RoVi1

Final Project

Group: Petr Batěk, Bjarki Páll Sigurdsson, Salman Taj

December 9, 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.

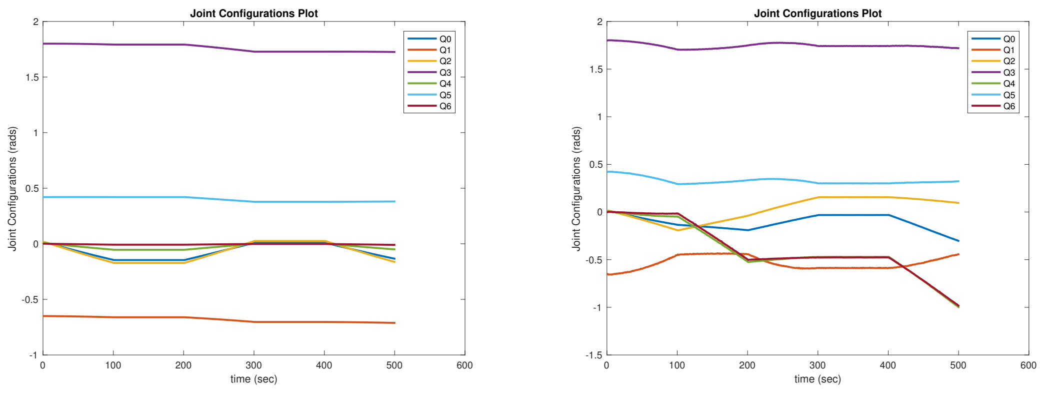
Model of the robot manipulator has velocity limits on joint movements, so it was necessary to check if the limits were satisfied before joint updates. We did so by measuring time for update computations, subtracting this time from workcell update period specified by `deltaT` and finally we divided update $\boldsymbol{dq}$ 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

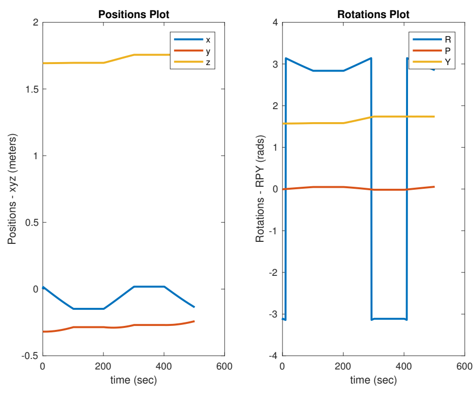
There are differences between joint coordinates for tracking single and 3 target points in the graphs 1a and 1b. The reason behind this is following. When the manipulator is tracking single point, it just follow its position. There is no orientation information about the marker when using single tracking point. Whereas during following of 3 target points, orientation of the marker gains important role, as the manipulator is trying to rotate its tool/camera frame to align its position and orientation with the marker.

In figure 2a and 2b are apparent jumps 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 $\langle -\pi,\ \pi \rangle$.

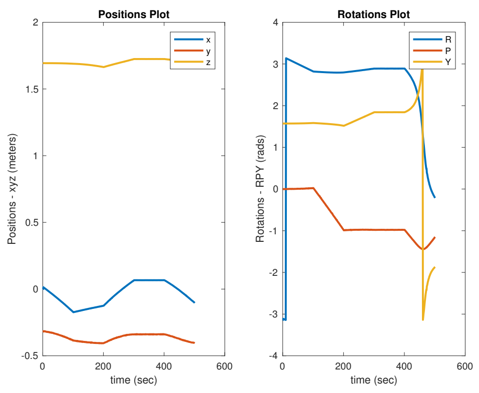
Simulation for different ΔTs

There are plotted maximum pixel errors on the figure 3. When looking to the Tracking of 3 points figure, there should be du and dv line for all of the 3 tracked points. However, because they were very close to each other, we ommited the another two lines in the graph for better clarity.

For timing purposes we used C++ Chrono library, for taking time instant of start of the computation of updates and the end instant. For differentiation of joint updates and



(a) Tracking Single Point



(b) Tracking 3 Points

Figure 2: Tool pose transformations

subsequently getting of velocity, we used difference of these two time stamps divided time which left for joints update. Due to non-real time OS, results might be different for each simulation and doesn't offer precise information. However for the purpose of the school exercise, we were able to limit joints speed velocities according to problem statement.

Medium Marker Sequence

The same problems and explanation apply for the Medium marker sequence. Joint coordinates and tool positions are plotted on the graphs 4 and 5. Only difference from Slow sequence is in the time axis. Due to faster and larger movements time span is shorter.

Simulation for different ΔTs

There are plotted maximum pixel errors on the figure 6. Again, when looking to the Tracking of 3 points figure, there should be du and dv line for all of the 3 tracked points. However, because they were very close to each other, we ommited the another two lines in the graph for better clarity.

We have to state the same as for the preceding test. All timing tests were performed on the regular laptop PC, running non-real time operating system. So results from another tests can differ.

Fast Marker Sequence

Joint coordinates and tool positions are plotted on the graphs 7 and 8. Span of time axis is again shorter.

Simulation for different ΔTs

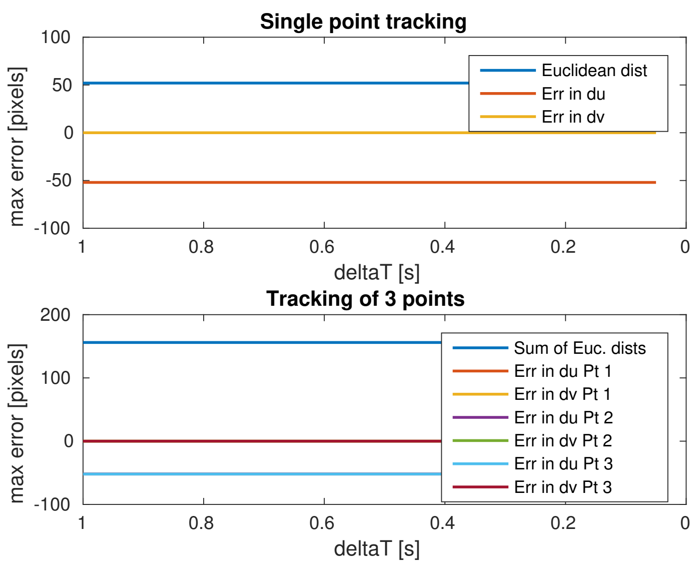
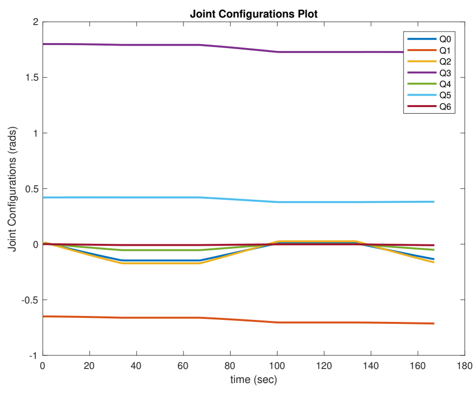
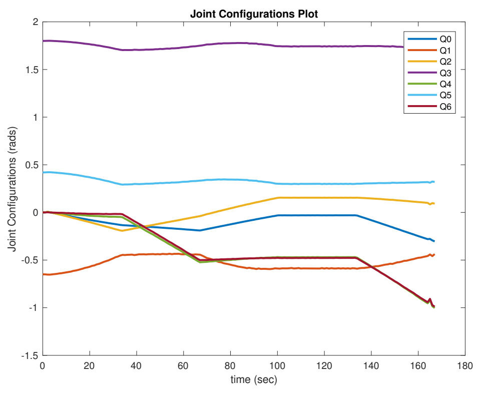


Figure 3: Maximum errors during Slow Sequence Marker following

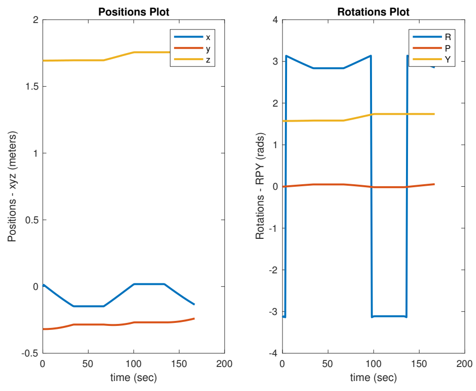


(a) Tracking Single Point

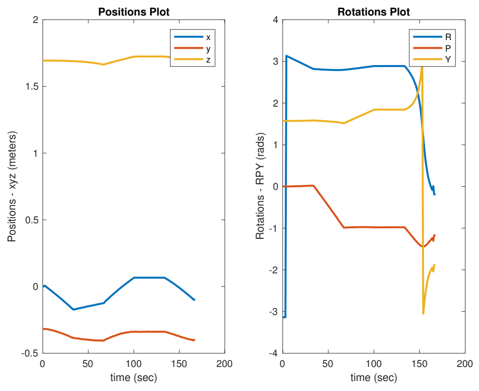


(b) Tracking 3 Points

Figure 4: Joint configurations



(a) Tracking Single Point



(b) Tracking 3 Points

Figure 5: Tool pose transformations

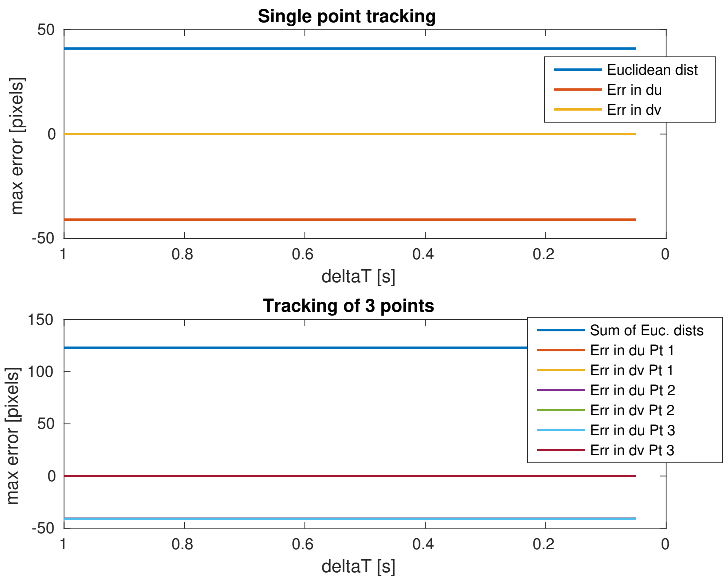
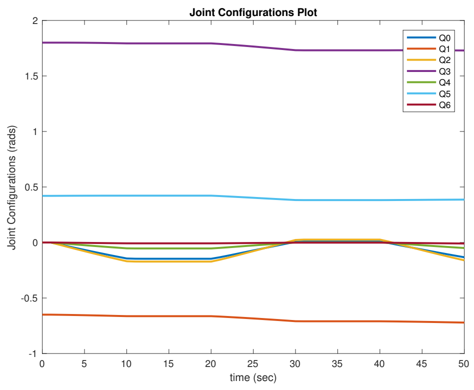
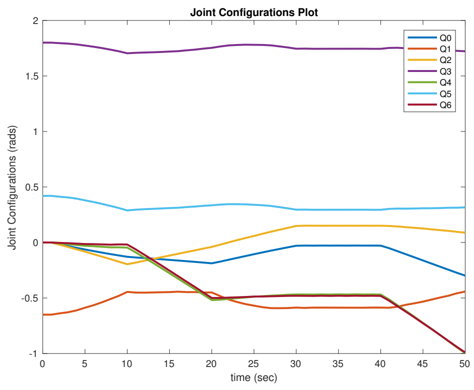


Figure 6: Maximum errors during Medium Sequence Marker following

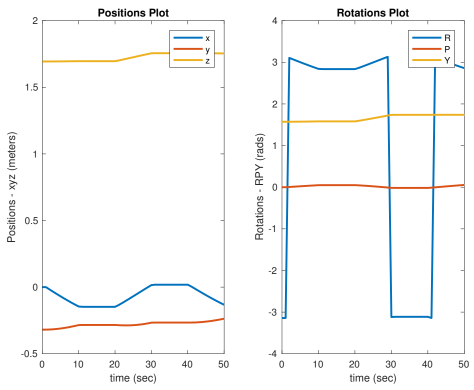


(a) Tracking Single Point

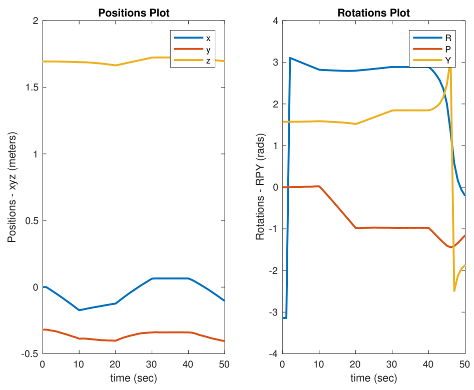


(b) Tracking 3 Points

Figure 7: Joint configurations



(a) Tracking Single Point



(b) Tracking 3 Points

Figure 8: Tool pose transformations

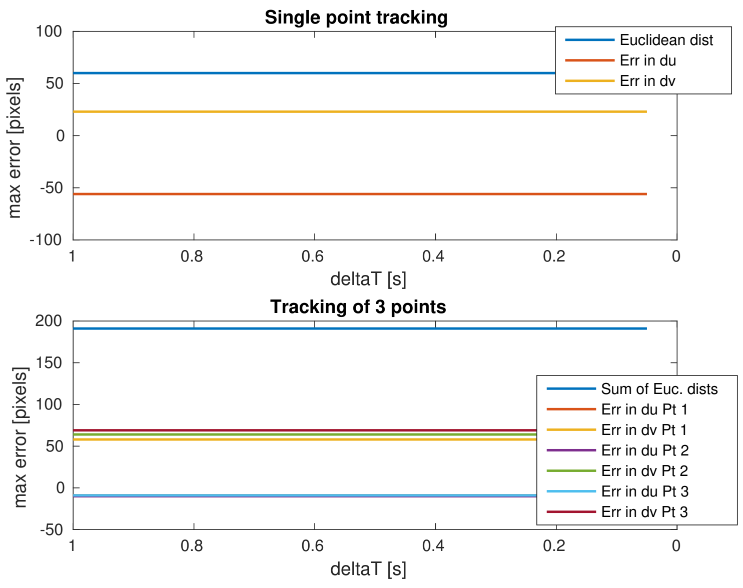


Figure 9: Maximum errors during Fast Sequence Marker following

There are plotted maximum pixel errors on the figure 9. Again, when looking to the Tracking of 3 points figure, there should be du and dv line for all of the 3 tracked points. However, because they were very close to each other, we ommited the another two lines in the graph for better clarity.

Conclusion for different speed sequences

Even though timing and velocity computation during test simulations wasn't precise due to non-realtime of the OS, we can conclude some final facts. Manipulator was able to follow the marker for each sequence with `deltaT` $> 200\,ms$. However in the faster sequences, the maximal pixel errors were larger. This is reasonable result for common sense.