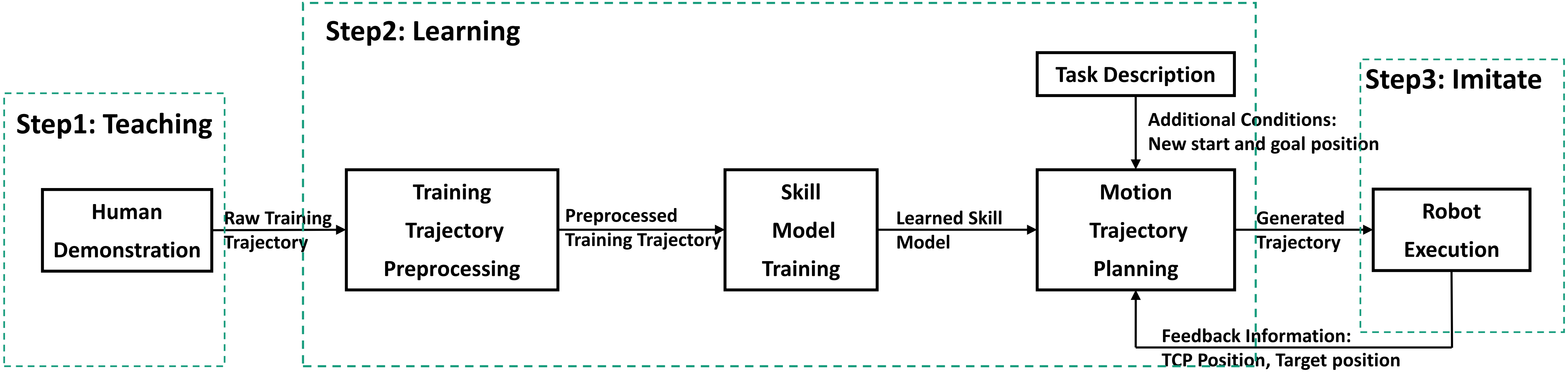
WORKFLOW OF LEARNING FROM DEMONSTRATION

Fraunhofer
IWU

System Overview of Learning from Demonstration



Workflow of Learning from Demonstration

- **Step 1 Teaching Process**
  - Human teacher demonstrate a motion under a motion capturing system (Openpose + ZED)
  - The demonstrated motion trajectory is recoreded as the raw training trajectory
- **Step 2 Learning Process**
  - Preprocess the raw training trajectory with Kalman filter
  - Find the keypoints of the training trajectory and divide the entire training trajectory into segments
  - Use DMP to get the model training trajectory
  - Set the new start and end position of the motion to generate a new motion trajectory
- **Step 3 Robot imitate human motion in simulation**
  - UR10 execute the generated motion trajectory in ROS Gazebo

Fraunhofer IWU

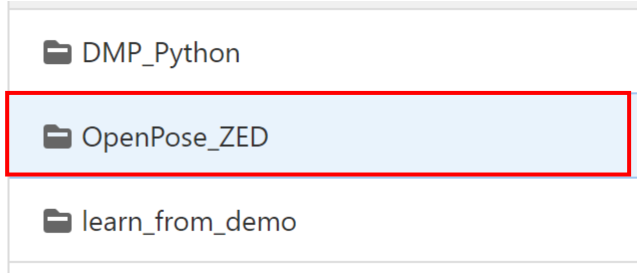
Step 1: Teaching Process

- **Software Setup**

	Information:
Operating System	Windows 10
Dependences	OpenCV 4.1.1 CUDA 10.0 with cudnn 7.4.2 ZED SDK 3.1.2 Openpose C++ API

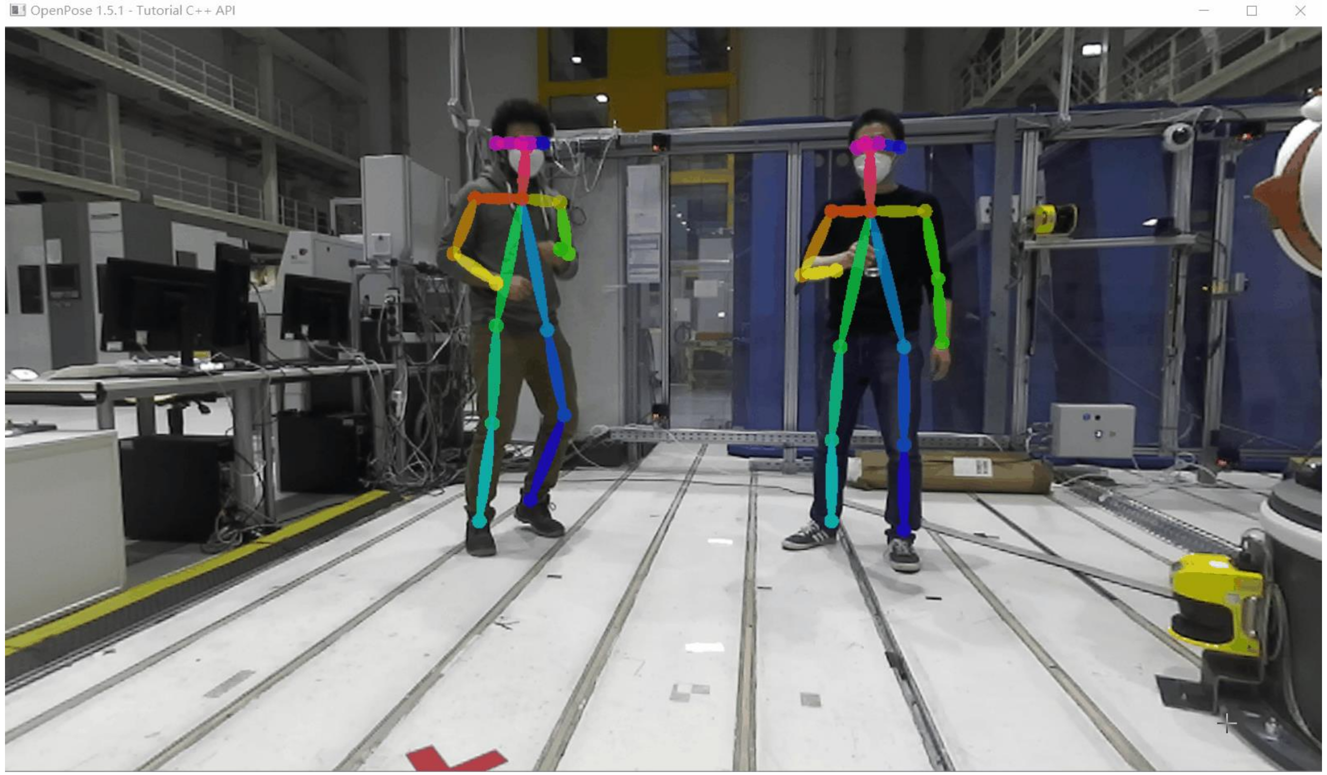
- **Usage**
  1. Input the name of the TXT.file used to save the demostated motion
  2. Press ‘s‘ for start saving
  3. Press ‘q’ for quite and save the record data into the TXT.file

- **Folder in Gitlab Project:**



Step 1 Teaching Process

Example of observational teaching



data5.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

x	y	z
-173.994247	-149.599991	-991.859741
-180.387680	-149.983185	-994.400330
-173.507706	-149.181641	-989.086121
-165.614716	-147.421799	-977.418152
-165.768311	-147.558517	-978.324646
-156.078201	-149.727768	-954.841675
-138.248749	-150.524475	-959.922424
-104.774483	-143.974426	-918.151550
-99.633179	-149.993591	-921.391785

An example of saved motion

Input: Human demonstration → **Teaching Module** → **Output**: Recorded motion trajectory

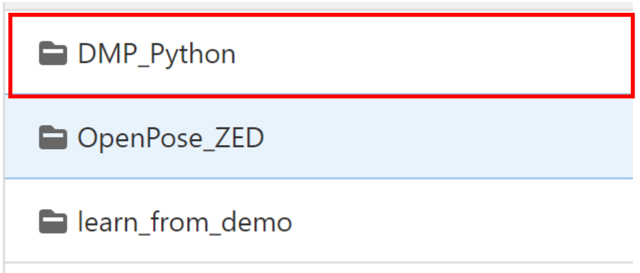
Step 2: Learning Process

- **Software Setup**

	Information:
Operating System	Windows 10, Linux (Ubuntu 16.04)
Python version	3.6
dependences	numpy scipy matplotlib PyYAML

- **Usage**
    1. Set parameters in ‚config_param1.yaml‘
    2. Set the name of the traininng trajectory
    3. Set the new start and end positions
    4. Set the name of the generated trajectory
    5. Run the python script ‘DMP_Keypoints.py’

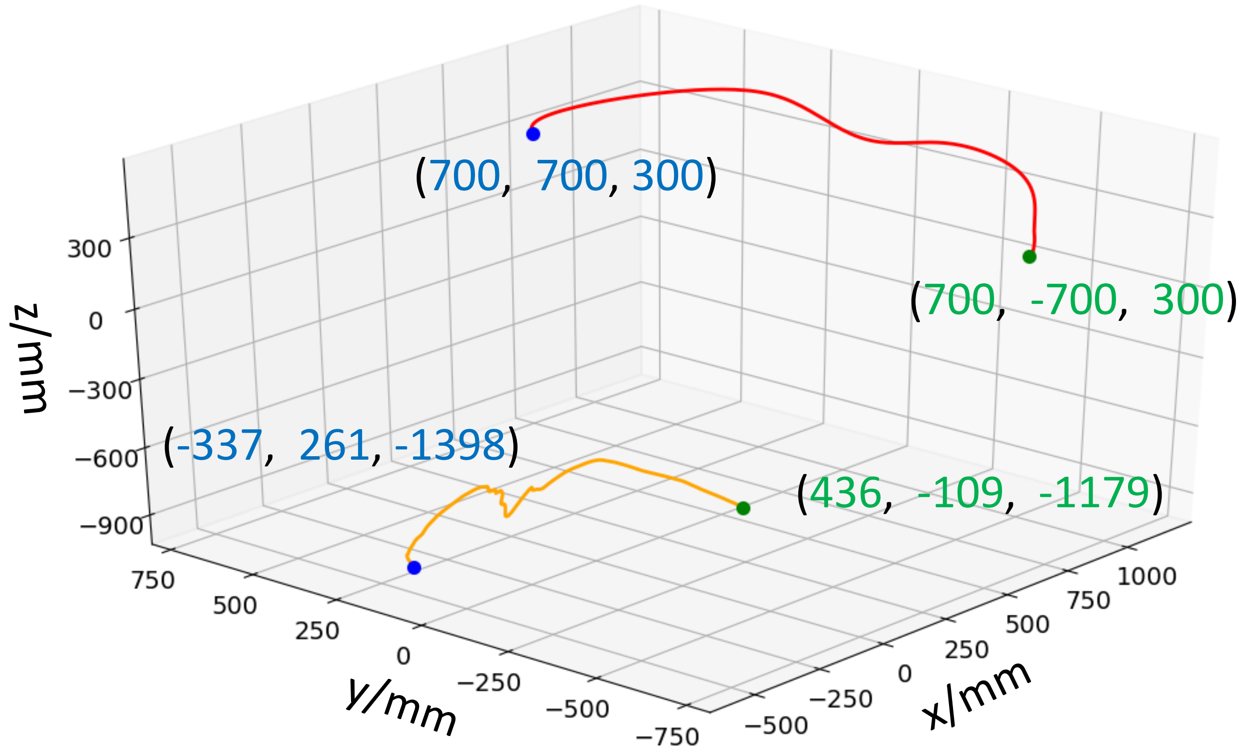
- **Folder in Gitlab Project:**



Fraunhofer IWU

Step 2 Learning Process

An example of generated motion trajectory (change start and goal position)




traj_gen_21.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

0.701000 ; 0.699247 ; 0.299124
0.702897 ; 0.697753 ; 0.297468
0.705590 ; 0.695530 ; 0.295127
0.708989 ; 0.692599 ; 0.292192
0.713007 ; 0.688980 ; 0.288759
0.717565 ; 0.684702 ; 0.284926
0.722590 ; 0.679798 ; 0.280805
0.728018 ; 0.674306 ; 0.276525
0.733790 ; 0.668275 ; 0.272241
0.739857 ; 0.661762 ; 0.268141
0.746177 ; 0.654834 ; 0.264444
0.752714 ; 0.647562 ; 0.261387
0.759437 ; 0.640018 ; 0.259201

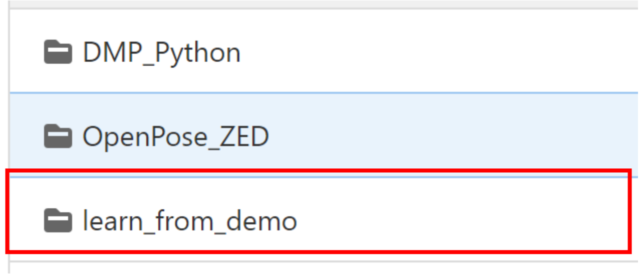
Input: Record motion trajectory → **Learning Module** → **Output**: generated motion trajectory

Step 3: Robot imitate human motion in simulation

- **Software Setup**

	Information:
Operating System	Linux (Ubuntu 16.04)
ROS version	Kinetic 1.12.14
Simulation	Gazebo
dependences	moveit universal_robot

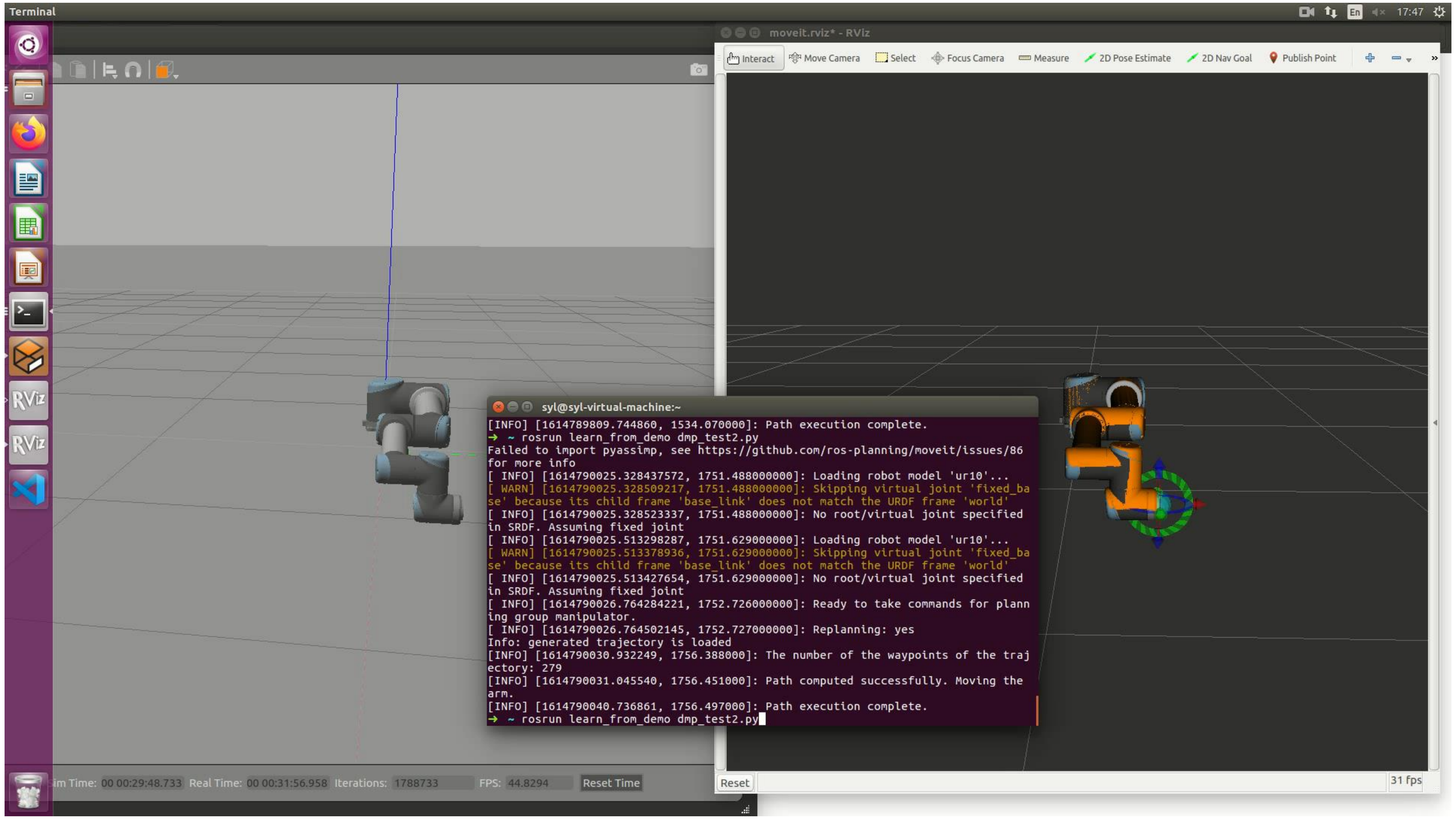
- **Folder in Gitlab Project:**



- **Usage**
  1. Start the simulation environment Gazebo and needed ros node:
     Roslaunch learn_from_demo dmp_ur10.launch
  2. Load the generated motion trajectory:
     Put the generated motion trajectory (txt.file) into folder
     open python script ‚robot_imitate.py‘ and input the file name of the generated motion trajectory
  3. UR10 Robot execute the generated motion trajectory
     rosrun learn_from_demo robot_imitate.py

Fraunhofer IWU

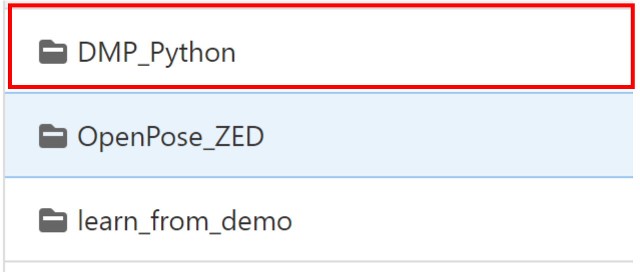
Example of Robot Execution



Input: generated motion trajectory → **Robot Imitate**

Fraunhofer IWU

Structure of DMP_Python (Code for model training)



- Structure of folder **DMP_Python**
  - config_param1.yaml # yaml file to set parameters
  - DMP_Standard.py # main function 1: for testing standard DMP (without keypoints)
    - DMP.py # functions for Dynamic Movement Primitive
    - plot_result3D.py # Plot results
  - DMP_Keypoints.py # main function 2: for testing DMP with keypoints)
    - Training_Traj_Preprocessor.py # functions for training trajectory preprocessing
    - Model_Training.py # functions for model training
    - Skill_Model_Optimizer.py # functions for adjusting the learned motion model
    - Trajectory_Generator.py # functions for generating trajectory by using adjusted learned model
    - plot_function.py
  - Load_Save_Data.py # Load training Data
  - kalman_filter.py # Kalman Filter
  - Obstacle_avoid.py # functions for testing obstacle avoidance
  - variance_calculation.py # Calculating the variance between training and generated motion trajectory

Parameters for model training (Set in the config_param1.yaml)

Parameter	Information (Comments)
nb_Samples	*Number of demonstrations*
nb_Data	*Length of training data after resampling (interpolated training data)*
KP	*Stiffness gain (Spring Damped System)*
alpha	*Decay factor (Phase Variable)*
dt	*Duration of time step*
nbStates	*Number of activation function (i.e. number of States in the GMM)*
threshold_keypoint	*threshold used for select keypoints: remove the keypoints locate too close*
threshold_goal	*threshold for judging if the end position is close enough to the goal position*
data_handwrite_path	*path and name of handwriting training trajectory*
data_handover_path	*path and name of handover training trajectory (from OpenPose+ZED)*
traj_gen_path	*path and name for saving the generated motion trajectory*

- The new start and end position of generated motion trajectory can be set in the **DMP_standard.py**
- Load handwritng or handover data can select in the main function in **DMP_standard.py** and **DMP_Keypoints.py**
- Dimension (x,y,z) of Keypoints found can be set in the function ***keypoint_finder*** in script ***Training_Traj_Preprocessor.py*** (line 219)
- When using the real data (get from openpose) use Kalman filter before model training (**DMP_Keypoints.py** line 290)