Vicon Motion Capture System for ROS Enabled Robots

Louis Lefevre

October 11, 2018

Contents

Chapter 1

Operating System

1.1 Installation of Ubuntu 14.04

First create a bootable usb stick:
https://tutorials.ubuntu.com/tutorial/tutorial-create-a-usb-stick-on-windows#0

When you have a bootable usb stick, you are ready to install Ubuntu 14.04:
https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop#0

1.2 Installation of ROS Indigo

All information needed to install ROS Indigo can be found here:
http://wiki.ros.org/indigo/Installation/Ubuntu

Before installing ROS Indigo, first setup your computer to accept software from packages.ros.org by writing the following lines in the command box.

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_rel
ease -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --re
cv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

Then make sure your Debian package index is up-to-date:

```
$ sudo apt-get update
$ sudo apt-get install ros-indigo-desktop-full
```

Install the ROS packages by the following commands.

```
$ # rosdep
$ sudo rosdep init
$ rosdep update
$ # rosinstall
$ sudo apt-get install python-rosinstall
```

Set up the workspace, a folder named *catkin_ws* in the */home* folder where all package files are placed.

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
$ #Now we need to set the path to the workspace
$ #This line is adding the cmd line at the end of the file .bashrc
$ #So we do not have to write the line each time we start a new terminal
$ echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

1.3 Intallation of Ubuntu 16.04

All information needed to install Ubuntu 16.04 can be found here:
https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop#0

1.4 Installation of ROS Kinetic

All information needed to install ROS Kinetic can be found here:
http://wiki.ros.org/kinetic/Installation/Ubuntu

Chapter 2

Vicon

Vicon is a developer of motion capture products and services for the life science, entertainment, and engineering industries. Motion capture (mocap) is the process of recording the movement of objects or people. The technology originated in the life science market for gait analysis but is now used widely by VFX studios, sports therapists, neuroscientists, and for validation and control of computer vision and robotics.

More information about Vicon can be found on https://www.vicon.com.

2.1 Applications

The **biomechanics and sports** science community encompasses many applications, including research, sports performance, and animal science. A fully flexible motion measurement system capable of capturing motion in all environments with minimal set-up is required to analyze an athlete's motion for sport performance enhancement or injury prevention and to understand animal biomechanics, analyze animal body behavior, and record positions, from horses to birds, fishes to dogs, rodents to insects.

The **clinical science** community focuses on analyzing body motion of people in rehabilitation and getting information for posture, balance, and motor control. Vicon systems can be used to measure or give real-time feedback on the movements of the whole body or a single part, including detailed hands, face, feet and spine across different applications. For example, stroke rehabilitation, posture analysis, balance studies, and reaching studies.

The **entertainment** community uses Vicon systems to capture actors' movements to use it in movies or video games.

The **object tracking** community uses motion tracking systems to track fast-moving UAVs, to evaluate product designs for manufacturing projects, industrial robots, localization of tools, or to analyze the movement of life-size buildings within the world's largest earthquake simulator. For these applications a highly accurate system that provides low latency data and that is easy to use is required.

The **virtual reality** community focuses on virtual reality tracking and ergonomic studies to improve interaction with the virtual world. Mocap systems can be used to undertake the 3D assessment of human interaction with products, work space, and living spaces in a wide variety of ergonomic and human factor applications. Applications in immersive reality needs accurate and true-to-life manipulation of graphics for animation or simulation. Vicon's low latency tracking eliminates the lag, inaccuracies, unpredictability and unresponsiveness of traditional magnetic or inertial technologies.

The **broadcast** community inserts visual content in live production.

More detailed information, i.e. specifications, about the applications can be found on https://www.vicon.com/motion-capture.

2.2 Working Principle

The Vicon motion capture system is a passive optical system. This technique uses retroreflective markers that are tracked by the infrared cameras by reflecting the light that is generated near the cameras' lens. The camera's threshold can be adjusted so only the bright reflective markers will be sampled, ignoring skin and fabric.

The set of infrared cameras are placed around the to-be-captured area and are more or less pointing to the center of this area as depicted in Fig. 2.1. The goal is that the markers in the captured volume can be seen by as many cameras as possible.
We need at least two cameras to know the position of one marker, but it is recommended to use as many cameras as possible, because the more cameras can capture a marker, the higher the accuracy of that marker's position and the lower the possibility the position of that marker is lost.
If we only use markers on the floor (i.e. in a 2D plane) without people or objects around so that the markers can never be occluded, we can use only 3 cameras and have good results. In more difficult situations with markers evolving in 3D, people or objects hiding one of the cameras, the number of cameras needed can increase quickly.

In the R&MM lab we are using 6 cameras. This is ok for the experiments we have executed. For experiments where full coverage around the to-be-captured subject is required, more cameras will be needed.

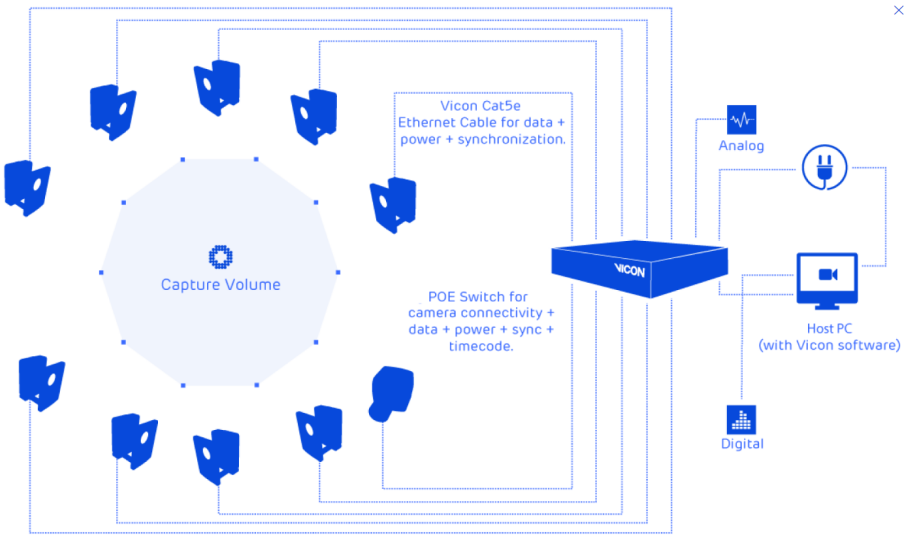


Fig. 2.1: Vicon Working Principle

More information can be found on https://www.vicon.com/what-is-motion-capture.

2.3 Hardware

2.3.1 Cameras

At the R&MM lab there are six Vero v2.2 cameras with a resolution of 1280*1024 pixels and a frame rate of 250fps. The higher the resolution of the camera, the preciser the positioning of the markers and the more details you can obtain from the markers which is interesting when using very small markers placed close to each other. The higher the frame rate, the easier to track the quick movements of the markers, which is also needed to use position of markers in a feedback loop with high sampling time.

The placement of the cameras really depends on the experiment we want to carry out. In the lab the cameras can be placed onto the walls and on a tripods. In the future it could be useful to have all of them on tripod to be able to change their places easily and quickly.

Fig. 2.2: Vicon cameras

2.3.2 Sync Box

As can be seen in Fig. 2.1, all cameras and the PC are connected to the synch box in order to get data coming from cameras and sending this to the PC. The sync box provides a single communication point between the cameras and the (Vicon) PC.
If you're not integrating third-party equipment like reference video or force plates into your system, this could be as simple as a POE switch. The real difference with the various sync boxes offered by manufacturers comes when you need to integrate fully synchronized third-party devices or require time-code. Vicon's Lock+ sync box offer many features such as Timecode, Genlock, VESA and synchronized triggering of your third-party devices.

More information can be found on https://www.vicon.com/products/vicon-devices/lock-sync-box.

Fig. 2.3: Cameras connected to the synch box

2.3.3 Active Wand

Before using Vicon we need to calibrate the Vicon system in order that the cameras know their location relative to each other.

Before starting with the calibration, we first have to mask all reflective elements (e.g. the cameras themselves, maybe clothes or shoes with that reflect infrared light) in the Vicon room, so no reflective markers that you will use may be visible for the cameras.
To start the calibration, press start in the "Calibrate Camera" menu. Thereafter, start waving the wand in the room. The system will acquire many frames and after having captured more than 2000 frames for each camera the calibration is finished.
There are two operating modes for different light condition. This means you can calibrate your motion capture cameras whether you are inside or outside in the brilliant sunshine.
To optimize the process, the active wand automatically synchronizes itself by the use of a photo-diode.

The precision and repeatability of your data is dependent on your system calibration. The LEDs on Vicon's Active Wand are machined placed and precision engineered to give you a more precise overall calibration. The Active Wand calibrates both optical and video cameras while accounting for sensor edge distortion at the same time, to ensure synchronized overlay across your entire volume.
Notice that better results will be obtained when the active wand is visible to as many cameras as possible during the waving process. The more cameras see the waving movement of the active wand

at a certain time, the better they can compute the position relative to each other.

More information can be found on https://www.vicon.com/products/vicon-devices/calibration.

Fig. 2.4: Active Wand

2.3.4 Reflective Markers

The reflective markers are placed on objects or on the body in order to detect their position.
In order to detect pose (i.e position and orientation) of a rigid object we need at least 3 markers. The position of the markers w.r.t. each other needs to be asymmetrical, in other words you may not form an equilateral triangle.

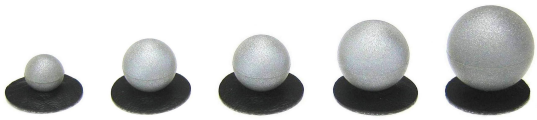


Fig. 2.5: Markers

2.3.5 Computer

The computer gives us the possibility to drive the Vicon system, calibrate, organize data capture, set parameters. Since the capture and display of motion capture data is a highly demanding task for a PC and the real-time data processing asks a lot of the processor, the recommended specifications of the computer are:

- Dell Precision XL Tower 5820
- Intel Xeon Processor W-2123 (3.6GHz, 3.9GHz Turbo, 4C, 8.25M Cache))
- 16GB (2x8GB) 2666MHz DDR4 RDIMM ECC
- 2 x 2TB 3.5inch Serial ATA (7,200 Rpm) Hard Drive
- Non RAID
- 8x Slimline DVD+/-RW Drive
- 2 GB NVIDIA Quadro P600
- Windows 10 Pro English

- INTEL ETHERNET i350 T4 SERVER ADAPTER RJ45 PCI−E – to connect Vicon System

More information can be found on https://www.vicon.com/faqs/operating-systems-and-pc-specification/what-is-the-recommended-pc-specification-to-run-my-vicon-tracker-system.

2.4 Software

Vicon offers several software packages, each of them for for different applications: https://www.vicon.com/products/software. Two of them are explained below.

2.4.1 Nexus

https://www.vicon.com/products/software/nexus
Nexus is the software that is already available in the lab. Other researchers are using Nexus in order to capture body motion. We will explain later how to use it. We also used it for our experiments.

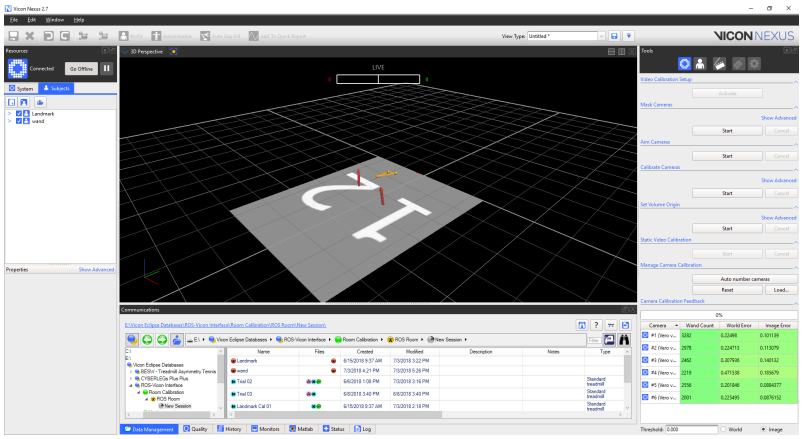


Fig. 2.6: Nexus

2.4.2 Tracker

https://www.vicon.com/products/software/tracker
Tracker is designed for engineering applications. It can be used for robot tracking, human factors engineering, design method optimization, virtual engineering, and previsualization to virtual reality.
It looks better to use for the experiments we execute, but we were also able to use Nexus.

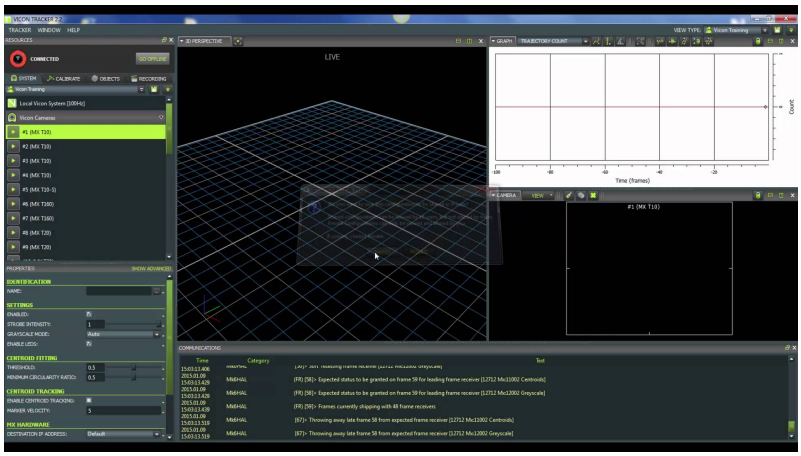


Fig. 2.7: Tracker

2.5 Calibration of the Vicon System

The mask camera function is used to make camera don't care about area where infrared signal is not significant. Because vicon can believed there are markers in this area but it is not the case. So when we use this function we have to be sure no markers are in the vision field of cameras.
In order to calibrate the cameras, press "start" in the calibration menu and wave with the active wand in the to-be-captures area, i.e. where you want to detect reflection markers. Cameras are blinking during this process and are all turned green when the process is finished.

In order to set the origin of the Vicon system, click "start" in the camera origin menu. We usually use the active wand, but there are also others methods in advanced options. Leave the active wand where you want (make sure it is in the camera view of all cameras) and press "set the origin". Now your vicon system is ready to use. To calibrate the system we can also adjust the focal of cameras.
During using of the Vicon sometimes cameras blink red, it means the camera pose has changed. Most of the time it is only due to vibration so it do not affect the capture.

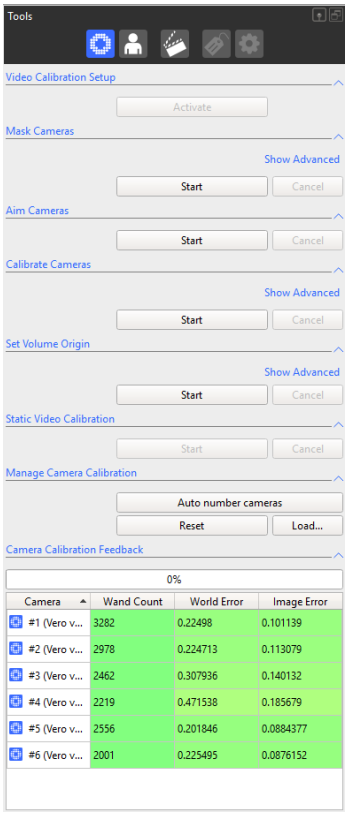


Fig. 2.8: Nexus

2.6 Vicon bridge

In order to use the Vicon system with ROS enabled robots, the Vicon bridge has to be installed. http://wiki.ros.org/vicon_bridge

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/ethz-asl/vicon_bridge.git
$ cd ..
$ catkin_make
```

We need to set parameters in the vicon.launch file

```
//change this line
<param name="datastream_hostport" value="vicon:801" type="str" />
//into
```

```
4 <param name="datastream_hostport" value="PC-IP-adress:801" type="str" />
```

The installation of the Vicon bridge has been tested on Ubuntu 14.04 and Ubuntu 16.04. It works well on both operating systems.

In order to get data from the Vicon system, we need to connect the Vicon computer and our computer on the same network, WiFi or Ethernet. The easiest way is to connect them via a Wifi network. Therefore we have to set the datastream_hostpot value to the IP of the Vicon computer.

On a Ethernet network in addition to that we need to parametrize the network, in the network manager you can create a new Ethernet connection by setting manualy in the IPv4 parameter an address (i.e address 169.254.123.105 mask 255.255.0.0 bridge 0.0.0.0)

For each robot I used with Vicon I did something different to connect Vicon to ROS.

With KUKA youBot : My computer was connected to the robot on wifi network. This network was generate by AdHoc network from my computer (ubuntu) So I can control the robot with my computer over the network. And my computer was connected with Ethernet cable to the Vicon computer. I had to manually parametrize the network.
With Baxter : Baxter, The baxter computer and The Vicon computer were all connected to a switch.
With Franka : The computer is connected to franka using an Ethernet cable and the computer and Vicon computer are connected using wifi network. The WiFi is generate by and AdHoc network from the Vicon computer (Windows 10).

Chapter 3

KUKA youBot

3.1 Hardware

The youBot consists of a mobile base and a 5DoF manipulator as depicted in Fig. 3.1.

Fig. 3.1: KUKA youBot

The mobile base has a length of $580\,\text{mm}$, a width of $380\,\text{mm}$, a height of $140\,\text{mm}$, a weight of $20\,\text{kg}$, a payload of $20\,\text{kg}$, and a maximum velocity of $0.8\,\text{m/s}$. It moves on four Mecanum wheels, which have rolls mounted around the circumference at a $45°$ angle to the wheel's plane and enable the youBot to combine any translational and rotational movements at any given time. This makes omnidirectional motion possible, including sideways and diagonal motion.

The 5DoF manipulator is mounted on the omni-directional platform. Its height and joint angle limitations are visualized in Fig. 3.2. A two-finger gripper at the end of the arm allows the robot to move objects of up to $70\,\text{mm}$ in length and with a weight of up to $500\,\text{g}$. The robotic arm consists of five joints, each driven by maxon drive systems. All joints have relative encoders to measure the joint angles.
Each time a relative encoder is powered on, it begins counting from zero, regardless of where the shaft is. Therefore, initial homing to a reference point is inevitable in all positioning tasks, both upon start up of the control system and whenever power to the encoder has been interrupted.

There is a real-time EtherCAT communication and an on-boad PC with embedded CPU, $2\,\text{GB}$ RAM, $32\,\text{GB}$ SSD Flash, and USB port. The computer specifications are listed in Table 3.1.

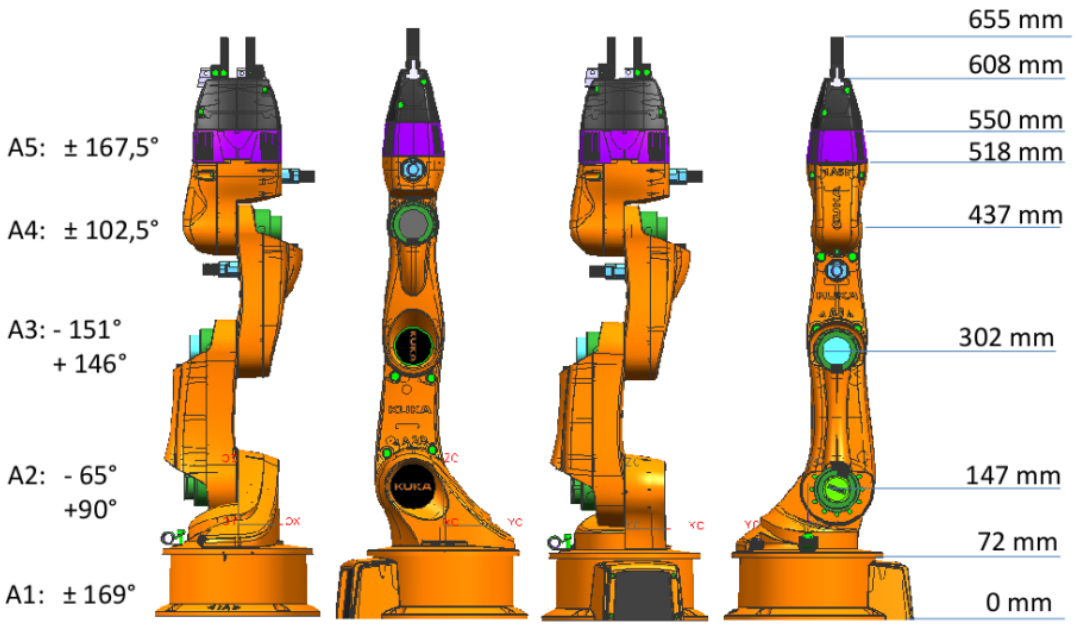


Fig. 3.2: KUKA youBot arm specs

Kinematics Processory	Intel AtomTM Dual Core D510 (1M Cache, 2 x 1.66 GHz)
Memory	2GB single-channel DDR2 667MHz
Graphics	Embedded Gen3.5+ GFX Core, 400-MHz render clock frequency, up to 22 MB shared memory
Harddrive	32GB SSD drive

Table 3.1: KUKA youBot on-board computer specs

3.2 Operating System

The original operating system installed on the KUKA youBot was Ubuntu 12.04. Since we had some communication problems when using this operating systems and since it was recommended to use Ubuntu 14.04 with the Vicon bridge, we installed Ubuntu 14.04 on the KUKA youBot.

In order to keep the 12.04 version, we installed the Ubuntu 14.04 version on the youBot's embedded computer via a dual-boot system. In this way, we are able to use one of the operating systems by selecting it when booting the computer.

To make a dual-boot possible, we first had to make a partition on the youBot's hard disk. We followed the steps of https://www.howtogeek.com/114503/how-to-resize-your-ubuntu-partitions/.

- Plug a screen, a mouse, a keyboard, and the installation USB stick into the youBot.
- During the boot a black screen with white text will appear. Press many times F11, whereafter you can select your USB stick to boot on it.
- Choose "try Ubuntu".
- Run Gparted.
- Right click on the main partition of the hard drive. Select "Resize/Move" (If the partition is mounted right click "unmount")

- Then we can choose the size of the partition, I decided to divide the size of the partition by two.
- Click "Apply all Operation" on the top, this operation can take a while.
- Right click on the unallocated partition, select "new" and create a ext4 partition with mount point "/".
- Click "Apply all Operation" on the top.
- Then on the Desktop, click on the installation program.
- Choose "install" and "something else".
- Select the last partition you created an click "install now".
- Follow instructions and restart the computer after the installation.

3.3 Driver

The youBot Wrapper installs the driver for the youBot and all its dependencies.

```
1 $ sudo apt-get install ros-indigo-youbot-driver-ros-interface ros-ind
2 igo-youbot-description
3 $ sudo setcap cap_net_raw+ep /opt/ros/indigo/lib/youbot_driver_ros_in
4 terface/youbot_driver_ros_interface
5 $ sudo ldconfig /opt/ros/indigo/lib
```

3.4 Wireless Access

In order to wireless access the robot, we need to use a WiFi network. Since we are never sure if an external network will be working, a more reliable solution is to create an AdHoc network from your own computer. By doing this, you allow the devices connected to that network to communicate with each other.

3.4.1 Create an AdHoc hotspot

In Ubuntu 14.04, we can create a hotspot with our computer by searching *Network* on the dashboard. In the Network window, depicted in Fig. 3.3, click on "Use as Hotspot" to run an AdHoc hotspot.

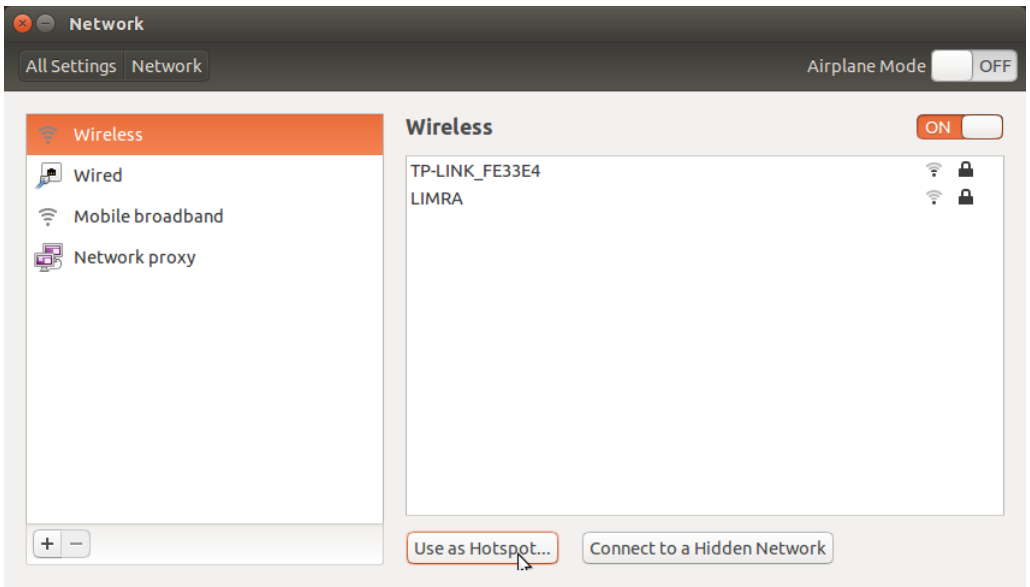


Fig. 3.3: Create a hotspot

The parameters of the youBot (AdHoc) network:

- name : youbot-desktop
- security : wep
- key : 97201a7b2b

So with this network my computer and the youBot can be connected.

3.4.2 Installation of SSH

SSH stands for Secure Shell. It allows someone to access a computer by using a specified network. This system is a secure access if we use an encrypted communication. To install the ssh-server on the youBot, open a terminal on the youBot and type the following command

```
$ sudo apt-get install openssh-server
```

This command only installs the program. The server is automatically started at each boot.

3.4.3 Parametrize the Kuka youBot

We want to be able to only push the start button of the robot and never touch something again. To do this,

- disable the screen shutdown,
- set the parameter to "automatically connect to this network" and disable it for other networks to be sure the robot will connect automatically to the youbot-network.

When someone else needs to connect with the youBot with his/her own created AdHoc network, he/she only needs to change parameters on Kukayoubot by pluging a keyboard, mouse, and screen.

3.4.4 Using SSH

Connect your computer to the youbot network and open a terminal on your computer:

```
$ # ssh login@IP
$ ssh youbot@10.42.0.1
```

This is the IP address of the youBot and the password of the youBot is *youbot*.

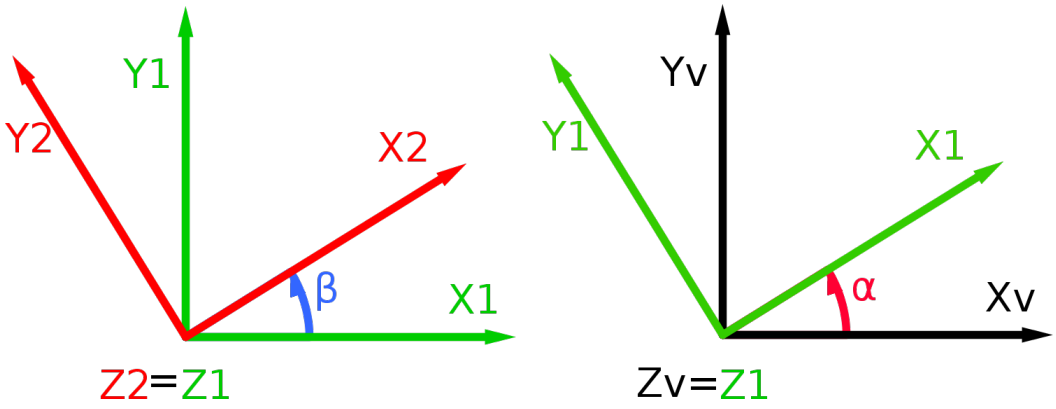
3.5 Mobile Base

In this section, the position and orientation of the youBot's mobile base will be controlled i) by odometry and ii) by using the Vicon system. Therefore, we created different frames that are depicted in Fig. 3.4.

In Fig. 3.4a, the ground frame of the robot is depicted in green and the mobile base frame is visualized in red, i.e. fixed to the mobile base of the youBot. In the starting position of the youBot, the red frame has the same orientation as the green frame, i.e. $\beta = 0$.

In Fig. 3.4b, the Vicon frame is depicted in black, i.e. the frame created when setting the origin of the Vicon system, and the ground frame is vizualised in green. In our case, when controlling the robot by using the Vicon system, $\alpha = \pi/2\,\text{rad}$.

The angle between the Vicon frame and the robot frame is $\rho_{\alpha+\beta}$, as can be seen in Fig. 3.5.



(a) robot frame (red) - ground frame (green) **(b)** ground frame (green) - Vicon frame (black)

Fig. 3.4: Orientation of the robot frame, ground frame, and Vicon frame w.r.t. each other.

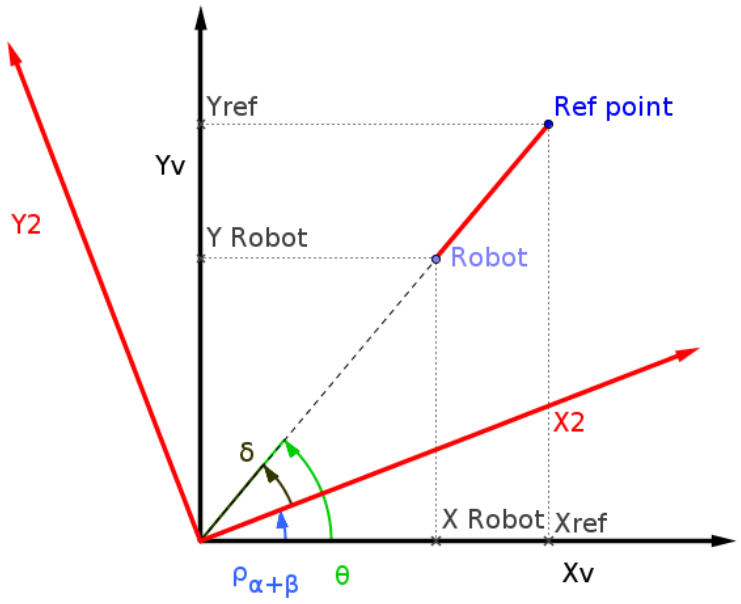


Fig. 3.5: Reference point

3.5.1 Pose Control via Odometry

The position and the orientation of the youBot's mobile base are both controlled by a proportional controller.

In the code below is shown **how the orientation is controlled**: *input* is the current orientation (i.e. β_{current}) and *ref* is the one we want to reach (i.e. β_{ref}).

```
1 class Prop{
2         private:
3         float P, threshold, precision ;
4
5         public:
6         void init(float mP, float mthreshold, float mprecision){
7                 P = mP ;
8                 threshold = mthreshold ;
9                 precision = mprecision ;
```

```
10          }
11
12          //compute of the command
13          float execute(float input, float ref){
14                  //command
15                  float cmd = P*(ref-input) ;
16                  //threshold
17                  if (cmd > threshold){
18                          cmd = threshold ;
19                  }
20                  else if (cmd < -threshold){
21                          cmd = -threshold ;
22                  }
23                  //precision
24                  if (fabs(ref-input)<precision){
25                          cmd = 0 ;
26                  }
27                  return (cmd) ;
28          }
29  };
```

In the code below is shown **how the position is controlled**: {*x_robot*,*y_robot*} is the current position of the youBot's center and {*x_ref*,*y_ref*} is the one the robot needs to reach.

```
1   class Prop2D{
2
3           private:
4           //proportionnal coefficient
5           float P ;
6           //absolute maximun lineare speed of the robot
7           float threshold ;
8           //when the the error is inferior to the precision
9           //the control is stoped
10          float precision ;
11
12          public:
13          //initialisation of attributs
14          void init(float mP, float mthreshold, float mprecision){
15                  P = mP ;
16                  threshold = mthreshold ;
17                  precision = mprecision ;
18          }
19
20          void execute(float x_robot, float y_robot, float alpha_beta,
21                  float x_ref, float y_ref, float* V_x, float* V_y){
22
23                  //distance error
24                  float x_error = x_ref-x_robot ;
25                  float y_error = y_ref-y_robot ;
26                  float distance_error =
27                          sqrt(x_error*x_error+y_error*y_error) ;
28                  //delta angle
29                  float theta ;
30                  theta = atan2(y_error, x_error) ;
31                  if (theta<0) theta+=2*pi ;
32                  float delta = theta-alpha_beta ;
33
```

```
                    //cacul of the command
                    float cmd = P*distance_error ;

                    //set the threshold
                    if (cmd > threshold){
                            cmd = threshold ;
                    }
                    else if (cmd < -threshold){
                            cmd = -threshold ;
                    }
                    //set the precision
                    if (fabs(distance_error)<precision){
                            cmd = 0 ;
                    }
                    //returned values
                    *V_x = cmd*cos(delta) ;
                    *V_y = cmd*sin(delta) ;
            }
};
```

alpha_beta is $\rho_{\alpha+\beta}$ on Fig.3.5 theta is θ on Fig.3.5 delta is δ on Fig.3.5 This code compute the command proportional to the distance from the point we want to reach and then project this value on the right frame(X2,Y2).

3.5.2 Drive the Mobile Base

To drive the youBot's mobile base we need to send data to the topic **cmd_vel** and we need to receive data from the topic **/odom**. To send data we need to make a **publisher** and to receive data we need to make a **subscriber**.

The data structure of the **cmd_vel** topic looks like this:

geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
        float64 x
        float64 y
        float64 z
    geometry_msgs/Vector3 angular
        float64 x
        float64 y
        float64 z

Where twist.linear is the linear speed and twist.angular is the angular speed of the KUKA youBot in the $\{X_2, Y_2, Z_2\}$ frame.

To publish data to the cmd_vel topic, we use this **publisher**:

```
//declaration of the publisher
//n is the pointer to the node
//<geometry_msgs::Twist> is the type of the topic
//"cmd vel" is the name of the topic
//1 is the size of the buffer

//init ROS node
ros::init(argc,argv, "youbot_position_angle");
//pointer to the node
ros::NodeHandle n ;
ros::Publisher pub = n.advertise<geometry_msgs::Twist>("cmd_vel", 1);

//publication
```

```
14
15 //variable with the same type than the topic we want publish
16 #include "geometry_msgs/Twist.h"
17 geometry_msgs::Twist twist;
18 //give a value
19 twist.linear.x = 1.57;
20 //publish
21 pub.publish(twist);
```

The data structure of the **/odom** topic looks like this:

```
std_msgs/Headerheader
        ...
string child_frame_id
geometry_msgs/PoseWithCovariance pose
        geometry_msgs/Pose pose
                geometry_msgs/Point position
                        float64 x
                        float64 y
                        float64 z
                geometry_msgs/Quaternion orientation
                        float64 x
                        float64 y
                        float64 z
                        float64 w
                float64[36] covariance
geometry_msgs/TwistWithCovariance twist
        ...
```

Which gives us information about the position and orientation of the robot's mobile base.

To get data from the robot's motion sensors, we first have to initialize/declare the subscriber.

```
1  //declaration of the subscriber
2  //n is the pointer to the node
3  //"odom" is the topic we subscribe
4  //init the node
5  ros::init(argc,argv, "youbot_odom_subscriber");
6  //pointer to the node
7  ros::NodeHandle n ;
8  //OdomCallback is the function called every time something is publish
9  //to the topic
10 ros::Subscriber sub= n.subscribe("odom",10,OdomCallback);
```

To receive data from the /odom topic, we use this **subscriber**:

```
1  //includes
2  #include <tf/transform_broadcaster.h>
3  #include <angles/angles.h>
4
5  void OdomCallback(const nav_msgs::Odometry::ConstPtr& msg){
6          //convertion of the quaternion
7          //euler angles
8          double roll, pitch , yaw ;
9          tf::Quaternion quater;
10         tf::quaternionMsgToTF(msg->pose.pose.orientation , quater) ;
11         tf::Matrix3x3(quater).getRPY(roll , pitch ,yaw) ;
12         yaw = angles::normalize_angle_positive(yaw) ;
13
14         //calibration :
15         if (!calibration){
```

```
16                  if (yaw>pi) counter=-1 ;
17                  previous=yaw;
18                  calibration = true ;
19          }
20          else{
21                  //anti trigo
22                  if ((yaw-previous)>pi){
23                          counter-- ;
24                  }
25                  //trigo
26                  else if ((yaw-previous)<-pi){
27                          counter++ ;
28                  }
29                  previous = yaw ;
30          }
31  }
```

Because the orientation of the robot is given in quaternions, we need to convert them to Euler angles. This is done in lines 10 and 11.

3.5.3 Pose Control via Vicon

Drive the youBot

For controlling and driving the youBot's mobile base with the Vicon system, we have to receive data about the robot's position and orientation from another topic, the **/vicon/marker** topic.

The data structure of the **/vicon/marker** topic looks like this:

```
std_msgs/Header header
        uint32 seq
        time stamp
        string frame_id
uint32 frame_number
vicon_bridgeMarker[] markers
        string marker_name
        string subject_name
        string segment_name
        geometry_msgs/Point translation
                float64 x
                float64 y
                float64 z
        bool occluded
```

The new **subscriber** becomes:

```
1  sub = n.subscribe("vicon/markers",10,MarkersCallback);
2  //--------------------
3  void MarkersCallback(const vicon_bridge::Markers::ConstPtr& msg){
4          float x = msg->markers[0].translation.x ;
5          float y = msg->markers[0].translation.y ;
6  }
```

Setup the Vicon system

- Turn on the VICON and the computer (Pwd VCN@r&mm)
- Launch Nexus
- Click on the upper left button to manage Eclipse Database (see Fig. 3.6) and create a new project.

- Then create a tree as you can see in Fig. 3.7.

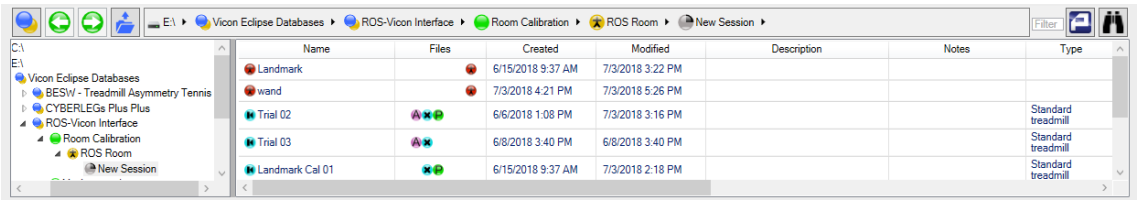


Fig. 3.6: Database in Nexus

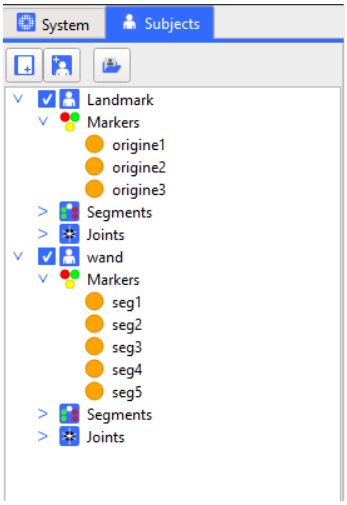


Fig. 3.7: Tree in Nexus

In order to organize all data coming from the Vicon system, we will give a name to all markers. We can create a new subject with the button on the upper left corner as in Fig.3.7.
To capture the subject make sure all markers of the rigid body are visible by Nexus. Capture the body a few seconds, see Fig. 3.8.

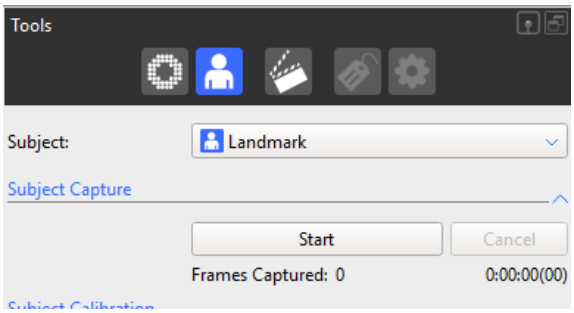


Fig. 3.8: Subject Capture in Nexus

Then click on reconstruct button, the icon with the gray bubbles in Fig. 3.9. Normally, all markers should appear in the 3D scenery.

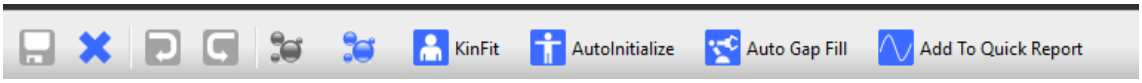


Fig. 3.9: Reconstruct in Nexus

To create a new segment, click on create segment and select the markers of each group in the order depicted in Fig. 3.10.

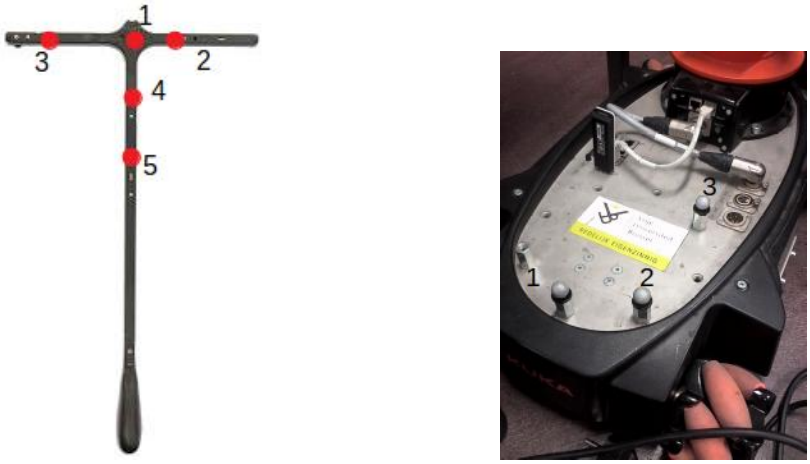


Fig. 3.10: Segments in Nexus: Active Wand + Mobile Base

When you click on *go online*, both subjects will appear on the scenery and everything is ready to be received by the vicon bridge
All markers are given in a table. If markers that belong to a subject are not recognized, all their coordinates are set to zero. If markers that are not in a subject are recognized they are added at the end of the table. At the beginning I believed that subject was always given in the same order but the order is the detection order. To be sure the order is always the same we need to activate subjects in the same order.
The position of the Vicon frame is shown in Fig.3.11.

Fig. 3.11: Vicon wand frame

Pose Control

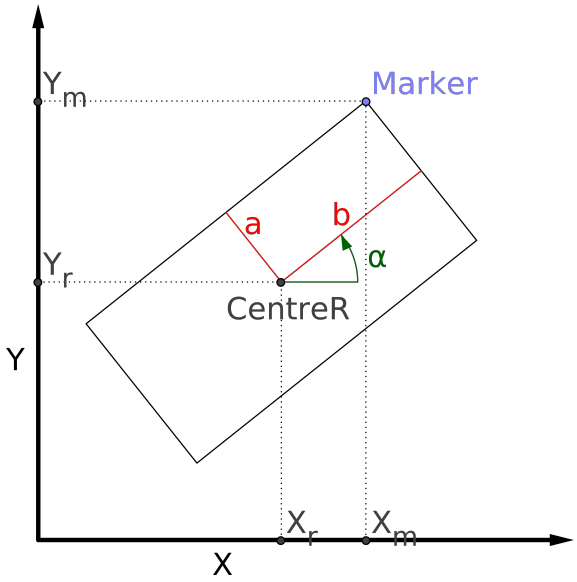


Fig. 3.12: Pose of youBot's mobile base in Vicon frame.

By knowing the offsets in width and length direction of the corner marker to the center of the youBot, i.e. a and b respectively as can be seen in Fig. 3.12, we can compute the diagonal length between the center and the corner marker. Having this distance, $c = \sqrt{a^2 + b^2}$, we have to project this distance to the $\{X_v, Y_v\}$ frame. These lengths have to be subtract to the coordinate of the marker we measure (on the corner).

```
1  //get the angle
2          float alpha ;
3          float dx = -(msg->markers[origine2].translation.x-msg->mark
4                                  ers[origine3].translation.x) ;
5          float dy = msg->markers[origine2].translation.y-msg->mark
6                                  ers[origine3].translation.y ;
7          alpha = atan2(dx,dy) ;
8          if (alpha<0) alpha+=2*pi ;
9
10         //get the position
11         float a=61 ;
12         float b=211 ;
13         float c = sqrt(a*a+b*b) ;
14         float phi = atan(a/b);
15         float offset_y = c*cos(alpha+phi);
16         float offset_x = c*sin(alpha+phi);
17         float pos_x = msg->markers[origine2].translation.x+offset_x ;
18         float pos_y = msg->markers[origine2].translation.y-offset_y ;
19         float pos_z = msg->markers[origine2].translation.z;
```

3.5.4 Pose Control via Teleoperation with Active Wand

Until now we were giving the pose of the robot using parameters given by the function launched on our computer, so gave as input (with our keyboard) one pose at each time.
We will now control the motion of the robot without a keyboard, but by using the active wand of the Vicon system. By this, we will teleoperate the robot.

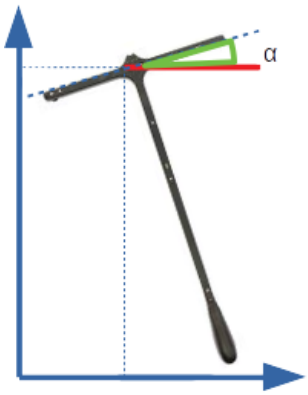


Fig. 3.13: Teleoperate the Mobile Base with the Active Wand Is this angle α the same as you used before?No it is alpha_wand

The code below shows how we controlled the pose of the mobile base with the active wand. Some conditions are added to ensure the robot cannot go off the treadmill.

```
1  void MarkersCallback(const vicon_bridge::Markers::ConstPtr& msg){
2          //get the wand position:
3          float wand_x = msg->markers[seg1].translation.x ;
4          float wand_y = msg->markers[seg1].translation.y ;
5
6          //get the wand angle
7          float alpha_w ;
8          float dx_w = -(msg->markers[seg4].translation.x-msg->mark
9                                  ers[seg1].translation.x) ;
10         float dy_w = msg->markers[seg4].translation.y-msg->mark
11                                  ers[seg1].translation.y ;
12         alpha_w = atan2(dx_w,dy_w) ;
13         if (alpha_w<0) alpha_w+=2*pi ;
```

3.6 Robotic Manipualtor

To control the robotic manipulator, an extra frame has to be added. In Fig. 3.14 we can see that the frame fixed to the robot's mobile base $\{X_r, Y_r\}$ is denoted in red and that the frame fixed to the arm $\{X_a, Y_a\}$ is denoted in blue.

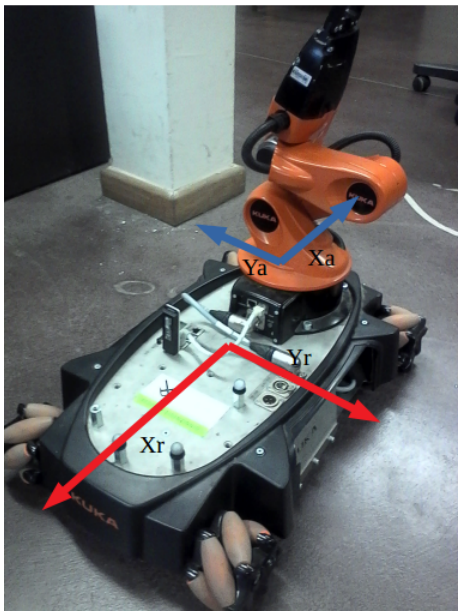


Fig. 3.14: Frame fixed to the mobile base and to the manipulator.

3.6.1 Direct Kinematics Control

First we control the robotic manipulator by direct kinematics, i.e. giving the joint angles as a reference. The drawback of this method is that we don't control directly the position of the end-effector, which is in many practical applications necessary.
The main function basically initialize the ROS node and send a position command to the arm.
The createArmPositionCommand function create an object of type brics_actuator ; it contain all information in order to control the arm. This object is build according to the Vector given as a parameter.

This main function create a node and a publisher. Then it create a object of type JointPositions in order to send it with the publisher. This object contain all information about joint angles of the arm.

```
1  int main(int argc, char **argv){
2
3          ros::init(argc, argv, "youbot_arm_test");
4          ros::NodeHandle n;
5
6          //ros::Publisher gripperPositionPublisher;
7
8          armPublisher = n.advertise<brics_actuator::JointPosi
9                  tions>("arm_1/arm_controller/position_command", 1);
10
11         sleep(1);
12
13         brics_actuator::JointPositions msg;
14         std::vector<double> jointvalues(5);
15
16         //fill the vector with values
17
18         msg = createArmPositionCommand(jointvalues);
19         armPublisher.publish(msg);
```

This function create a JointPositions Object, it take a vector as an input. Then it fill the object using values from the vector. And it also fill field with default values.

```
1  // create a brics actuator message with the given joint position values
2  brics_actuator::JointPositions createArmPositionCo
3                  mmand(std::vector<double>& newPositions) {
```

```
4           int numberOfJoints = 5;
5           brics_actuator::JointPositions msg;
6
7           if (newPositions.size() < numberOfJoints)
8                   return msg; // return empty message if not enough
9                   //values provided
10
11          for (int i = 0; i < numberOfJoints; i++) {
12                  // Set all values for one joint, i.e. time,
13                  //name, value and unit
14                  brics_actuator::JointValue joint;
15                  joint.timeStamp = ros::Time::now();
16                  joint.value = newPositions[i];
17                  joint.unit = boost::units::to_string(boost::units::si:
18                                  :radian);
19
20                  // create joint names: "arm_joint_1" to "arm_joint_5"
21                  //(for 5 DoF)
22                  std::stringstream jointName;
23                  jointName << "arm_joint_" << (i + 1);
24                  joint.joint_uri = jointName.str();
25
26                  // add joint to message
27                  msg.positions.push_back(joint);
28          }
29
30          return msg;
31  }
```

3.6.2 Inverse Kinematics Control

To control directly the position of the end-effector, we have to use inverse kinematics. With this we give the desired position of the end-effector in task space, whereby the joint angles necessary to reach this desired position are calculated.

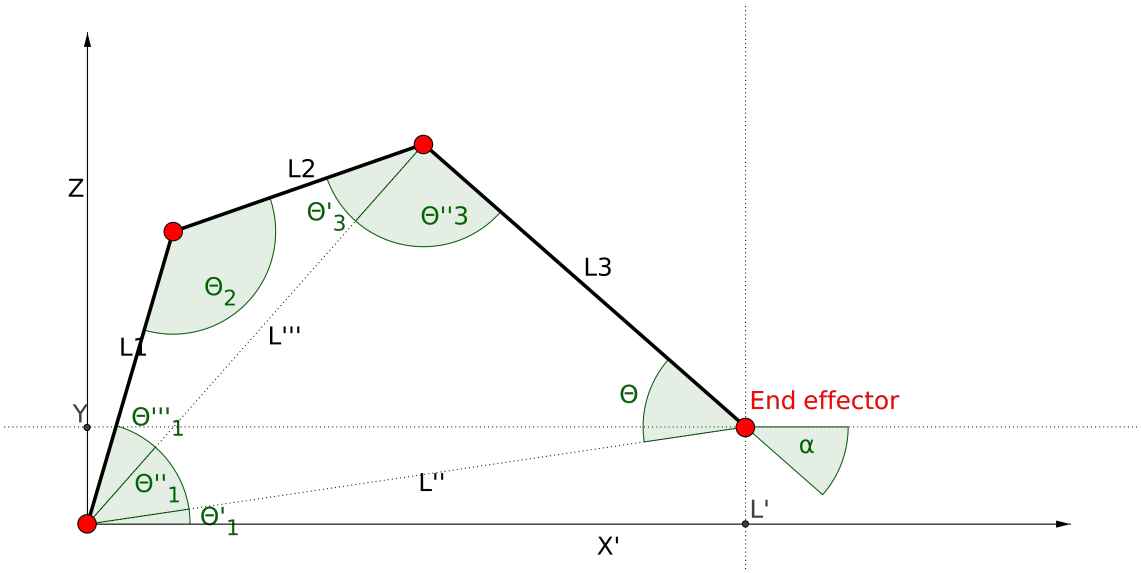


Fig. 3.15: Inverse kinematics calculation

The joints values we need are θ_1, θ_2, θ_3, and θ_0 which is the angle of the first vertical joint on the base. We need to fix the values of the joint 4 and 5 to have a complete constraint geometry.

First we we can compute all distances so we know all distances in the three triangles.

$$L' = \sqrt{X^2 + Y^2} \tag{3.1}$$
$$L'' = \sqrt{L'^2 + Z^2} \tag{3.2}$$
$$L''' = \sqrt{{L_3}^2 + L''^2 - 2L_3 L'' cos(\theta)} \tag{3.3}$$

We can directly compute θ:

$$\theta'_1 = atan2(Z, L') \tag{3.4}$$
$$\theta = \alpha + \theta'_1 \tag{3.5}$$
$$\tag{3.6}$$

Then we can compute θ_1

$$\theta''_1 = acos(\frac{L''^2 + L'''^2 - {L_3}^2}{2L''L'''}) \tag{3.7}$$
$$\theta'''_1 = acos(\frac{{L_1}^2 + L'''^2 - {L_2}^2}{2L'_1 L'''}) \tag{3.8}$$
$$\theta_1 = \theta'_1 + \theta''_1 + \theta'''_1 \tag{3.9}$$
$$\tag{3.10}$$

Then we can compute θ_3

$$\theta'_3 = acos(\frac{{L_2}^2 + L'''^2 - {L_1}^2}{2L_2 L'''}) \tag{3.11}$$
$$\theta''_3 = \pi - \theta''_1 + \theta \tag{3.12}$$
$$\theta_3 = \theta'_3 + \theta''_1 \tag{3.13}$$
$$\tag{3.14}$$

Then we can compute θ_2

$$\theta_2 = \pi - \theta'_3 - \theta'''_1 \tag{3.15}$$
$$\tag{3.16}$$

Then we can compute θ_0

$$\theta_0 = \text{atan2}(Y, X) \tag{3.17}$$

With these equations we can get the values of the joint angles θ_0, θ_1, θ_2, and θ_3 knowing the end-effector pose. My calculation does not work when the end effector position is lower than zero on the z-axis.
In order to be able to control the end-effector pose we need to know α and the lengths of all links.

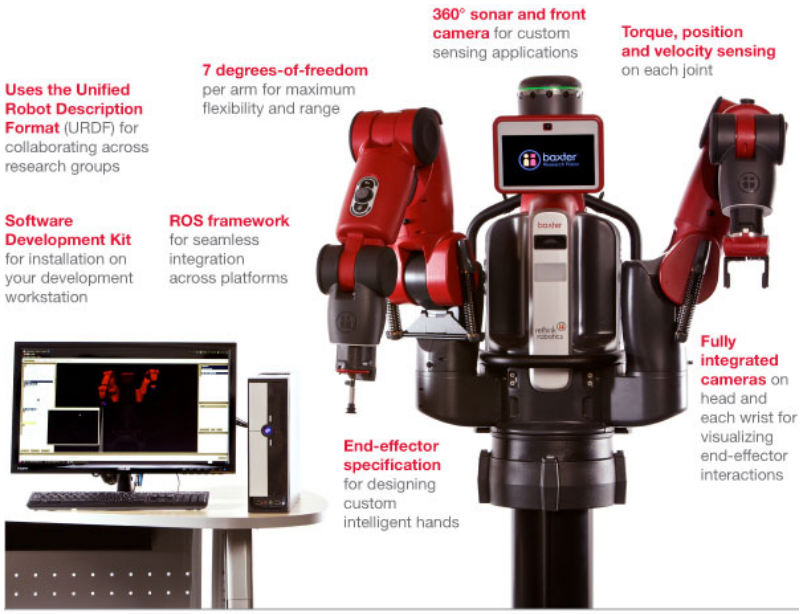
3.6.3 Teleoperation Control with the Active Wand

- Move the arm position using the wand.
- Move the arm position and change the angle of the end effector or the angle of the before last joint.
- Move the robot position while the arm is keeping a given position.

Chapter 4

Rethink Robotics Baxter

Baxter is an industrial robot created in 2013 by Rethink Robotics, a startup founded by Rodney Brooks. This is a collaborative robot also called cobot. It means that humans can work next to the robot without danger. The robot's joints are compliant because the actuators consists of springs, i.e. SEA (Series-Elastic-Actuators). See Fig. 4.1.



Baxter Research Robot – Basic Specifications

Physical
- Robot height: 3'1" without optional pedestal
- Robot height with pedestal: 5'10" - 6'3" (adjustable)
- Arm length to end-effector plate: 41"
- Torso mounting plate diameter: 13.3" (for mounting on table)
- Body weight, without pedestal: 165 lbs
- Body weight, with pedestal: 306 lbs
- Degrees of freedom: 14 (7 per arm)
- Pedestal footprint: 36"x32"

Electrical
- Supply voltage: 120 Volts Alternating Current
- Rated current: 6 Amps

Environmental
- Protection classification: IP50
- Operating temperature range: 32-104° F (0-40°C)

End Effectors
- Vacuum cup with interchangeable cups
- Electric parallel gripper with interchangeable 'fingers' and user-adjustable 'fingertips'

Workstation Requirements (workstation not provided)
- Ubuntu 10.04 LTS and ROS Electric, with minimum specifications:
  - Intel i5 or above
  - 4GB memory or above
  - Min. 2GB of free disk space
  - Ethernet port

Fig. 4.1: Rethink Robotics Baxter

4.1 Body Posture Detection

The goal is to detect the posture of the human upper body. Therefore we place in total 10 reflection markers at the following places on the human upper body:

- 1 on the head,
- 1 on the chest,
- 2 on the shoulders,
- 2 on the elbows,
- 2 on the wrists,
- 2 on the hips.

This can be seen in Fig. 4.2.

Fig. 4.2: Markers to Detect Upper Body Posture

4.1.1 Create a body pattern in Nexus

- Create a new subject called "body"
- Capture the subject few seconds, make sure every marker attached to the body is visible.

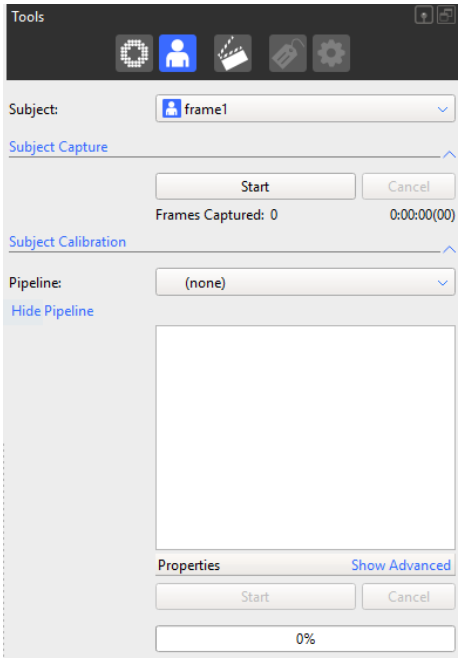


Fig. 4.3: Vicon: subject capture

- Then click on the reconstruct button (gray bubbles).

Fig. 4.4: Vicon: reconstruct button

- Create each segment between joints by selecting the markers.

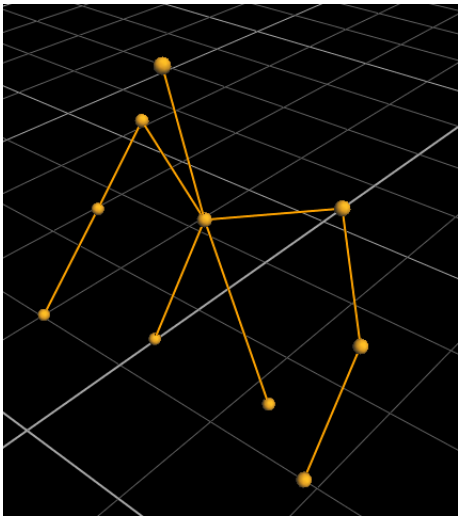


Fig. 4.5: Vicon: creating a segment by selecting body markers

- Create links (here they called it ball joint) between each segment, select parent and child segment in the tree on the left side.

The final tree should looks like this:

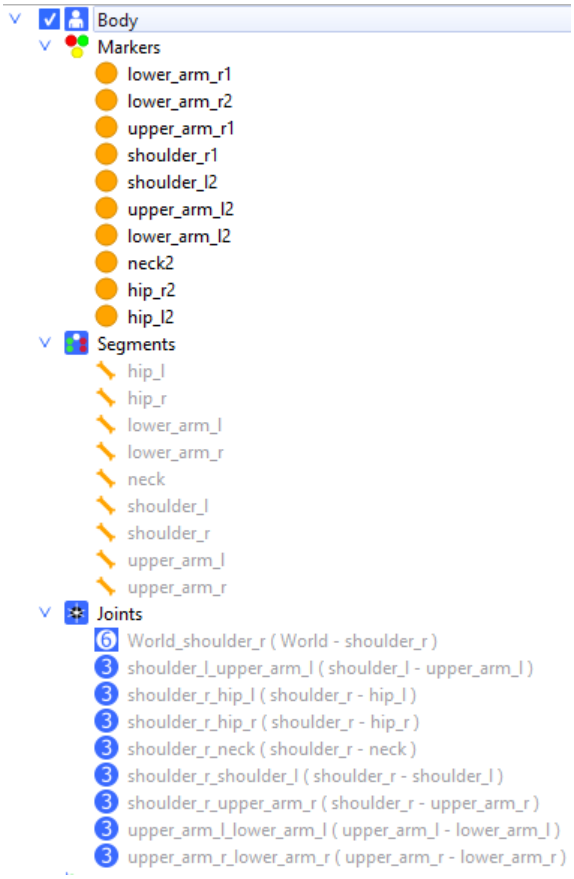


Fig. 4.6: Vicon: final tree

4.2 Camera setup

In order to detect the 10 markers attached to the upper body, we have to position the cameras in such a way that the chance to get occlusions is minimal. Therefore we positioned the cameras as can be seen in Fig. 4.7.
Unfortunately, also with this camera configuration, some of the markers are sometimes occluded by the human body or one of the robot arms.

Fig. 4.7: Camera setup for Baxter robot

4.3 REBA

REBA stands for Rapid Entire Body Assessment. This assessment form uses a systematic process to evaluate the whole body postural MSD (Musculoskeletal disorder) and risks associated with job tasks. A single page worksheet, as depicted in Fig. 4.8, is used to evaluate required or selected body posture, forceful exertions, type of movement or action, repetition, and coupling. The lower the score, the lower the MSD risk, as can be seen in Fig. 4.9. More information can be found on https://ergo-plus.com/reba-assessment-tool-guide/.

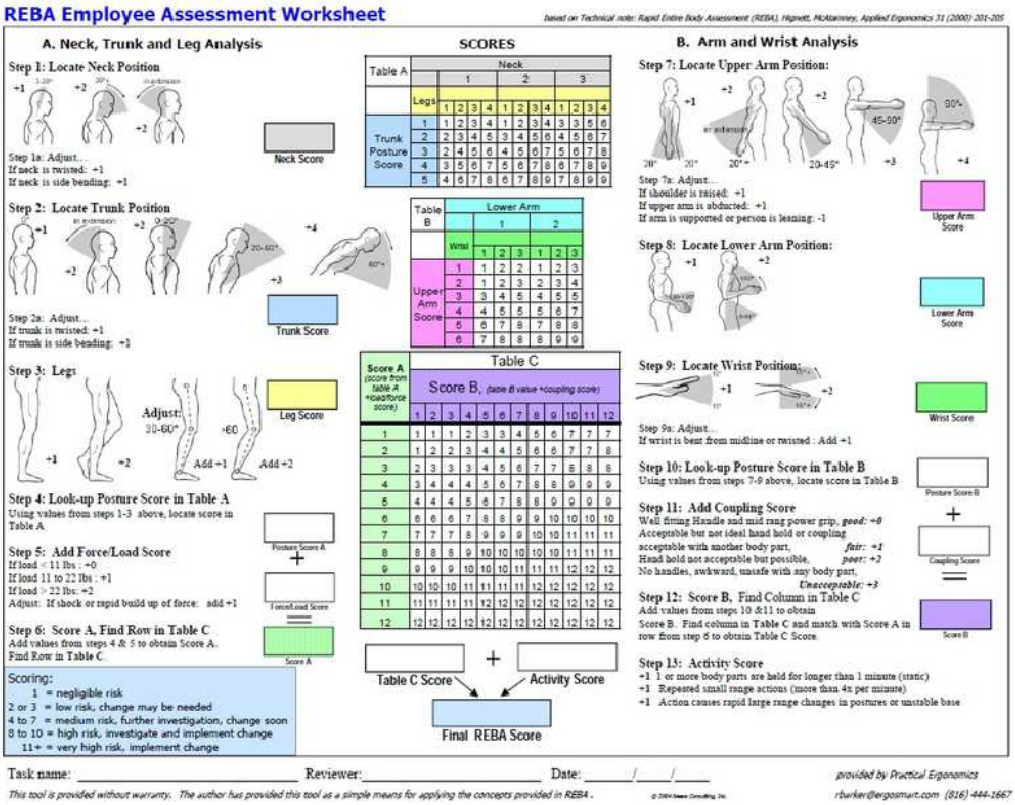


Fig. 4.8: REBA table

Score	Level of MSD Risk
1	negligible risk, no action required
2-3	low risk, change may be needed
4-7	medium risk, further investigation, change soon
8-10	high risk, investigate and implement change
11+	very high risk, implement change

Fig. 4.9: REBA Score

4.4 REBA Score from Vicon Data

This program is publishing tf transformation of the position of each marker, so another ros node already coded by master student can simply calculate the REBA value.

```
#import the vicon_bridge msg
from vicon_bridge.msg import Markers
#callback function
def callback(data):
        br = tf.TransformBroadcaster()
        #transform data into tf
        for i in range(10):
                br.sendTransform(
                (data.markers[i].translation.x,
                data.markers[i].translation.y,
                data.markers[i].translation.z),
                tf.transformations.quaternion_from_euler(0,0,0),
                rospy.Time.now(),
                data.markers[i].marker_name,
                "body")

def main():
        print "body_tracking started"
        rospy.init_node("body_tracking")
        sub = rospy.Subscriber('/vicon/markers', Markers, callback)
        print "publishing to tf"
        while not rospy.is_shutdown():
                rospy.sleep(0.001)
```

Chapter 5

Franka Panda

5.1 Connection with Vicon System

Since the Panda robot has only one Ethernet socket that is already connected to the computer that controls the robot, it was not possible to connect the Vicon computer and the Panda robot via Ethernet, like we did with the Baxter robot.
To solve this connection problem we used an AdHoc network. In the case of the youBot, we created an AdHoc network from the computer controlling the robot with Ubuntu as operating system. In the case of the Panda robot, we created the AdHoc network on the Vicon computer with Windows 10 as operating system. So the Panda robot is connected via Ethernet with the computer controlling it (in Ubuntu) and this computer (that send commands to Panda) is connected with the Vicon computer via the AdHoc network created on the Vicon computer.

Follow this few lines to setup the AdHoc network on Windows 10.

- Open Administrator Powershell : Windows Key + X
- Select Windows PowerShell(Admin)
- Type this line in the terminal netsh wlan set hostednetwork mode=allow ssid=adhocname key=pasword (adhocname is the name of the network, key is the password)
- Then each time you want to start the AdHoc network type this in the terminal netsh wlan start hostednetwork

5.2 Robot Frame

For the experiments with the Panda robot, we did not use the active wand to fix the Vicon frame, since it was not accurate and not easy to know the position of robot w.r.t. the frame fixed by active wand. A more accurate solution is to place three markers on the robot, which fix the center and the orientation of the Vicon frame. During calibration, be careful that you don't use the "standard" way to fix the center of the Vicon frame. This will be explained later.

To fix the Vicon frame, i.e. $\{X_V, Y_V, Z_V\}$, we placed three markers on the robot, see Fig. 5.1. The transformation between the Vicon frame and the robot frame (which origin is in the base of the robot) is this way easy: a translation over the z-axis and an orientation around the z-axis.
To make sure the frame is always at the same place we need to align the first and second joint of the robot as denoted in Fig. 5.2.

Fig. 5.1: Vicon Frame: position of the three markers.

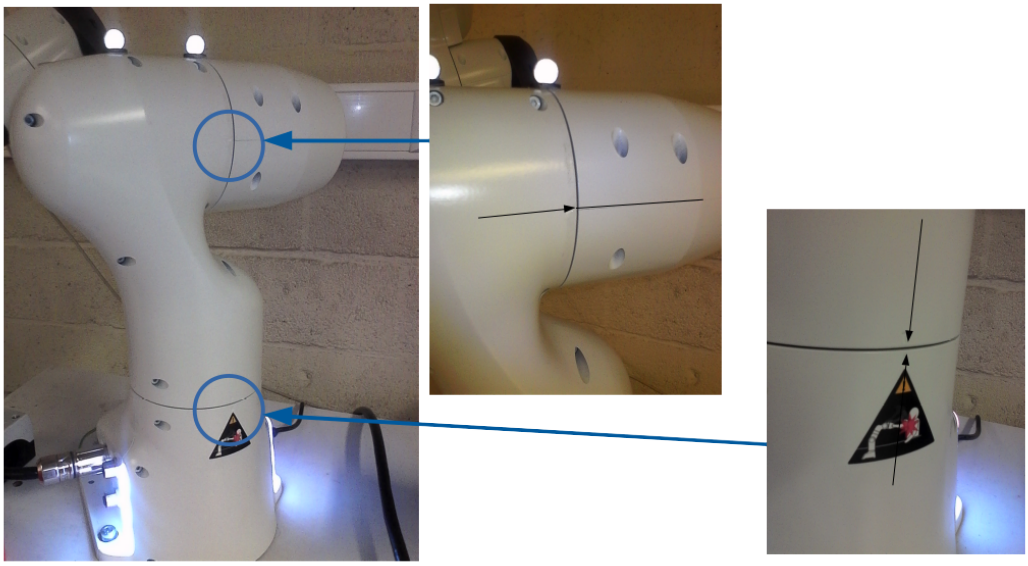


Fig. 5.2: Vicon Frame: alignment of the first and second joints.

In order to set the origin of the Vicon frame, we have to select three markers. The first one is number 0, the second one is marker number 1, and the third one is marker number 2.

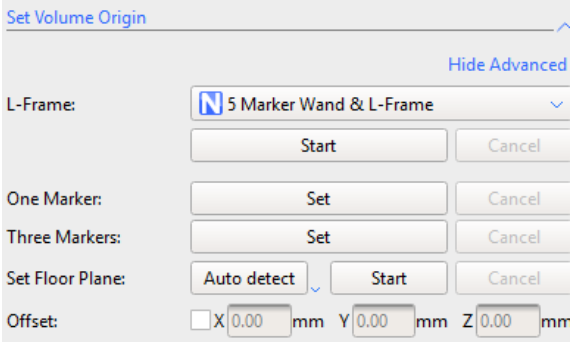


Fig. 5.3: Robot Frame

5.3 Teleoperation of Franka with a Human Arm

5.3.1 Marker Positioning

We placed six markers on an arm, see Fig. 5.4.

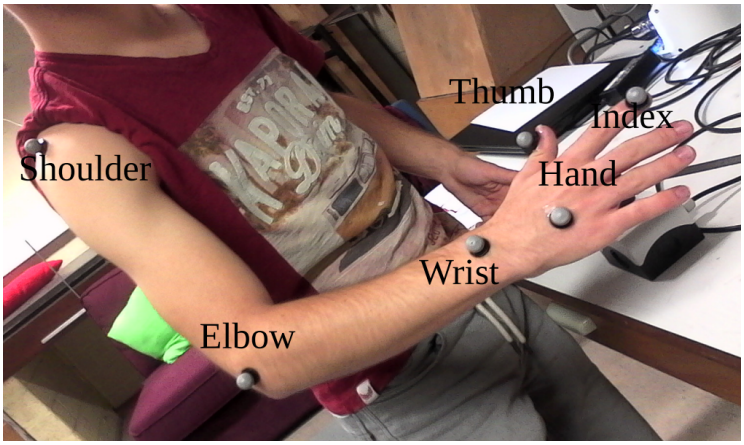


Fig. 5.4: Arm with markers

In Nexus, we created a subject of six markers so that the arm could be recognized, as can be seen in Fig. 5.5 and 5.6.

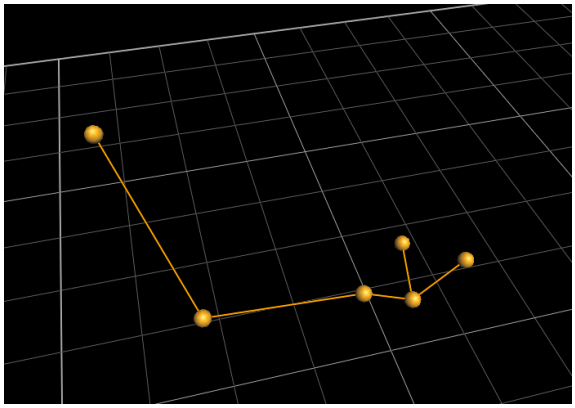


Fig. 5.5: Arm in Nexus

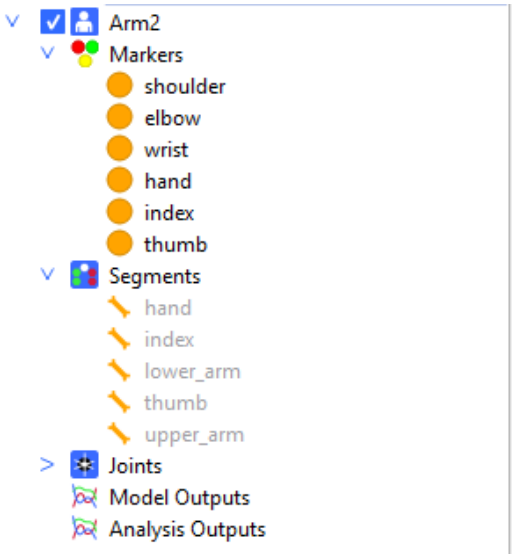


Fig. 5.6: Tree denoting the arm in Nexus

5.3.2 Control Based on Arm Movements

- The end-effector position is controlled using the wrist position. When we start the program the starting position of the wrist is taken as the reference point.
- We can open and close the gripper by joining and opening the thumb and the index. The program calculates the distance between the thumb and the index. If this distance is smaller than a threshold value, the gripper closes, otherwise the gripper opens.
- We can control the 7th joint angle by twisting thumb and index. The program calculates the angle between the vector (elbow,wrist) and (index,thumb).
- We can control the 6th joint angle by bending up and down the wrist. The program calculates the distance between the hand and the wrist.

5.4 Pick and place program

5.4.1 Frames

We created 3 frames using for each of them 3 markers. Markers 0 and 1 are always at the same place and we change the distance of the 3rd marker from the marker 0 in order to distinguish them, see Fig. 5.7.

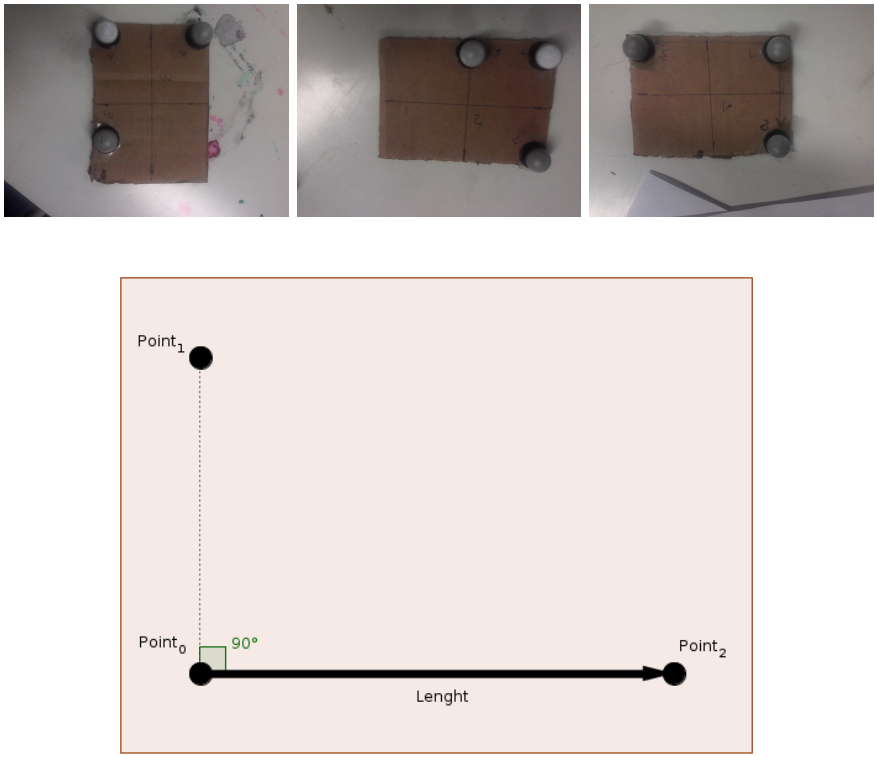


Fig. 5.7: Distance between marker 0 and marker 2 has to be different for each frame.

In Nexus, the three frames can be recognized, as visualized in Fig. 5.8.

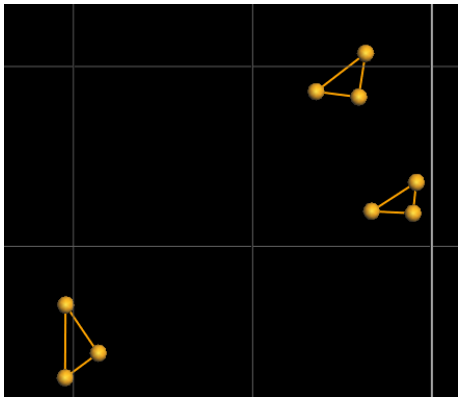


Fig. 5.8: The three different frames in Nexus

5.4.2 Remark

To make sure frames are well recognize we need to follow few instructions.

- All cameras need to be oriented in the direction of the area.
- We need many different points of view with the different cameras.
- Environmental Drift Tolerance can be changed a bit too, see Fig. 5.9. This parameter changes the error we allow to recognize a marker pattern.

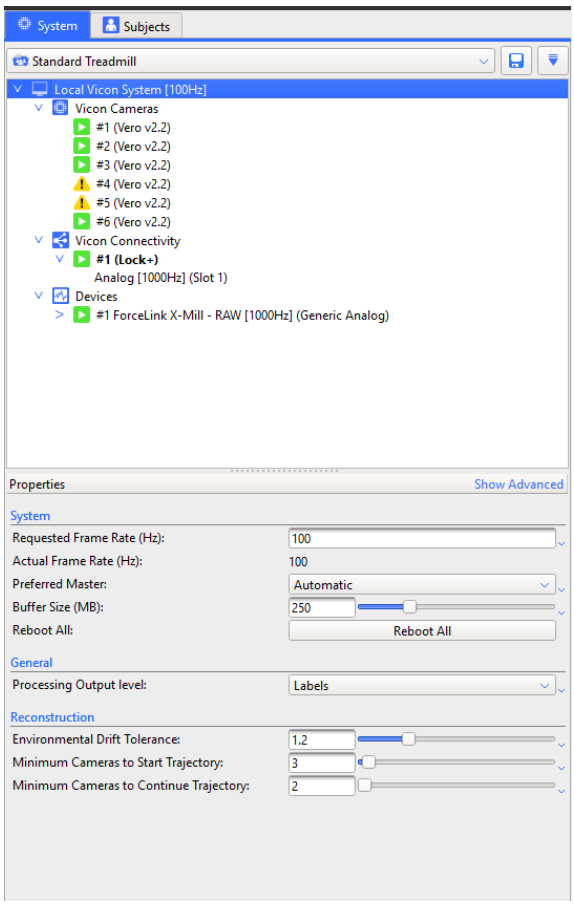


Fig. 5.9: Environmental Drift Tolerance in Nexus