RoVi1

Final Project

Group: Petr Batěk, Bjarki Páll Sigurdsson, Salman Taj

December 4, 2016

Project Description

1 Tracking points using image Jacobian

We have implemented algorithm for visual servoing in this part. For this part we selected specific marker points as described in problem statement and use mathematical camera model to get image pixel coordinates of the points. Image recognition thus wasn't used in this part.

To compute joint updates it was first needed to compose matrix $\boldsymbol{Z}_{image}$ as described in Robotics Notes:

$$\boldsymbol{Z}_{image}(\boldsymbol{q}) = \boldsymbol{J}_{image}\boldsymbol{S}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q}) \tag{1}$$

where $\boldsymbol{J}(\boldsymbol{q})$ is manipulator Jacobian. For its computation we used function from RobWork library. $\boldsymbol{J}_{image}$ is image Jacobian matrix. We implemented function for its computation called `calculateImageJ` which can be found in the file `inverseKinematics.cpp`. We used fixed value for z coordinate. Since we used frame `cameraSim` to model the camera we set $z = -0.5$ for every function call. Finally matrix $\boldsymbol{S}(\boldsymbol{q})$ was composed by inserting transpose of the rotational matrix $\boldsymbol{R}_{base}^{tool}$ twice to its diagonal.

The next information necessary to compute joints updates is difference or move of target points $\overrightarrow{\boldsymbol{dU}}_{image}$. We have programmed function `calculate_dUImage` to solve for $\overrightarrow{\boldsymbol{dU}}_{image}$.

Having matrix $\boldsymbol{Z}_{image}$ and $\overrightarrow{\boldsymbol{dU}}_{image}$ it was possible to solve for joint positional update $\boldsymbol{dq}$ using method of Linear Least Squares.

We adapted two equations from robotics notes into single expression for $\boldsymbol{dq}$ computation:

$$\boldsymbol{dq} = \boldsymbol{Z}^T \left(\boldsymbol{Z}\boldsymbol{Z}^T\right)^{-1} \overrightarrow{\boldsymbol{dU}}_{image} \tag{2}$$

In this equation we used $\boldsymbol{Z}$ to denote $\boldsymbol{Z}_{image}$. For this solving of this LSM problem we have implemented function `compute_dQ_LSM` which can be found in `inverseKinematics.cpp`.

`algorithm2` is the function, where are all of described functions organized together to compute joint updates $\boldsymbol{dq}$ base on the manipulator state and the error in image coordinates $\overrightarrow{\boldsymbol{dU}}_{image}$.

We have implemented J_{image} and $\overrightarrow{\boldsymbol{dU}}_{image}$ composition in a scalable way, so the same functions can be used to track one or multiple target points.

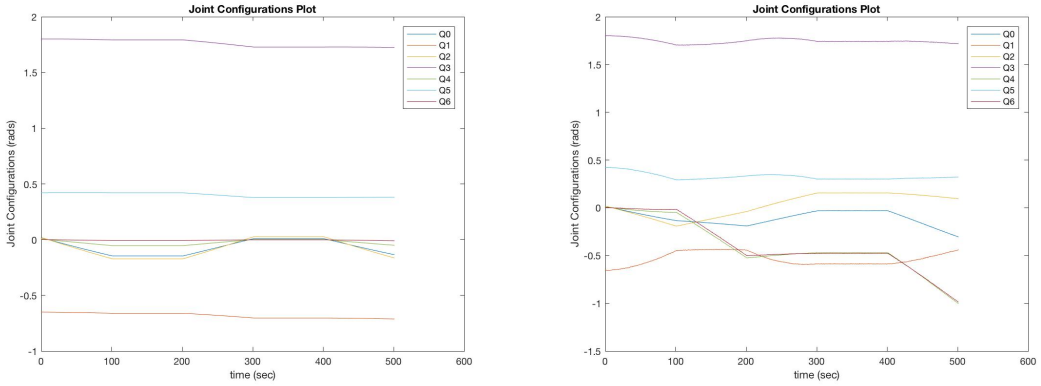
Model of the robot manipulator has velocity limits on joint movements, so it was necessary to check if the limits were satisfied before joint updates. We did so by measuring time for update computations, subtracting this time from workcell update period specified by `deltaT` and finally we divided update $\boldsymbol{dq}$ by the result of subtraction. Velocity of joint movement is the result of the operation. By comparing the actual velocity with manipulator limits it was possible to find out if the limits are satisfied. If they aren't algorithm simply saturate joint movement in order to hold all conditions. For comparison and saturation, function `saturateDQ` was implemented.

In the following section we provide simulation results from tests of inverse kinematics.

1.1 Simulation Tests

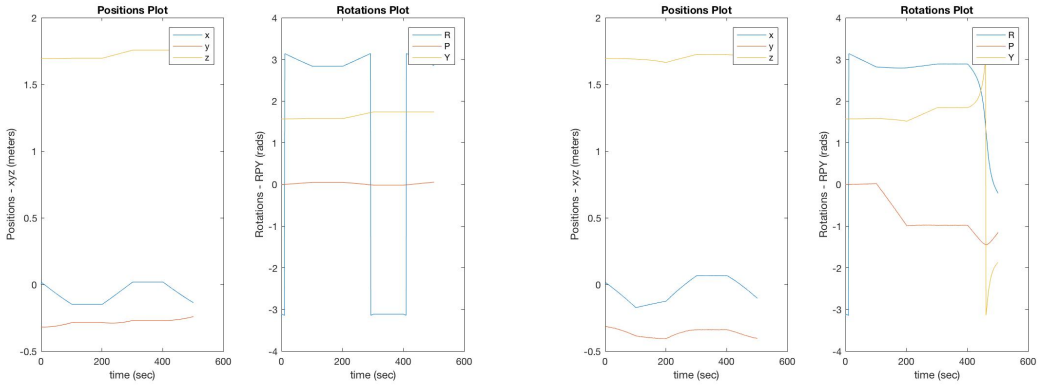
During simulations, we recorded joint configurations, tool/camera frame position and orientation for `deltaT` $= 1000ms$ and finally we performed tests for different values for `deltaT` in the range $50\,ms <$ `deltaT` $< 1000\,ms$ and plotted maximum errors of $\overrightarrow{\boldsymbol{dU}}_{image}$

Slow Marker Sequence



(a) Tracking Single Point (b) Tracking 3 Points

Figure 1: Joint configurations



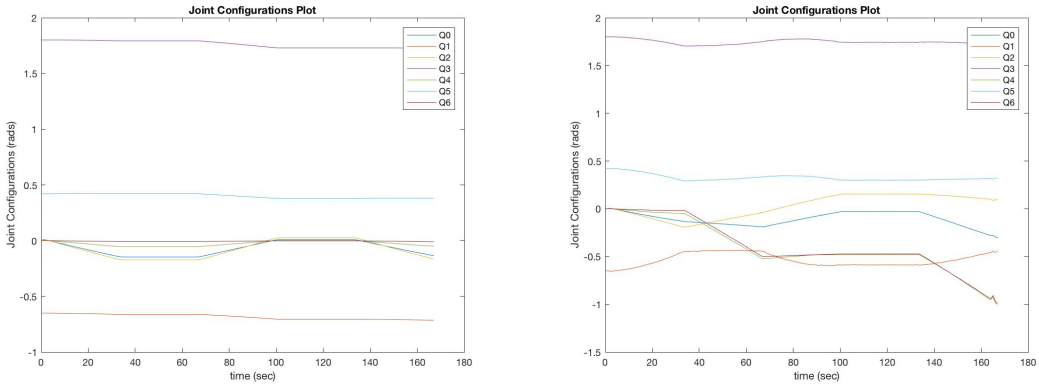
(a) Tracking Single Point (b) Tracking 3 Points

Figure 2: Tool pose transformations

In figure 2a and 2b are apparent jump in roll angle around z axis. This however doesn't imply jumps in tool orientation. Abrupt jumps in the graphs were caused because of switching between $-\pi$ and π rad angle. In the real world, change in orientation is small.

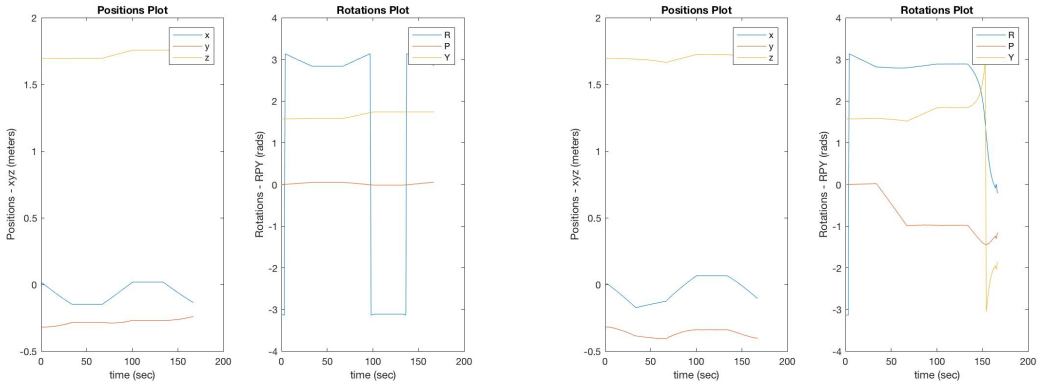
Implementation of Robwork transformations probably keeps all orientation angles in the interval $\langle -\pi,\ \pi \rangle$.

Medium Marker Sequence



(a) Tracking Single Point (b) Tracking 3 Points

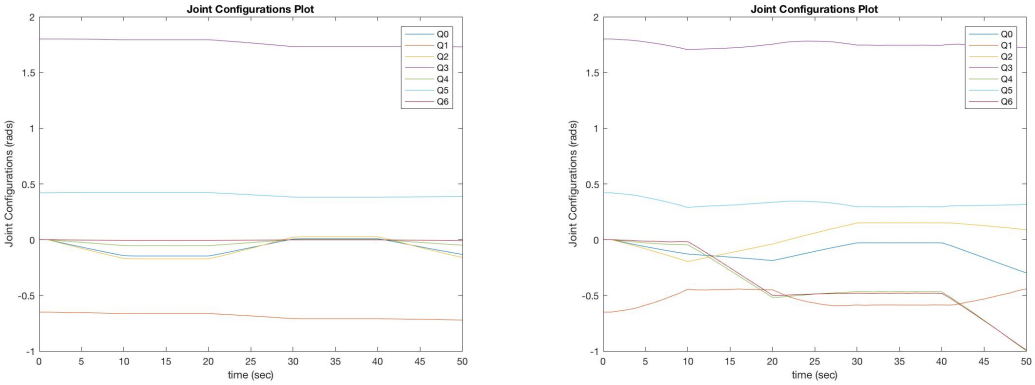
Figure 3: Joint configurations



(a) Tracking Single Point (b) Tracking 3 Points

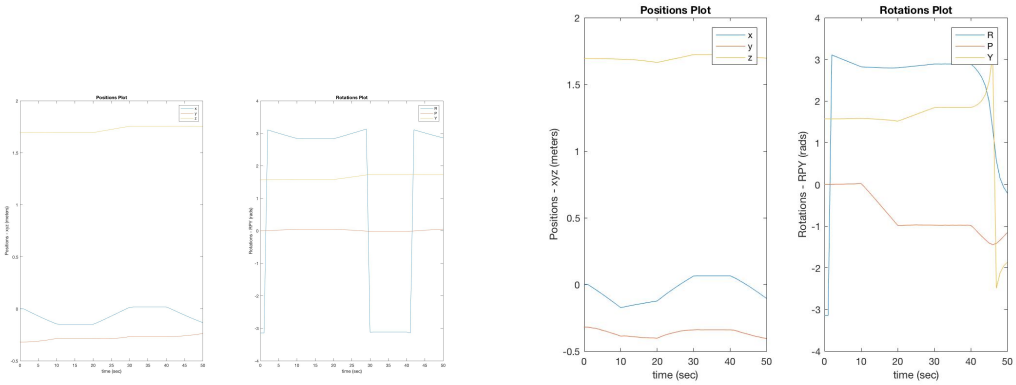
Figure 4: Tool pose transformations

Fast Marker Sequence



(a) Tracking Single Point (b) Tracking 3 Points

Figure 5: Joint configurations



(a) Tracking Single Point (b) Tracking 3 Points

Figure 6: Tool pose transformations