# Installation

Copy the RobustSim.ini file to your C:\users\XXX\Documents folder. The folders with the interfaces can be placed anywhere on your computer. In MacOS and Linux it must be placed either in the /home/user directory or in its documents directory.

# Launch

If you are using the MATLAB "MainRobust.m" then launch robust_interface/Robust.exe file.

If you are using the MATLAB "MainDexrov.m" then launch dexrov_interface/DexROV.exe file.

If you do not launch the proper interface, you will have weird results on screen, as the robot kinematics differ between the simulation (Matlab) and the rendering (the Unity interface).

# How to move the camera

Pressing 'END' allows you to control the POV with the mouse.

W: forward
S: backward
A: left
D: right
Q: go up
E: go down

The initial robot position is below the initial camera position (it is near the pipeline)

# Matlab Simulation

## Main files (Entry points of the simulation)

- MainRobust.m: main file to launch to use the ROBUST model
- MainDexrov.m: main file to launch to use the DexROV model

## Data structures

- struct uvms: contains most of the UVMS data; the structure is modified by several scripts to divide the code in logical functions
- struct mission: an auxiliary data structure that can be used to implement different phases of a more complex mission
- struct plt: an auxiliary data structure that contains several data vectors saved for end-of-simulation plots
- InitUVMS.m initializes the UVMS data structure with the initial values; add here additional initializations to prevent Matlab errors
- InitDataPlot.m initializes the plt data structure with pre-allocated values; if you add data to be plotted, add the corresponding initialization here

## Script files to be changed

- ComputeJacobians.m: script that calculates the task Jacobians; you can add further Jacobians here
- ComputeTaskReferences.m: script that calculates the task reference velocities; add here the computation of new tasks
- ComputeTaskActivations.m: script that calculates the activation functions; add here the computation of task activations. For complex behaviour, you can exploit the mission data structure to activate/deactivate tasks depending on the mission phase, therefore smoothly changing action
- UpdateMissionPhase.m: a script that can be used to implement the mission phases; this script should contain the logic that changes the auxiliary mission data structure
- PrintPlot.m: a script that plots the relevant data at the end of the simulation; can be modified to produce additional plots

## Basic features already implemented

The main files have a skeleton already implemented. In particular, the main simulation loops contains:

- The calls to the UpdateTransform, ComputeJacobians, Compute TaskReferences, ComputeTaskActivation scripts. Therefore, adding new control tasks can be done by modifying the aforementioned scripts.
- A control task is already implemented:
    1. Arm Tool Position Control (suffix “t”): position and orientation control (6 DOF) of the tool of the manipulator, specified through the transformation eTt. Default value is the identity matrix, therefore the end-effector frame is controlled.

  The relevant Jacobians, Activations and task references can be found in the aforementioned script. These control tasks serve as examples to implement further ones.
- The main TPIK algorithm, with the repetition of the iCAT_task() calls.
- Kinematic integration of the derivatives, collection of data for final plots.