D.5 50 Degrees

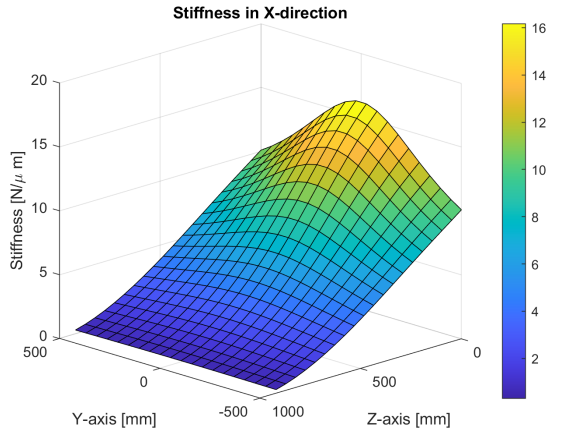

Figure D.25: Stiffness for double sled for load in X-direction at 50° head angle

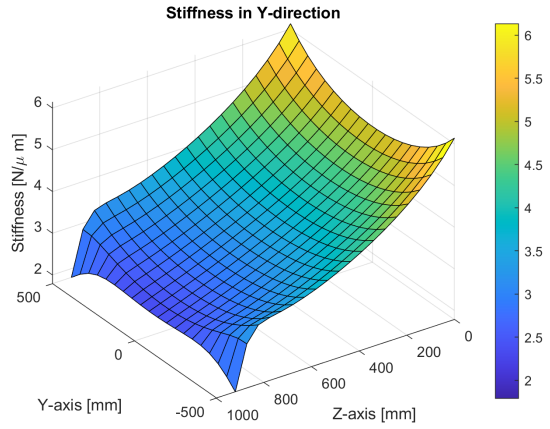

Figure D.26: Stiffness for double sled for load in Y-direction at 50° head angle

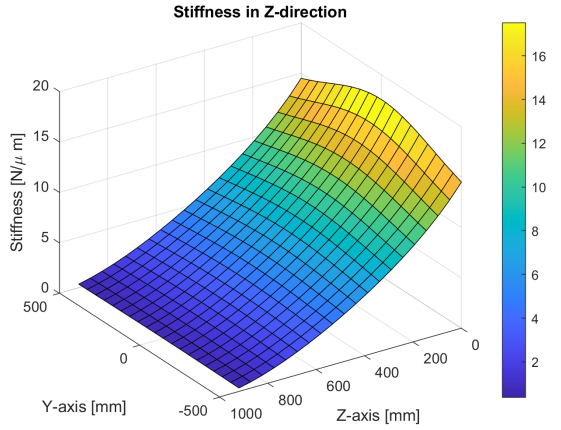

Figure D.27: Stiffness for double sled for load in Z-direction at 50° head angle

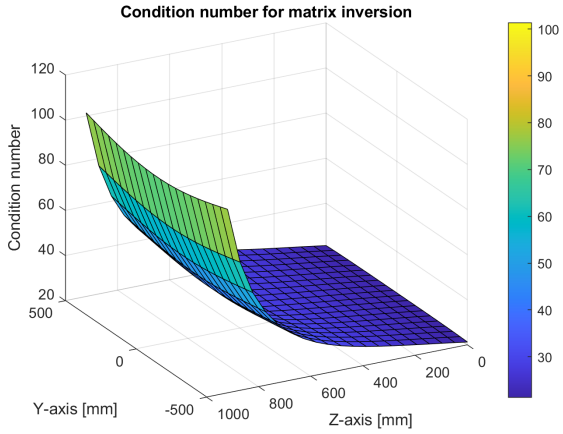

Figure D.28: Double sled condition number for matrix inversion at 50° head angle

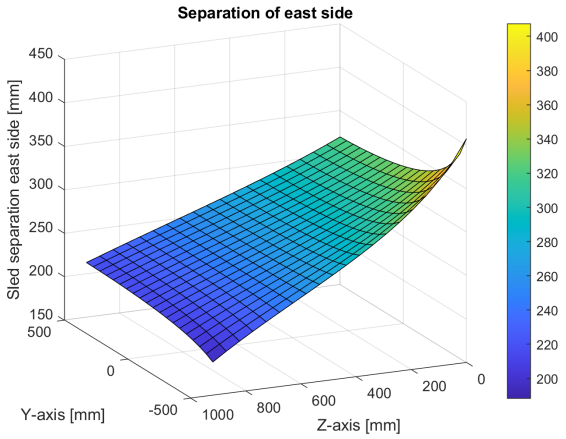

Figure D.29: Necessary sled separation at 50° head angle

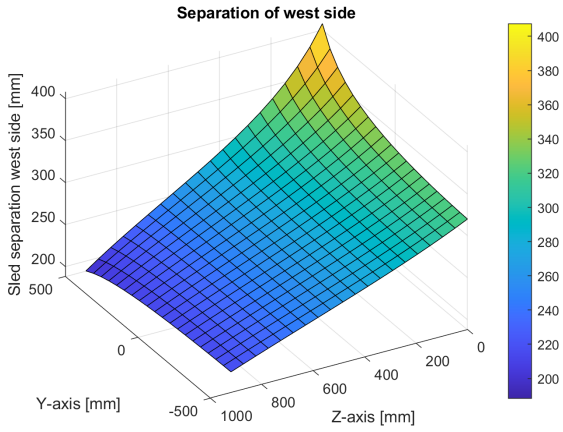

Figure D.30: Necessary sled separation at 50° head angle

D.6 60 Degrees

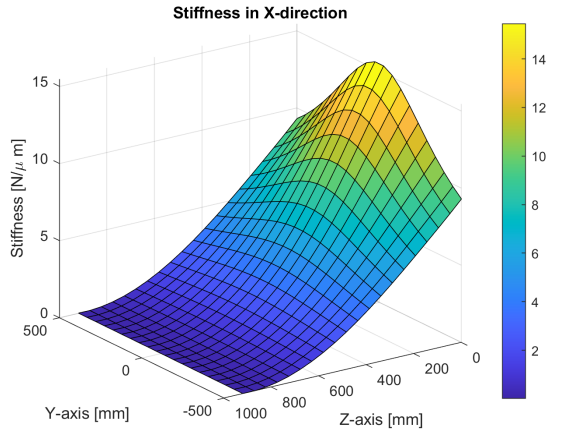

Figure D.31: Stiffness for double sled for load in X-direction at 60° head angle

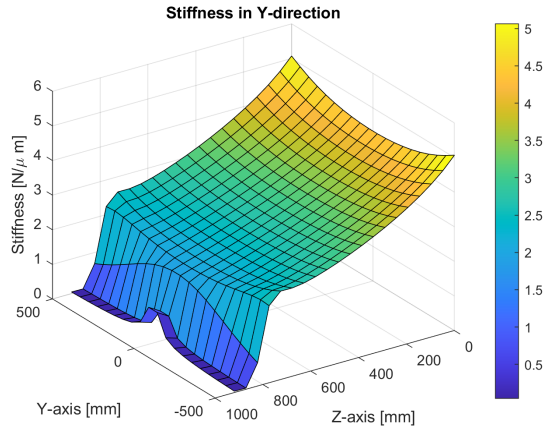

Figure D.32: Stiffness for double sled for load in Y-direction at 60° head angle

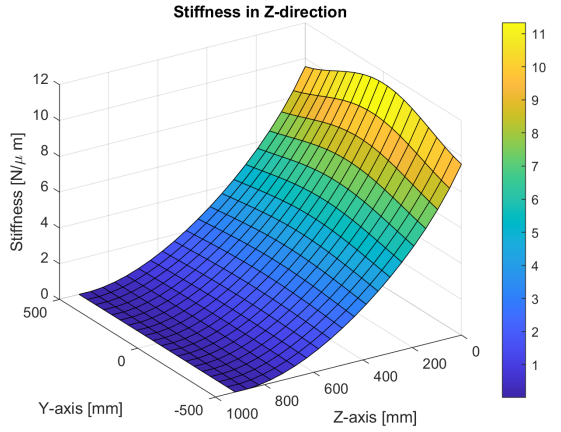

Figure D.33: Stiffness for double sled for load in Z-direction at 60° head angle

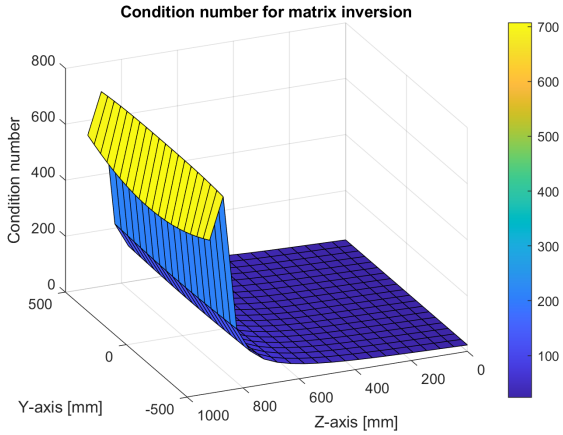

Figure D.34: Double sled condition number for matrix inversion at 60° head angle

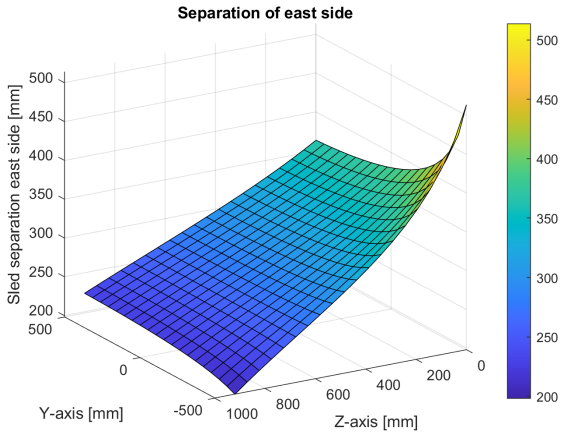

Figure D.35: Necessary sled separation at 60° head angle

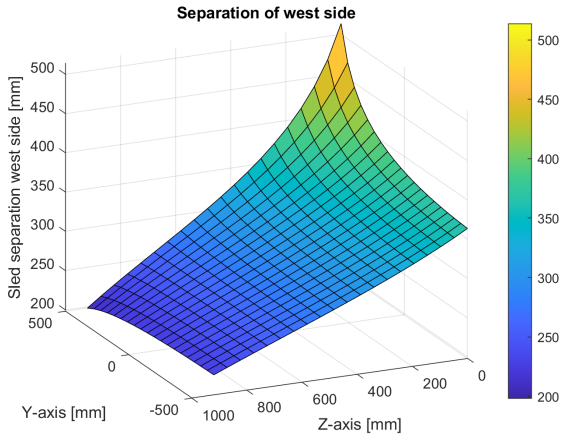

Figure D.36: Necessary sled separation at 60° head angle

D.7 70 Degrees

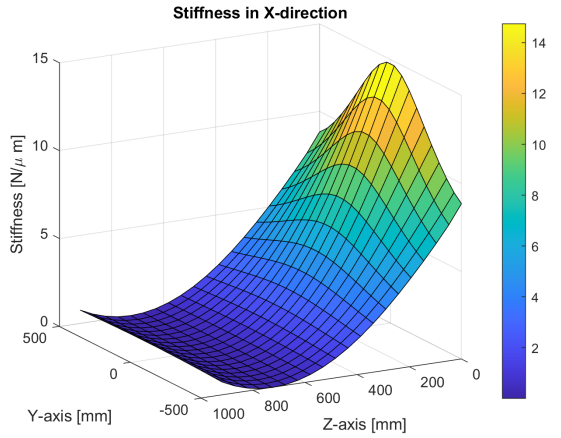

Figure D.37: Stiffness for double sled for load in X-direction at 70° head angle

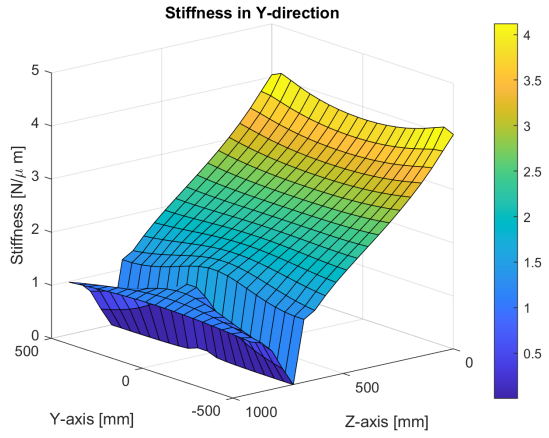

Figure D.38: Stiffness for double sled for load in Y-direction at 70° head angle

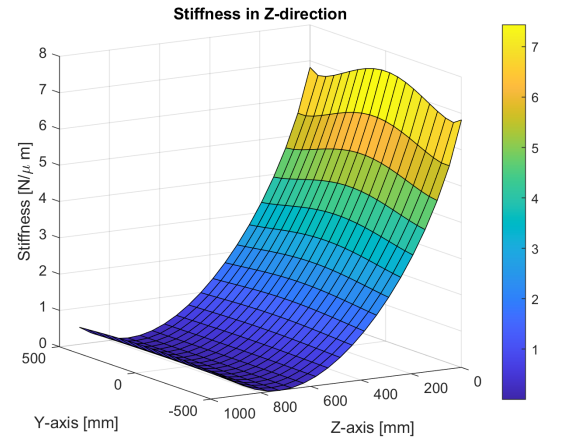

Figure D.39: Stiffness for double sled for load in Z-direction at 70° head angle

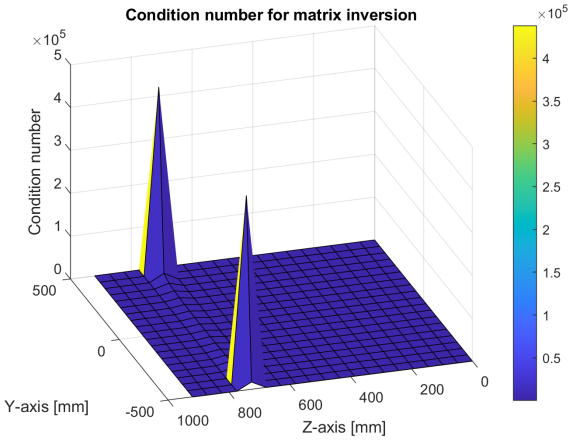

Figure D.40: Double sled condition number for matrix inversion at 70° head angle

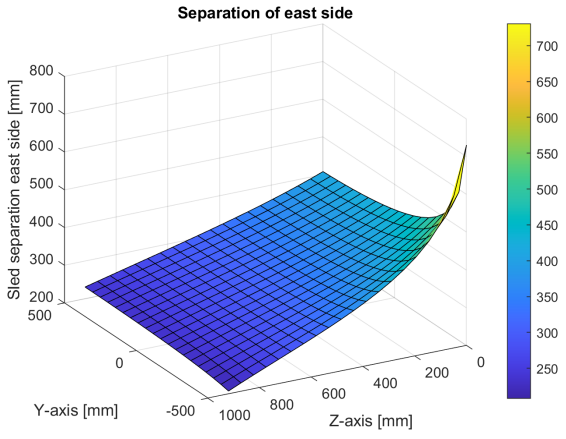

Figure D.41: Necessary sled separation at 70° head angle

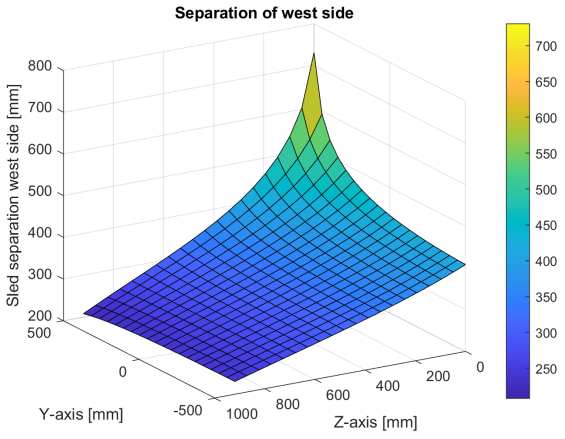

Figure D.42: Necessary sled separation at 70° head angle

D.8 80 Degrees

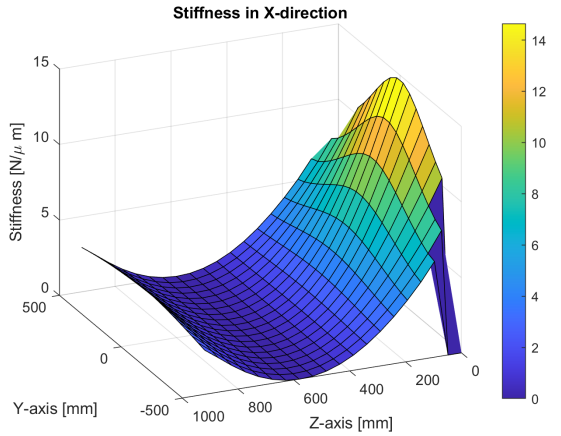

Figure D.43: Stiffness for double sled for load in X-direction at 80° head angle

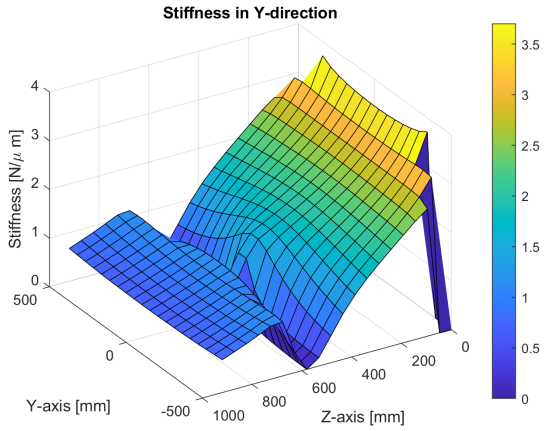

Figure D.44: Stiffness for double sled for load in Y-direction at 80° head angle

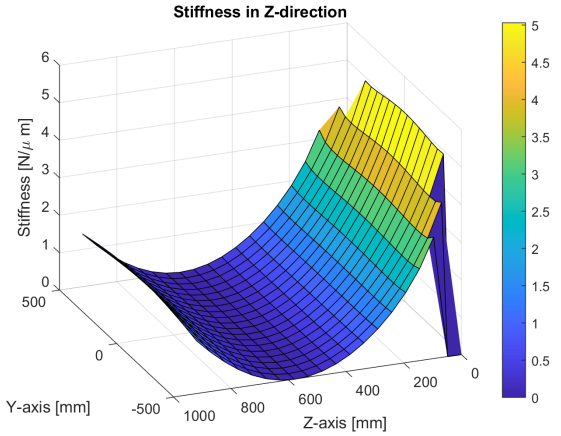

Figure D.45: Stiffness for double sled for load in Z-direction at 80° head angle

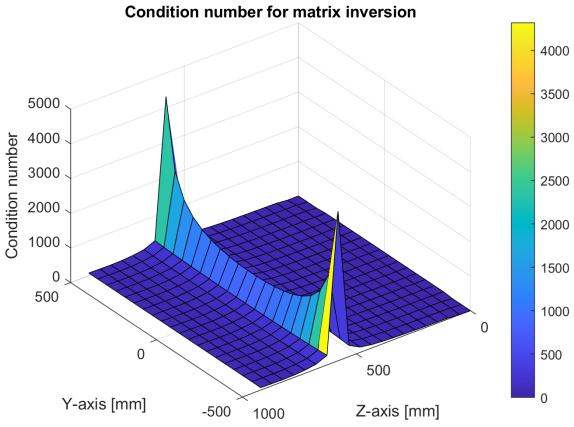

Figure D.46: Double sled condition number for matrix inversion at 80° head angle

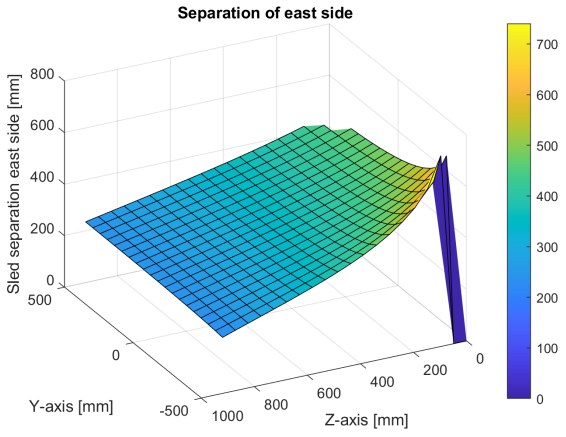

Figure D.47: Necessary sled separation at 80° head angle

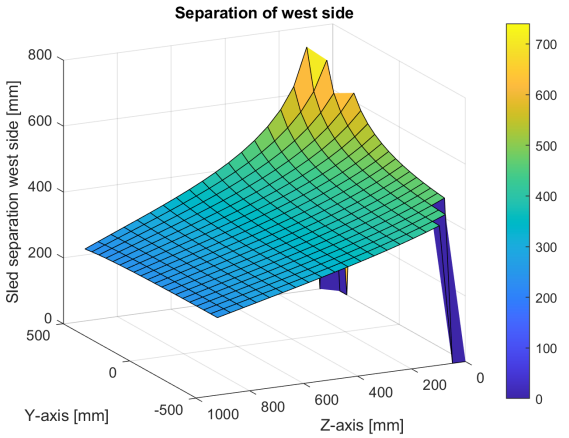

Figure D.48: Necessary sled separation at 80° head angle

D.9 90 Degrees

Figure D.49: Stiffness for double sled for load in X-direction at 90° head angle

Figure D.50: Stiffness for double sled for load in Y-direction at 90° head angle

Figure D.51: Stiffness for double sled for load in Z-direction at 90° head angle

Figure D.52: Double sled condition number for matrix inversion at 90° head angle

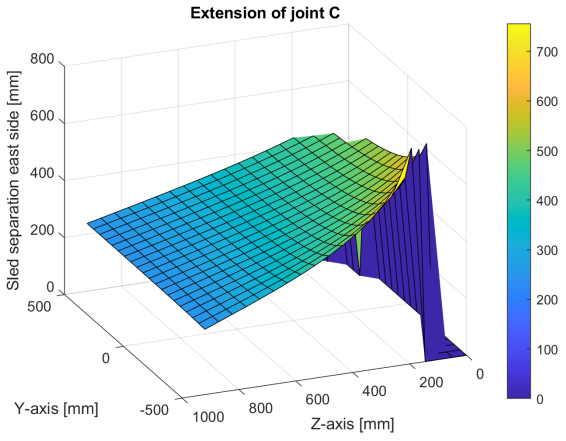

Figure D.53: Necessary sled separation at 90° head angle

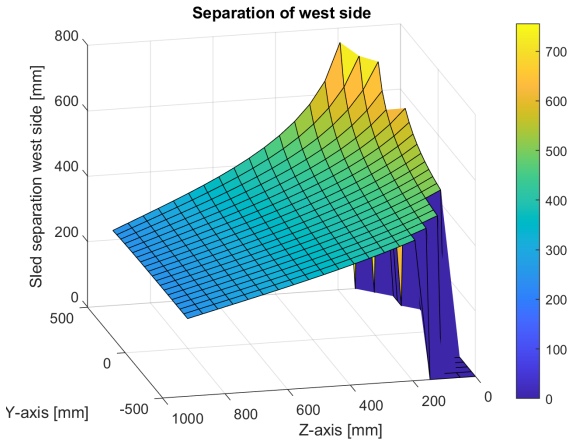

Figure D.54: Necessary sled separation at 90° head angle

Appendix E

Various code snippets used in this report in Matlab code

E.1 Stiffness calculation script for system as is

```
%Importing dimensions from external file
Constants = const();
A1m = Constants(1:3);
...

%Linkage geometry
L_long = 1.61; %[m]
L_short = 1.45; %[m]
k_long = 78.2; %[N/mu m]
k_short = 81.2; %[N/mu m]


%For loop parameters
n_i = 20;
n_j = 20;
Resultsx = zeros(n_i,n_j);
Resultsy = zeros(n_i,n_j);
Resultsz = zeros(n_i,n_j);
AxesY = linspace(-500,500,n_i);
AxesZ = linspace(0,920,n_j);
SingularityCheck = zeros(n_i,n_j);

for i=1:n_i
    z = AxesZ(i) * 1E-3;
    for j=1:n_j
        y = AxesY(j) * 1E-3;
        TCP = [0; y; z];

        %Calculations
        %Finding global position of ball socket joint on tool platform
        A1 = TCP + A1m;
        B1 = TCP + B1m;
        C1 = TCP + C1m;
        D1 = TCP + D1m;
        E1 = TCP + E1m;
        F1 = TCP + F1m;

        %Finding x-position of ball socet joint on carriage
        A2_x = -sqrt(L_long^2 - (A2_y-A1(2))^2 - (A2_z-A1(3))^2) + A1(1);
        B2_x = A2_x;
        C2_x = -sqrt(L_short^2 - (C2_y-C1(2))^2 - (C2_z-C1(3))^2) + C1(1);
```

```
        D2_x = C2_x + 0.085;
        E2_x = -sqrt(L_short^2 - (E2_y-E1(2))^2 - (E2_z-E1(3))^2) + E1(1);
        F2_x = E2_x - 0.085;

        %Resulting fixture position
        A2 = [A2_x; A2_y; A2_z];
        B2 = [B2_x; B2_y; B2_z];
        C2 = [C2_x; C2_y; C2_z];
        D2 = [D2_x; D2_y; D2_z];
        E2 = [E2_x; E2_y; E2_z];
        F2 = [F2_x; F2_y; F2_z];

        %Finding directional vectors of arms
        Ua = (A2-A1)/norm(A2-A1);
        Ub = (B2-B1)/norm(B2-B1);
        Uc = (C2-C1)/norm(C2-C1);
        Ud = (D2-D1)/norm(D2-D1);
        Ue = (E2-E1)/norm(E2-E1);
        Uf = (F2-F1)/norm(F2-F1);

        %Finding moment arm at ball socket joints from TCP
        rA = cross((A1-TCP),Ua);
        rB = cross((B1-TCP),Ub);
        rC = cross((C1-TCP),Uc);
        rD = cross((D1-TCP),Ud);
        rE = cross((E1-TCP),Ue);
        rF = cross((F1-TCP),Uf);

        %Static matrix H
        H = [Ua, Ub, Uc, Ud Ue, Uf; rA, rB, rC, rD, rE, rF];

        %Cartesian stiffness matrix
        k_L = [k_long, k_long, k_short, k_short, k_short, k_short];
        k_Cart = H*diag(k_L)*H';

        %Deflection of rods due to moment pr. force at TCP
        delta_x = k_Cart \ [1;0;0;0;0;0];
        delta_y = k_Cart \ [0;1;0;0;0;0];
        delta_z = k_Cart \ [0;0;1;0;0;0];

        %Stiffness in global cartesian direction
        k_x = 1/norm(delta_x(1:3));
        k_y = 1/norm(delta_y(1:3));
        k_z = 1/norm(delta_z(1:3));

        %Plotting the results
        Resultsx(i,j) = k_x;
        Resultsy(i,j) = k_y;
        Resultsz(i,j) = k_z;
        SingularityCheck(i,j) = cond(H);
    end
end
```

E.2 Various functions used in the scripts

```
function y = A_rot(px,py,pz)
    %Rotation matrix. Bryant angles chosen is [x-y-z]
    Ax = [1, 0, 0;
          0, cosd(px), -sind(px);
          0, sind(px), cosd(px)];
    Ay = [cosd(py), 0, sind(py);
          0, 1, 0;
          -sind(py), 0, cosd(py)];
    Az = [cosd(pz), -sind(pz), 0;
          sind(pz), cosd(pz), 0;
          0, 0, 1];

    y = Ax * Ay * Az;

end

function y = D_lineline(point1,vector1,point2,vector2)
    %This function computes a distance value between two non paralell lines
    cr = cross(vector1,vector2);
    pointpoint = point2 - point1;
    y = abs(dot(cr/norm(cr),pointpoint));
end

function y = D_linepoint(point,pointline,vectorline)
    %This function computes a distance value between a point and a line
    pointpoint = pointline - point;
    y = norm(cross(pointpoint,vectorline) / norm(vectorline));
end

function y = SolutionCheck(Actual, Check, Range)
    %This function wil find if the solution the NR solver has found is
    %within a range. It will return boolean 1 if it is and boolean 0 if not
    if Actual < Check - Range | Actual > Check + Range
        y = boolean(0);
    else
        y = boolean(1);
    end
end
```

E.3 Script for a Newton-Raphson solver calculating new global positions for rotation around XZ-axis for head given a X-rotation of head

```
%Initial guess
pz = -px;

%NR-solver finding z rotation of head and x position of top gantry
while error > 1E-5 && count < 1000 && breaksignal == boolean(0)
    A = A_rot(px,0,pz);
    A1 = TCP + A*A1m;
    B1 = TCP + A*B1m;

    dA1xdpz = -A1m(1)*sind(pz) - A1m(2)*cosd(pz);
    dA1ydpz = A1m(1)*cosd(px)*cosd(pz) - A1m(2)*cosd(px)*sind(pz);
    dA1zdpz = A1m(1)*sind(px)*cosd(pz) - A1m(2)*sind(px)*sind(pz);

    dB1xdpz = -B1m(1)*sind(pz) - B1m(2)*cosd(pz);
    dB1ydpz = B1m(1)*cosd(px)*cosd(pz) - B1m(2)*cosd(px)*sind(pz);
    dB1zdpz = B1m(1)*sind(px)*cosd(pz) - B1m(2)*sind(px)*sind(pz);

    f1 = A1(1) - sqrt(L_long^2 - (A2_y-A1(2))^2 - (A2_z-A1(3))^2);
    f2 = B1(1) - sqrt(L_long^2 - (B2_y-B1(2))^2 - (B2_z-B1(3))^2);
    f = f1 - f2;

    df1 = dA1ydpz * (A2_y-A1(2)) + dA1zdpz * (A2_z-A1(3));
    df2 = sqrt(L_long^2 - (A2_y-A1(2))^2 - (A2_z-A1(3))^2);
    df3 = dB1ydpz * (B2_y-B1(2)) + dB1zdpz * (B2_z-B1(3));
    df4 = sqrt(L_long^2 - (B2_y-B1(2))^2 - (B2_z-B1(3))^2);
    df = dA1xdpz - df1/df2 - dB1xdpz + df3/df4;

    pz = pz - f / df;

    f1_new = A1(1) - sqrt(L_long^2 - (A2_y-A1(2))^2 - (A2_z-A1(3))^2);
    f2_new = B1(1) - sqrt(L_long^2 - (B2_y-B1(2))^2 - (B2_z-B1(3))^2);
    f_new = f1_new - f2_new;

    count = count + 1;
    error = norm(f_new);

    if count > 50 && error > 0.2
        breaksignal = boolean(1);
    end
end

%Calculations
%Finding global position of ball socket joint on tool platform
A = A_rot(px,0,pz);
A1 = TCP + A*A1m;
B1 = TCP + A*B1m;
C1 = TCP + A*C1m;
D1 = TCP + A*D1m;
E1 = TCP + A*E1m;
F1 = TCP + A*F1m;
```

E.4 Newton-Raphson solver for finding one of the joint positions of the rotational sled

```
errorE = 1; CountE = 0; RE = 0;
while errorE > 1E-5 && CountE < 1E4
    Ay = A_rot(0,RE,0);
    S_C2 = Ay*S_C2m;
    S_D2 = Ay*S_D2m;

    f1 = C1(1) - S_C2(1);
    f2 = sqrt(L_short^2 - (CoR_Ry+S_C2(2)-C1(2))^2 - ...
        (CoR_Rz+S_C2(3)-C1(3))^2);
    f3 = D1(1) - S_D2(1);
    f4 = sqrt(L_short^2 - (CoR_Ry+S_D2(2)-D1(2))^2 - ...
        (CoR_Rz+S_D2(3)-D1(3))^2);
    f = f1 - f2 - f3 + f4;

    dSC2xdR = -sind(RE) * S_C2m(1) + cosd(RE) * S_C2m(3);
    dSC2zdR = -cosd(RE) * S_C2m(1) - sind(RE) * S_C2m(3);
    dSD2xdR = -sind(RE) * S_D2m(1) + cosd(RE) * S_D2m(3);
    dSD2zdR = -cosd(RE) * S_D2m(1) - sind(RE) * S_D2m(3);

    df1 = dSC2zdR * (CoR_Rz + S_C2(2)-C1(2));
    df2 = dSD2zdR * (CoR_Rz + S_D2(2)-D1(2));
    df = -dSC2xdR - 0.5 * df1 / f2 + dSD2xdR + 0.5 * df2 / f4;

    RE = RE - f/df;

    %Ensuring joint angle is within one rotation
    if RE < 0 | RE > 360
        RE = wrapTo360(real(RE));
    else
        RE = real(RE);
    end

    %Ensuring only one possible solution
    if RE < 180 && RE > 90
        RE = RE + 180;
    elseif RE < 270 && RE > 180
        RE = RE - 180;
    end

    Ay = A_rot(0,RE,0);
    S_C2 = Ay*S_C2m;
    S_D2 = Ay*S_D2m;

    f1_new = C1(1) - S_C2(1);
    f2_new = sqrt(L_short^2 - (CoR_Ry+S_C2(2)-C1(2))^2 - ...
        (CoR_Rz+S_C2(3)-C1(3))^2);
    f3_new = D1(1) - S_D2(1);
    f4_new = sqrt(L_short^2 - (CoR_Ry+S_D2(2)-D1(2))^2 - ...
        (CoR_Rz+S_D2(3)-D1(3))^2);
    f_new = f1_new - f2_new - f3_new + f4_new;

    errorE = abs(f_new);
    CountE = CountE + 1;
end

%Finding vectors from CoR to joints in global coordinate system
S_C2 = A_rot(0,RE,0)*S_C2m;
```

```
S_D2 = A_rot(0,RE,0)*S_D2m;
S_E2 = A_rot(0,RW,0)*S_E2m;
S_F2 = A_rot(0,RW,0)*S_F2m;

%Finding x-position of rotational point on carriage
A2_x = -sqrt(L_long^2 - (A2_y-A1(2))^2 - (A2_z-A1(3))^2) + A1(1);
B2_x = A2_x;
CoR_Rx = -sqrt(L_short^2 - (CoR_Ry+S_D2(2)-D1(2))^2 - ...
    (CoR_Rz+S_D2(3)-D1(3))^2) - S_D2(1) + D1(1);
CoR_Lx = -sqrt(L_short^2 - (CoR_Ly+S_E2(2)-E1(2))^2 - ...
    (CoR_Lz+S_E2(3)-E1(3))^2) - S_E2(1) + E1(1);

%Resulting ficture positions
CoR_R = [CoR_Rx; CoR_Ry; CoR_Rz];
CoR_L = [CoR_Lx; CoR_Ly; CoR_Lz];
A2 = [A2_x; A2_y; A2_z];
B2 = [B2_x; B2_y; B2_z];
C2 = CoR_R + S_C2;
D2 = CoR_R + S_D2;
E2 = CoR_L + S_E2;
F2 = CoR_L + S_F2;

%Test for length of linkages
L_C = norm(C2-C1);
L_D = norm(D2-D1);
L_E = norm(E2-E1);
L_F = norm(F2-F1);
Test_C = SolutionCheck(L_C, L_short, 0.001);
Test_D = SolutionCheck(L_D, L_short, 0.001);
Test_E = SolutionCheck(L_E, L_short, 0.001);
Test_F = SolutionCheck(L_F, L_short, 0.001);


%Plotting if NR solver is within error
if Test_C && Test_D && Test_D && Test_E && Test_F
    Resultsx_neg(i,j) = px;
    JointCoordx_negE(i,j) = RE;
    JointCoordx_negW(i,j) = RW;
end
```

E.5 Script algebraically calculating both joint positions of the delta light concept

```
%Finding global position of ball socket joint on tool platform
A1 = TCP + A*A1m;B1 = TCP + A*B1m;
C1 = TCP + A*C1m;
D1 = TCP + A*D1m;
E1 = TCP + A*E1m;
F1 = TCP + A*F1m;

%Finding x-position of ball socet joint on carriage
A2_x = -sqrt(L_long^2 - (A2_y-A1(2))^2 - (A2_z-A1(3))^2) + A1(1);
B2_x = A2_x;
C2_x = -sqrt(L_short^2 - (C2_y-C1(2))^2 - (C2_z-C1(3))^2) + C1(1);
D2_xm = C2_x - 0.085;
F2_x = -sqrt(L_short^2 - (F2_y-F1(2))^2 - (F2_z-F1(3))^2) + F1(1);
E2_xm = F2_x - 0.085;

%Calculating Delta light joint angles
LD_xz = sqrt(L_short^2 - (D2_y-D1(2))^2);
LE_xz = sqrt(L_short^2 - (E2_y-E1(2))^2);
D_1Dm = [(D2_xm-D1(1));(D2_zm-D1(3))];
D_1Em = [(E2_xm-E1(1));(E2_zm-E1(3))];
d_1Dm = norm(D_1Dm);
d_1Em = norm(D_1Em);
betaD = asind(D_1Dm(2)/d_1Dm);
betaE = asind(D_1Em(2)/d_1Em);
Angle_D = acosd((r^2 + d_1Dm^2 - LD_xz^2)/(2*r*d_1Dm));
Angle_E = acosd((r^2 + d_1Em^2 - LE_xz^2)/(2*r*d_1Em));
alphaD = 360 - betaD - abs(Angle_D);
alphaE = 360 - betaE - abs(Angle_E);

%Joint positions of delta light
D2_x = D2_xm + r * cosd(alphaD);
D2_z = D2_zm + r * sind(alphaD);
E2_x = E2_xm + r * cosd(alphaE);
E2_z = E2_zm + r * sind(alphaE);
JointCoordinate_E = alphaD;
JointCoordinate_W = alphaE;
```

E.6 An example of a compare script for double sled concept

```
%Test conditions
n_i = 20; %number of discretizations for z-direction
n_j = 20; %number of discretizations for y-direction
limit = 0.3; %Limit for lowered stiffness

%Importing dimensions from external file
Constants = const();
A1m = Constants(1:3);
...

%Linkage geometry
L_long = 1.61; %[m]
L_short = 1.45; %[m]
k_long = 78.2; %[N/mu m]
k_short = 81.2; %[N/mu m]

%Matrix creation
Stiffnessx = zeros(n_i,n_j);
Stiffnessy = zeros(n_i,n_j);
Stiffnessz = zeros(n_i,n_j);
resultsx = zeros(n_i,n_j);
resultsy = zeros(n_i,n_j);
resultsz = zeros(n_i,n_j);
AxesY = linspace(-500,500,n_i);
AxesZ = linspace(0,920,n_j);

%Calculating stiffness of system as is
for i=1:n_i
    z = AxesZ(i) * 1E-3;
    for j=1:n_j
        y = AxesY(j) * 1E-3;
        TCP = [0; y; z];

        %Calculations
        %Finding global position of ball socket joint on tool platform
        A1 = TCP + A1m;
        B1 = TCP + B1m;
        C1 = TCP + C1m;
        D1 = TCP + D1m;
        E1 = TCP + E1m;
        F1 = TCP + F1m;

        %Finding x-position of ball socet joint on carriage
        A2_x = -sqrt(L_long^2 - (A2_y-A1(2))^2 - (A2_z-A1(3))^2) + A1(1);
        B2_x = A2_x;
        C2_x = -sqrt(L_short^2 - (C2_y-C1(2))^2 - (C2_z-C1(3))^2) + C1(1);
        D2_x = C2_x + 0.085;
        E2_x = -sqrt(L_short^2 - (E2_y-E1(2))^2 - (E2_z-E1(3))^2) + E1(1);
        F2_x = E2_x - 0.085;

        %Resulting fixture position
        A2 = [A2_x; A2_y; A2_z];
        B2 = [B2_x; B2_y; B2_z];
        C2 = [C2_x; C2_y; C2_z];
        D2 = [D2_x; D2_y; D2_z];
        E2 = [E2_x; E2_y; E2_z];
        F2 = [F2_x; F2_y; F2_z];
```

```
        %Finding directional vectors of arms
        Ua = (A2-A1)/norm(A2-A1);
        Ub = (B2-B1)/norm(B2-B1);
        Uc = (C2-C1)/norm(C2-C1);
        Ud = (D2-D1)/norm(D2-D1);
        Ue = (E2-E1)/norm(E2-E1);
        Uf = (F2-F1)/norm(F2-F1);

        %Finding moment arm at ball socket joints from TCP
        rA = cross((A1-TCP),Ua);
        rB = cross((B1-TCP),Ub);
        rC = cross((C1-TCP),Uc);
        rD = cross((D1-TCP),Ud);
        rE = cross((E1-TCP),Ue);
        rF = cross((F1-TCP),Uf);

        %Static matrix H
        H = [Ua, Ub, Uc, Ud Ue, Uf; rA, rB, rC, rD, rE, rF];

        %Cartesian stiffness matrix
        k_L = [k_long, k_long, k_short, k_short, k_short, k_short];
        k_Cart = H*diag(k_L)*H';

        %Deflection of rods due to moment pr. force at TCP
        delta_x = k_Cart \ [1;0;0;0;0;0];
        delta_y = k_Cart \ [0;1;0;0;0;0];
        delta_z = k_Cart \ [0;0;1;0;0;0];

        %Stiffness in global cartesian direction
        k_x = 1/norm(delta_x(1:3));
        k_y = 1/norm(delta_y(1:3));
        k_z = 1/norm(delta_z(1:3));

        %Saving the results
        Stiffnessx(i,j) = k_x;
        Stiffnessy(i,j) = k_y;
        Stiffnessz(i,j) = k_z;
    end
end

%Checking stiffness of new system against old values
for i=1:n_i
    z = AxesZ(i) * 1E-3;
    for j=1:n_j
        y = AxesY(j) * 1E-3;
        TCP = [0; y; z];
        py = 0;
        k_x = Stiffnessx(i,j);
        k_y = Stiffnessy(i,j);
        k_z = Stiffnessz(i,j);
        testx = Stiffnessx(i,j);
        testy = Stiffnessy(i,j);
        testz = Stiffnessz(i,j);
        while py<90 && (limit*testx < k_x | limit*testy < k_y | ...
            limit*testz < k_z)
            px=0;       %[deg]
            py=py + 0.2; %[deg]
            pz=0;       %[deg]
            A = A_rot(0,py,0);

            %Calculations
```

```
%Finding global position of ball socket joint on tool platform
A1 = TCP + A*A1m;
B1 = TCP + A*B1m;
C1 = TCP + A*C1m;
D1 = TCP + A*D1m;
E1 = TCP + A*E1m;
F1 = TCP + A*F1m;

%Finding x-position of ball socet joint on carriage
A2_x = -sqrt(L_long^2 - (A2_y-A1(2))^2 - (A2_z-A1(3))^2) + A1(1);
B2_x = A2_x;
C2_x = -sqrt(L_short^2 - (C2_y-C1(2))^2 - (C2_z-C1(3))^2) + C1(1);
D2_x = -sqrt(L_short^2 - (D2_y-D1(2))^2 - (D2_z-D1(3))^2) + D1(1);
E2_x = -sqrt(L_short^2 - (E2_y-E1(2))^2 - (E2_z-E1(3))^2) + E1(1);
F2_x = -sqrt(L_short^2 - (F2_y-F1(2))^2 - (F2_z-F1(3))^2) + F1(1);

%Resulting fixture position
A2 = [A2_x; A2_y; A2_z];
B2 = [B2_x; B2_y; B2_z];
C2 = [C2_x; C2_y; C2_z];
D2 = [D2_x; D2_y; D2_z];
E2 = [E2_x; E2_y; E2_z];
F2 = [F2_x; F2_y; F2_z];

%Finding directional vectors of arms
Ua = (A2-A1)/norm(A2-A1);
Ub = (B2-B1)/norm(B2-B1);
Uc = (C2-C1)/norm(C2-C1);
Ud = (D2-D1)/norm(D2-D1);
Ue = (E2-E1)/norm(E2-E1);
Uf = (F2-F1)/norm(F2-F1);

%Finding moment arm at ball socket joints from TCP
rA = cross((A1-TCP),Ua);
rB = cross((B1-TCP),Ub);
rC = cross((C1-TCP),Uc);
rD = cross((D1-TCP),Ud);
rE = cross((E1-TCP),Ue);
rF = cross((F1-TCP),Uf);

%Static matrix H
H = [Ua, Ub, Uc, Ud Ue, Uf; rA, rB, rC, rD, rE, rF];

%Cartesian stiffness matrix
k_L = [k_long, k_long, k_short, k_short, k_short, k_short];
k_Cart = H*diag(k_L)*H';

%Deflection of rods due to moment pr. force at TCP
delta_x = k_Cart \ [1;0;0;0;0;0];
delta_y = k_Cart \ [0;1;0;0;0;0];
delta_z = k_Cart \ [0;0;1;0;0;0];

%Stiffness in global cartesian direction
k_x = 1/norm(delta_x(1:3));
k_y = 1/norm(delta_y(1:3));
k_z = 1/norm(delta_z(1:3));

%Saving relevant results
if limit*testx < k_x && isreal(F2(1)+C2(1)+D2(1)+E2(1))
    resultsx(i,j) = py;
end
```

```
            if limit*testy < k_y && isreal(F2(1)+C2(1)+D2(1)+E2(1))
                resultsy(i,j) = py;
            end

            if limit*testz < k_z && isreal(F2(1)+C2(1)+D2(1)+E2(1))
                resultsz(i,j) = py;
            end
        end
    end
end
```

Bibliography

[1] June 2021. URL: https://www.youtube.com/watch?v=h43cqrE246A.

[2] en. Feb. 2022. URL: https://metrom.com/5-axis-machining-center/.

[3] URL: https://www.photonics.com/images/Web/Articles/2018/3/19/Robotic_Torque2.jpg.

[4] URL: https://acrome.net/post/the-basics-of-the-stewart-platform.

[5] URL: https://www.starrag.com/en-us/machine/ecospeed-f-2035/132.

[6] en-US. URL: https://www.starrag.com/en-us/machine/ecospeed-f-2035/132.

[7] URL: https://cognibotics.com/wp-content/uploads/2023/01/2022-09-01-Multirobot-from-ELHA-1-copy.webp.

[8] URL: https://pagesperso.ls2n.fr/~briot-s/ANR_ARROW.html.

[9] URL: https://ms.copernicus.org/articles/10/589/2019/ms-10-589-2019.html.

[10] URL: https://qph.cf2.quoracdn.net/main-qimg-b3ff8a5f14c4a4e7525533518dbcd7f0.

[11] Stéphane Caro et al. "Sensitivity Analysis of the Orthoglide, a 3-DOF Translational Parallel Kinematic Machine." In: *Computing Research Repository - CORR* (Jan. 2007). DOI: 10.1115/DETC2004-57379.

[12] JoseLuis Olazagoitia and Scott Wyatt. "New PKM Tricept T9000 and Its Application to Flexible Manufacturing at Aerospace Industry." In: *SAE Technical Papers* 2142 (Sept. 2007), pp. 37–48. DOI: 10.4271/2007-01-3820.

[13] Andreas Pott, Tim Boye, and Manfred Hiller. "Design and Optimization of Parallel Kinematic Machines under Process Requirements." In: (Jan. 2006).

[14] Raza Ur-Rehman et al. "Kinematic and Dynamic Analyses of the Orthoglide 5-axis." In: (Oct. 2008).

[15] ABB Robotics. *Product Specification IRB 390*. URL: https://search.abb.com/library/Download.aspx?DocumentID=3HAC066568-001&LanguageCode=en&DocumentPartId=&Action=Launch. (accessed: 01.02.2023).

[16] PKM Tricept. *Tricept Product Range*. URL: http://www.pkmtricept.com/files/catalogo_en.pdf. (accessed: 08.01.2023).

[17] Ilya Tyapin and Geir Hovland. "The Gantry-Tau parallel kinematic machine—kinematic and elastodynamic design optimisation." en. In: *Meccanica* 46.1 (Feb. 2011), pp. 113–129. ISSN: 0025-6455, 1572-9648. DOI: 10.1007/s11012-010-9394-9.