Completed

Inverse Kinematics using *ikfast* on a 7 DOF Robotic Arm

Anshul Kanakia

May 13, 2012

Abstract

This paper describes integration and use of the OpenRAVE, *ikfast* module as an inverse kinematics solver for the Correll Lab Arm Manipulator (CLAM arm). It also explains the general working of the Robot Operating System (ROS) in the context of motion planning and control of robotic arms.

1 Introduction

This paper aims to be a suitable tutorial for integration and use of the OpenRAVE *ikfast* module as an inverse kinematics solver, used during motion planning for the Correll Lab Arm Manipulator (CLAM Arm).

Most ik solver software is custom made for a specific robot to address its mechanical constraints. Different classes of ik solvers exist for different types of robots. For example, there are specialized algorithms for solving ik for differential wheeled robots, compared to algorithms for solving ik for robotic arms with varying degrees of freedom, etc. For robotic arms, there are some generalized numerical solvers for forward and inverse kinematics. One such software package is called KDL (Kinematics and Dynamics Library) and is supplied with the OROCOS robot control library. As opposed to *ikfast*, KDL is capable of solving ik solutions for more than 6 degrees of freedom, whereas *ikfast* is restricted to 6 DOF solutions as it applies analytic techniques for computation[1]. Since the CLAM arm is a 7 DOF manipulator, using *ikfast* restricts movement of one of its joints thereby reducing the total area of the valid solution space. But if the correct joint is selected and the workspace for the arm is restricted, as it generally is in many cases, then this restriction is negligible compared to the speedup achieved during motion planning. Compared to KDL, *ikfast* finds solutions in joint space almost 3 orders of magnitude faster [2]. 超TM快

While most inverse kinematics solvers (IK solvers) are numerical in nature, the *ikfast* module analytically solves robot ik equations and generates optimized c++ files [2]. There are some considerable advantages to using *ikfast* compared to traditional, numerical ik solvers, namely :

- Analytical solvers are extremely fast at computing ik solutions. In general, an analytical solver can compute solutions in the order of microseconds as opposed to milliseconds for most numerical solvers.

- Having the analytic solution to the ik problem gives us the ability to study the null space of the solution set, which is not possible with numerical methods.[2]

The CLAM arm uses the Open Motion Planning Library (OMPL) ROS stack for motion planning. The default ik solver included with the OMPL stack, KDL, is part of the OROCOS project, as metioned earlier. In this tutorial, we mean to switch this solver out for our *ikfast* generated analytical solver. IK solvers are used in motion planning during the trajectory verification stage, which will be explained shortly. They are generally called thousands of times a second and hence need to be fast. Motion planning involves finding a set of trajectories from an initial end-effector state to a desired end-effector state through a series of valid rotations and translations called *trajectory*. Once a trajectory is computed, each individual rotation and translation must be verified such that,

OMPL 的IK 默认为 KDL

↘3维空间采样后IK得6轴姿态，从而判断碰撞.

- mechanical constraints do not restrict the arm from being in the desired position,
- there are no collisions with environmental objects,
- there are no collisions with other parts of the arm (self-collisions).

This is known as the trajectory verification stage. Most numerical ik solvers will do environment collision checking first in cartesian space as it is impractical to convert obstacle positions from cartesian space to joint space. These are needless calculations which result in a significant increase in computation time, which should be avoided.

The following section will explaing how to set up *ikfast* with the CLAM arm stack in ROS. In the conclusion section I will detail some future work in this project.

2 Methods

There are two principle methods for setting up OpenRAVE. The first method is a source install. This method is completely independent of the ROS framework. The second method involves setting up the OpenRAVE planning stack in ROS. We will use the first method for two reasons,

1. Since we are only interested in the *ikfast* module of OpenRAVE, downloading the source and isolating just the *ikfast* module is easy.
2. This method is not affected by problems caused due to ROS and other operating system updates.

The source code may be downloaded from the OpenRAVE website. After installation, the two OpenRAVE python modules that are important to us are *ikfast* and *inversekinematics*. The *ikfast* and *inversekinematics* modules are used to generate and test the analytical solution for the given robot model files, respectively. The *inversekinematics* module is recommended since you can generate, test and visualize your results, all from this single module. OpenRAVE allows rigid body and joint specifications of the robot model in two main formats, COLLADA and *robot.xml*. Both specifications are human readable and more information about them may be found online.

Once a robot.xml or COLLADA file is available you can use *ikfast* to generate the inverse kinematic solution c++ file by using the command,

```
openrave.py --database inversekinematics --robot=robot_model.robot.xml
--iktype=rotation3d
```

给机器人描述，生成 IK 函数

where "iktype" is the solver type to build. Currently supported types are, Transform6D, Rotation3D, Translation3D, Direction3D, Ray4D, Lookat3D, TranslationDirection5D, TranslationXY2D, TranslationXYOrientation3D, TranslationLocalGlobal6D, TranslationXAxisAngle4D, TranslationYAxisAngle4D, TranslationZAxisAngle4D, TranslationXAxisAngleZNorm4D, TranslationYAxisAngleXNorm4D, and TranslationZAxisAngleYNorm4D[3]. This command will generate a *.cpp* file with the following global functions [2],

```
// Computes all IK solutions given a end effector coordinates
// and the free joints.
bool ik(const IKReal* eetrans, const IKReal* eerot, const IKReal* pfree,
std::vector<IKSolution>& vsolutions);

// Computes the end effector coordinates givennn the joint values.
// This function is used to double check ik.
void fk(const IKReal* joints, IKReal* eetrans, IKReal* eerot);

// returns the number of free parameters users has to set apriori
int getNumFreeParameters();

// the indices of the free parameters indexed by the chain joints
int* getFreeParameters();

// the total number of indices of the chain
int getNumJoints();

// the size in bytes of the configured number type
int getIKRealSize();

// the ikfast version used to generate this file
const char* getIKFastVersion();

// the ik type ID
int getIKType();

// a hash of all the chain values used for double checking
// that the correct IK is used.
const char* getKinematicsHash();
```

把IK函数包装为 service 以供调用

The *ik(...)* method is most useful from this file and may be wrapped as a ROS message or service (derived from the *kinematics_msgs* ROS package) so that it may be used by the CLAM arm, OMPL arm navigation package for trajectory planning. This has been done in the *clam* ROS stack's *clam_arm_kinematics* package.

3 Conclusion

In this tutorial I discussed the method for setting up OpenRAVE's *ikfast* module with the existing OMPL motion planner for the Correll Lab Arm Manipulator. The end goal of this project is to be able to use *ikfast*, along with OMPL motion planners to quickly compute trajectories for the CLAM arm to navigate to blocks on a table and pick them up. To that extent, the future work of this project involves using the ROS message and services generated by this tutorial to move the dynamixel motors on the physical arm, accurately. There are many external factors to consider before we can achieve this including,

1. camera/kinect sensor input to the system,
2. implementing object detection from sensory input,
3. calibration of the motors with sensor and control inputs,
4. and designing grasping routines for the arm, to name a few.

References

[1] The orocos project. `http://www.orocos.org/wiki/orocos/kdl-wiki`.

[2] Rosen Diankov. ikfast module. `http://openrave.programmingvision.com/en/main/openravepy/ikfast.html`.

[3] Rosen Diankov. inverse kinematics module. `http://openrave.programmingvision.com/en/main/openravepy/databases.inversekinematics.html`.