Implementation of Brain-Machine Interface Commands on Baxter Robot System

By

Lana Pantskalashvili

April 2022

Introduction

The document describes the control of the robotic manipulator using BMI datasets. Particularly, velocity values are received from the monkey's motor cortex during a reach-to-grasp task (*"Massively parallel recordings in macaque motor cortex during an instructed delayed reach-to-grasp task," Brochier*) and transferred to the assistive device for imitation purposes. To imitate the movement, the research version of the Baxter robot is used from Rethink Robotics.

Presented is the background outlining the used software, description of the Baxter robot system in terms of its necessity for a successful control, followed by the methods of movement implementation, and system validation by analyzing collected data during the experiments.

Background

The research version of the Baxter robot is controlled via a workstation allowing researchers to execute custom programs on Baxter.
The software necessary to control the robot is:

- Baxter Software Developers Kit (SDK) workstation
- The Operating System: Ubuntu 14.04.2
- Robot Operating System (ROS) Kinetic
- Python.

Additional development system components include:

- Docker virtual filesystem
- Baxter Simulator (Gazebo).

Description of Baxter Robot

Baxter is a humanoid robot with two 7-degree-of-freedom arms. The 7th joint provides additional kinematic redundancy and makes the robot over-actuated. This feature was the main decisive factor in the selection of the correct algorithm to execute the experiments.

Additionally, Baxter's arm can be operated in four control modes: position, "raw" position, velocity, and torque. The last three modes are advanced control modes and require additional safety precautions which is another significant detail since in this case, a velocity controller and a "raw" position controller are considered to control Baxter's limbs.

Methods

The task is to utilize ≤5 seconds worth of data while submitting joint velocity commands to Baxter so that it will imitate monkey's hand movement starting with the initial "home" position on the switch, reaching target object and finally, returning to the initial position.

Based on the design of the task, manipulator's initial right-hand pose was manually set in the direction of motion during the experiment by using Baxter's Zero-G mode, meaning the arm was moved across and joint angle data was recorded from Inverse Kinematics Solver using end-effector pose.

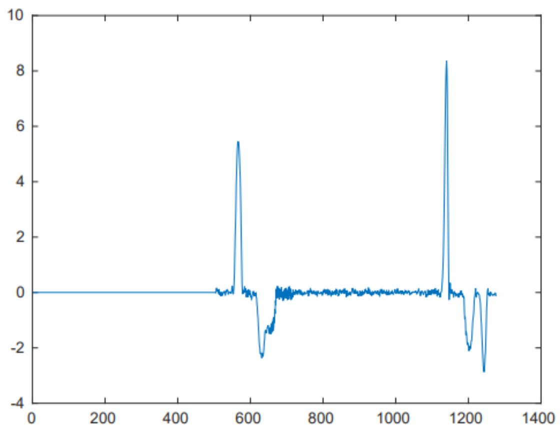

Fig. 1 – "Raw" decoder data

The data known during the experiment is end-effector velocity. To publish precise commands to the robotic manipulator, the linear twist V_x needs to be translated into joint velocities from cm/s. to rad/s. To control the robot, two advanced control modes were experimented: joint velocity (Gazebo simulator) and "raw" joint position control mode (real Baxter).

For cartesian control, traditional robots require inverse kinematics, specifically Jacobian matrix:

$$X = J(q)\dot{q},$$

where X is the column matrix with end-effector velocities, $\dot{q}$ is the column matrix with desired joint velocities, and J(q) – manipulator Jacobian, a function of the current pose and a matrix that maps end-effector velocities to joint velocities.

However, Baxter has 7-DOF limbs and consequently, the Jacobian matrix is not a square matrix (6 x 7). The solution is to use the pseudoinverse method, also referred to as Moore-Penrose Inverse of J:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{+}\boldsymbol{X}.$$

$\boldsymbol{J}^{+}$ - Jacobian matrix obtained by pseudoinverse. $\dot{\boldsymbol{q}}$ is the desired 7 x 1 matrix with joint velocities (dq_n, *n standing for the total DOF*) in rad/s.:

{'right_s0': dq_1, 'right_s1': dq_2,
'right_e0': dq_3, 'right_e1': dq_4,
'right_w0': dq_5, 'right_w1': dq_6, 'right_w2': dq_7}.

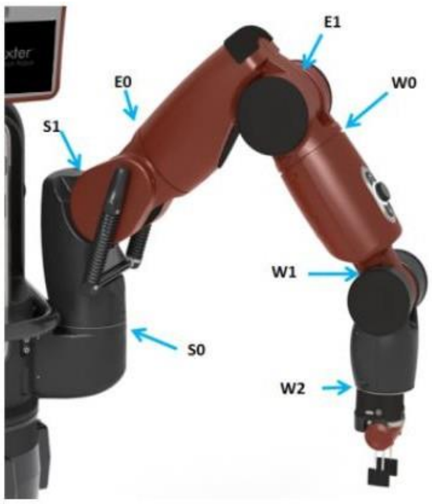

Fig. 2 – Baxter 7-DOF arm

X is the 6 x 1 matrix where the first three elements of the matrix represent linear velocities, the last three elements stand for angular velocities of the end-effector: $[V_x, V_y, V_z, \omega_x, \omega_y, \omega_z]^T$.

Since the only data known to us is the end-effector velocity along X-axis, the matrix has the following form:

[bmi_data[*index*], 0, 0, 0, 0, 0]^T

and is expected to move only in X direction.

Validation

1. Joint velocity control mode

When controlling the Baxter robot in advanced control mode, there are several details to pay attention to, including maintaining controller frequency to monitor the rate of publishing data to the arms and making sure the robot system is not overexerting itself by collecting data and analyzing it. During the velocity control mode, safety features are turned off, therefore joints moving at high velocities could cause Baxter to crash into an obstacle resulting in substantial damage to both the obstacle and the robot.

Joint data was collected during testing the functionality in the Gazebo simulator. To collect the data, ROS subscriber was used to subscribe to *JointState* listener published on the topic: */robot/joint_states* so that ROS would constantly (1 KHz.) publish messages holding sensor data describing the states of joints. The data is retrieved using the *callback* method. In figure 3, collected joint data is plotted, including all joint positions (rad.), velocities (rad/sec.), and efforts (Nm.). The horizontal axis denotes the state of the current BMI data cell (each is 10ms.). As it can be observed, velocity and torque values are changing rapidly. Implementation of the “raw” decoder data on the real robot impractical due to unexpected uncontrollable movements of the robot.

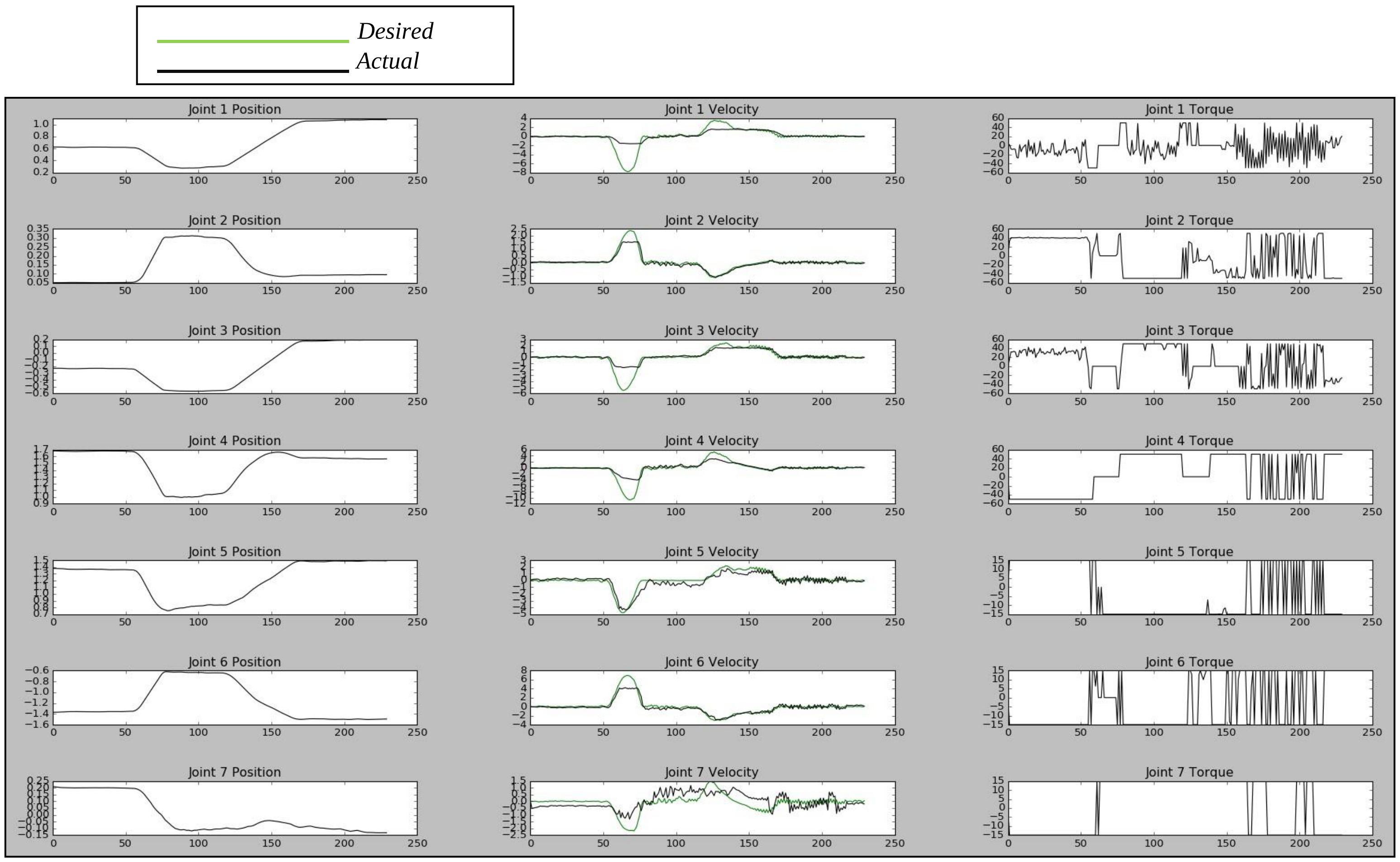

Fig. 3 – Joint data from Baxter before modifications

It is important to note that the first four joints (shoulder and elbow) can have a maximum input torque of 50 Nm., the remaining three joints (wrist) can be manipulated by up to 15Nm. amount of input torque. The graphs demonstrate that collected joint effort data (very rapidly) is approaching (or equal to) the maximum allowed torque values that the Baxter robot is manufactured with. Rapid undesirable changes can be seen in joint velocity graphs, as well, especially for the wrist joints (5, 7). Shortly, the robot does not have enough workspace to move freely by commanding such high velocities and it is moving in numerous unwanted directions (Fig. 4).

Figure 4 shows the displacement of the end-effector (m/s.) during the experiment when “raw” decoder data was used. End-point data was collected by subscribing to Baxter’s *Endpoint* listener that is published on */robot/limb/<side>/endpoint_state* topic (100 Hz.). Similarly to the previous graphs, the horizontal axis denotes the state in the BMI dataset.

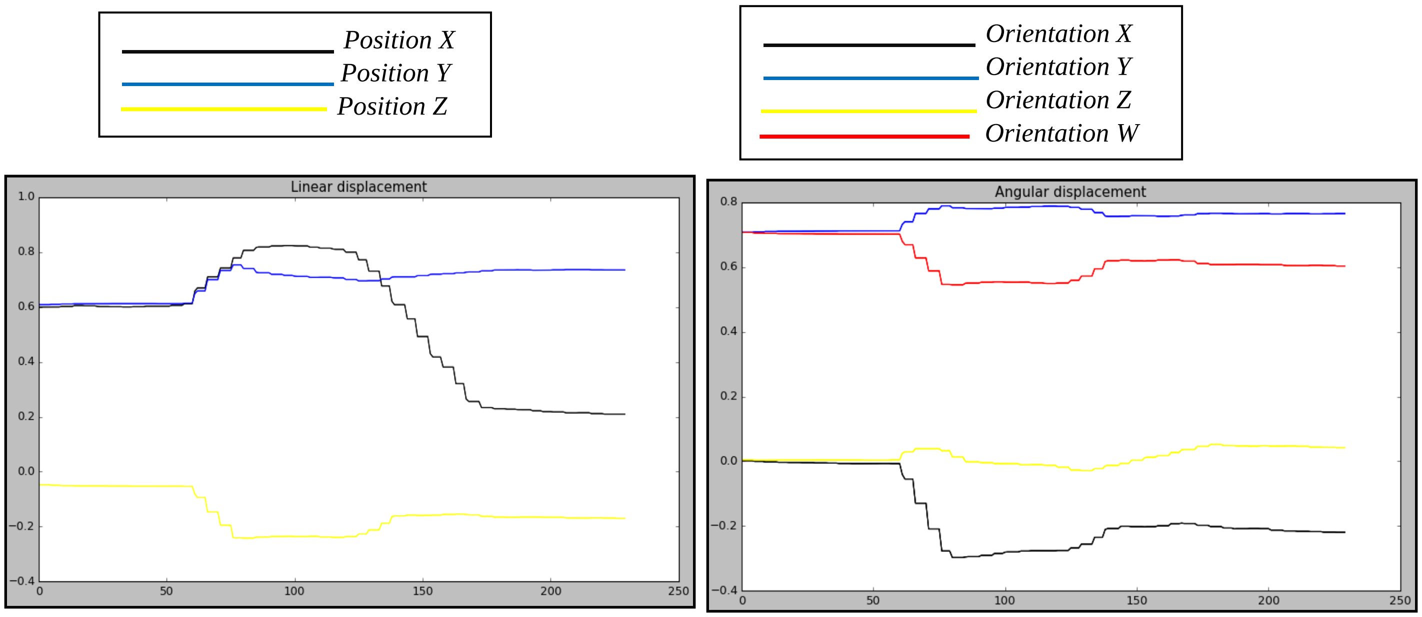

Fig. 4 –End-effector data from Baxter before modifications

To control the system, the solution is to scale down velocities and supersample dataset using interpolation. This is a way to maintain the same controller frequency (100Hz.) throughout the procedure. It should be noted that this increases the duration of the task.

It was decided to scale velocities down by a factor of 30. As for the interpolation, the factor is 6. Therefore, the system is slowed down 6 times.

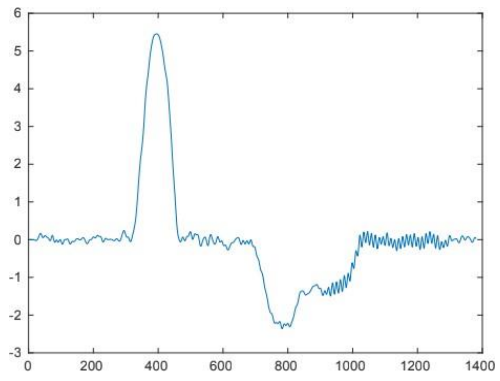

Fig. 5 – Decoder data after interpolation*

**To make demonstration of the procedure more interesting, some of the periods when no motion is taking place, are ignored.*

Joint sensor data obtained from Baxter robot after dataset interpolation and velocity scaling looks less "severe" but not valid enough for the mode to be tested on the real robot. Despite the movement being desirable, data analysis proves its erroneousness.

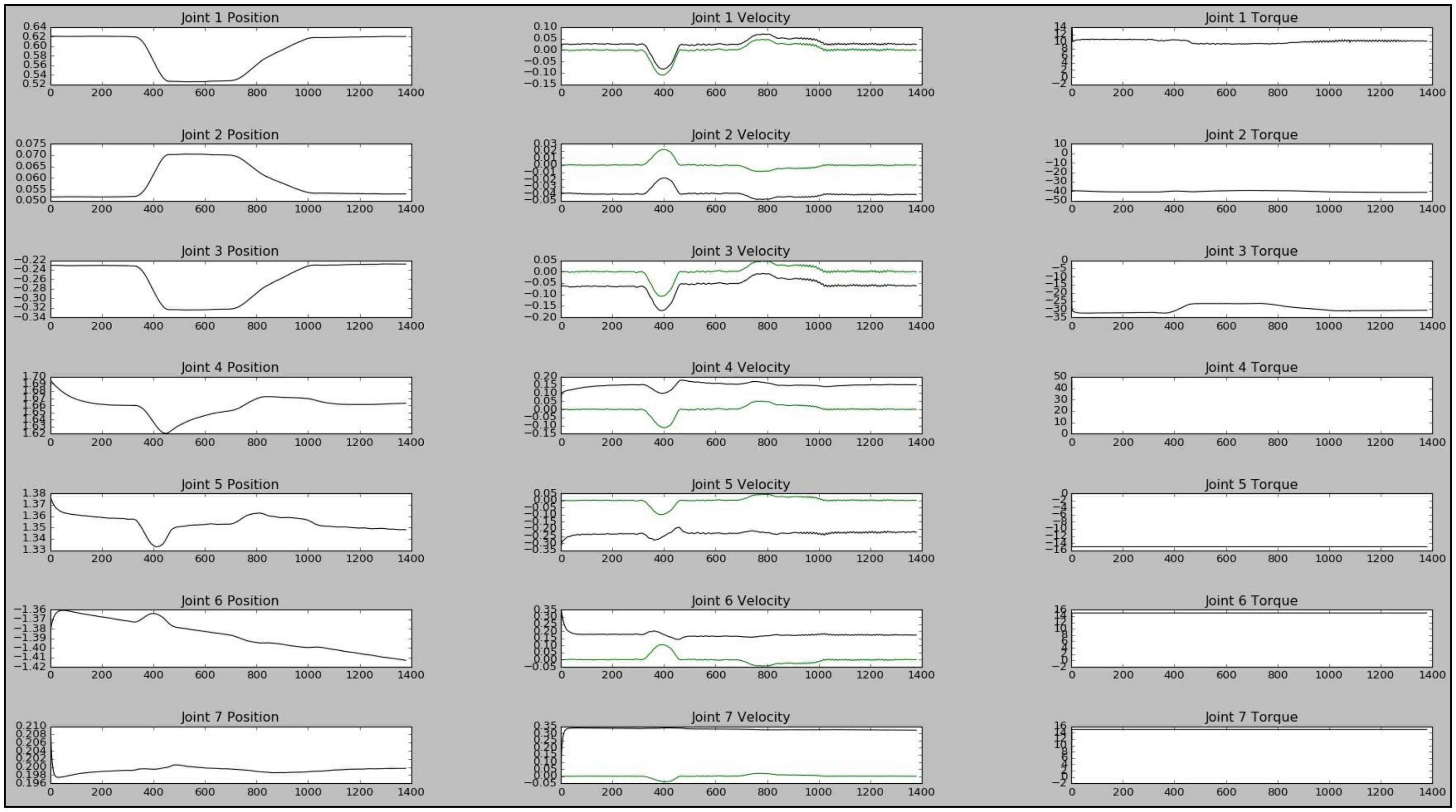

Fig. 6 – Joint data from Baxter after modifications

Similarly, the end-effector data is "smoother" but not exactly what it is supposed to be according to the original task. The end-effector is not following planned cartesian trajectory and is moving outside expected workspace for which one reason can be unfortunate selection of initial position. In other words, the arm is unable to freely accept velocities, it is trying to move outside X-axis to compensate for that. For the future, the implemented mode has a potential to be used if expected positions are provided from the BMI.

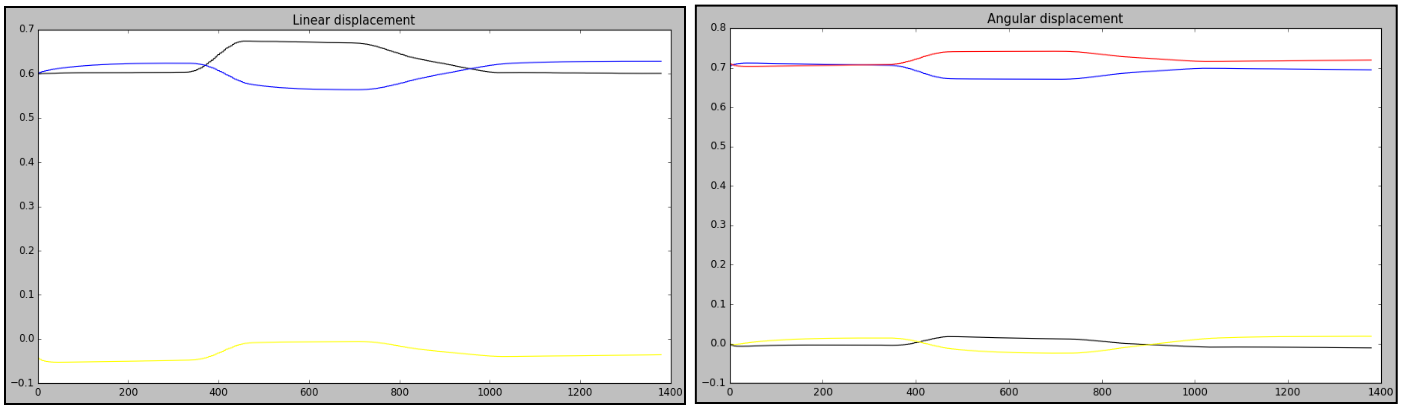

Fig. 7 –End-effector data from Baxter after modifications

2. “Raw” joint position control mode

Because of the *blocking* nature of the joint position control mode, “raw” joint position control mode was used. It allows a much more direct position control mode than a regular position control mode since the commands are sent directly to Joint Controller Boards (JCBs).

By using this specific mode, we are switching from commanding Baxter velocities to defining desired trajectory for it. To calculate trajectory, the following steps must be followed:

1. $\dot{q}_r(t) = J^+(q_r(t))X_r(t)$

2. $q_r(t+\Delta t) = q_r(t) + \Delta t\dot{q}_r(t)$

q_r – reference position, X_r – reference end-effector velocity, $\dot{q}_r$ – reference joint velocity, Δt – 10ms. respecting original dataset.

Joint data collected from the robot demonstrates the correctness of the implementation. Joint positions obtained from sensors are following commanded desired positions. Velocities are more or less within reasonable range and similar to desired velocities plotted previously, as for the efforts, the peak values are minimized. To make information more human-readable joint positions are converted from radians to degrees; case is similar for velocities, plotted graphs show deg/sec. measurements.

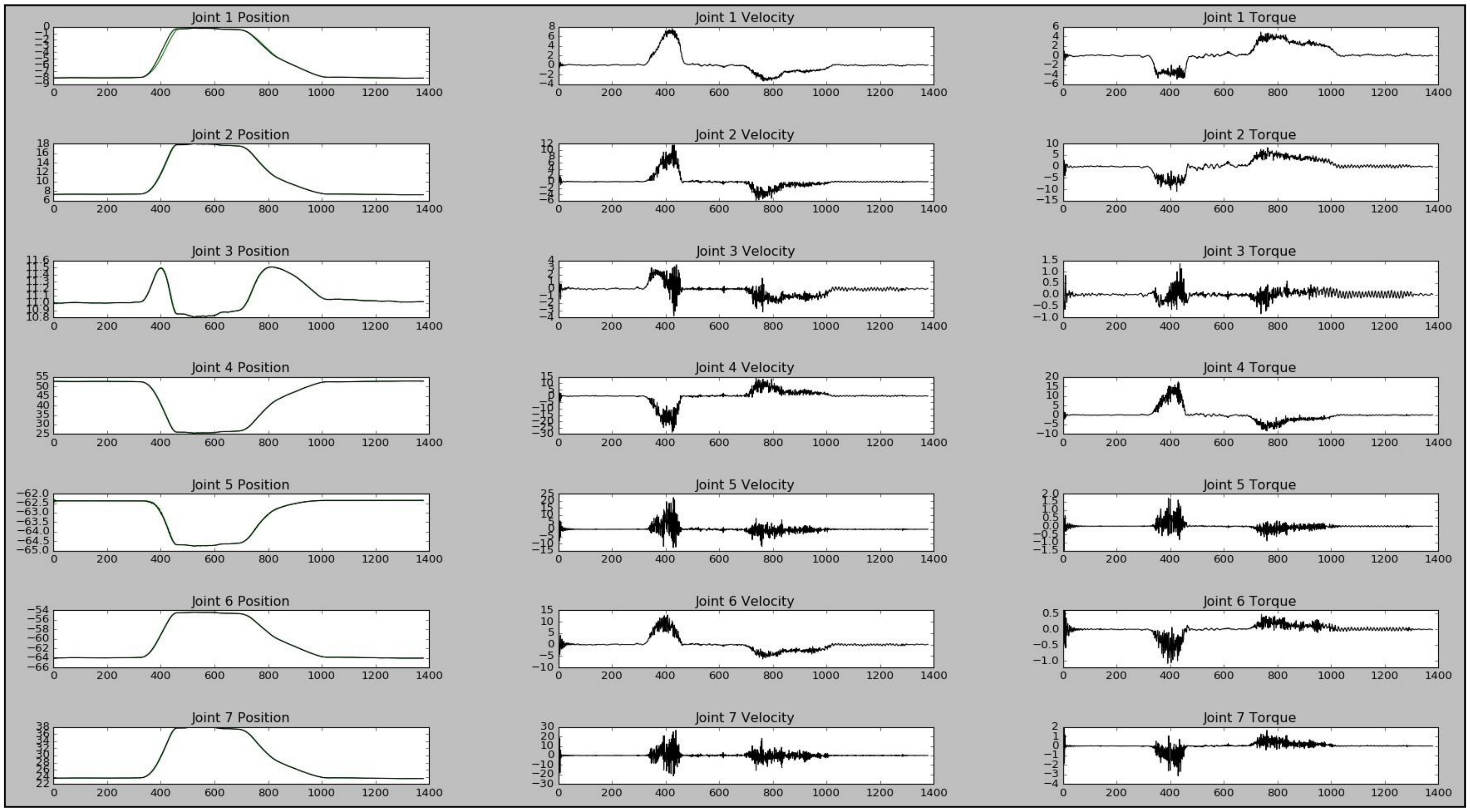

Fig. 7 – Joint data from Baxter, “raw” position control mode

The end-effector data is valid, as well because displacement is happening along X-direction only with negligible changes in Y-direction and orientation.

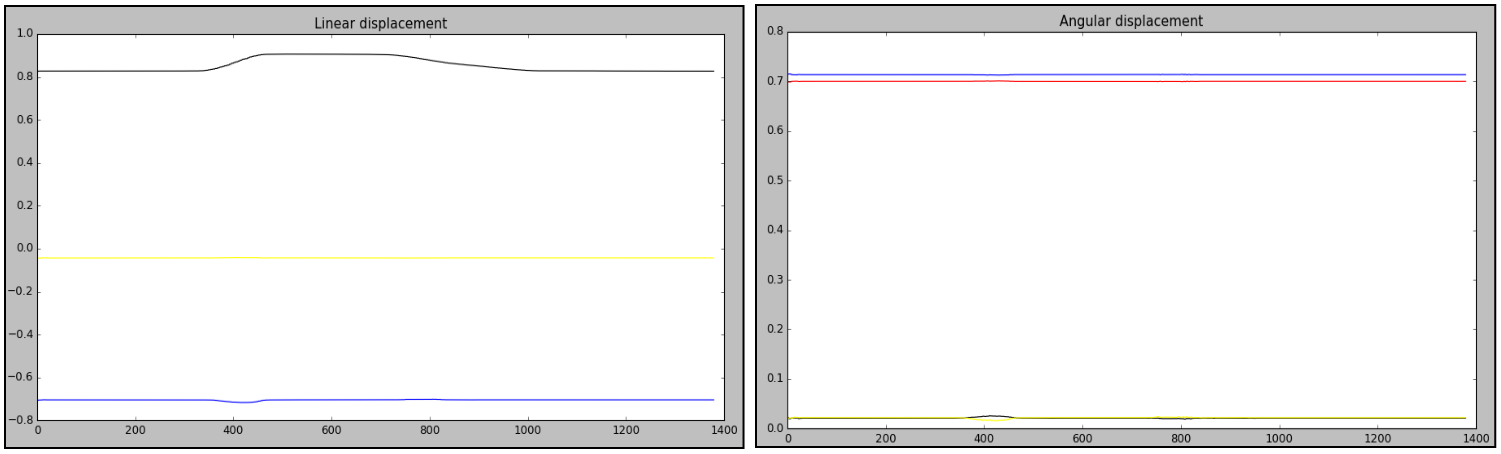

Fig. 8 – End-effector data from Baxter, “raw” position control mode

The “raw” joint position control mode was used to test the experiment on the Baxter robot in the Dynamic Systems and Control Laboratory (DSCL). Data obtained from “real” Baxter is almost identical to the data obtained through simulations beforehand.

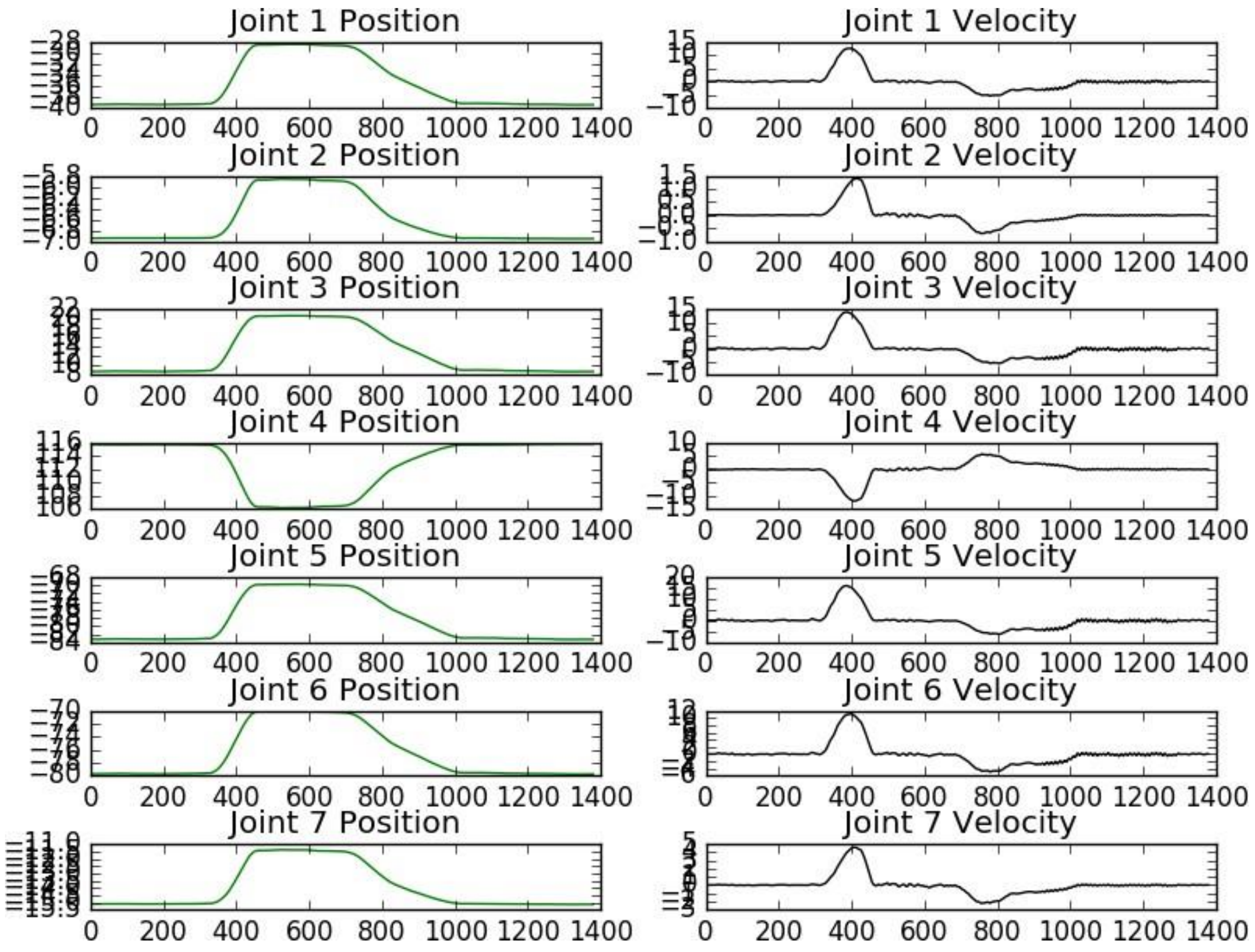

Fig. 9 – Expected joint data from Baxter (DSCL, SDSU)

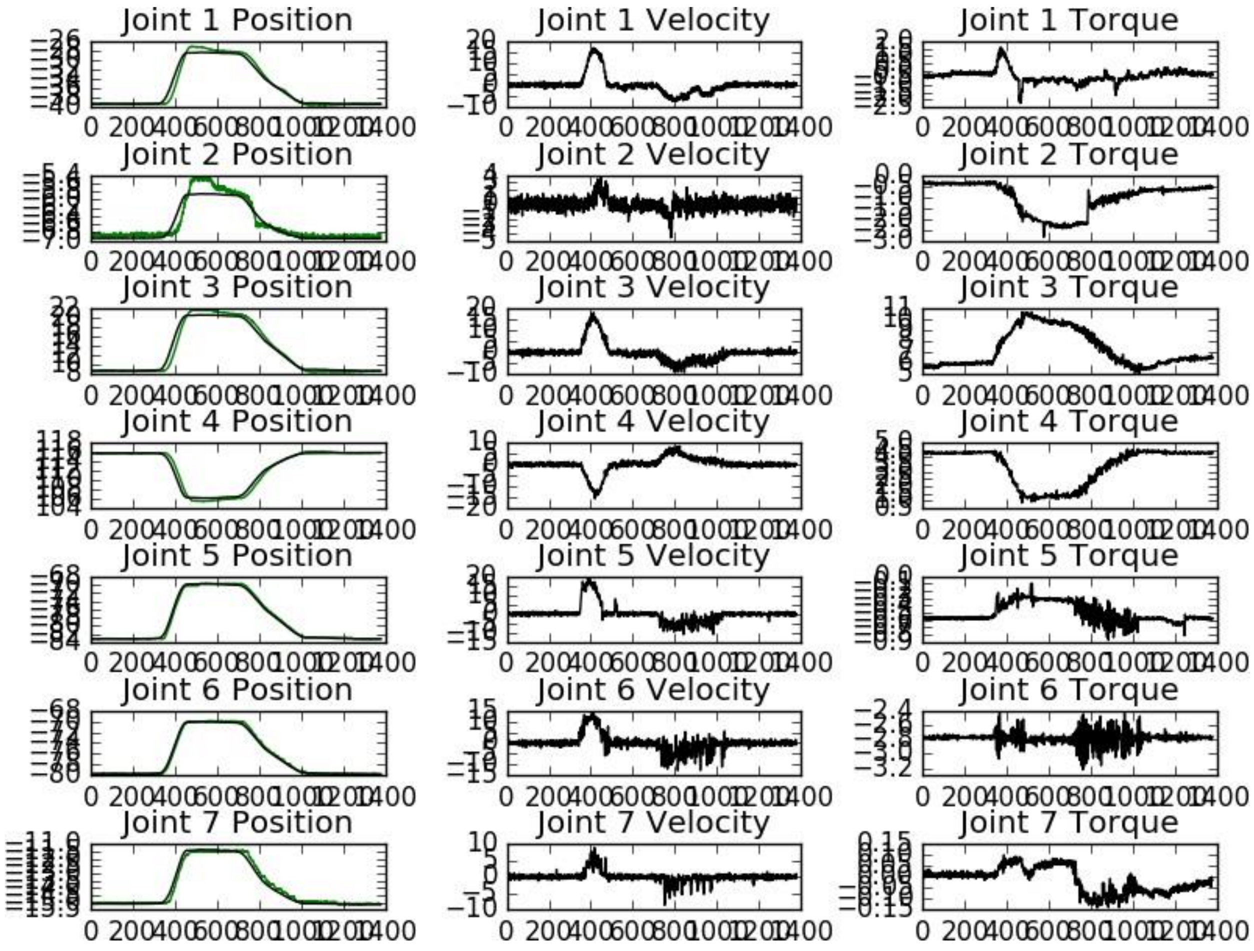

Fig. 10 – Actual joint data from Baxter after the experiment (DSCL, SDSU)

Suggestions for future research

Besides the implemented control modes, another alternative can be torque control mode which allows controlling the robot at the lowest level, but the mode will require a custom controller. It would be interesting to experiment torque control mode with a combination of telemanipulation.

Besides control modes, ROS can be used to control the system more uniquely. A separate *catkin* workspace is created with a custom BMI-ROS package consisting of a custom BMI message and a custom BMI topic which will publish BMI commands to the Baxter robot. However, it is not integrated into the Baxter workstation. It will require additional research to be fully implemented for the real procedures. For this one, it is not necessary for the control of the robot but simply a nice feature to have for the BMI.

References:

- Baxter Humanoid Robot Kinematics, Robert L. Williams II, Ph.D., Mechanical Engineering, Ohio University
- Control and benchmarking of a 7-DOF robotic arm using Gazebo and ROS, Bowei Zhang and Pengcheng Liu
- Design and Implementation of a Modular Manipulator Architecture, Ognjen Sosa
- Design of an Omnidirectional Robot Using Hybrid Brain Computer Interface, Bryan Ghoslin
- Introduction to Baxter, T.L. Harman and Carol Fairchild
- Manipulator Jacobian based on the elementary transform sequence, Peter Corke
- The Control of Baxter Robot and its Interaction with Objects Using Force Sensitive Ario Hands, Guided by Kinect, Pedro Lino
- Training Robot Manipulation Skills through Practice with Digital Twin of Baxter, Igor Verner, Dan Cuperman, Sergey Gamer, Alex Polishuk