Week 6

Inverse-Dynamics Tutorial

Kangmin Lee

This tutorial covers inverse-dynamics from Operational Space Control to QP, HQP, and TSID focusing on the math, intuition, and practical design patterns behind them. Through code and simulation, you'll learn to formulate tasks and constraints and compute constraint-satisfying torques suitable for robots.

Installing dependencies

1. **tinyXML2**

```
sudo apt install libtinyxml2-dev
```

2. **Eigen3**

```
sudo apt install libeigen3-dev
```

3. **Pinocchio**

```
sudo apt install ros-$ROS_DISTRO-pinocchio
```

설치 방법 : https://auctus-team.gitlabpages.inria.fr/components/control/qontrol/md_doc_installation.html

Installing Qontrol

1. **Clone the git repository**

```
git clone https://gitlab.inria.fr/auctus-team/components/control/qontrol.git
```

2. **move to Qontrol**

```
cd qontrol && mkdir build && cd build
```

3. **Run the cmake command**

```
cmake ..
```

4. **Build the library**

```
make -j4
```

Launch examples

1. **Go to your build folder**

```
cd build/examples
```

2. **Run example file**

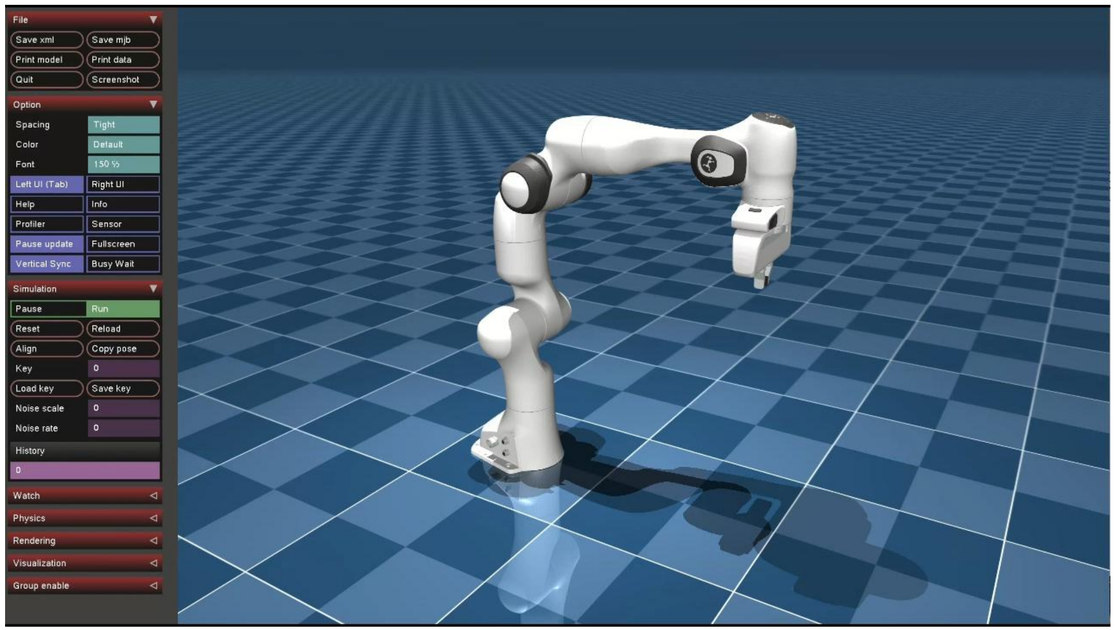
```
./example_name robot_name
```

```
ex1) ./torqueQontrol panda
```

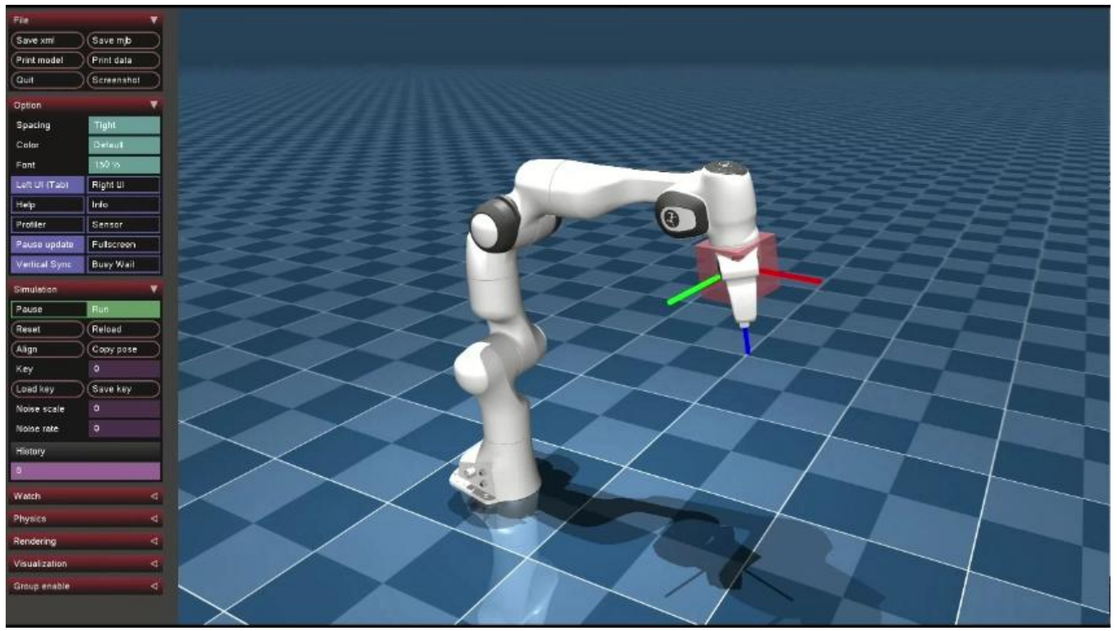
```
ex2) ./torqueQontrol_interactive panda
```

예제 실행 결과

1. 궤도 추적 제어



2. Interactive 제어



- 빨간색 상자를 더블 클릭!!
- Ctrl + 좌/우 클릭으로 상자를 이동/회전

실습할 코드

https://gitlab.inria.fr/auctus-team/components/control/qontrol.git

- 경로 : examples 폴더

1. 궤도 추적 제어 코드 : torqueQontrol.cpp

2. Interactive 제어 코드 : torqueQontrol_interactive.cpp

코드 뜯어보기 (torqueQontrol.cpp)

```cpp
#include "mujoco/mujoco_sim.h"
#include "Qontrol/Qontrol.hpp"
#include "trajectory_generation/trajectory_generation.h"

using namespace Qontrol;

class MujocoQontrol : public MujocoSim
{
public:
//---------------------------------------- simulation ----------------------------
std::shared_ptr<Model::RobotModel<ModelImpl::PINOCCHIO>> model;
std::shared_ptr<JointTorqueProblem> torque_problem;
std::shared_ptr<Task::CartesianAcceleration<ControlOutput::JointTorque>> main_task;
std::shared_ptr<Task::JointTorque<ControlOutput::JointTorque>> regularisation_task;

Qontrol::RobotState robot_state;
TrajectoryGeneration* traj;
std::string resource_path;
```

- model → $M(q)$, $h(q,\dot{q})$, $J(q)$ 같은 로봇 동역학/기구학 계산
- torque_problem → QP형식 문제 정의
- main_task → EE 가속도 태스크 정의
- Regularisation_task → 토크 정규화 태스크 정의
- traj → EE의 목표 궤적 설정

코드 뜯어보기 (torqueQontrol.cpp)

```
34  void initController() override
35  {
36    model =
37        Model::RobotModel<ModelImpl::PINOCCHIO>::loadModelFromFile(resource_path+"robot.urdf");
38
39    const int ndof = model->getNrOfDegreesOfFreedom();
40
41    torque_problem = std::make_shared<Qontrol::JointTorqueProblem>(model);
42    main_task = torque_problem->task_set->add<Task::CartesianAcceleration>("MainTask");
43    regularisation_task = torque_problem->task_set->add<Task::JointTorque>("RegularisationTask",1e-5);
44
45    auto joint_configuration_constraint = torque_problem->constraint_set->add<Constraint::JointConfiguration>("JointConfigurationConstraint");
46    auto joint_velocity_constraint = torque_problem->constraint_set->add<Constraint::JointVelocity>("JointVelocityConstraint");
47    auto joint_torque_constraint = torque_problem->constraint_set->add<Constraint::JointTorque>("JointTorqueConstraint");
48
49    mju_copy(d->qpos, m->key_qpos, m->nu);
50    robot_state.joint_position.resize(ndof);
51    robot_state.joint_velocity.resize(ndof);
52
53    traj = new TrajectoryGeneration(resource_path+"trajectory.csv", m->opt.timestep);
54  }
```

- **Urdf 로딩 + 모델 생성**
- **JointTorqueProblem 생성**
- **태스크 추가**

  Task::CartesianAcceleration("MainTask") → EE의 카르테시안 가속도 $\ddot{x}$를 원하는 값으로 맞추는 태스크

  Task::JointTorque("RegularisationTask", 1e-5) → 토크를 "원하는 토크"에 가깝게 만듦

  > 메인 태스크는 EE 궤적 추종, 정규화 태스크는 토크를 자연스럽게 만드는 역할

코드 뜯어보기 (torqueQontrol.cpp)

```
34  void initController() override
35  {
36    model =
37        Model::RobotModel<ModelImpl::PINOCCHIO>::loadModelFromFile(resource_path+"robot.urdf");
38
39    const int ndof = model->getNrOfDegreesOfFreedom();
40
41    torque_problem = std::make_shared<Qontrol::JointTorqueProblem>(model);
42    main_task = torque_problem->task_set->add<Task::CartesianAcceleration>("MainTask");
43    regularisation_task = torque_problem->task_set->add<Task::JointTorque>("RegularisationTask",1e-5);
44
45    auto joint_configuration_constraint = torque_problem->constraint_set->add<Constraint::JointConfiguration>("JointConfigurationConstraint");
46    auto joint_velocity_constraint = torque_problem->constraint_set->add<Constraint::JointVelocity>("JointVelocityConstraint");
47    auto joint_torque_constraint = torque_problem->constraint_set->add<Constraint::JointTorque>("JointTorqueConstraint");
48
49    mju_copy(d->qpos, m->key_qpos, m->nu);
50    robot_state.joint_position.resize(ndof);
51    robot_state.joint_velocity.resize(ndof);
52
53    traj = new TrajectoryGeneration(resource_path+"trajectory.csv", m->opt.timestep);
54  }
```

- **제약 추가**

  JointConfiguration : 관절각 제한

  JointVelocity : 속도 제한

  JointTorque : 토크 제한

  ➢ 이 값들은 urdf에 적힌 limit을 읽어와서 사용

- **TrajectoryGeneration("trajectory.csv", dt) : csv 파일에서 궤적을 읽음, 미리 주어진 ref 파일**

코드 뜯어보기 (torqueQontrol.cpp)

```
56  void updateController() override
57  {
58    const int ndof = model->getNrOfDegreesOfFreedom();
59
60  for (int i=0; i<ndof ; ++i)
61    {
62      robot_state.joint_position[i] = d->qpos[i];
63      robot_state.joint_velocity[i] = d->qvel[i];
64    }
65    model->setRobotState(robot_state);
```

- 로봇의 상태를 계속 업데이트

```
67    traj->update();
68    pinocchio::SE3 traj_pose(traj->pose.matrix());
69
70    pinocchio::SE3 current_pose(model->getFramePose(model->getTipFrameName()).matrix());
71    const pinocchio::SE3 tipMdes = current_pose.actInv(traj_pose);
72    auto err = pinocchio::log6(tipMdes).toVector();
```

- 말단 pose와 목표 pose 차이를 오차 벡터로 만든다

코드 뜯어보기 (torqueQontrol.cpp)

```
74    Eigen::Matrix<double, 6, 1> p_gains;
75    p_gains << 1000, 1000, 1000, 1000, 1000, 1000;
76
77    Eigen::Matrix<double, 6, 1> d_gains = 2.0 * p_gains.cwiseSqrt();
78    Eigen::Matrix<double, 6, 1> xdd_star =
79        p_gains.cwiseProduct(err) +
80        d_gains.cwiseProduct(traj->velocity - model->getFrameVelocity(model->getTipFrameName())) + traj->acceleration;
```

- 원하는 가속도 시스템을 설계

$$\ddot{x}^* = K_p e + K_d \left(\dot{x}_{\text{des}} - \dot{x}\right) + \ddot{x}_{\text{des}}$$

```
82    main_task->setTargetAcceleration(xdd_star);
```

- TSID/QP의 카르테시안 가속도 태스크에 $\ddot{x} \approx \ddot{x}^*$ 가 되도록 문제를 세팅

코드 뜯어보기 (torqueQontrol.cpp)

```
87      torque_problem->update(m->opt.timestep);
88
89      if (torque_problem->solutionFound())
90      {
91        sendJointTorque(torque_problem->getJointTorqueCommand());
92      }
93    }
94    };
```

- **QP 풀기 + MuJoCo로 토크 전송**

코드 뜯어보기 (torqueQontrol_interactive.cpp)

- 나머지 제약과 제어 부분은 동일
- 목표를 설정하는 부분만 다름

```
84      Eigen::Matrix<double, 6, 1> xdd_star =
85          p_gains.cwiseProduct(err) +
86          d_gains.cwiseProduct(- model->getFrameVelocity(model->getTipFrameName())) ;
```

$$\ddot{x}^* = K_p e - K_d \dot{x}$$

- 동일한 일반 식의 특수한 케이스일 뿐

$$\ddot{x}^* = K_p e + K_d \left(\dot{x}_{\text{des}} - \dot{x} \right) + \ddot{x}_{\text{des}}$$

$$\dot{x}_{\text{des}} = 0$$

$$\ddot{x}_{\text{des}} = 0$$

- 궤적이 주어졌을 때에는 매 시간마다 어떻게 움직여야 하는지 위치, 속도, 가속도 값이 전부 주어짐
- Interactive의 경우 현재 박스의 위치로 이동한 후 '정지'에 해당 > 따라서 속도, 가속도 값이 0

Thank you