**Practical Assessment 2 – Advanced Robotics – Tran Minh Hoang – s3818101**

----------------------------------------------------------------------------------

**Problem 1:**

**1.1**

* **Identifying DH parameters.**

I chose given reference frames to identify DH parameters.

DH parameters are allocated **manually** with the method:

A close-up of a white background

Description automatically generated

With h = 0.672m, d3 = 0.15m, a2 = 0.4318 m, a3 = 0.0203 m, d4 = 0.4318m, d5 = 0.1m, we have:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Link index | Link length | Link twist () | Link offset () | Joint angle () |
|  | **1** | 0 | 0 | 0 |  |
|  | **2** | 0 | -90 | 0 |  |
|  | **3** | a2 = 0.4318 | 0 | d3 = 0.15 |  |
|  | **4** | a3 = 0.0203 | -90 | d4 = 0.4318 |  |
| **5** | 0 | 90 | 0 |  |
| **6** | 0 | -90 | 0 |  |
|  | **7** | 0 | 0 | d5=0.1m | 0 |

* **Finding Forward kinematics**

Recall the general transformation matrix for **DH parameters** from the i-1th to the ith reference frame:

A close-up of a number

Description automatically generated

Method to find :

* With each transformation matrix () formed with the formula, the is found:

Ex\_1\_Part\_1\_1\_Forward\_Kinematics.m is created to find **transformation matrix**  with the above algorithm:



The result of is:

|  |  |  |  |
| --- | --- | --- | --- |
| s6\*(c4\*s1 + s4\*(c1\*s2\*s3 - c1\*c2\*c3)) + c6\*(c5\*(s1\*s4 - c4\*(c1\*s2\*s3 - c1\*c2\*c3)) - s5\*(c1\*c2\*s3 + c1\*c3\*s2)) | c6\*(c4\*s1 + s4\*(c1\*s2\*s3 - c1\*c2\*c3)) - s6\*(c5\*(s1\*s4 - c4\*(c1\*s2\*s3 - c1\*c2\*c3)) - s5\*(c1\*c2\*s3 + c1\*c3\*s2)) | - s5\*(s1\*s4 - c4\*(c1\*s2\*s3 - c1\*c2\*c3)) - c5\*(c1\*c2\*s3 + c1\*c3\*s2) | (2159\*c1\*c2)/5000 - (3\*s1)/20 - (s5\*(s1\*s4 - c4\*(c1\*s2\*s3 - c1\*c2\*c3)))/10 - (c5\*(c1\*c2\*s3 + c1\*c3\*s2))/10 - (203\*c1\*s2\*s3)/10000 + (203\*c1\*c2\*c3)/10000 - (2159\*c1\*c2\*s3)/5000 - (2159\*c1\*c3\*s2)/5000 |
| - s6\*(c1\*c4 - s4\*(s1\*s2\*s3 - c2\*c3\*s1)) - c6\*(c5\*(c1\*s4 + c4\*(s1\*s2\*s3 - c2\*c3\*s1)) + s5\*(c2\*s1\*s3 + c3\*s1\*s2)) | s6\*(c5\*(c1\*s4 + c4\*(s1\*s2\*s3 - c2\*c3\*s1)) + s5\*(c2\*s1\*s3 + c3\*s1\*s2)) - c6\*(c1\*c4 - s4\*(s1\*s2\*s3 - c2\*c3\*s1)) | s5\*(c1\*s4 + c4\*(s1\*s2\*s3 - c2\*c3\*s1)) - c5\*(c2\*s1\*s3 + c3\*s1\*s2) | (3\*c1)/20 + (2159\*c2\*s1)/5000 + (s5\*(c1\*s4 + c4\*(s1\*s2\*s3 - c2\*c3\*s1)))/10 - (c5\*(c2\*s1\*s3 + c3\*s1\*s2))/10 - (2159\*c2\*s1\*s3)/5000 - (2159\*c3\*s1\*s2)/5000 - (203\*s1\*s2\*s3)/10000 + (203\*c2\*c3\*s1)/10000 |
| s4\*s6\*(c2\*s3 + c3\*s2) - c6\*(s5\*(c2\*c3 - s2\*s3) + c4\*c5\*(c2\*s3 + c3\*s2)) | s6\*(s5\*(c2\*c3 - s2\*s3) + c4\*c5\*(c2\*s3 + c3\*s2)) + c6\*s4\*(c2\*s3 + c3\*s2) | c4\*s5\*(c2\*s3 + c3\*s2) - c5\*(c2\*c3 - s2\*s3) | (2159\*s2\*s3)/5000 - (2159\*c2\*c3)/5000 - (203\*c2\*s3)/10000 - (203\*c3\*s2)/10000 - (2159\*s2)/5000 - (c5\*(c2\*c3 - s2\*s3))/10 + (c4\*s5\*(c2\*s3 + c3\*s2))/10 |
| 0 | 0 | 0 | 1 |

Note that:

* s1, s2, s3, s4, s5, s6 are sin(t1), sin(t2), sin(t3), sin(t4), sin(t5), sin(t6)
* c1, c2, c3, c4, c5, c6 are cos(t1), cos(t2), cos(t3), cos(t4), cos(t5), cos(t6)

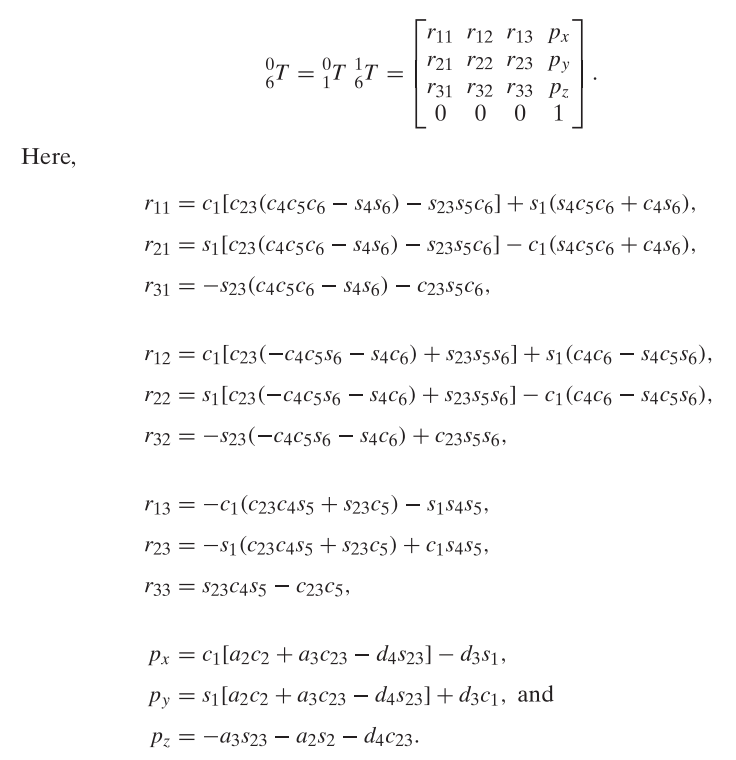
While t1, t2, t3, t4, t5, t6 are .

It is clear that the answer is only with respect to 6 thetas ().

Checking with the solution from the book, we have each :

|  |  |
| --- | --- |
|  |  |

With the final Forward Kinematic answer - :



It is obvious that the book’s result and my result are identical. In the book, cos, cos, sin, sin are reduced into and , which are and while in my result, they are kept unchanged. However, 2 results refer to 1 transformation matrix.

**Note that the book does not derive the end-effector position but only the 6th reference frame of the robot.**

**1.2 Derive Jacobian with MATLAB.**

The approach:

The transformation matrix describes the end-effector position at , which can be used to determine Jacobian Matrix

A group of math equations

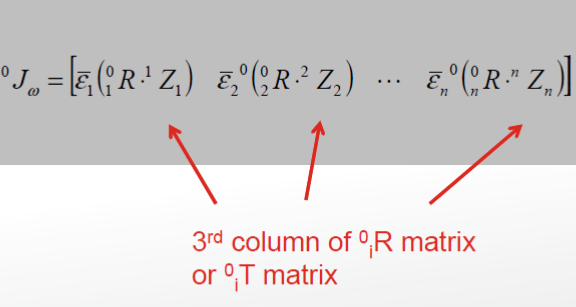
Description automatically generated

In this exercise, all joints are revolute, only angles are considered in the partial derivative.

Call , the partial derivative of A with respect to ( ) is actually the Jacobian elements for **linear velocities** located from row 1 to 3 in the ith column of the Jacobian matrix.

Example:

While with **angular velocities**, from the formula from the lecture:



Continuing with the previous example, we have:

is taking the 3rd column of the ith transformation matrix from only position 1 to 3.

The result:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| (c5\*(c2\*s1\*s3 + c3\*s1\*s2))/10 - (2159\*c2\*s1)/5000 - (s5\*(c1\*s4 + c4\*(s1\*s2\*s3 - c2\*c3\*s1)))/10 - (3\*c1)/20 + (2159\*c2\*s1\*s3)/5000 + (2159\*c3\*s1\*s2)/5000 + (203\*s1\*s2\*s3)/10000 - (203\*c2\*c3\*s1)/10000 | (c5\*(c1\*s2\*s3 - c1\*c2\*c3))/10 - (2159\*c1\*s2)/5000 + (2159\*c1\*s2\*s3)/5000 + (c4\*s5\*(c1\*c2\*s3 + c1\*c3\*s2))/10 - (2159\*c1\*c2\*c3)/5000 - (203\*c1\*c2\*s3)/10000 - (203\*c1\*c3\*s2)/10000 | (c5\*(c1\*s2\*s3 - c1\*c2\*c3))/10 + (2159\*c1\*s2\*s3)/5000 + (c4\*s5\*(c1\*c2\*s3 + c1\*c3\*s2))/10 - (2159\*c1\*c2\*c3)/5000 - (203\*c1\*c2\*s3)/10000 - (203\*c1\*c3\*s2)/10000 | -(s5\*(c4\*s1 + s4\*(c1\*s2\*s3 - c1\*c2\*c3)))/10 | (s5\*(c1\*c2\*s3 + c1\*c3\*s2))/10 - (c5\*(s1\*s4 - c4\*(c1\*s2\*s3 - c1\*c2\*c3)))/10 | 0; |
| (2159\*c1\*c2)/5000 - (3\*s1)/20 - (s5\*(s1\*s4 - c4\*(c1\*s2\*s3 - c1\*c2\*c3)))/10 - (c5\*(c1\*c2\*s3 + c1\*c3\*s2))/10 - (203\*c1\*s2\*s3)/10000 + (203\*c1\*c2\*c3)/10000 - (2159\*c1\*c2\*s3)/5000 - (2159\*c1\*c3\*s2)/5000 | (c5\*(s1\*s2\*s3 - c2\*c3\*s1))/10 - (2159\*s1\*s2)/5000 - (203\*c2\*s1\*s3)/10000 - (203\*c3\*s1\*s2)/10000 + (2159\*s1\*s2\*s3)/5000 + (c4\*s5\*(c2\*s1\*s3 + c3\*s1\*s2))/10 - (2159\*c2\*c3\*s1)/5000 | (c5\*(s1\*s2\*s3 - c2\*c3\*s1))/10 - (203\*c2\*s1\*s3)/10000 - (203\*c3\*s1\*s2)/10000 + (2159\*s1\*s2\*s3)/5000 + (c4\*s5\*(c2\*s1\*s3 + c3\*s1\*s2))/10 - (2159\*c2\*c3\*s1)/5000 | (s5\*(c1\*c4 - s4\*(s1\*s2\*s3 - c2\*c3\*s1)))/10 | (c5\*(c1\*s4 + c4\*(s1\*s2\*s3 - c2\*c3\*s1)))/10 + (s5\*(c2\*s1\*s3 + c3\*s1\*s2))/10 | 0; |
| 0 | (2159\*c2\*s3)/5000 - (203\*c2\*c3)/10000 - (2159\*c2)/5000 + (2159\*c3\*s2)/5000 + (203\*s2\*s3)/10000 + (c5\*(c2\*s3 + c3\*s2))/10 + (c4\*s5\*(c2\*c3 - s2\*s3))/10 | (2159\*c2\*s3)/5000 - (203\*c2\*c3)/10000 + (2159\*c3\*s2)/5000 + (203\*s2\*s3)/10000 + (c5\*(c2\*s3 + c3\*s2))/10 + (c4\*s5\*(c2\*c3 - s2\*s3))/10 | -(s4\*s5\*(c2\*s3 + c3\*s2))/10 | (s5\*(c2\*c3 - s2\*s3))/10 + (c4\*c5\*(c2\*s3 + c3\*s2))/10 | 0; |
| 0 | -s1 | -s1 | - c1\*c2\*s3 - c1\*c3\*s2 | - c4\*s1 - s4\*(c1\*s2\*s3 - c1\*c2\*c3) | - s5\*(s1\*s4 - c4\*(c1\*s2\*s3 - c1\*c2\*c3)) - c5\*(c1\*c2\*s3 + c1\*c3\*s2); |
| 0 | c1 | c1 | - c2\*s1\*s3 - c3\*s1\*s2 | c1\*c4 - s4\*(s1\*s2\*s3 - c2\*c3\*s1) | s5\*(c1\*s4 + c4\*(s1\*s2\*s3 - c2\*c3\*s1)) - c5\*(c2\*s1\*s3 + c3\*s1\*s2); |
| 1 | 0 | 0 | s2\*s3 - c2\*c3 | -s4\*(c2\*s3 + c3\*s2) | c4\*s5\*(c2\*s3 + c3\*s2) - c5\*(c2\*c3 - s2\*s3); |

Below is the MATLAB code (file name is Ex\_1\_Part\_1\_2\_Jacobian\_Matrix.m):



Note that file Ex\_1\_Part\_1\_1\_Forward\_Kinematics.m must be included in the same folder as Ex\_1\_Part\_1\_2\_Jacobian\_Matrix.m as the code for part 1.1 ran at the top of Ex\_1\_Part\_1\_2\_Jacobian\_Matrix.m

**1.3**

In this assessment, our names are Hoang, Hieu. The string needing to be drawn is “HH”.

The result:

|  |  |
| --- | --- |
| Figure 1.3.1 HH trajectory with the robot touching the finishing point | Figure 1.3.2 Trajectory for 2 letters “HH” in Isometric view |
| Figure 1.3.3 Zoomed trajectory for 2 letters “HH” | |

Discussion:

* Sharp corners are handled by curves to avoid discontinuities, causing the missing right-top corners of letters HH.

The approach:

Using the week3.mlx and PA2\_Question1\_3.mlx given on Canvas, I created a file to generate the 3D trajectory for the 2 letters “HH” and another one to verify whether the trajectory is possible for the robot to draw.

* **About generating the 3D trajectory:**

The code is at file Generate\_letter\_HH.m:



+ The scale of the robot is changed to 0.15 while moving up distance is 0.1 so that the robot’s workspace can include the trajectory.

+ Maximum velocity of the trajectory is kept at 0.5 m/sec

Below is the result of this phase:

|  |  |
| --- | --- |
| Figure 1.3.4 the horizontal plane’s robot traces. | Figure 1.3.5 the 3D trajectory |

The trajectory variable is manually exported into a file called “Letter\_HH\_trajectory.mat” for later use.

* **About whether the trajectory fits the robot’s workspace:**

The code is at file Ex\_1\_Part\_1\_3\_Drawing\_HH.m:



In this phase, the trajectory file is loaded again into the workspace and then, checked if it fits the robot’s workspace with below steps:

1. Assuming that the end-effector points downward while moving, its rotational matrix is always kept at:
2. Taking trajectory data from the trajectory file (, ), we can form the transformation matrix with an offset for components to avoid collision between the end-effector and the base:
3. The transformation matrix is then transferred to joint configuration by inverse kinematics to verify whether the robot can reach the position (If the code line to find inverse kinematics has no error message, then all trajectory points are reachable. With the power of the computer the process would be fast).
4. After a few times of trial-and-error, the offset value of **0.25** is selected.
5. **The PUMA robot of the MATLAB model is smaller than the PUMA robot given in the exercise, which means the trajectory is within the workspace of the PUMA in this exercise.**

**Note that the desired angles for the trajectory will be mentioned in exercise 1.4.**

**1.4**

**The Simulink file to control the robot** is Ex\_1\_4\_Control\_Puma.m

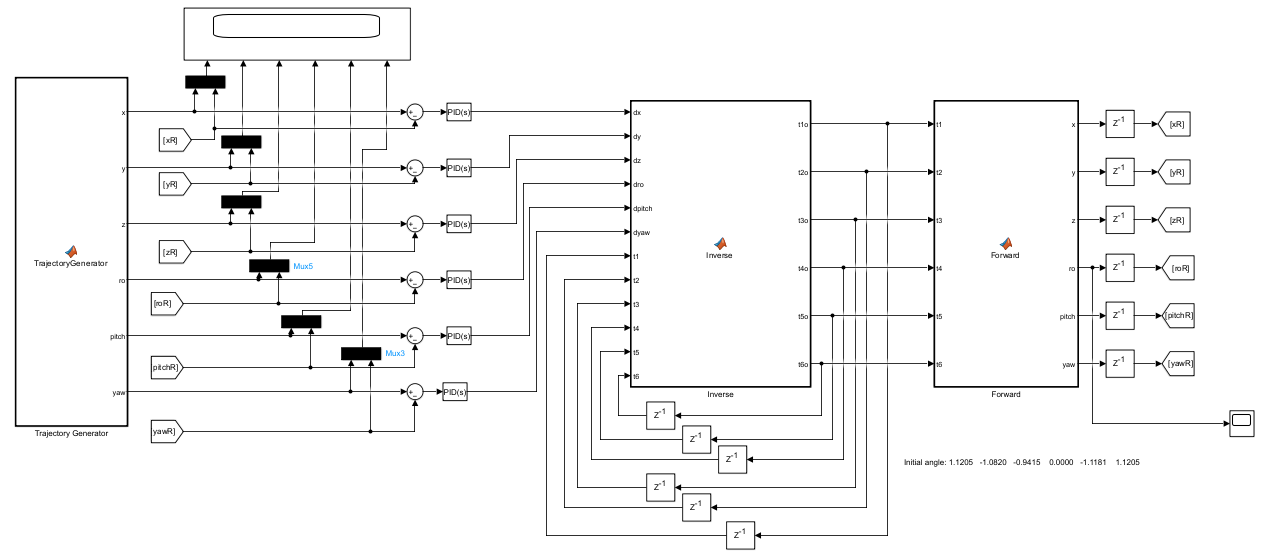
**Control idea:**

1. To be able to control a system, first, we have to generate desired data (x, y, z, ro, pitch, yaw in this exercise). The desired x,y,z are already provided in exercise 1.3.

From the question, => the Euler angles (XYZ rotation) of the end-effector are pi, 0, 0 (rad/s) for alpha, beta, gamma (noted as ro, pitch, yaw in the Simulink file) respectively. Since programming calculations with pi are not always accurate, the desired value for ro is pi – 0.01 while those for pitch and yaw values are 0 and 0.

1. Build **the control block diagram:**

Here is the screenshot of the control block diagram.



With the desired trajectory, a closed loop control system is formed with a trajectory generator to create the desired path of the system, an inverse Jacobian matrix to translate the changes in Cartesian space (x, y, z, ro, pitch, yaw) into the changes of system joint spaces () and thus, get the current joint angles, a forward kinematics block to transfer the current joint angles to the current system’s Cartesian data to be referenced with the desired trajectory.

Note:

* Delay blocks are used at the first step of every run to set initial values of the system.
* In some cases, errors of angles fed to the Inverse block are false. For example, pi and -pi are about the same location but the error is 2\*pi and thus, makes 6-theta changes bigger than those in the real system after going through Jacobian inverse block.
* Hence, first, ro angles, recognized changing a lot after a few tests from -pi to pi, are processed before fed again to the system: if ro < 0, ro will have the new value of ro + pi – 0.001. Secondly, PID controllers are added and tuned to transfer errors to correct signals.
* Since the system needs to stabilize before it can move, an offset time is added to make the robot stays at the starting position for a while before writing the 2 letters “HH”.

1. **Tuning process**

PID blocks for x, y and z are tuned first since their changes are virtually small. Then pitch, yaw, ro.

**Discussion**

A screenshot of a computer

Description automatically generated

The tuned values of PID (from left to right) are:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | y | z | ro | pitch | yaw |
| 10 0 0 | 6 0 0 | 6 0 0 | 1 0 0.01 | 3.6 0 0.1 | 3 0 0.008 |

In the figure, it is clear that the desired and real paths are virtually identical.

**Problem 2:**

**2.1**

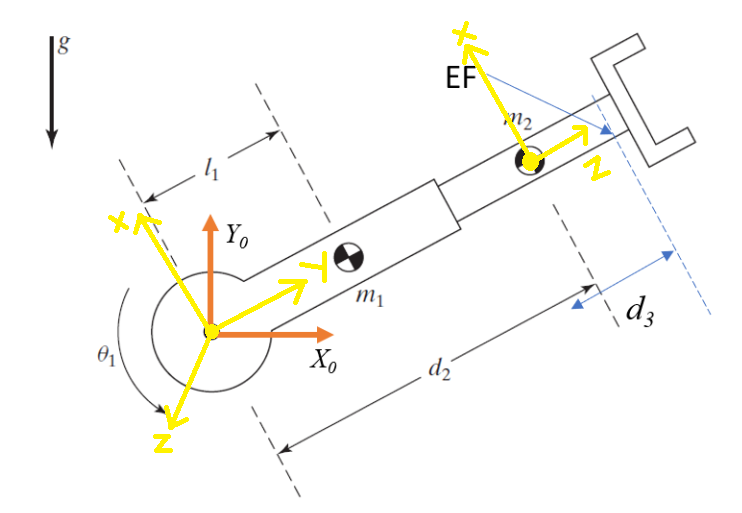


Figure: R-P robot arm with frame

x, y, z at initial point is x1, y1, z1; x, z at midpoint 2 is x2, z2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Link index | Link length | Link twist | Link offset | Join value |
| 1 | 0 | 0 | 0 |  |
| 2 | 0 |  |  | 0 |

Table: DH parameters of R-P robot arm

Forward kinematics:

Inverse kinematic:

x =

y =

=>

=>

**2.2**

We use Lagrangian method to find the solution:

1. Derive T = sum of kinetic energy and V = sum of potential energy
2. Derive torque (for revolute joint) and force (for prismatic joint) with the formula below:

A math equation with numbers and plus and a plus

Description automatically generated with medium confidence

**Code flow:**

Step 1: input the inertia tensor.

I1 =

A black background with white numbers

Description automatically generated

I2 =

A black background with numbers and letters

Description automatically generated

Step 2: input the velocity calculation and position formula for center of mass 1 and center pf mass 2.

x\_c1\_in\_0(t) =



y\_c1\_in\_0(t) =



v\_c1\_in\_0(t) =

A black background with numbers and symbols

Description automatically generated

v\_c1\_in\_0\_short(t) =



x\_c2\_in\_0(t) =



y\_c2\_in\_0(t) =



v\_c2\_in\_0\_short(t) =



Step 3: calculate the kinetic energy.

k1(t) =

A black background with numbers and symbols

Description automatically generated

k2(t) =



k(t) =

A black background with white text

Description automatically generated

Step 4: calculate the potential energy.

u(t) =



du\_dq1(t) =



du\_dq2(t) =



Step 5: apply all to Lagrangian formula.

dk\_dqd1(t) =



dk\_dqd2(t) =



dk\_dqd1\_dt(t) =

dk\_dqd2\_dt(t) =

A black background with white text

Description automatically generated

dk\_dq1(t) =



dk\_dq2(t) =



tau\_1\_short(t) =



tau\_2\_short(t) =



**2.3**

|  |  |  |
| --- | --- | --- |
| Position trajectory | Velocity trajectory | acceleration trajectory |
| Tau | Theta scope | Tau2 |

|  |  |
| --- | --- |
| Theta scope 2 | A screen shot of a graph  Description automatically generatedError of theta and d2 |

Code for this exercise is at \Ex\_2\_5\Ex2\_3.slx

Here is the control diagram:

A diagram of a computer

Description automatically generated

Control flow:

There are 2 versions of controllers: PD and full dynamics compensation to test. All are critically damped.

Desired are generated from with derivative block. Desired are generated from (by derivative).

**2.4**

Below are trajectories of the robots in cubic paths and linear blends.

|  |  |
| --- | --- |
| Cubic path | A graph with a line  Description automatically generatedLinear blend |

**For cubic path**

For each section, there are 4 conditions to solve to set of equations of 4 coefficients of each joint:

* for revolute joint
* for prismatic joint .

All are calculated with the formula (.

All are calculated with the formula ( – d3) (d3 = 0.1m)

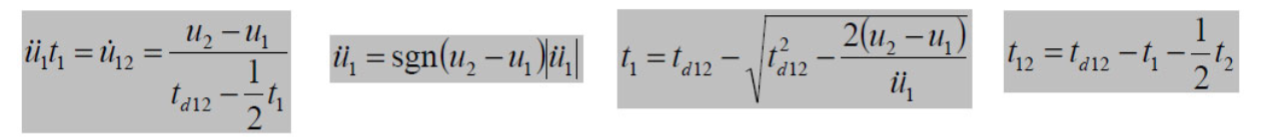
initial of section 1 and ending of section 3 are zero.

connecting 2 sections (1 and 2, 2 and 3) are assumed to be 0.1 and 0.2 respectively.

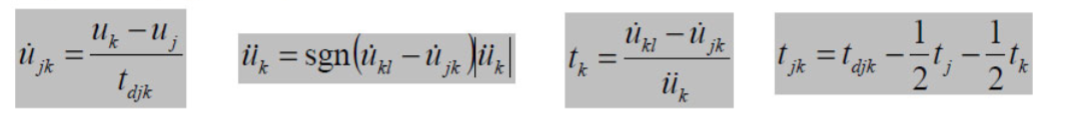
**For linear blend**

There are 3 sections in total.

Segment 1 are calculated as:



Segment 2 are calculated as:



Segment 3 are calculated as:

A close-up of a computer screen

Description automatically generated

After and are determined,

* parabolic blends are generated with the quadratic equation with a = 50 degree/s for revolute joint and 5 m/s^2 for prismatic joint and is the initial velocity () of the considered segment
* linear paths are generated by , with is and the current point of the end-effector.

**Code for cubic paths:**



**Code for linear blend paths:**



**2.5**

Here is the result.

A screen shot of a computer

Description automatically generated

It is clear in the figure that

* the blue and green lines are coincident (real theta and desired theta)
* the red and yellow lines are coincident (real d2 and desired d2)

Code for the Linear Blend Control is \Ex\_2\_5\Ex2\_5.slx

Here is the Simulink block diagram

A diagram of a computer

Description automatically generated

Non-linear control is used for the system with kp and kv = 2\*sqrt(kp)

To run the code, time step must be kept the same in discrete and continuous model block.