Dustan Kraus

ECEN 671 Math of Signals and Systems

December 8, 2016

Final Project Report

# Introduction

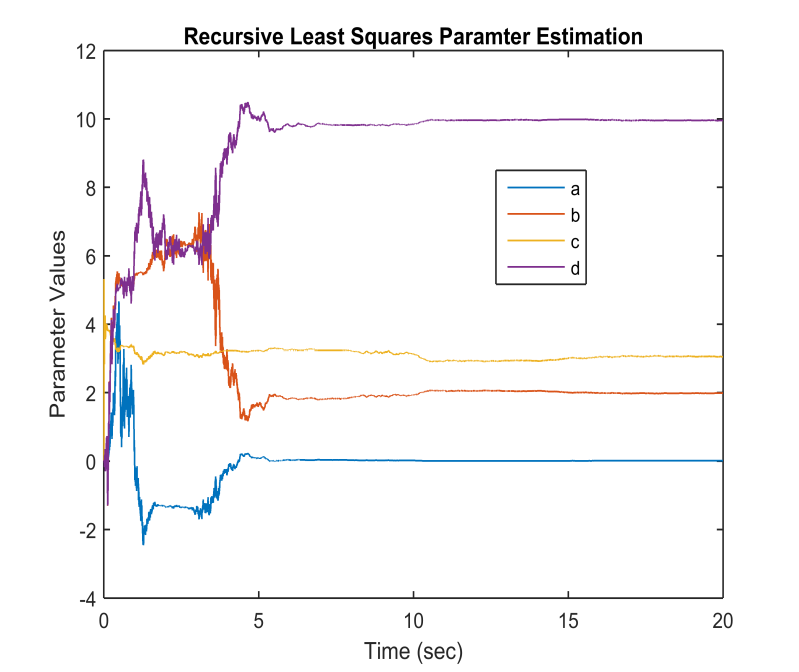
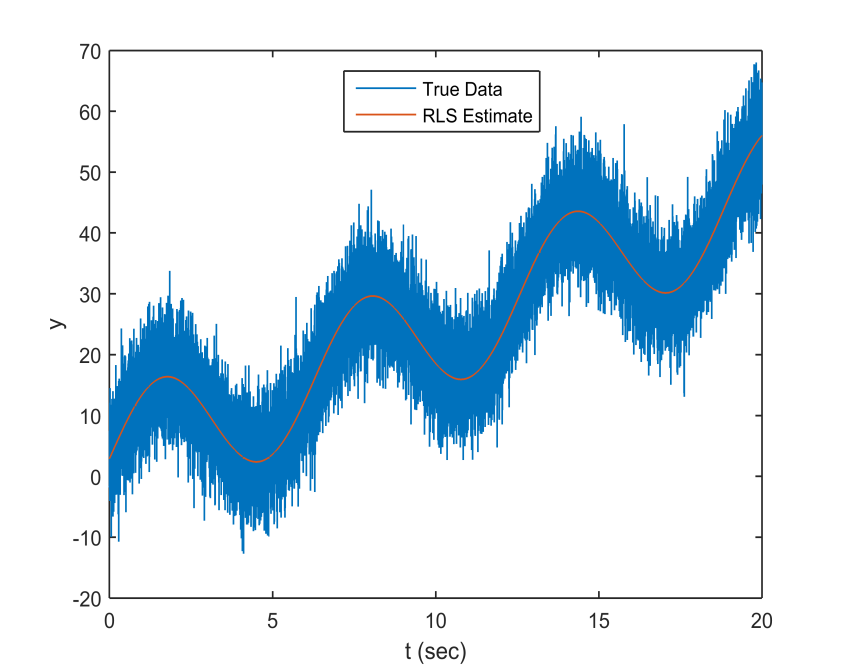
The purpose of this project was to build my understanding of the applications of the material we have learned in class to robotics. In my proposal I stated that I planned to spend 20 hours researching, summarizing, and implementing these math applications with my research. I learned a lot over the course of this project (having spent well over 20 hours), and feel that my understanding of the utility of the material we have covered in class has grown greatly.

# Application 1: Recursive Least Squares Algorithm

I began this project prior to the second midterm, and this part was especially useful on it as there was a recursive least squares problem on the second midterm. One of the topics we learned in class that I felt would be especially useful is the recursive least squares algorithm. As I reviewed the literature, one of the uses I found for this algorithm is in robotic parameter estimation [1]. You can use the recursive least squares algorithm to get online parameter estimates which continue to update as you get more and more data (known inputs and measured outputs). I wanted to build my knowledge of the recursive least squares algorithm, so I simulated a time vector and data set with four known parameters (a, b, c, and d) with added noise from the standard normal distribution resulting in the following data set equation:

y = a\*t2 + b\*t + c + d\*sin(t) + 5\*noise

I then wrote a recursive least squares algorithm to solve for a, b, c, and d. I initialized a, b, c, and d as 0 and Pm as the identity matrix. Using my algorithm I was able to successfully solve for the parameters even in the presence of a high signal to noise ratio. My results are shown on the plots below.



|  |  |  |
| --- | --- | --- |
|  | True Value | RLS Estimate |
| a | 0.01 | 0.0085 |
| b | 2 | 2.0274 |
| c | 3 | 2.9066 |
| d | 10 | 10.0486 |

As shown above, my RLS algorithm was able to successfully find a, b, c, and d. The plot on the left shows the RLS estimate plotted on top of the true data, and the plot on the right shows the convergence of the RLS estimates of a, b, c, and d to the true values.

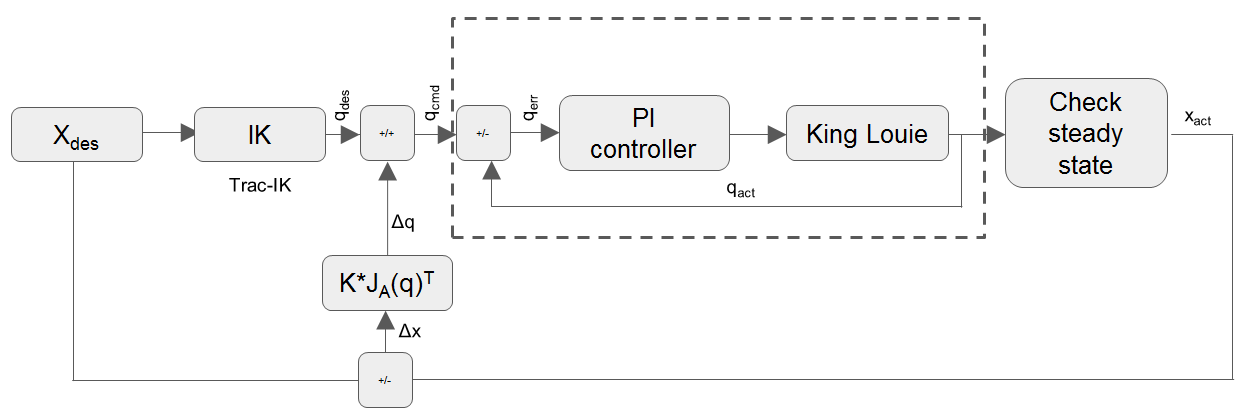
# Application 2: Special Euclidian Group SE(3)

During the first part of the semester, we discussed metric, norm, and inner product spaces. What I took away from this, is that if you can show that you are in one of these spaces, then there are many math tools available for use. I wanted to learn about spaces or groups that are particularly important in robotics where special math tools are available. One of these groups is SE(3). This group contains homogeneous transformation matrices of the form:

R

R is a rotation matrix in SO(3), and x, y, and z make up a translation vector. These transformation matrices describe the rotation and translation required to move between coordinate frames. A cool property of these matrices is that if you multiply them together in a certain order, you can find different translations. For example, if you had the transformation from coordinate frame 1 to 0 (), and the tranformation from coordinate frame 2 to 1 (), you can get the transformation from coordinate frame 2 to 1 by simply multiplying the matrices together as follows: = . Another property of these matrices is that = ()-1.

For my research this semester, I have been desiging a task space (cartesian space) controller for one of the soft robots in the lab (King Louie). Up til now, the lab has only controlled King Louie in joint space. The controller I designed is shown below:



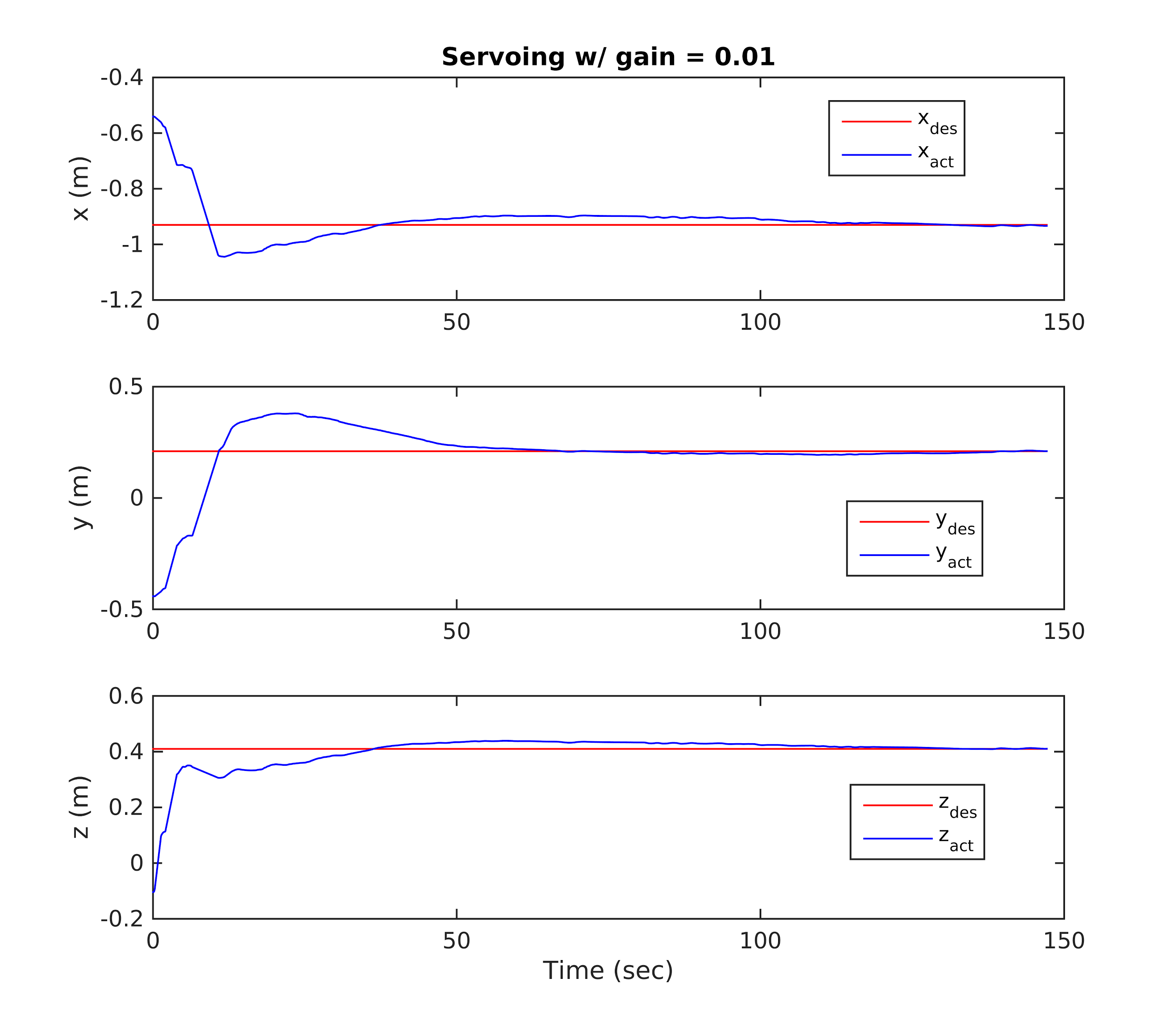
The algorithm works as follows:

1. Set a reachable xdes
2. Calculate the initial joint angles (qcmd) to command using TracIK (an inverse kinematics library in python)
3. Send qcmd to the existing PID joint space controller and wait for steady state to be reached
4. Once steady state is reached, output the actual end effector position (x­act) and calculate ∆x = xdes-x­act
5. Calculate the Jacobian (JA) using sympybotics (a python robotics library)
6. Calculate a ∆q using a numerical inverse kinematics approximation ∆q = (JA)T\*K\*∆x
7. Send a new commanded joint angle to the PID controller qcmd(tk+1) = qcmd(tk) + ∆q
8. Repeat steps 3-7 until ∆x is within a desired tolerance

One of the biggest challenges in this algorithm has been step 4. In order to calculate ∆x, I needed xact. The HTC Vive virtual reality system uses two cameras (Lighthouse 1 and 2), to track two controllers and a headset. The vive provides a transformation from the controllers to the Lighthouse 1 camera ( and ); however, what I needed was the transformation from King Louie’s world frame to his end effector (). King Louie’s world frame is in the middle of his first joint. So, I stuck controller 2 at a fixed, known displacement () from the King Louie world frame and controller 1 at the end effector. In order to get = , I used the properties of SE(3), namely, = -1 = . One problem is that changes through each loop of the control scheme, so I wanted a more efficient way of calculating the inverse than just using the inv() function in python. Upon further research, I found another property of SE(3) that made this possible: ( )-1 = = . So, I could calculate the inverse of the transformation matrix without actually taking the inverse!

Step 6 is an optimization similar to what we learned in class for the gradient descent method. I calculate ∆q by determining a small step in the right direction (using virtual force to pull the end effector towards the target) that reduces the error. This continues until the error is within some tolerance (I used 0.0001 m as my error tolerance).

I implemented this in my algorithm and was able to successfully implement the task space servoing algorithm (using the 2-norm of ∆x as a completion metric) on King Louie with the response shown in the figure below.



# Application 3: Jacobian Null Space and Singular Values

In robotics, the Jacobian is used to convert from joint velocities to linear and angular velocities () where vis the linear velocity and w is the angular velocity of the end effector, J is the Jacobian, and is the joint velocities. I wanted to learn what the singular values of the Jacobian mean, so I did some research [2] and a singular value decomposition of the Jacobian to obtain the following:

where are the singular values. What this shows is that as a joint’s singular value gets very small, you could command very large joint velocities, and the effect on would be minimal. This is bad for many reasons. First such large joint velocities may not be possible, so the robot wouldn’t do what it’s commanded. Second, if the large joint velocities are possible, the arm could move swiftly and hurt someone. Because of this, configurations where some singular values of the Jacobian become very small should be avoided.

I was also very interested in what the null space of the Jacobian meant in terms of joint torques. Upon reading papers and articles, and analyzing the joint torque equation (). I found that the null space is very powerful in robotics. The null space of the Jacobian in a certain configuration is all the forces and torques that can be applied to the end effector that will not require any joint torques. These are called reciprocal wrenches.

# Conclusion

Overall, this project has been very eye opening for me. There are tons of applications of what we’ve learned in this class to robotics. I also learned that in order to robustly control robots, I need to do some degree of mathematical analysis using the Jacobian.

# References

1. P. K. Khosla, “Estimation of robot dynamics parameters: Theory and application,” in Proc. 2nd Int. IASTED Conf. on Applied Control and Identification (Los Angeles, CA, Dec. 10-12, 1986). Chicago, IL: ACTA Press.
2. Buss, S. (2004). Introduction to inverse kinematics with jacobian transpose, pseudoinverse, and damped least squares methods. Unpublished, available at http://math.ucsd.edu/ sbuss/ResearchWeb/ikmethods/iksurvey.pdf