RBE/ME 501 course project

Force Controllers for daVinci in AMBF Simulator

Team
Farid Tavakkologhaddam
Sapan Agrawal
Kartik Patath
Hao Yang
Tianyu Zhu

Instructor
Prof. Gregory Fischer

Introduction & Background

Da Vinci Research Kit (dVRK):

- An Open teleoperated surgical robotic system consisting of master and slave sides

Patient Side Manipulator (PSM):

- Comprised of two tool manipulator arms and one endoscope
- 7 DOF for dexterous and natural hand manipulation

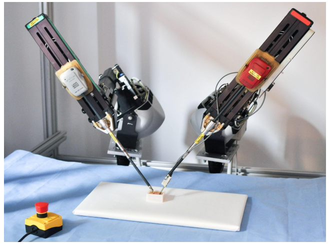
Master Tool Manipulator (MTM):

- Comprised of two haptic manipulator arms
- 7 DOF for dexterous and natural hand manipulation

Control challenges

- Lack of feedback from haptic devices
- No compliant controller available for either the PSM or MTM

a) Clinical da Vinci system

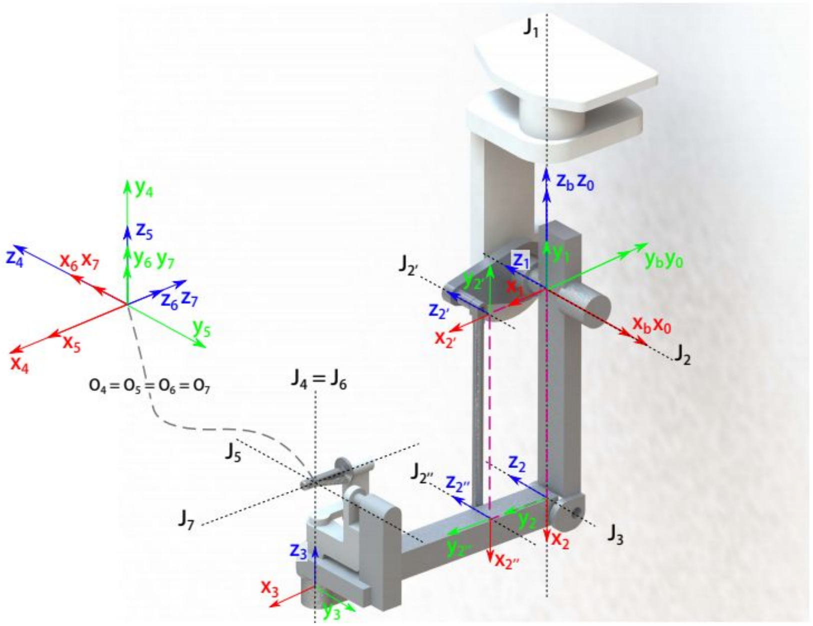


b) PSM arms of dVRK [1,2]

c) dVRK MTM [5]

MTM Kinematic Model

- Overall structure rotates about the vertical axis of J1 of an angle θ1
- Revolute joints with axes J2, J2' , J2'' and J3 form a 2-DOF parallelogram mechanism
- Two actuated joints of the parallelogram are those about axes J2 (angle θ2) and J3 (angle θ3)
- the axes J4, J5, J6 and J7 intersect in the same point and correspond to revolute joints with angles θ4, θ5, θ6 and θ7
- All the joints are actuated by a motor, with the exception of the two revolute joints of the parallelogram about axes J2' and J2''

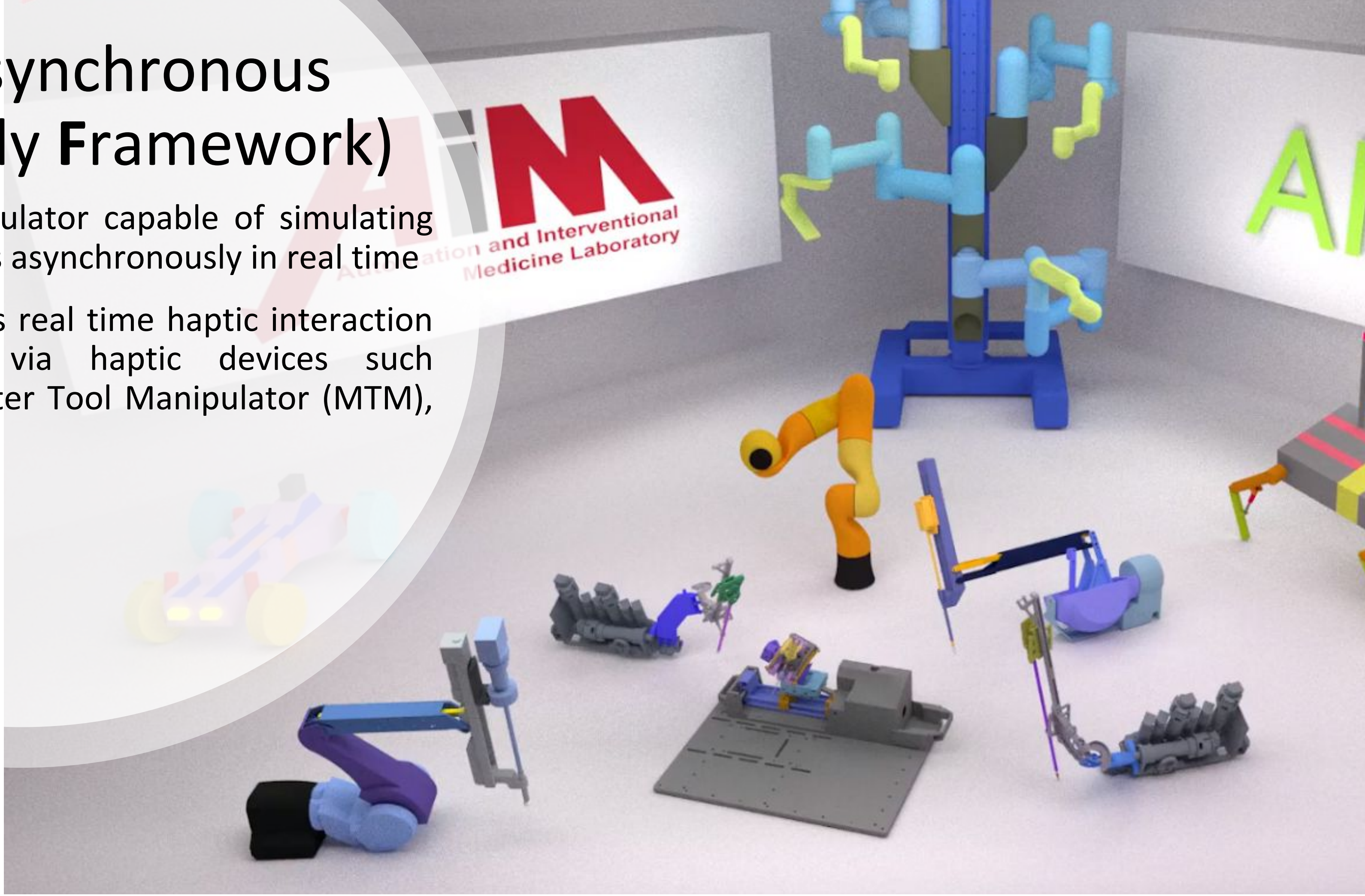


Master tool Manipulator (MTM) kinematics with Denavit-Hartenberg frames [7]

G. A. Fontanelli, F. Ficuciello, L. Villani and B. Siciliano, "Modelling and identification of the da Vinci Research Kit robotic arms," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, 2017, pp. 1464-1469.

AMBF (**A**synchronous **M**ulti **B**ody **F**ramework)

- AMBF is a simulator capable of simulating multiple robots asynchronously in real time
- It also provides real time haptic interaction with robots via haptic devices such as dVRK’s Master Tool Manipulator (MTM), Razer Hydras



Why AMBF!?

- Support for closed loop/parallel mechanisms (such as the dVRK MTM)
- Built-in position (PID) and effort (torque) controller
- Compatible with ROS and Python client
- Readily available robot models such as dVRK and KUKA lbr
- Uses YAML file format, a human-readable file format, for model description!

Robot representation (YAML vs. URDF)

YAML

- "**YA**ML is not a **M**arkup **L**anguage"
- Non-hierarchical model

  (i.e each link has its own parent and child)
- Can define closed loop mechanisms
- Can be used as a universal description format

URDF

- **U**nified **R**obot **D**efinition **F**ormat
- Hierarchical model

  (i.e single parent and multiple children)
- No support closed loop mechanisms
- Not a universal format

YAML file description

- Body

```yaml
BODY link1:
  name: link1
  mesh: link1.STL
  mass: 1.0
  collision margin: 0.001
  scale: 1.0
  location:
    orientation: {p: -0.0, r: 0.0, y: 0.0}
    position: {x: 0.0, y: 0.0, z: -1.197}
  inertial offset:
    orientation: {p: 0, r: 0, y: 0}
    position: {x: 0.0, y: -0.017, z: 0.134}
  friction: {rolling: 0.01, static: 0.5}
  damping: {angular: 0.95, linear: 0.95}
  restitution: 0
  collision groups: [0]
  color components:
    ambient: {level: 1.0}
    diffuse: {b: 0.0054, g: 0.2702, r: 0.8}
    specular: {b: 1.0, g: 1.0, r: 1.0}
    transparency: 1.0
```

- Joint

```yaml
JOINT base-link1:
  name: base-link1
  parent: BODY base
  child: BODY link1
  parent axis: {x: 0.0, y: 0.0, z: 1.0}
  parent pivot: {x: 0.0, y: 0.0, z: 0.103}
  child axis: {x: 0.0, y: 0.0, z: 1.0}
  child pivot: {x: 0.0, y: 0.0, z: 0.0}
  joint limits: {high: 2.094, low: -2.094}
  controller: {D: 2.0, I: 0, P: 1000.0}
  type: revolute
```

RBDL:

- **R**igid **B**ody **D**ynamics **L**ibrary is a C++ based dynamics library
- Available for use in python using the wrapper module
- Contains essential rigid body dynamics algorithms
    - Articulated Body Algorithm (ABA) for forward dynamics
    - Dynamics modeling using Recursive Newton-Euler Algorithm (RNEA)
    - Composite Rigid Body Algorithm (CRBA) for the efficient computation of the joint space inertia matrix
- Comprehensive support for both kinematic and dynamics modeling
    - Forward and inverse kinematics
    - Handling of external constraints such as closed-loop contacts
    - Forward and inverse dynamics, Jacobian, inertia , …

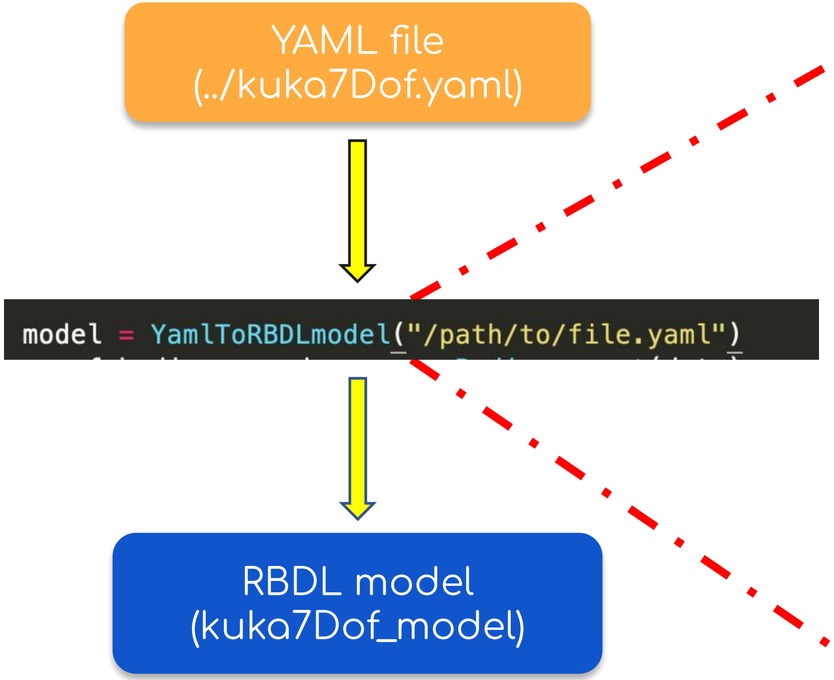
Challenge：NO Documentation ！！！

Ref: https://rbdl.bitbucket.io

YAML to RBDL model

- For calculation of the dynamics, RBDL takes “rbdl.model()” object as an input to access all the required information such as:
  - Mass, inertia, center of mass location, …
  - Description of the kinematic chain of the robot model
  - Joint and contact types
- Parameters from a YAML file need to be parsed into an RBDL model
- This conversion is handled by a custom-made parser class which constructs the RBDL model by receiving the YAML file as an input!

YAML to RBDL parser




```python
def YamlToRBDLmodel(data, Bodies, Joints):

    file_path = "/home/sonu/KUKA_7Dof/blender-kuka7dof.yaml"
    data = yaml_loader(file_path)
    model = rbdl.Model()
    model.gravity=[0,0,-9.81]
    no_of_bodies, random_var = Bodies_count(data)
    mass = get_mass_array(data, Bodies)
    mass_arr = np.array([[1],[1],[1],[1],[1],[1],[1],[1]])#good
    com_val = get_inertial_offset(data, Bodies)
    inertia_val = get_inertia_values(data, Bodies)
    parent_dist = get_parent_pivot(data, Joints)
    J_type = get_joint_type(data, Joints)
    joint_rot_z = rbdl.Joint.fromJointType ("JointTypeRevoluteZ")
    joint_fixed= rbdl.Joint.fromJointType ("JointTypeFixed")
    child_axes  = get_child_axes(data, Joints)
    parent_axes = get_parent_axes(data, Joints)

    for i in range(0, no_of_bodies):
      # Creating of the transformation matrix between two adjacent bodies
      trans = rbdl.SpatialTransform()
      if i == 0 :
        trans.E = np.eye(3);
        # print("R is\n", trans.E);
        trans.r = [0.0, 0.0, 0.0];
        # print("T is\n", trans.r);
        print("T is\n", trans);
      else:
        # get parent and child axes of the pair
        child_axis = dicttoVec(child_axes[i-1][0])
        parent_axis = dicttoVec(parent_axes[i-1][0])
        # Find the rotation matrix between child and parent
        r_mat = rot_matrix_from_vecs(mathutils.Vector(child_axis), mathutils.Vector(parent_axis))
        r_mat_np = MatToNpArray(r_mat)
        # print("R is", r_mat_np)
        trans.E = r_mat_np
        # print("R is\n", trans.E);
        trans.r = parent_dist[i];
        # print("T is\n", trans.r);
        print("T is\n", trans);

      # Creating Inertia matrix with just principle Inertia values
      I_x = inertia_val[i][0]
      I_y = inertia_val[i][1]
      I_z = inertia_val[i][2]
      inertia_matrix = np.array([[I_x, 0, 0], [0, I_y, 0], [0, 0, I_z]])
      # Creating each body of the robot
      body = rbdl.Body.fromMassComInertia(mass[i], com_val[i], inertia_matrix)

      # Specifying joint Type
      if i == 0:
        joint_type = joint_fixed
      else:
        joint_type = joint_rot_z
      # joint_type = rbdl.Joint.fromJointType(joint_name[i][0])
      print("joint type is", joint_type);
      # Adding body to the model to create the complete robot
      model.AppendBody(trans, joint_type, body)
      # print(i)

    return model
```

Just specify the YAML file path and Voila!

Gravity compensation

- Used to overcome torques generated by the robot's masses

Generalized equation of motion equation

$$\vec{\tau} = M(\vec{q})\ddot{\vec{q}} + V(\vec{q}, \dot{\vec{q}}) + G(\vec{q}) + \vec{\tau}_d$$

Using inverse dynamics given zero velocity and acceleration terms, we have:

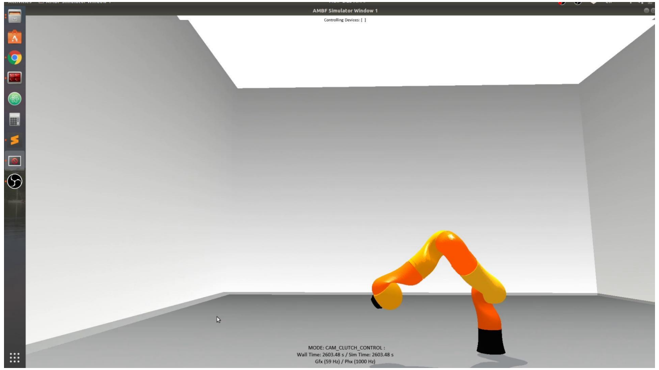
$$\boldsymbol{\tau} = G(q)$$

M: inertia matrix
V: velocity dependent matrix
τ_d: disturbance torques
G: gravity matrix

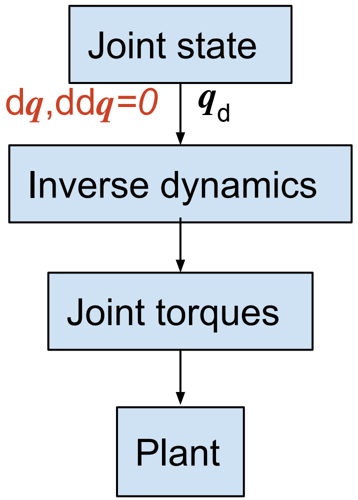
- Purpose
  - To test the accuracy of the dynamics model
  - Is further used for the impedance controller
- Criteria for the implementation
  - The robot should hold any given position
  - No need for use of any other controllers (meaning accurate model)

Gravity compensation (contd.)

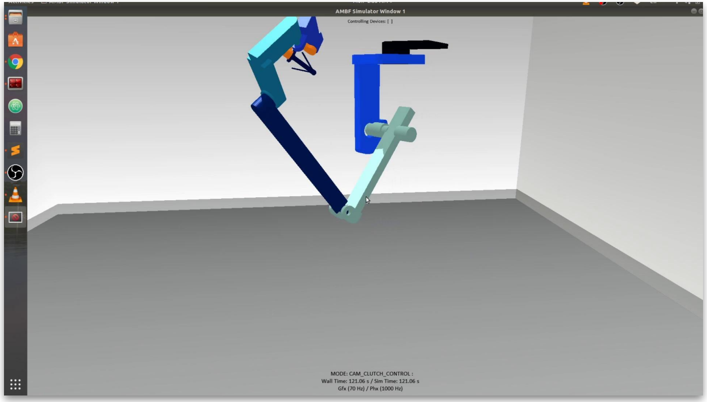
Gravity compensation for the KUKA arm:



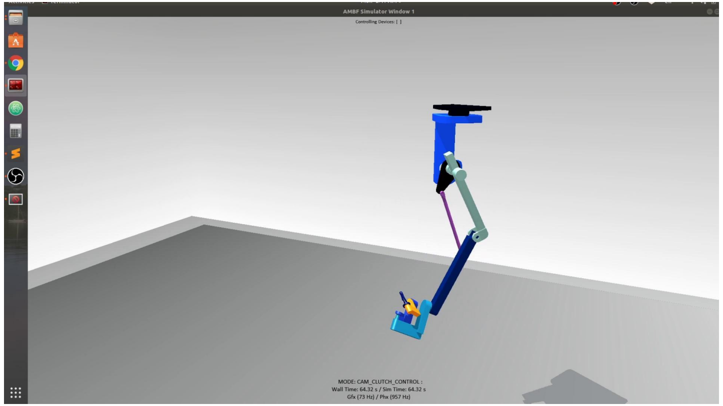
Gravity compensation



Gravity compensation for the MTM:



a) Serial MTM arm



b) Full MTM arm

Impedance controller

Goal:
Control the relationship between the robot motion and the contact force as required for the task.

Benefits:

- Simultaneous force and motion control
- Suitable for tasks involving human-robot interaction

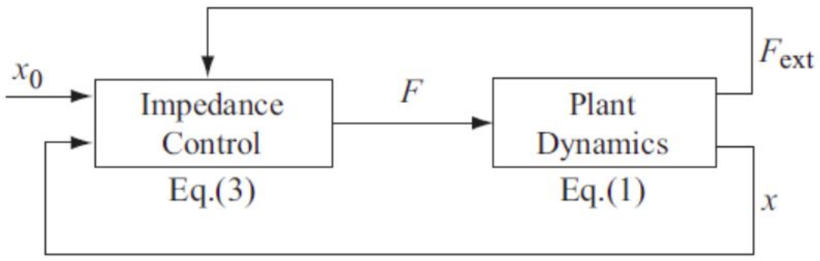
Impedance Control for Soft Robots*

P. Song, Y. Yu and X. Zhang, "Impedance Control of Robots: An Overview," *2017 2nd International Conference on Cybernetics, Robotics and Control (CRC)*, Chengdu, 2017, pp. 51-55
*Keppler, Manuel, et al. "Elastic Structure Preserving Impedance (ESπ) Control for Compliantly Actuated Robots." *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018.

Mathematical Formulation

$$m\ddot{x} = F + F_{\text{ext}}$$

$$F = \left(\frac{m}{M_d} - 1\right) F_{\text{ext}} + m\ddot{x}_0 - \frac{m}{M_d}\left(D_d\dot{e} + K_d e\right)$$



D_d : Damping Matrix

K_d : Stiffness Matrix

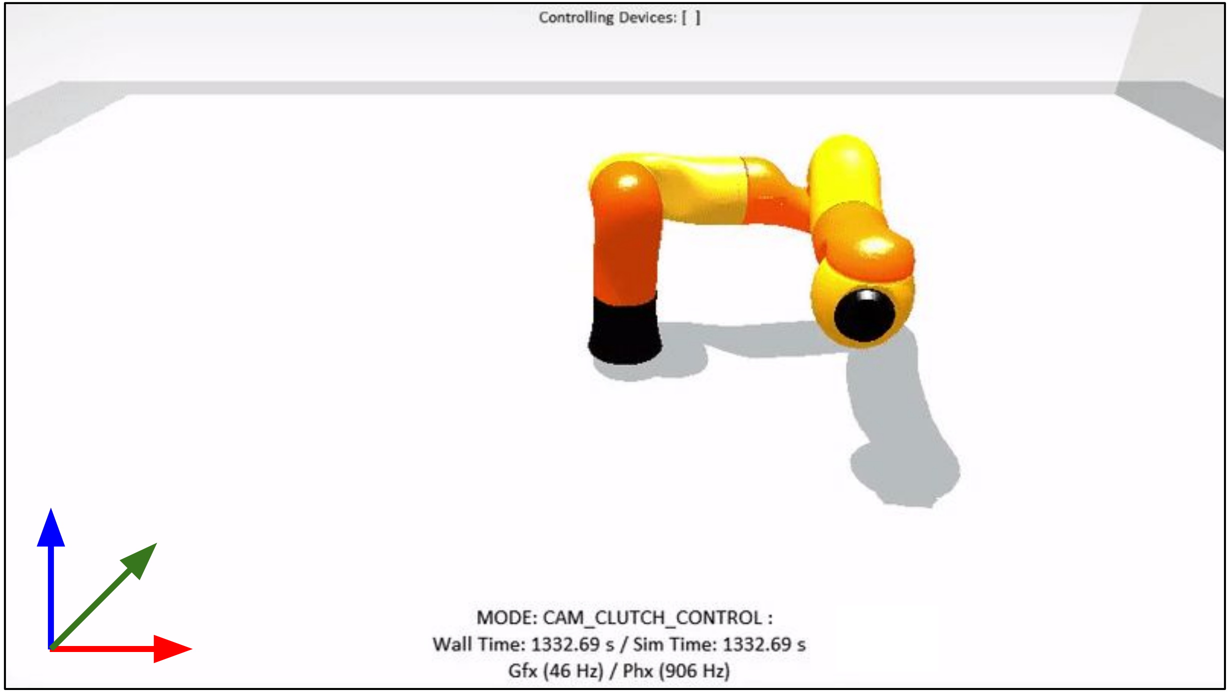
M_d :Desired Inertia Matrix

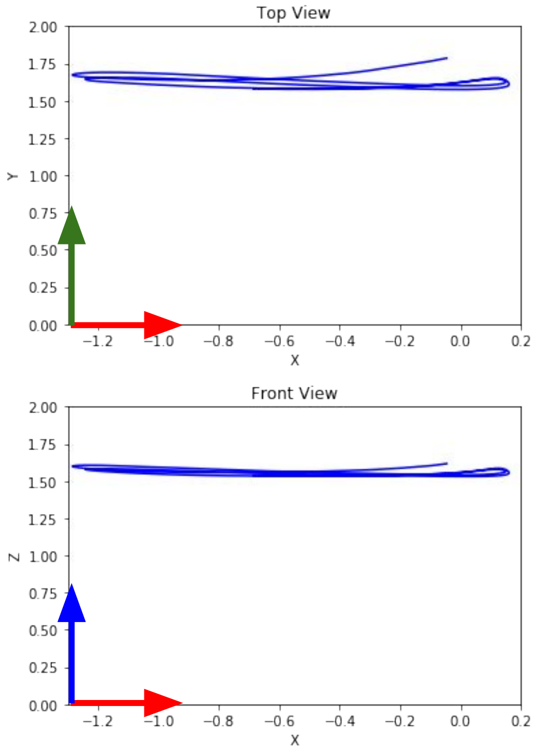
m : Inertia Matrix in Task Space, $m = (J^T)^{-1} M(q) J^T$

M(q): Inertia Matrix in Joint Space

Reference:
RBE-501 Robot Dynamics Lecture 13
Robotics 2 by Prof. Prof. Alessandro De Luca: http://www.diag.uniroma1.it/deluca/rob2_en.php

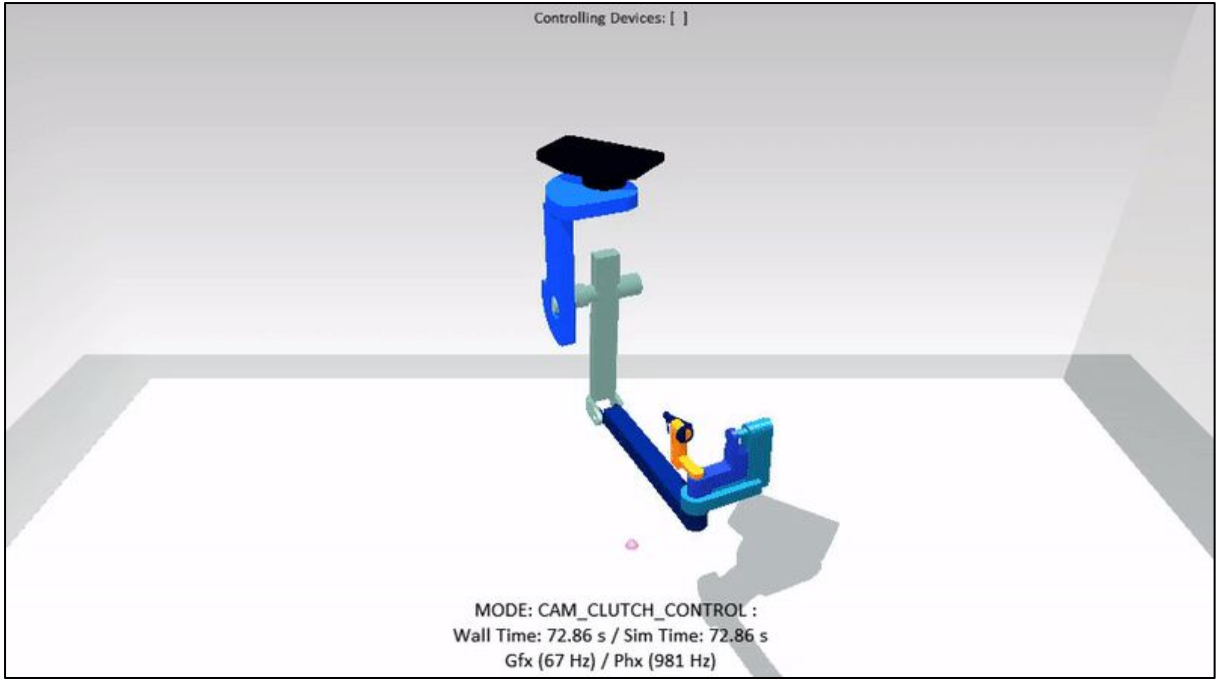
Trajectory Tracking via Impedance Control on KUKA-LBR



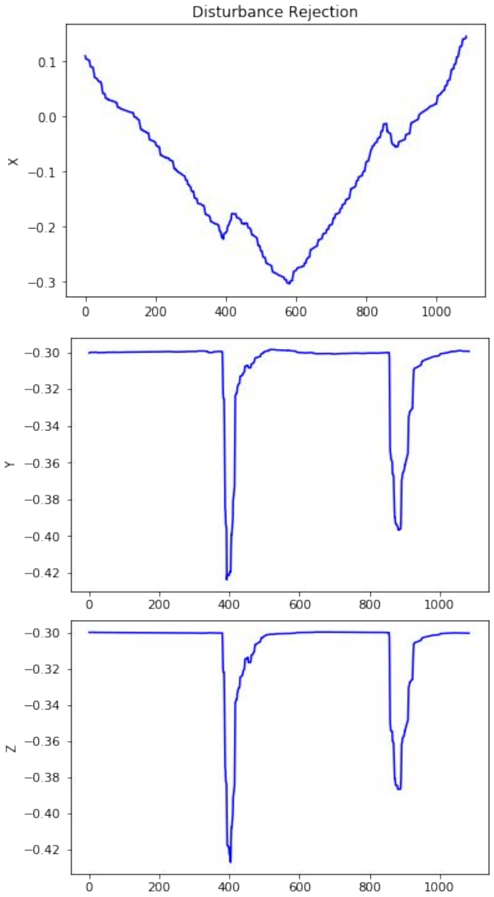


KUKA-LBR tracking linear trajectory along X axis

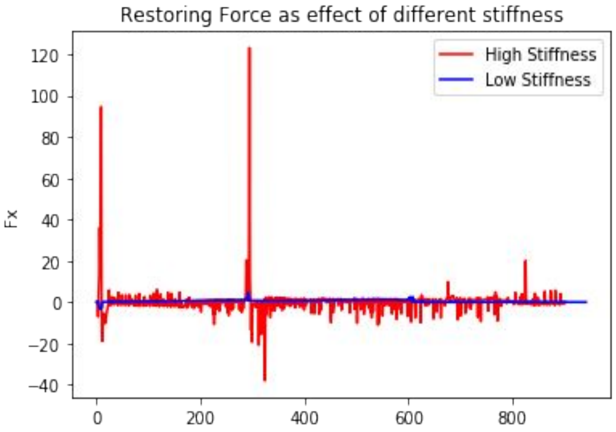
Rejection to External Perturbation

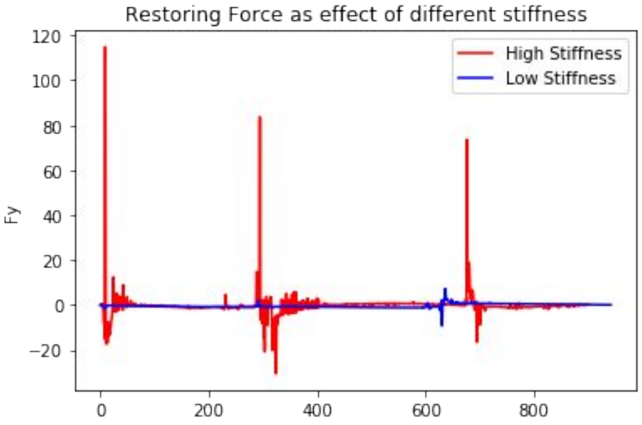


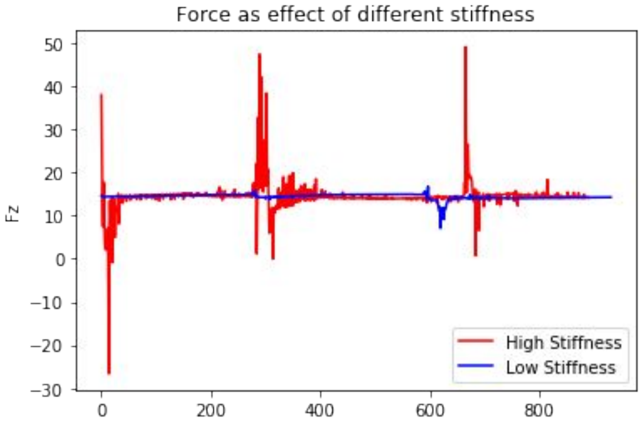
Disturbance Rejection via Impedance control for MTM



Force at the end Effector for different stiffness







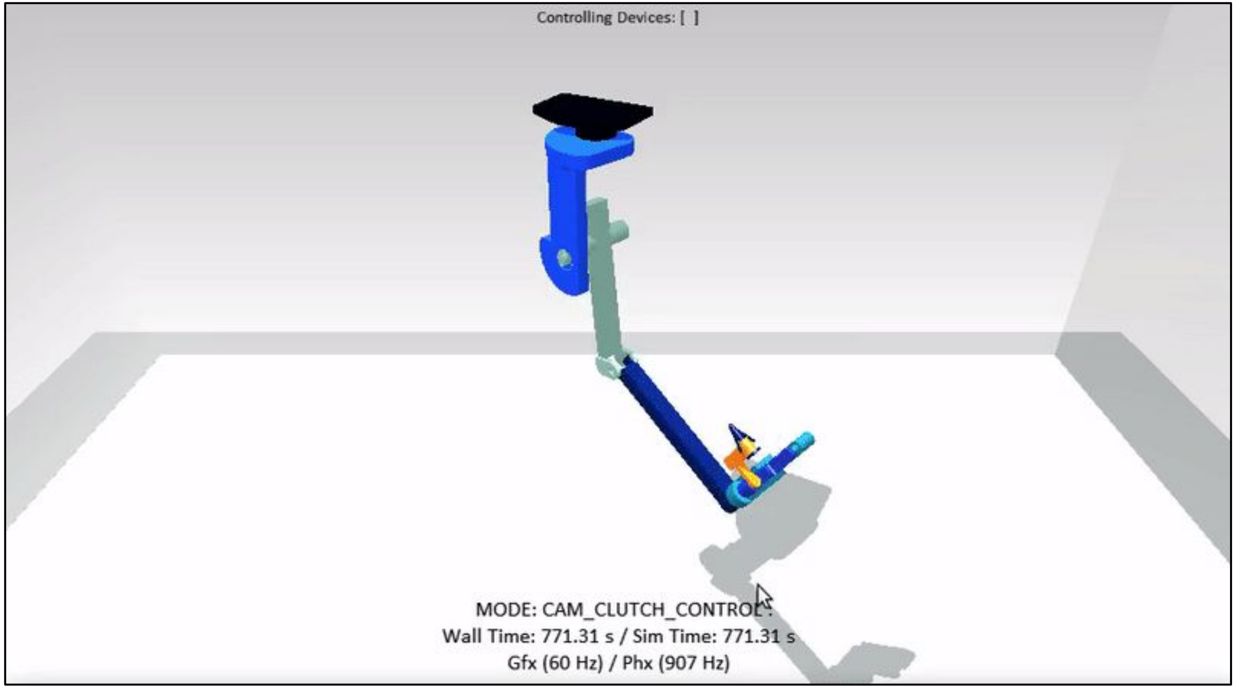
Gains:

K_p = 1

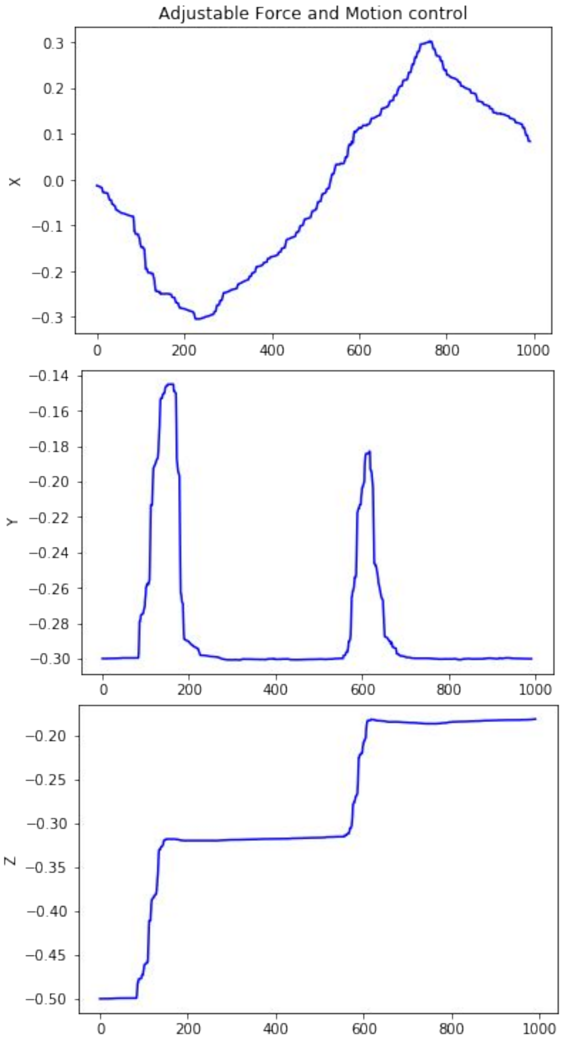
K_d = 0.1

M_d = 0.005

Control of Force and Motion Relationship

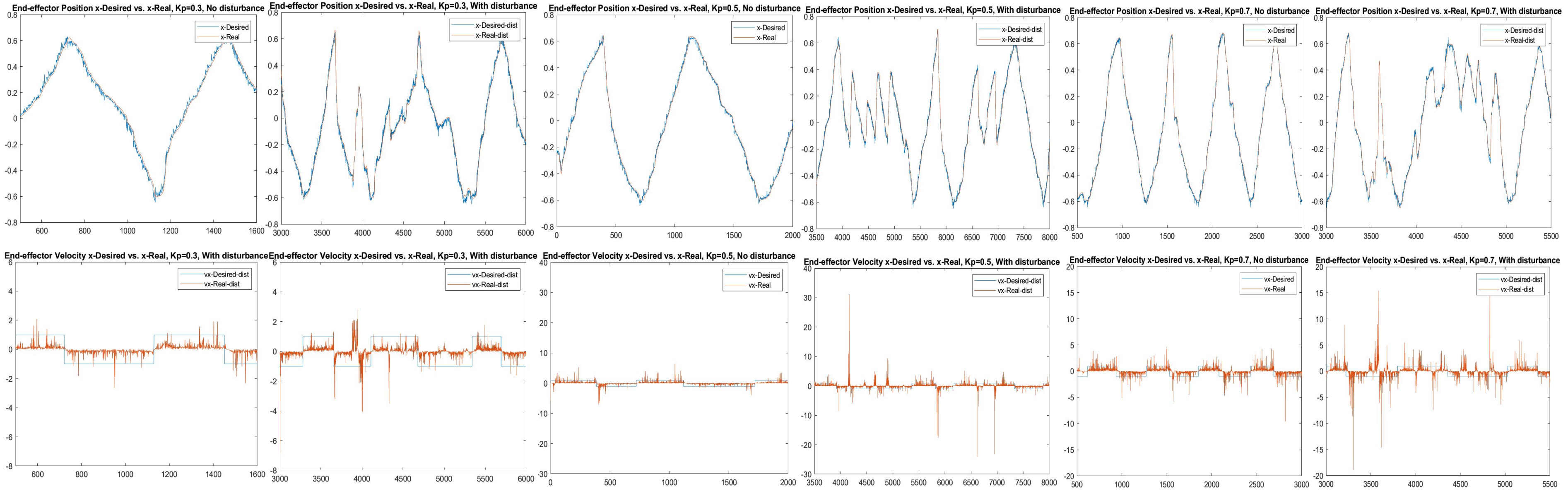


MTM tracking the linear trajectory along X axis while being flexible in motion along Z axis



KUKA-LBR Tip Position/Velocity Within/out Disturbance, Kp=0.3/0.5/0.7: Desired vs. Real

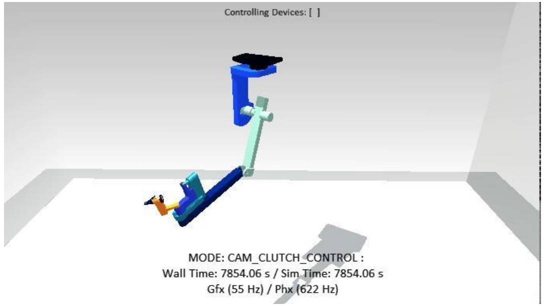
x-axis unit: ms
y-axis unit: m

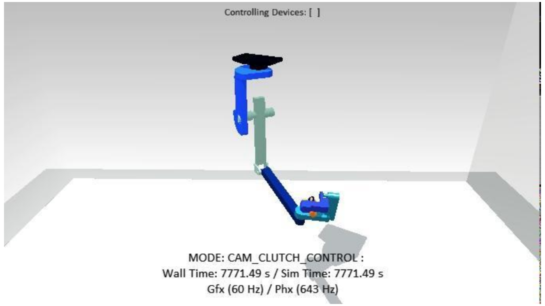


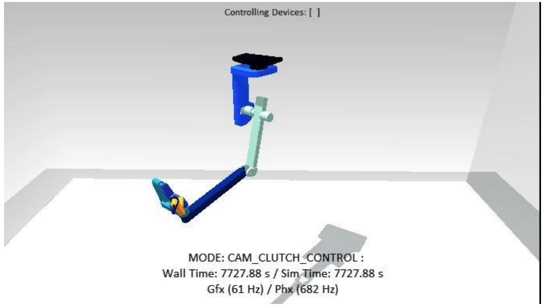
x-axis unit: ms
y-axis unit: m/ms

MTM impedance controller with Kp=0.3/0.5/0.7

X_desired = -0.4 m ~ +0.35 m
V_desired = 0.5 m/s







Kp=0.3
Kd=0.0001
Md=0.02

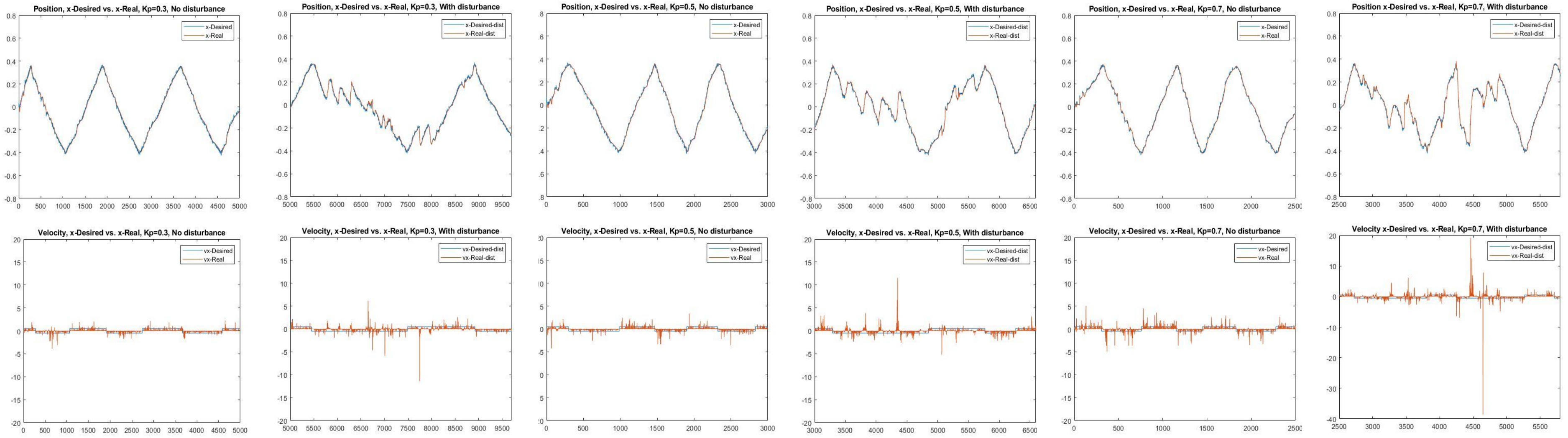
Kp=0.5
Kd=0.0001
Md=0.02

Kp=0.7
Kd=0.0001
Md=0.02

As Kp increases, feedback force and real-time velocity increase:
Advantage: response rate increase
Drawback: system easily being disturbed or unstable

MTM Tip Position/Velocity Within/out Disturbance, Kp=0.3/0.5/0.7: Desired vs. Real

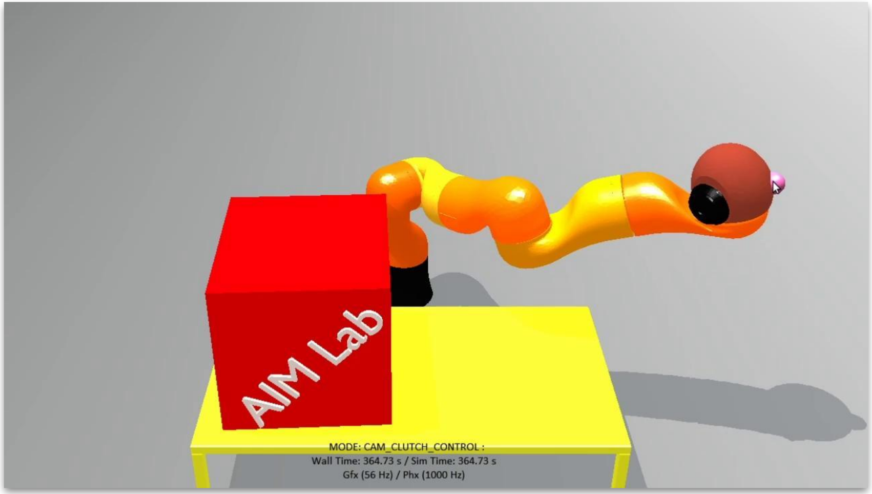
x-axis unit: ms
y-axis unit: m



x-axis unit: ms
y-axis unit: m/ms

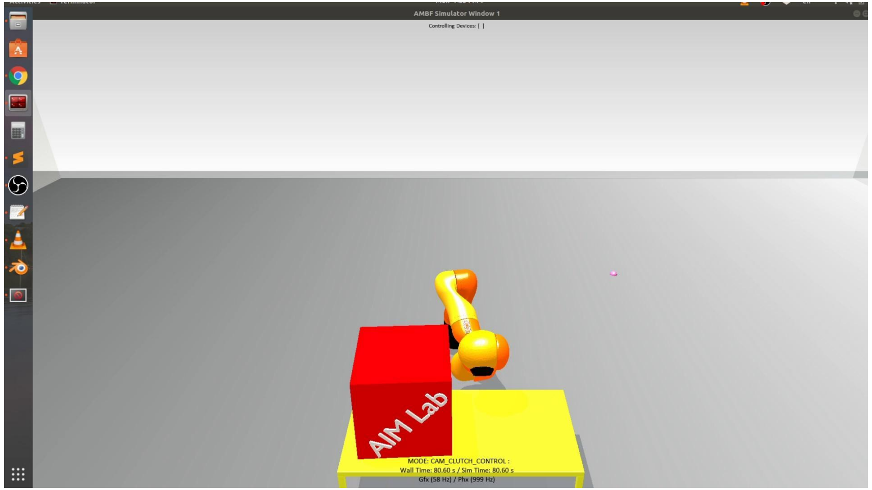
Performance comparison (PD VS. Impedance)

Compliance to obstacles:



Impedance controller with low stiffness in the x direction

- Ability to set desired stiffness of the end-effector in the cartesian space
- Low stiffness provides lower interaction force at the end-effector
- Desired when operating in environments where robot interaction with the surrounding environment is of importance



PD controller

- Lack of compliance when encountering an obstacle
- Relatively easier implementation
- Desirable in applications where fast response and short settling times are required

Summary

- ✅ Derive the dynamics models of MTM and KUKA arms
- ✅ Implement the impedance controller on MTM and KUKA arms
- ✅ Fully automated method to parse YAML file into model in RBDL
- ✅ Two fully documented working robot model examples with working dynamics and controllers for the AMBF simulator
- ✅ New function for the python API to get inertia of the bodies from AMBF
- ✅ Additional controller implementation gravity compensation and CTC

Acknowledgments

Special thanks to (Dr.)Adnan Munawar for his support and guidance throughout this project

References

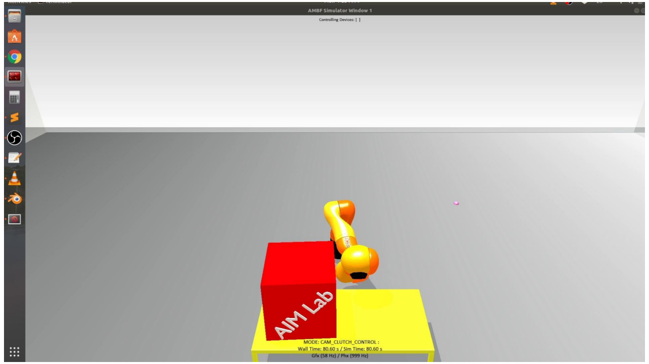
- [1] www.Intuitive.com
- [2] https://sites.google.com/site/davidvgealy/research
- [3] https://github.com/WPI-AIM/ambf
- [4] https://www.kuka.com
- [5] https://research.intusurg.com/index.php/Main_Page

Thank you!

Inverse dynamics controller

Computed torque controller (CTC)

- Given desired position, velocity, acceleration compute the required joint torques
- Error terms are added with the desired acceleration
- Purpose
  - Test the full dynamics model
  - Used for comparison with the impedance controller to highlight the differences



Kuka arm using Impedance controller following a trajectory in the Y direction

$$\vec{\tau} = M(\vec{q})\ddot{\vec{q}} + V(\vec{q}, \dot{\vec{q}}) + G(\vec{q}) + \vec{\tau}_d$$

$$u = M(q)a_q + V(q, \dot{q}) + G(q)$$

u = The set output (i.e. motor torque)

$\vec{a}_q = \ddot{\vec{q}}_d$ = The second derivative of the input command trajectory

$$a_q = \ddot{q}_d(t) - K_P e - K_V \dot{e}$$

$$e = q - q_d$$

$$\dot{e} = \dot{q} - \dot{q}_d$$

K_P: diagonal stiffness matrix
K_V: diagonal damping matrix
e: position error
de: velocity error

$$u = M(q)\left[\ddot{q}_d(t) - K_P e - K_V \dot{e}\right] + V(q, \dot{q}) + G(q)$$