Haptic Sensor Integrated Robot Arm

Kanishk Navale
Student Assistant-Haptic Intelligence
Max-Planck-Institut für Intelligente Systeme
70569 Stuttgart - DE

June 14, 2021

Contents

List of Figures

1 Robot & Physics Solver

1.1 Robot: 'nyu_finger'

The robot is constructed in accordance with the Open Dynamic Robot Initiative. The 'nyu_finger' is one of the robots. The initial robot driver [1] is developed by Julian Viereck licensed under BSD 3-Clause License with Copyright(c) 2020 Max Planck Gesellschaft, New York University. The below image emphasis on the build of the robot along with its kinematic description.

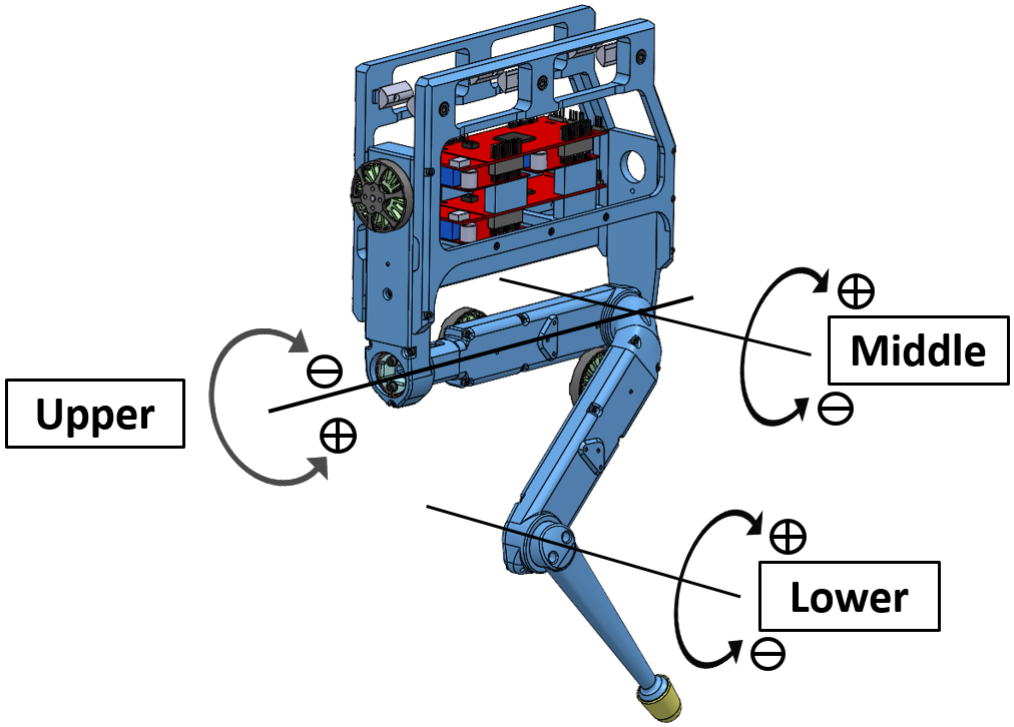

Figure 1: Illustration of the manipulator.

1.2 Physics Solver: Pinocchio

"Pinocchio instantiates the state-of-the-art Rigid Body Algorithms for poly-articulated systems based on revisited Roy Featherstone's algorithms. Besides, Pinocchio provides the analytical derivatives of the main Rigid-Body Algorithms like the Recursive Newton-Euler Algorithm or the Articulated-Body Algorithm."
source: stack-of-tasks/pinocchio [2]

The code library contains limited detailed description for some functions. This can cause few issues during the implementation of the robot functionalities.

2 Kinematics Solver

The Pinocchio library constructs the robot kinematic-tree using the .urdf files in the 'nyu_finger' repository. The library comes with several functionalities solving the same tasks. The robot needs a reference frame to compute the transformation. This is solved by invoking a visualizer & approximating frames used by the robot. The correct kinematic function is found by trying out all the similar functions & counter-checking it mathematically.

2.1 Visualizers

2.1.1 MeshCat

The 'MeshCat Visualizer' comes equipped with the Pinocchio. This visualizer runs in a web browser. As the robot needs the 'sudo' privileges in the Linux machine to run the visualizer could not run in a web browser anymore due to the limitations of the visualizer.

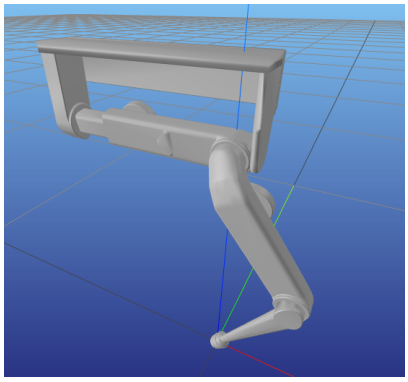

Figure 2: Rendering in 'MeshCat Visualizer'.

2.1.2 NYUFingerSimulator

The 'NYUFingerSimulator' is used to simulate the robot. With a few code tweaks the simulator is turned into visualizer solving the issue with the previous visualizer.

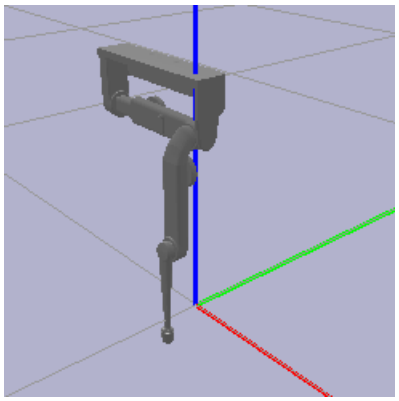

Figure 3: Description of the simulator.

2.2 Inverse Kinematics

The inverse kinematics for the robot is implemented based on optimization. The algorithm in the below figure explains the optimization and method in much detail.

- Assume initial posture q_0. We want to reach a desired endeff position y^* in T steps:

Input: initial state q_0, desired y^*, methods ϕ^{pos} and J^{pos}
Output: trajectory $q_{0:T}$
1: Set $y_0 = \phi^{\text{pos}}(q_0)$ *// starting endeff position*
2: **for** $t = 1 : T$ **do**
3: $y \leftarrow \phi^{\text{pos}}(q_{t\text{-}1})$ *// current endeff position*
4: $J \leftarrow J^{\text{pos}}(q_{t\text{-}1})$ *// current endeff Jacobian*
5: $\hat{y} \leftarrow y_0 + (t/T)(y^* - y_0)$ *// interpolated endeff target*
6: $q_t = q_{t\text{-}1} + J^{\sharp}(\hat{y} - y)$ *// new joint positions*
7: Command q_t to all robot motors and compute all $T_{W \to i}(q_t)$
8: **end for**

Figure 4: Description of the Inverse Kinematics solver.

source: Introduction to Robotics - Marc Toussaint

2.3 Robot Teleoperation using Keyboard

'pygame' is a pyhton3 library for developing games. Moreover, it provides a realtime interface modules for the Keyboard. The event handlers for the Keyboard is much stable with it. The inverse kinematics is mapped to certain keys moving the robot relatively in the cartesian space with fixed step distances. This functionality is tested in the simulator. In the upcoming phase, the keyboard will be replaced by the haptic sensor. The sensor is capable to move the robot in the cartesian space with varying velocity or force.

3 Dynamics Solver

3.1 Inverse Dynamics Compensation

As the robot actuators are BLDC motors they are not equipped with the brakes. The moment the robot stops receiving joint commands the robot loses it's configuration due to gravity. To solve this, Inverse Dynamic terms like Gravity, Corrolis, Centrifugal and Inertial Compensations are used to compute joint torques to preserve the robot stance. The joint commands is issued at an rate of 60Hz.

3.2 Position & Velocity Control

For controlling the robot's position and velocity seperate PID loops are implemented. As per the experiments PID based controller is not yielding desired results with varying steady state error inspite having individual gain vectors for each actuators. The motor driver expects a joint command every 1ms posing an issue in realtime code performance. Presently, PID based controller is being ported to LQR based controller for better results.

4 Robot Class

The robot class implements all the functionalities into an object. There are few tasks like Inverse Dynamics Compensations & Updating the simulator running in the background. Overview and brief description of robot capabilites are as follows,

1. Simulator,
   Reads the positions of robot joints in realtime & updates the visualizer at rate of 60Hz.

2. Inverse Dynamic Compensations,
   Preserves the robot stance against the gravity & external forces by issuing compensating torques at the rate of 1000Hz.

3. Teleoperation,
   If enabled, the robot can be moved using keyboard. The 'pygame' runs at an rate of 1Hz.

4. Reading Haptic Sensor,
   The robot reads the sensor at the rate of 1000Hz.

5. Point-to-Point Motion Functionality,
   This functionality is extended from the Inverse Kinematics Solver. The robot can between two or more points in the cartesian space in Joint or Linear Travesial Mode. It also 3D plots the position of the TCP if enabled.

4.1 Handling Relative Encoders

The actuators is equipped with the relative encoders meaning during power cycle the actuator's states are lost. For time being, the robot is crudely positioned at an zero pose and the offsets are biased. This procedure sometimes causes a mismatch in computing terms for robot dynamics. Another way to turn them into absolute encoders is to implement a CMOS at a hardware level.

4.2 Pipelining Model & .urdf File Updates

As the .urdf file encompasses dynamic parameters like as mass, interia etc. any changes in the hardware of the robot needs calibration in the code side. The better way to get around this is to build an pipeline which translates 3D model to .urdf files to hardware changes. The pipeline can be constructed using services in the project management systems. This eliminates the calibration procedure in the codebase everytime when systems engineering components are added to robots.

5 Haptic Sensor

The sensor is based of capacitance sensor array of $\mathbb{R}^{n \times n}$. The robot reads 21 channels of data from the sensor using serial port. The serial port is dynamically searched by the robot. The data can be parallely visualized in chart & matrix form as described by the following image,

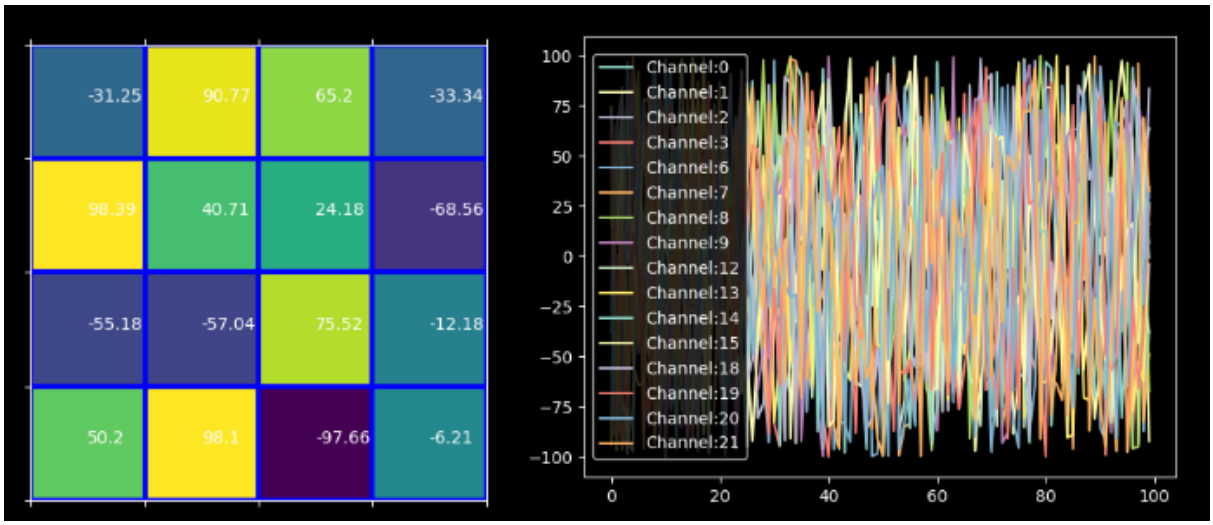

Figure 5: Illustration of the sensor monitoring.

6 Upcoming Tasks

1. Test the LQR Controller.
2. Implement Impedence Controller working alonside with the sensor.
3. Improve code performance.
4. Implement CMOS based Encoders. (To be discussed)
5. Converting Sensor reading into force vector of $\mathbb{R}^3$

References

[1] Felix Grimminger, Avadesh Meduri, Majid Khadiv, Julian Viereck, Manuel Wüthrich, Maximilien Naveau, Vincent Berenz, Steve Heim, Felix Widmaier, Thomas Flayols, Jonathan Fiene, Alexander Badri-Spröwitz, and Ludovic Righetti. An open torque-controlled modular robot architecture for legged locomotion research. *IEEE Robotics and Automation Letters*, 5(2):3650–3657, 2020.

[2] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The pinocchio c++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *2019 IEEE/SICE International Symposium on System Integration (SII)*, pages 614–619, 2019.