-- The export tag contains other, unspecified, tags -->

<export>

<!-- Other tools can request additional information be placed here -->

</export>

</package>

### 程序启动文件

#### walking.launch

<launch>

<!-- 机器人模型 和 rviz显示 -->

<param name="robot\_description" command="$(find xacro)/xacro $(find hit\_spider)/model/xacro/EISpider.xacro" />

<node pkg="robot\_state\_publisher" type="robot\_state\_publisher" name="robot\_state\_publisher" output="screen" />

<!-- 显示程序 -->

<node pkg="hit\_spider" type="walkingNode" name="walkingNode" output="screen" />

<node pkg="rviz" type="rviz" name="rviz" args="-d $(find hit\_spider)/config/walking\_rviz.rviz" output="screen" />

<!-- 静态坐标变换 -->

<node pkg="hit\_spider" type="static\_information\_publisher" name="static\_information\_publisher" output="screen" />

</launch>

#### planning.launch

<launch>

<!-- MCTS规划程序 -->

<node pkg="hit\_spider" type="main" name="main\_node" output="screen" />

</launch>