Vietnamese-German University

Elective Subject project

Computer assisted surgery

Trần Thanh Long

Matriculation: 1068054

EEIT2011-B41

May 2015

Declare of authorship

I hereby declare that I conducted this project in the Faculty of Computer Science and Engineer, Vietnamese-German University, under the supervision of Prof. Peter Nauth and Lab Engineer Le Dinh Than. All works I finished individually and they have been not in previous application. All source of knowledge used have been duly acknowledged.

29.5.2015

Signature

………………………………

Matriculation: 1068054

Acknowledgements

I studied and worked with this project because of deep interest with medical application. Prof. Peter Nauth has also given me much inspirations as he was lecturing about robot applications. His helpful guidance and sincere encouragement stimulated my ability and goals. So I would appreciate deeply him for what he gave and taught to me.

I also thank to Lab Engineer Lê Đình Thận for his guidance as well as extra sessions which improve my knowledge.

To my classmates I would also appreciate Nguyễn Vũ Hội, Trần Quang Trung for giving me useful information and instructions about ROS and Gazebo applications.

Abstract

Today computers as well as robots have spread widely into our life, they are helping us enhance our life quality as it become easier and faster. Medicine has gradually applied computer technique and robot to reduce human limits (accuracy, long-time working, psychological problems….). This concept has inspire me to build a robotic arm system assisting surgeons. There are also many similar systems which have built successfully worldwide but Viet Nam. I would try to make it real for my country. I hope I could build a good systems as others and improve medical quality.

Table of contents

Page

List of figures

Page

1. Introduction

1.1/ Concepts and ideas

“Minimally invasive surgery (MIS) is a type of surgery performed through several small incisions (usually less than one inch in diameter), or puncture site. It features some outstanding advantages compared to traditional open surgery with significantly reduce tissue traumas and decrease recovery time. Therefore, this procedure has become a common method during the last decades.”- *Medical Image Processing Systems for Minimally Invasive Spine Surgery-YanQiu Chen and PeiLli Sun.*

Though this technique has improved, surgeons still have limitation, they are not accurate with that scale for long time and there are quite lot impacts. So medicine must looking for help from robots. They are stable and accurate positioning, moreover they can be automatic.

Computer assisted surgery is a system which contains robotic arms conducting the surgery while the surgeon control it via controller. All hardware should be manipulated by software to make it easier for surgeons. The diagram show how this systems was organized.

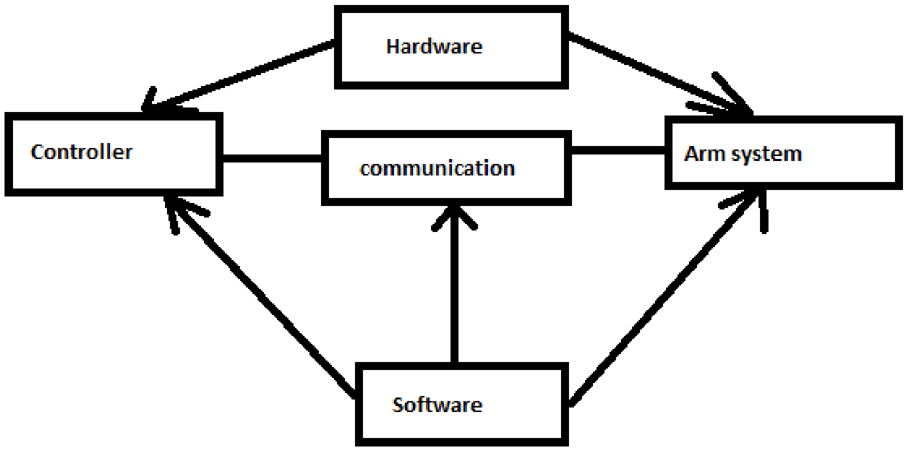


Fig. 1.1 Overview of System

1.2/ Design

1.2.1/ Design of arm systems

Robotic arms have been developed so strongly that they could do almost what human's hand could do. However, the complexity of surgery is high, the system has at least three arms to manipulate what compulsory actions in an operation. The most famous system is da Vinci Surgical System [1]:

[1] http://www.davincisurgery.com/da-vinci-surgery/da-vinci-surgical-system/

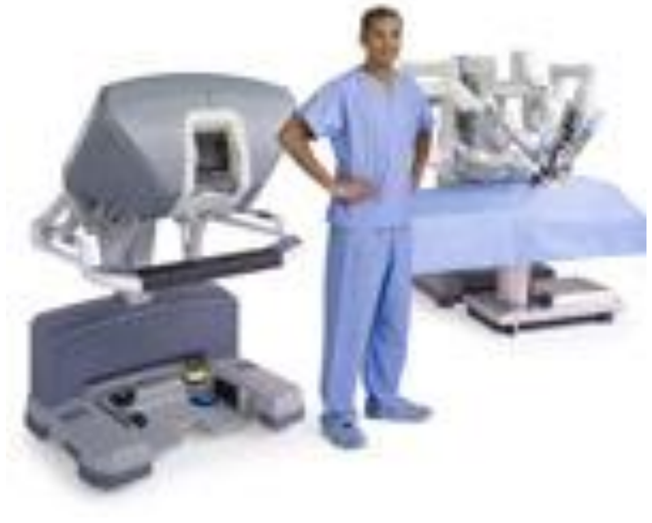


Fig 1.2.1.1[2] Whole system

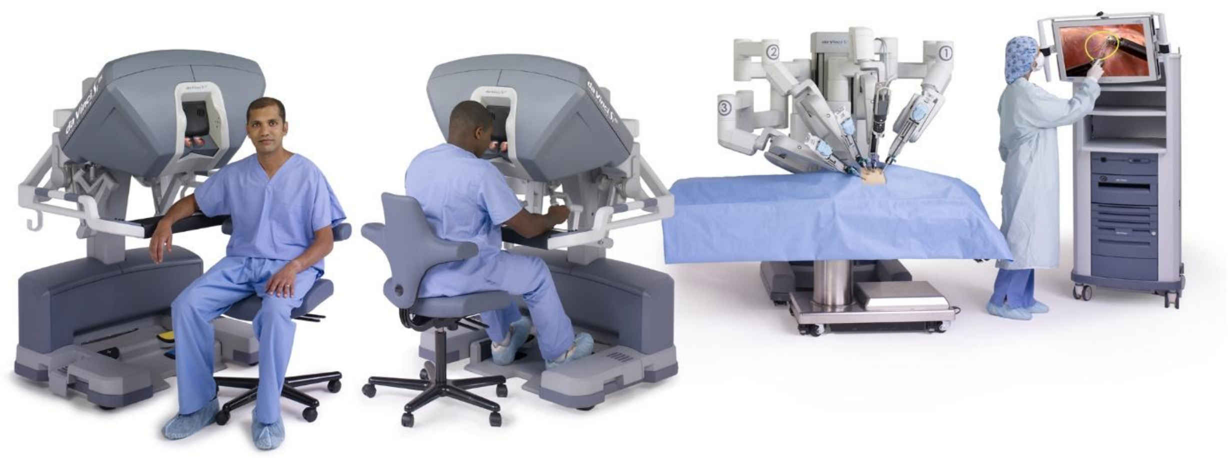


Fig 1.2.1.2[3] Whole system with surgeons

[2] http://www.davincisurgery.com/da-vinci-surgery/da-vinci-surgical-system/features.php
[3] http://www.intuitivesurgical.com/company/media/images/systems-si/000628_si_quad_dual_console_2_surgeons_2000x792.jpg

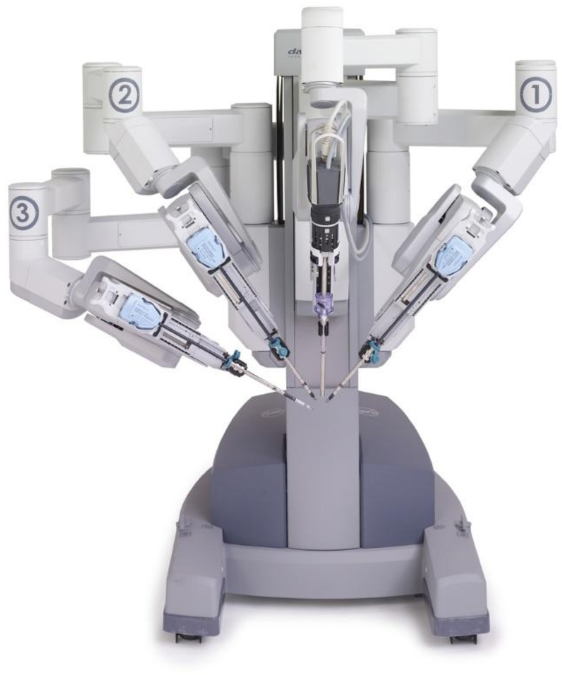


Fig 1.2.1.3[4] Arm system

However, I have decided to change a little bit about the structure, which contains three arm with changeable end-effector. My systems could be similar to MiroSurge system:

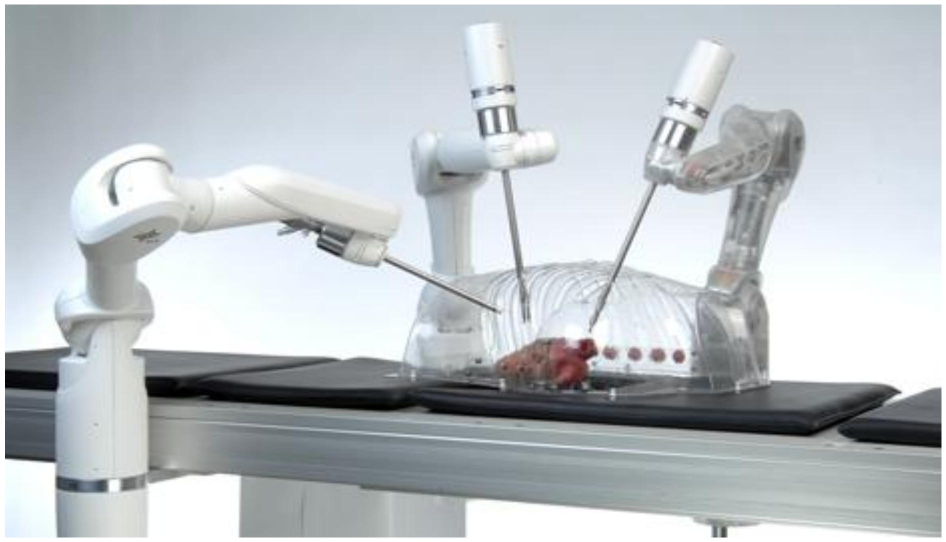


Fig 1.2.1.4[5] MiroSurge systems

[4] http://www.intuitivesurgical.com/
[5] http://www.dezeen.com/2013/11/18/robot-surgeons-to-operate-on-beating-human-hearts/

The system will contains three arms with same structure: has 7 DOF (degree of freedom) and good flexibility, moreover the end-effector could be change to manipulate right functions:

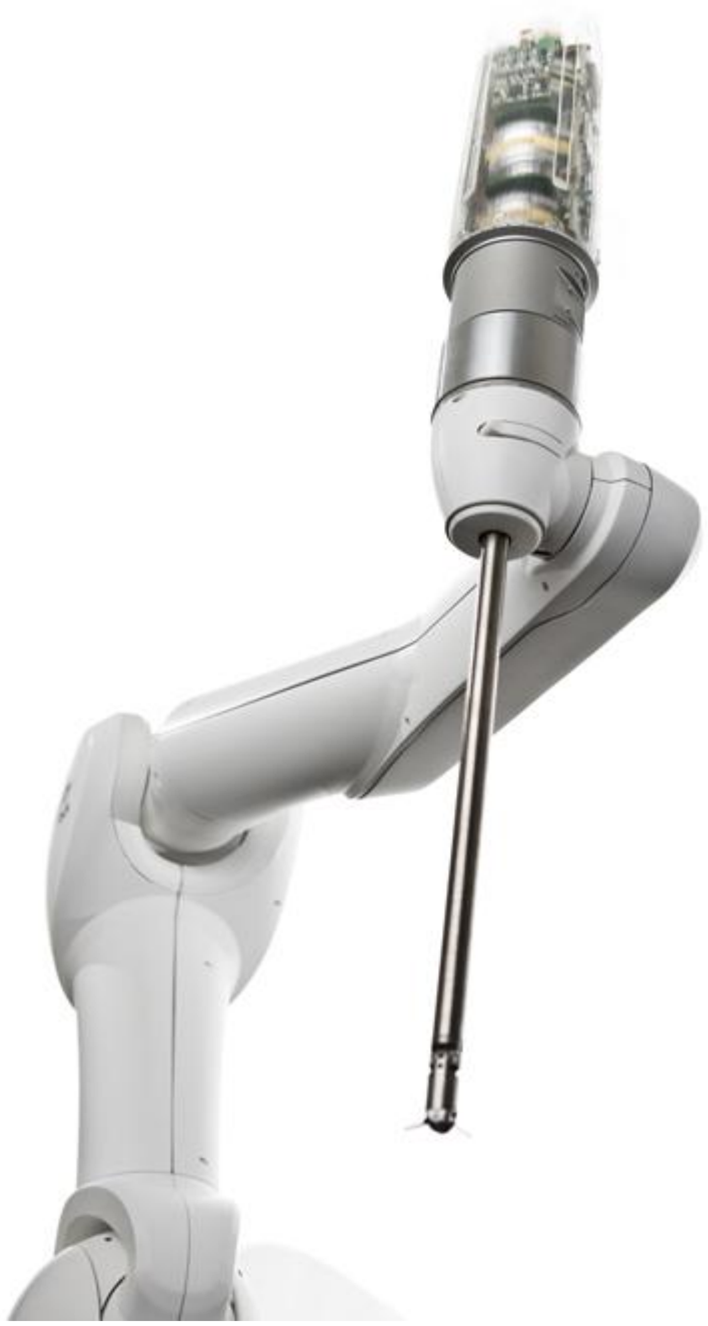


Fig 1.2.1.5[6] single arm with tool tip

[6] http://www.dezeen.com/2013/11/18/robot-surgeons-to-operate-on-beating-human-hearts/

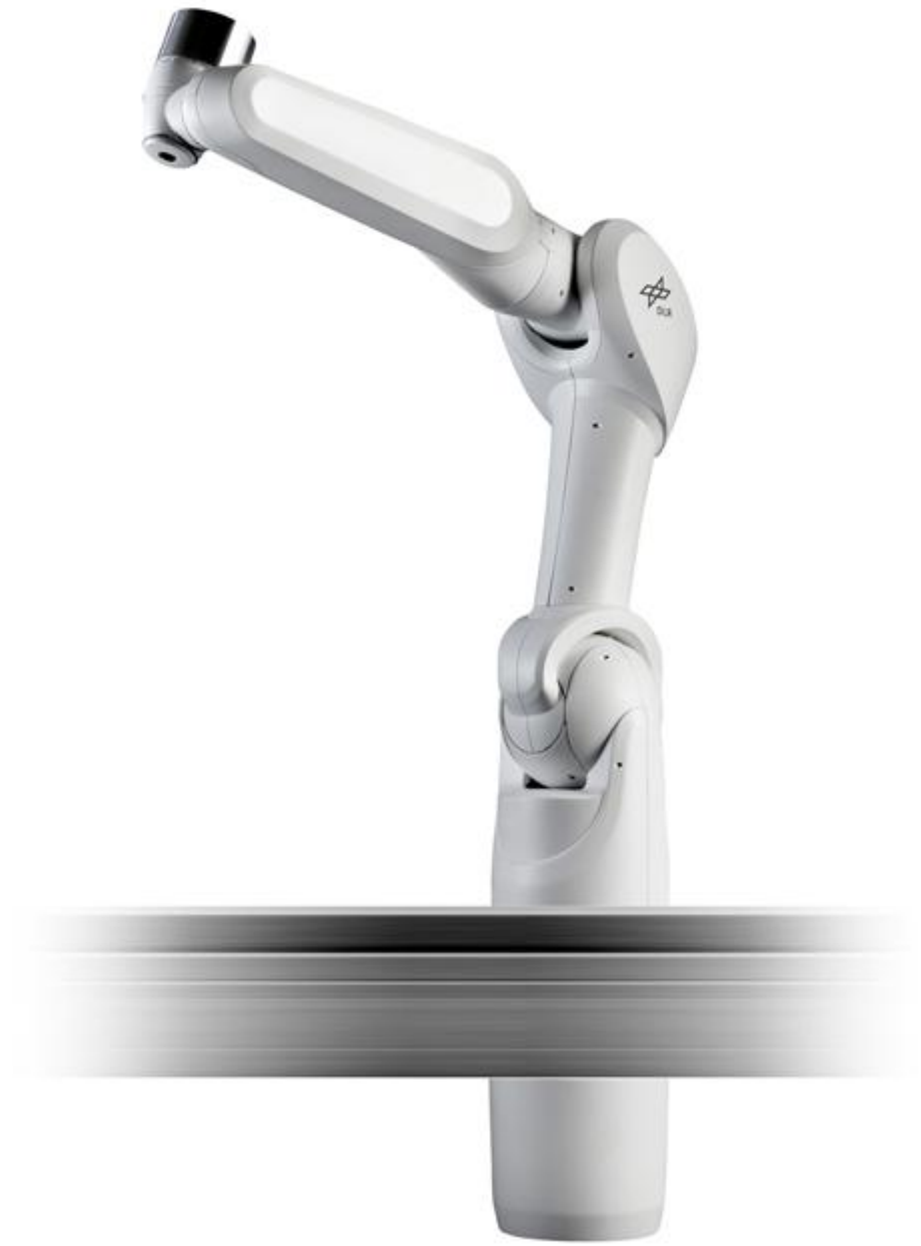


Fig 1.2.1.6[7] Single arm without tool tip

Due to limit of time, I managed to build one arm model with 5 DOF and simulated it on computer.

[7] http://www.dezeen.com/2013/11/18/robot-surgeons-to-operate-on-beating-human-hearts/

2. Required knowledge

2.1/ Physics: torque, force calculating

We could apply some simple physic laws to calculate torque at joint as describes below:

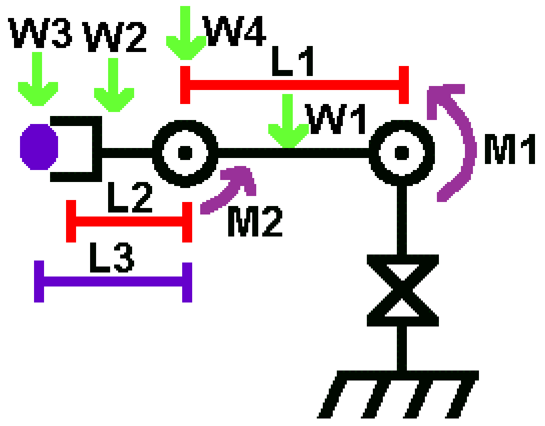


Fig 2.1.1[8] arm joints

Torque about Joint 1:

M1 = L1/2 * W1 + L1 * W4 + (L1 + L2/2) * W2 + (L1 + L3) * W3

Torque about Joint 2:

M2 = L2/2 * W2 + L3 * W3

2.2/ D-H parameters

Denavit-Hartenberg parameters are 4 parameters used to describe the relationship between joints and links in robotic arm. From these parameters we can calculate matrix transformation from one reference frame to another frame.

What is joint and link? We could see demonstration as showed below:

[8] http://www.societyofrobots.com/robot_arm_tutorial.shtml

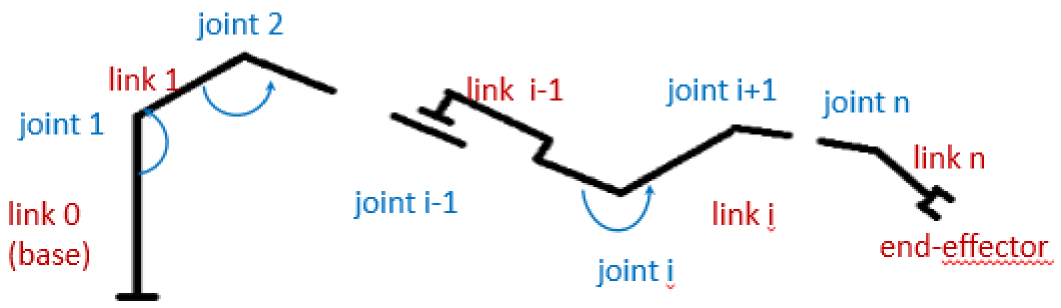


Fig 2.2.1 Links and joints

This diagram shows a robotic arm with n links and n joints. And D-H parameters are illustrated as picture below:[9]

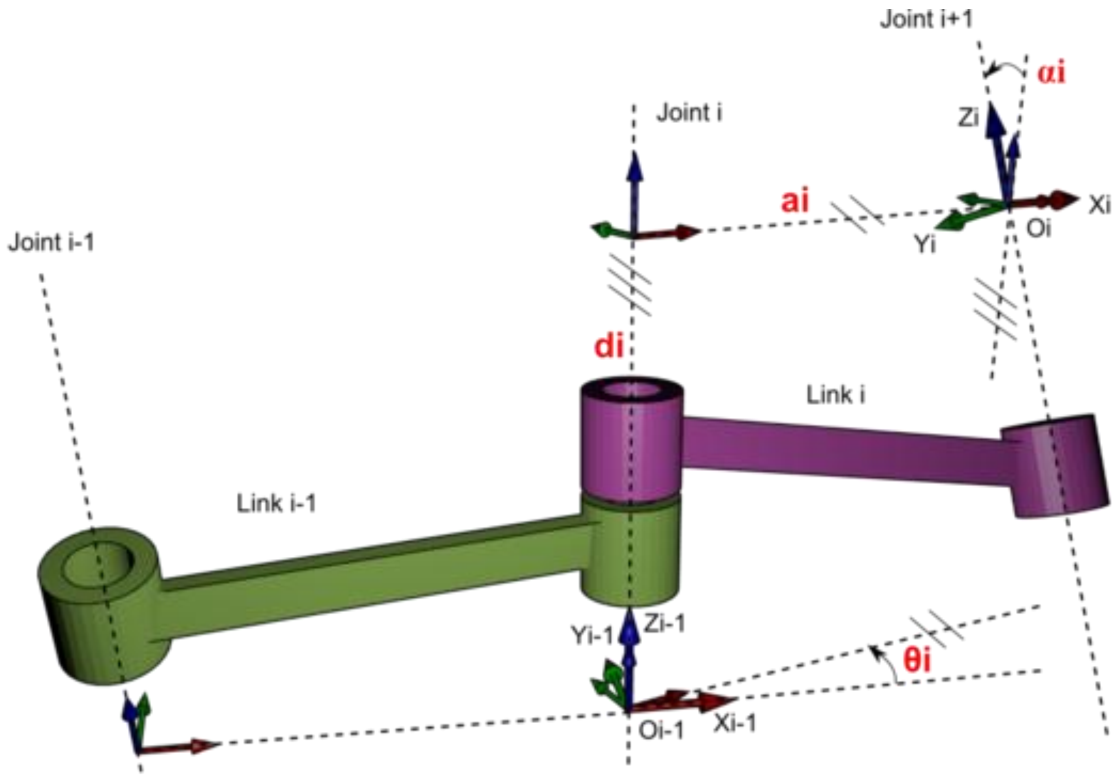


Fig 2.2.2 DH parameter determining

[9]http://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters

The four parameters of classic DH convention are shown in red text, which are $\theta_i, d_i, a_i, \alpha_i$. With those four parameters, we can translate the coordinates from $O_{i-1}X_{i-1}Y_{i-1}Z_{i-1}$ to $O_iX_iY_iZ_i$.

Here we have:

d: offset along previous z to the common normal
θ: angle about previous z, from old x to new x
a: length of the common normal. Assuming a revolute joint, this is the radius about previous z.
α: angle about common normal, from old z axis to new z axis

2.3/ Transformation matrix

To transform from link n – 1 to link n, we have to multiply 4 matrix which are corresponding to translation d_n, translation a_n (or r_n), rotation θ_n and rotation α_n. Here we have ${}^{n-1}T_n$:

$$ {}^{n-1}T_n = \mathrm{Trans}_{z_{n-1}}(d_n) \cdot \mathrm{Rot}_{z_{n-1}}(\theta_n) \cdot \mathrm{Trans}_{x_n}(r_n) \cdot \mathrm{Rot}_{x_n}(\alpha_n) $$

We look into these matrix to see how transformation happens. Each type of transformation require a type of matrix whereas the parameters are included. Here we have: [10]

$$ \mathrm{Trans}_{z_{n-1}}(d_n) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ \hline 0 & 0 & 0 & 1 \end{array}\right] $$

$$ \mathrm{Rot}_{z_{n-1}}(\theta_n) = \left[\begin{array}{ccc|c} \cos\theta_n & -\sin\theta_n & 0 & 0 \\ \sin\theta_n & \cos\theta_n & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] $$

[10] http://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters

$$\mathrm{Trans}_{x_n}(r_n) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & r_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$\mathrm{Rot}_{x_n}(\alpha_n) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_n & -\sin\alpha_n & 0 \\ 0 & \sin\alpha_n & \cos\alpha_n & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$${}^{n-1}T_n = \left[\begin{array}{ccc|c} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & r_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & r_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} & R & & T \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

where *R* is the 3×3 submatrix describing rotation and *T* is the 3×1 submatrix describing translation.

We can take an example of spherical arm[11]:

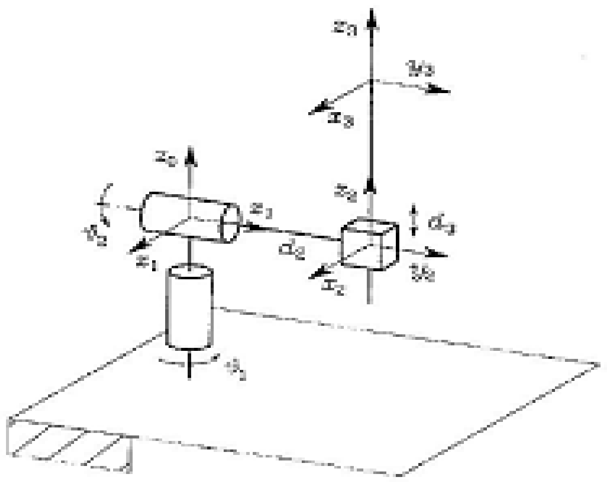


Fig 2.3.1 Spherical arm

[11] Direct kinematics –Mr. Le Dinh Than 's slides

We have table of D-H parameters:

Link	a_i	α_i	d_i	v_i
1	0	$-\pi/2$	0	v_1
2	0	$\pi/2$	d_2	v_2
3	0	0	d_3	0

For each joint we calculate the transformation matrix:

$$A_1^0(v_1) = \begin{bmatrix} c_i & 0 & -s_i & 0 \\ s_i & 0 & c_i & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2^1(v_2) = \begin{bmatrix} c_i & 0 & s_i & 0 \\ s_i & 0 & -c_i & 0 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^2(v_3) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then we have a final transformation matrix by multiply them respectively,

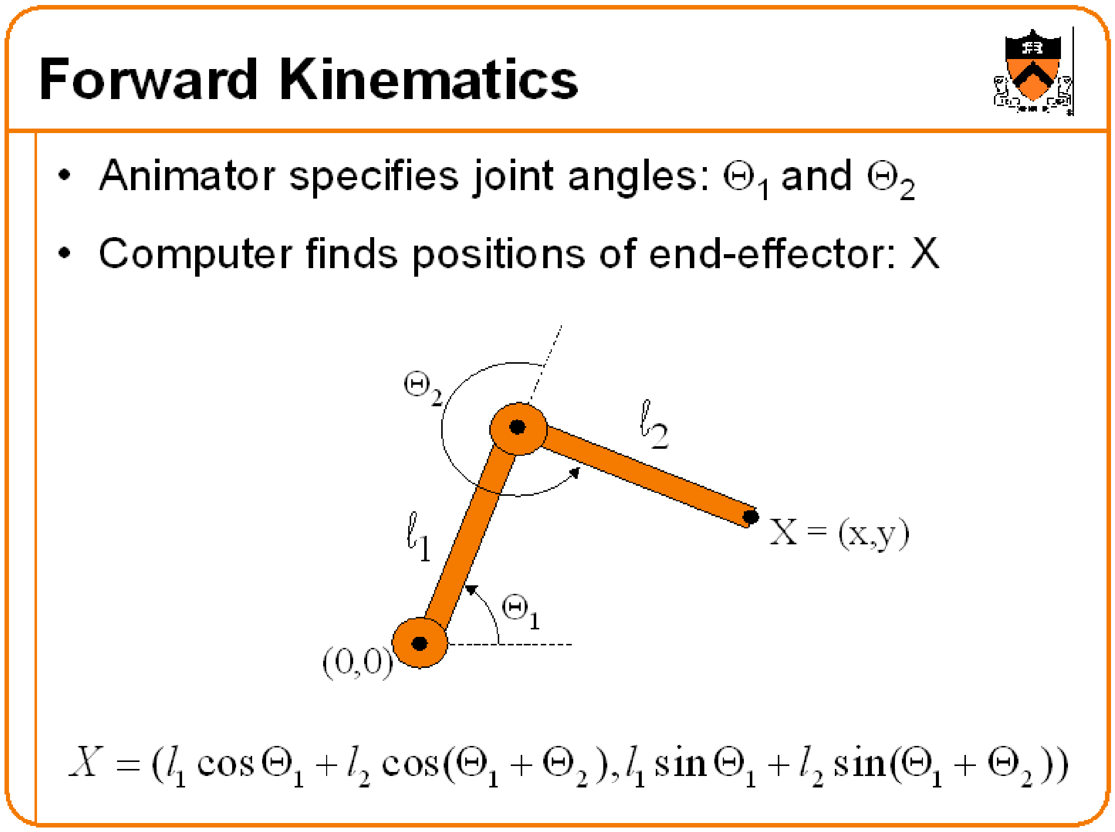
$$T_3^0(q) = A_1^0 A_2^1 A_3^2 = \begin{bmatrix} c_1c_2 & -s_1 & c_1s_2 & c_1s_2d_3 - s_1d_2 \\ s_1c_2 & c_1 & s_1s_2 & s_1s_2d_3 - c_1d_2 \\ -s_2 & 0 & c_2 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.4/ Kinematics

2.4.1/ Forward kinematics or direct kinematics

Problems: Given joint parameters we calculate the position of end-effector (last joint of robotic arm)

Basic solution for 2 joints robotic arm is illustrated below[12]:



Forward Kinematics

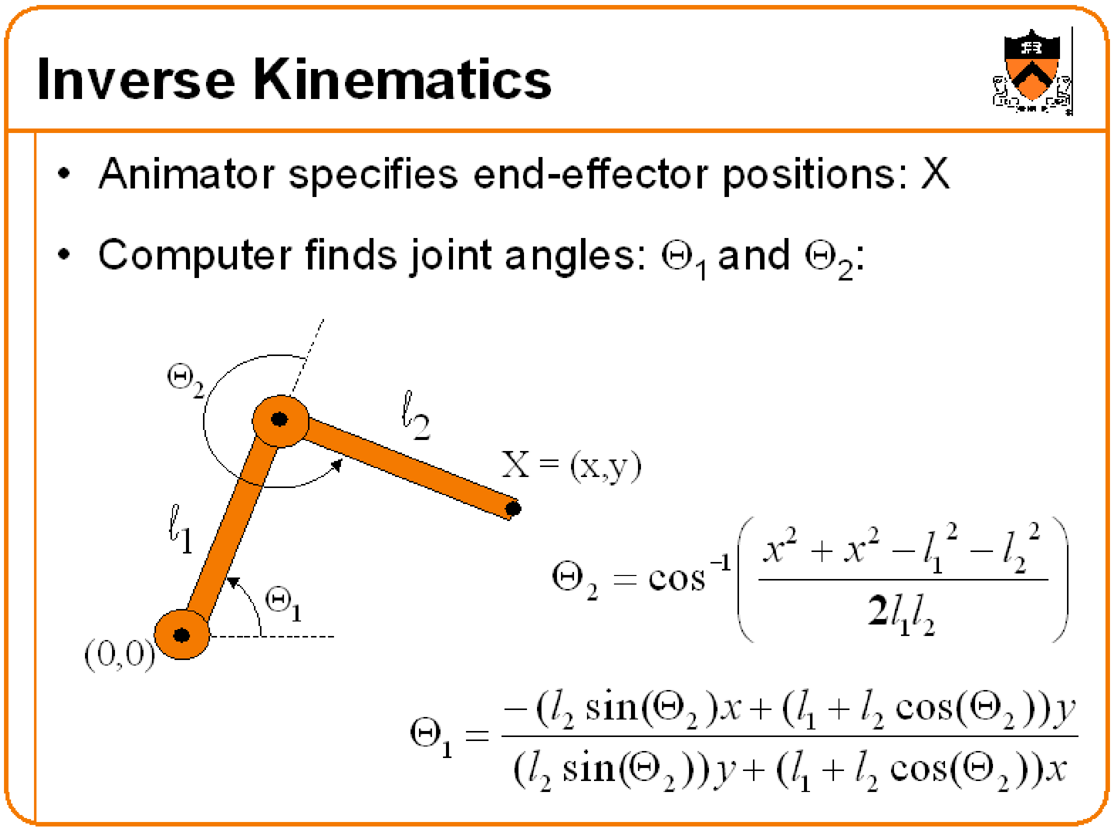
- Animator specifies joint angles: Θ_1 and Θ_2
- Computer finds positions of end-effector: X

$$X = (l_1 \cos\Theta_1 + l_2 \cos(\Theta_1 + \Theta_2), l_1 \sin\Theta_1 + l_2 \sin(\Theta_1 + \Theta_2))$$

2.4.2/ Inverse Kinematics

Problem: contradict to direct kinematics, given the position of end-effector, calculating the parameters of each joints. Inverse kinematics is complex and has many solution for one problem. Here is the solution for two joints arm[13]:

[12] https://www.cs.princeton.edu/courses/archive/fall00/cs426/lectures/kinematics/sld004.htm
[13] http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/kinematics/sld008.htm



Inverse Kinematics

- Animator specifies end-effector positions: X
- Computer finds joint angles: Θ_1 and Θ_2:

$$\Theta_2 = \cos^{-1}\left(\frac{x^2 + x^2 - l_1^{\,2} - l_2^{\,2}}{2 l_1 l_2}\right)$$

$$\Theta_1 = \frac{-(l_2 \sin(\Theta_2)x + (l_1 + l_2 \cos(\Theta_2))y}{(l_2 \sin(\Theta_2))y + (l_1 + l_2 \cos(\Theta_2))x}$$

2.5/ Solidwork-Model design

Solidwork is a very strong and professional software for mechanics/ architecture design. It helps designer create 3D model with high complexity as well as many factors integration (force, toque, materials…). To build robotic arm, I first have the model design then testing it by simulation through Gazebo and ROS. However, the beginning step does not require a complex model design, just drawing basic shape (sphere, cubic or cylinder…) because Gazebo just support simple models.

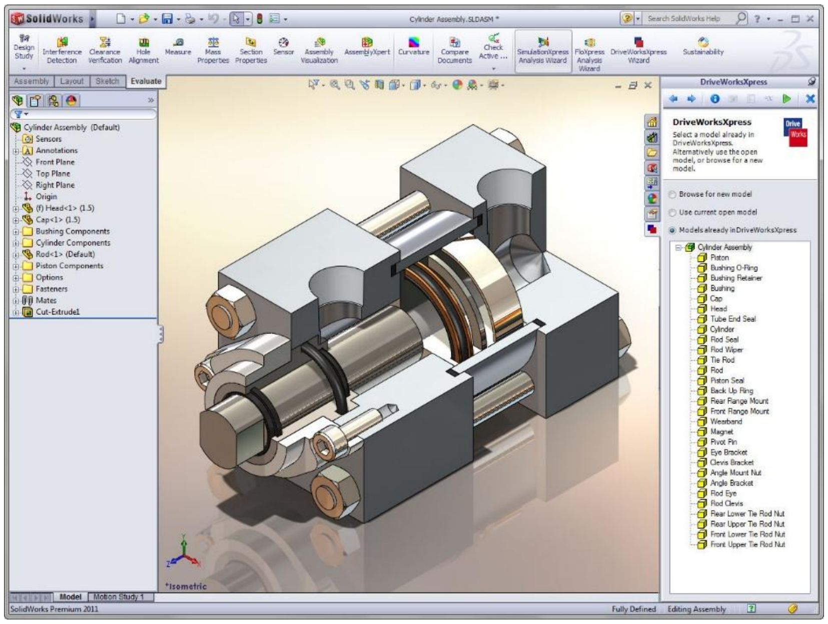


Fig 2.5.1[14] SolidWorks sample

2.6/ ROS and Gazebo

2.6.1/ ROS

"ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license"- wiki.ros.org

ROS is built on Linux so I must get used with it, the most challenging obstacle is switch to a new environment. To understand basically this tool, tutorials from http://wiki.ros.org/ are very helpful.

Beginning with ROS, I must get familiar with package (Linux unique characteristic), terminal (command line in Windows), configure myself everything and install app by coding. These works cost a lot of time but seem to be very efficient in future.

2.6.2/ Gazebo

[14] www.solidworks.com

"Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, and perform regression testing using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. At your fingertips is a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. Best of all, Gazebo is free with a vibrant community." - http://gazebosim.org/

With Gazebo, we can observe our robot's actions to fix it, to improve its design. The main task to do with Gazebo is coding sdf file which is similar to a source file of any programming languages. Its structure is similar to HTML language (mainly tag and information).

The most problem is how to integrate or connect Gazebo with ROS, these could cause a lot of errors when running simulations.

With these basic knowledge, we can understand and build a simple robotic arm. But to create a surgery robotic arm, we must also apply force sensors-feedback control loop, image processing and advance surgical techniques.

2.7/ Force sensors and feedback control loop

Force sensors are attached at tool tip of the arm, when tool tip interact with tissue or obstacle, it send back sensing force to hand of surgeon. That helps surgeon adjust force to suitable level. Here is block diagram of this process:

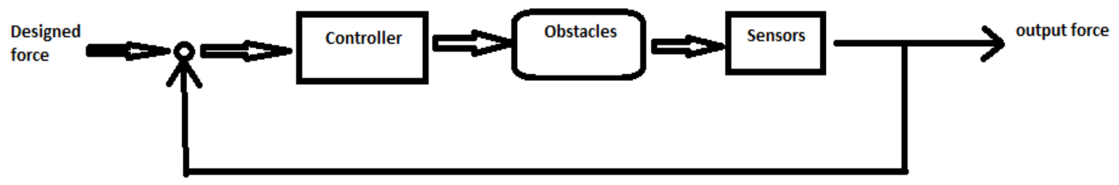


Fig 2.7.1 Diagram of control loop

This would be the main innovation of computer assisted surgery project because the surgeon can manipulation with highest conveniences, they can feel impact with patient to adjust the pressure or force exactly.

In addition, we can build automation system based on this feature, to do primary surgical actions like: suturing, bandage binding…

2.8/ Image processing

For automatic surgery, image processing would be applied to help computer control robot to manipulate basic actions. Pattern recognition and classifier play an important role in this setting. Because the human‘s anatomy has a very complex anatomical anatomic structure, the computers are easy to cause errors which could lead to lethal problems.

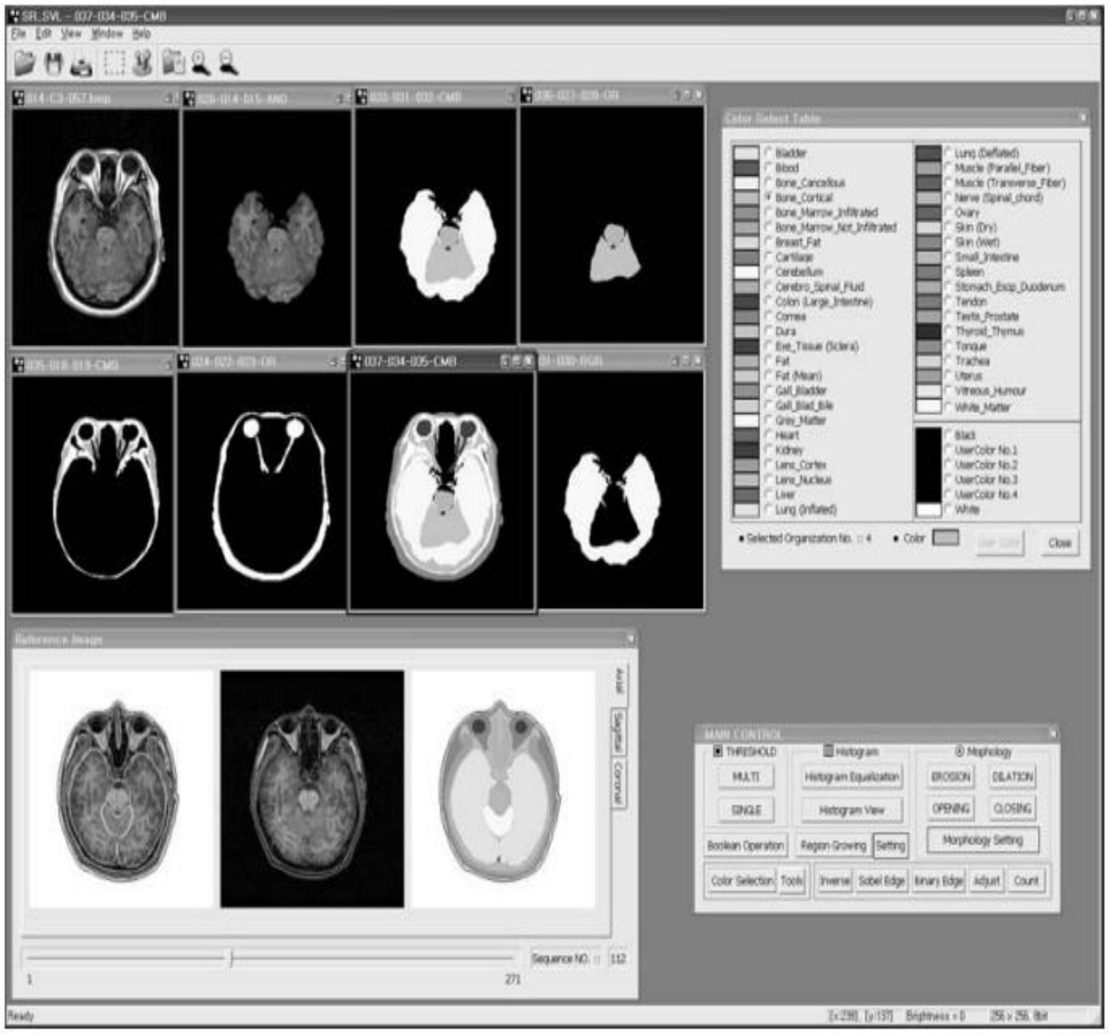
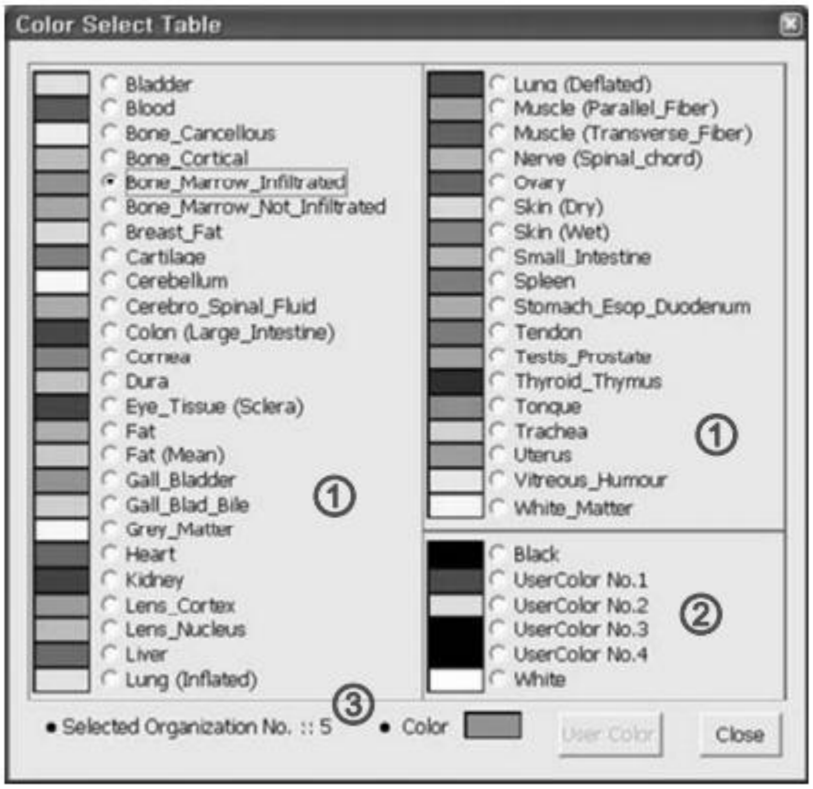


Fig 2.8.1[15] Overview of the user interface of Human Analyzer

[15, 13] *An Intuitive Menu Based Image Processing System for Medical Images*-Done-Sik Yoo, Woo Young Choi, Ji-Wook Jeong, Jeong Won Lee, Seunghwan Kim.


Color Select Table

Bladder
Blood
Bone_Cancellous
Bone_Cortical
[✓ Bone_Marrow_Infiltrated]
Bone_Marrow_Not_Infiltrated
Breast_Fat
Cartilage
Cerebellum
Cerebro_Spinal_Fluid
Colon (Large_Intestine)
Cornea
Dura
Eye_Tissue (Sclera)
Fat
Fat (Mean)
Gall_Bladder
Gall_Blad_Bile
Grey_Matter
Heart
Kidney
Lens_Cortex
Lens_Nucleus
Liver
Lung (Inflated)
Lung (Deflated)
Muscle (Parallel_Fiber)
Muscle (Transverse_Fiber)
Nerve (Spinal_chord)
Ovary
Skin (Dry)
Skin (Wet)
Small_Intestine
Spleen
Stomach_Esop_Duodenum
Tendon
Testis_Prostate
Thyroid_Thymus
Tongue
Trachea
Uterus
Vitreous_Humour
White_Matter

Black
UserColor No.1
UserColor No.2
UserColor No.3
UserColor No.4
White

● Selected Organization No. :: 5 ● Color

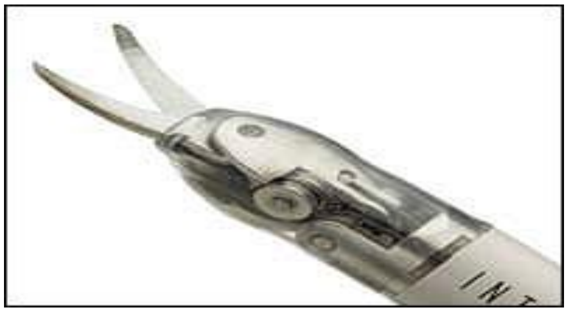
Fig 2.8.2[13] Tissue classified based on color

However, this technique should be integrated for further applications, it will lead to develop a humanoid robot that could replace surgeon completely.

Through this system is very complex to realization, this project should build other simple hardware to transmit vision inside human body, similar to endoscopy invasive surgery. We need to improve the resolution of transmitting videos or images to support surgeon thoroughly.

2.9/ Surgical techniques

A requirement for automatic arms is teaching them how to make a surgery procedure. Moreover, design of the tool tip must be appropriate in size, mobility and settings. There are some basic actions which could be manipulated by robot arm's tool tip: cutting-scissor shape tool tip (fig 2.9.1), holding – gripper tool tip (fig 2.9.2)…



Fig[16] 2.9.1 Scissor tool tip

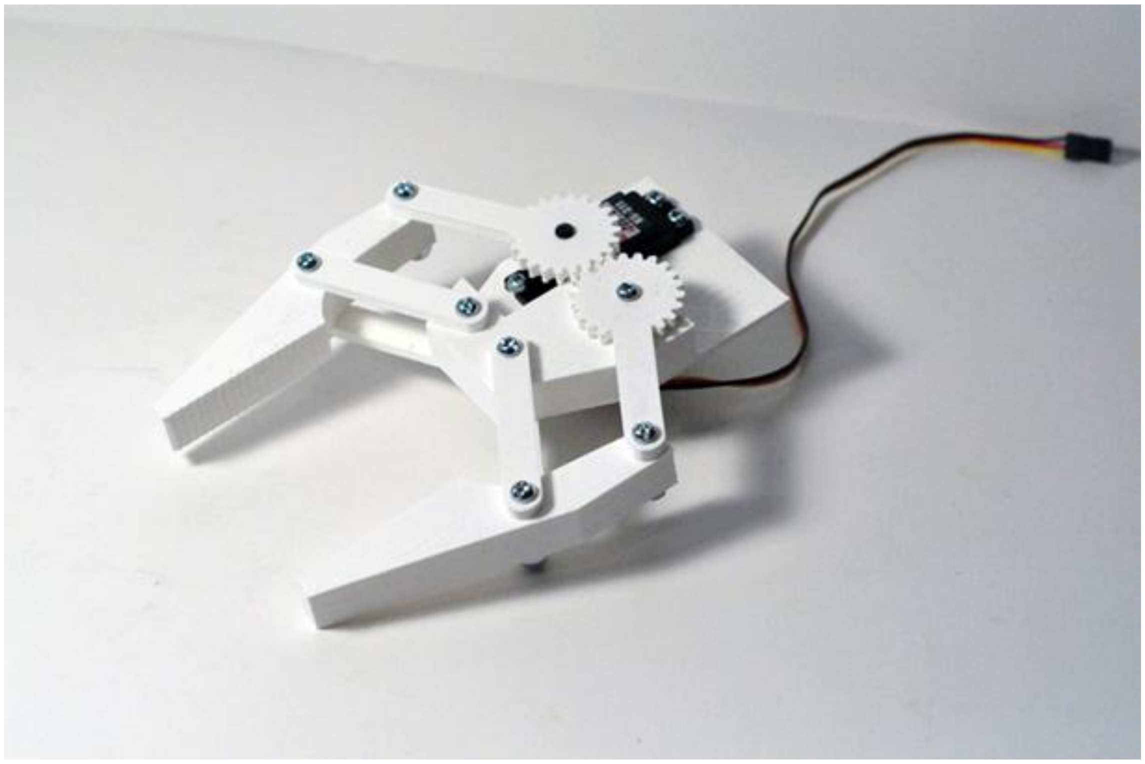


Fig 2.9.2[17] Gripper tool tip

[16] http://www.medscape.com/viewarticle/814129
[17] http://www.kwartzlab.ca/2012/05/3d-printed-robot-gripper/

3/ Model building

3.1/ Ideal and calculation

I design the robotic arm following the model of 5 DOF arm with high flexibility. Moreover, simple design could help simulate on gazebo more easily. Here is the sample of this arm:

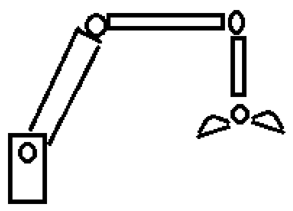


Fig 3.1.1 5 DOF arm

This arm will basically grab an object within the workspace as well as drag object to a known position. How it works depends on algorithm which solves direct and inverse kinematic problems.

To calculate the DH parameter, a drawing of reference frame of each joint is proficient. However, there is a trick to choose the right axis. Z axis would be the same with the axis of joint where X axis would be the same on the connecting axis.

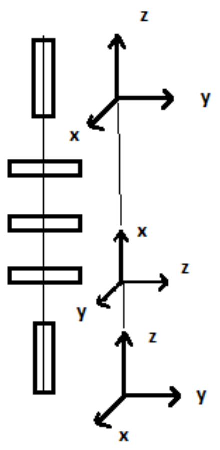


Fig 3.1.2 Reference frame of each joint

i	ai	αi	di	θi
1	0	-90’	0	q1
2	a1	0	0	q2
3	a2	0	0	q3
4	0	90	0	q4

Then I can calculate the transformation matrices:

$$ {}^{1}A_{2} = \begin{bmatrix} c1 & 0 & -s1 & 0 \\ s1 & 0 & c1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{2}A_{3} = \begin{bmatrix} c2 & -s2 & 0 & a2c2 \\ s2 & c2 & 0 & a2s2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$^{3}A_{4}=\begin{bmatrix} c3 & -s3 & 0 & a3c3 \\ s3 & c3 & 0 & a3s3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{4}A_{5}=\begin{bmatrix} c4 & 0 & s4 & 0 \\ s4 & 0 & c4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then we have transformation matrix:

$T={}^{1}A_{2}{}^{2}A_{3}{}^{3}A_{4}{}^{4}A_{5}=$

$$\begin{bmatrix} c1234-c12s23-c12s34-c13s24 & -s1 & c123s4-c1s234-c124s3-c134s2 & a3c123-a3c1s23+a2c12 \\ c234s1-c4s123-s134c2-c3s124 & c1 & c23s14-s1234-s13c24-c34s12 & a3c23s1-a3s123+a2s1c2 \\ -s2c34-c24s3+s234-c23s4 & 0 & -s24c3-c2s34+s23c4-c234 & -a3s2c3-a3c2s3-a2s2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.2/ Design on SolidWorks

Now I build the arm design on SolidWorks:

The dimensions of each part were also shown on pictures (centimeters)

The first part is the base:

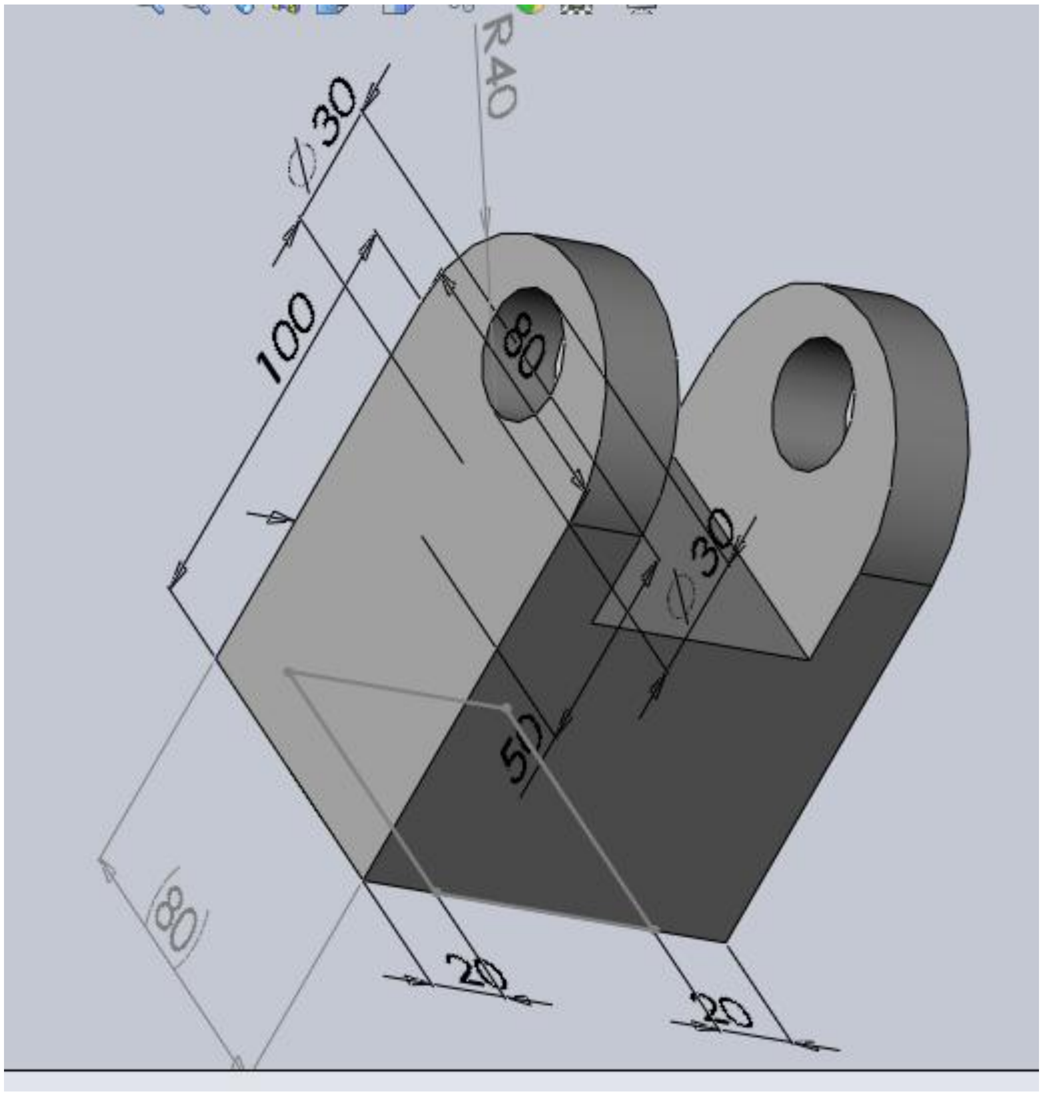


Fig 3.2.1 Base

The second part is the short arm:

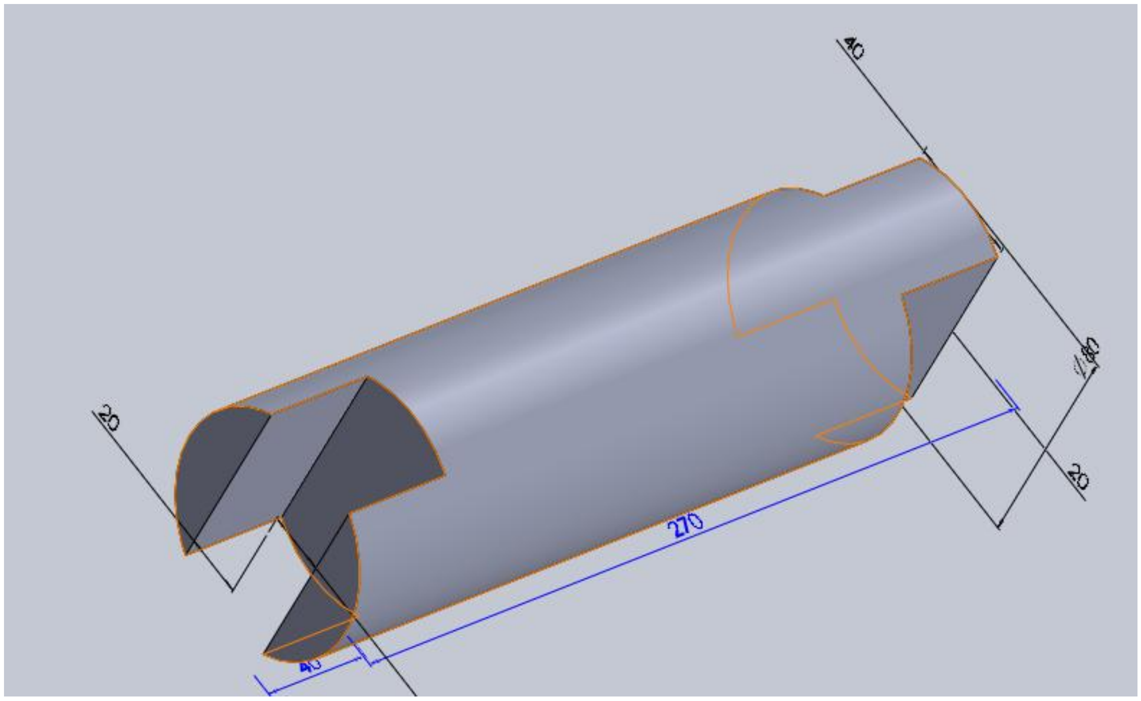


Fig 3.2.2 arm_1, arm 2

The third one is similar the arm_1

And the last part is the hand:

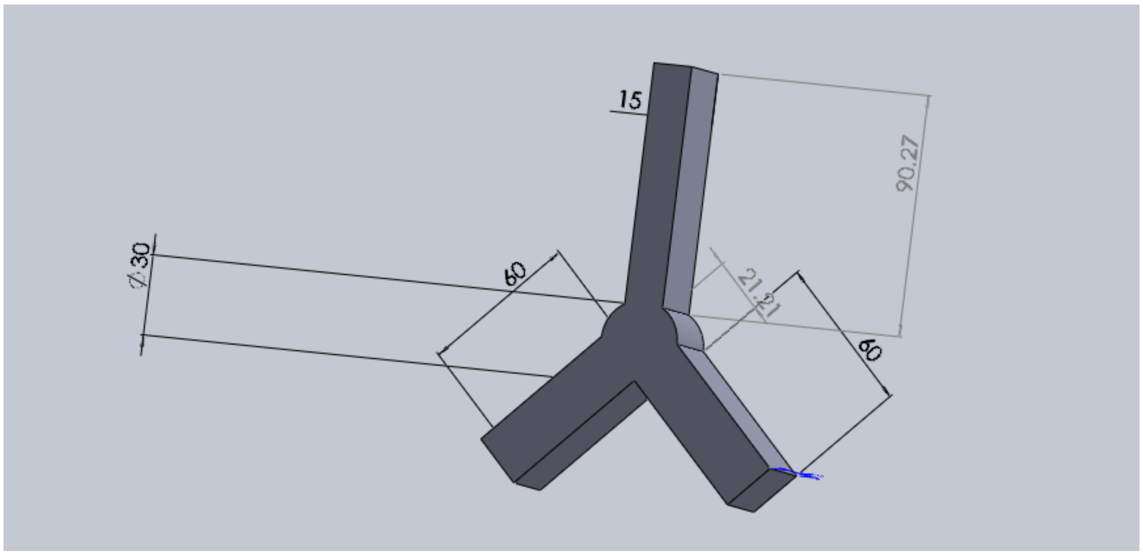


Fig 3.2.3 the hand

Then I combine all parts to make a complete robotic arm:

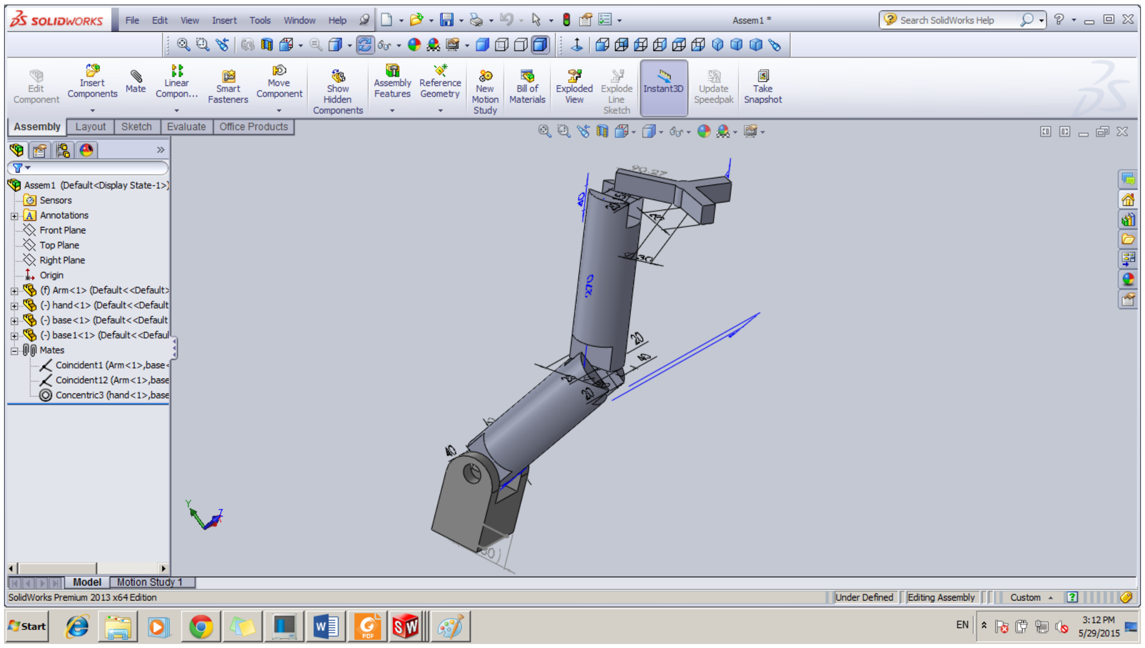

SOLIDWORKS
Assem1 *
Assem1 (Default<Display State-1>)
Sensors
Annotations
Front Plane
Top Plane
Right Plane
Origin
(f) Arm<1> (Default<<Default>
(-) hand<1> (Default<<Default
(-) base<1> (Default<<Default
(-) base1<1> (Default<<Defaul
Mates
Coincident1 (Arm<1>,base-
Coincident12 (Arm<1>,base
Concentric3 (hand<1>,base
Model
Motion Study 1
SolidWorks Premium 2013 x64 Edition
Under Defined
Editing Assembly
Custom
3:12 PM
5/29/2015

Fig 3.2.4a complete arm

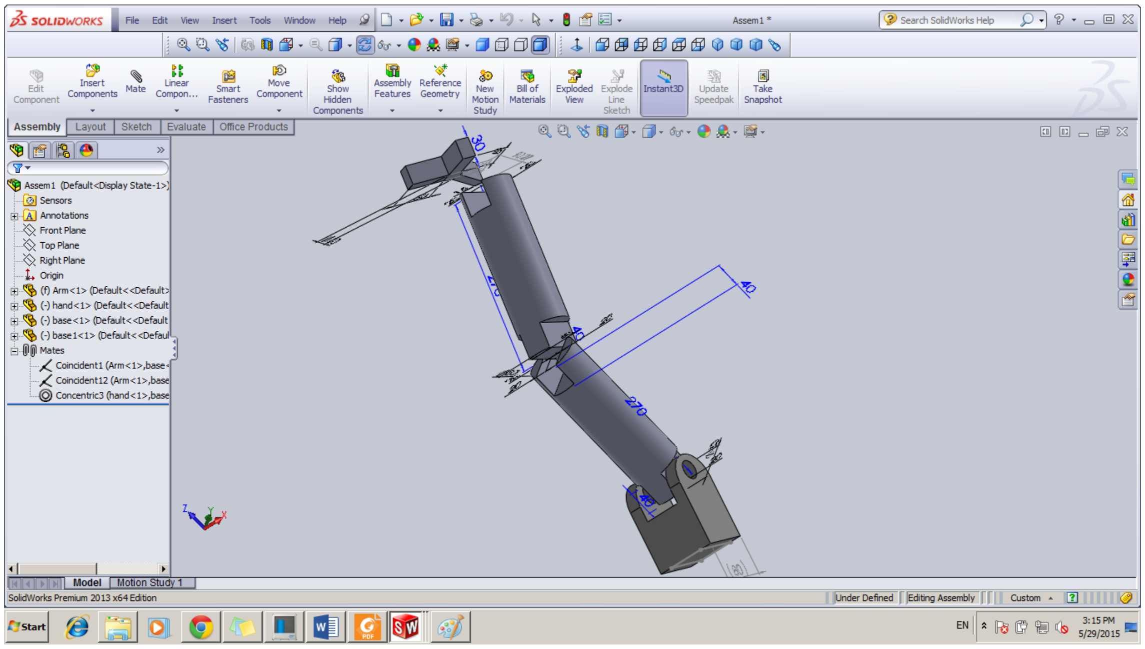

SOLIDWORKS
Assem1 *
Assem1 (Default<Display State-1>)
Sensors
Annotations
Front Plane
Top Plane
Right Plane
Origin
(f) Arm<1> (Default<<Default>
(-) hand<1> (Default<<Default
(-) base<1> (Default<<Default
(-) base1<1> (Default<<Defaul
Mates
Coincident1 (Arm<1>,base-
Coincident12 (Arm<1>,base
Concentric3 (hand<1>,base
30
270
40
270
Model
Motion Study 1
SolidWorks Premium 2013 x64 Edition
Under Defined
Editing Assembly
Custom
3:15 PM
5/29/2015

Fig 3.2.4b complete arm (alternative view)

However, this design just work on simulation because they are quite simple. To build a real arm we need to rebuild this design with more details and deep calculation to attach the motor as well as wires.

4/ Simulation

4.1/ ROS

4.1.1/ Configuration ROS environments

After installing of ROS, we must make some configuration to help it work correctly.

Create ROS workspace (catkin_workspace) by code in terminal windows:

```
$ mkdir -p ~/catkin_ws/src

$ cd ~/catkin_ws/src

$ catkin_init_workspace
```

Then sourcing the new setup.bash file

```
$ source devel/setup.bash
```

To make sure our workspace is properly overlayed by the setup script, make sure ROS_PACKAGE_PATH environment variable includes the directory we are in.

4.1.2/ Navigating ROS filesytem

First we will inspect a package in ros-tutorials, install it using

```
$ sudo apt-get install ros-<distro>-ros-tutorials
```

Filesystem has these tool:

- rospack: allows us to get information about packages.
- roscd: is part of rosbash suite. It allows you to change directory (cd) directly to a package or a stack.
- roscd log: will take us to the folder where ROS stores log files. Note that if we have not run any ROS programs yet, this will yield an error saying that it does not yet exist.
- rosls is part of the rosbash suite. It allows you to ls directly in a package by name rather than by absolute path.

4.1.3/ Creating ROS package

For a package to be considered a catkin package it must meet a few requirements:

- The package must contain a catkin compliant package.xml file

    - That package.xml file provides meta information about the package
- The package must contain a CMakeLists.txt which uses catkin. Catkin metapackages must have a boilerplate CMakeLists.txt file.
- There can be no more than one package in each folder
    - This means no nested packages nor multiple packages sharing the same directory

To create a package with dependencies, using:

```
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

To customize the package, we can change contents of package.xml and CMakeLists.txt then saving them.

4.1.4/ ROS nodes and ROS topics

Graph concepts

- Nodes: A node is an executable that uses ROS to communicate with other nodes.
- Messages: ROS data type used when subscribing or publishing to a topic.
- Topics: Nodes can *publish* messages to a topic as well as *subscribe* to a topic to receive messages.
- Master: Name service for ROS (i.e. helps nodes find each other)
- rosout: ROS equivalent of stdout/stderr
- roscore: Master + rosout + parameter server (parameter server will be introduced later)

Some command we could use to manage ROS nodes:

- roscore: is the first thing you should run when using ROS.
- rosnode: displays information about the ROS nodes that are currently running. The rosnode list: command lists these active nodes
- rosrun: allows you to use the package name to directly run a node within a package (without having to know the package path).
- roslaunch: starts nodes as defined in a launch file.

Topic

Topics are named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. In general, nodes are not aware of who they are communicating with. Instead, nodes that are interested in data subscribe to the relevant topic; nodes that generate datapublish to the relevant topic. There can be multiple publishers and subscribers to a topic.

Topics are intended for unidirectional, streaming communication. Nodes that need to perform remote procedure calls, i.e. receive a response to a request, should use services instead. There is also the Parameter Server for maintaining small amounts of state.

Topic tools

- rqt_graph: creates a dynamic graph of what's going on in the system. rqt_graph is part of the rqt package.
- rostopic echo: shows the data published on a topic.
- rostopic list: returns a list of all topics currently subscribed to and published.
- rostopic type: returns the message type of any topic being published.
- rostopic pub: publishes data on to a topic currently advertised.
- rostopic hz: reports the rate at which data is published.
- rqt_plot: displays a scrolling time plot of the data published on topics

With these basic tool, we could basically understand to ROS but to connect Gazebo with ROS, we need some extra steps which I shown on 4.3

4.2/ Gazebo

We used the Gazebo version 2.2.3 on Ubuntu Indigo. This software require a lot of setup and initial file to work.

4.2.1/ Files involved to start

World file:

The world description file contains all the elements in a simulation, including robots, lights, sensors, and static objects. This file is formatted using SDF (Simulation Description Format), and typically has a .world extension.

The Gazebo server (gzserver) reads this file to generate and populate a world.

A number of example worlds are shipped with Gazebo. These worlds are located in <install_path>/share/gazebo-<version>/worlds.

Model file:

A model file uses the same SDF format as world files, but should only contain a single <model> ... </model>. The purpose of these files is to facilitate model reuse, and simplify world files. Once a model file is created, it can be included in a world file using the following SDF syntax:

```
<include>
  <uri>model://model_file_name</uri>
</include>
```

A number of models are provided in the online model database (in previous versions, some example models were shipped with Gazebo). Assuming that we have an Internet connection when running Gazebo, we can insert any model from the database and the necessary content will be downloaded at runtime.

Environment variables

Gazebo use a number of environment variables to locate files, and setup the communication between the server and clients.

Gazebo server

The server is the workhorse of Gazebo. It parses a world description file given on the command line, and then simulates the world using a physics and sensor engine.

The server can be started using the following command. Note that the server does not include any graphics; it's meant to run headless.

```
gzserver <world_filename>
```

The <world_filename> can be:

1. relative to the current directory,
2. an absolute path, or
3. relative to a path component in GAZEBO_RESOURCE_PATH.

Worlds that are shipped with Gazebo are located in <install_path>/share/gazebo-<version_number>/worlds.

For example, to use the empty.world which is shipped with Gazebo, use the following command

```
gzserver worlds/empty.world
```

Graphical Clients

The graphical client connects to a running gzserver and visualizes the elements. This is also a tool which allows you to modify the running simulation.

The graphical client is run using:

```
gzclient
```

The gazebo command combines server and client in one executable. Instead of running gzserver worlds/empty.world and then gzclient, you can do this:

```
gazebo worlds/empty.world
```

Plugins

Plugins provide a simple and convenient mechanism to interface with Gazebo. Plugins can either be loaded on the command line, or specified in a world/model file (see the SDF format). Plugins specified on the command line are loaded first, then plugins specified in the world/model files are loaded. Most plugins are loaded by the server; however, plugins can also be loaded by the graphical client to facilitate custom GUI generation.

Example of loading a plugin on the command line:

```
gzserver -s <plugin_filename> <world_file>
```

The same mechanism is used by the graphical client:

```
gzclient -g <plugin_filename>
```

4.2.2/ Gazebo's model directory structure

Gazebo can dynamically load models into simulation either programmatically or through the GUI. The model's directory structure is included: (example model_1)

Database.config

model_1:

- model.config: Meta-data about model_1
- model.sdf: SDF description of the model
- meshes: a directory for all COLLADA and STL files
- material: a directory which should only contain the textures and scripts directories
    - textures: A directory for image files (jpg, png, etc).
    - scripts: A directory for ORGE material scripts
- plugins: A directory for plugin source and header files

The format of Database.config (is only required for online repostitories):

```
<?xml version='1.0'?>

<database>

  <name>name_of_this_database</name>

  <license>Creative Commons Attribution 3.0 Unported</license>

  <models>

    <uri>file://model_directory</uri>

  </models>

</database>
```

The format of model.config (contains meta information about the model):

```
<?xml version="1.0"?>


<model>
  <name>My Model Name</name>
  <version>1.0</version>
  <sdf version='1.5'>model.sdf</sdf>


  <author>
    <name>My name</name>
    <email>name@email.address</email>
  </author>


  <description>
    A description of the model
  </description>
</model>
```

4.2.3/ Example of making a mobile robot (rectangular base with two wheels):

Setup model directory

1. Create a model directory:

```
mkdir -p ~/.gazebo/models/my_robot
```

2. Create a model config file:

```
gedit ~/.gazebo/models/my_robot/model.config
```

3. Paste in the following contents:

```
<?xml version="1.0"?>
<model>
  <name>My Robot</name>
  <version>1.0</version>
  <sdf version='1.4'>model.sdf</sdf>


  <author>
   <name>My Name</name>
   <email>me@my.email</email>
  </author>


  <description>
    My awesome robot.
  </description>
</model>
```

4. Create a ~/.gazebo/models/my_robot/model.sdf file.

```
gedit ~/.gazebo/models/my_robot/model.sdf
```

5. Paste in the following.

```
<?xml version='1.0'?>
<sdf version='1.4'>
  <model name="my_robot">
  </model>
</sdf>
```

At this point we have the basic contents for a model. The model.config file describes the robot with some extra meta data. The model.sdf file contains the necessary tags to instantiate a model named my_robot using Gazebo linked against SDF version 1.4.

Build model’s structure

Modify the ~/.gazebo/models/my_robot/model.sdf by following command:

```
gedit ~/.gazebo/models/my_robot/model.sdf
```

Add following code to make a car with two wheels:

```
<?xml version='1.0'?>
    <sdf version='1.4'>
```

```
      <model name="my_robot">
<static>false</static>
<link name='chassis'>
          <pose>0 0 .1 0 0 0</pose>
          <collision name='collision'>
            <geometry>
              <box>
                <size>.4 .2 .1</size>
              </box>
            </geometry>
          </collision>

          <visual name='visual'>
            <geometry>
              <box>
                <size>.4 .2 .1</size>
              </box>
            </geometry>
          </visual>

          <collision name='caster_collision'>
            <pose>-0.15 0 -0.05 0 0 0</pose>
            <geometry>
              <sphere>
              <radius>.05</radius>
            </sphere>
          </geometry>
          <surface>
            <friction>
              <ode>
                <mu>0</mu>
                <mu2>0</mu2>
                <slip1>1.0</slip1>
                <slip2>1.0</slip2>
              </ode>
            </friction>
          </surface>
        </collision>

        <visual name='caster_visual'>
          <pose>-0.15 0 -0.05 0 0 0</pose>
          <geometry>
            <sphere>
              <radius>.05</radius>
            </sphere>
          </geometry>
        </visual>
      </link>
      <link name="left_wheel">
        <pose>0.1 0.13 0.1 0 1.5707 1.5707</pose>
        <collision name="collision">
          <geometry>
            <cylinder>
              <radius>.1</radius>
              <length>.05</length>
            </cylinder>
          </geometry>
        </collision>
        <visual name="visual">
          <geometry>
            <cylinder>
              <radius>.1</radius>
              <length>.05</length>
            </cylinder>
          </geometry>
        </visual>
      </link>

      <link name="right_wheel">
        <pose>0.1 -0.13 0.1 0 1.5707 1.5707</pose>
        <collision name="collision">
          <geometry>
            <cylinder>
              <radius>.1</radius>
              <length>.05</length>
            </cylinder>
          </geometry>
        </collision>
        <visual name="visual">
          <geometry>
            <cylinder>
              <radius>.1</radius>
              <length>.05</length>
            </cylinder>
          </geometry>
        </visual>
      </link>
    <joint type="revolute" name="left_wheel_hinge">
        <pose>0 0 -0.03 0 0 0</pose>
        <child>left_wheel</child>
        <parent>chassis</parent>
        <axis>
          <xyz>0 1 0</xyz>
        </axis>
      </joint>

      <joint type="revolute" name="right_wheel_hinge">
        <pose>0 0 0.03 0 0 0</pose>
        <child>right_wheel</child>
        <parent>chassis</parent>
        <axis>
          <xyz>0 1 0</xyz>
        </axis>
      </joint>
    </model>
      </sdf>
```

Then start gazebo, we will get a basic mobile robot[18] (the robot in red circle):

[18] http://gazebosim.org/tutorials?tut=build_robot&cat=build_robot

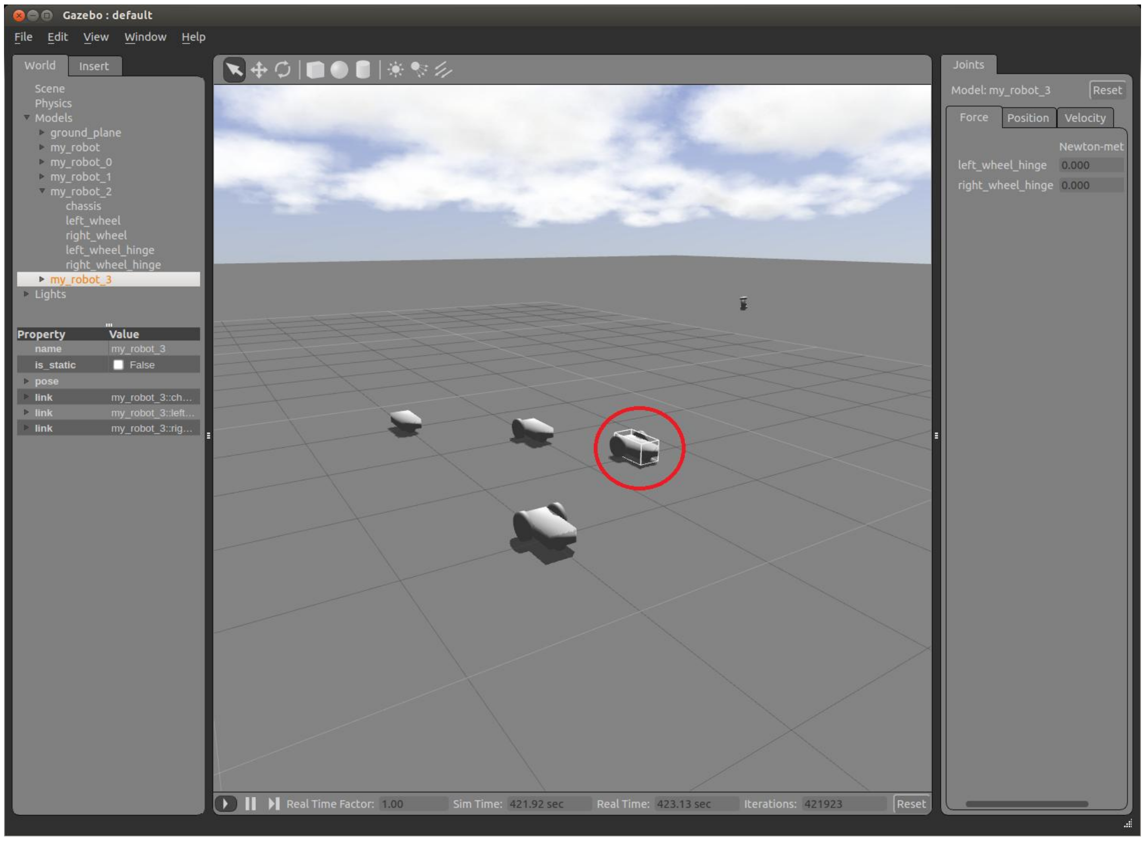


Fig 4.2.3.1 Mobile robot with rectangular base and two wheels

These basic steps could help me create first sample of robot as well as gaining knowledge of using Gazebo. However, connecting Gazebo to ROS demand a numbers of complex works which could consume much time.

4.3/ Using Gazebo integrating with ROS

4.3.1/ Install gazebo_ros_pkgs

4.3.2/ Testing Gazebo with ROS Integration

Be sure to always source the appropriate ROS setup file.

To rune Gazebo via ROS:

```
roscore &
rosrun gazebo_ros gazebo
```

The Gazebo GUI should appear with nothing inside the viewing window[19].

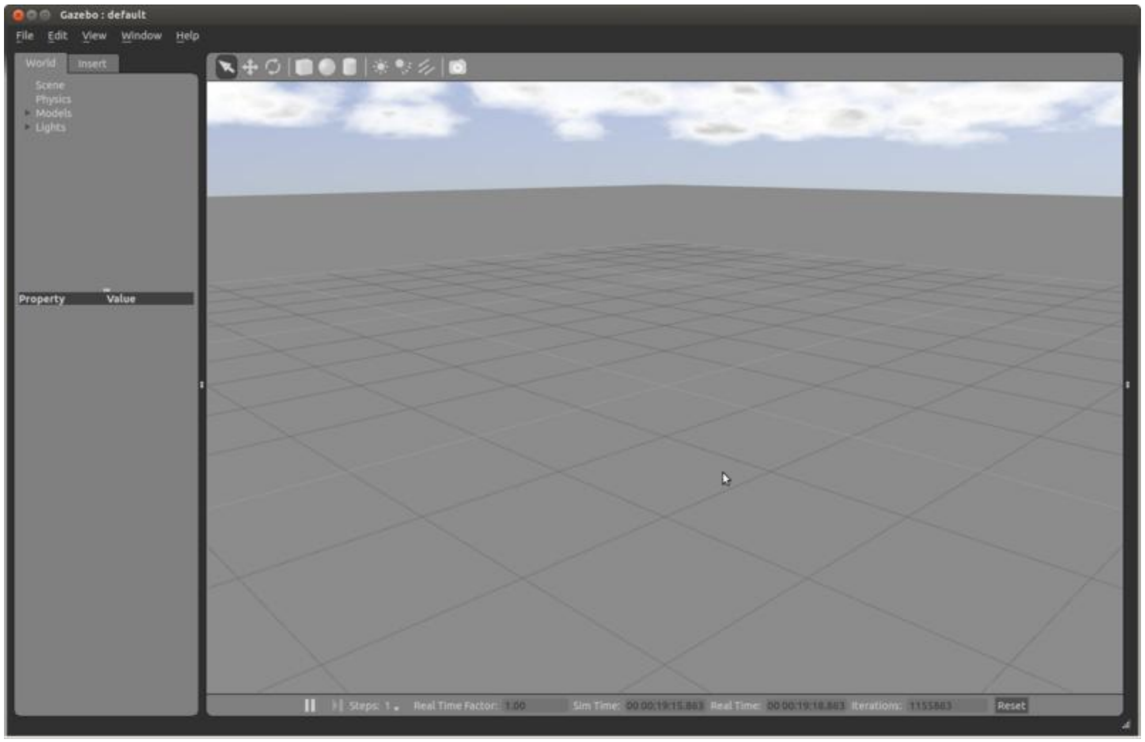


Fig 4.3.2.1 empty Gazebo

There are some other ROS ways to start Gazebo (each command either start Gazebo):

```
rosrun gazebo_ros gazebo
rosrun gazebo_ros gzserver
rosrun gazebo_ros gzclientrosrun gazebo_ros debug
roslaunch gazebo_ros empty_world.launch
```

Then we should using *roslaunch* to start Gazebo, world files and URDF models. Configuration for Gazebo into ROS also need Gazebo plugins, control, ROS communication.

We can now make a package for robotic arm with its contents then simulation it by Gazebo through ROS according to above required steps.

[19] http://gazebosim.org/tutorials?tut=ros_installing&cat=connect_ros

5/ Conclusion

With design and simulation of single arm, this project has reach initiative step for completing whole systems. In this time, I have earned some achievements which could be improved for thesis in near future:

- ✓ Solved primitive mathematical problems as well as physical ones about robotic arm.
- ✓ Building model of robotic arm using software.
- ✓ Learn basic ROS and Gazebo for simulation robot.
- ✓ Evaluating progress and requirements to manage time for improving this project.

And what are the rest to finish whole system:

- ✓ Building complete 3 arms with changeable tooltip
- ✓ Building controller for surgeons with high sensitive sensor
- ✓ Building CPU for system to help it analyze as well as automatic working
- ✓ Integrate cameras system broadcasting from surgical area to surgeons.
- ✓ Building communication system for arm-system and controller
- ✓ Building software to simplify controlling the system

There would be many aspects I need to learn to build complete this systems:

- ✓ Image processing
- ✓ Force sensor
- ✓ Force feedback control
- ✓ Accurate controlling

If these works could be finished or upgraded, we could build this systems inland then reducing cost. Surgical robot would be more practical and bring conveniences as well as enhancing medicine.

6/ Appendix

Books:

- *Robotics* –Prof. Peter Nauth
- *KUKA youBot User Manual*- Locomotec
- *Robotics Modeling Planning and Control* –Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriol
- *Basic Surgical Techniques*-György Wéber MD, PhD, med. habil-János Lantos MSc, PhD- Balázs Borsiczky MD, PhD Andrea Ferencz MD, PhD-Gábor Jancsó MD, PhD-Sándor Ferencz MD-Szabolcs Horváth MD-Hossein Haddadzadeh Bahri MD-Ildikó Takács MD-Borbála Balatonyi MD.

Articles/Slides:

- *Direct Kinematics, Euler and roll-pitch-yaw, Position and Orientation of Rapid Bodies* – Mr. Le Đinh Than.
- *Pattern Recognition – PhD. Nguyen Minh Hien*
- *Planning and Real Time Control of a Minimally Invasive Robotic Surgery System*- Andreas Tobergte, Rainer Konietschke, and Gerd Hirzinger
- *DLR MiroSurge: a versatile system for research in endoscopic telesurgery*- Ulrich Hagn, R. Konietschke, A.Tobergte, M.Nickl, S.Joerg, B,Kuebler, G.Passig, M. Groeger, F. Froehlich, L. Le Tien, A.Albu-Schaeffer, A. Nothhelfer, F. Hacker, M.Grebenstein, G.Hirzinger.
- *Telemanipulator for Remote Minimally Invasive Surgery*- Ulrich Hagn, R. Konietschke, A.Tobergte, M.Nickl, S.Joerg, B,Kuebler, M. Groeger, T. Ortmaier, U.Seibold and G.Hirzinger.
- http://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters
- http://www.davincisurgery.com/da-vinci-surgery/da-vinci-surgical-system/
- http://www.intuitivesurgical.com/
- http://www.dezeen.com/2013/11/18/robot-surgeons-to-operate-on-beating-human-hearts/

- http://gazebosim.org/tutorials
- http://wiki.ros.org/
- https://forum.solidworks.com/?scid=community_forums