# Human Centred Robotics

## Tutorials Part 2:
## Xtion Pro Depth Sensor and ROS Transforms

# Overview

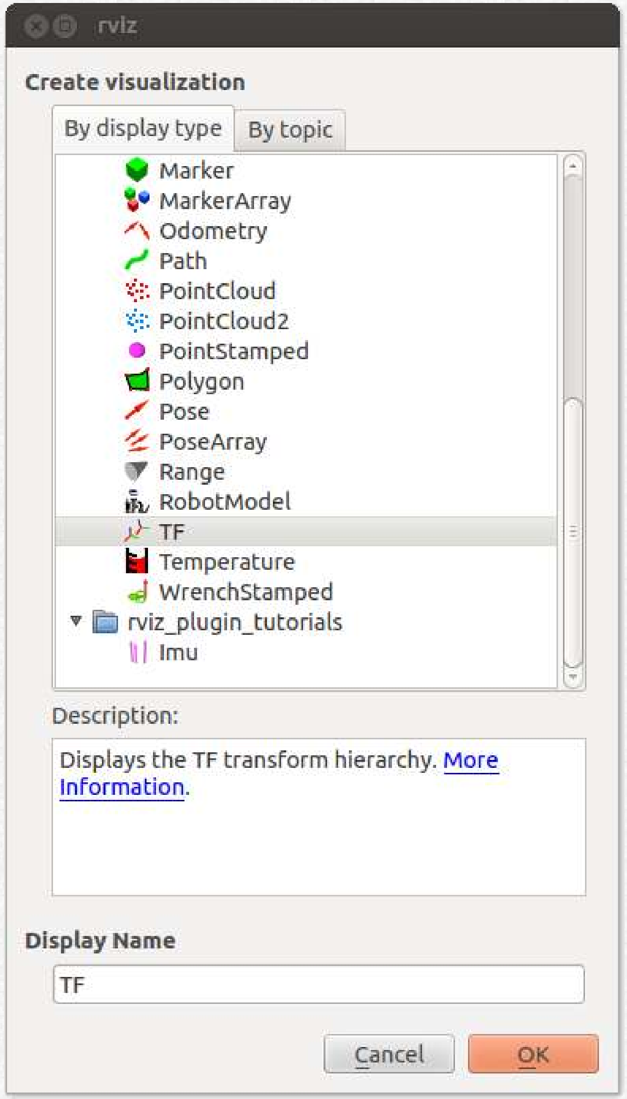
- Introduction to Xtion Pro
- ROS Transforms tutorial
  - What are frames?
  - Listening to frames
  - Broadcasting frames
  - Transforming points
- Goal: Robot to follow you!

# Depth Sensors: Xtion Pro

- RGB camera and 2 microphones
- Depth sensor
  - Infra-red emitter sends a laser grid
  - Recognised by infrared sensor
  - Distortion → Depth
- Skeleton detection OpenNI and NITE
  - Wrapper in ROS and already installed in your laptops

# Let's get the Xtion working

- Fire up RViz
  - roscore
  - rosrun rviz rviz
- Add → TF



Description:

Displays the TF transform hierarchy. More Information.

Display Name

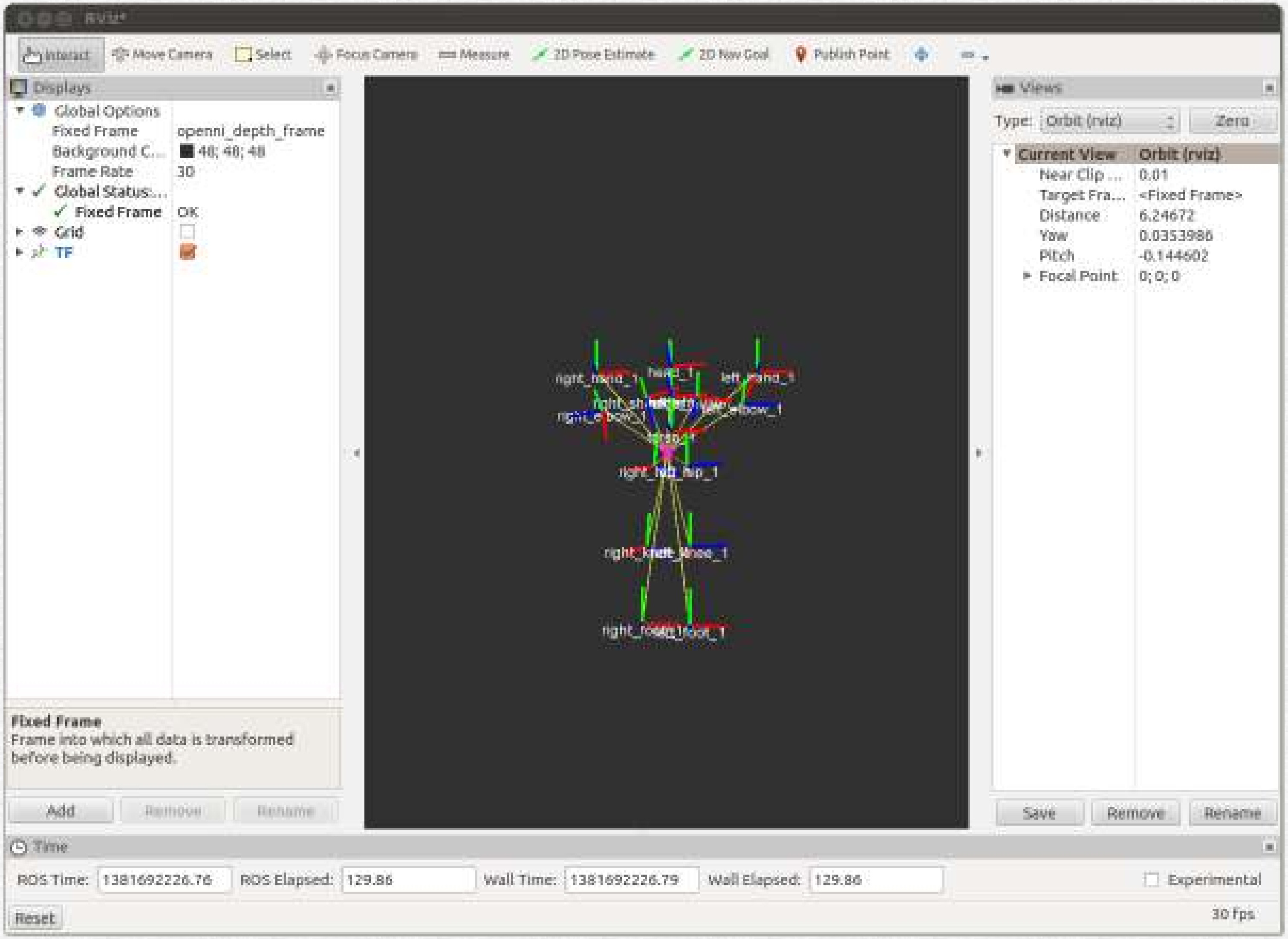
TF

# Almost there...

- Connect the Xtion Pro to the USB port
- In another terminal type

  -
    ```
    rosrun openni_tracker openni_tracker
    ```
- Back to RViz
  - Under Global Options set Fixed Frame to openni_depth_frame

# Et voilà!



Displays

- Global Options
  - Fixed Frame: openni_depth_frame
  - Background C...: 48; 48; 48
  - Frame Rate: 30
- ✓ Global Status:...
  - ✓ Fixed Frame: OK
- Grid: □
- TF: ☑

**Fixed Frame**
Frame into which all data is transformed before being displayed.

Views

Type: Orbit (rviz) Zero

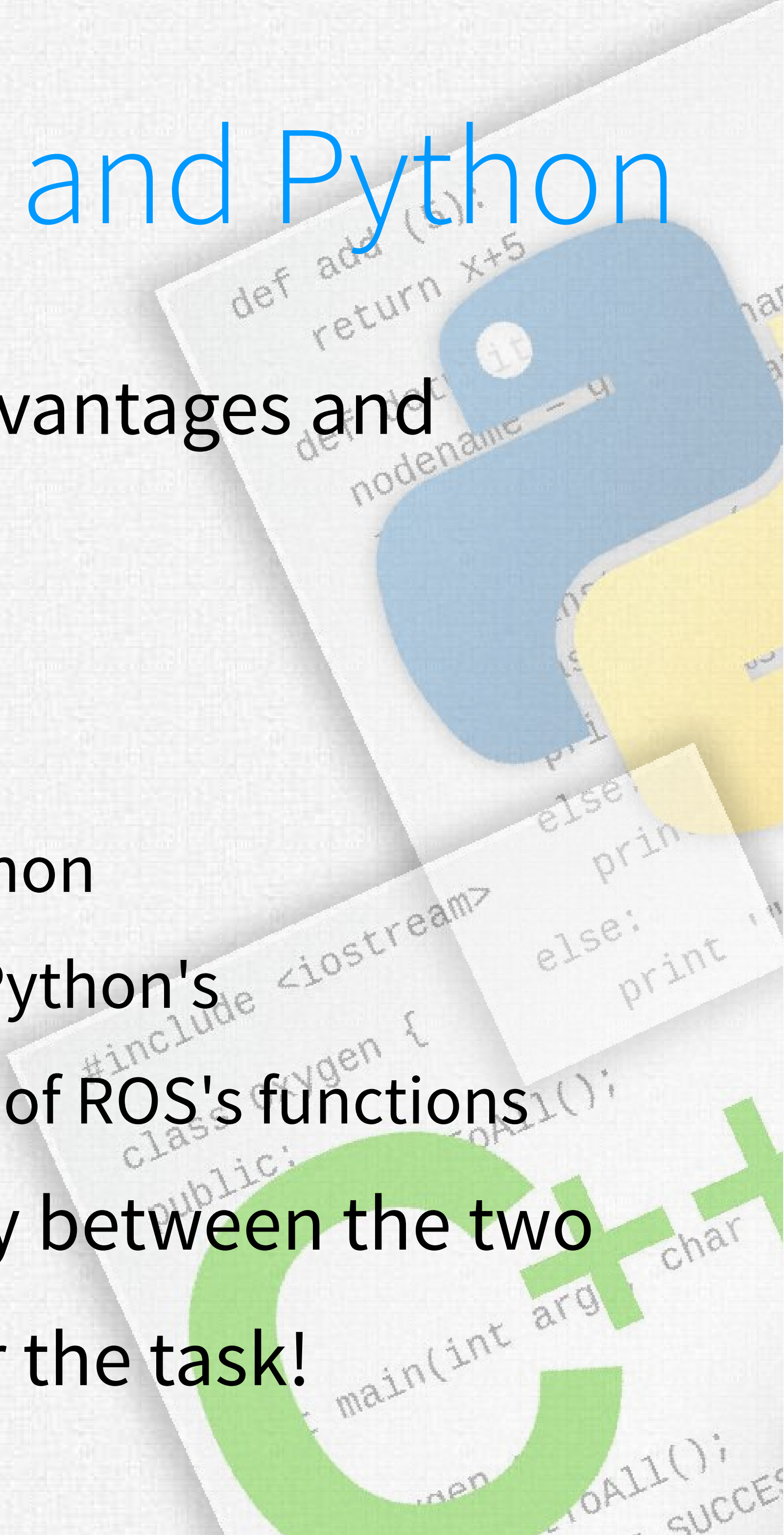
| Current View | Orbit (rviz) |
|---|---|
| Near Clip ... | 0.01 |
| Target Fra... | <Fixed Frame> |
| Distance | 6.24672 |
| Yaw | 0.0353986 |
| Pitch | -0.144602 |
| Focal Point | 0; 0; 0 |

ROS Time: 1381692226.76 ROS Elapsed: 129.86 Wall Time: 1381692226.79 Wall Elapsed: 129.86 □ Experimental

30 fps

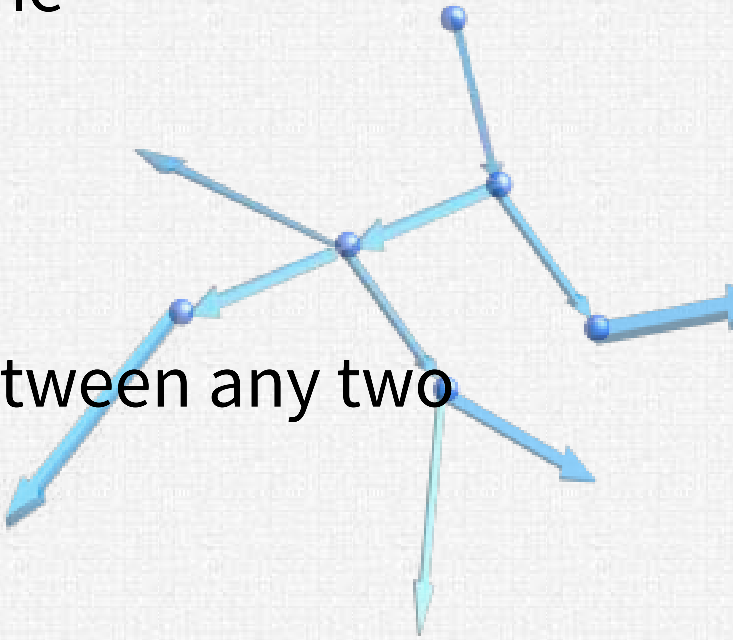
# C++ and Python

- Both are great languages with advantages and disadvantages
  - C++ creates **faster** executables
  - C++ needs to be **compiled**
  - Writing code takes **less time** in Python
  - C++ syntax is **more complex** than Python's
  - Python does **not** have **access** to all of ROS's functions
- ROS has excellent interoperability between the two
- Use the one that is best suited for the task!



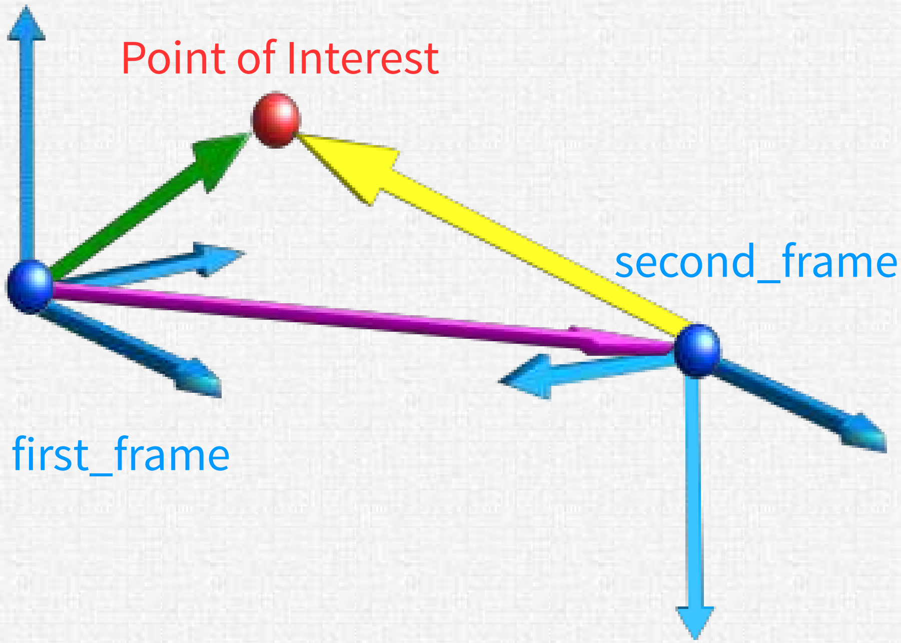
# Frames and ROS TF

- Frame stands for **frame of coordinates**
  - Each frame has its own axes
  - Each frame is defined in terms of its parent
  - We can write the position of points/objects/areas with respect to the most convenient frame
- TF is a ROS core library
  - It builds a tree of frames
  - This allows to find the transform between any two connected frames



# Why are frames important?

- Point of interest is easier to describe from the first frame (green vector)
- But we need to know its value with respect to the second frame (yellow vector)
  - Note second frame may be moving!
- ROS can do this for us, as long as we provide the pink vector



# Listening to frames

**Objective**: Log the transform details between the Xtion's root frame (openni_depth_frame) and the user's torso

- Read the source code **carefully**
- Look out for //READMEs, //TODOs and //HINTs
- Ask if you don't understand something!
- Make sure Xtion is connected before you run the program

**Go to**: ~/hcr2013/exercises/part2

**Source code**: ./src/step1/src/Main.cpp

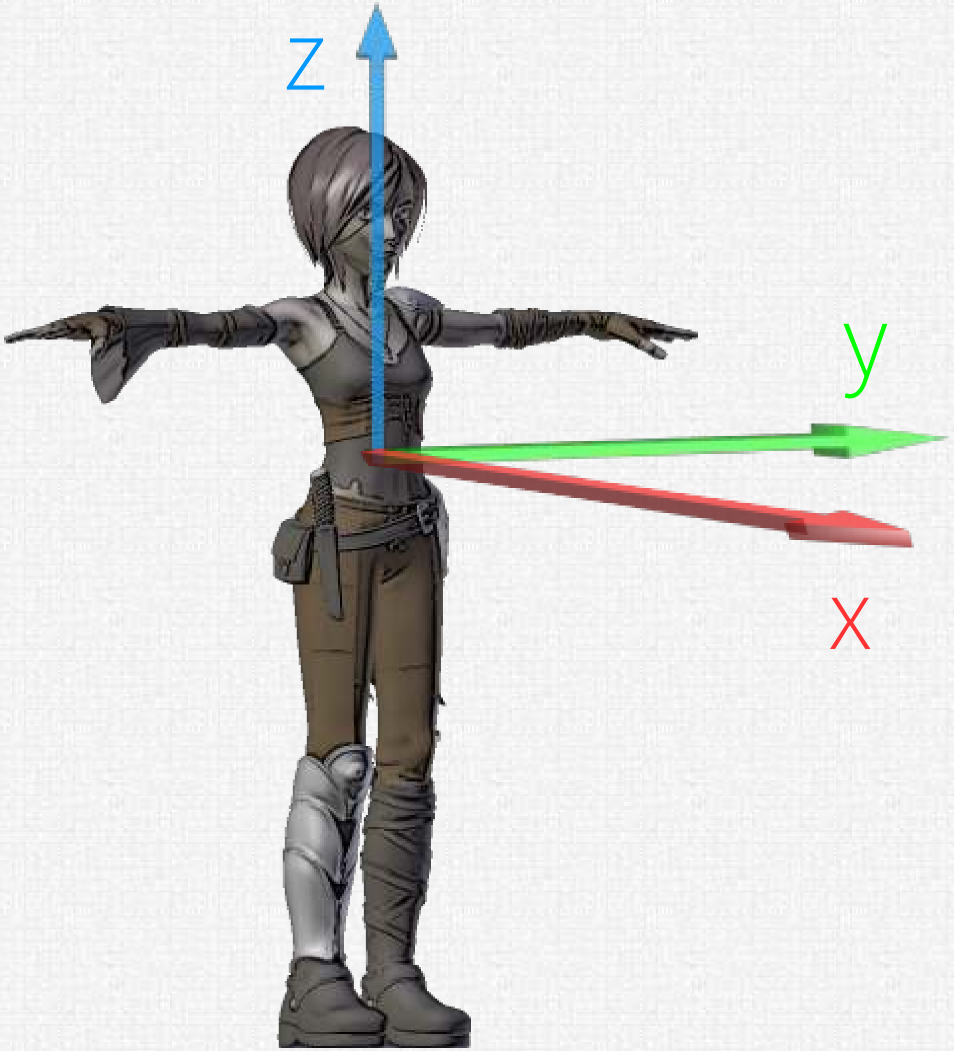
**Compile**: catkin_make

**Setup**: source devel/setup.bash

**Launch**: roslaunch step1 tf.launch

**Solution**: ~/hcr2013/solutions

# Standard ROS Frame



# Broadcasting frames

**Objective**: Create a new frame (hat) on top of the head of the user

- Read the source code **carefully**
- Look out for //READMEs, //TODOs and //HINTs
- Ask if you don't understand something!
- Make sure Xtion is connected before you run the program

**Go to**: ~/hcr2013/exercises/part2

**Source code**: ./src/step2/src/Main.cpp

**Compile**: catkin_make

**Setup**: source devel/setup.bash

**Launch**: roslaunch step2 tf.launch

**Solution**: ~/hcr2013/solutions

# Transforming 3D points

**Objective**: log the coordinates of all “virtual flies” with respect to the user's right hand

- Read the source code **carefully**
- Look out for //READMEs, //TODOs and //HINTs
- Ask if you don't understand something!
- Make sure Xtion is connected
- Add PointCloud to RViz and set topic to flies

**Go to**: ~/hcr2013/exercises/part2

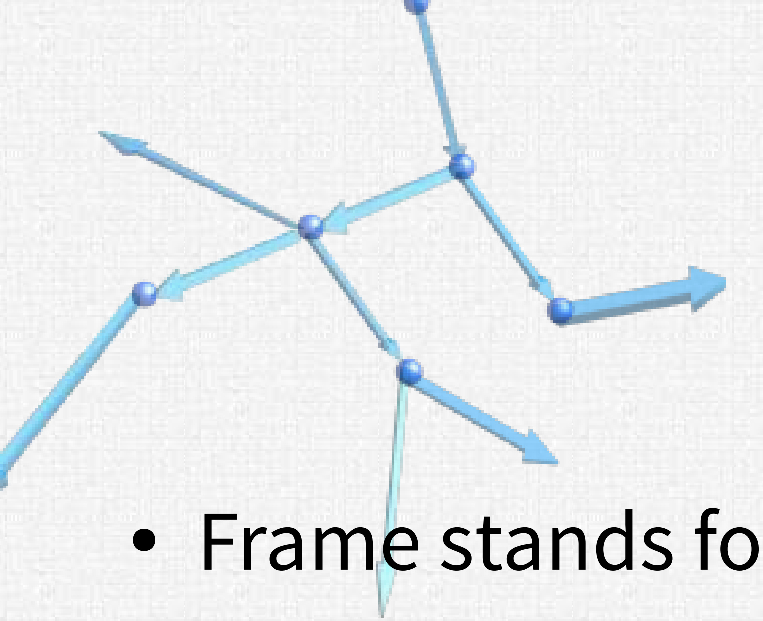
**Source code**: ./src/step3/src/Main.cpp

**Compile**: catkin_make

**Setup**: source devel/setup.bash

**Launch**: roslaunch step3 tf.launch

**Solution**: ~/hcr2013/solutions



# ROS TF recap

- Frame stands for **frame of coordinates**
- In ROS we can transform between different frames, so long as they are connected
  - To transform points we use transformPoints()
  - To get a transform we use lookupTransform()
  - To create a new frame we use sendTransform()
  - ros::Time(0) to use the most recent transform

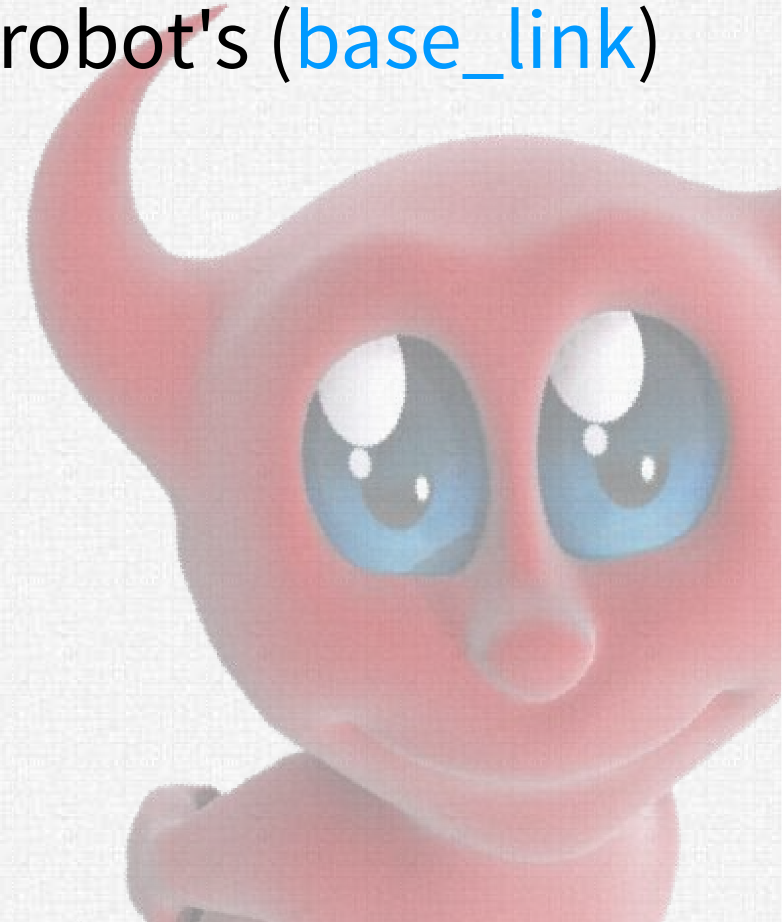
# Getting the P3-AT to follow you

- Previously, P3-AT teleoperation with joystick
- Today, use Xtion to replace joystick input
- First off, we need to know where is the Xtion positioned with respect to the robot
- Next, we figure out where is the user in relation to the robot
- Finally, we send a velocity command to the robot

Image by Randall Munroe *http://what-if.xkcd.com/5*

# The devil is in the details

- Use TF to link Xtion's root frame (openni_depth_frame) with the robot's (base_link)
  - This is done in tf_broadcaster
- First part of the activity



# TF Broadcaster

**Objective**: Link Xtion's root frame to the robot's frame

**Note**: You will need to measure the offset between the Xtion and the P3-AT

- Read the source code **carefully**
- Look out for //READMEs, //TODOs and //HINTs
- Ask if you don't understand something!
- Cannot execute just yet

**Go to**: ~/hcr2013/exercises/part2

**Source code**: ./src/xtion_follower/src/tf_broadcaster/Main.cpp

**Compile**: catkin_make

**Solution**: ~/hcr2013/solutions

# What about the rest?

- Transform a point at (0,0,0) with respect to the user's torso to the robot coordinates
- Project new point to a 2D plane (Xtion deals with 3D, but robot moves in 2D)
- Compute euclidean distance and angle between robot and human and use them to move the robot

# Xtion Follower

**Objective**: Transform point at (0,0,0) with respect to user's torso to the robot's frame. Project to a 2D plane (ignore z component!). Compute distance and angle difference.

- Read the source code **carefully**
- Look out for //READMEs, //TODOs and //HINTs
- Ask if you don't understand something!
- To launch, see next slide

**Go to:**
~/hcr2013/exercises/part2

**Source code**:
./src/xtion_follower/src/
xtion_follower/Main.cpp

**Compile**: catkin_make

**Setup**: source devel/setup.bash

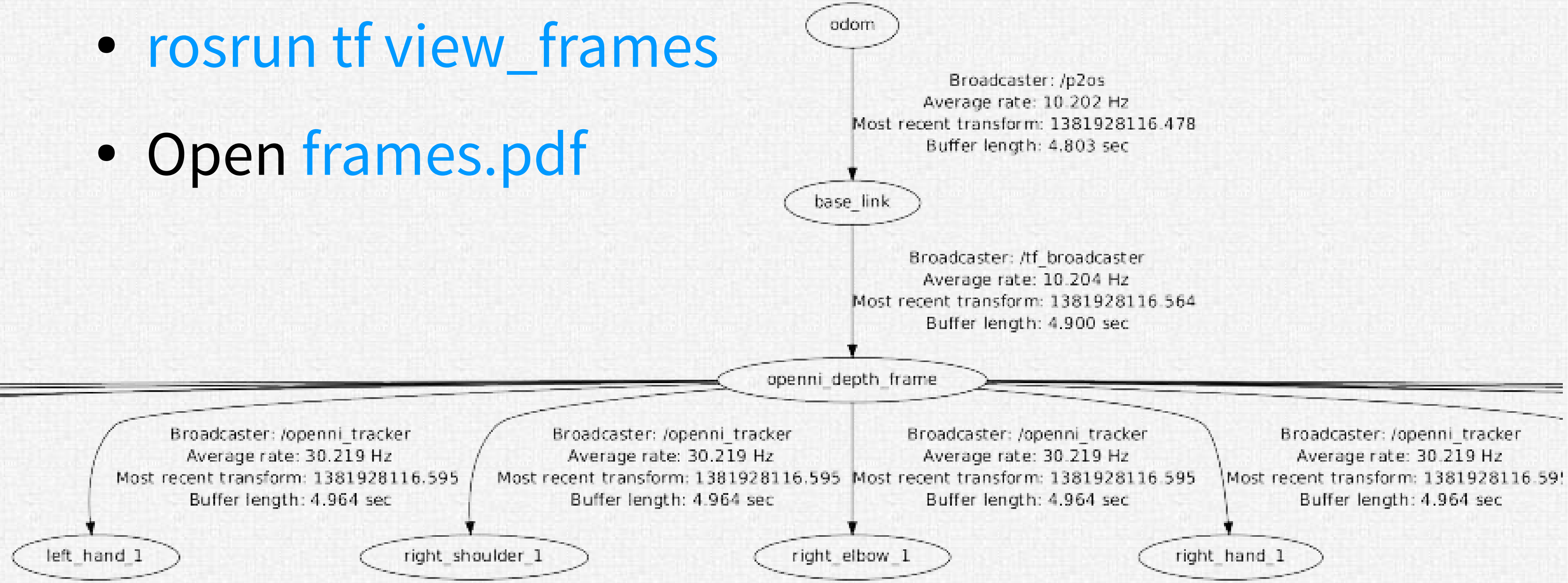
**Solution**: ~/hcr2013/solutions

# Still with us?

- If code compiles, we will review the code and let you run it on the P3-AT!
- Execute:
  - roslaunch xtion_follower xtion_follower.launch
  - Remember to connect both the Xtion Pro and the robot
- Keep trying until it works correctly!

# Debugging TF

- You can use RViz or rqt_graph, but sometimes that is not enough
- rosrun tf view_frames
- Open frames.pdf



odom

Broadcaster: /p2os
Average rate: 10.202 Hz
Most recent transform: 1381928116.478
Buffer length: 4.803 sec

base_link

Broadcaster: /tf_broadcaster
Average rate: 10.204 Hz
Most recent transform: 1381928116.564
Buffer length: 4.900 sec

openni_depth_frame

Broadcaster: /openni_tracker
Average rate: 30.219 Hz
Most recent transform: 1381928116.595
Buffer length: 4.964 sec

left_hand_1

Broadcaster: /openni_tracker
Average rate: 30.219 Hz
Most recent transform: 1381928116.595
Buffer length: 4.964 sec

right_shoulder_1

Broadcaster: /openni_tracker
Average rate: 30.219 Hz
Most recent transform: 1381928116.595
Buffer length: 4.964 sec

right_elbow_1

Broadcaster: /openni_tracker
Average rate: 30.219 Hz
Most recent transform: 1381928116.59
Buffer length: 4.964 sec

right_hand_1

# In summary

- TF is a powerful tool that greatly simplifies common robotic tasks
- Xtion Pro can be used to easily develop interesting Human-Robot Interaction experiences

# That is all… for now

- “You have taken your first step into a larger world”
- Meetings with Theo and Miguel on the 2nd & 11th
- Have fun!