DEGREE PROJECT IN ELECTRICAL ENGINEERING, SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2016

Simulation in the Control Loop

Control and Collision Detection for Collaborative Robots

MARTIN TÖRNQVIST

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING

Abstract

Control and simulation are two areas heavily utilized in robotics research, and simulation is often used as a way to test and optimize control algorithms. Some controllers are even able to perform simulations, as is the case with ABB's virtual controller. Despite many similarities in the dynamics of model based controllers and physics simulation, using a simulation to perform dynamics calculations for controllers remains relatively uncharted territory. This report presents a general simulation-based controller that integrates simulation into every part of the control loop: from motion planning to model-based control and collision detection. The controller is implemented and tested on the ABB Yumi collaborative robot.

Contents

1 Introduction **5**
- 1.1 Algoryx Simulation . . . 5
- 1.2 Problem Statement . . . 5

2 Conventions **6**

3 Background **7**
- 3.1 Previous Work . . . 7
- 3.2 Co-Robotics . . . 7
- 3.3 Physics Simulations . . . 7
  - 3.3.1 The Spook Solver . . . 9
- 3.4 Model-Based Control (MBC) . . . 9
  - 3.4.1 Computed Torque Control . . . 10
- 3.5 Motion Planning Methods . . . 10
  - 3.5.1 Off-line and On-line Planning . . . 11
  - 3.5.2 Path and Trajectory Planning . . . 11

4 Theory: Simulation In-The-Loop **12**
- 4.1 Computing the Force . . . 13
- 4.2 Path Planning for Robot Manipulators . . . 13
- 4.3 Collision Detection . . . 13

5 Implementation **13**
- 5.1 The Functional Mock-up Interface (FMI) . . . 14
- 5.2 FMUs . . . 14
  - 5.2.1 Motion Planner . . . 14
  - 5.2.2 Control simulation . . . 16
  - 5.2.3 Robot . . . 17
- 5.3 The Algorithm: Putting it All Together . . . 19
- 5.4 Robot Models . . . 20
  - 5.4.1 Simple Sphere . . . 20
  - 5.4.2 ABB Yumi . . . 20

6 Results **22**
- 6.1 Test Scenarios . . . 22
  - 6.1.1 Step Test . . . 23
  - 6.1.2 Offline Step Test . . . 23
  - 6.1.3 Collisions . . . 23
  - 6.1.4 Hand Guiding . . . 23
- 6.2 Graphs . . . 23

7 Conclusion **24**
- 7.1 Result Discussion . . . 24

7.2 Limitations 25
7.3 Future Work 25

8 Acknowledgements **26**

List of Figures

1 A SCARA robot showing the joint positions and end effector [7]. 6
2 The two types of constraints used as joints in robot simulations [12]. . . . 8
3 Block diagram of computed torque control [21]. 11
4 A simple schematic showing the different modules used in the controller. u refers to forces and q to positions. 12
5 Schematic view of off-line planning. 15
6 Schematic view of on-line planning. 16
7 A visualized representation of a 1 degree of freedom robot perform a single step of the algorithm in section 5.3. 18
8 A simple sphere simulation. 20
9 ABB Yumi [31]. 22

1 Introduction

This report aims at bringing two areas in robotics closer together: simulation and control. While simulation have long been used to try different control techniques, incorporating a complete physics simulation in the actual control loop is a fairly new concept. The reason this is an interesting subject, is the fact that constraint-based multibody simulations solve the same dynamic problem as model-based controllers do.

There has been some research on using simulation in the loop [1]–[3], but the focus tends to be on automatically deriving controllers, rather than running the simulation as part of the control paradigm. Another close subject are the virtual controllers employed by ABB, that utilize the fact that the controller solves the dynamic problem to create simulations using the same code that controls the robot [4]. While this obviously works for developing successful controllers, it does not allow for the robot to interact dynamically with its environment.

This report is divided into a few sections. Section 2 defines some terms and concepts used throughout the report; section 3 gives some background information on physics simulations and model-based control; section 4 describes the thoughts behind the simulation in the loop-approach, and presents a mathematical foundation; section 5 explains how the controller was implemented in this project, and how it was verified using a few models and test scenarios; section 6 presents the results that was found when running the tests; and finally, section 7 concludes the report and gives some thoughts on how the work can be continued in the future.

1.1 Algoryx Simulation

The project was conducted at Algoryx Simulation AB in Umeå, Sweden; a physics simulation company that was founded in 2007 as a spin-out from Umeå University. The company has about 20 employees, and their products include the AgX Dynamics physics engine, the Algodoo 2D educational engine, and Dynamics for SpaceClaim which integrates CAD-modelling with real-time dynamics. Their physics engine AgX Dynamics is utilizing the Spook method [5] as its solver.

1.2 Problem Statement

The goal of the project is to develop a generalized tool for simulating human-robot interaction, including a model-based controller, a motion planner and the possibility of simulating a wide variety of collisions. The tool will be built using the AgX Dynamics physics engine, and will be fully modular, where the controller, the path planner and the simulated robot are all easily replaced and improved upon. The tool should be general and stable enough to allow the robot simulation to be replaced by a real robot, although implementing this is beyond the scope of this thesis.

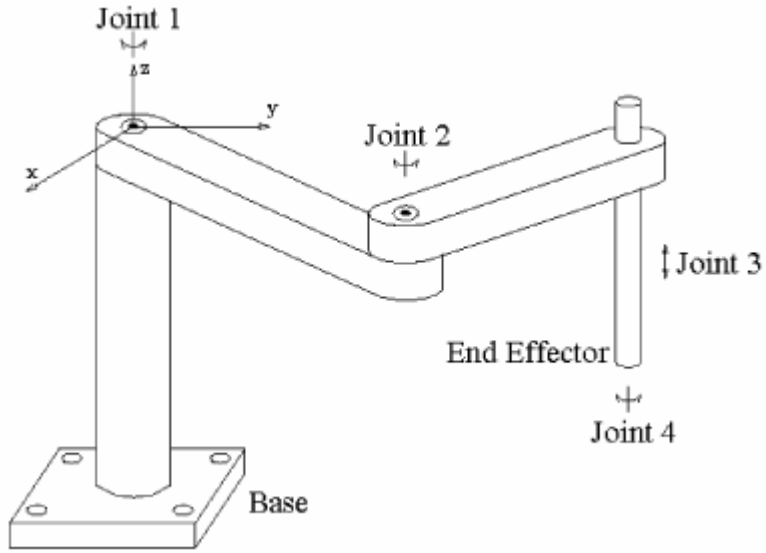


Figure 1: A SCARA robot showing the joint positions and end effector [7].

2 Conventions

To avoid ambiguity and unnecessary repetition, below is a list of definitions used in this report. These definitions will override any meaning the words may originally have held, unless explicitly stated otherwise.

Robot arm A kinematic chain of links connected by joints. Each joint allows movement in a single degree of freedom, and can be rotational or translational (see figure 1).

Joint Space The set of joint positions used to describe the state of the robot arm, usually described as $q = [q_1, q_2, \ldots, q_n]^T$.

Configuration Space The set of configurations that are reachable by the robot arm.

End Effector The tool attached to the tip of the last link in the kinematic chain (see figure 1). Some common attachments are grippers, spray nozzles and welders [6]. The position of the end effector is sometimes called the tool center point (TCP).

Rotational and Translational Characteristics When describing joint characteristics such as positions, velocities and torques, rotational and translational characteristics are treated in the exact same way. These characteristics will therefore collectively be named by their translational equivalent, i.e. position refers to angular or translational position, velocity refers to angular or translational velocity, and force refers to torque or translational force.

3 Background

This section will give some background to the different topics covered later in the report: physics simulations in general and the Spook solver in particular; model-based control; and motion planning. Furthermore, it gives a quick overview of the state of the research in the area of simulation in the loop control, and a short introduction to co-robotics.

3.1 Previous Work

There has been some research on using simulation in the loop [1]–[3], but the focus tends to be on automatically deriving controllers, rather than running the simulation as part of the control paradigm. Another close subject are the virtual controllers employed by ABB, that utilize the fact that the controller solves the dynamic problem to create simulations using the same code that controls the robot [4]. While this obviously works for developing successful controllers, it does not allow for the robot to interact dynamically with its environment.

3.2 Co-Robotics

Co-robotics is the field in robotics dealing with collaborative, co-operative and compliant robots. What characterizes a collaborative robot is the fact that it's designed to collaborate either with humans, or co-operate with other robots. This sets high standards for safety and adaptability, as a robot must never hurt a human being [8], while still being able to solve complex tasks. One solution to this problem is to build compliant robots, which are light and weak enough that they won't be able to cause much pain, even if they were to try.

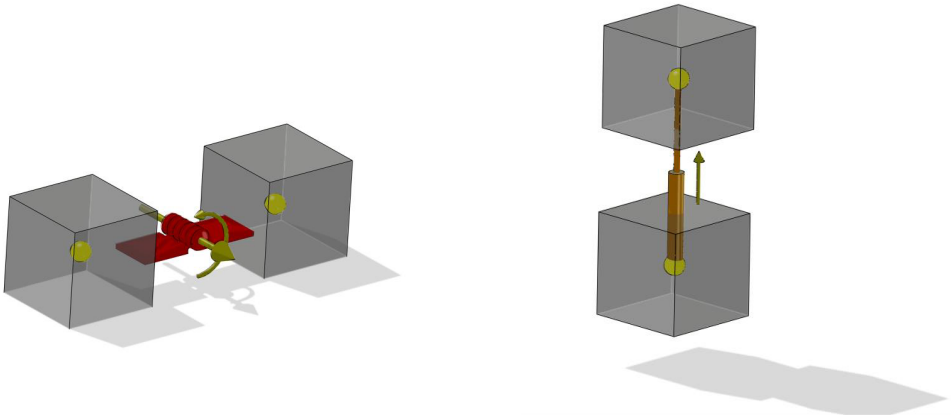
Nonetheless, collision detection is another important part of co-robotic design, as a collision detected early may avoid harm both to the human, and, more likely, to the robot.

3.3 Physics Simulations

Physics simulations are conducted by physics engines, which can be used to simulate various physical systems. Physics engines are typically classified into two categories: real-time and scientific. Although there is some overlap, real-time engines aim at simulating a system in real-time, for use in e.g. games and training simulators, while scientific engines value accuracy and precision over speed.

In robotics, the main goal of simulating is to solve the forward dynamics problem: determining the motion of bodies from the forces acting upon them [9]. This can be done by solving a set of differential equations [10]:

$$M(q)\ddot{q} + c(q, \dot{q}) = f(q) + G\lambda \tag{3.1}$$



(a) A hinge constraint.

(b) A prismatic constraint.

Figure 2: The two types of constraints used as joints in robot simulations [12].

where

q : $n \times 1$ vector of generalized coordinates[1]
$M(q)$: $n \times n$ inertia matrix containing masses and moments of inertia of the different bodies
$c(q, \dot{q})$: $n \times 1$ vector containing inertial forces that are nonlinear functions of the velocities (e.g. centrifugal and coriolis forces)
$f(q)$: $n \times 1$ vector containing external forces (e.g. gravity)
λ : $k \times 1$ vector containing constraint forces
G : $n \times k$ Jacobian matrix where each column represents the direction of a constraint (see below).

Solving this is the goal of most physics engines, and there have been countless hours of research devoted to it. The system is modeled as a set of bodies, which are attached to eachother or the world with constraints. A constraint is a restriction on the movement of a certain body, and an ideal constraint completely removes one or more degrees of freedom (DOFs) from a body, allowing the object to move only in the remaining DOF. In 3D space there are 6 DOFs, 3 translational and 3 rotational. Constraining e.g. all the translational DOFs would create a ball joint, allowing free rotation around a fixed point. In robotics, most movement can be described by two types of joints, each removing all but one degree of freedom: the hinge and the prismatic constraint. The hinge allows rotation around a single axis, while the prismatic allows movement in a straight line (see figure 2). The SCARA robot in figure 1 has 3 hinges and 1 prismatic joint.

The force at which a constraint is enforced is determined by its violation: the displacement from a certain state of rest. This can be compared to a simple mass-spring-damping system, $F = -kx - c\dot{x}$, where k is the spring constant, c is the damping coefficient and x is the displacement [13].

[1] A set of coordinates that describe the system fully, e.g. joint positions for a robot manipulator [11].

3.3.1 The Spook Solver

This project uses the AgX Dynamics physics engine, developed by Algoryx Simulation AB. It uses the Spook solver described in [5] to solve and discretize the dynamics problem (3.1). The resulting velocities when applying the constraint forces λ_k to a system is given by

$$v_k = v_{k-1} + M^{-1}{G_k}^T \lambda_k + hM^{-1} f_e \tag{3.2}$$

where

v_k : vector of joint velocities at time step k
M : mass matrix containing masses and moment of inertias
G_k : jacobian matrix containing information about the constraints
f_e : vector containing external forces
h : the time step

.

Correspondingly, when forcing a constraint to a certain velocity, the resulting force applied to the constraint is given by

$$\lambda_k = \left({G_k}^T M^{-1} G_k + \Sigma\right)^{-1} G_k \left(v_k - v_{k-1}\right) \tag{3.3}$$

where

Σ : regularization matrix

.

These equations can be derived from (3.1) [14], but this is non-trivial and beyond the scope of this thesis. Further reading is available in [5].

3.4 Model-Based Control (MBC)

Model-based control (MBC) is a collective name given to control techniques where the controller takes into account an explicit description of the system to be controlled. For complex, multivariate systems this is often superior to using regular PID control, since effects such as gravity and inertia can be compensated for directly and are not dependent on the robustness of the PID [15]. An important special case of MBC is model predictive control (MPC), where the controller takes into account not only the current configuration of the system, but predicted future ones as well [16]. This is common in large process industries and power plants [17].

There is lots of literature on how to apply MBC on a robot arm, including [18], [19] and [20]. I will focus on a method known as computed torque control, which was first described in [21].

3.4.1 Computed Torque Control

Computed torque control is a type of motion-based control, which is built on the dynamic model of a robot arm

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \tag{3.4}$$

where

$$\begin{aligned} q &: \text{vector of joint positions} \\ M(q) &: \text{inertia matrix for the configuration given by q} \\ C(q,\dot{q}) &: \text{matrix containing centrifugal and Coriolis forces} \\ g(q) &: \text{vector containing gravitational forces} \\ \tau &: \text{vector containing external forces applied by actuators} \end{aligned}$$

and the goal becomes to compute the external forces τ needed to get to a desired state $[q_d, \dot{q}_d, \ddot{q}]^T$. Comparing this equation with equation (3.1), one realizes that by modelling the actuator forces as a single degree of freedom constraint (hinge or prismatic)[1], the models become identical. This indicates that the computed torque control, and a physics simulation, solve similar problems. This is the basic idea behind this thesis, and section 4 will discuss how simulations and controllers can be incorporated into a single control loop.

The control law for computed torque control is given by

$$\tau = M(q)\left[\ddot{q}_d + K_v\dot{\tilde{q}} + K_p\tilde{q}\right] + C(q,\dot{q})\dot{q} + g(q) \tag{3.5}$$

where

$$q_d : \text{vector of desired joint desired positions}$$

and the position error $\tilde{q} = q_d - q$, which yields the following closed-loop equation:

$$\tau = M(q_d - \tilde{q})K_p\tilde{q} + M(q_d - \tilde{q})K_v\dot{\tilde{q}} + M(q)\ddot{q}_d + C(q,\dot{q})\dot{q} + g(q) \tag{3.6}$$

. The loop is further visualized in figure 3.

3.5 Motion Planning Methods

Motion planning is the process of generating a set of motions that will take the robot from one state to another, while avoiding collisions and staying within its configuration space [22]. This can be accomplished in a wide variety of ways, including using potential fields [23], inverse kinematics, probabilistic methods [24] and much more. What method to use depends on many factors, including the type of robot to be controlled (autonomous ground vehicle, robot arm, drone, etc.), its application (does it need to interact with its

[1] see section 3.3

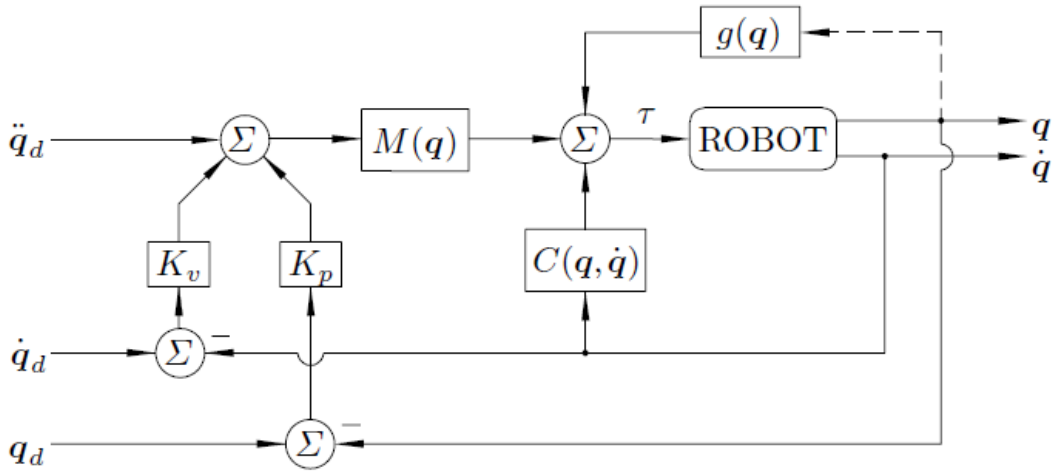


Figure 3: Block diagram of computed torque control [21].

environment? Is it subject to much noise? Does it need to avoid objects?) and what the controller looks like (position based, velocity based, PID, MBC[1], feed forward part). These methods can be classified into various categories, and this report will focus on two of those: off-line vs on-line, and position vs velocity. Further reading about motion planning in general is available in e.g. [25] and [26].

3.5.1 Off-line and On-line Planning

The main difference between off-line and on-line planning is during what phase the planning takes place. Off-line planners compute the complete trajectory before the motion is started, while on-line planners plan the route during movement. This gives on-line planners the ability to adapt the trajectory to new information, such as unforeseen obstacles, something which is generally desirable. On the other hand, as on-line planning requires continually solving the motion planning problem, it is much more computationally demanding, and finding an optimal route is often impossible [22].

The differences between the two strategies has led to them being utilized for different applications. Off-line planning is generally used for repeatable tasks in a static environment, such as mounting and some pick-and-place applications. On the other hand, on-line planners are used when the environment is dynamic, or where the goal is not known beforehand [27].

3.5.2 Path and Trajectory Planning

While it is sometimes sufficient only to specify the desired end-point of a motion, or a limited set of points ensuring a collision-free path, when dealing with robots it is often

[1]Model-Based Control, see section 3.4

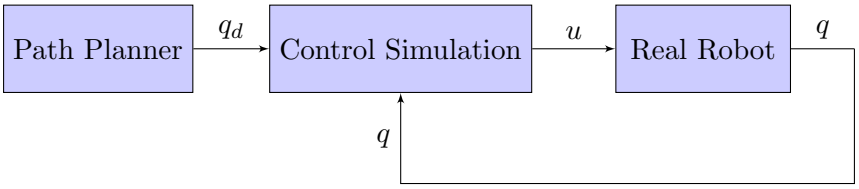


Figure 4: A simple schematic showing the different modules used in the controller. u refers to forces and q to positions.

desirable to specify at which velocities this path should be followed, to have complete control over the motion. This is especially important when the robot is made up of a dynamic chain, such as a multiple degrees of freedom robot arm (such as the SCARA robot in figure 1. An example of a controller that utilizes a complete trajectory is the computed torque controller described in 3.4.1 (see equation (3.5) in particular). Planning the velocity may also be important when avoiding moving obstacles [28].

A planner providing both position and velocity information is sometimes called a trajectory planner, while its simpler counterpart dealing only with position is known as a path planner. In discrete applications, a trajectory planner may be realized as a path planner with positions given at each time step. These positions can then be used to calculate velocity as well by interpolation, i.e.

$$v_k = \frac{q_k - q_{k-1}}{h} \tag{3.7}$$

.

4 Theory: Simulation In-The-Loop

This section handles the theoretical description and reasoning behind the simulation in-the-loop approach. The idea is to run a real-time simulation alongside the actual system to be controlled, where the simulation will continually exchange information with its real world counterpart. Using this in a controller will allow the controller to take into account complex dynamics of the system, along with any environment aspects that the simulation have access to, such as external forces (e.g. gravity) and objects to be gripped or avoided. By comparing output from the real system with the simulated one, deviations, such as collisions, can be detected and reacted to. This is discussed in more detail in section 4.3.

In this report, the systems to be controlled will be limited to single- or multi-joint kinematic chains (e.g. robot arms), where the controlled variable will be the forces of the motors, and the feedback will be the positions (see fig. 4).

The system to be controlled will be referred to as the "real robot", the simulation to be used for control will be referred to as the "control simulation", and the model of the real-world system as the "control robot".

4.1 Computing the Force

As stated in section 3.3.1, the Spook solver calculates forces to counteract any violation in speed and position specified by the constraints. By specifying these violations directly, the solver can therefore be forced to calculate the forces required to reach a certain position or velocity. By using the mass matrix M of the system and the jacobian G of the constraints, the force λ can be calculated from the velocity using (3.3). A short description of the solver is presented in section 3.3.1, and for the interested reader a detailed description can be found in [5].

4.2 Path Planning for Robot Manipulators

Path planning for robot arms involves the process of converting the trajectory of the end-effector to a set of joint angles that the controller can handle. This can be done in a variety of ways, where inverse kinematics is probably the most well-known approach. Inverse kinematics uses the kinematic model of the robot, including lengths of the links and axes of rotations to recursively calculate the joint angles, given a position in space of the tip of the last link (in comparison, forward kinematics is used to calculate the position of the end effector from the specified joint angles). This, however, often requires finding the jacobian matrix of the system and calculating its inverse for every time step, something that can be difficult to achieve in real-time for many degrees of freedom (more than 6) [29].

The method used in this approach is in some ways much simpler: Use the same model as in the control simulation to force the end effector to the desired position, and read the values of the joint angles required to get there. This guarantees a physically viable route, even though it is in no way optimized.

4.3 Collision Detection

Collision detection is the process of getting the robot to realize if it has collided with another object. While this is a simple process for humans that have access to millions of pressure sensors in the skin, a robot that has a joint position sensor as its only input will find this problem less than trivial.

The approach attempted in this project utilizes the difference in expected and actual movement to predict whether or not an unforeseen collision was experienced.

5 Implementation

To implement the controller described in section 4, an algorithm was devised. The algorithm breaks down the method into a few steps, which are repeated each time step, and it is described in section 5.3. To realize the algorithm, however, the problem of how to simultaneously run several simulations and exchange information between them had to be solved. The solution was the Functional Mock-up Interface (FMI) standard, which is described in section 5.1

5.1 The Functional Mock-up Interface (FMI)

The Functional Mock-up Interface is a tool for connecting simulations, controllers and other functions. Each module, called a Functional Mock-up Unit (FMU), has a set of inputs and outputs, where the output of one FMU can be bound to the input of another. Such a pair of input-output is called a binding. A separate file, called a master, is used to specify when the different events are to occur, e.g. in what order the simulations should step, and when the bindings should exchange information. A detailed description of the FMI standard can be found in [30].

5.2 FMUs

There are four FMUs present in this solution: The path planner, the control simulation, the [robot simulation] and the controller. In this implementation, the control simulation and the robot are both simulations; the controller is a simple interface; and the path planner have been implemented in two different ways.

5.2.1 Motion Planner

The motion planner is in charge of providing the controller with joint positions at each time step. These positions can be calculated either before any motion has begun (off-line planning), or during motion (on-line planning). Advantages and disadvantages of these strategies are described in section 3.5.1. Below is a description of how these planning strategies were implemented in this project.

The motion planners control the trajectory of the simulation by moving the end effector. This movement is generated in such a way that it accelerates and decelerates by a constant acceleration for the same amount of time. Furthermore a maximum velocity can be set, at which point the velocity will remain the same until it is time to decelerate. This is realized using algorithm 1

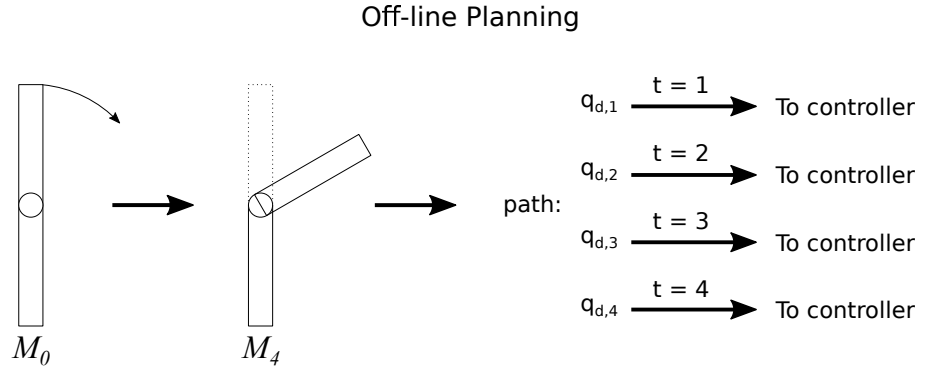


Figure 5: Schematic view of off-line planning.

Algorithm 1: Moving the end effector

Input: Desired end position vector $\boldsymbol{p_d}$
Data: Current velocity vector $\boldsymbol{v_0}$
Current position vector $\boldsymbol{p_0}$
Acceleration a
Maximum velocity v_{max}
Time step h
Result: Next velocity vector $\boldsymbol{v_1}$
$\tilde{\boldsymbol{p}} = \boldsymbol{p_d} - \boldsymbol{p_0}$
$d_1 = \|\tilde{\boldsymbol{p}}\|$ `/* Distance between current position and desired */`
$d_2 = \frac{\boldsymbol{v_0}^2}{2*a}$ `/* Distance required to reach 0 velocity */`
if $d_1 \leq d_2$ **then**
  $\boldsymbol{v_1} \leftarrow \tilde{\boldsymbol{p}}(\boldsymbol{v_0} - a \cdot h)$ `/* Decelerate */`
else
  if $\boldsymbol{v_0} \leq v_{max}$ **then**
    $\boldsymbol{v_1} \leftarrow \tilde{\boldsymbol{p}}(\boldsymbol{v_0} + a \cdot h)$ `/* Accelerate */`
  else
    $\boldsymbol{v_1} \leftarrow \boldsymbol{v_0}$ `/* Hold the same speed */`
  end
end
return $\boldsymbol{v_1}$

Off-line Planning The off-line planner utilizes the same model as the control simulation, and generates its trajectory before sending any information to the controller. The motion is generated by forcing the end effector to move in a certain trajectory, and storing the positions of the joints at each time step in a list $Q = [q_{d,1}, q_{d,2}, \ldots, q_{d,k}, \ldots, q_{d,n}]$. When the controller asks for a new set of joint positions, the motion planner simply pops the next value in the list and hands it over (see figure 5).

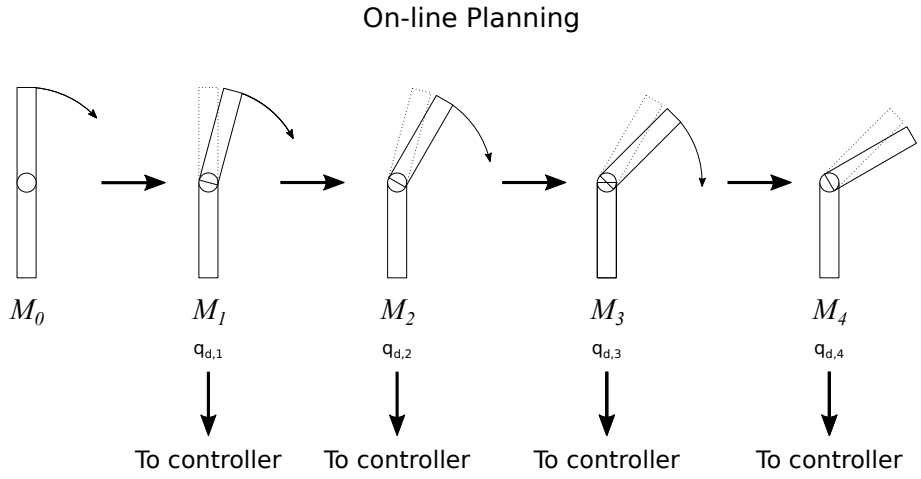


Figure 6: Schematic view of on-line planning.

On-line Planning: Planning-In-The-Loop The on-line planner is realized by running the motion planning simulation simultaneously with the controller, and sending its position at each time step (see figure 6). By synchronizing its state with the control simulation

5.2.2 Control simulation

Control simulation state at timestep k: $\mathcal{C}_k$

The control simulation holds the model of the real robot to be controlled, and its mission is to compute the torque required to move from the true robot's current state, to the next state as presented by the motion planner. The model should be as detailed as possible, and include any obstacles, loads and dynamics present in the actual situation.

Moving from one state to the next is realized simply by setting the motors to the desired velocity, stepping the simulation and reading the torques (see algorithm 2). There is, however, no guarantee that the actual robot will respond to the torques in the exact same way as the simulated one, and a difference in position $\Delta q_k = q_k^r - q_k^c$ will be present for each time step k. Unless handled, this Δq_k will accumulate, and eventually result in instability, as the control robot will calculate torques with a completely different state as its starting point.

To deal with this problem, a feedback part must be added to the controller. This feedback consists of two steps:

1. Roll back the entire state of the simulation one step, from $\mathcal{C}_k$ to $\mathcal{C}_{k-1}$
2. Perform a step according to algorithm 2, using the measured positions of the real robot joints, q_k^r, as its input, resulting in a new state $\mathcal{C}'_k$.

Using $\mathcal{C}'_k$ as the starting point for the next step will ensure that errors will not accumulate, and result in a more stable controller. The process is described in section 5.3 and

visualized in figure 7

Algorithm 2: controlSim.step() function

Input: Desired position for each joint q_d
Data: Current position for each joint q_0
Time step h
Result: Torques applied by the actuators τ
for $i \leftarrow 1$ **to** *number of joints* **do**
  $v_d = \frac{q_{d,i} - q_{0,i}}{h}$
  actuators[i].setVelocity(v_d);
end
Update dynamics according to (3.3)
for $i \leftarrow 1$ **to** *number of joints* **do**
  $\tau_i \leftarrow$ **actuators**[i].getForce()
end
return τ

5.2.3 Robot

The robot FMU holds information about the robot itself. Ideally this would be a real robot, being fed forces from the controller, and returning position measurements. In this project, however, the robot will consist of another model, identical to the one used in the control simulation. An algorithmic description of the step is shown in algorithm 3.

For testing controller robustness, noise was added to the measurement output, and the mass of the links was altered slightly (see section 6.1).

Algorithm 3: realSim.step() function

Input: Forces to be applied to the motors τ
Result: Positions of the motors after updating the dynamics q
for $i \leftarrow 1$ **to** *number of joints* **do**
  actuators[i].setTorque(τ_i);
end
Update dynamics according to
for $i \leftarrow 1$ **to** *number of joints* **do**
  $q_i \leftarrow$ **actuators**[i].getPosition()
end
return q

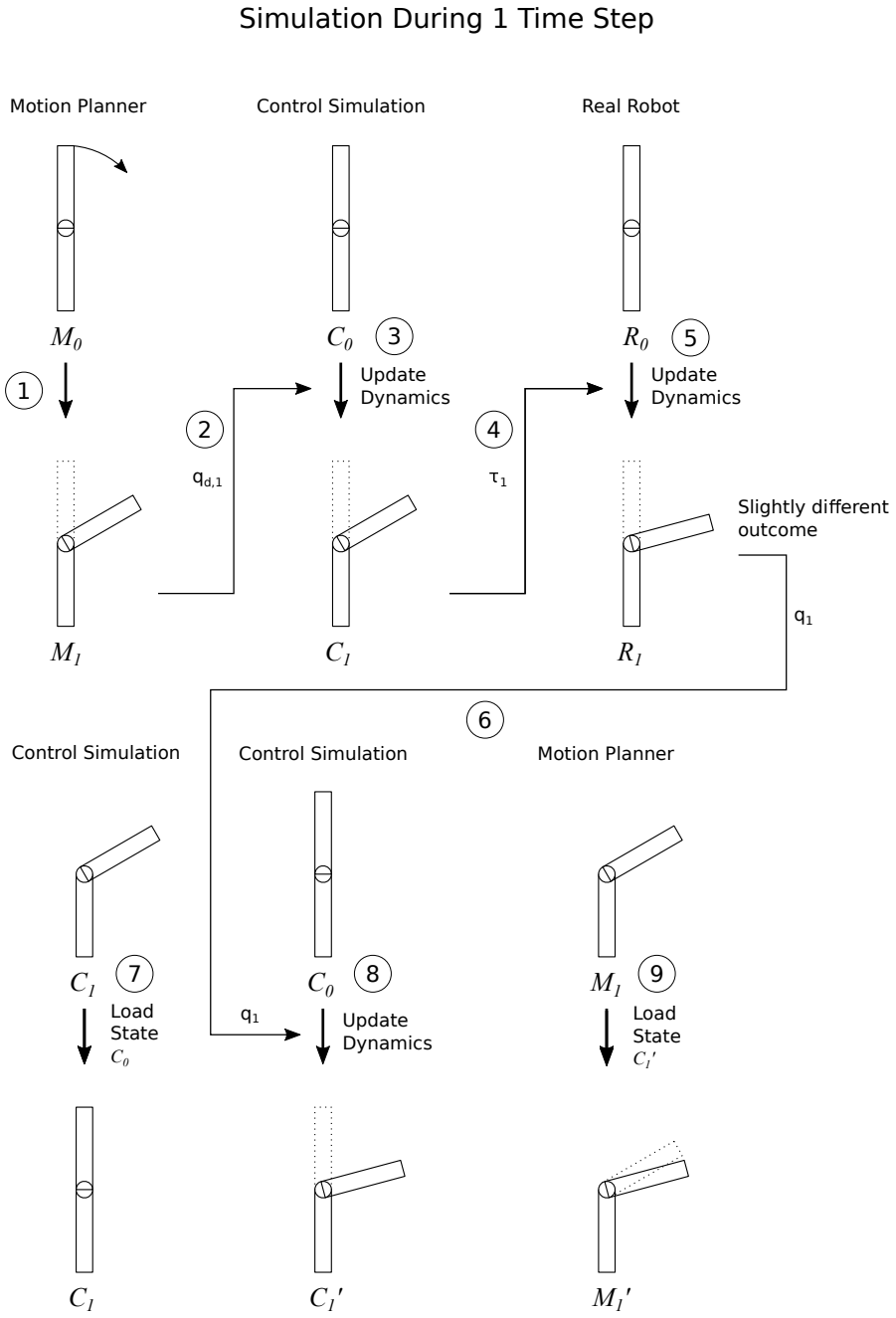


Figure 7: A visualized representation of a 1 degree of freedom robot perform a single step of the algorithm in section 5.3.

5.3 The Algorithm: Putting it All Together

Combining the FMUs described above yields the complete controller algorithm. This algorithm can be described as a set of steps that are repeated each time step:

1. (For on-line planning only) Step the motion planner simulation and use the joint position q_d as the next desired position vector.
2. Transfer the desired position vector to the control simulation.
3. Step the control simulation with the desired position vector q_d according to algorithm 2.
4. Transfer the vector of forces τ to the real robot.
5. Apply the forces to the real robot actuators.
6. Transfer the resulting position vector q to the control simulation.
7. Step back the control simulation to its starting state.
8. Step the control simulation with the real robot position vector q according to algorithm 2.
9. (For on-line planning only) Load the control simulations state into the motion planner.

.

Utilizing equations (3.7), (3.3) and (3.2), the algorithm can also be described in mathematical terms:

$$v_k^c = \frac{q_k^m - q_{k-1}^c}{h} \tag{5.1}$$

$$\lambda_k^c = \left(G_k^{cT} M^{-1} G_k^c + \Sigma \right)^{-1} G_k^c \left(v_k^c - v_{k-1}^c \right) \tag{5.2}$$

$$v_k^r = v_{k-1}^r + M^{-1} G_k^{rT} \lambda_k^c + h M^{-1} f_e \tag{5.3}$$

$$q_k^r = q_{k-1}^r + h v_k^r \tag{5.4}$$

$$x_{k,rollback}^c = x_k^c \tag{5.5}$$

$$v_{k,rollback}^c = \frac{q_k^r - q_{k-1}^c}{h} \tag{5.6}$$

$$\lambda^c_{k,rollback} = \left({G^c_k}^T M^{-1} G^c_k + \Sigma \right)^{-1} G^c_k \left(v^c_{k,rollback} - v^c_{k-1} \right) \tag{5.7}$$

, where the upper index describes which model is referred to: c is the control model, m is the motion planner model, and r is the real robot model. The lower index refers to discrete time. Below is a table describing the various equations, and how they relate to the algorithm:

5.4 Robot Models

One of the beauties of this type of controller is its ability to be implemented on many different kinds of applications, as long as you have its physical model. The simulations will be in the presence of a uniform gravitational field in the negative z-direction, with a constant acceleration of 9.82 m/s^2.

5.4.1 Simple Sphere

One of the most basic models to control is that of a sphere attached to a translational axis. In this scenario, the sphere is attached to the z-axis. The specifications of the sphere used in this report are:

Mass : 1 kg
Radius : 0.1 m

The motion planner is programmed to accelerate and decelerate with 0.1 m/s^2, and reach a maximum velocity of 1 m/s.

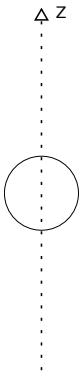


Figure 8: A simple sphere simulation.

5.4.2 ABB Yumi

Yumi is ABB's first collaborative robot, and features 2 arms with 7 degrees of freedom each (see figure 9. To model it, its CAD-files were downloaded from [31], and imported into the *SpaceClaim* editor. With the plugin *Dynamics for SpaceClaim*, the positions of the joints could be identified and actuators attached. The density was assumed to

Equation	Algorithm Step	Description
(5.1)	-	Convert the desired position to a velocity (which is required by (3.3)).
(5.2)	3	A step in the control simulation with the velocity given by the motion planner and obtain a force.
(5.3)	5	Calculate new velocities for the bodies of the real robot simulation, using the forces calculated in (5.2).
(5.4)	5	Calculate the next set of robot positions, using the velocities obtained in (5.3).
(5.5)	7	Step back the control simulation by loading its previous state.
(5.6)	-	Calculate a velocity with the real robot's joint positions as desired positions.
(5.7)	8	A step in the control simulation. The forces received here could be used to predict the behavior of the robot, although this was not attempted in this project.

Table 1: Description of equations (5.1)-(5.7)

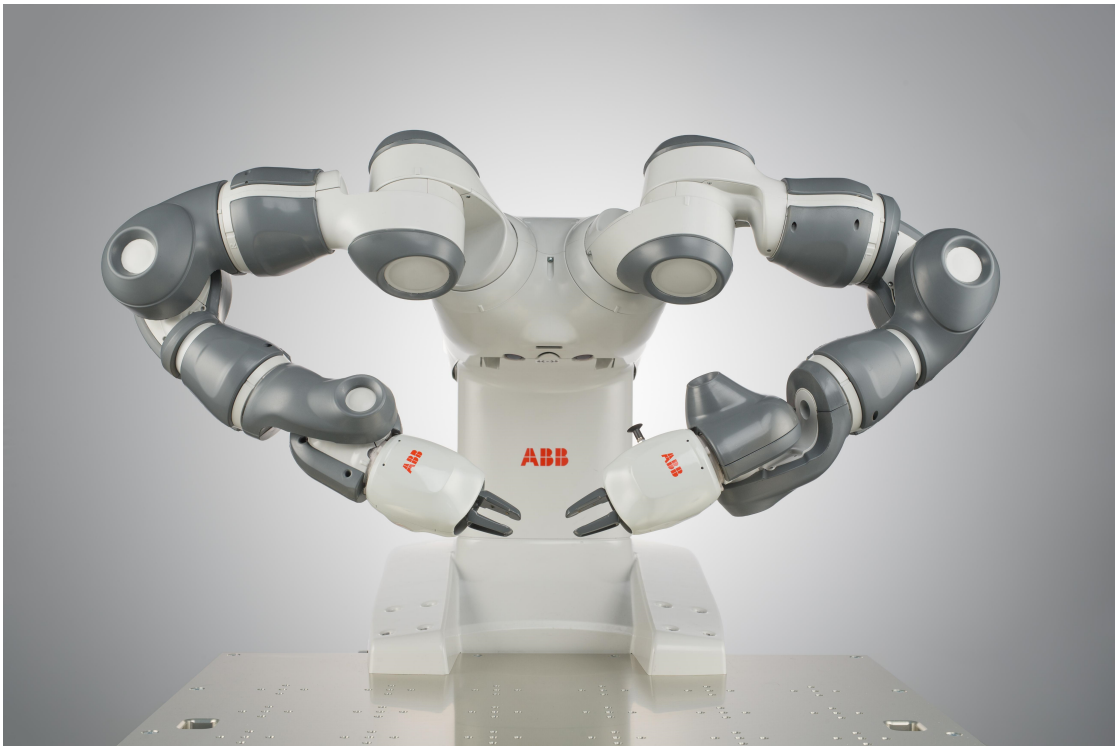


Figure 9: ABB Yumi [31].

be constant, and the total mass was set to 412 kg in accordance with its datasheet. According to its datasheet[1]. *Dynamics for SpaceClaim* automatically sets centres of gravity and moments of inertia according to the specified mass distributions.

6 Results

Presented below are the results obtained using various test cases, as well as a description of how the results are presented in the form of plots.

6.1 Test Scenarios

To test some aspects of the controller, a few test scenarios were conducted. Each simulation was conducted using a time step h of $\frac{1}{200}$ and $\frac{1}{2000}$ respectively to investigate the impact of updating frequencies on the stability. The goal is to test the same properties in all the models, but as the models are very different, individual tests had to be designed. These are described below.

[1] see Appendix A

6.1.1 Step Test

The step test aims at evaluating the controller's performance in moving from one point to another, with start and end velocities being 0.

Sphere A step in the sphere simulation consists of moving the sphere from $z = 0[m]$ to $z = 1[m]$.

Yumi A step for Yumi consists of moving the end-effector of each arm 0.3 m along the y-axis in opposite directions [32], i.e. the axis that runs through the sides of the robot base.

6.1.2 Offline Step Test

The offline step test uses an offline planner as described in section 5.2.1. A standalone robot model performs the same step as in 6.1.1 and stores the joint values for each time step in a table. This table is sent to the motion planner, which hands the controller one set of positions at a time.

6.1.3 Collisions

Collision testing is intended to show how the controller handles collisions, and if they can be detected, and possibly avoided.

Sphere By placing a plane perpendicular to the z-axis while performing a step, the sphere will collide with the plane.

6.1.4 Hand Guiding

Hand guiding is the act of manually moving the end effector by grabbing it and forcing it to a new position. This can be accomplished by setting a desired speed at the motion planner to 0, while updating its position according to the real robot's configuration.

Sphere To examine the effects of drifting when attempting hand guiding, the actuators of the trajectory simulation sphere was set to 0, and the simulation was run for 3 seconds.

6.2 Graphs

The results are presented as a series of graphs displaying various data about the simulations. These graphs are explained below:

Force Comparison ($F_{compare}$) Shows the force τ sent to the real robot (blue) and the force τ' applied by the control robot to reach the same position as the real robot (red). Compare the forces acquired in steps 4 and 8 of the algorithm in section 5.3.

	$F_{compare}$	F_{diff}	$P_{compare}$	P_{diff}	$V_{compare}$	V_{diff}
Sphere Step	x	x	x	x	x	x
Yumi Step		x	x	x		x
Sphere Offline	x	x	x	x	x	x
Yumi Offline		x	x	x		x
Sphere Collision	x	x	x	x		
Sphere Guiding	x	x	x			

Table 2: Matrix showing graph availability.

Force Difference (F_{diff}) Shows the difference between the forces above, i.e. $\tau - \tau'$

Path Comparison ($P_{compare}$) Shows the path specified by the motion planner q_d (blue) and the path taken by the robot q (red)

Path Difference (P_{diff}) Shows the difference between the paths above, i.e. $q - q_d$

Velocity Comparison ($V_{compare}$) Shows the velocity v_c required by the control robot to reach the position given by the motion planner (blue) and the velocity v applied by the real robot (red).

Velocity Difference (V_{diff}) Shows the difference between the forces above, i.e. $v - v_c$

What graphs are available for each scenario is visualized in table 2. Each simulation is conducted with two different time steps: $h = \frac{1}{200}$ and $h = \frac{1}{2000}$. Due to the size and number of the graphs, they are presented in Appendix A.

7 Conclusion

This paper has presented a possible new paradigm for controlling robot manipulators, where real-time physics simulations are present in every part of the control loop controller, from on-line motion planning to control of the individual actuator forces. The approach has shown promise in simulations, where signal noise and mass bias was added to the controlled robot to emulate differences between the model and real-world plant.

Another advantage of the system is its modularity, where both the motion planner and the control simulation can be replaced with other types of implementations using the FMI standard. These implementations doesn't have to consist of simulations at all, as long as they share the same in- and outputs.

7.1 Result Discussion

The results show that the controller is stable for all the different test scenarios, and also that it is able to handle some model error and noise, although to what extent will have

to be examined outside of this project. The force comparisons show some bias, and this is due to the model error where the real robot simulation has a 10% increase in mass. This difference is also the cause of the drift present in the hand guiding test, as the motion planner will update its position each time step with the incorrect position of the real robot (see step 9 of the algorithm in section 5.3.

Comparing the different sizes of time step, it is obvious that a smaller time step is able to reduce the error significantly. This is expected, as the controller will be able to account for the errors faster.

The collision test shows that the controller is able to detect collisions, as either a difference in joint position between expected and actual position, or as a force difference between the force needed to reach the expected position and the actual position (compare with equations (5.2) and (5.7) but the specifics of such collisions will have to be examined further.

Whether on-line or off-line planning makes very little difference for the result. This is probably due to the fact that the path was generated by an identical model as the control simulation robot. The hand-guiding scenario, however, requires an on-line planner to function, as the trajectory is not known beforehand.

7.2 Limitations

Even though the approach shows promise in simulation, there are some limitations to consider. In its current state, the algorithm requires that the the actuators can be modeled as constraints acting directly on their respective joint. This is a good approximation for smaller robots with simple actuator dynamics, like the ABB Yumi, but will present a serious problem when the motors need to be modeled with a drive-line, e.g. when the motor is not located at the joint, or when a gear box is utilized.

Further, when applying hand-guiding, the controller is susceptible to drift as the motion planner's only mission is to remain at 0 velocity, while the real robot is bound to make small deviations from its origin at each timestep due to model imperfections. Through feedback (part 5-8 in the main algorithm from section 5.3), these deviations are replicated in the motion planner simulation. This might be solved by applying feedback to the motion planner only when the difference in state between it and the control simulation becomes too large. This has, however, not been attempted in this project.

7.3 Future Work

The most obvious continuation of this work would be the implementation on a real robot, or at least research into what would be required to do so. Close to this is also the issue of a more quantitative performance benchmarking, as the testing conducted in this report is merely intended as a proof of concept. The controller performance could then be compared to other controllers, such as PID and computed torque control.

Another interesting topic to investigate is optimization of motion planning. The strategy implemented in this paper is guaranteed to find a viable path, but it does not attempt to find any sort of optimal path. Optimal motion planning is a huge subject, and look-

ing into how physics simulations could be utilized for this purpose would be a very interesting topic.

8 Acknowledgements

I would like to thank my supervisor, **Kenneth Bodin**, who helped me with both inspiration and motivation throughout the project. I would also like to thank all the employees at Algoryx Simulation AB, and especially **Michael Brandl**, **Nils Hjelte**, **Emanuel Dahlberg** and **Claude Lacoursière** for providing much needed support and insight.

References

[1] N. Keivan and G. Sibley, "Towards autonomous robotic systems: 14th annual conference, taros 2013, oxford, uk, august 28–30, 2013, revised selected papers," in, A. Natraj, S. Cameron, *et al.*, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, ch. Realtime Simulation-in-the-Loop Control for Agile Ground Vehicles, pp. 276–287, ISBN: 978-3-662-43645-5. DOI: 10.1007/978-3-662-43645-5_29. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-43645-5_29.

[2] M. Olsson, "Simulation and execution of autonomous robot systems," eng, PhD thesis, Lund University, 2002, p. 100, ISBN: 91-628-5120-9.

[3] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, Oct. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[4] P. Abreu, M. R. Barbosa, and A. M. Lopes, "Robotics virtual lab based on off-line robot programming software," in *Experiment@ International Conference (exp.at'13), 2013 2nd*, Sep. 2013, pp. 109–113. DOI: 10.1109/ExpAt.2013.6703040.

[5] C. Lacoursière, "Ghosts and machines : Regularized variational methods for interactive simulations of multibodies with dry frictional contacts," PhD thesis, Umeå University, Computing Science, 2007, p. 444.

[6] R. Aparnathi and V. V. Dwivedi, "The novel of six axes robotic arm for industrial applications," *IAES International Journal of Robotics and Automation (IJRA)*, vol. 3, no. 3, pp. 161–167, 2014. [Online]. Available: http://iaesjournal.com/online/index.php/IJRA/article/view/4892.

[7] M. F. Mendes, W. Kraus Jr., and E. R. d. Pieri, "Variable structure position control of an industrial robotic manipulator," en, *Journal of the Brazilian Society of Mechanical Sciences*, vol. 24, pp. 169–176, Jul. 2002, ISSN: 0100-7386. [Online]. Available: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0100-73862002000300004&nrm=iso.

[8] I. Asimov, *I, Robot*. Facwcett Crest, 1933.

[9] A. Boeing and T. Bräunl, "Evaluation of real-time physics simulation systems," in *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*, ser. GRAPHITE '07, Perth, Australia: ACM, 2007, pp. 281–288, ISBN: 978-1-59593-912-8. DOI: 10.1145/1321261.1321312. [Online]. Available: http://doi.acm.org/10.1145/1321261.1321312.

[10] Y.-T. Wang and V.-j. Kumar, "Simulation of mechanical systems with multiple frictional contacts," *Journal of Mechanical Design*, vol. 116, no. 2, pp. 571–580, 1994.

[11] J. Ginsberg, *Engineering dynamics*. Cambridge University Press, 2008, vol. 10, ch. Generalized Coordinates and Kinematical Constraints, pp. 396–408.

[12] *Agx dynamics user manual*, 2.15.0.0, Algoryx Simulations AB, Feb. 2016.

[13] W. Schiehlen, "Multibody system dynamics: Roots and perspectives," *Multibody System Dynamics*, vol. 1, no. 2, pp. 149–188, ISSN: 1573-272X. DOI: 10.1023/A:1009745432698. [Online]. Available: http://dx.doi.org/10.1023/A:1009745432698.

[14] C. Lacoursière, personal communication, Mar. 4, 2016.

[15] R. Kelly, V. S. Davila, and A. Loría, "Control of robot manipulators in joint space," in. London: Springer London, 2005, ch. What Does Control of Robots Involve? Pp. 7–17, ISBN: 978-1-85233-999-9. DOI: 10.1007/1-85233-999-3_2. [Online]. Available: http://dx.doi.org/10.1007/1-85233-999-3_2.

[16] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989, ISSN: 0005-1098. DOI: http://dx.doi.org/10.1016/0005-1098(89)90002-2. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0005109889900022.

[17] P. Agachi, Z. Nagy, *et al.*, *Model Based Control: Case Studies in Process Engineering*. Wiley, 2007, ISBN: 9783527609222. [Online]. Available: https://books.google.se/books?id=kLeQqpbDj20C.

[18] C. H. An, C. G. Atkeson, and J. M. Hollerbach, *Model-based Control of a Robot Manipulator*. Cambridge, MA, USA: MIT Press, 1988, ISBN: 0-262-01102-6.

[19] R. Kelly, V. S. Davila, and A. Loría, *Control of Robot Manipulators in Joint Space*. London: Springer London, 2005, ISBN: 978-1-85233-999-9. DOI: 10.1007/1-85233-999-3_2. [Online]. Available: http://dx.doi.org/10.1007/1-85233-999-3_2.

[20] F. Flacco, "Modeling and control of robots with compliant actuation," PhD thesis, Universitá di Roma, Dipartimento di Ingegneria Informatica, Automatica e Gestionale, 2012.

[21] R. Kelly, V. S. Davila, and A. Loría, "Control of robot manipulators in joint space," in. London: Springer London, 2005, ch. Computed-torque Control and Computed-torque+ Control, pp. 227–241, ISBN: 978-1-85233-999-9. DOI: 10.1007/1-85233-999-3_13. [Online]. Available: http://dx.doi.org/10.1007/1-85233-999-3_13.

[22] Z. Shiller, "Motion and operation planning of robotic systems: Background and practical approaches," in, G. Carbone and F. Gomez-Bravo, Eds. Cham: Springer International Publishing, 2015, ch. Off-Line and On-Line Trajectory Planning, pp. 29–62, ISBN: 978-3-319-14705-5. DOI: 10.1007/978-3-319-14705-5_2. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-14705-5_2.

[23] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 1, pp. 23–32, Feb. 1992, ISSN: 1042-296X. DOI: 10.1109/70.127236.

[24] J. Cortes and T. Simeon, "Probabilistic motion planning for parallel mechanisms," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 3, Sep. 2003, 4354–4359 vol.3. DOI: 10.1109/ROBOT.2003.1242274.

[25] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, Available at http://planning.cs.uiuc.edu/.

[26] H. Choset, K. M. Lynch, *et al.*, *Principles of Robot Motion:Theory, Algorithms, and Implementations*. MIT Press, 2005, ISBN: 9780262255912.

[27] Z. Shiller and S. Dubowsky, "On computing the global time-optimal motions of robotic manipulators in the presence of obstacles," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 6, pp. 785–797, Dec. 1991, ISSN: 1042-296X. DOI: 10.1109/70.105387.

[28] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *Int. J. Rob. Res.*, vol. 5, no. 3, pp. 72–89, Sep. 1986, ISSN: 0278-3649. DOI: 10.1177/027836498600500304. [Online]. Available: http://dx.doi.org/10.1177/027836498600500304.

[29] L. Barinka and R. Berka, "Inverse kinematics-basic methods," Czech Technical University, Dept. of Computer Science and Engineering, Tech. Rep., 2002.

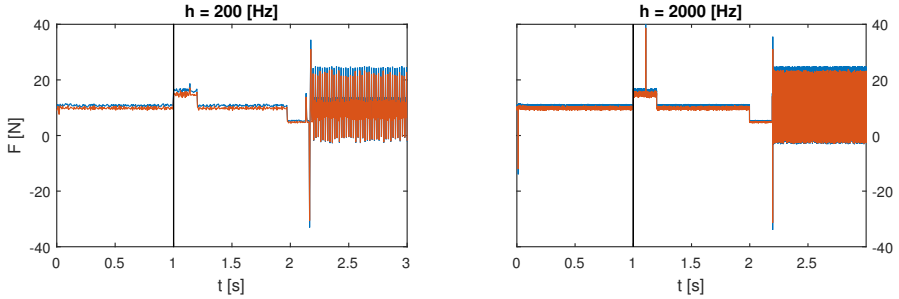
[30] T. Blochwitz, M. Otter, *et al.*, "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models," eng, in *Proceedings of the 9th International Modelica Conference*, Munich, Germany: The Modelica Association, 2012, pp. 173–184, ISBN: 978-91-7519-826-2. [Online]. Available: http://dx.doi.org/10.3384/ecp12076173.

[31] ABB. (2016). Yumi - abb, [Online]. Available: http://new.abb.com/products/robotics/yumi (visited on 03/02/2016).

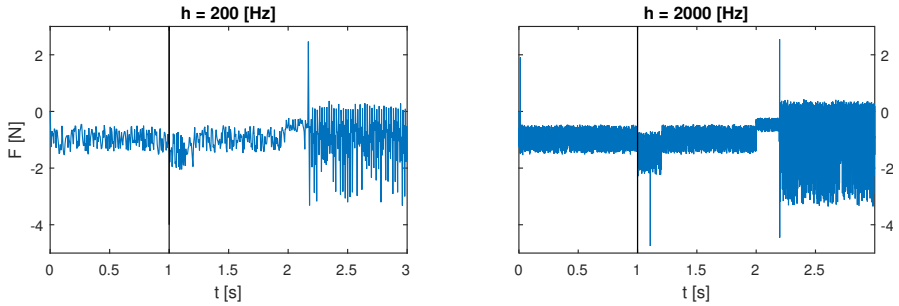
[32] M. Törnqvist. (Mar. 7, 2016). Agx yumi step, Youtube, [Online]. Available: https://www.youtube.com/watch?v=OUJ-wwdkC70.

Appendix A: Result Graphs

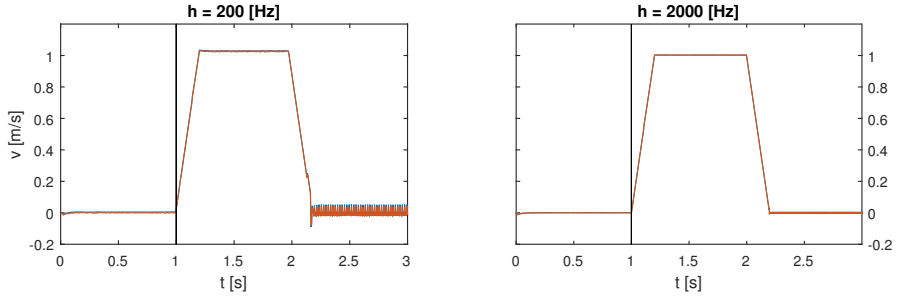
Sphere Step: Force Comparison



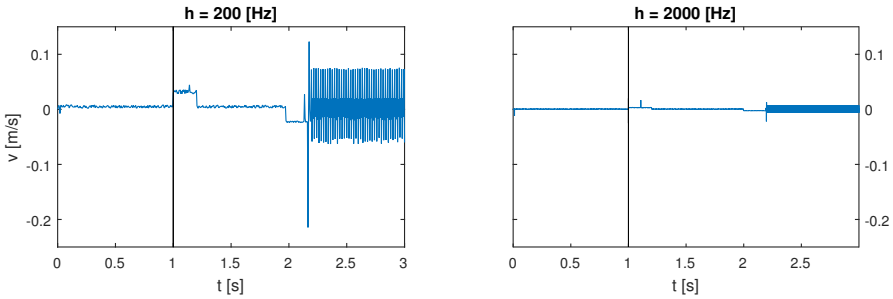
Sphere Step: Force Difference



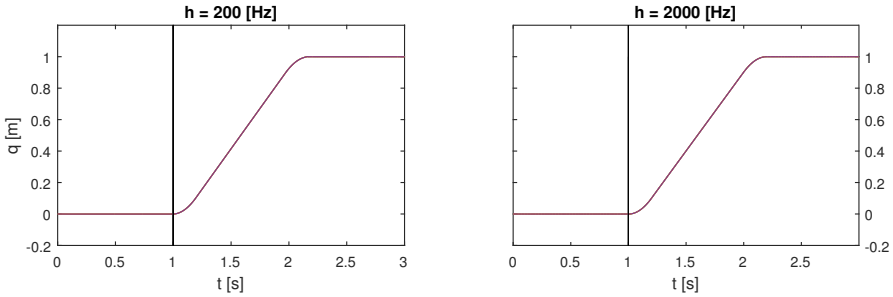
Sphere Step: Velocity Comparison



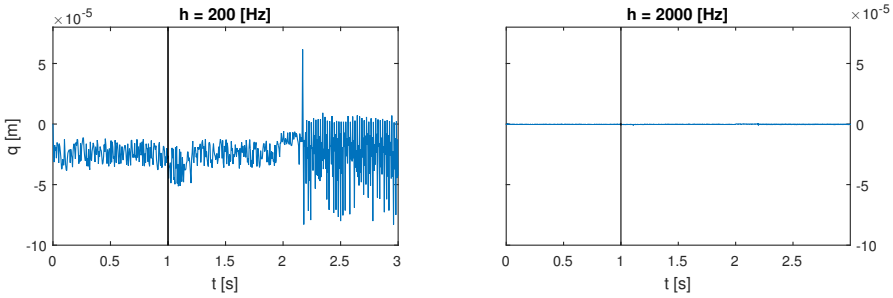
Sphere Step: Velocity Difference



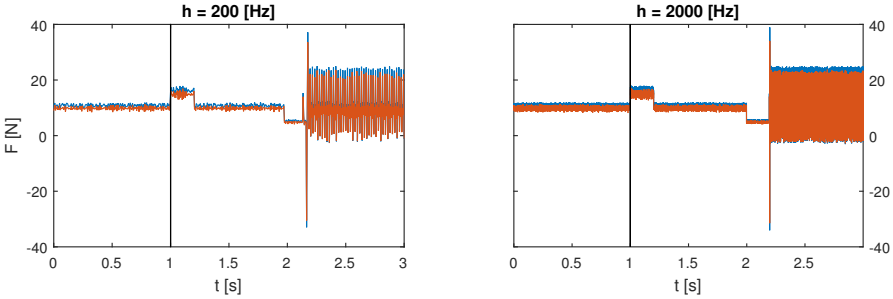
Sphere Step: Path Comparison



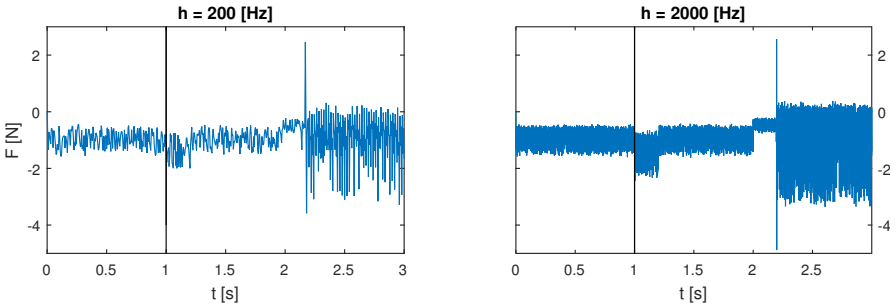
Sphere Step: Path Difference



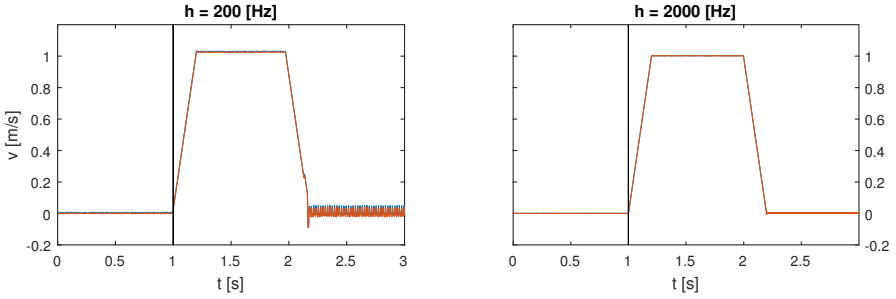
Sphere Offline: Force Comparison



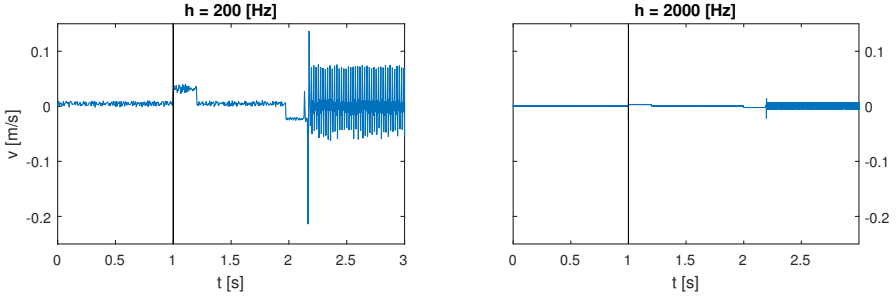
Sphere Offline: Force Difference



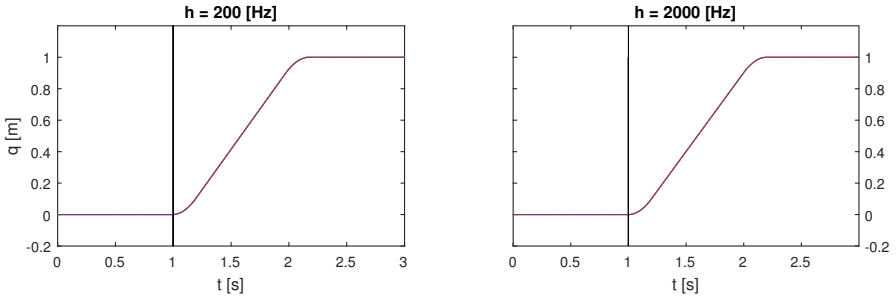
Sphere Offline: Velocity Comparison



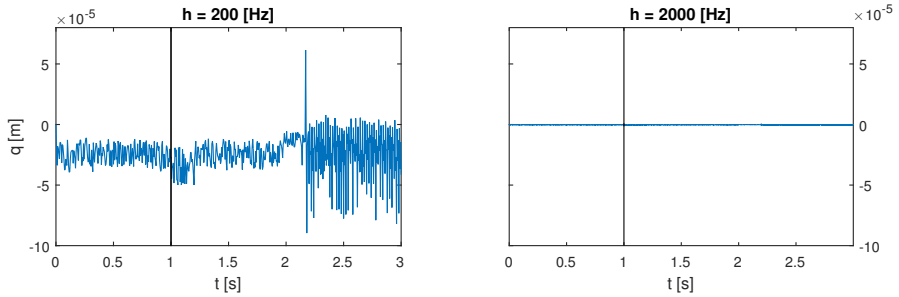
Sphere Offline: Velocity Difference



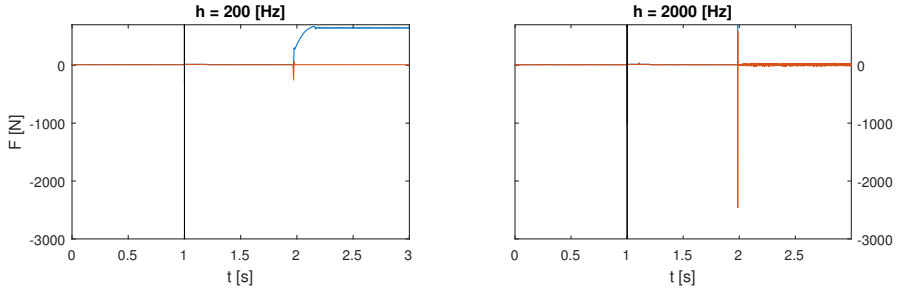
Sphere Offline: Path Comparison



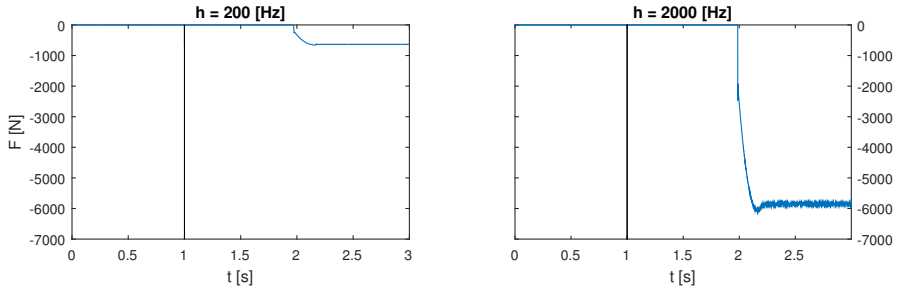
Sphere Offline: Path Difference



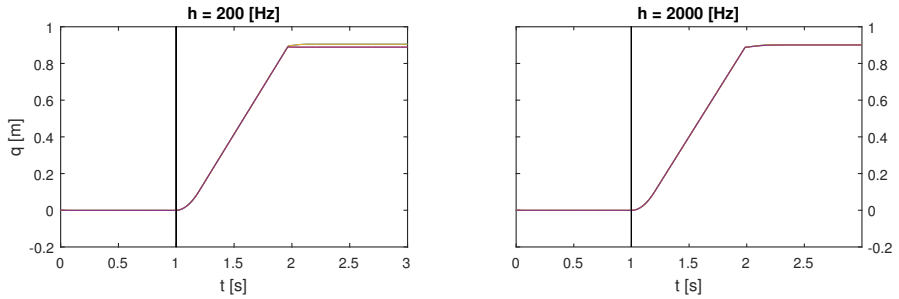
Sphere Collision: Force Comparison



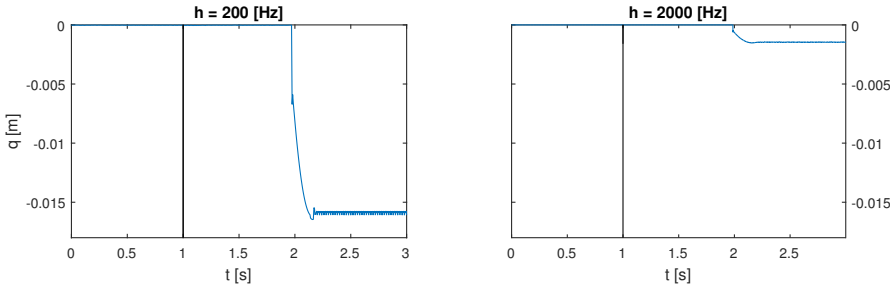
Sphere Collision: Force Difference



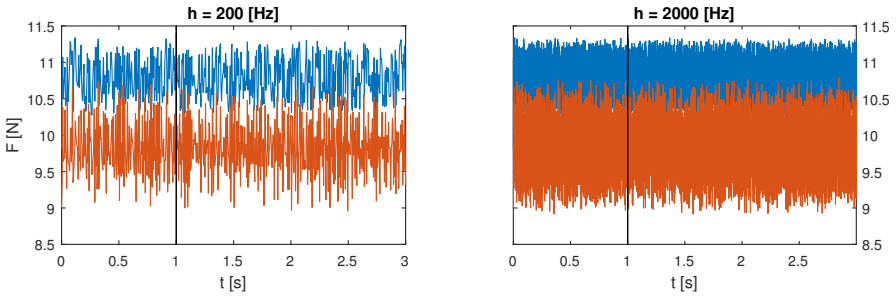
Sphere Collision: Path Comparison



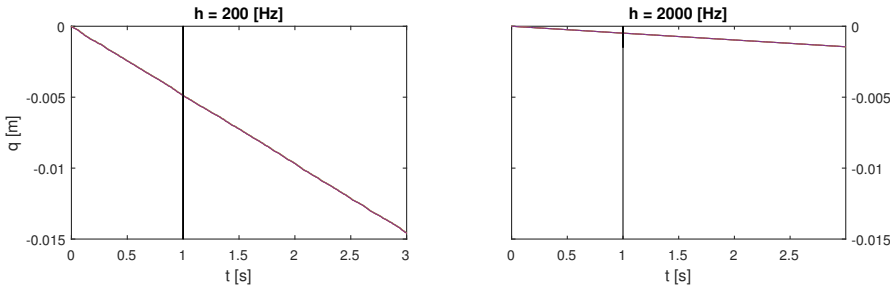
Sphere Collision: Path Difference



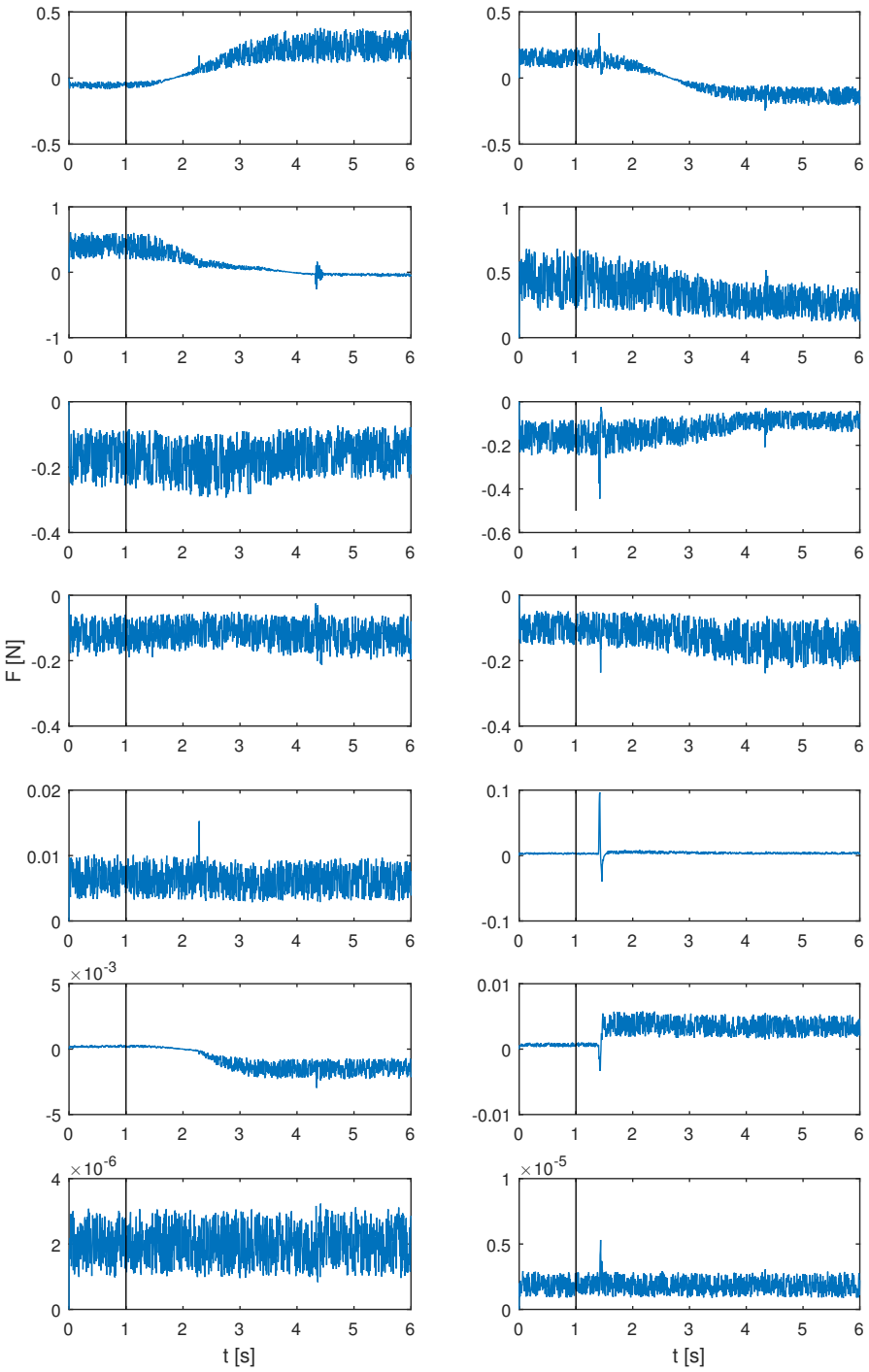
Sphere Guiding: Force Comparison



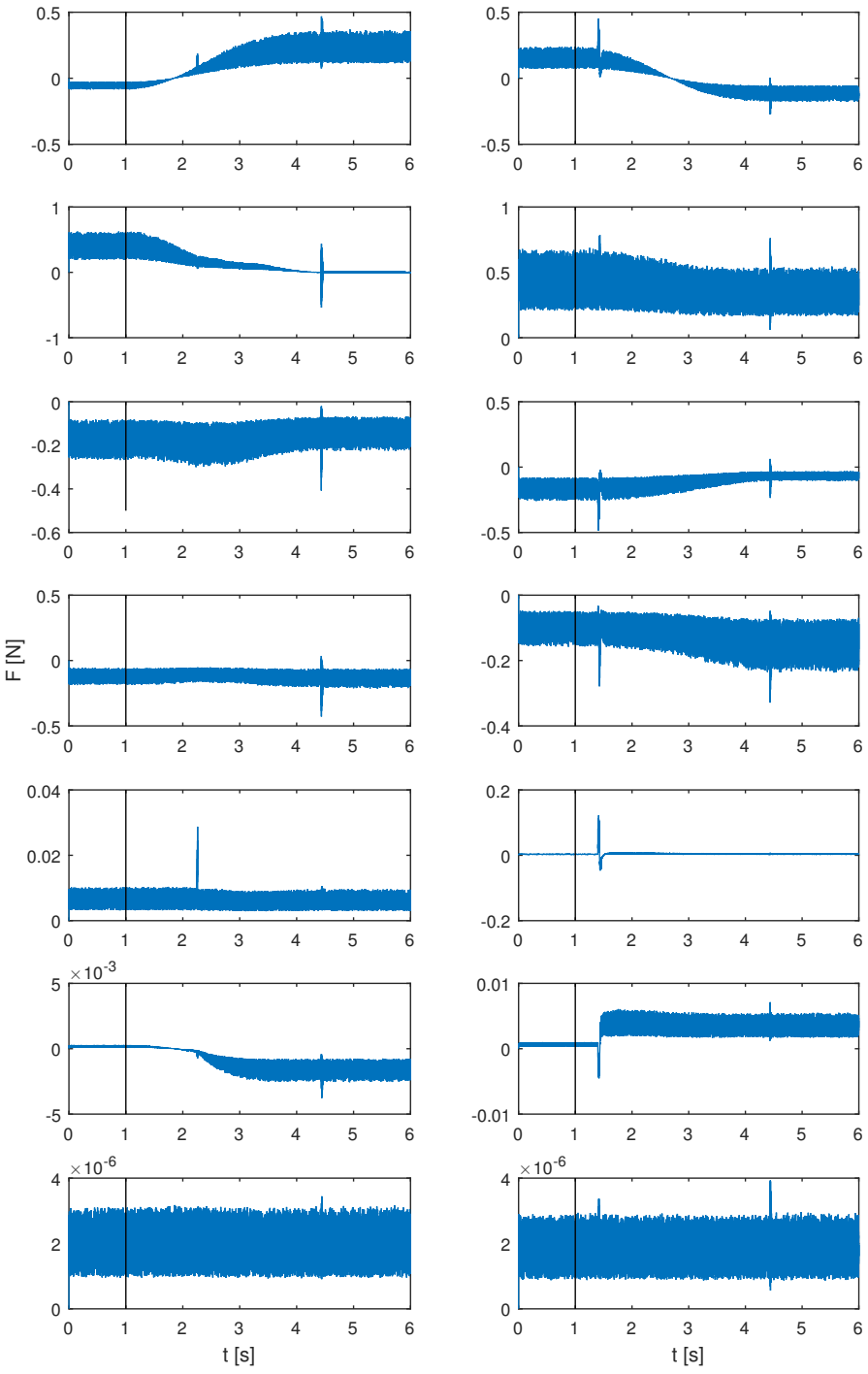
Sphere Guiding: Path Comparison



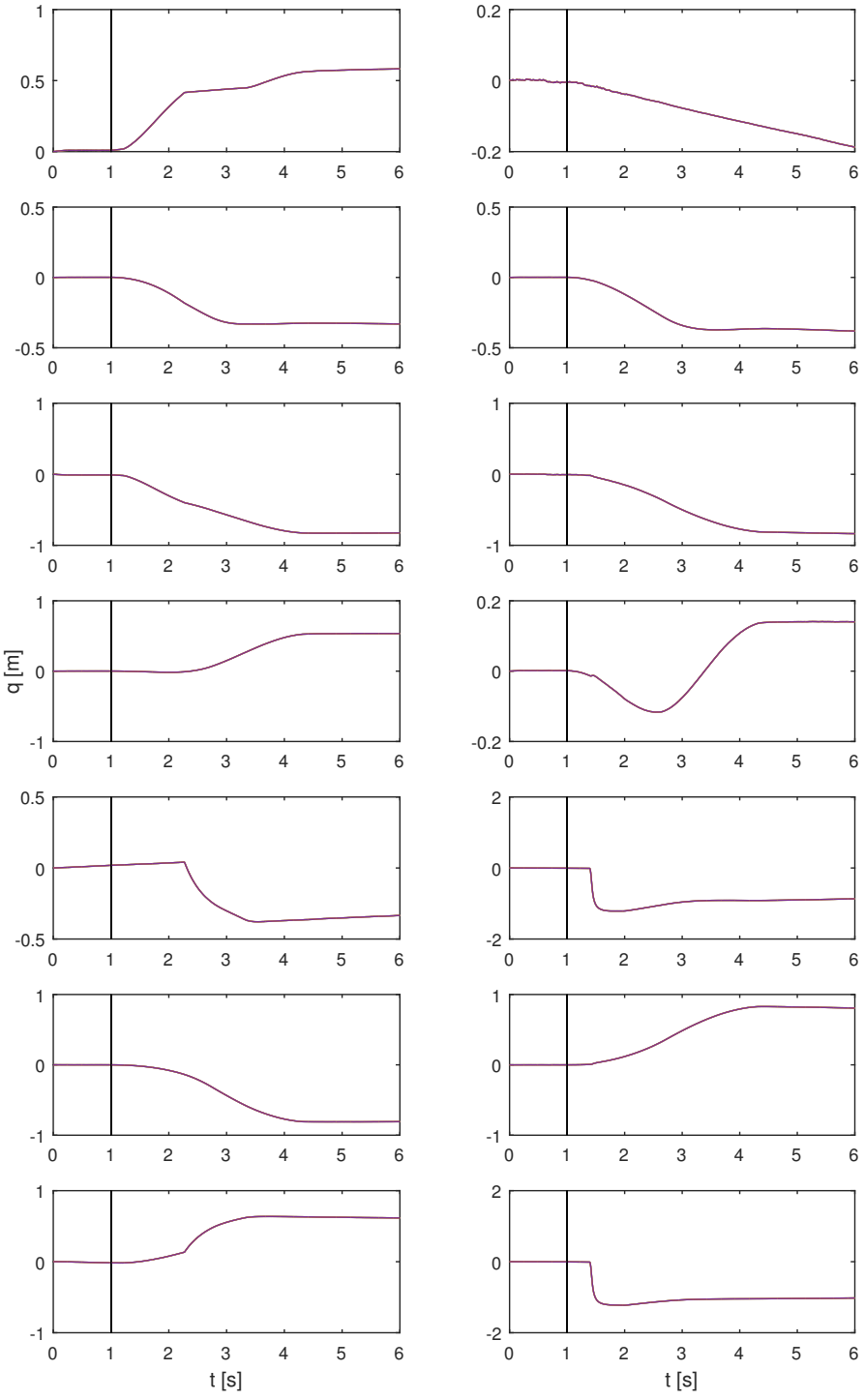
Yumi Step: Force Difference, h = 1/200



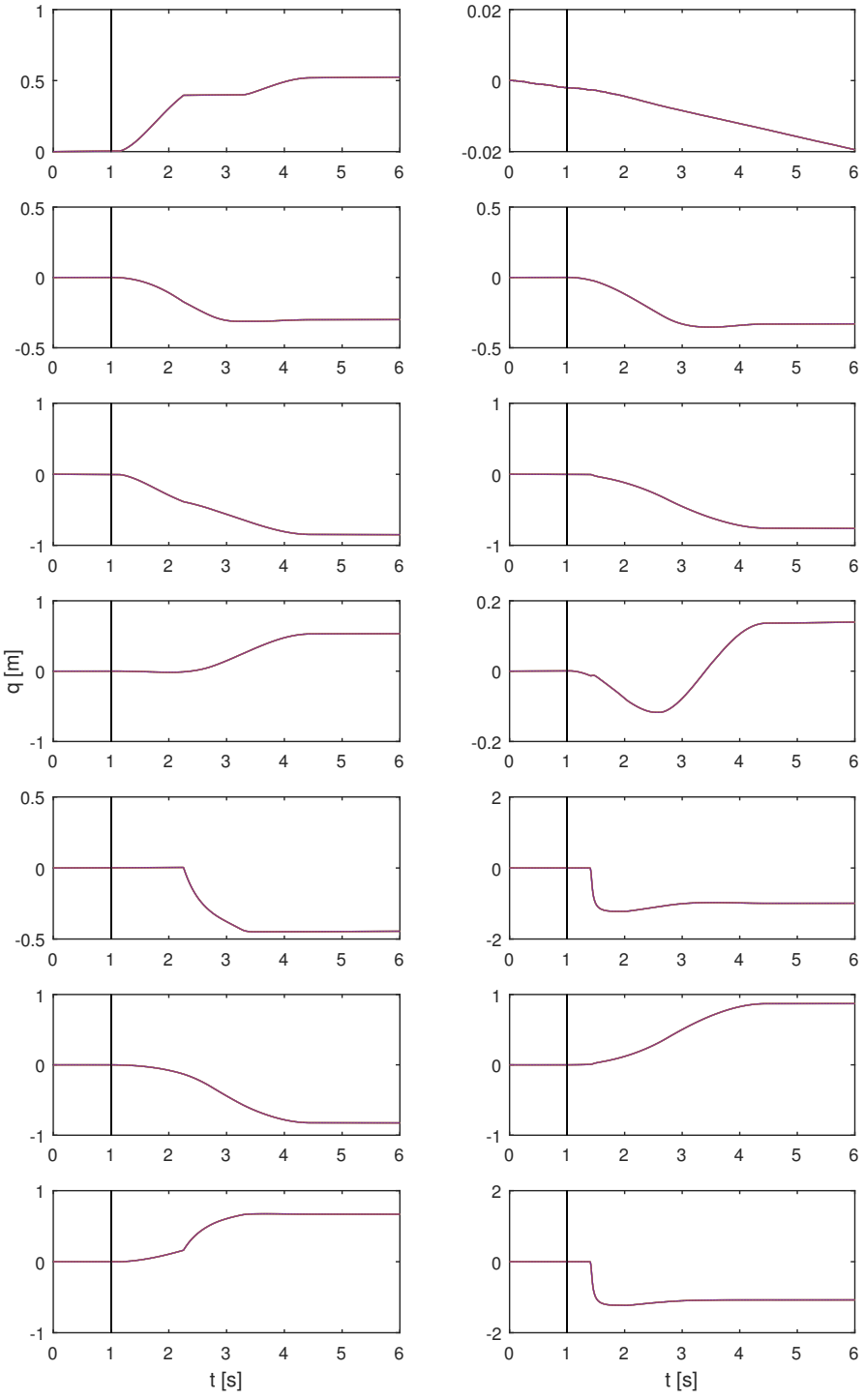
Yumi Step: Force Difference, h = 1/2000



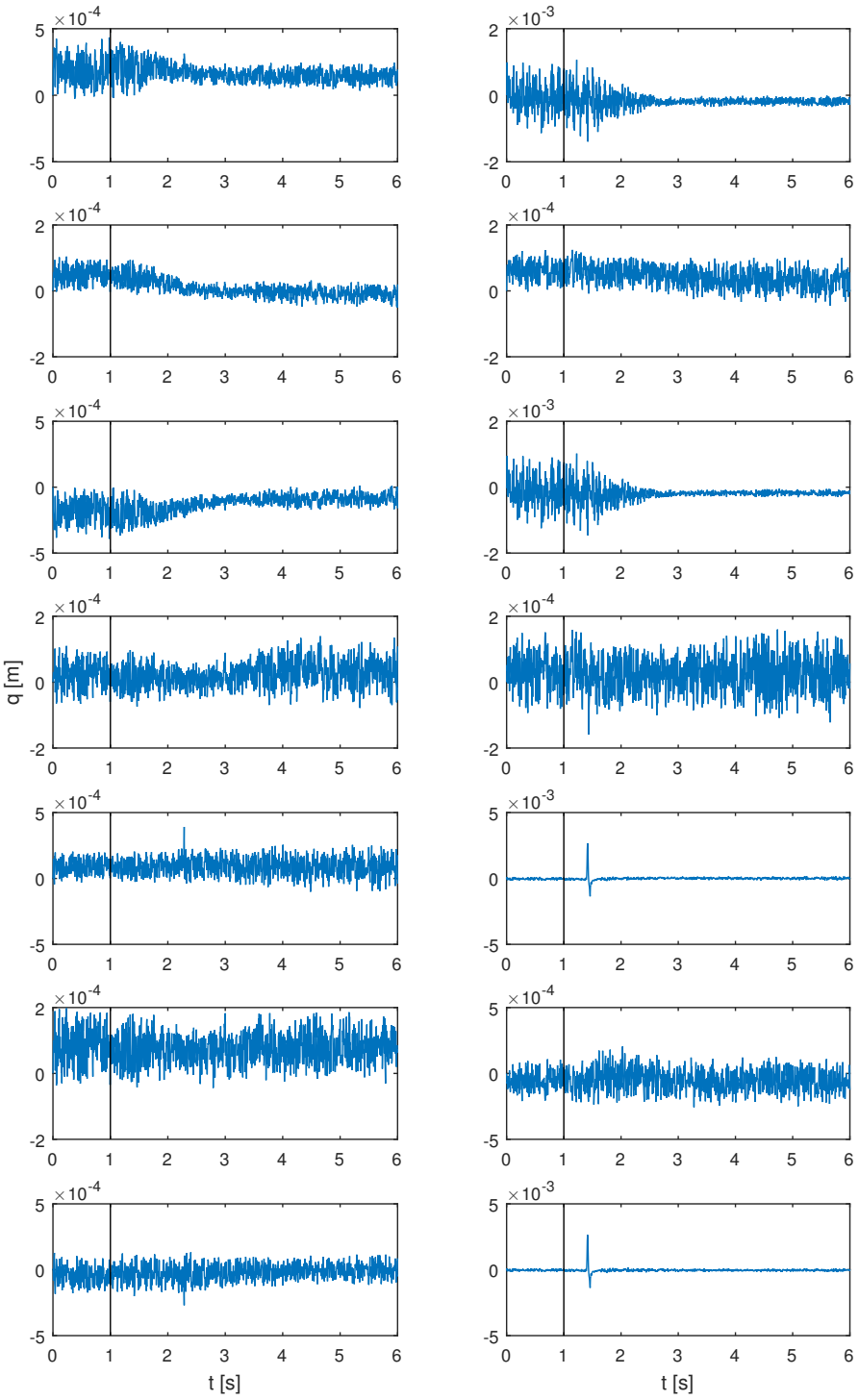
Yumi Step: Path Comparison, h = 1/200



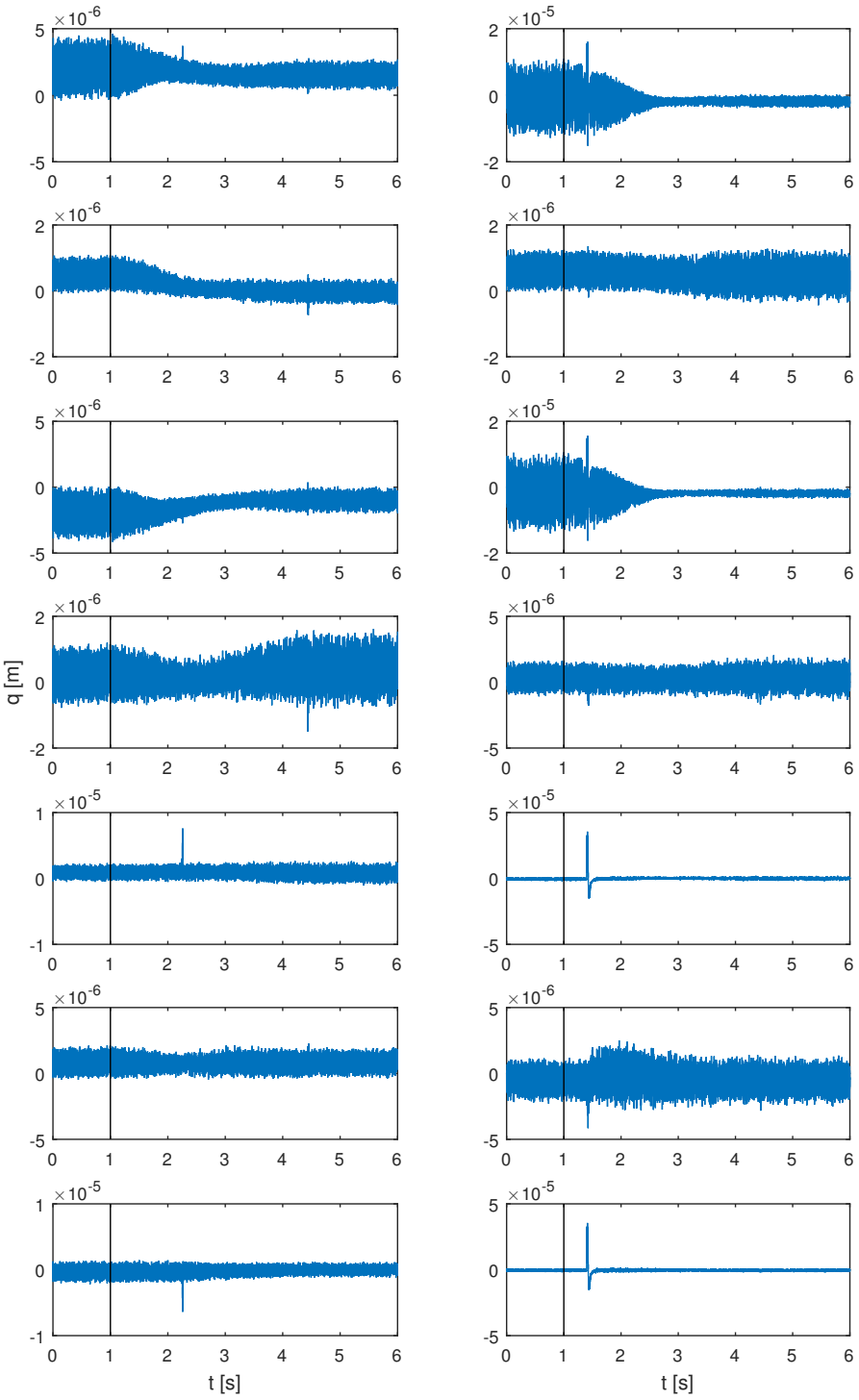
Yumi Step: Path Comparison, h = 1/2000



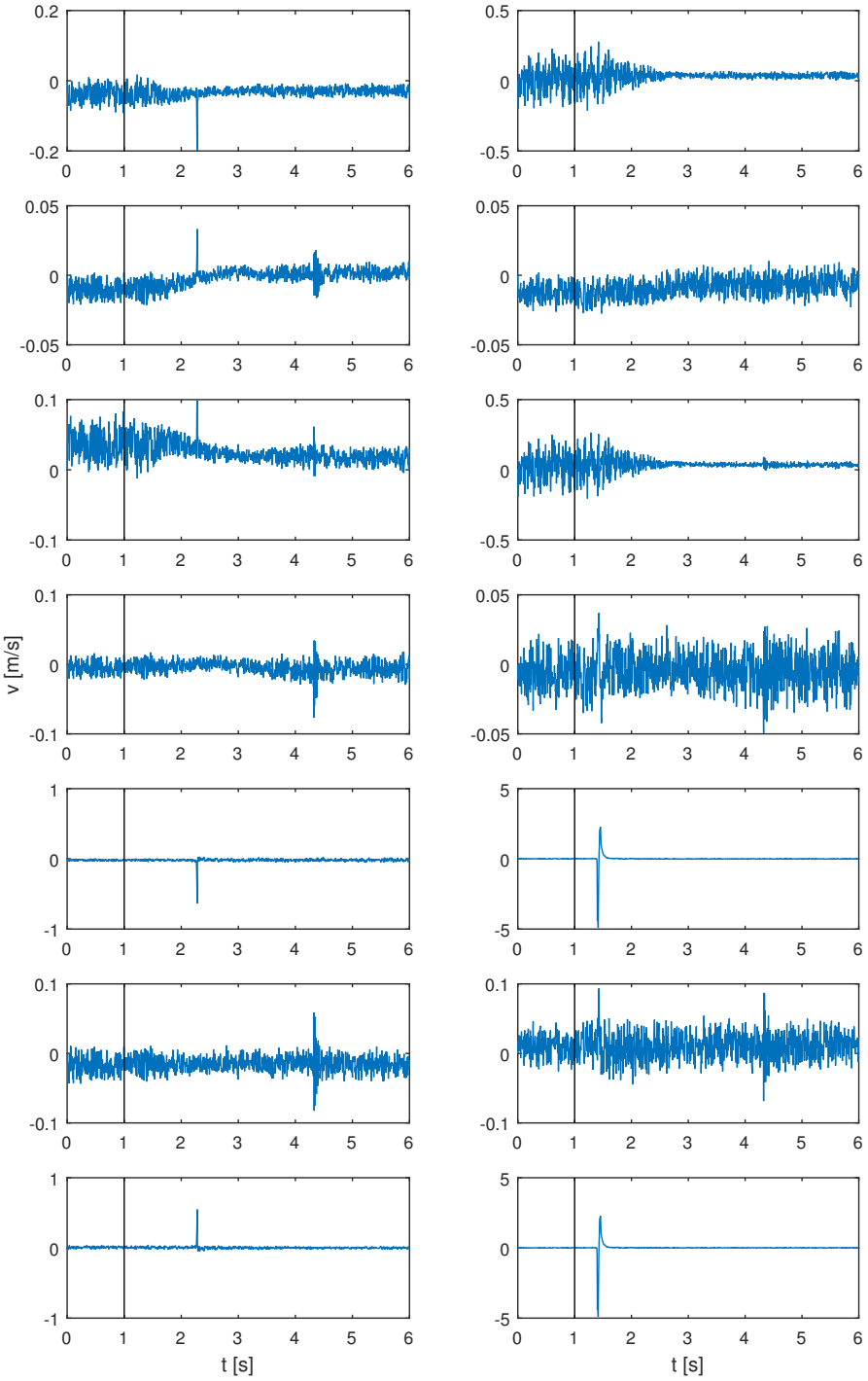
Yumi Step: Path Difference, h = 1/200



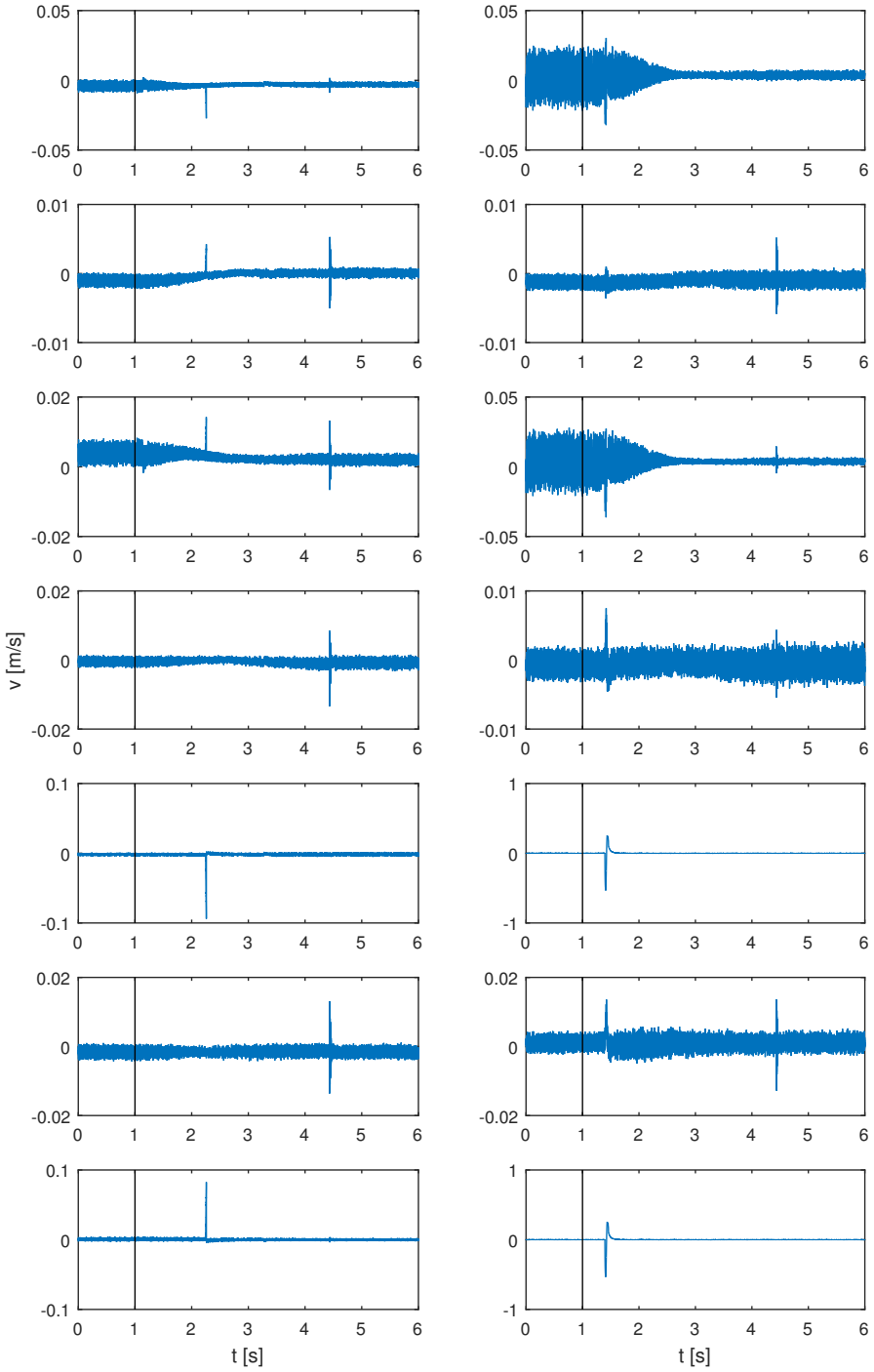
Yumi Step: Path Difference, h = 1/2000



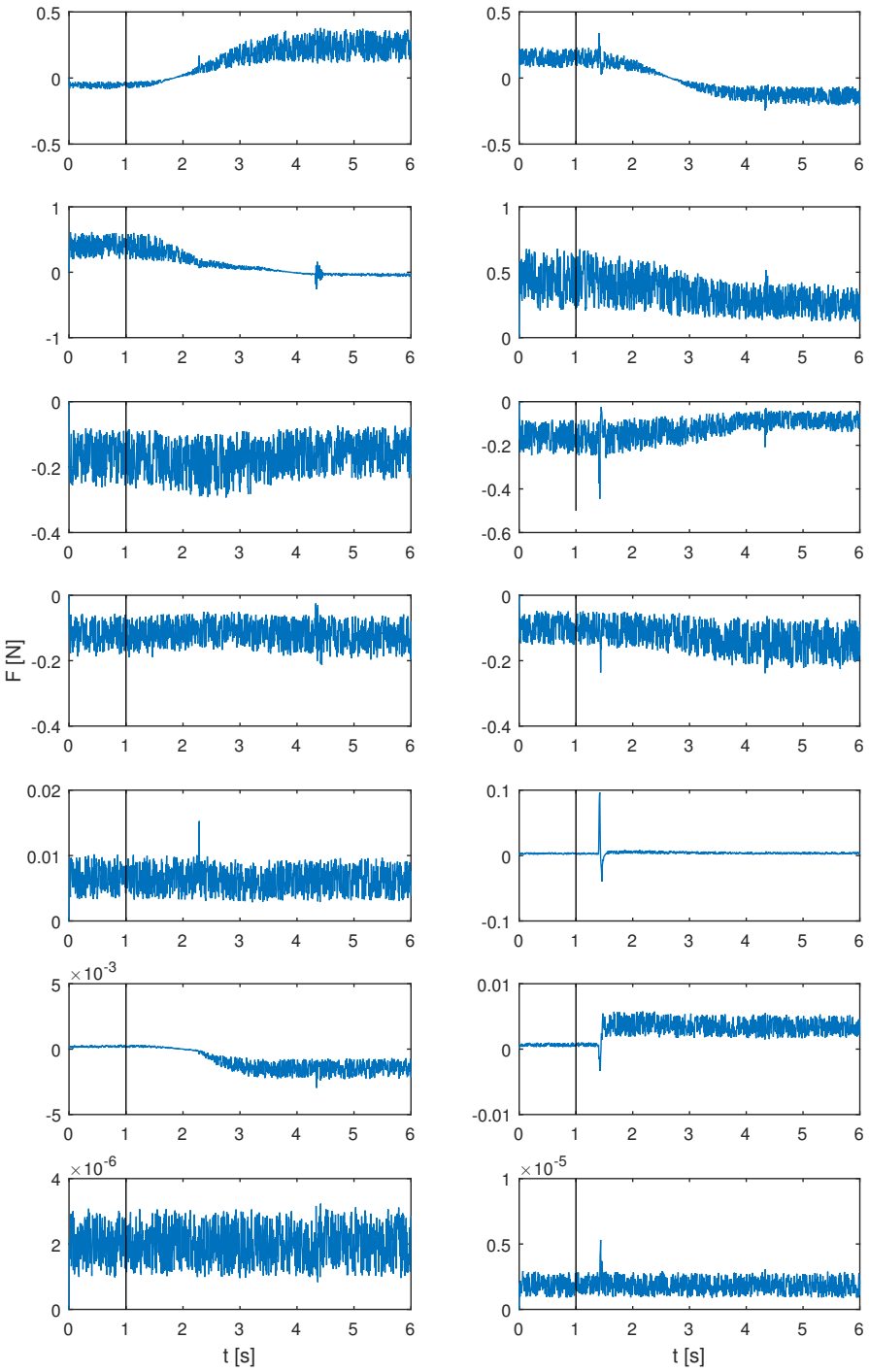
Yumi Step: Velocity Difference, h = 1/200



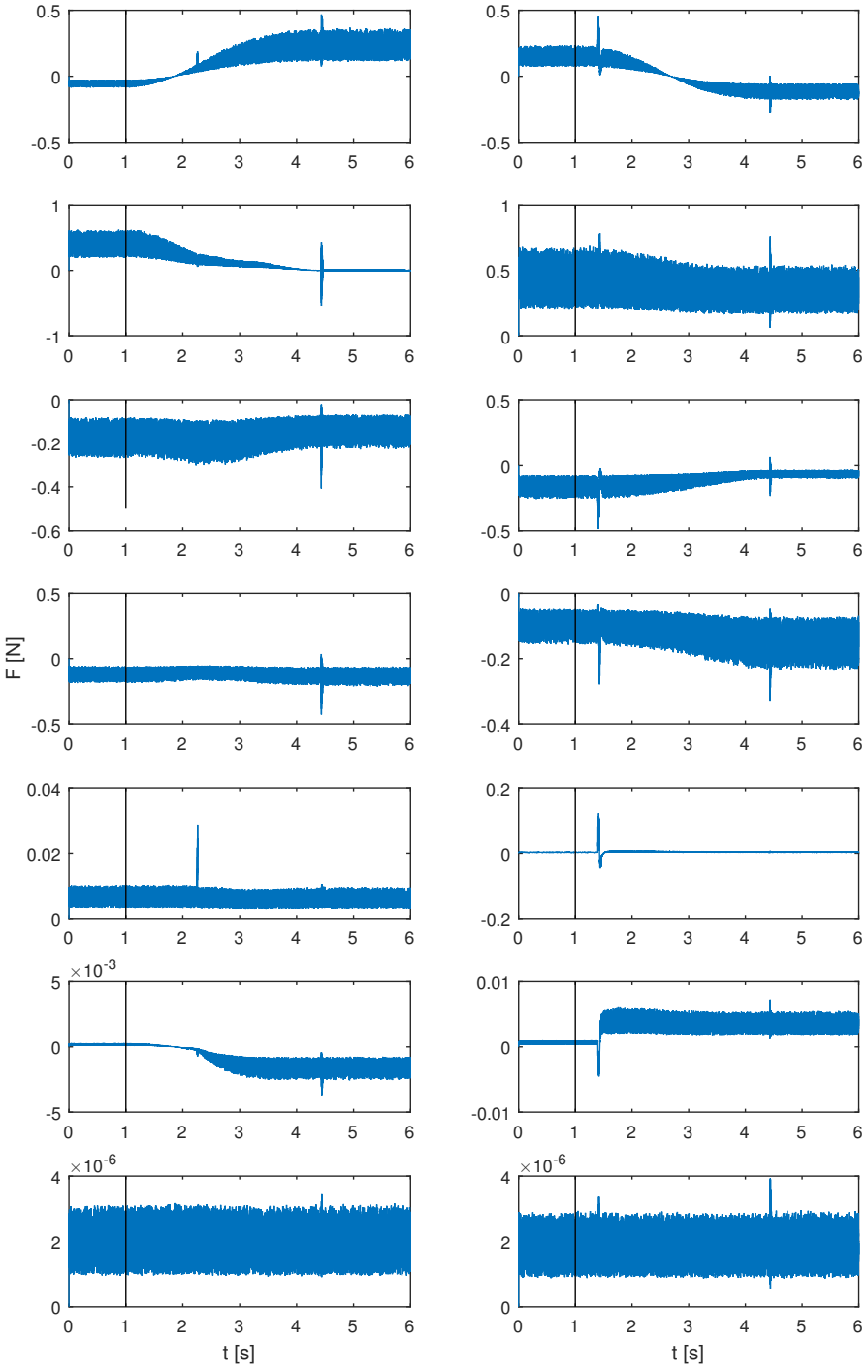
Yumi Step: Velocity Difference, h = 1/2000



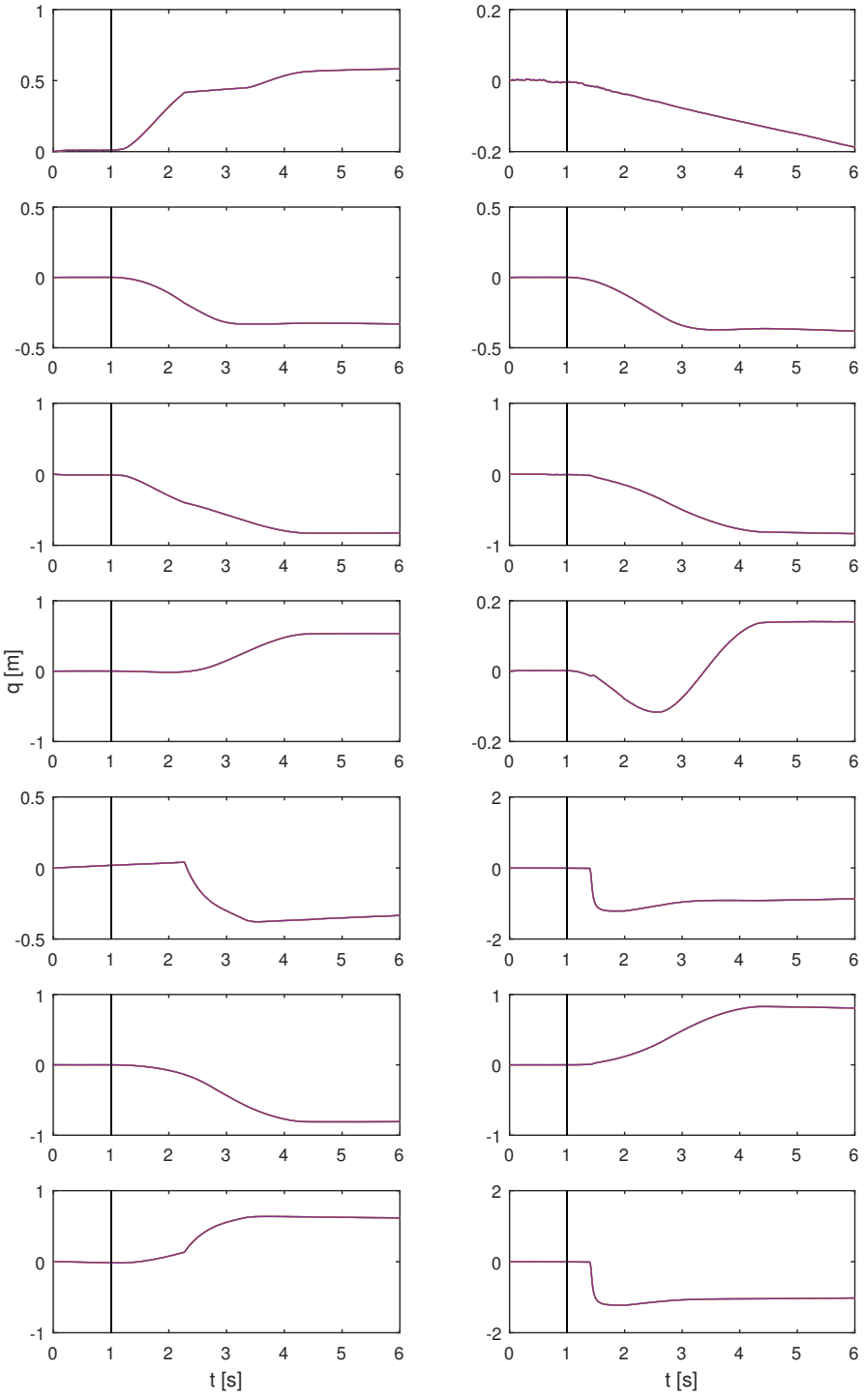
Yumi Offline: Force Difference, h = 1/200



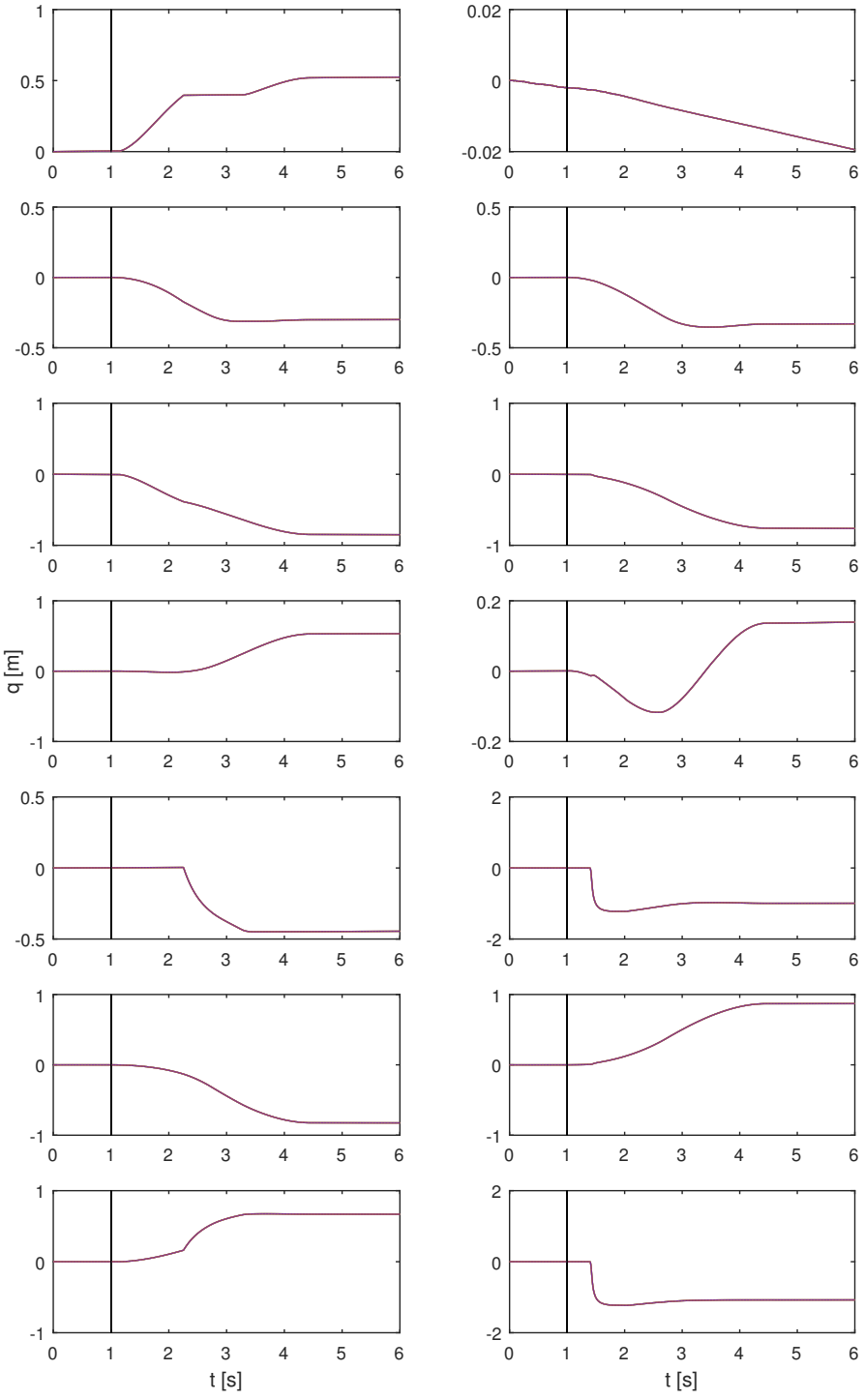
Yumi Offline: Force Difference, h = 1/2000



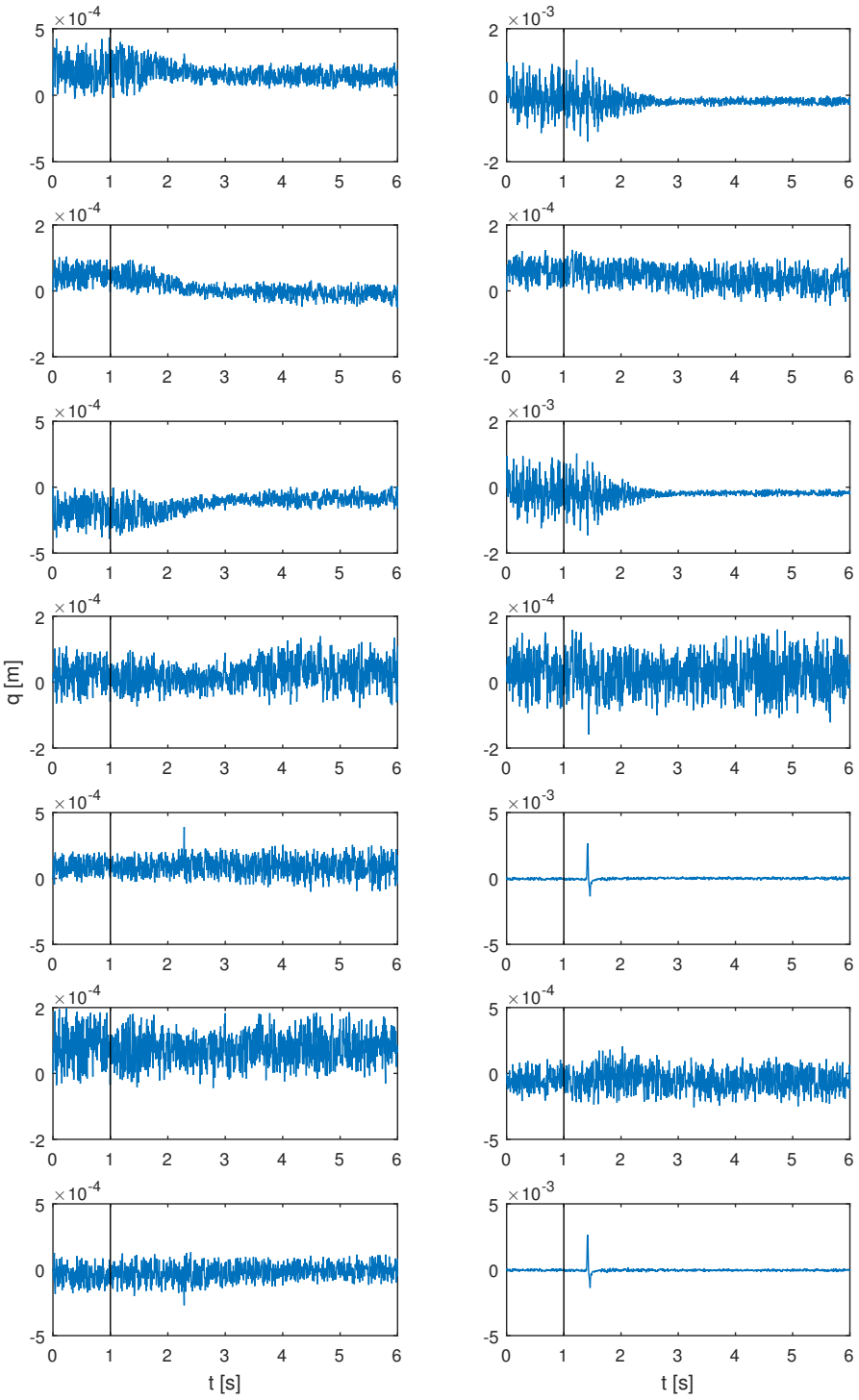
Yumi Offline: Path Comparison, h = 1/200



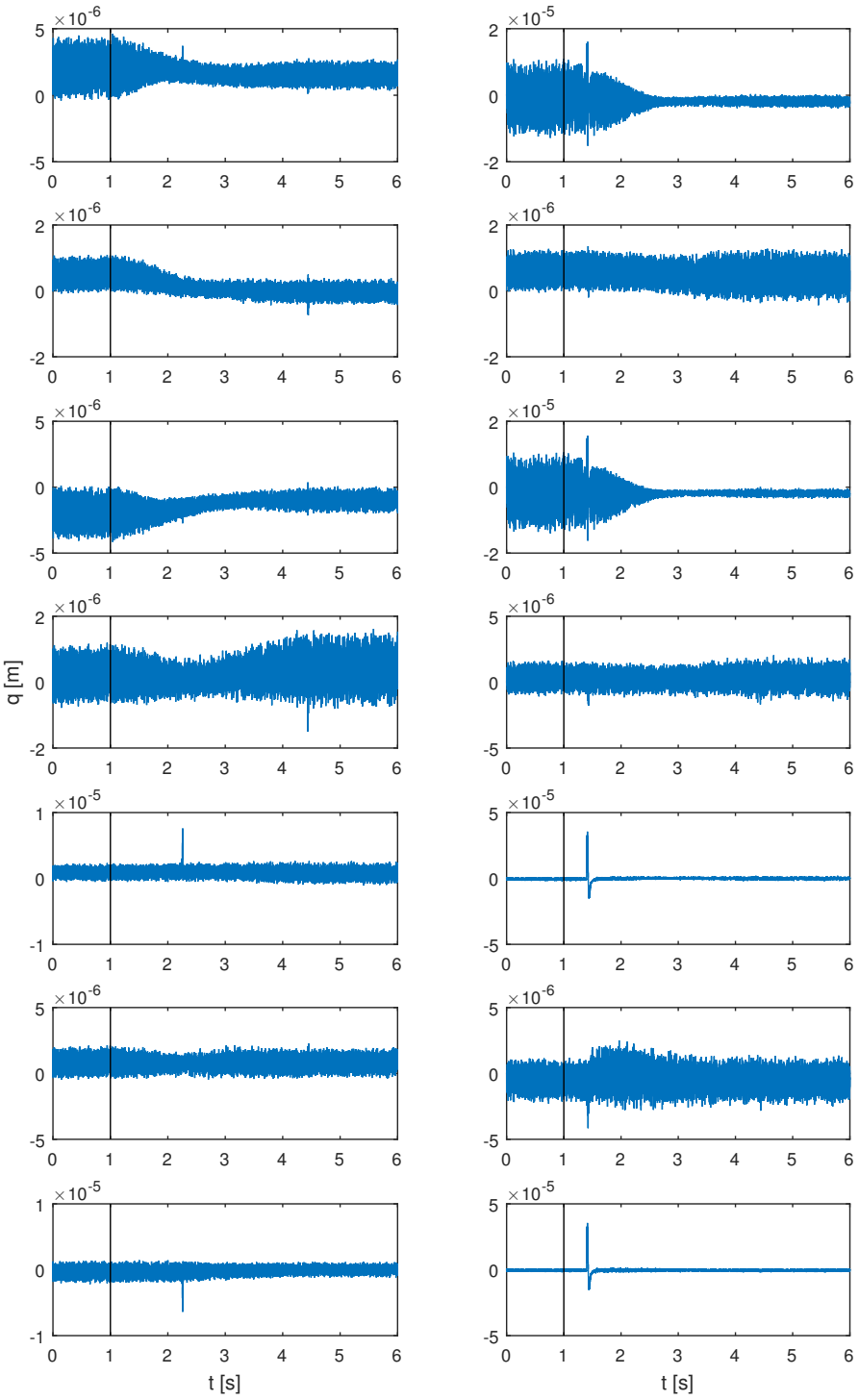
Yumi Offline: Path Comparison, h = 1/2000



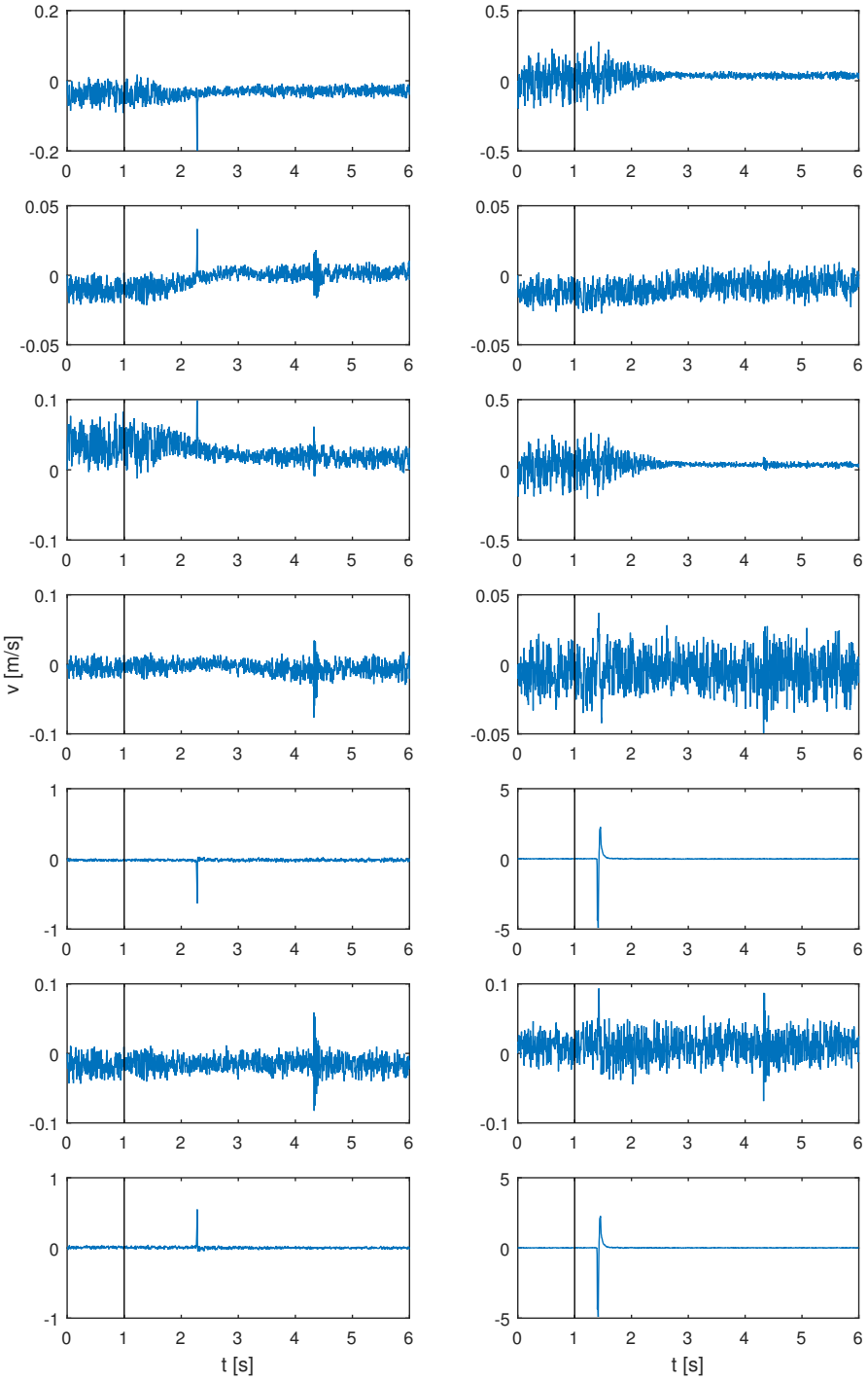
Yumi Offline: Path Difference, h = 1/200



Yumi Offline: Path Difference, h = 1/2000



Yumi Offline: Velocity Difference, h = 1/200



Yumi Offline: Velocity Difference, h = 1/2000