TECHNISCHE UNIVERSITÄT DRESDEN

FACULTY OF COMPUTER SCIENCE
INSTITUTE OF SOFTWARE AND MULTIMEDIA TECHNOLOGY
CHAIR OF COMPUTER GRAPHICS AND VISUALIZATION
PROF. DR. STEFAN GUMHOLD

Bachelor's Thesis

for obtaining the academic degree
Bachelor of Science

Immersive Exploration of 6D Pose or Configuration Spaces

Shi Baoqi
(Born 1. March 1994 in Wuhan, Mat.-No.: 3935715)

Tutor: Prof. Dr. Stefan Gumhold

Dresden, October 26, 2020

Task Description

Motivation

The pose of a rigid body describes its orientation and position in space. Both parts have three degrees of freedom such that the space of all poses is 6D. In VR applications the poses of controlers and head mounted displays are tracked with a high framerate, in computer vision 6D poses are estimate for objects seen in 2D images, in robotics poses are the target configuration of end effectors that guide inverse kinematics algorithms and finally in geometry processing poses are used to describe the transformation between 3D scans. For the analysis of algorithms that work with poses it is important to visualize 6D pose space.

The goal of this thesis is the design, implementation and evaluation of an immersive pose space exploration technique for VR. The approach should be based on a general oracle function that defines whether an input pose belongs to pose space or not.

Subtasks

Literature review on visualization of multi-dimensional samples, spaces and functions.

Implementation of two oracle functions based on an inverse kinematics (IK) solver.

Implementation of a parallel pose space sampling algorithm and execution of the algorithm on the oracle function for several IK problems.

Design and implementation of two visualizations one for 3D pose space aggregated over orientations and one to show all orientations in pose space for a given 3D location.

Design and implementation of a VR based pose space exporation strategy exploiting the developed visualization.

Evaluation of the developed approaches with respect to performance.

Optional Goals

Implement more oracle functions from another application domain

Implement interface to compare two different pose spaces

Evaluate VR experience with informal user study

Declaration of authorship

I hereby declare that I wrote this thesis on the subject

Immersive Exploration of 6D Pose or Configuration Spaces

independently. I did not use any other aids, sources, figures or resources than those stated in the references. I clearly marked all passages that were taken from other sources and cited them correctly.

Furthermore I declare that – to my best knowledge – this work or parts of it have never before been submitted by me or somebody else at this or any other university.

Dresden, October 26, 2020 Shi Baoqi

Shi Baoqi

Kurzfassung

In der Robotik sind 6D-Pose die Konfiguration von Endeffektoren. Die Erreichbarkeit des Roboters kann durch Berechnung der IK ermittelt werden. Für die Analyse des Arbeitsbereichs des Roboters ist es wichtig, den Posespace zu visualisieren. Es ist also ein Problem, intuitiv anzuzeigen, ob sich eine 6d-Pose im Posespace befindet. In dieser Arbeit stellen wir einen integrierten Ansatz zur Visualisierung und Abtastung von Posen in VR in Echtzeit vor. Der theoretische Hintergrund und verschiedene Ansätze zur Abtastung und Visualisierung des 6D-Posenraums werden diskutiert. Bei der Entwicklung dieser Ideen wird ein Beispiel für eine umfassende Erforschung der VR vorgestellt, das auf die Stärken und Schwächen des bestehenden Projekts hinweist. Am Ende dieser Arbeit werden nützliche Vorschläge für die zukünftige Verbesserung des neu entwickelten Ansatzes gegeben.

Abstract

In robotics, 6D poses are the target configuration of end effectors. The reachability of the robot can be known by calculating IK. For the analysis the workspace of robot, it is important to visualize the pose space. So how to intuitively indicate whether a 6d pose is in the pose space is a problem. In this thesis, we present an integrated approach to visualization and sampling poses in VR in real time. Theoretical background and diffrent approaches of sampling and visualize of 6D pose space will be discust. Developing these ideas, an example of immersive exploration in VR will be presented, which pointing out the strength and weakness of the existing project.At the end of this thesis, useful suggestions will be provided for future enhancement of newly-developed approach.

Contents

1 Introduction **2**

2 Related work **5**

3 Development **7**

3.1 Devices and Frameworks 7

3.2 Model 9

3.2.1 3D Model 9

3.2.2 3D Collision Model 12

3.2.3 Kinematics Model 13

3.3 Rendering 16

3.4 Load transformation of Controller 18

3.5 Sampling and Visualization 20

3.5.1 Sampling and Visualization of a fixed pose 23

3.5.2 Sampling and Visualization for fixed rotation 26

3.5.3 Sampling and Visualization for fixed translation 27

4 Exploration **32**

5 Conclusion **34**

Bibliography **35**

1 Introduction

The pose of a rigid body describes it's orientation and position in space. Both parts have three degrees of freedom(DOF) such that the space of all poses is 6D.

In computer vision, 6d pose is often used to detect objects, and it is often necessary to estimate the 6d pose of the object in a static image.

In robotics 6D poses are the target configuration of end effectors. The position and orientation of the end effectors of the robot arm determine whether in the space the robot reachable. This can also be described by an oracle function,

$$f(x, y, z, \theta_x, \theta_y, \theta_z) \rightarrow \{0(notreachable), 1(reachable)\}$$

In reality, there may be obstacles around the robot, so through collision detection the oracle function can be extend to express reachability and collision:

$$f(x, y, z, \theta_x, \theta_y, \theta_z) \rightarrow \{0(notreachable), 1(reachablewithoutcollision), 2(reachablewithcollision)\}$$

For the analysis the reachability of robot it is important to visualize this oracle function. So how to intuitively indicate whether a 6d pose is in the pose space is a problem.

Since sampling the 6d poses is very time-consuming, so the idea to solve this problem is of integrating the sampling process into the exploration strategy in real-time.

In this thesis, Virtual Reality is used for exploration. Virtual Reality is a computer-generated world that uses three-dimensional images to create a spatial impression in the brain. In VR glasses, spatial environments are created by 360-degree VR applications, in which users can look around and interact with their surroundings.

In this thesis, we present an integrated approach to visualization and sampling poses in VR in real time. By using VR, the users can better define and perceive the rigid body poses. Through the VR controller, users can interact with the environment, sample and display the robot poses they want to study, if a pose is in posespace, the robot joint configuration will also be set up accordingly.

Our approach is use 3 arrows to accurately indicate the position and orientation of a end effector with

fixed pose or rotation, and with help to certain convex polyhedra like Goldberg polyhedrons to sampling and visualize the poses with a fixed position.

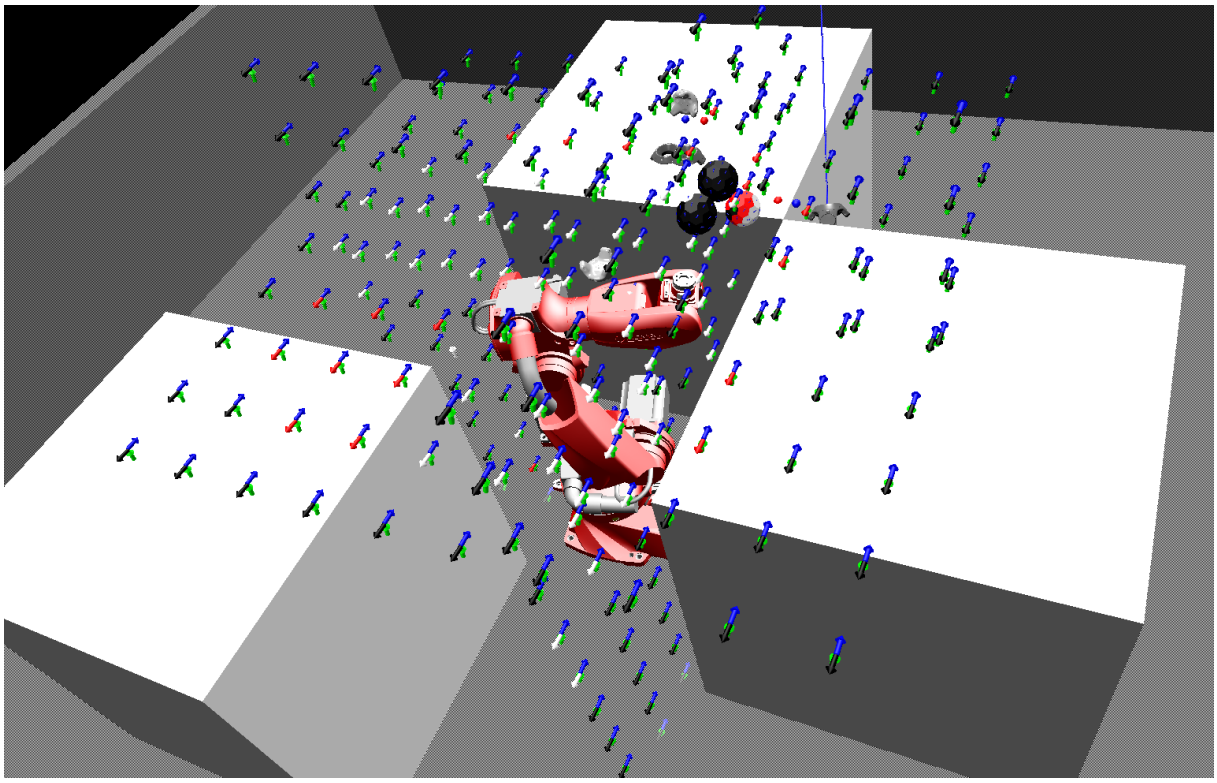


Figure 1.1: Example for our approach to visualize the 6D pose space

Figur 1.1 is a example about our approach to visualize the 6D pose space.

The sampled rigid poses are shown with arrow groups (build from 3 arrows) or Goldberg polyhedrons with different color, where green and blue arrows are for reference and the remaining arrow or surface are for indicate result of oracle functions with different color.

St1te 1 white: reachable with out collision

St2te 2 red: reachable with collision

St3te 3 black: not rechable.

white for reachable, red for collision black for not reachable.

The structure of the paper is as follows. In the first section, I will simply elaborate the theoretical background and some approach of sampling and visualize of 6D pose space. Some instructive studies and related literature will be introduced in this part as well. Afterwards, the implementation of our approach will be presented in greater detail. In a further section, an example of immersive exploration in VR will be given. Developing these ideas, we will be pointing out the strength and weakness of the

existing project. Lastly, conclusions are drawn in the final section to sum up the paper. Meanwhile, useful suggestions will be provided for future enhancement of the approach.

2 Related work

In order to obtain posespace, many different methods have been developed. Kee and Karwowski proposed an analysis method to generate a 3D reachability representation.[KK02] Another approach of generating posespace data is to randomly sample joint values while using forward kinematics to determine the posture of the end effector.[BPW93] Both of these two method were time-consuming when calculating the complete workspace of the robot.

In robotics, inverse kinematics uses kinematic equations to determine joint parameters to provide the required configuration (position and rotation) for each end effector of the robot.[Pau81] By implementing the inverse kinematics, the 6D pose space can also be sampled. The voxel position represents the position of the end effector.

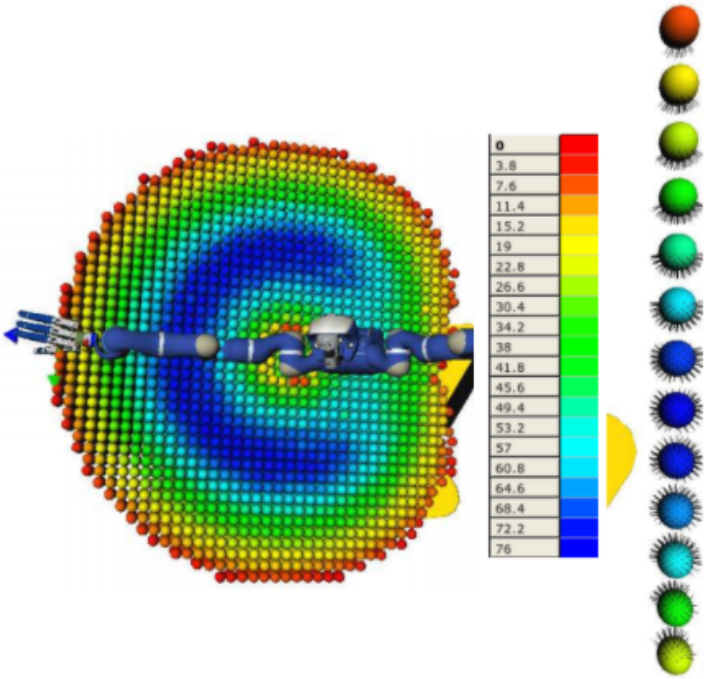


Figure 2.1: Capturing Robot Workspace Structure: Representing Robot Capabilities

Zacharias, Christoph Borst and Gerd Hirzinger provide a way to use a colored voxel to visualize the pocespace.[ZBH07] This visualize method has been showed in figur 2.1. The voxel position represents

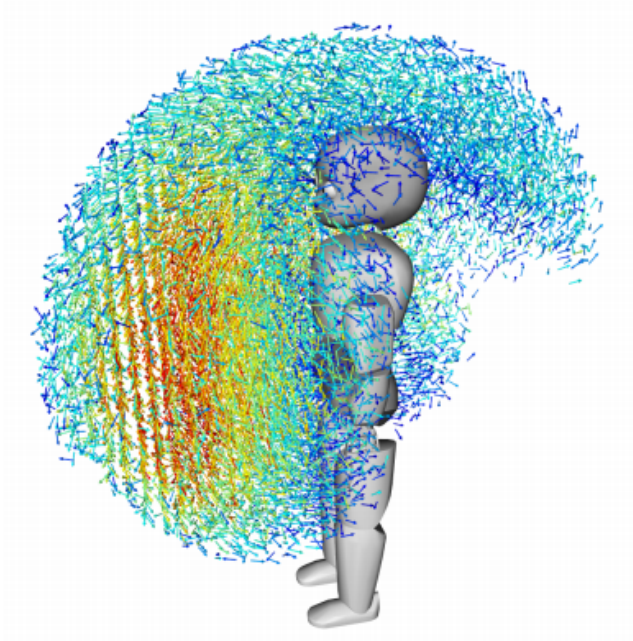


Figure 2.2: Manipulabitity Analyse

the position of the end effector. By sampling N rotation at each position of end effector, n poses which in the workspace can be obtained. Divide the number of samples in the posespace n by N to get a percentage number a, this percentage is correspond to a color.

Nikolaus Tamim and Dillmann used a single arrow instead of a ball for represent the reachability. [VAD13]. Figur 2.1 showed their way of expression of posespace. The arrow can indicate the direction of the endeffector and the rotation that can be performed. They use color to indicate the angle of spin.

Both methods have their own limitations, neither of them can accurately express the 7 data of oracle function, especially rotation and spin are difficult to express accurately by color. So we invented a method to sampling and visualize 6d posespace in our project, through this method, 6D posespace can be clearly expressed in vr.

3 Development

Posespace is a collection of all poses that the end effector of a robot can reach. This approach is based on a general oracle function, which helps scientist to clarify whether an input pose belongs to pose space. In VR, the pose of the robot could be sampled more conveniently and accurately.

An innovative method for sampling the 6D poses in VR (real-time) and visualization for 6D posespace of the robotic arm have been developed in this paper. The detailed code can be found in `github.com/Orooz/pose_Vis`.

Figure 3.1: Procedure flow chart

Figure 3.1 shows the basic design ideas of the program,

step 1 Load the robot model file

step 2 Render basic scenes and models

step 3 Read the position and rotation of the controller as the transformation information of the robot end effector

step 4 Sample Poses based on the transformation

step 5 Visualize 6D Posespace

This chapter will elaborate on them.

3.1 Devices and Frameworks

Cgv framework was introduces to develop the VR application. Cgv framework is an opensource framework that provided by the chair of Computer Graphics and Visualization at TU Dresden. This framework is developer-friendly and supports OpenGL rendering. It provides a Viewer named cgv_viewer to dis-

play the scene on the computer screen and controls variables through the GUI. In addition, if a VR device is connected, the Scene will be displayed in the VR device. Users can directly control the parameters through the controller.

The robot is a rigid multibody system, which is composed of many rigid bodies. These rigid bodies are usually called "links", which connected by "joint". The joint can manipulate the translation and rotation of the link. A common problem in robotics is how to calculate the joint configuration of the robotic arm when the end of the robotic arm (also known as the end effector) is at the target position and direction. The mathematical process of calculating the variable joint parameters is also called Inverse kinematics[Bus04].

To calculate Inverse Kinematics, we use a framework named "Roboticslibrary".[RG17] Roboticslibrary is an opensources library in robotics research, it contains motion planning algorithms and rigid-body dynamics calculations. RL can be run on different Operating Systems, for example, Windows and Linux. Different from the Robot Operating System[QFFL](ROS, a probably most active middleware and software framework in robotics research), RL is a homogeneous library with full real-time support. Due to the high frame rate required by VR, real-time computing is necessary, which is not well supported by ROS. In addition, RL takes up less space than the ROS (150MB vs 4GB), which could make the development process more convenient and flexible, Since both RL and CGV can be rendered by using Opengl, which is a specification of a cross-platform and cross-programming language programming interface (API) for developing 2D and 3D computer graphics applications. So RL and cgv are compatible with each other.

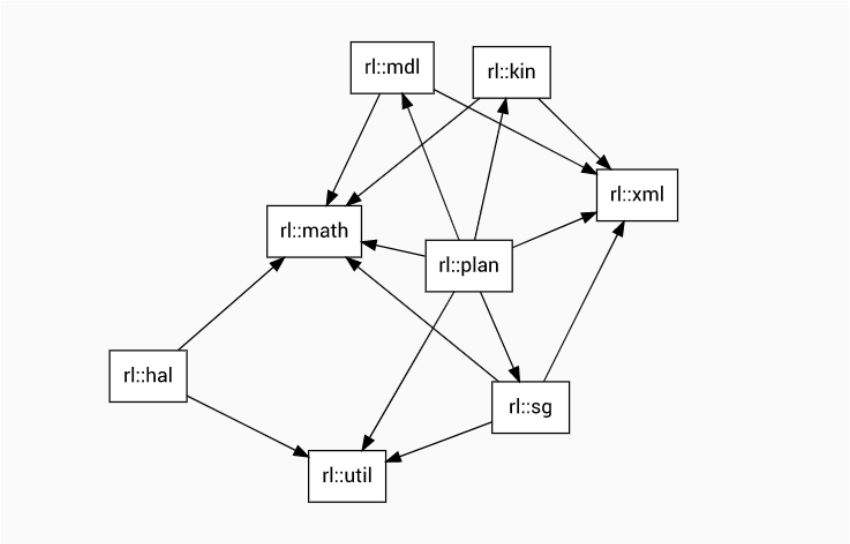


Figure 3.2: The Robotics Library is structured as a hierarchical set of components.

Figure 3.2 shows the structure of RL. 5 of these components will be used in our project:

- rl::math is used to calculate the matrix, which can easily manipulate the translation and rotation of objects
- rl::mdl is used to load and edit the kinematics model of the object
- rl::xml is used to read the description file of the object model, because the description files are saved in the xml file type
- rl::sg is used to load the 3D model of the object
- rl::kin is used to calculate inverse kinematics to get oracle function

Detailed application of those 5 components in projects would be discussed in later sections.

3.2 Model

As mentioned in the previous section, import of the robot model is the first step of the whole project. The RL library contains more than 20 different types of robot models from companies such as kuka, universe, mekabot, and universal. The model can also be designed by the user. For a robot model, there are 3 types of different model files, which are:

- 3D model: used to visualize the robot
- 3D collision model: used for collision detection
- Kinematics model: used to describe the configuration information of the robot joint

In this article, Comau racer 999 was selected for the resarch process. Racer 999 is a six-axis articulated unit made by company Comau. This robot is suited for applications (including assembly, handling, machine tooling, packaging and more) with a reach of 999mm. Users can replace Racer 999 with other robot models, or customize their own robots as well.

Figure 3.3 shows the file structure of different models. These models can be loaded in *init()* method.

3.2.1 3D Model

The left part of the figure 3.3 shows the structure of the 3D model. A 3D model consists of a scene file *boxes.wrl*, a folder containing a robot model *comau-racer-999*, a 3D model file with scene *comau-racer-999_boxes.wrl*, and a 3D model description file with scene *comau-racer-999_boxes.xml* composition. *.wrl* is a file type called Virtual Reality Modeling Language.

Virtual Reality Modeling Language is a standard file format for representing 3D interactive vector graphics. It has a tree-like structure. A VRML scene is composed of several nodes.

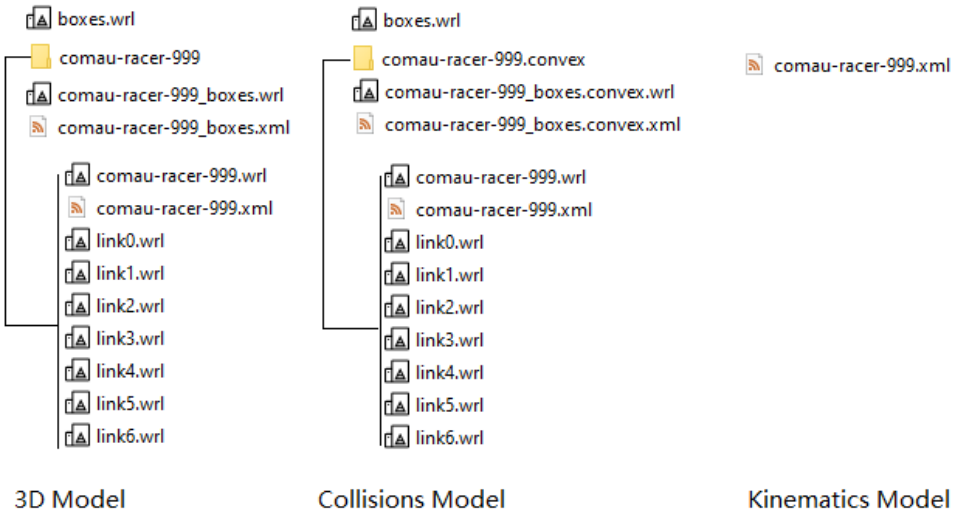


Figure 3.3: example of robotsmodel file

Node types of their own are available for basic geometric bodies such as cuboids, cylinders, cones and spheres. The left part of Figure 3.5 shows the scene file box.wrl. This file is composed of 8 boxes. One of the deformed boxes serves as the floor, the four deformed boxes serve as walls, and the remaining three serve as obstacles.

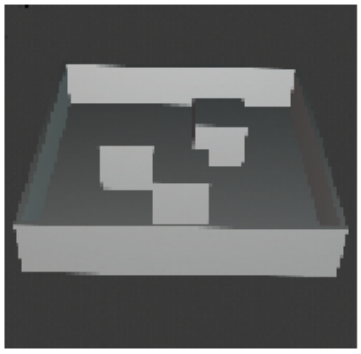
The right part of figure 3.5 is a visualization of the 3D model of the link0.wrl file, which is the foundation for Racer-999 robot. Like OBJ-Format (a common geometry definition file format), the complicated graphic objects are also built on a list of points, edges and faces. The edges and faces are saved based on the Index of Point. On the faces, material properties can be assigned to the geometric bodies. In addition, transparent textures are possible with the help of PNG images. The light sources then provide the corresponding shading of the objects through the lighting model.

```
#VRML V2.0 utf8
Transform {
    children [
        DEF link0 Transform {
            children [
                Inline {
                    url "link0.wrl"
                }
            ]
        }
        DEF link1 Transform {
            rotation 1 0 0 -1.570796
            translation 0.15 0 0.43
            children [
                Inline {
                    url "link1.wrl"
                }
            ]
        }
        DEF link2 Transform {
            ...
        }
        ...
    ]
}
```

Figure 3.4: comau-racer-999.wrl

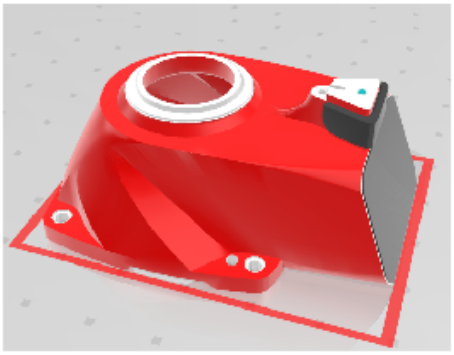
Unlike OBJ-Format, the bodies can be grouped hierarchically by utilizing transform nodes. Transformation operations such as scaling, rotation and translation can be applied to all nodes below this node.

box.wrl



```
#VRML V2.0 utf8
DEF room2 Transform {
  children [
    DEF floor Transform {
      translation 0 0 0
      children [
        Shape {
          appearance Appearance {
            material Material {
              transparency 0.5
            }
          }
          geometry Box {
            size 5 5 0.1
          }
        }
      ]
    }
    DEF box Transform {
      ...
    }
    ...
  ]
}
```

link0.wrl



```
#VRML V2.0 utf8
DEF link0 Transform {
  translation 0 0 0
  scale 0.001 0.001 0.001
  rotation -0.5 0.5 0.5 4
  children [
    DEF Racer_999_p01 Transform {
      children [
        Shape {
          appearance Appearance {
            material Material {...}
          }
          geometry IndexedFaceSet {
            coord Coordinate {point [...]}
            creaseAngle 0.610865
            normal Normal {vector[...]}
            texCoord TextureCoordinate {point [...]}
            coordIndex [...]
            texCoordIndex [...]
            normalPerVertex TRUE
            normalIndex [...]
          }
        },
        Shape {... },
        ...
      ]
    }
  ]
}
```

Figure 3.5: Model of box.wrl and link0.wrl

Through tansformation, various objects can be combined together. Figure 3.4 shows how the various objects of the robot are combined.

```
<scene href="comau-racer-999_boxes.wrl">
    <model name="comau-racer-999">
        <body name="link0"/>
        <body name="link1"/>
        ...
    </model>
    <model name="boxes">
        <body name=""/>
    </model>
</scene>
```

Figure 3.6: comau-racer-999_boxes.xml

The tree structure in the VRML file makes it easy to create forward kinematics. A finger moves with it when the arm is moved. So VRML is very suitable for use in the field of robotics.

The XML file marks all the models and the links that each model has just like in figure 3.6. By reading the XML, the nodes corresponding to the links could be found easily. Besides, speed for calculating the translation, rotation and scaling increased. Those results could be applied to the VRML model.

In order to render VRML files, a library called "Open Inventor" was introduced to this research. Open Inventor is included in Roboticslibrary. Open Inventor is an object-oriented 3D graphics toolkit for C++. It can provide higher-level programming for OpenGL. Main goal is to improve the convenience and efficiency of programmers. With OI,objects from many pre-rolled shapes (such as cubes and polygons) could be clarified. Those objects could then be easily modified into new shapes. The system will automatically apply occlusion culling to the objects in the picture. OI also includes many controller objects and systems, which can be applied to the scene, thus simplifying common interactive tasks.

We can easily read the 3D model file with the following command:

```
SoDB::init();
//sc1 is the scene of 3d model
sc1 = new rl::sg::so::Scene();
sc1->load("Path\\comau-racer-999_boxes.xml");
```

3.2.2 3D Collision Model

The middle part of the figure 3.3 shows the structure of the Collision model. The collision model is like a 3d model. It consists of a scene file *boxes.wrl*, a folder containing a robot collision model *comau-racer-999.convex*, a collision model file with scene *comau-racer-999_boxes.convex.wrl*, and a collision model description file with scene *comau-racer-999_boxes.convex.xml* composition.

Because collision detection needs to calculate a large amount of polygon data, the collision model does not need to be as accurate as 3d model, and should be composed of fewer and simpler polygons as much

as possible. The figure 3.7 shows the collision model of the robot racer999 base, because it does not require rendering and display, so there is no need to define materials, normal, and textures in the VRML file.

Considering that there is no collision detection in the open Inventor library, a physics engine is needed to read the collision model of the robot.

Robotis Library supports physics engines such as Bullet, Open Dynamics Engine, Research with Lepton Colliders. Since only collision detection is needed in this research, a relatively small physics engine called "SOLID" was employed in the working progress.

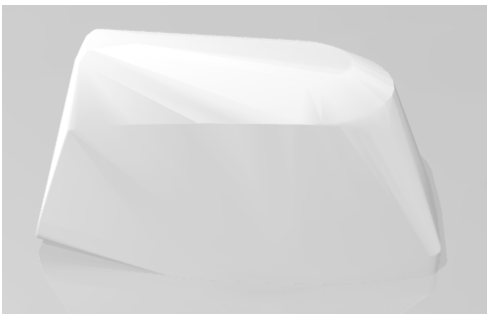


Figure 3.7: Collisionsmodel link0.wrl

"SOLID" is a library for collision detection of three-dimensional objects undergoing rigid motion and deformation. "SOLID" is especially suited for collision detection of objects and worlds described in VRML.

The collision model file can be loaded with the following command:

```
//sc2 is the scene of 3d model
sc2 = new rl::sg::solid::Scene();
sc2->load("Path\\comau-racer-999_boxes.convex.xml");
```

3.2.3 Kinematics Model

Compared with 3D models and collision models, the composition of the kinematics model is very simple, which has only one xml description file. Kinematics is the study of how things move, it describes the motion of a hierarchical skeleton structure. The kinematics description of kinematics model describes a tree-like structure. The nodes replicate the class structure used by the robotics library.

For human body animation, coordinate change is not complicated. As long as the current joint is multiplied by the node rotation matrix and child node offset matrix, the coordinate transformation from parent node to child node can be completed. The transformation matrix is $M = R \cdot T_{offset}$.

However, the coordinate transformation of robots is not that simple. Compared with humans, robots are more complex, which is mainly reflected in two aspects:

- Human joints are spherical structures with three degrees of freedom; while robots generally have

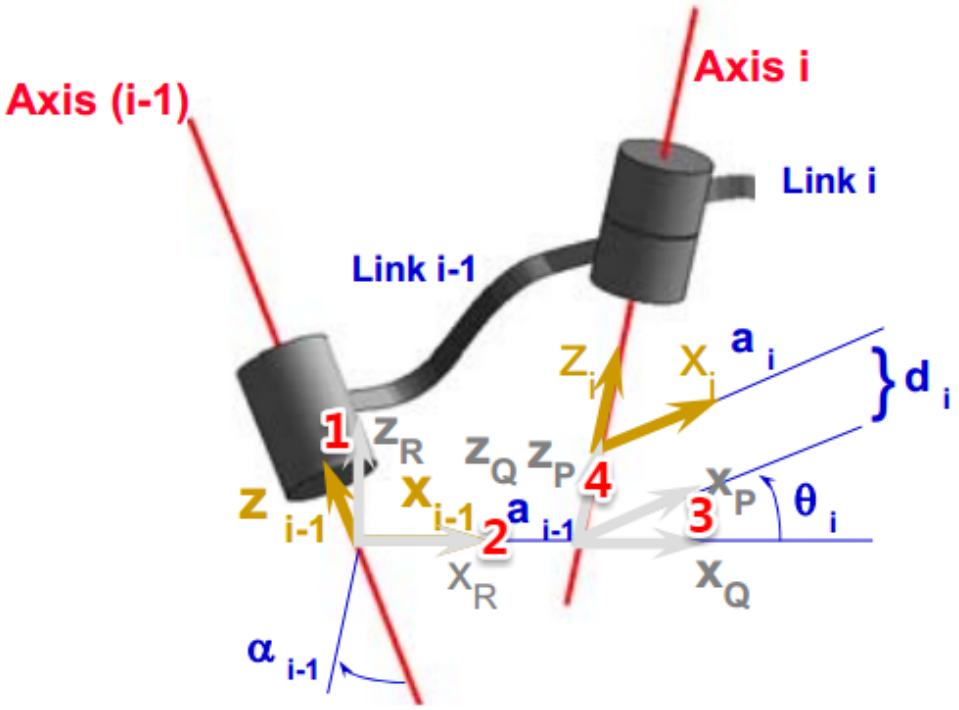


Figure 3.8: DH parameters

Source: https://www.cnblogs.com/caster99/p/4717358.html

only one free joint. Therefore, using a robot to simulate a human wrist requires three joints.

- Human joints are connected by arms, and the distance between the two joints will not change during the rotation; the connection between the joints of the robot is more complicated

To solve this problem, Jacques Denavit and Richard Hartenberg introduced a convention named "Denavit Hartenberg convertion" in 1955, in order to standardize the coordinate frames for spatial linkages.[DH55] Richard Paul proved its value in the kinematics analysis of robotic systems in 1981.[Pau81]

In Figure 3.8, the two rotary joints rotate around $Axis_{i-1}$ and $Axis_i$ respectively. The two joints are not directly connected, and the length of the link between the two joints may not be fixed or straight. The transformation information cannot be directly calculated by simply multiplied the rotation matrix with the offset matrix to perform coordinate transformation as in the human animation. This seems quite complicated, but due to the appearance of the DH parameter, the coordinate transformation becomes quite intuitive and simple. DH parameter has 4 parameters:

step 1 α_{i-1}, the angle between $Axis_{i-1}$ and $Axis_i$.

step 2 a_{i-1}, the distance between $Axis_{i-1}$ and $Axis_i$.

step 3 θ_i, the angle between a_{i-1} and a_i. a_i refers to the distance between $Axis_i$ and $Axis_{i+1}$.

step 4 d_i,the distance between a_{i-1} and a_i.

α_{i-1} and a_{i-1} are used to describe the properties of the connecting rods, and θ_i and d_i are used to describe the connection state between the connecting rods. We can choose the coordinate origin at the intersection of $Axis_{i-1}$ and a_{i-1}. So in robotics coordinates z-Axis can be along $Axis_{i-1}$, and the x-Axis can be along a_{i-1}. The y-Axis must be equal to the cross product of the x-Axis and the z-Axis, Given that the position of y-Axis was determined, the direction can also be determined according to the right-handed rule.

According to these four parameters, $Axis_{i-1}$ can be transformed into $Axis_i$. Coordinate transformation is mainly divided into 4 steps:

step 1 Rotate $Axis_{i-1}$ by α_{i-1} around the a_{i-1} axis.

step 2 $Axis_{i-1}$ is translated a_{i-1} along the a_{i-1} axis.

step 3 Rotate $Axis_{i-1}$ around $Axis_i$ by θ_i.

step 4 Translate$Axis_{i-1}$ along the $Axis_i$ direction by d_i.

According to these four transformations, transformations matrix M of $Axis_{i-1}$ could be converted to $Axis_i$. The specific formula is as follows:

$$ {}^{i-1}_{i}\mathrm{M}(\alpha_{i-1}, \alpha_i, \theta_i, d_i) = R_x(\alpha_{i-1}) \cdot T_x(a_{i-1}) \cdot R_z(\theta_i) \cdot T_z(d_i) \tag{3.1} $$

So:

$$ {}^{i-1}_{i}\mathrm{M} = \begin{bmatrix} cos\theta_i & -sin\theta_i & 0 & a_{i-1} \\ sin\theta_i cos\alpha_{i-1} & cos\theta_i cos\alpha_{i-1} & -sin\alpha_{i-1} & -d_i sin\alpha_{i-1} \\ sin\theta_i sin\alpha_{i-1} & cos\theta_i sin\alpha_{i-1} & cos\alpha_{i-1} & d_i cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2} $$

The kinematics model file also uses DH parameters to constrain the model. The available nodes in a model description tree include one world node *world*. The nodes associated with the robot's links *body*. Intermediate nodes simply called *frame*. A static transform defines a rotation and translation between two frames a and b.

The Denavit-Hartenberg convention used here defines an order of d, θ, a, and α. In case of a revolute joint, which controls the translation, the effect of the rotation is added to the θ value. For a prismatic joint, which controls the rotation,the effect of the rotation is added to the d value. As the structure showed in Figure 3.9 ,three transforms are required to create one DH revolute joint: a *fixed* node for translation d, the revolute joint itself, and a *fixed* node for translation a and rotation α.

Through the kinematics model, it is rather convenient to manipulate the robot and to calculate the for-

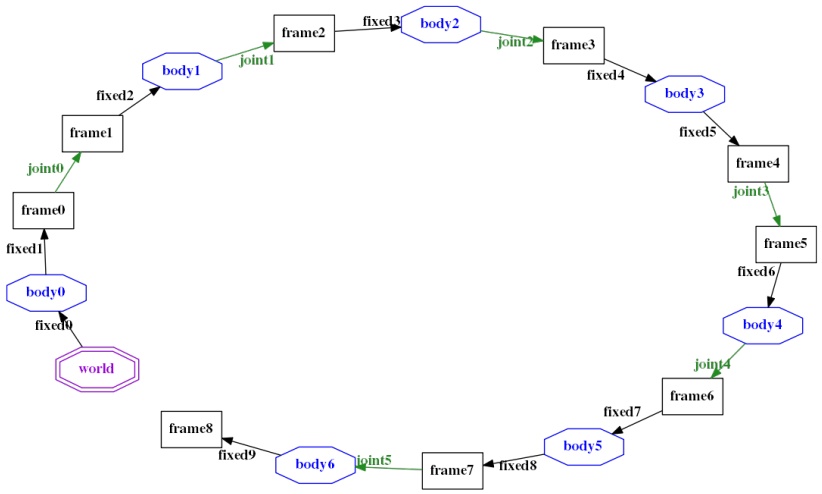


Figure 3.9: A example of tree structure of the robot described

Source: https://www.roboticslibrary.org/tutorials/create-a-robot-model/

ward and inverse kinematics of the robot. The kinematics model of robot could be loaded by Using the following command.

```
//model is kinematics model of robot
rl::mdl::XmlFactory factory;
std::shared_ptr<rl::mdl::Model> model(factory.create("Path\\comau-racer-999.xml"));
```

3.3 Rendering

In the previous section, the robot model file and scene file has been loaded. This section will explain how them could be rendered.

The difference between the common coordinate system in robotics research and in cgv is crucial.

Three different coordinate systems are shown in figure 3.10.

The green axis is the coordinate system of robotic, both of robot model and simple box scene with collision are using this coordinate system when loading. The blue axis represents the coordinate system in the cgv library. This is also the coordinate system of viewport and controller of VR and cgv_viewer.

In order to simplify the calculation, a third coordinate system was introduced for this program. This coordinate system is represented by the coordinate axis shown in red in Figure 3.10. One advantage of this coordinate system is that the positive increment of the coordinates is consistent with the increment direction of the robot coordinates. All three coordinate axis are ruled by the right handed, so they can

transform each other.

In the robot model, the z-Axis$_{robotics}$ represents the height of the robot. However, in cgv, the z-Axis$_{cgv}$ is from the outside to the inside of the screen, and the y-Axis$_{cgv}$ represents the height. By rotating the model by 270 degrees around x-Axis$_{cgv}$, the model can be displayed in the viewport as in common scene. With these mapping:

$$\begin{aligned} -x_{robotic} &\mapsto x_{our} \\ z_{robotis} &\mapsto y_{our} \\ y_{robotics} &\mapsto z_{our} \end{aligned} \tag{3.3}$$

The coordinates of robotics can convert to coordinates in this research without calculation. Just take the inverse of x and z, the coordinate system of cgv could also be obtained.

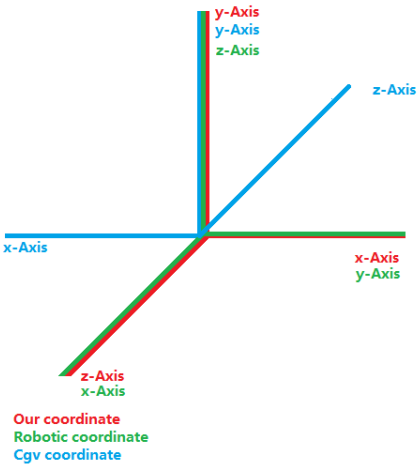


Figure 3.10: Coordinate in 3 different

The following code were written in the *draw()* method. These code is used to render the scene with the model:

```
GLint vp[4];
//read VIEWPORT configuration
glGetIntegerv(GL_VIEWPORT, vp);
//set VIERPORT properties
SbViewportRegion myView(vp[2], vp[3]);
myView.setViewportPixels(vp[0], vp[1], vp[2], vp[3]);
//renderaction with opengl
SoGLRenderAction myAction(myView);
//set TransparencyType to calibration the Viewport
myAction.setTransparencyType(SoGLRenderAction::SCREEN_DOOR);
ctx.push_modelview_matrix();
mat4 P;
//rotate the model
quat(vec3(1, 0, 0), rl::math::DEG2RAD * 270).put_homogeneous_matrix(P);
ctx.mul_modelview_matrix(P);
myAction.apply(sc1->root);
ctx.pop_modelview_matrix();
```

The current viewport data (offset viewport of VR headset) must be passed to the opengl renderer of the Inventor Library. After rotate the model through the utilization of quaternions, the apply method is executed. Then, the opengl renderer will be called. Scene with robot model will be displayed in VR and cgv_viewer.

3.4 Load transformation of Controller

Because the calculation of IK is time-consuming, sampling n times for each degree of freedom in a space with 6 DoF requires n^6 time. As the sampling accuracy n increases, the sampling time will also have corresponding exponential growth. For sampling situations where the time complexity is $O(n^6)$, real-time rendering is basically impossible.

Therefore, a sampling method based on the controller posture was choosen for later research process. Given the position and rotation of the controller as the transformation of the end effector of the robot for sampling, to sample a pose or a series of poses related to this transformation, is the only thing needed. This makes real-time rendering possible in VR.

Because barely one controller is required, the left controller would be used as the main controller. In order to improve performance, the posture of the controller will be saved merely when the click event is triggered. There is a switch *control_handle* in *cgv::gui::EID_POSE* to control this event. This switch will be turned on when the controller button is pressed and turned off when the button is lifted.

The position of the controller *control_pos* is stored in a vector of vec3, and the rotation *control_rot* is stored in a 3x3 matrix mat3. In robotics, Euler angles are usually used to indicate the orientation of an object.

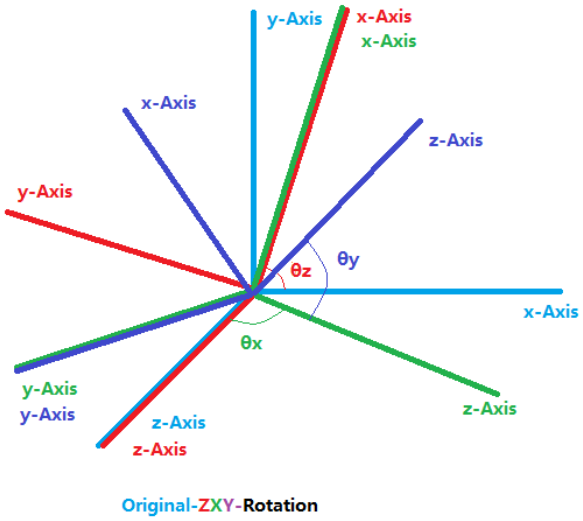


Figure 3.11: ZXY eulerangle system

Swiss mathematician Leonhard Euler used euler angles to describe the orientation of a rigid body in three-dimensional Euclidean space. [Die06] The rotational position is generated from any other by a sequence of three rotations around special axes, the rotation angles being the independent parameters. The first axis of rotation is fixed in space, the other two are axes that were previously rotated. The rotations of the body represent a geometric transformation. The transformations are calculated with the help of rotation matrices. Any rotation matrix R can be decomposed as a product of three elemental rotation matrices. The Rotation around the x-axis, y-axis and z-axis by angle

θ is defined with:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{bmatrix} \tag{3.4}$$

$$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \tag{3.5}$$

$$R_z(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

The most commonly used rotation order is Z'XZ". The rigid body rotates around the z axis by $\theta_{z'}$ first, then around the new x axis by θ_x, and finally around the rotated z axis by $\theta_{z"}$. Because the z-axis is upward in robotics, the last step actually describes the partial rotation of the object. However, considering different coordinate systems, the z-axis of the robot system coordinates is the y-axis of our coordinate axis. So in order to simplify the calculation, we use Y to express the pure self rotation in last step. Figure 3.11 shows the ZXY Euler angle system. The original coordinate system shown in blue is rotated by θ_z around the Z axis, the new coordinate system is shown in red. Then, this coordinate system is rotated around the X axis by θ_x , which is represented in green. Finally, it turned the θ_y around the Y axis, which is represented by purple.

The Rotation matrix can be calculated with:

$$R = R_z(\theta_z)\dot{R}_x(\theta_x)\dot{R}_y(\theta_y) \tag{3.7}$$

So the matrix:

$$\begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} cos\theta_y cos\theta_z - sin\theta_x sin\theta_y sin\theta_z & -cos\theta_x sin\theta_z & cos\theta_z sin\theta_y + cos\theta_y sin\theta_x sin\theta_z \\ cos\theta_z sin\theta_x sin\theta_y + cos\theta_y sin\theta_z & cos\theta_x cos\theta_z & cos\theta_y cos\theta_z sin\theta_x + sin\theta_y sin\theta_z \\ -cos\theta_x sin\theta_y & sin\theta_x & cos\theta_x cos\theta_y \end{bmatrix} \tag{3.8}$$

Since the rotation matrix has been stored in *controll_rot*, the 3 rotation angles can be calculated. The simplest term is $sin\theta_x = r_{21}$, so $\theta_x = asin(r_{21})$. There are three cases to consider.

1. If $\theta_x \in (-\pi/2, \pi/2)$, then $cos\theta_x \neq 0$, so $\theta_y = atan2(-r_{20}, r_{22})$ and$\theta_z = atan2(-r_{01}, r_{11})$.

2. If $\theta_x = \pi/2$, then$sin\theta_x = 1$ and $cos\theta_x = 0$, the rotation matrix can be simplify as:

$$\begin{bmatrix} r_{00} & r_{02} \\ r_{10} & r_{12} \end{bmatrix} = \begin{bmatrix} cos(\theta_y + \theta_z) & sin(\theta_y + \theta_z) \\ sin(\theta_y + \theta_z) & -cos(\theta_y + \theta_z) \end{bmatrix} \tag{3.9}$$

So

$$\theta_y + \theta_z = atan2(r_{02}, r_{00}), \tag{3.10}$$

there is one degree of freedom, and the factorization is not unique. Let the $\theta_y = 0$, then $\theta_z = atan2(r_{02}, r_{00})$.

3. If $\theta_x = -\pi/2$, then$sin\theta_x = -1$ and $cos\theta_x = 0$, the rotation matrix can be simplify as:

$$\begin{bmatrix} r_{00} & r_{02} \\ r_{10} & r_{12} \end{bmatrix} = \begin{bmatrix} cos(\theta_y - \theta_z) & sin(\theta_y - \theta_z) \\ -sin(\theta_y - \theta_z) & cos(\theta_y - \theta_z) \end{bmatrix} \tag{3.11}$$

So

$$\theta_y - \theta_z = atan2(r_{02}, r_{00}), \tag{3.12}$$

there is one degree of freedom, and the factorization is not unique. Let the $\theta_y = 0$, then $\theta_z = -atan2(r_{02}, r_{00})$.

Through the *calAngle()* function, these three euler angles can be calculated.

So far, the 6 parameter variables of the oracle function have been calculated, the translation with 3 DoF position: x, y, z and the rotation with 3 DoF angle $\theta_x, \theta_y, \theta_z$. In the next section, the oracle function will be sampled and then visualize in the screen.

3.5 Sampling and Visualization

As mentioned in the previous section, the result of oracle function can be obtained by calculating inverse kinematics. The mission of inverse kinematics is to compute how joints of the hierarchy would rotate or translate, when the end effector reached some specific goal state.

Representing the orientation as Euler angles, the pose e of the end effector is a 6-dimensional vector,

$$\vec{e} = (x, y, z, \theta_x, \theta_y, \theta_z). \tag{3.13}$$

Each joint in one kinematic chain has a relative transformation ${}^{i-1}\widetilde{T_i} = (q_{i1}, ..., q_{in_i})$ with n_i parameters. Gathering all $n = \sum_i n_i$ parameters in the parameter vector $\vec{q}$ with the N chain transform is

$${}^0\widetilde{T}(\vec{q}) = {}^0\widetilde{T_1} \cdot {}^1\widetilde{T_1} \cdot ... \cdot {}^{N-1}\widetilde{T_N}. \tag{3.14}$$

The inverse kinematics is described as a inverse function $\vec{f}^{-1}$ that maps the 6D pose to the n-dimensional parameter vector.

$$\vec{q} = \vec{f}^{-1}(\vec{e}). \tag{3.15}$$

It is challenging to solve the IK. Sometimes, there are several possible solutions. There could be no solutions at other times. To find a solution, a complex and expensive computations is required.

RL library support 2 method to calculate the IK:

1 Pseudo inverse Jacobian method

2 Sequential Quadratic Programming method

The Jacobian is a linear approximation to $\vec{f}$.

$$\vec{J} = \frac{\vec{e}}{\vec{q}}. \tag{3.16}$$

So the parameter vector $\vec{q}$ can be calculate with help of the inverse of Jacobian Matrix as:

$$\vec{q} = J^{-1}\vec{e}. \tag{3.17}$$

The Jacobian Matrix is the matrix of partial derivatives of entire system. It defines how the end effector $\vec{e}$ changes relative to instantaneous changes in the system.

$$J = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \cdots & \frac{\partial x}{\partial q_N} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \cdots & \frac{\partial y}{\partial q_N} \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \cdots & \frac{\partial z}{\partial q_N} \\ \frac{\partial \theta_x}{\partial q_1} & \frac{\partial \theta_x}{\partial q_2} & \cdots & \frac{\partial \theta_x}{\partial q_N} \\ \frac{\partial \theta_y}{\partial q_1} & \frac{\partial \theta_y}{\partial q_2} & \cdots & \frac{\partial \theta_y}{\partial q_N} \\ \frac{\partial \theta_z}{\partial q_1} & \frac{\partial \theta_z}{\partial q_2} & \cdots & \frac{\partial \theta_z}{\partial q_N} \end{bmatrix}. \tag{3.18}$$

For one column of the Jacobian Matirx,

$$\frac{\partial \vec{e}}{\partial \vec{q_i}} = \begin{bmatrix} \frac{\partial x}{\partial q_i} & \frac{\partial y}{\partial q_i} & \frac{\partial z}{\partial p_i} & \frac{\partial \theta_x}{\partial p_i} & \frac{\partial \theta_y}{\partial p_i} & \frac{\partial \theta_z}{\partial p_i} \end{bmatrix}^T. \tag{3.19}$$

By add a small $\Delta\vec{q}$ to q_i, the moves of the end effector $\Delta\vec{e}$ can also be calculated:

$$\Delta\vec{e} = \vec{e'} - \vec{e}. \tag{3.20}$$

where e' is the new transformation of end effector. So the Jacobian Matrix can be calculate numerically as:

$$\frac{\partial \vec{e}}{\partial \vec{q_i}} \approx \frac{\Delta e}{\Delta q} = \begin{bmatrix} \frac{\Delta x}{\Delta q_i} & \frac{\Delta y}{\Delta q_i} & \frac{\Delta z}{\Delta p_i} & \frac{\Delta \theta_x}{\Delta p_i} & \frac{\Delta \theta_y}{\Delta p_i} & \frac{\Delta \theta_z}{\Delta p_i} \end{bmatrix}^T. \tag{3.21}$$

There is no guarantee, that Jacobian Matrix is invertible. Because:

1. Jacobian Matrix is typical not a square matrix. (maybe no result)
2. Singularities. (unlimited result)

Singularity reduces the mobility of the manipulator. Usually, any movement of the manipulator in the Cartesian direction is lost. This is also called "loss of freedom". At a joint space singularity, infinite inverse kinematic solutions may exist. Even the Jacobian Matrix is invertible, the properties of the matrix will be changed as the pose vector changes. By using the pseudo inverse, the inverse matrix can be find effectively when there a non square matrix exist.

$$J^{+} = (J^{T} \cdot J)^{-1} \cdot J^{T}. \tag{3.22}$$

Because forward kinematic (by setting the parameter vector to get the transformation of end effector) is nonlinear, so that the Jacobian can only be used as an approximation that is valid near the current configuration. That implies the process of computing a Jacobian must repeat by taking a small step of joint configuration until the pose get close towards the goal enough. The pseudo code is as follows.

Algorithm 1 Algorithm of the Jacobian Method

Input: Transformation of current state of end effector e, of goals state of end effector g

Output: Joints configuration parameter

1: **while** e is so far from g **do**
2: compute the Jacobian Matrix J
3: compute the pseudo inverse of the Jacobian matrix J^{+}
4: compute change in joint DoFs: $\Delta q = J^{+} \cdot \Delta e$
5: apply the change of DoFs, move a small step of $\alpha \cdot \Delta q$
6: **end while**

With the pseudo inverse Jacobian method, IK can be solved very fast, but the correct rate is only about 80%.[BA15]

There are some limitation of the pseudo inverse Jacobian method:

1. For robots with joint limits, there exhibits false negative failures in some cases.
2. No actions will be taken, when the target transformation is very near to the current pose of the end effector.
3. This method do not implement the tolerance of the IK solver itself.
4. Insufficient support for Cartesian pose tolerances.

One way to avoid these failures is to solve IK by using sequential quadratic programming method. SQP is an iterative algorithm for nonlinear optimization, which can be applied to problems where the objective function and constraints are two consecutive differentiable. As mentioned before, Jacobian Matrix can

be written as

$$J \approx \frac{\Delta e}{\Delta q} = \frac{e' - e}{q' - q}. \tag{3.23}$$

. Then the tolerance function is:

$$N(q') = J(q' - q) - (e' - e). \tag{3.24}$$

By introducing a quadratic cost function

$$Z(q') = \frac{1}{2}(q' - q)^T M(q' - q) - f^T q'. \tag{3.25}$$

with positive defined weight matrix M and a weight vector f, the Lagrange function can be set up:

$$L(q', \lambda) = Z(q') + \lambda^T N(q'). \tag{3.26}$$

with the necessary condition $\frac{\partial L}{\partial q'^T} = 0$ and $\frac{\partial L}{\partial \lambda^T} = 0$. so this leads to a linear system of equations:

$$\begin{bmatrix} M & J^T \\ J & 0 \end{bmatrix} \begin{bmatrix} q' \\ \lambda \end{bmatrix} = \begin{bmatrix} Mq + f^T \\ Jq + (e' - e) \end{bmatrix}. \tag{3.27}$$

which can then be solved efficiently as the matrix on the left hand side is usually sparse.[LBSH08]. To calculate IK, compared with the pseudo inverse Jacobian method, the Sequential Quadratic Programming method has a higher correct rate, but the speed will be correspondingly slower.[BA15] The pseudo inverse Jacobian method is used as the default method in the project to obtain the fluency of real-time calculation in the VR scene.

As mentioned in the previous section, the kinematic data of the robot can be obtained by reading the kinematic model. When solving the IK, the kinematic model will continue to apply new joint parameters until the target state of end effector is found. If a pose is in posespace, it will return a true value, if not, it will return a false value. So far, the entire oracle function can be obtained:

$$f(x, y, z, \theta_x, \theta_y, \theta_z) = \{1(true), 0(false)\}. \tag{3.28}$$

3.5.1 Sampling and Visualization of a fixed pose

The Sampling of this project is based on the pose of the controller. To sample a fixed posture, the 6 DOF parameters of the controller need to be analyzed, these parameters should be mapped to the coordinate axis of the robot system for IK calculation. For rotation parameters, the ZXY Euler system is used for conversion.

When the robot arm is fully extended, the maximum length it can reach is l_{max} . If the distance between the position of a pose and the position of the robot base is greater than l_{max}, then the pose cannot be in

posespace. So in order to avoid unnecessary calculations and to improve performance, the pose position should be evaluate firstly. Given that the robot base is on the original point $(0, 0, 0)$ of coordinate system, when$x^2 + y^2 + z^2 > l_{max}^2$, then the result of IK can be directly set as false(0). Otherwise, IK will be calculated as normal.

After the IK solving process is completed, the 7 parameters of these oracle functions will be saved in a vector *result*. If a pose is reachable, the parameters of each joint of this situation will be read and passed to the collision model to calculate the state of collision and 3D model to visualize the current pose of robot.

After the pose of the collision model is configured, each body of robot will perform a collision detection with the obstacle(box) of scene. If anybody has a collision with the obstacle, the last variable of vector will be set as 2 instead of 1.

In this project, arrows are used to indicate whether the fixed pose is in the pose space. Each arrow has 3 attributes which can be used for information description:

1 Position, which in the center of the circle at the bottom of the arrow. It describes where the arrow is. In this project, this position could help to describe the position (x,y,z) of pose.

2 Normal describes the direction of the arrow, which can help to describe the non-spin rotation angle of the other 2 degrees of freedom. Because the arrow is axisymmetric, it cannot be used to represent spin. Just like combinations of latitude and longitude can represent any position on the earth, the direction from the center of the sphere to any point on the sphere is determined by only two parameters.

3 Color can indicate the state of information. In the project, it is used to describe the type of pose (whether the pose is in posespace or collided) or the type of arrow (whether the arrow is used for auxiliary reference).

Unlike other visualization methods of posespace [VAD13][MG18], 3 arrows are used in the project to indicate the state of a pose. Figure 3.12 is an example, it is a visualization of the pose with position $(0, 1, 0.2)$ and rotation $(0, 45, 0)$ and the result of IK 1 (this pose is posespace and the robot is not collision with other objects);

1. The green arrows indicate the values of x, y, z, θ_x and θ_z, The position of the green arrow indicates the position of the end effector.

$$\vec{p_1} = \begin{bmatrix} x & y & z \end{bmatrix}^T \tag{3.29}$$

The direction pointed by green arrow is the one, that the vector (0,1,0) (which exists on the y axis) rotates $(\theta_x, \theta_y, \theta_z)$ according to the ZXY Euler system.

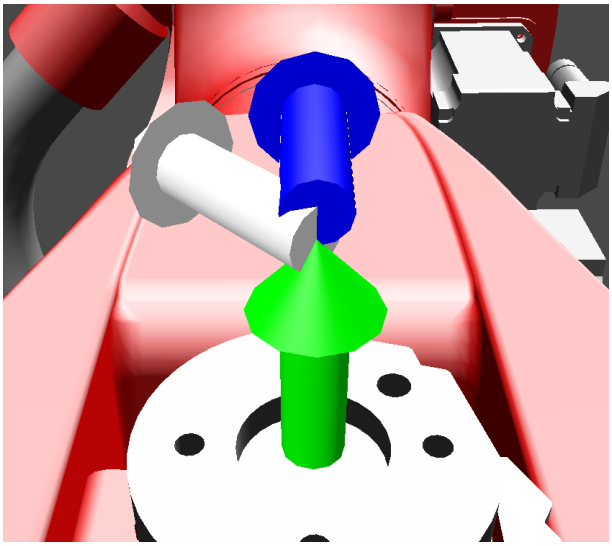


Figure 3.12: Visualization of a fixed Pose

With (3.8), the normal of arrow can be calculated as:

$$\vec{n}_1 = R_1 \cdot \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T = \begin{bmatrix} -cos\theta_x sin\theta_z & cos\theta_x cos\theta_z & sin\theta_x \end{bmatrix}^T \tag{3.30}$$

This also explains that the value of θ_y is independent of the direction of the blue arrow.

2. The blue arrow is just a reference for θ_y. It shows the situation when the end effector does not spin, for this state $\theta_y = 0$. The position of the blue arrow is nearby the top of the green arrow, which is slightly higher than the top of the green arrow. This is to avoid overlapping with the third arrow.This position can also describe the blue arrow based on which green arrow is defined.

$$\vec{p}_2 = \vec{p}_1 + 1.1 * arrowlength * \vec{n}_1 \tag{3.31}$$

The direction pointed by blue arrow is the one, that the vector (0,0,-1) (which exists on the z axis) rotates $(\theta_x, 0, \theta_z)$ according to the ZXY Euler system. With (3.8), the normal of arrow can be calculated as:

$$\vec{n}_1 = R_2 \cdot \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T = \begin{bmatrix} sin\theta_x sin\theta_z & -sin\theta_x cos\theta_z & cos\theta_x \end{bmatrix}^T \tag{3.32}$$

3. The 3rd arrow indicate the values of θ_y and results of IK. If the pose is not reachable, then the arrow will be indicated with black colour. If the pose is reachable but collides with the scene, the arrow will be indicated by red. If the pose is reachable but does not collide with the environment, Then it will be shown in white. θ_y can be expressed by the angle between the blue arrow and this arrow. The sign of the value can be defined with the right-hand rule. For example, in Figure 3.12, the white arrow is at 45 degrees counterclockwise from the blue arrow, so $\theta_y = 45$, which also

shows that the pose is in posespace and do not collide with other objects. The position of the blue arrow is on the top of the green arrow.

$$\vec{p_3} = \vec{p_1} + *arrowlength * \vec{n_1} \tag{3.33}$$

The direction pointed by this arrow points is the direction after the vector (0,0,-1) on the z axis rotates $(\theta_x, \theta_y, \theta_z)$ according to the ZXY Euler system. With (3.8), the normal of arrow can be calculated as:

$$\vec{n_3} = R_3 \cdot \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T = \begin{bmatrix} sin\theta_y cos\theta_z + sin\theta_x cos\theta y sin\theta_z \\ sin\theta_y sin\theta_z - sin\theta_x cos\theta_y cos\theta_z \\ cos\theta_x cos\theta y \end{bmatrix} \tag{3.34}$$

By reading the value in the vector *result*, the position, direction and color of the three arrows can be calculated, and they will be stored in the three vectors *arc_pose*, *arc_normal*, and *arc_color* during the visualization process. These vectors will be bound to the arrow rendering in the *draw()* method and rendered in the scene.

3.5.2 Sampling and Visualization for fixed rotation

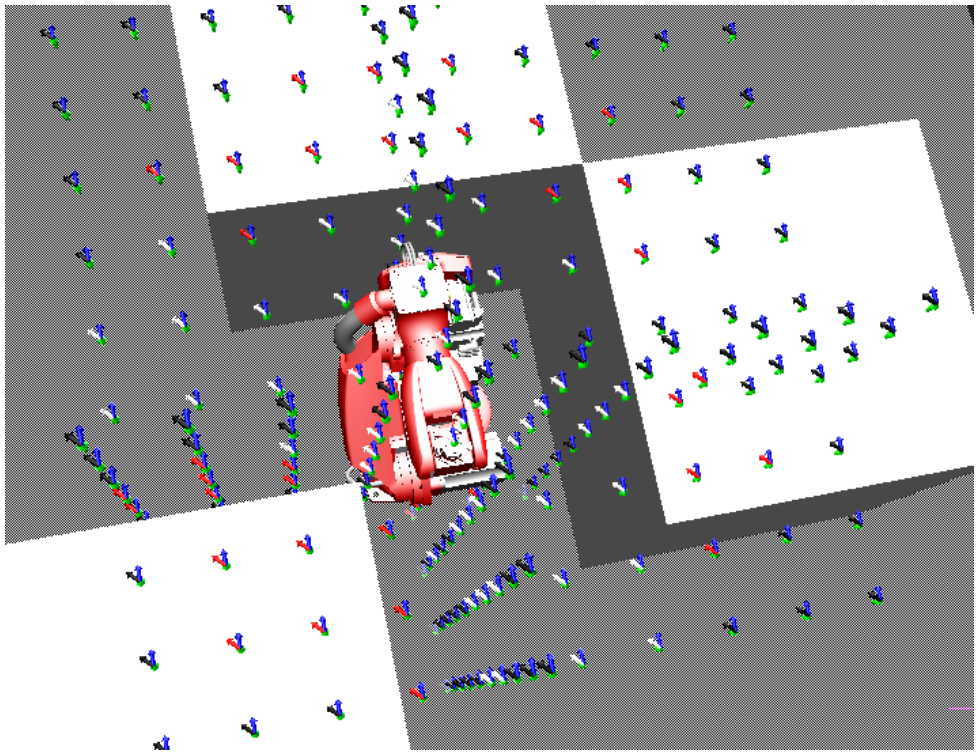


Figure 3.13: A series of samples based on a fixed angle

Figure 3.13 shows a series of posture sampling and visualization based on a fixed rotation $\theta_x = 0, \theta_y = 45, \theta z = 0$. For sampling with a fixed rotation, the same sampling and visualization strategy as the fixed pose is adopted.

The entire sampling time is about 10s. For a fixed angle, the poses on x-axis plane, y-axis plane, and z-axis plane of the controls position will be sample of this method. Because the positions of the sampled poses are different, the arrows can also be clearly expressed in space. The arrow group clearly shows the fixed rotation, and the position of the arrow group can also be simply indicated.

For sampling on the x plane, the values of x, θ_x, θ_y and θ_z are fixed, the y_i and z_j can be sampled from the negative maximum distance $-l_{max}$ until maximum distance l_{max} the robot can reach l_{max}. The sampling method on the y and z planes is as same as the sampling on the x plane. After each sampling for a fixed pose, the vector *result* will be saved in a new vector named *final_result*.

These *result* will be read and calculated from *final_result* when visualizing, and then the attributes of the arrows will be stored in the vector accordingly.

3.5.3 Sampling and Visualization for fixed translation

For a fixed translation sampling, the sampling and visualization methods for fixed points are no longer applicable. The arrow group will be gathered around the position of the end effector and obscure each other, so it is difficult to distinguish the rotation angle of the samples.

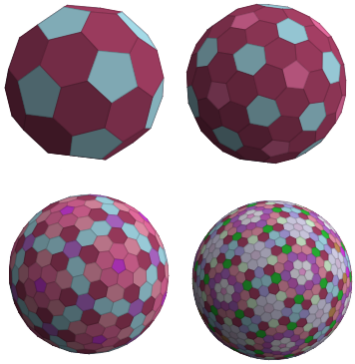


Figure 3.14: Goldberg polyhedrons

In this project, a sampling and visualization method based on Goldberg polyhedron was adopted. In mathematics, a Goldberg polyhedron is a convex polyhedron made of hexagons and pentagons.[Har13] The simplest Goldberg polyhedron is Regular dodecahedron. The more complex the Goldberg polyherdron is, the more that Goldberg polyherdron will look like a ball, so the Goldberg polyhedron are a good approximation to a sphere for many purposes.

Figure 3.14 shows some Goldberg polyhedron. Through the position of goldberg polyhedron, we can indicate the location of sampling. Through the vector from the center of the sphere to the center of the regular pentagon or to the center of the regular hexagon on the Goldberg polyhedron, we can solve for θ_x and θ_z. Just like the direction of the green arrow in the fixed-pose sampling strategy. A goldberg

polyhedron with more faces also can describe more sampling of θ_x and θ_z.

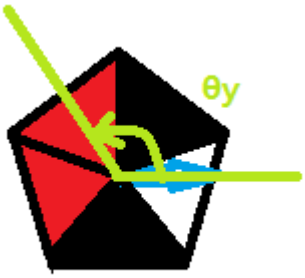


Figure 3.15: Describe of θ_y

Add an arrow that indicates the spin angle $\theta_y = 0$ on the polygon surface, just like the blue arrow in the fixed pose strategy. So by cutting the polygon into many triangles, these triangles all have the same vertex on the center of the polygon, then this can form an angle with the arrow and the median through the vertex. The median of a triangle is a line segment joining a vertex to the midpoint of the opposite side. The angle is then θ_y like described in figure 3.15.

The more triangles are divided into the polygon, the more angles of θ_y are sampled. The color of the triangle can represent the state of pose like the third arrow color in the fixed pose sample.

In order to sample the poses, the Goldberg polyhedron must be constructed. In this project, the method of Conway notation was employed to construct Goldberg polyhedron.

The Conway polyhedron notation, invented by John Horton Conway and popularized by George W. Hart, is used to describe polyhedrons based on seed polyhedrons modified by various prefix operations.[CBGS08]

As a example, the Goldberg polyhedron can be constructed by following step:

1.Step The regular isosahedron can be select as seed polyhedron. Regular isosahedron is the polyhedron, whose faces are 20 equilateral triangles.

2.Step With the the truncation operation, witch is an operation cuts polytope vertices and creating a new face in place of each vertex, the isosahedron can be transformed to a truncated icosahedron. The upper left of figur 3.15 is a truncated icosahedron, which is the polyhedron used in football.

3.Step With the dual operation, witch is swap the vertex and face elements.

4.Step By apply a "Truncate" operation again, the Goldberg polyhedron upper right of figur 3.15 is successfully generated.

More Goldberg polyhedron can be obtained by repeating the 3rd and 4th steps. In this project, Goldberg polyhedron can be construct simply with *cgv::simple::mesh::simplemesh* by input a Conway polyhedron notation like "tdtI", witch means "truncated dual truncated isosahedron".

To sampling the poses of a fixed translation, the position data $\{x, y, z\}$ of controller is needed for transla-

tion data $\{-x, z, y\}$ and passed to goldberg polyhedron. This is also the coordinate mapping mentioned in (3.3). In the project, the goldberg polyhedron is also scaled to a suitable size with $scale = 0.05$ for visualization.

In order to sample and visualize fixed displacement poses, in each face of the goldberg polyhedron, the following steps are required:

1.Step Calculate the face center as *center* with:

1: **for** each vertex in vertices **do**
2: $center = center + vertex$
3: **end for**
4: $center = center/number\ of\ vertices$

2.Step Calculate the vector from the center of the sphere to the center of the face as $\vec{n}_y$.

3.Step Decompose $\vec{n}_y$ into 2 Euler angles θ_x and θz with

$$\vec{n}_y = \begin{bmatrix} a & b & c \end{bmatrix}^T = \begin{bmatrix} -cos\theta_x sin\theta_z & cos\theta_x cos\theta_z & sin\theta_x \end{bmatrix}^T \tag{3.35}$$

So:

1. if $c = -1$, then $\theta_x = -90$ and $\theta_z = 0$.
2. if $c = 1$, then $\theta_x = 90$ and $\theta_z = 0$.
3. if $b > 0$, then $\theta_x = \arcsin(c)$ and $\theta_z = \arctan(\frac{-a}{b})$
4. if $b < 0$, then $\theta_x = \arcsin(c)$ and $\theta_z = \arctan(\frac{-a}{b}) + 180$
5. if $a > 0$,then $\theta_x = \arcsin(c)$ and $\theta_z = -90$
6. if $a < 0$,then $\theta_x = \arcsin(c)$ and $\theta_z = 90$

4.Step store the *center* as position of arrows, the vector $\vec{n}_{arc} = \begin{bmatrix} sin\theta_x sin\theta_z & -sin\theta_x cos\theta_z & cos\theta_x \end{bmatrix}^T$ as the normal of arrows, $[0, 0, 1]^T$ as the color of arrows, to visualize the blue arrow for reference θ_y.

5.Step Cut the polygon into triangles with the *center* as the vertex, save the position of triangles in a vector *triangle_position*. And $\vec{n}_y$ as normal of the triangles *triangle_normal*.

6.Step Calculate the median $\vec{n}_{median}$ through the *center* for each triangle. $\vec{n}_{median} = (p_{n1} + p_{n2})/2 - center$

7.Step Calculated θ_y by median $\vec{n}_{median}$ and normal of arrows $\vec{n}_{arc}$:

$$\theta_y = \arctan 2((\vec{n}_{arc} \times \vec{n}_{median}) \cdot \vec{n}_y, \vec{n}_{arc} \cdot \vec{n}_{median}) \tag{3.36}$$

1: **for** vertex$_i$ in vertices **do**
2: **for** n in N section **do**
3: p$_{n1}$=vertex$_i$+n*(vertex$_{i+1}$-vertex$_i$)/N
4: p$_{n2}$=vertex$_i$+(n+i)*(vertex$_{i+1}$-vertex$_i$)/N
5: **end for**
6: **end for**

8.Step Solve IK with $x, y, z, \theta_x, \theta_y, \theta_z$, if the result is false, then save the color $0, 0, 0$ in the vector *triangle_color*. If result is true, then pass the joint configuration to other models of robot to visualize and perform a collision detection. If there are a collision, then save the color $1, 0, 0$ in the vector *triangle_color*. If there are nom collision, then save the color $1, 1, 1$ in the vector *triangle_color*.

In the *draw()* method, the 3 vectors *triangle_position*, *triangle_normal*, and *triangle_color* are bound to the triangle renderer.

Figure 3.16 is a sampling and visualization for fixed translation with $x = -0.2, y = 1, z = 0.2$ based on the Goldberg polyhedron with diffrent Conway notation. With The default notation "tdtI", the sampling time is about 10s.

In order to obtain faster sampling, it can be obtained by using simpler Conway Notation. For example, the sampling of "D"(regular dodecahedron) and "I"(regular icosahedron) is need about 3-4 seconds, the sampling of "C"(cube) is more quickly only need about 1 second, and the sampling of "tI"(truncated icosahedron) can also be completed in about 6 seconds. In order to obtain more and more accurate sampling data, a more complex Conway Notation like "tdtdtdtdI" (see bottom middle of figure 3.16) can also be used to construct a Goldberg polyhedron mesh that helps sampling and visualization.

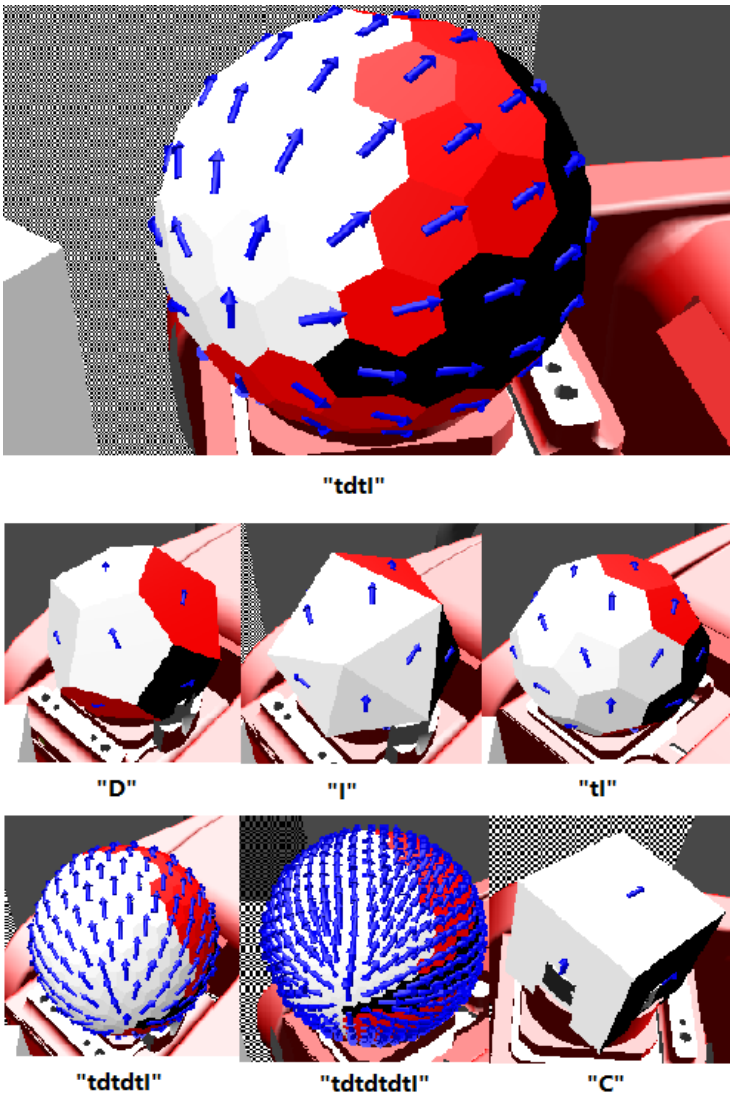


Figure 3.16: Sampling and Visualization for fixed translation

4 Exploration

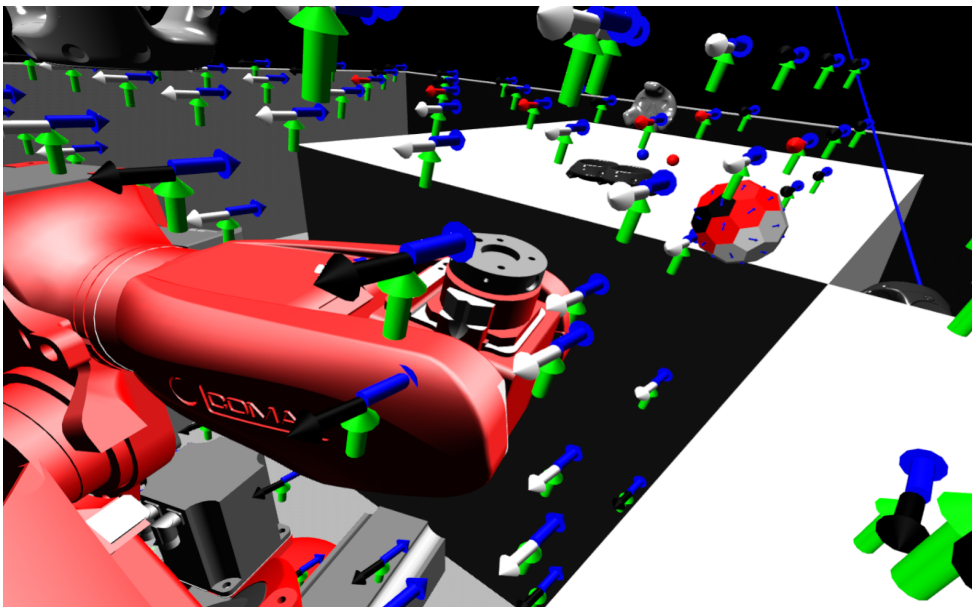


Figure 4.1: Visualization in VR

In order to provide users with an immersive experience, the time for a sampling with a fixed pose is controlled within 50 microseconds. For fixed rotation sampling and fixed translation sampling, the default configuration can let time also be controlled within 10 seconds.

Figur 4.1 shows the 6D posespace from the VR perspective. Users can sample and visualize poses in real time through the controller, there are 5 instructions bound to the controller.

1. Press the upper side of the touchpad of controller, a fixed pose sampling will be performed based on the transformation of the current controller.

2. Press the left side of the touchpad, a sampling for a fixed translation will be performed based on the current position of the controller.

3. Press the right side of the touchpad, a fixed rotation sampling will be performed based on the current orientation of the controller.

4. By press the bottom of the touchpad, the currently existing visualization of sampling will be cleared. This helps users to re-select the sample they want when there is a lot of messy visual information.

5. By clicking the trigger button of the controller, the collection strategy of the controllers transformation will be switched. Default strategy is save the transformation of the controller as accurately as possible, this allows users to easily obtain the pose they want to sample. The other strategy is map the position of controller to the position on the grid, this can prevent the visualization of the two samples from overlapping each other as much as possible.Under this strategy, visualization will be clearer and more intuitive, but the accuracy will also be reduced. Users can freely switch between the two strategies at any time.

Through these 5 interactive methods, users can freely explore in VR and to compare different poses. Immersive exploration has many unique advantages.

1. With the help of built-in sensors, the VR glasses can not only calculate the viewing angle, but also the position of the viewer, so that the user can also move around the scene and view the posespace from different positions.

2. Through exploration, users can sample posespace based on their sense of space. By manipulating the controller, the user can imagine the hand as an endeffector of the robot to test whether the pose is in the posespace.

3. With exploration, users can compare different poses and perceive, at which angle the robot has the largest range of motion, at which points the robot can be more flexible, or in which direction and position the robot will collide with obstacles. Furthermore, users can also perceive where the robot can move freely, or where the robot cannot reach at all.

Nevertheless, immersive exploration has disadvantages as well.

1. The demand for real-time is very high, so the number of samples processed in each time units is strictly controlled. If large amount of posespace data is needed, it will be better to sample the data offline in advance.

2. Immersive exploration mainly relies on the perception of space, so it is difficult to quantify posespace data during the exploration process.

3. Stuttering or waiting will reduce the experience of immersive exploration

5 Conclusion

This project proposed a new sampling and visualization scheme for robot poses and applied it to a VR software, allowing users to immersive exploring the 6D configuration spaces.

Through the real-time calculation of IK, 3 arrows can be used to express the 6D information of an object in real time in VR, and the color can express whether it is in the posespace.

For a fixed position, Goldberg poly can be used to sample and visualize. Through the processing of the polygon surface color, the pose collision and reachability information can also be clearly expressed.

Although this sampling and visualization method is very innovative and practical, it still has many limitations in immersive exploration.

1. IK from RL can only deliver one result at one time. For a single pose, there may be more than one configuration of joints. In this case, our method are currently unable to express all the configurations.
2. IK is relatively slow, while VR demands high frame rate. It is hard to sample multiple situation at real-time. The increase in the number of samples at a time also means that the waiting time for calculation might be longer.
3. In some cases, IK cannot find the solution at one-time sample, even there exists a solution. This will directly lead to errors in conveying visual information.
4. It is intuitive to use the surface of a convex polyhedron to represent colors, but because it does not represent the range of angles but only a single angle, some other angle information that is inconsistent with the color will be hidden by the this color of faces.
5. When the color of a certain surface is uniform, the information of the number of divided triangles will be lost, which means, without prior notice, the spin sampling information of end effector will be lost.

In addition, there is also still a lot of room for improvement.

1. IK can be solved in a background thread, this can greatly improve the performance of sampling and greatly enhance the experience of immersive exploration.
2. Use the *TRAC_IK* solver to replace the current IK solver. *TRAC_IK* can handles joint-limited chains much better without increasing time consumed during the process.[BA15]

Bibliography

[BA15] Patrick Beeson and Barrett Ames. Trac-ik: An open-source library for improved solving of generic inverse kinematics. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 928–935. IEEE, 2015.

[BPW93] Norman I Badler, Cary B Phillips, and Bonnie Lynn Webber. *Simulating humans: computer graphics animation and control*. Oxford University Press, 1993.

[Bus04] Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 17(1-19):16, 2004.

[CBGS08] JH Conway, H Burgiel, and C Goodman-Strass. Chapter 21: Naming archimedean and catalan polyhedra and tilings. *The symmetries of things. AK Peters*, 2008.

[DH55] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. ASME E, Journal of Applied Mechanics*, 22:215–221, June 1955.

[Die06] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35, 2006.

[Har13] George Hart. Goldberg polyhedra. In *Shaping space*, pages 125–138. Springer, 2013.

[KK02] Dohyung Kee and Waldemar Karwowski. Analytically derived three-dimensional reach volumes based on multijoint movements. *Human factors*, 44(4):530–544, 2002.

[LBSH08] Wildan Lalo, Thorsten Brandt, Dieter Schramm, and Manfred Hiller. A linear optimization approach to inverse kinematics of redundant robots with respect to manipulability. In *The 25th International Symposium on Automation and Robotics in Construction. ISARC*, pages 175–180, 2008.

[MG18] Abhijit Makhal and Alex K Goins. Reuleaux: Robot base placement by reachability analysis. In *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pages 137–142. IEEE, 2018.

[Pau81] Richard P Paul. *Robot manipulators: mathematics, programming, and control: the computer*

control of robot manipulators. Richard Paul, 1981.

[QFFL] Morgan Quigley, Josh Faust, Tully Foote, and Jeremy Leibs. Ros: an open-source robot operating system.

[RG17] Markus Rickert and Andre Gaschler. Robotics Library: An object-oriented approach to robot applications. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 733–740, Vancouver, BC, Canada, September 2017.

[VAD13] Nikolaus Vahrenkamp, Tamim Asfour, and Rüdiger Dillmann. Robot placement based on reachability inversion. In *2013 IEEE International Conference on Robotics and Automation*, pages 1970–1975. IEEE, 2013.

[ZBH07] Franziska Zacharias, Christoph Borst, and Gerd Hirzinger. Capturing robot workspace structure: representing robot capabilities. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3229–3236. Ieee, 2007.

Acknowledgments

My deepest gratitude goes first and foremost to Professor Gumhold, my supervisor, for his constant encouragement and guidance. At the very moment when Covid-19 was raging, we maintained a relatively stable frequency of online meetings. He paid great attention to the success of my thesis and provided me with a lot of suggestions and help during my writing process. His serious attitude towards academics and his guidance on key parts of my project are not only the key elements for my successful completion of the thesis, but also gave me a lot of inspiration for my future studies, career and ideas for life.

Second, I would like to express my heartfelt gratitude to LIN Tianfang and YU Zhongyuan. There was an old saying in China: You will understand the gain and loss if you take people as mirror. A good companion can make you realize your shortcomings and keep improving. LIN has more experience than me in how to independently complete projects and write papers as a student, constantly provided valuable advice to me, which broadened my mind. YU and I exchanged a lot of opinions on how to carry out a project by our own, he saves me from my narrow mindset. Moreover, LIN, thank you for helping me open the door of the laboratory every time!

Last my thanks would go to my beloved family for their loving considerations and great confidence in me all through these years. I'd like to thank my girl friend. Many thanks for my cat, every time when I feel a bottleneck in writing, he will let me hold and would not run away.

Thank you to everyone I met, and hope that we can survive the difficult time raging on Covid19 and have a better life.

Copyright Information