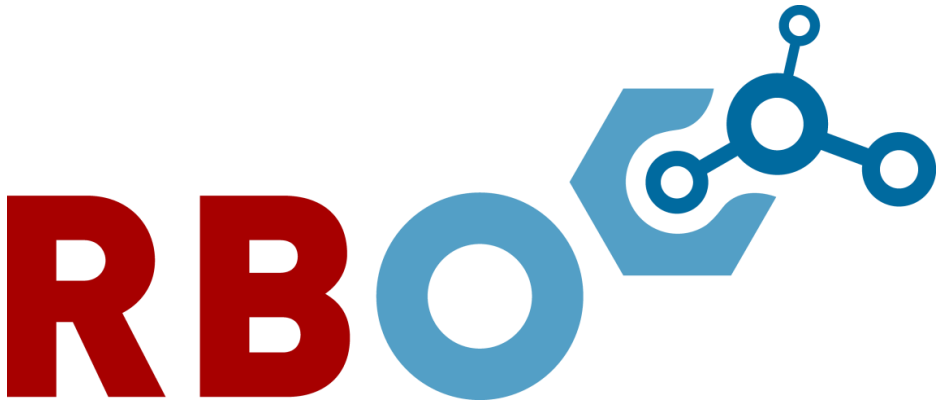


Robotics – Großübung #1

Introduction to the PUMA simulator and programming

Technische Universität Berlin

Organizational stuff

- *Random groups – How to work together*
- *Handing in Assignments – Deadlines are hard deadlines*
- Extra presentation slot
- Key cards

Organizational stuff - Presentations

- Everybody needs to be able to answer 'high level' questions about *ALL* tasks of the assignment.
- For implementation, please specify which team member worked on which (sub) task.
  You can split up implementation of (sub) tasks. But over all, every one needs to contribute equally to the implementation.

The Puma560 Simulator

- Software that simulates the Kinematics, Dynamics, Friction etc. of the Puma560 robot
- *pumasim* binary
- Provides a GUI for controlling, monitoring and configuring the simulation

Running the simulator natively

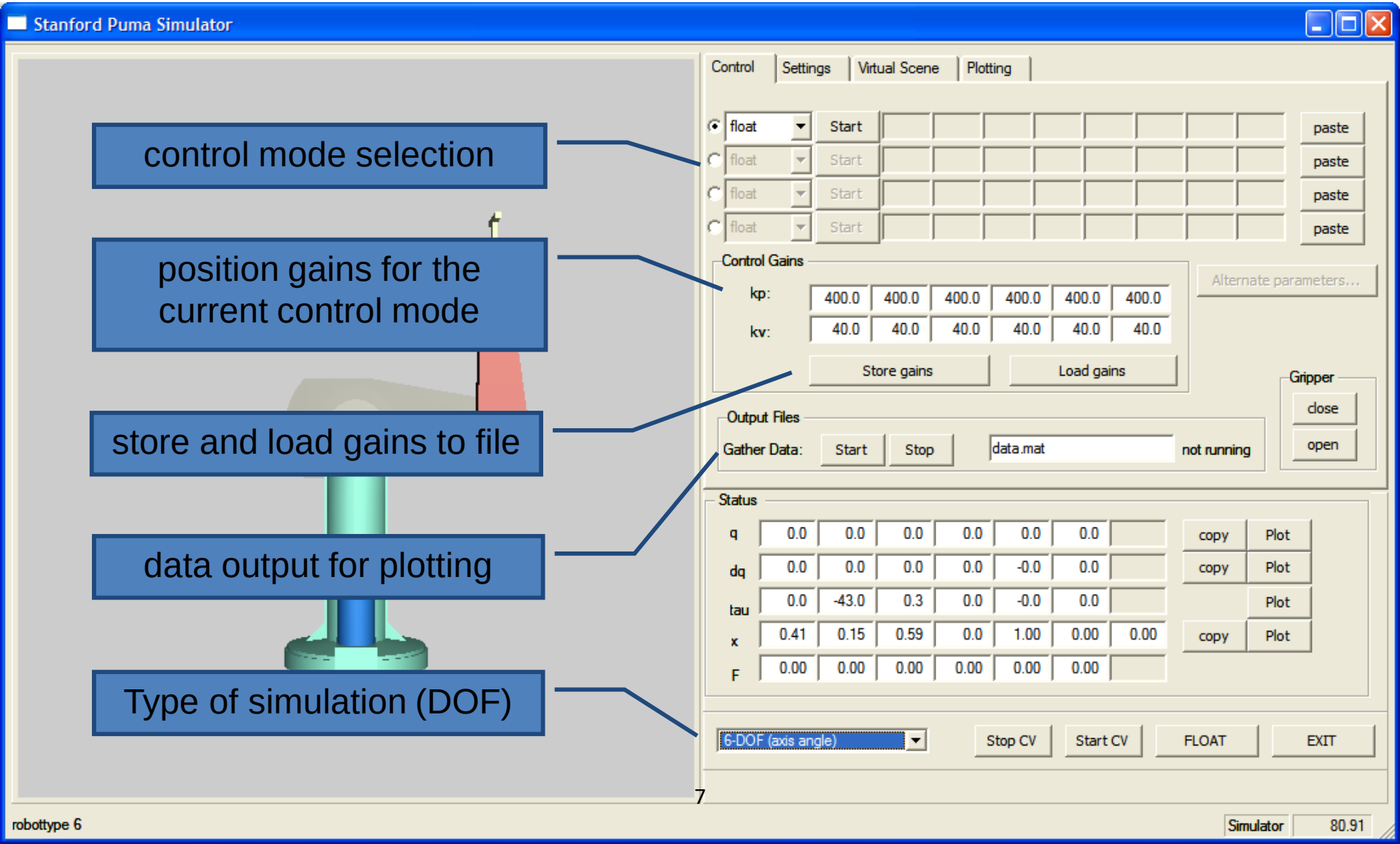
- Known to work on Ubuntu 16.04 (14.04, 15.10)
- Download the pumasimulator-xxxx.tgz

  tar –xzvf pumasimulator-xxxx.tgz
  cd pumasimulator
- Read the *Readme.md* for installation instructions

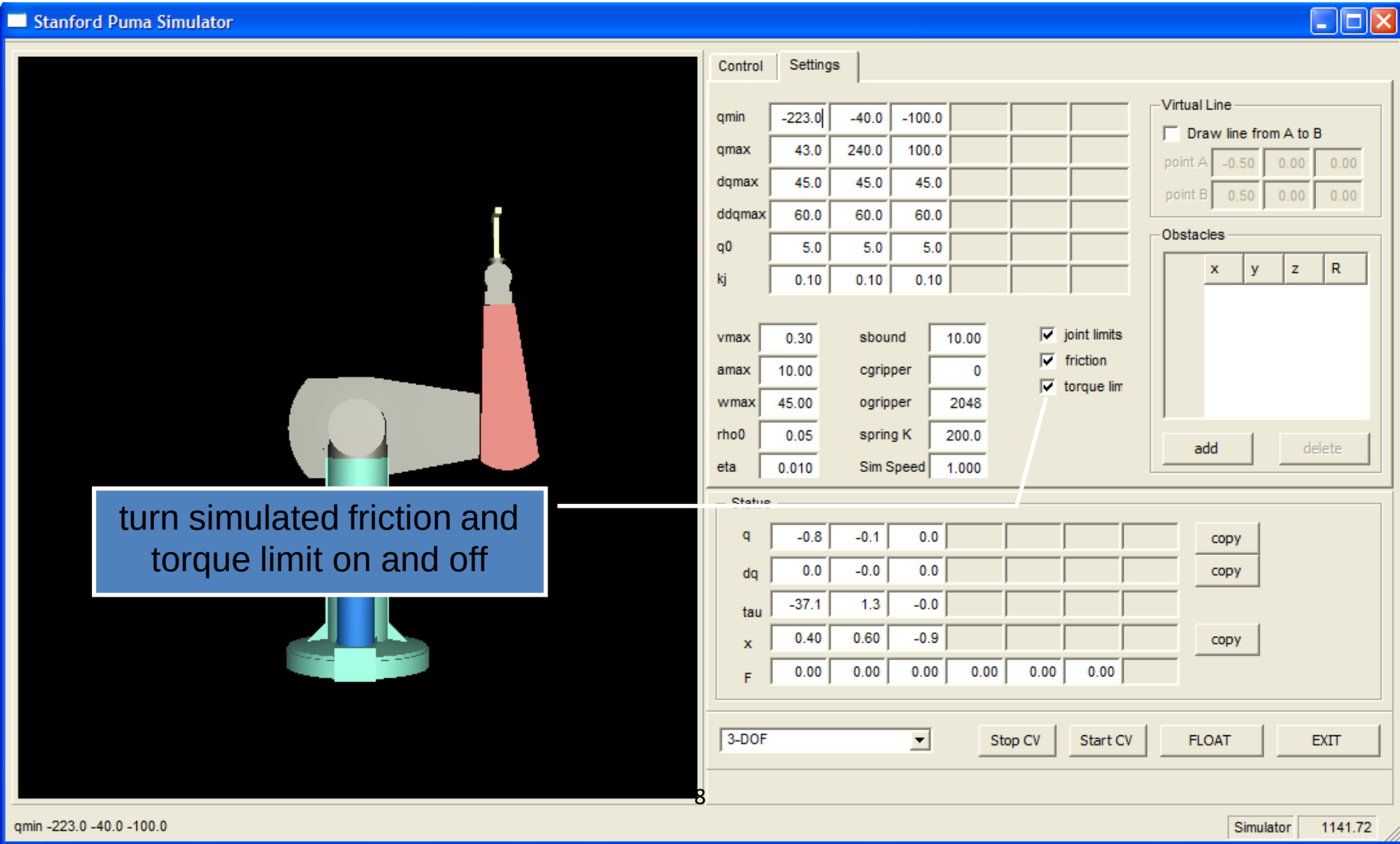
Running the simulator with a VM image

- Install Oracle **Virtualbox 5.2**
  http://virtualbox.org/
- Download the Virtual Machine image (.ova) from ISIS
  - Tip: Do it on the campus net (ca. 2.6GB)
- Start the machine
- Login: student
- Password: student
- Open a terminal and type *pumasim*
- Install *gnuplot* in terminal for live plotting

Puma Simulator



Puma Simulator Settings



Simulator internals

- Controller are called every 2 ms
- Predefined names
- The pumasim executable does not contain controllers, but loads them from the shared library *controlDLL.so*
- Pumasim first looks for the library in the current working directory, then in /opt/pumasim
- You only need to compile controlDLL.so

Compile System for controlDLL

- Cmake based compile template:

```
cd 1/
mkdir build
cd build
cmake ..
make
```

- This creates a controlDLL.so from control.cpp
- To use your controller, call *pumasim* in the *build/* directory:

```
pumasim
```

control.cpp

- Here you implement your robot controller
- **Important:**
  - File must also be compilable on the Real-Time-PC running QNX.
- `init…()` functions are called when you click on „Start“ (controller).
- …`control()` functions are called periodically in the servo loop with 500Hz.
- A lot of global variables are declared via the structure *gv*: they contain the simulator/robot’s stat

Before you start coding

- **Read** *Notes and Restrictions on Coding.pdf* (available on ISIS)

- Information about available math library (vector, matrix, etc.), global variables, etc.

P-controller

- Important variables for the P-controller in the gv struct:

```
tau  : joint torques
q    : joint position
kp   : position gains for the current control mode
qd   : desired joint position
```

- You can tune your controller via the GUI
- You can visualize signals by writing them to a text file (.mat)

data.mat

- Plain ASCII text file
- Each line is a timepoint:

time q(1..n) dq(1..n) qd(1..n) tau(1..n) x(1..m) dx(1..m) xd(1..m)

- n = DOF
- m = 7 in „6-DOF (quaternions)“ mode
  else m = DOF
- You can import it into Excel, Matlab, gnuplot, Octave, matplotlib, a.s.o. and make nice graphs

gains_*.txt

- Text file containing separate gains for all controllers for a specific robot mode
  - gains_1.txt = gains during 3DOF mode
  - gains_6.txt = gains during 6DOF quaternion mode
  - a.s.o.

- Please do not edit the text file manually, but use *Store gains* and *Load gains in the GUI*

Q&A