The University of Texas at Austin
Mechanical Engineering
Cockrell School of Engineering

ME397: Algorithms for Sensor-Based Robotics (ASBR)
Prof. Farshid Alambeigi, Spring 2023

Homework/Programming Assignment #2 **Homework/midterm Due: 04/05/2023- 5:00 PM**

Name/EID: EOA468
Email: eoandreyska@utexas.edu
Signature (*required*)
I/We have followed the rules in completing this Assignment.

Name/EID: SZF77
Email: zuhairfarra15@gmail.com
Signature (*required*)
I/We have followed the rules in completing this Assignment.

Question	Points	Total
HA 1	25	
HA 2	25	
HA 3	25	
HA 4	25	
PA	100	
PA. k (Bonus)	15	
PA. m (Bonus)	30	
Presentation* (Bonus)	20	

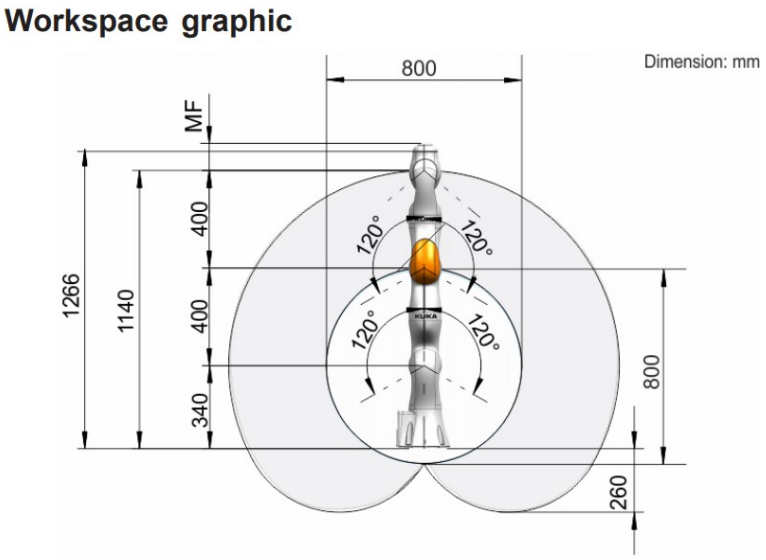
Instruction:

1. Remember that this is a graded assignment. It is the equivalent of a **midterm take-home exam**.
2. * **You should present the results of the PA in the class** and receive extra bonus depending on the quality of your presentation!
3. **For PA questions,** you need to write a report showing how you derived your equations, describes your approach, test functions, and discusses the results. You should show your test results for each function.

3. You are to work **alone** or **in teams of two** and are **not to discuss the problems with anyone** other than the TAs or the instructor.
4. It is open book, notes, and web. But you should cite any references you consult.
5. Unless I say otherwise in class, it is due before the start of class on the due date mentioned in the Assignment.
6. **Sign and append** this score sheet as the first sheet of your assignment.
7. Remember to submit your assignment in Canvas.

PROGRAMING ASSIGNMENT:

The KUKA LBR iiwa7 R800 was the model used for our robot. It is a 7 degree of freedom RRRRRRR open chain manipulator. The workspace graphic from the robot's data sheet is shown below, wherein the link dimensions and overall reach is shown [2]. Additionally, we used the robot's model from MATLAB as reference due to the availability of joint-to-joint transformations that provide further clarity.



The joint limits of the robot were also obtained from the datasheet, and are shown in the table below.

Joint	Motion Angle Range
A1 (closest to base)	±170°
A2	±120°
A3	±170°
A4	±120°
A5	±170°
A6	±120°
A7 (closest to end effector)	±175°

Diagram / home position used for the programing assignment:

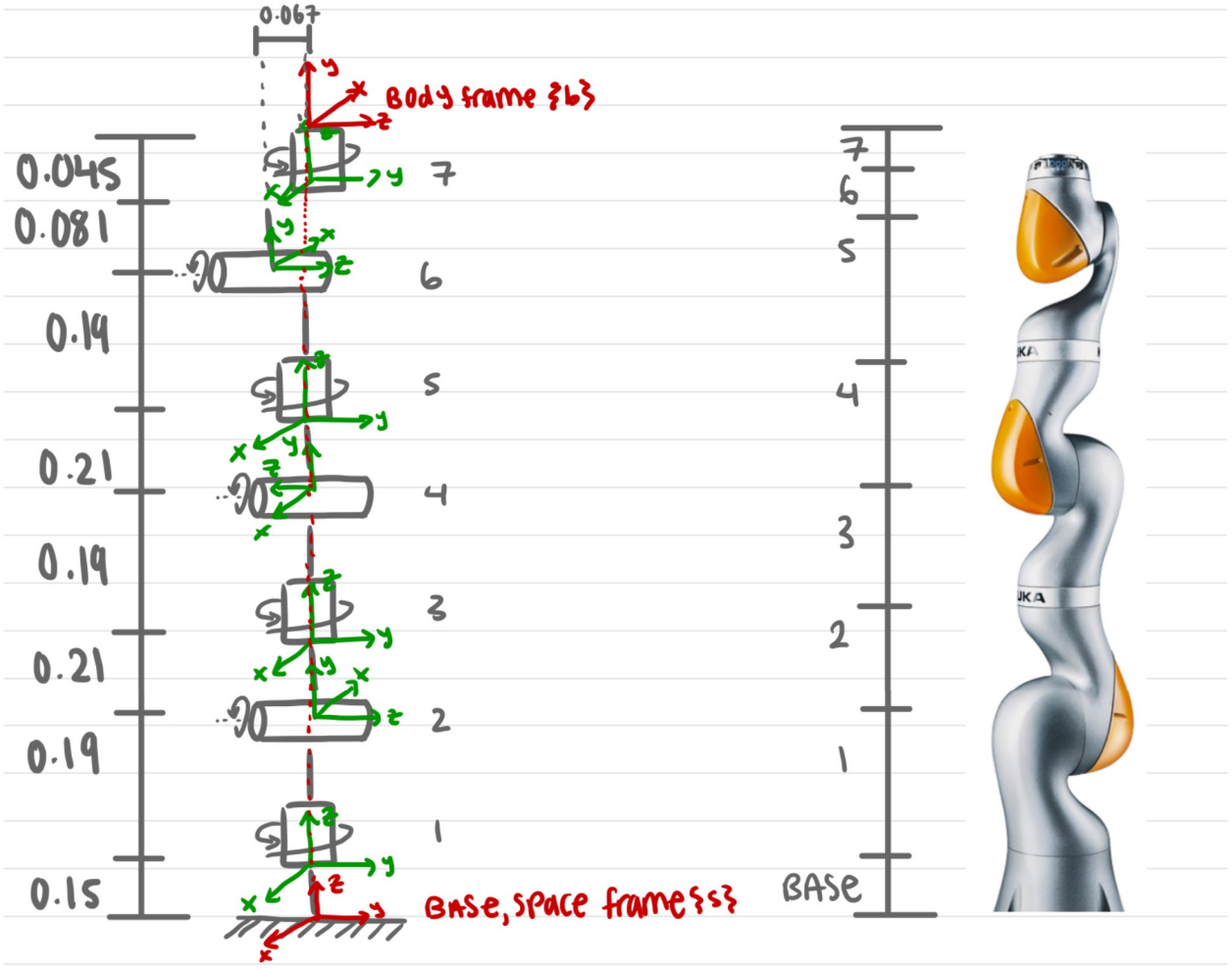


Image from the KUKA iiwa data sheet [2].

A. Space form forward kinematics (FK)

The following values for ω, q, v, *and* M were determined for the forward kinematics of the body form using the above schematic, where $v = -\omega x q$. The matrix exponential representation for screw motions for each joint are constructed, multiplied in order, and post multiplied by M.

Forward kinematics : space form

	ω	q	v
1	(0, 0, 1)	(0, 0, 0)	(0, 0, 0)
2	(0, 1, 0)	(0, 0, 0.15+0.19)	(-0.34, 0, 0)
3	(0, 0, 1)	(0, 0, 0)	(0, 0, 0)
4	(0, -1, 0)	(0, 0, 0.15+0.19+0.21+0.19)	(0.74, 0, 0)
5	(0, 0, 1)	(0, 0, 0)	(0, 0, 0)
6	(0, 1, 0)	(0, 0, 0.15+0.19+0.21+0.19+0.21+0.19)	(-1.14, 0, 0)
7	(0, 0, 1)	(0, 0, 0)	(0, 0, 0)

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1.266 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{space} = e^{[S_1]\theta_1} e^{[S_2]\theta_2} e^{[S_3]\theta_3} e^{[S_4]\theta_4} e^{[S_5]\theta_5} e^{[S_6]\theta_6} e^{[S_7]\theta_7} M$$

where →

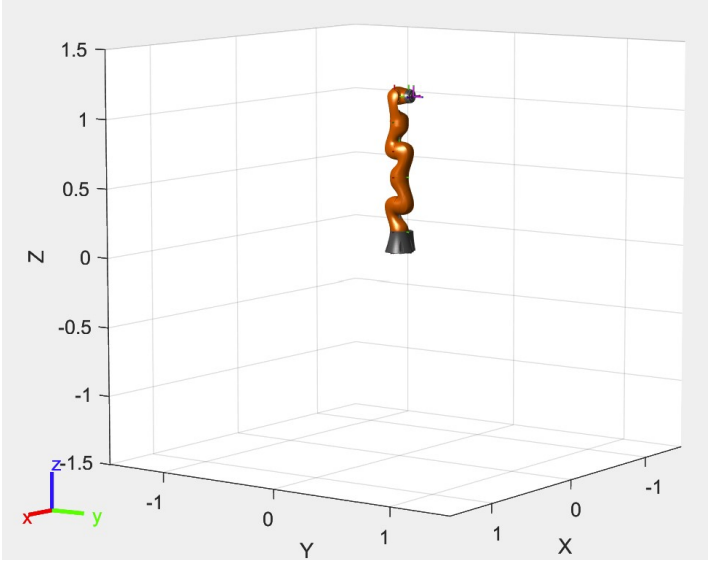
$$e^{[S_n]\theta_n} = \begin{bmatrix} e^{[\omega_n]\theta_n} & I\theta + (1-\cos\theta)[\omega_n] + (\theta - \sin\theta)[\omega_n]^2 v \\ 0 & 1 \end{bmatrix}$$

B. Function "FK_space.m"

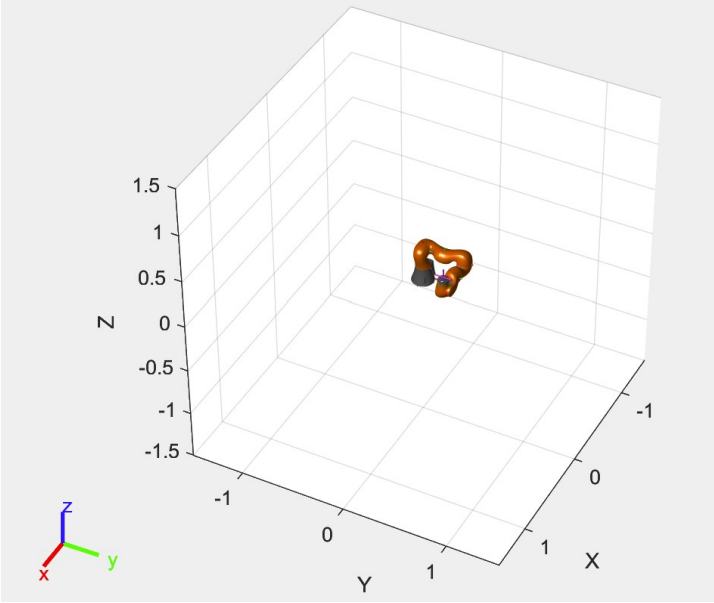
The following test cases were given to the robot. The first two configurations were chosen, and the third was set randomly by the robot class in MATLAB. The angles of each joint at the configurations are shown below. Our function to calculate the forward kinematics (T1) was compared to the built-in robotics toolbox function in MATLAB for our robot (KUKA iiwa).

Visualization for the test cases used in Part B and C:

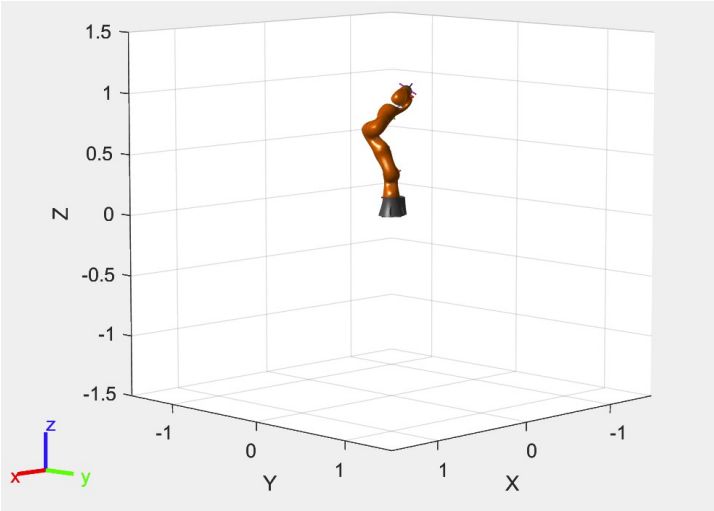
Case1:



Case 2:

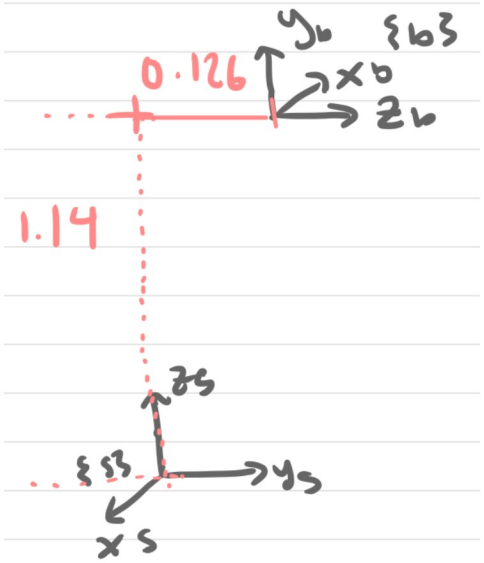


Case 3:



	Thetas	T1 using our functions	T1 using the built in robotics toolbox function for the KUKA iiwa
Case 1	0 0 0 0 pi/2 pi/2 pi/2	-1 0 0 0 0 0 1 0.126 0 1 0 1.14 0 0 0 1	1 0 0 0 0 0 1 0.126 0 1 0 1.14 0 0 0 1
Case 2	pi/2 pi/2 pi/2 pi/2 pi/2 pi/2 pi/2	0 1 0 0 .4 - 1 0 0 0.4 0 1 1 0.466 0 0 0 1	0 1 0 0 .4 - 1 0 0 0.4 0 1 1 0.466 0 0 0 1
Case 3	(in degrees) -41 22 -147 -71 76 18 -105	-0.4885 -0.1732 -0.8552 -0.3149 0.6890 -0.6779 -0.2563 -0.1251 -0.5354 -0.7144 0.4505 1.0072 0 0 0 1.0000	-0.4885 -0.1732 -0.8552 -0.3149 0.6890 -0.6779 -0.2563 -0.1251 -0.5354 -0.7144 0.4505 1.0072 0 0 0 1.0000

The results are the same, so this function passes the test. The results make sense intuitively as well, which can be seen in the schematic below for test case 1.



Going from the {s} coordinate frame (base) to the {b} coordinate frame {end effector}, the coordinate frame would be flipped: R = [-1 0 0 ;0 0 1;0 1 0] (xb = -xs, yb = zs, zb = ys), and would be translated up 1.14 units in the z direction, and 0.126 units in the y direction P = [0; 0.126; 1.14].

The code for this problem is in “FK_space.m”, “FK_test.m”, and “KUKA_T”.

C. Body form Forward Kinematics, and space and body form Jacobian of the robot.

The following values for ω, q, v, *and* M were determined for the forward kinematics of the body form using the above schematic, where $v = -\omega x q$. The matrix exponential representation for screw motions for each joint are constructed, multiplied in order, and pre multiplied by M.

Forward kinematics: Body form

	ω	q	v
1	(0, 0, 1)	(0, 0, 0)	(0, 0, 0)
2	(0, 1, 0)	(0, 0, -0.045-0.081)	(0.926, 0, 0)
3	(0, 0, 1)	(0, 0, 0)	(0, 0, 0)
4	(0, -1, 0)	(0, 0, -0.045-0.081-0.19-0.21)	(-0.526, 0, 0)
5	(0, 0, 1)	(0, 0, 0)	(0, 0, 0)
6	(0, 1, 0)	(0, 0, -0.045-0.081-0.19-0.21-0.19-0.21)	(0.126, 0, 0)
7	(0, 0, 1)	(0, 0, 0)	(0, 0, 0)

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1.266 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{body} = M e^{[B_1]\theta_1} e^{[B_2]\theta_2} e^{[B_3]\theta_3} e^{[B_4]\theta_4} e^{[B_5]\theta_5} e^{[B_6]\theta_6} e^{[B_7]\theta_7}$$

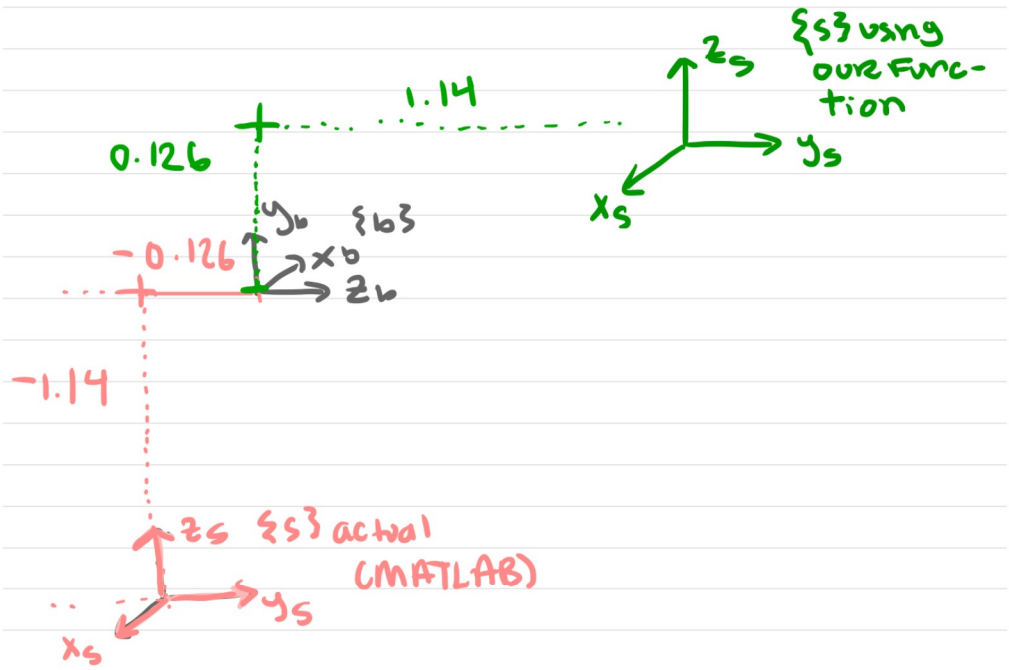
where →

$$e^{[B_n]\theta_n} = \begin{bmatrix} e^{[\omega_n]\theta_n} & I\theta + (1-\cos\theta)[\omega_n] + (\theta - \sin\theta)[\omega_n]^2 v \\ 0 & 1 \end{bmatrix}$$

The following test cases were given to the robot. The first two configurations were chosen, and the third was set randomly by the robot class in MATLAB. The angles of each joint at the configurations are shown below. Our function to calculate the forward kinematics (T1) was compared to the built-in robotics toolbox function in MATLAB for our robot (KUKA iiwa).

	Thetas (theta1,..., theta7)	T1 using our functions	T1 using the built in robotics toolbox function for the KUKA iiwa
Case 1	0 0 0 0 pi/2 pi/2 pi/2	-1 0 0 0 0 0 1 0.126 0 1 0 1.14 0 0 0 1	1 0 0 0 0 0 1 -1.14 0 1 0 -0.126 0 0 0 1
Case 2	pi/2 pi/2 pi/2 pi/2 pi/2 pi/2 pi/2	0 1 0 0 .4 - 1 0 0 0.4 0 1 1 0.466 0 0 0 1	0 1 0 -0 .466 - 1 0 0 -0.4 0 1 1 -0.4 0 0 0 1
Case 3	(in degrees) -41 22 -147 -71 76 18 -105	-0.4885 -0.1732 -0.8552 -0.3149 0.6890 -0.6779 -0.2563 -0.1251 -0.5354 -0.7144 0.4505 1.0072 0 0 0 1.0000	-0.4885 0.6890 -0.5354 0.4716 -0.1732 -0.6779 -0.7144 0.5802 -0.8552 -0.2563 0.4505 -0.7552 0 0 0 1.0000

The results are not the same, so there may be something wrong with our function. The physical representation of what our function “see’s” from the end effector to the base vs. what the MATLAB function see’s can be seen below for test case 1.



Based on this diagram, it appears that our function isn't seeing the actual base ({s} frame, pink) with respect to the end effector ({b} frame) correctly, but is seeing it as the end-effector with respect to the base (green). The MATLAB function result looks correct, and would translate and rotate the body frame {b} to the {s} frame. Our function's rotation matrix (R) looks correct for each test case; the P is what is off, and is translating the {b} frame in the positive yb and zb directions.

The code for this problem is in "FK_body.m", "FK_test.m", and "KUKA_T".

D and E: J_space and J_body functions

To calculate the spatial and body jacobians of our robot, used the approach laid out by Lynch and Park [1], described by the following formulation, wherein the Jsn terms represent the spatial jacobian terms associated with the screw axes of the robot, and Vs is the spatial twist.

$$\mathcal{V}_s = \underbrace{\mathcal{S}_1}_{J_{s1}}\dot{\theta}_1 + \underbrace{\mathrm{Ad}_{e^{[\mathcal{S}_1]\theta_1}}(\mathcal{S}_2)}_{J_{s2}}\dot{\theta}_2 + \underbrace{\mathrm{Ad}_{e^{[\mathcal{S}_1]\theta_1}e^{[\mathcal{S}_2]\theta_2}}(\mathcal{S}_3)}_{J_{s3}}\dot{\theta}_3 + \cdots$$

$$\mathcal{V}_b = \underbrace{\mathcal{B}_n}_{J_{bn}}\dot{\theta}_n + \underbrace{\mathrm{Ad}_{e^{-[\mathcal{B}_n]\theta_n}}(\mathcal{B}_{n-1})}_{J_{b,n-1}}\dot{\theta}_{n-1} + \cdots + \underbrace{\mathrm{Ad}_{e^{-[\mathcal{B}_n]\theta_n}\cdots e^{-[\mathcal{B}_2]\theta_2}}(\mathcal{B}_1)}_{J_{b1}}\dot{\theta}_1$$

Similarly, the Jbn terms represent the body jacobian terms associated with the screw axes of the robot, and Vb is the body twist. Both equations for Js and Jb can be seen below:

$J_{space\,i}(\theta) = Ad_{e^{[S_1]\theta_1}\cdots e^{[S_{i-1}]\theta_{i-1}}}(S_i)$

for the i^{th} column of J_{space}, i^{th} joint

$J_{body\,i}(\theta) = Ad_{e^{-[B_n]\theta_n}\cdots e^{[B_{i+1}]\theta_{i+1}}}(B_i)$

for the i^{th} column of J_{body}, i^{th} joint

where:

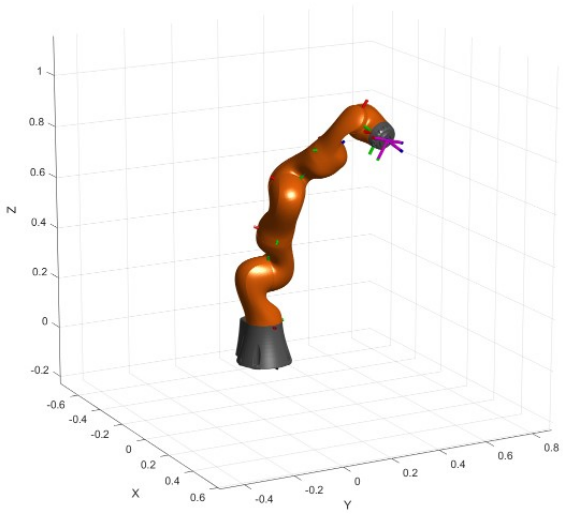
$Ad_T = \begin{bmatrix} R & 0 \\ [p]R & R \end{bmatrix}$, and

$e^{[X_n]\theta_n} = \begin{bmatrix} e^{[\omega_n]\theta_n} & I\theta + (1-\cos\theta)[\omega_n] + (\theta - \sin\theta)[\omega_n]^2 v \\ 0 & 1 \end{bmatrix}$

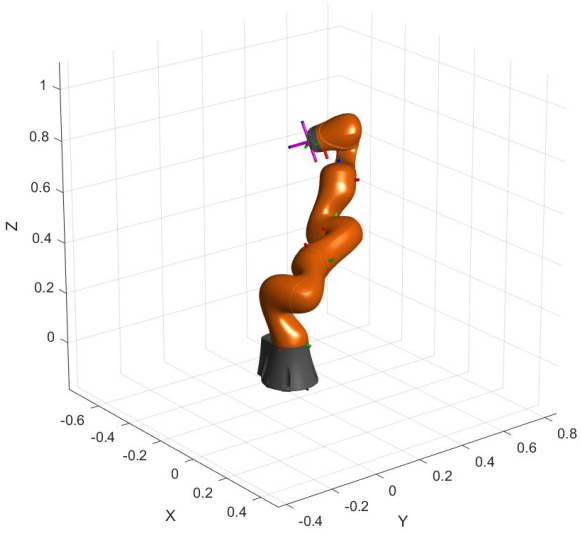
($X = S$ or B)

The following test cases were given to the robot. The configurations were set randomly by the robot class in MATLAB. The angles of each joint at the configurations are shown below.

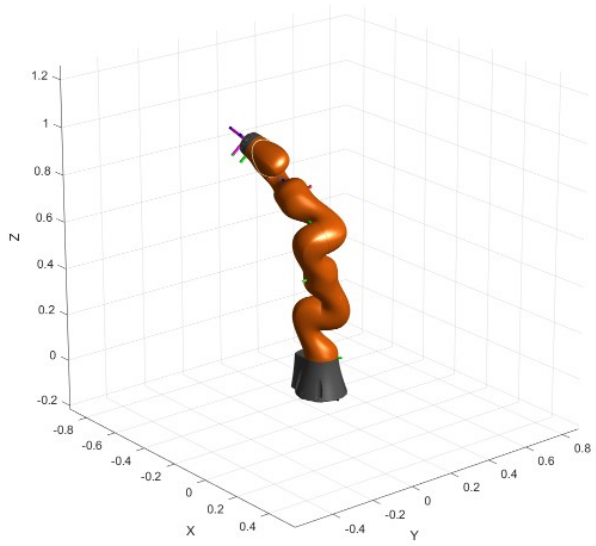
	Angle 1	Angle 2	Angle 3	Angle 4	Angle 5	Angle 6	Angle 7
Case 1	3°	25°	-115°	32.5°	117°	68°	-82°
Case 2	-63°	-76°	-18°	-41.5°	-75°	104°	-35°
Case 3	-41°	22°	-147°	-71°	76°	18°	-105°



Test Configuration 1



Test Configuration 2



Test Configuration 3

The spatial and body jacobian functions were passed the aforementioned list of angles with the screw axis matrices as input arguments. The returned jacobians were compared against one another using the following formulation.

$$J_s(\theta) = \mathrm{Ad}_{T_{sb}}\left(J_b(\theta)\right) = [\mathrm{Ad}_{T_{sb}}]J_b(\theta).$$

The matrix Tsb was calculated using the forward kinematics function discussed earlier in this document and is the end effector configuration with respect to the fixed spatial frame.

Space Jacobians for test cases:

Case 1	Js1	Js2	Js3	Js4	Js5	Js6	Js7
wx	0	-0.05136	0.418776	-0.84266	0.535051	-0.32797	0.923206
wy	0	0.99868	0.021538	0.381818	0.514744	0.857312	0.201794
wz	1	0	0.907834	0.379654	0.669895	-0.3968	-0.32706
vx	0	-0.33955	-0.00732	-0.2652	-0.35616	-0.91765	-0.26612
vy	0	-0.01746	0.142384	-0.6561	0.263998	-0.16709	1.021302
vz	0	0	0	0.071218	0.081615	0.397444	-0.12105

Case 2	Js1	Js2	Js3	Js4	Js5	Js6	Js7
wx	0	0.891025	-0.44053	-0.88169	-0.44162	0.390583	-0.68098
wy	0	0.453954	0.864682	-0.36613	0.418316	-0.70679	-0.65297
wz	1	0	0.241362	-0.29761	0.793714	0.589824	-0.33151
vx	0	-0.15434	-0.29399	0.056895	0.091911	0.835639	0.322227
vy	0	0.302949	-0.14978	-0.43734	-0.05292	0.502638	-0.63046
vz	0	0	0	0.369469	0.079032	0.048952	0.579885

Case 3	Js1	Js2	Js3	Js4	Js5	Js6	Js7
wx	0	0.656008	0.286191	0.164992	-0.79666	0.524441	-0.85055
wy	0	0.754754	-0.24875	0.964083	0.005855	-0.49042	-0.26424
wz	1	0	0.925321	0.208138	0.604396	0.696023	0.454687
vx	0	-0.25662	0.084574	-0.70533	-0.06429	0.399202	0.207353
vy	0	0.223043	0.097305	0.093338	-0.63492	0.641328	-0.71679
vz	0	0	0	0.126781	-0.0786	0.151092	-0.02868

Body Jacobians for test cases:

Case 1	Jb1	Jb2	Jb3	Jb4	Jb5	Jb6	Jb7
wx	0.279916	-0.92654	0.359512	-0.49372	-0.12238	-0.99122	0
wy	-0.90259	-0.34318	-0.92839	-0.27477	-0.91738	0.132226	0
wz	-0.32706	0.15411	0.094043	-0.82507	0.378736	0	1
vx	-0.52589	0.005639	-0.22311	-0.37901	-0.11559	0.01666	0
vy	-0.11923	0.306581	-0.10437	0.177392	0.015419	0.124894	0
vz	-0.12105	0.716623	-0.17737	0.167721	1.73E-18	0	0

Case 2	Jb1	Jb2	Jb3	Jb4	Jb5	Jb6	Jb7
wx	-0.9416	0.334826	-0.1926	-0.03582	-0.79506	-0.57511	0
wy	0.059034	0.268598	-0.91877	-0.34439	-0.55893	0.818073	0
wz	-0.33151	-0.90319	-0.34463	0.938143	-0.23554	0	1
vx	-0.20221	-0.56235	-0.10625	0.198798	-0.07043	0.103077	0
vy	0.031094	0.392329	0.115722	-0.29721	0.100177	0.072464	0
vz	0.579885	-0.0918	-0.24913	-0.10152	6.94E-18	0	0

Case 3	Jb1	Jb2	Jb3	Jb4	Jb5	Jb6	Jb7
wx	-0.5295	0.193194	-0.80317	0.47083	0.079707	-0.96629	0
wy	-0.71617	-0.62371	-0.54392	-0.82949	-0.29917	-0.25745	0
wz	0.454687	-0.75741	0.243042	-0.30045	0.950865	0	1
vx	-0.27891	-0.63187	-0.24633	-0.45596	-0.0377	-0.03244	0
vy	0.188005	0.102577	0.41443	-0.24798	-0.01004	0.121753	0
vz	-0.02868	-0.24564	0.113455	-0.0299	0	0	0

The jacobians found from testing are equivalent, and therefore, pass the test.

The test code for this section can be found in “main_ROBOT_w_Test.com”, in section “J_space and J_body testing”.

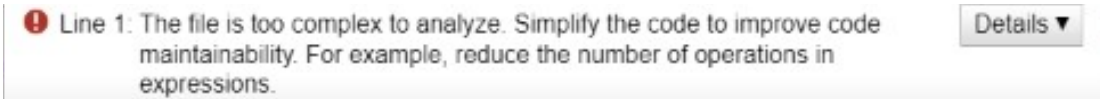
F. Write a function "singularity.m" that analytically calculates the singularity configurations of the robot. Need to add a function to check if at a singularity.

ANALYTICAL CALCULATION:

To perform the analytical calculation, we needed to set detM equal to zero. To calculate detM symbolically, we used:

```
syms M [6 6];
detM = simplify(det(M))
```

in the command window. The output of this were in terms of M1_1, M1_2,... which we replaced with M(1,1) and M(1,2) and so on, and we fed the symbolic values for those variables into the detM equation, with M being the symbolic space Jacobian. When we tried to solve for det(M) directly, matlab would error out saying:



This error also happened when we did this method:

```
syms M [7 7];
detM = simplify(det(M))
```

for 7DOF. Our computers and/or matlab couldn't symbolically solve for the 7DOF singularities, so we went forward calculating 6DOF.

TEST CASE:

The output of detM == 0 using VPA solve is:

$$\text{Thetas} = \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

The function can only solve for one value, and our home configuration is all thetas = 0. This result makes sense, since it's the closest singularity to the home configuration (the home configuration is at a singularity).

The code for this problem is in "singularity_test.m" and "singularity_analytical.m".

NUMERICAL CALCULATION:

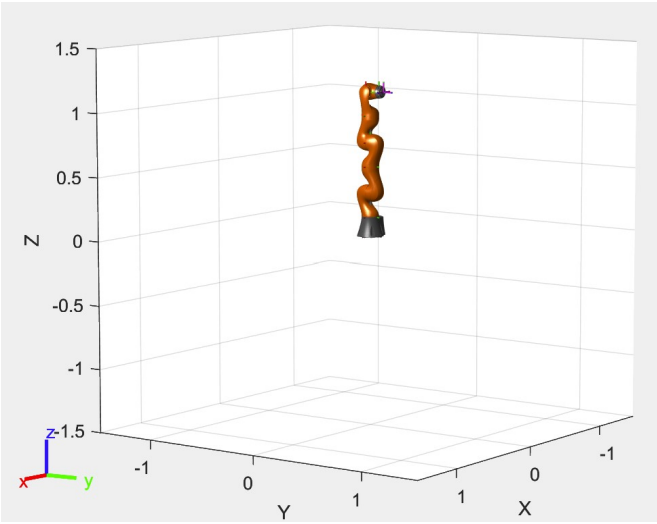
To calculate if the robot was at a singularity for any given configuration, we checked the rank of the jacobian (J), and the determinate of $J*J^{-1}$. If the Jacobian's rank was equal to 6 (maximal rank of our robot) and the determinate of $J*J^{-1}$ was not 0, then the Jacobian / the configuration of the robot was not at a singularity, and the function would return a 0 value to indicate "false" for the robot being at a singularity. For all other values, the function would return a "1" for "true" the robot being at a singularity.

TEST CASES:

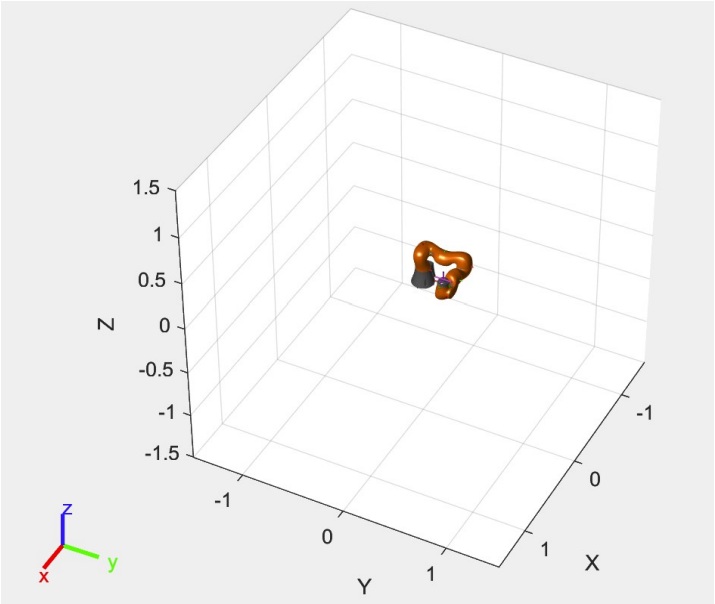
	Thetas	Output from Singularity_numerical Function	Meaning of output
Case 1	0 0 0 0 pi/2 pi/2 pi/2	1	True; robot is at a singularity
Case 2	pi/2 pi/2 pi/2 pi/2 pi/2 pi/2 pi/2	0	False; robot is not at a singularity
Case 3	(in degrees) -41 22 -147 -71 76 18 -105	0	False; robot is not at a singularity

Visualization of the test cases:

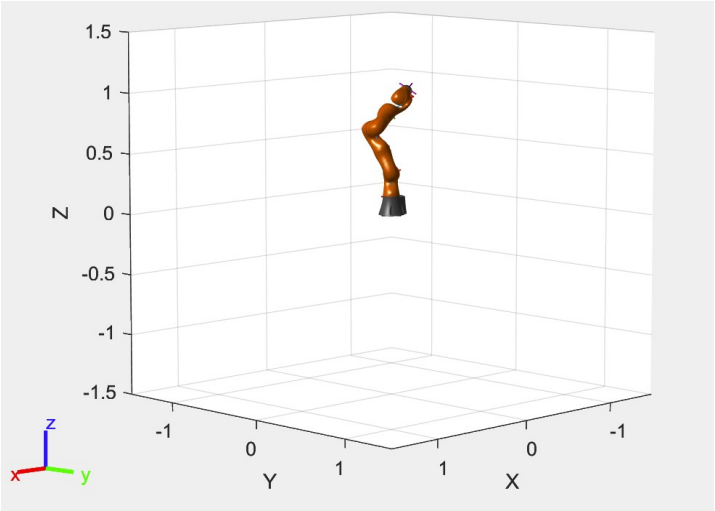
Case1:



Case 2:



Case 3:



By inspection, these results make sense. For the first test case, there are more than one collinear revolute axes, which indicates the robot is at a singularity. For the other two cases, the robot appears to be able to still move in all directions, which would indicate that the robot is not at a singularity, and has a nonzero velocity at the end effector along all 6 axes.

The code for this problem is in “singularity_test.m” and “singularity_numerical.m”.

G. Manipulability Ellipsoids

Manipulability ellipsoids represent the of a configuration is at a singularity, and the directions that an end-effector can move the most freely and with the least effort.The plots for the manipulability ellipsoids were calculated by getting the eigenvectors and eigenvalues of the Jacobian (J*J'). The square root of eigenvalues correspond to the principal semi-axis, or the "length", "width", and "height" of the ellipse. The eigenvector corresponds to the rotation of the ellipse. From this 3x3 rotation matrix, the euler angles were obtained, and the ellipse was rotated around each axis by its corresponding euler angle.

The equations used to calculate ellipsoid metrics are below:

$$Istropy = \mu1(A) = \sqrt{\frac{\lambda max(A)}{\lambda min(A)}}$$

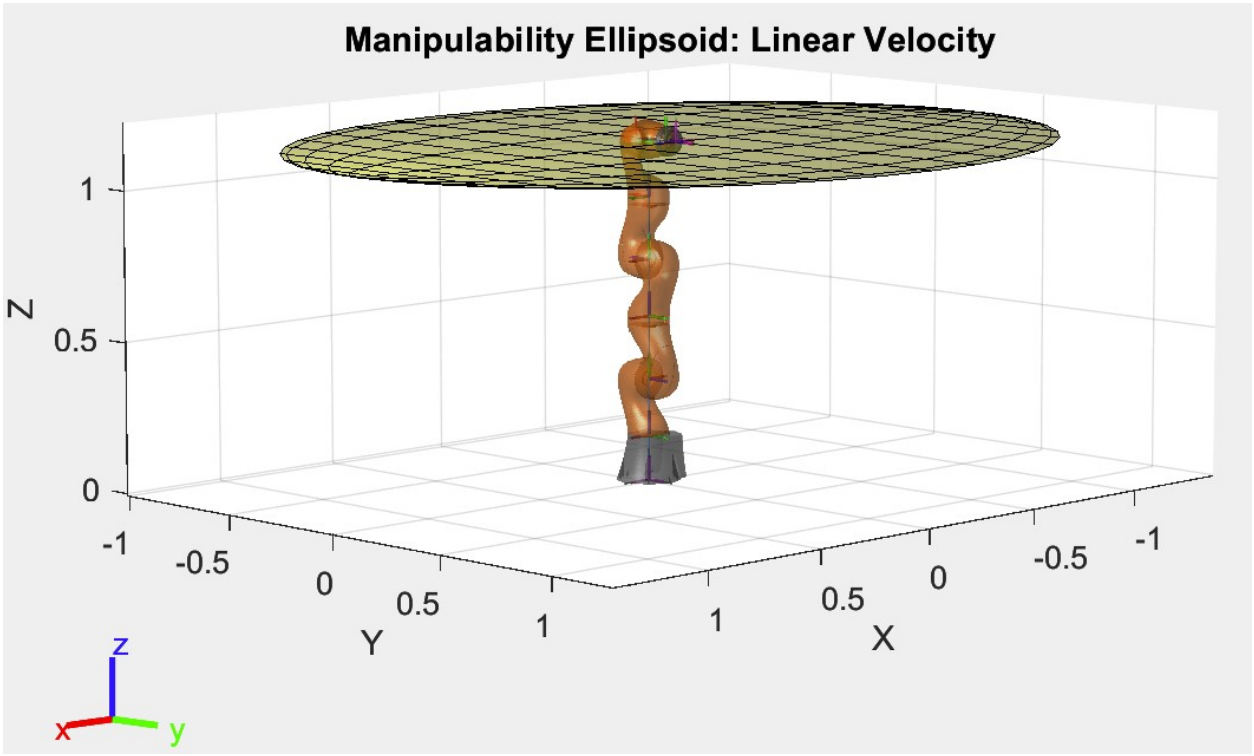
$$Condition\ Number = \mu2(A) = \frac{\lambda max(A)}{\lambda min(A)}$$

$$Volume = \mu3(A) = \sqrt{\lambda1(A) * \lambda2(A)...} = \sqrt{det(A)}.$$

Test cases:
Linear:

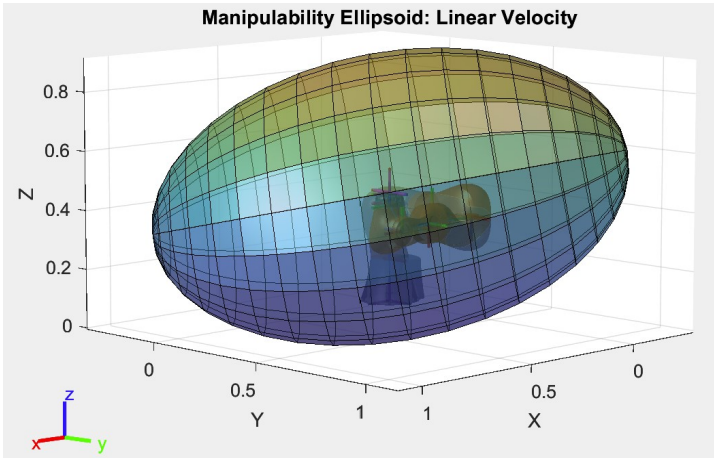
	Thetas	Istrophy	Condition number	Ellipsoid Volume
Case 1	0 0 0 0 pi/2 pi/2 pi/2	Infinity	Infinity	0
Case 2	pi/2 pi/2 pi/2 pi/2 pi/2 pi/2 pi/2	1.5409	2.3742	0.4399
Case 3	(in degrees) -41 22 -147 -71 76 18 -105	2.8698	8.2358	0.3899

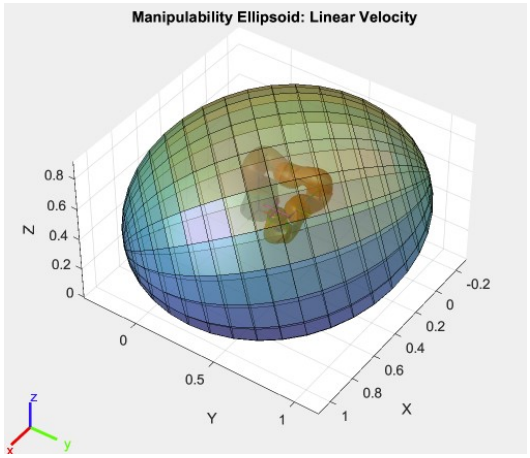
Case 1:



From the plot, it can be seen that the ellipsoid is 2D, and has a singularity along the Z axis. The robot is not able to move further along that axis, and there are more than one collinear revolute axes. This result is consistent with what we found in part F for this test case: that it is at a singularity. It makes sense that the istrophy is infinity, since the smallest semi-axis length is 0, the istrophy value would be divided by 0. The condition number makes sense since infinity^2 is still infinity. The ellipsoid volume being zero makes sense since it's not ellipsoid (it's 2D), and the equation for the volume involves all semi-axial lengths^(½) to be multiplied together.

Case 2:

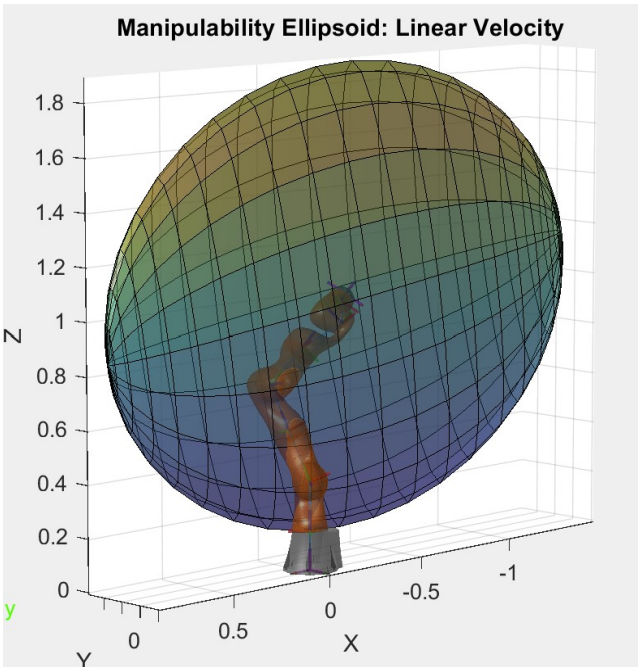


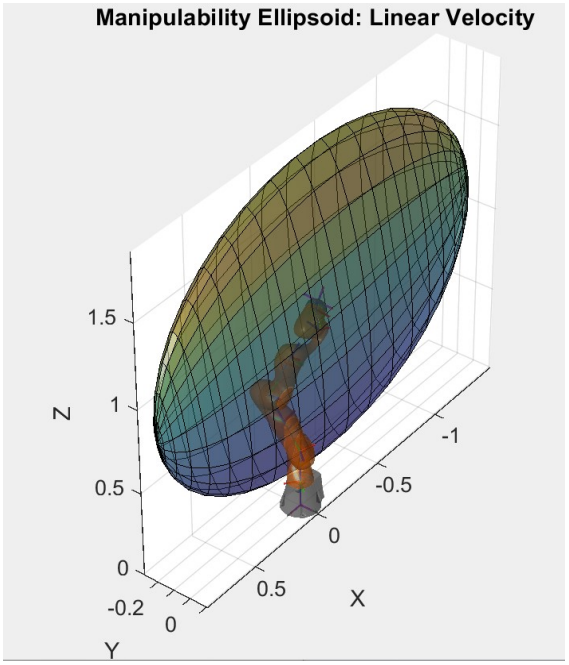


In this configuration, the robot is not at a singularity, and the ellipsoid is close to equal in size in each direction corresponding to the robot's instantaneous linear speed in each direction. The istrophy values are reasonable, looking at the ellipsoid's largest dimension, it looks about 1.5 of the smallest. The condition

number is equal to the istrophy value squared. The volume value makes since, since the λ values in each direction are small but close to equal in size, making it the largest of the 3 test conditions.

Case 3:





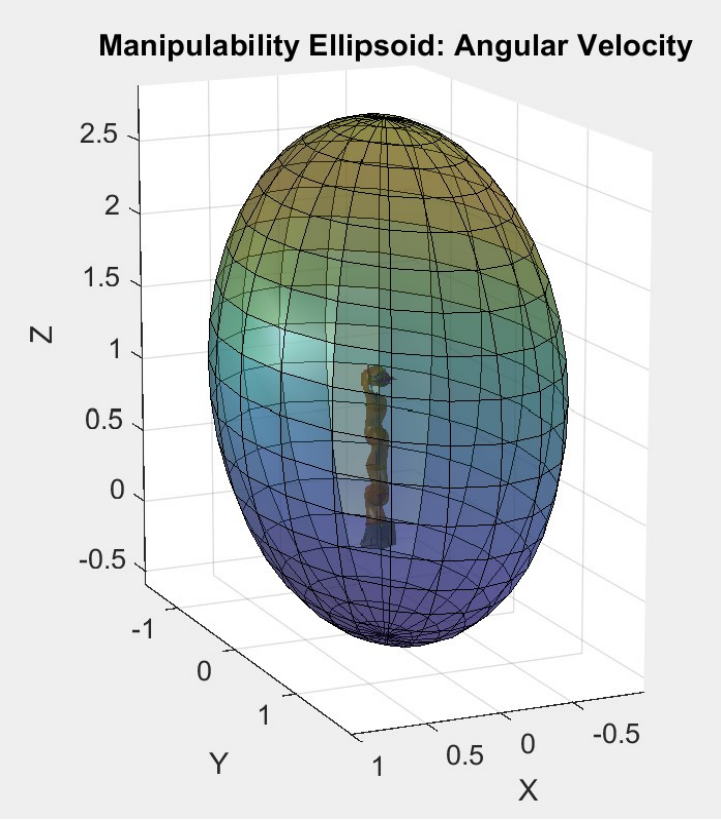
In this configuration, the robot is not at a singularity, but is almost at a singularity in the y direction. It would take much more effort for the robot to move in this direction. In the other directions, the robot can move more freely. The large istrophy value make sense, since the largest dimension of the ellipse is much larger than the smallest. The condition number is the istrophy^2. The volume makes sense as well – it is almost as large as the volume of the ellipse in case 2, even though they are very different shapes. For this condition the largest dimension is larger than in case 2.

Angular:

	Thetas	Istrophy	Condition number	Ellipsoid Volume
Case 1	0 0 0 0 pi/2 pi/2 pi/2	1.7321	3	3
Case 2	pi/2 pi/2 pi/2 pi/2 pi/2	1.2247	1.5000	3.4641

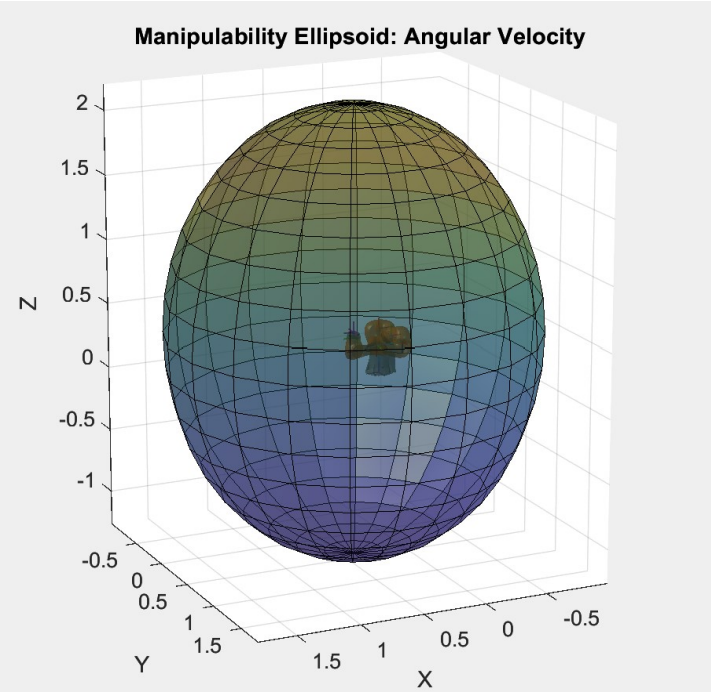
	pi/2 pi/2			
Case 3	(in degrees) -41 22 -147 -71 76 18 -105	1.5290	2.3377	3.2653

Case 1:



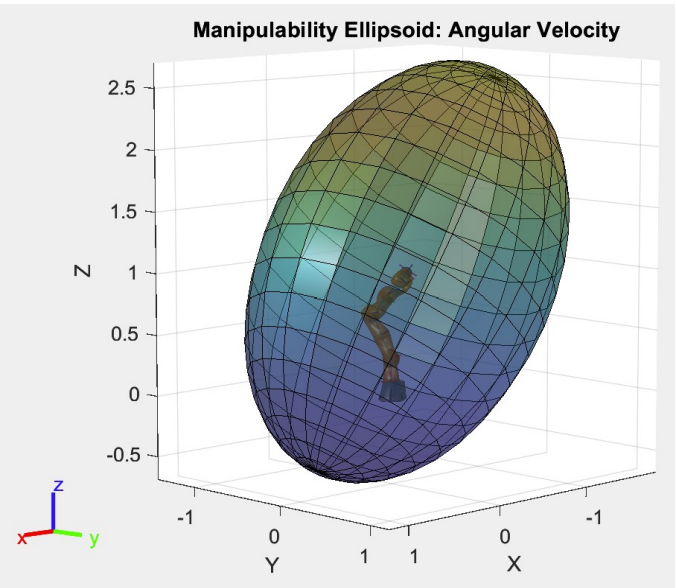
These ellipses represents how easily the robot can rotate along each axis. The istrophy value makes sense; the largest dimension of the ellipse looks about 1.7 of the smallest. The condition number is istrophy^2. The volume seems reasonable, as the dimensions are larger in comparison to the linear velocity ellipses. While the robot can't translate linearly (there is a singularity) the robot can still rotate along all the axes.

Case 2:



The istrophy value makes sense; this ellipse is close to circular, so the istrophy should be close to 1 (all dimensions are close in size). The condition number is istrophy^2. The volume seems reasonable, as the dimensions are large (compared to the other cases) in each dimension.

Case 3:



The istrophy value makes sense; the dimensions of the ellipse don't look very similar in size, with the largest looking about 1.5 of the smallest. The condition number is istrophy^2. The volume seems reasonable, similar to case 2, but rotated and stretched.

The code for this problem is in "ellipsoid_plot_angular.m", "ellipsoid_plot_linear.m"), "J_isotropy.m", "J_condition.m", "J_ellipsoid_volume.m".

H. Numerical inverse kinematics

The numerical inverse kinematics algorithm follows from the formulation introduced by Lynch and Park [1].

$$\theta^{i+1} = \theta^i + J_b^\dagger(\theta^i)\mathcal{V}_b$$

$$[\mathcal{V}_b] = \log T_{bd}(\theta^i)$$

$$T_{bd}(\theta^i) = T_{sb}^{-1}(\theta^i)T_{sd} = T_{bs}(\theta^i)T_{sd}$$

The Moore-Penrose psuedoinverse of the body jacobian is used along with the body twist vector to calculate the necessary change in angle to converge to the desired end effector configuration.

The J_inverse_kinematics function takes as inputs the desired configuration of the robot, the joint angles of the robot at the starting configuration, the screw axis matrix of the robot, the home configuration of the robot M, and the desired error thresholds for the rotational and translational velocities.

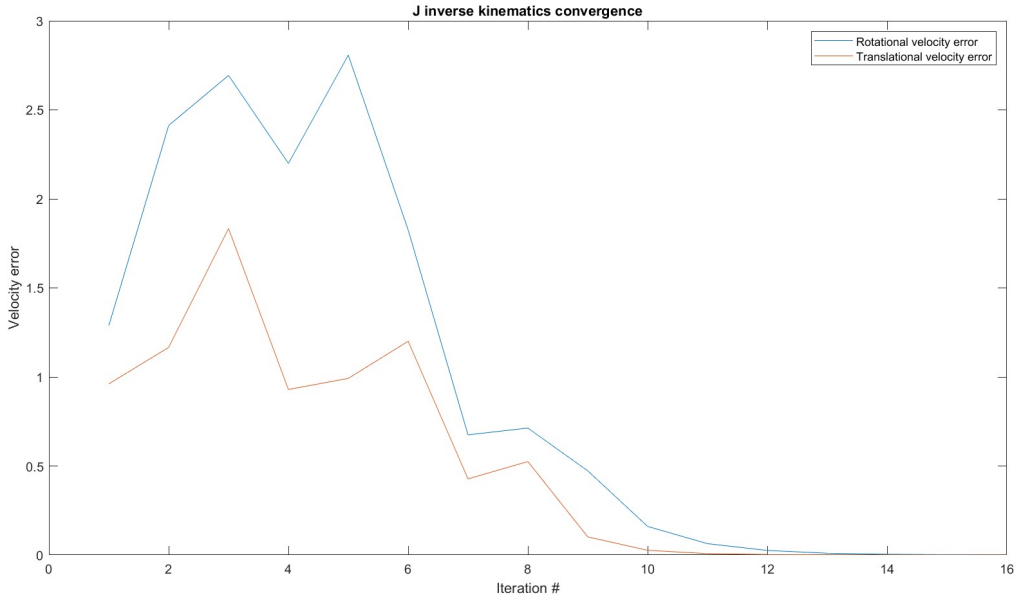
The output of the function includes the joint angles of the robot at each iteration of the algorithm until convergence, the errors after each iteration, and the manipulability ellipsoid volume through each iteration.

The function was tested with three different desired configurations. These configurations are the same used for the jacobian testing, and they were generated randomly using MATLAB's random configuration function.

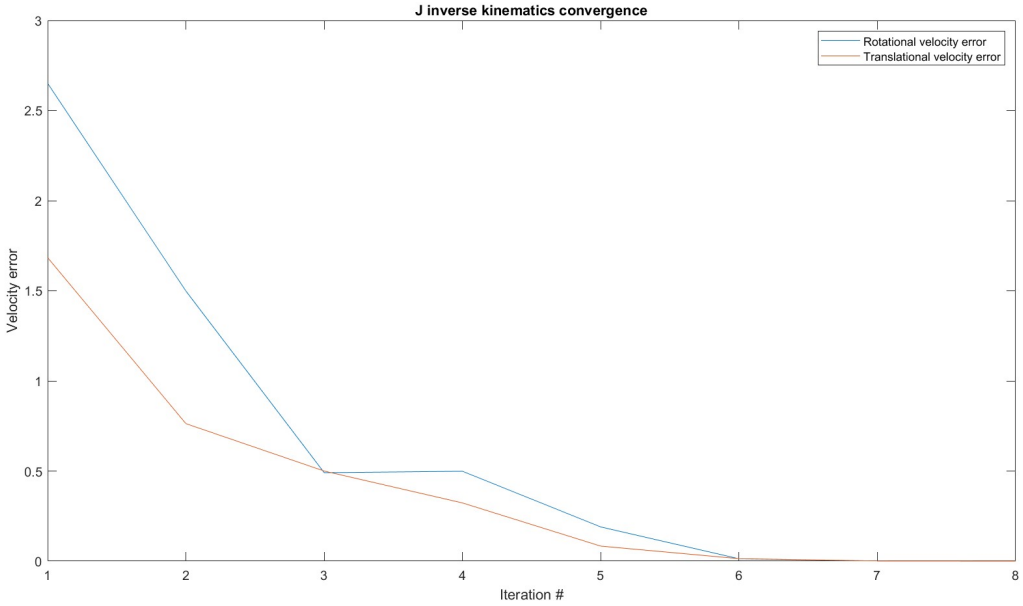
The angles used for the initial guess were based on the angles at the desired configuration but given noise. This noise is scalable by a coefficient and is based on a constant deviation from the joint angles.

The results of the testing can be seen by the plots shown below of the error converging to the desired values.

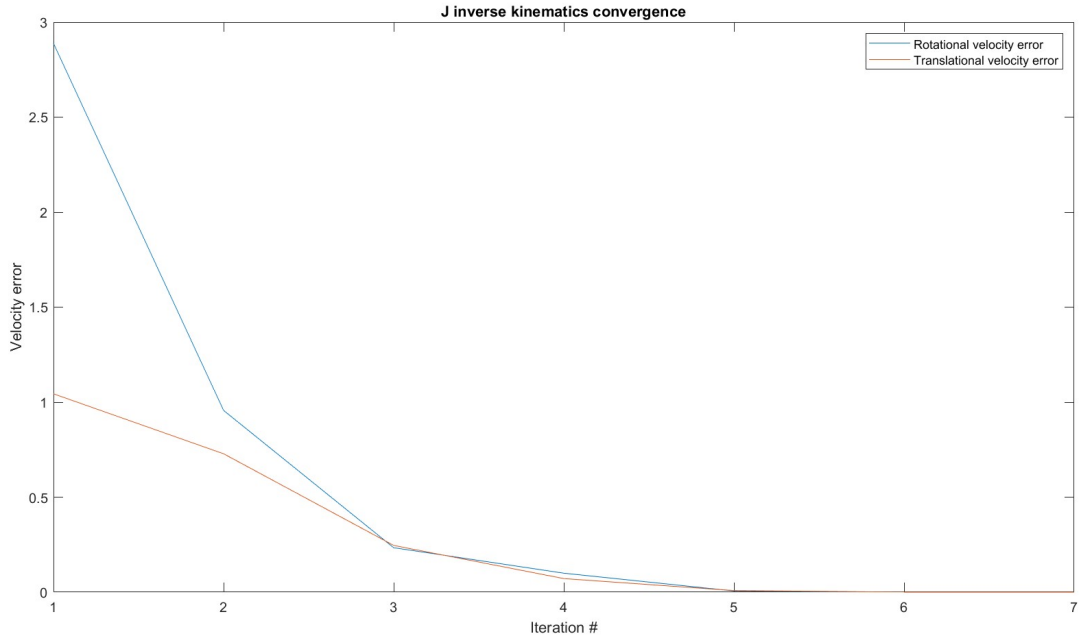
Test Case 1

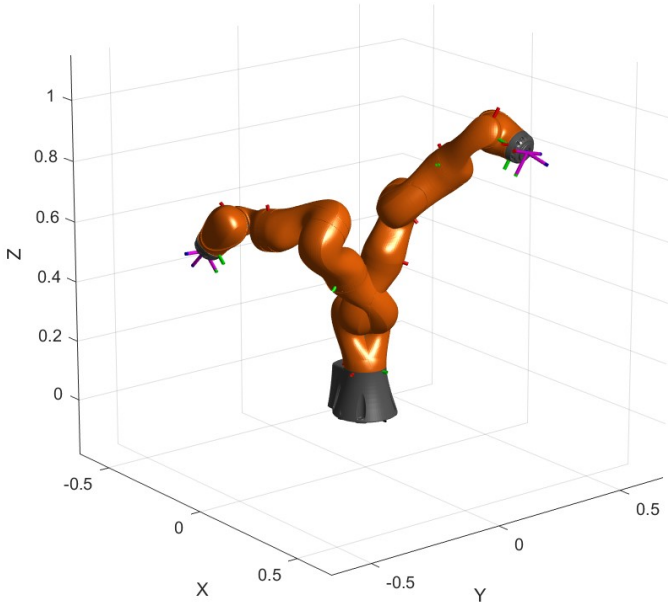


Test Case 2



Test Case 3





Test Configuration 1: Initial Configuration (leftmost) vs. Final Configuration (rightmost)

The test code for this section can be found in “main_ROBOT_w_Test.com”, in section “Inverse Kinematics Testing”, subsection “Test for J_inverse_kinematics”.

I. Jacobian Transpose Algorithm

Similar to the algorithm used above in part G, the jacobian transpose uses the twist vector to solve for the angles at the desired configuration. Instead of using the psuedoinverse of the jacobian however the transpose of the jacobian is used along with a positive definite matrix K in the following formulation.

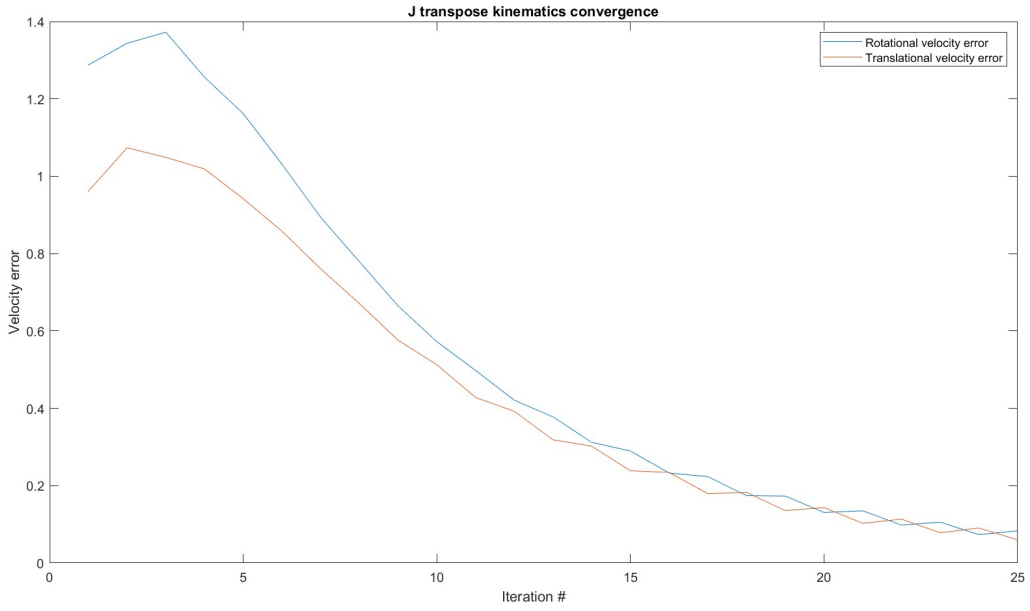
$$\dot{q} = J_s^T(\theta) K v_s$$

The inputs of the function follow from J_inverse_kinematics with the addition of the K matrix

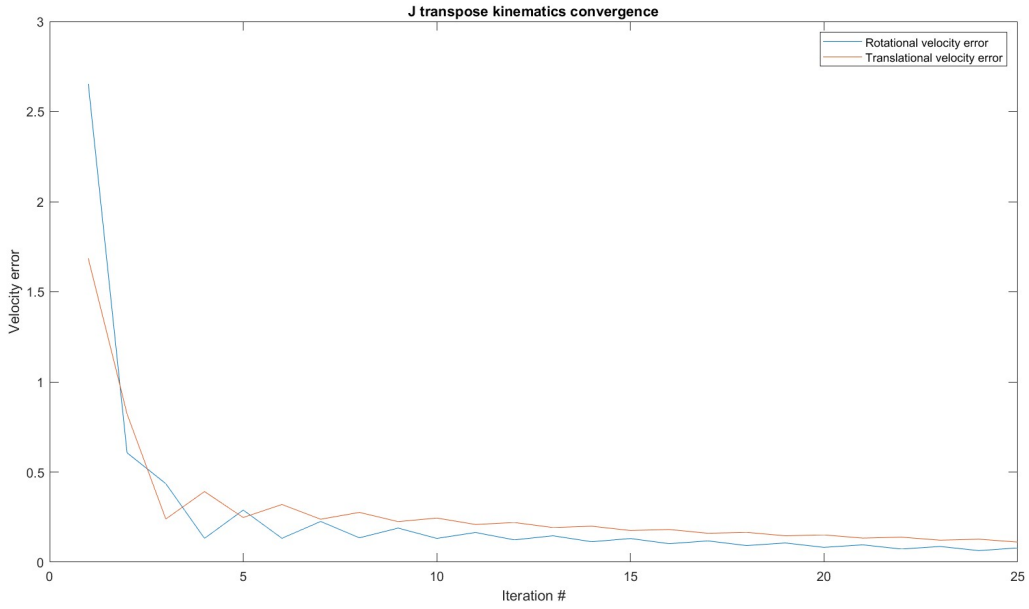
The outputs of the function are the same as those of J_inverse_kinematics which are the joint angles at the desired configuration for each iteration of the algorithm, the errors throughout and the manipulability measure.

The test cases used for this function are the same as that of J_inverse_kinematics, with the results shown below.

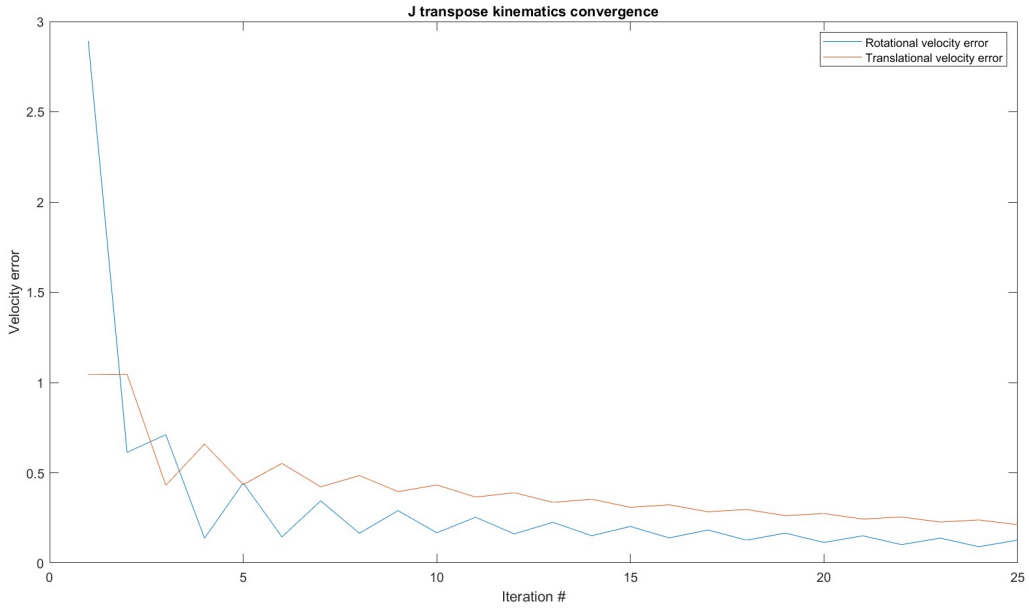
Test Case 1



Test Case 2



Test Case 3



The test code for this section can be found in “main_ROBOT_w_Test.com”, in section “Inverse Kinematics Testing”, subsection “Test for J_transpose_kinematics”.

J. Redundancy resolution

The LBR iiwa 7 is a redundant arm when it is nonsingular given that it has 7 joints. To solve the inverse kinematics problem, several joint angles may result in the same desired configuration. To this end, the redundancy can be leveraged to optimize for certain behaviors in the process of moving between configurations.

Here we use redundancy to avoid singularities by defining a secondary objective function shown below [3].

$$w(\theta) = \sqrt{\det\left(J_s(\theta)J_s^T(\theta)\right)}$$

$$\dot{q}_0 = k_0\left(\frac{\partial w(\theta)}{\partial \theta}\right)^T$$

$$\dot{q} = J^{\dagger}v_s + (I_n - J^{\dagger}J)\dot{q}_0$$

This function represents the volume of the manipulability ellipsoid.

The gradient of this objective function is then calculated for each joint angle and by doing so we can use the direction of maximum ascent (greatest gradient) to contribute to the calculation of joint angles by adding it to the expression used in part G that minimizes the norm of joint velocities.

We faced a challenge in calculating the symbolic gradient of the objective function in MATLAB. The computational time of a single iteration was significant enough to justify using other methods to obtain the gradient.

Our approach was a numerical one that can be seen in the following formulation.

$$\frac{\partial w}{\partial \theta} = \begin{bmatrix} \dfrac{w(\theta + d\theta_1) - w(\theta)}{d\theta_1} \\ \dfrac{w(\theta + d\theta_2) - w(\theta)}{d\theta_2} \\ \vdots \\ \dfrac{w(\theta + d\theta_7) - w(\theta)}{d\theta_7} \end{bmatrix}$$

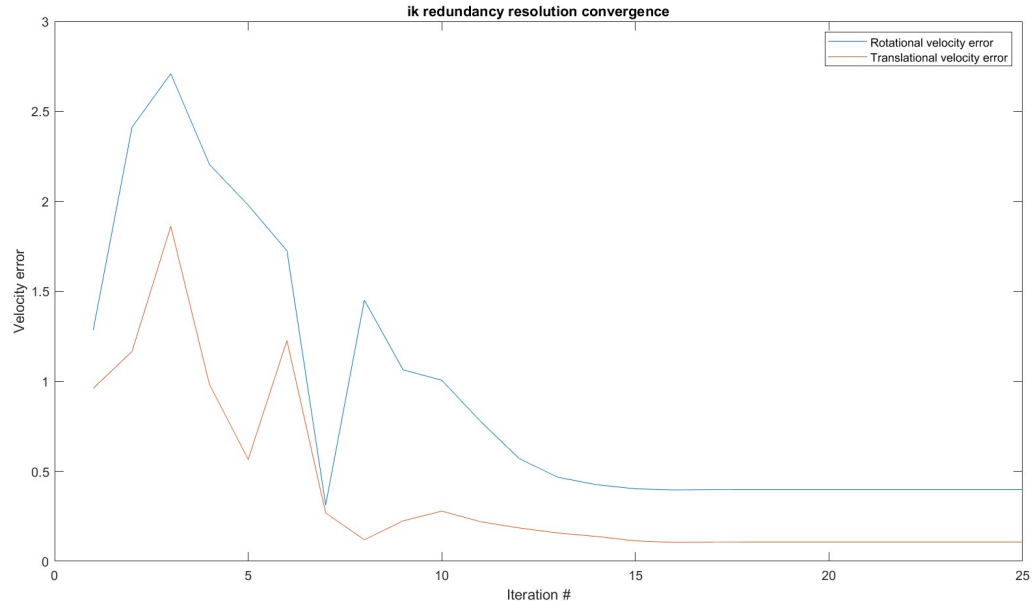
While this is not as accurate as using the symbolic partial differentiation method, it was much more computationally feasible.

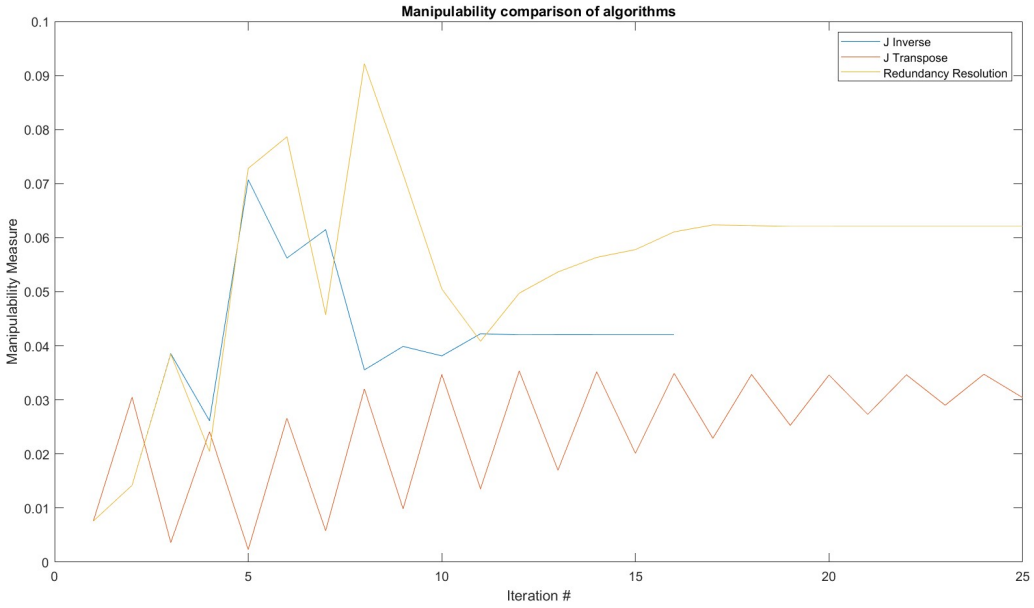
One calculated the gradient was used in the following equation, wherein k is a weighting factor for the secondary objective function.

The ik_Redundancy_Resolution function takes the same inputs as the J_transpose_kinematics function, and produces the same outputs.

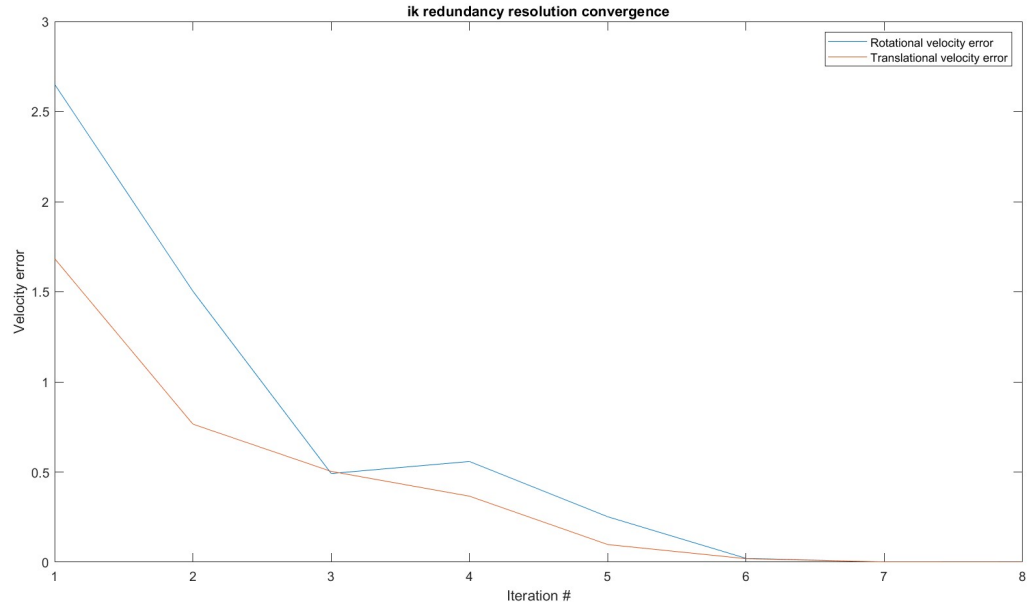
The test cases used for this function are the same as those used in the previous sections. The results are as follows. One set of plots shows the error after each iteration and the other shows the manipulability measure compared to the other two algorithms after each iteration. We would expect that this algorithm shows an increase in manipulability in comparison.

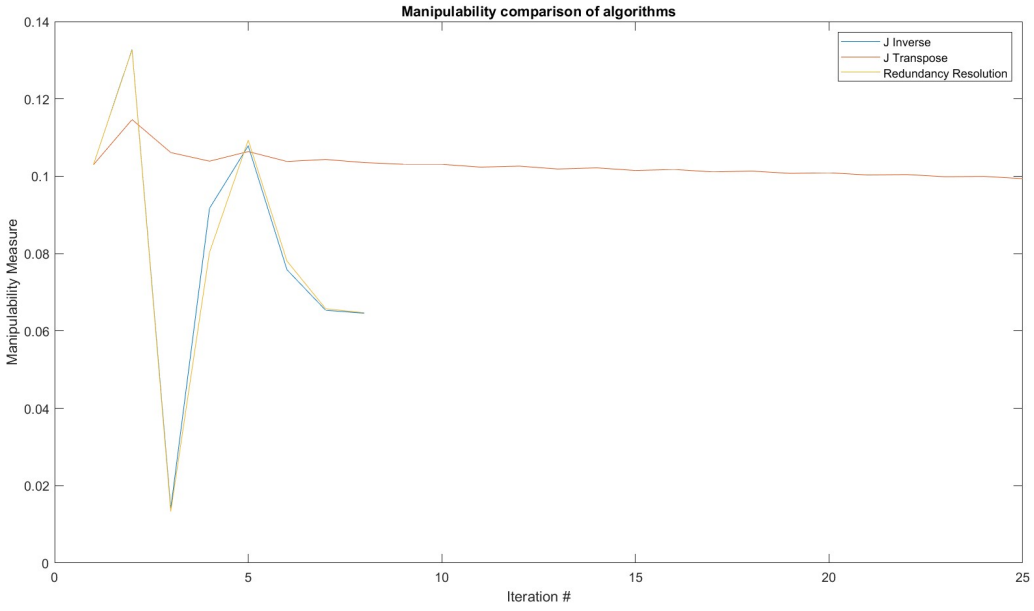
Test Case 1


ik redundancy resolution convergence
Rotational velocity error
Translational velocity error
Velocity error
Iteration #
0
0.5
1
1.5
2
2.5
3
0
5
10
15
20
25

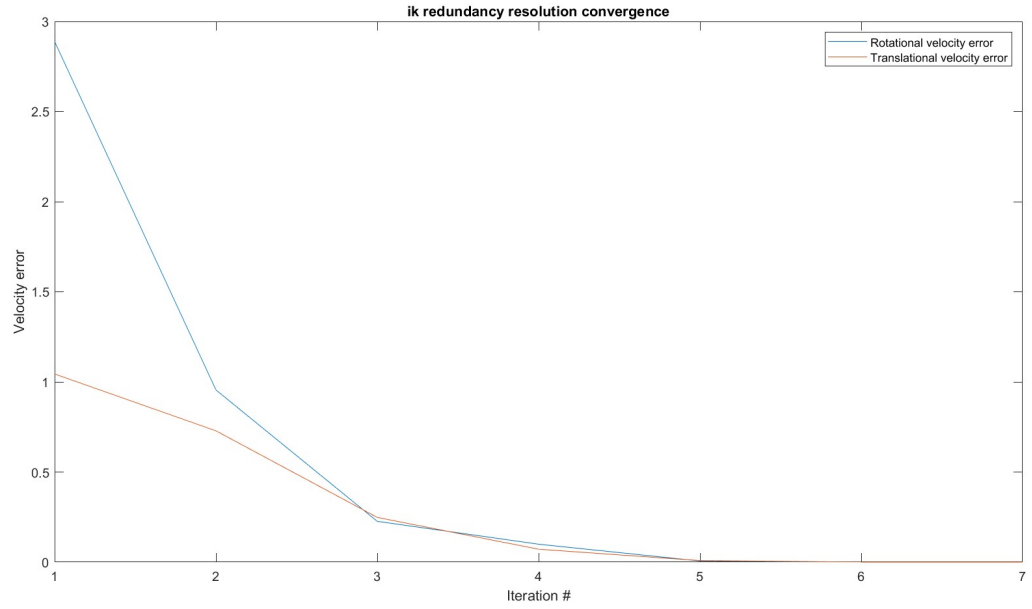

Manipulability comparison of algorithms
J Inverse
J Transpose
Redundancy Resolution
Manipulability Measure
Iteration #
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
0
5
10
15
20
25

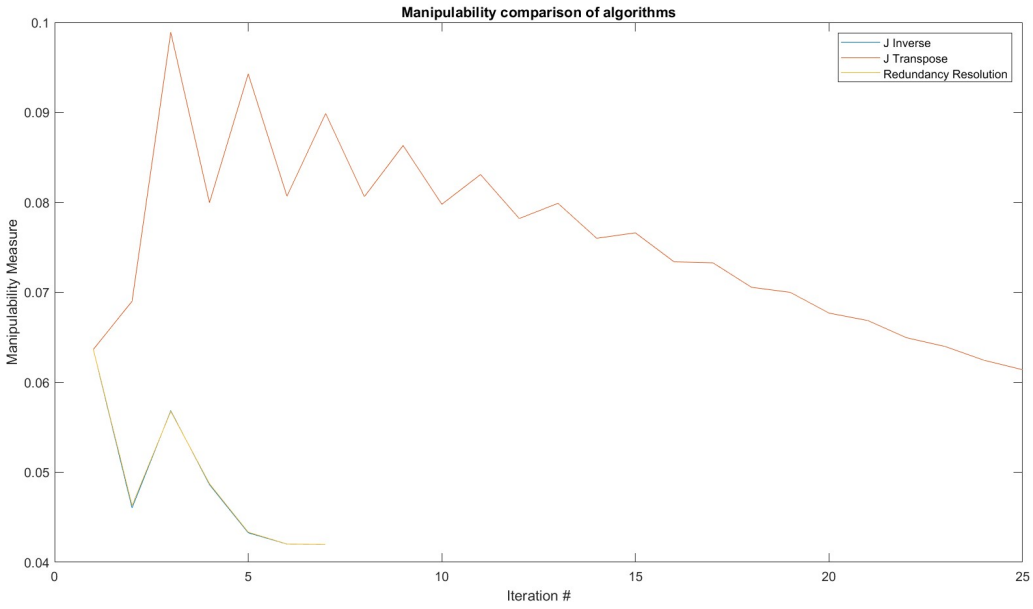
Test Case 2





Test Case 3





The test code for this section can be found in “main_ROBOT_w_Test.com”, in section “Inverse Kinematics Testing”, subsection “Test for redundancy_resolution”.

It is evident that in test case 1 the manipulability measure is greater than those of the other two algorithms. However, for the other two cases, it is almost identical to the inverse jacobian method.

More test cases should be used to further identify the difference between algorithms in terms of resulting manipulability measures. One note regarding test case 1 is that the manipulator reaches the joint limit and does not move away from it, therefore, the error does not converge to the desired values.

A comparison of the three algorithms converging to the same end effector configuration shows that the J_inverse_kinematics algorithm and the inverse kinematics with redundancy resolution converge in a fewer number of iterations than the jacobian transpose algorithm.

M. (Bonus) Graphical simulation of the robot in ROS, Matlab Robotic System Toolbox, VREP, or similar software has extra bonus.

This was done throughout the document for the test cases (not a video).

How the work was split up:
HA1: both worked on this
HA2: ellie
HA3: ellie
HA4: zuhair
PA.A: both worked on this
PA.B: both worked on this
PA.C: both worked on this
PA.D: both worked on this
PA.E: both worked on this
PA.F: ellie primarily, but both worked/debugged
PA.G: ellie primarily, but both worked/debugged
PA.H: zuhair primarily, but both worked/debugged
PA.I: zuhair primarily, but both worked on this
PA.J: zuhair primarily, but both worked/debugged

REFERENCES:

[1] Lynch and Park, “Modern Robotics,” Cambridge U. Press, 2017 Chapters 4-5.

[2] KUKA, “LBR iiwa 7 R800 Datasheet”, https://www.kuka.com. PDF.

[3] Bruno, Siciliano, Sciavicco Lorenzo, Villani Luigi, and Oriolo Giuseppe. “Robotics: modelling, planning and control.” Advanced Textbooks in Control & Signal Processing 4 (2009(: 76-82, Chapter 3.