Robotics, Vision and Control

Master's degree in Computer Engineering for Robotics and smart Industry

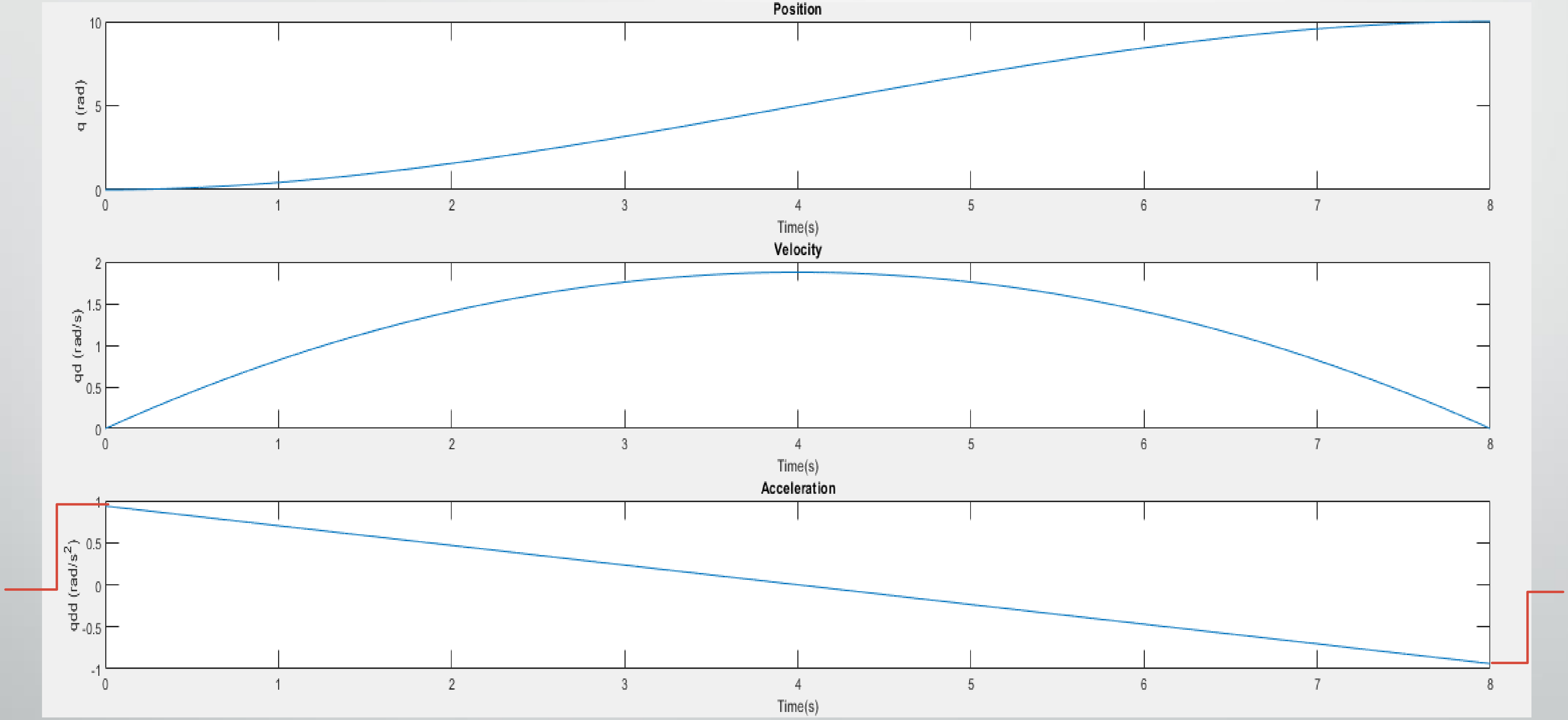
Project – Exam

Edoardo Fiorini

General overview

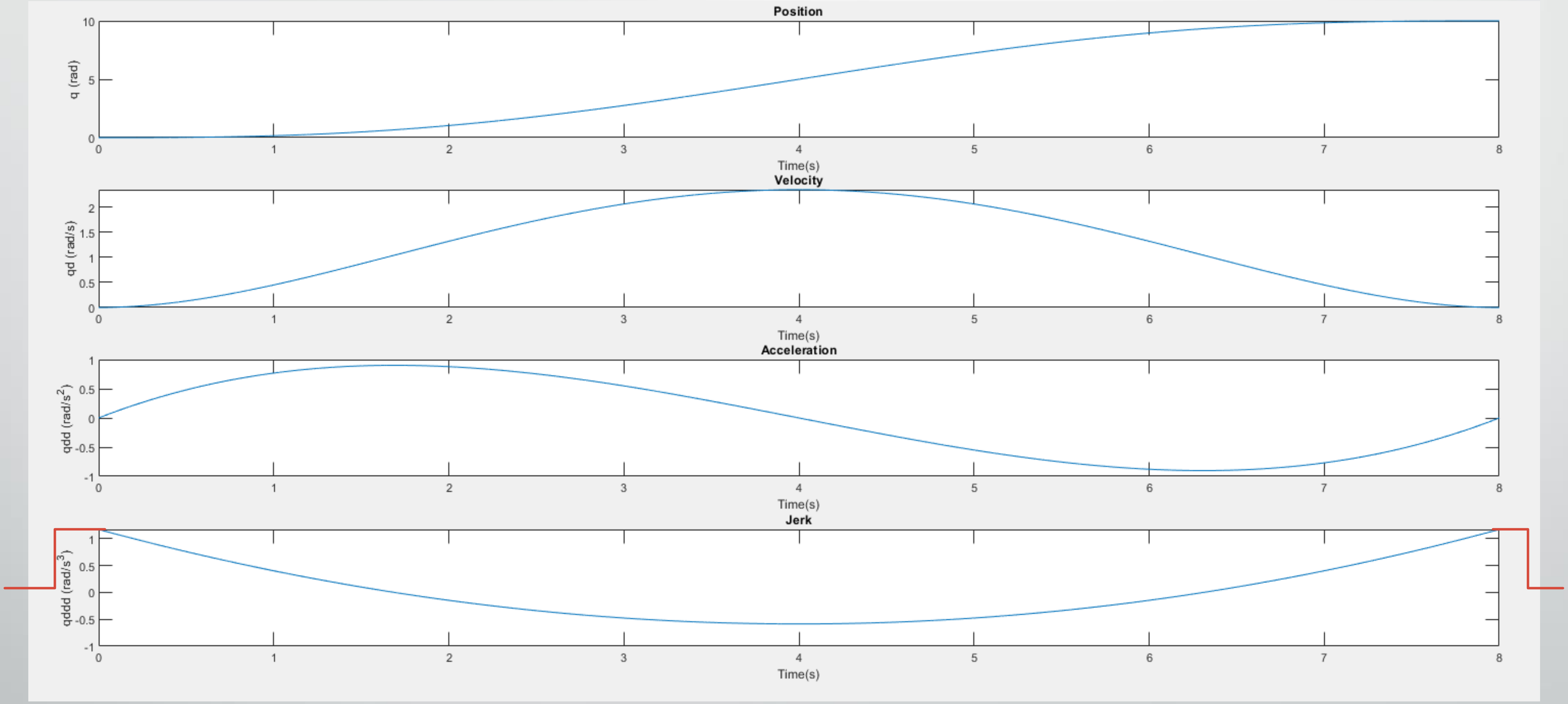
1. Robotics
    a. Point-to-point and multi points trajectory in configurational space
    b. Operational space trajectory
    c. Model-based trajectory
2. Vision
    a. 3D acquisition system
    b. Image analysis
    c. 3D analysis
    d. Hand eye calibration

3th-order polynomial



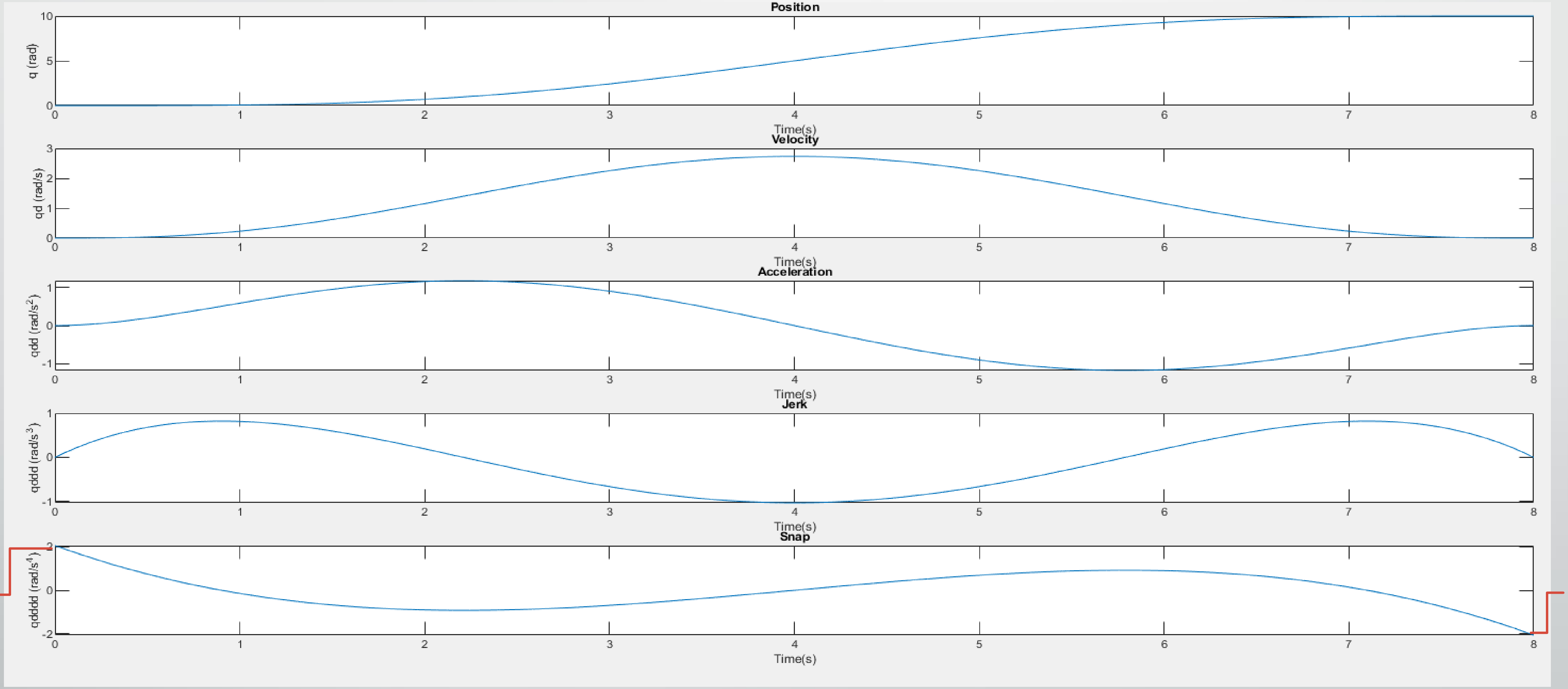
[q, qd, qdd, T] = cubic_polynomials(qi, qf, dqi, dqf, ti, tf, Ts);

5th-order polynomial



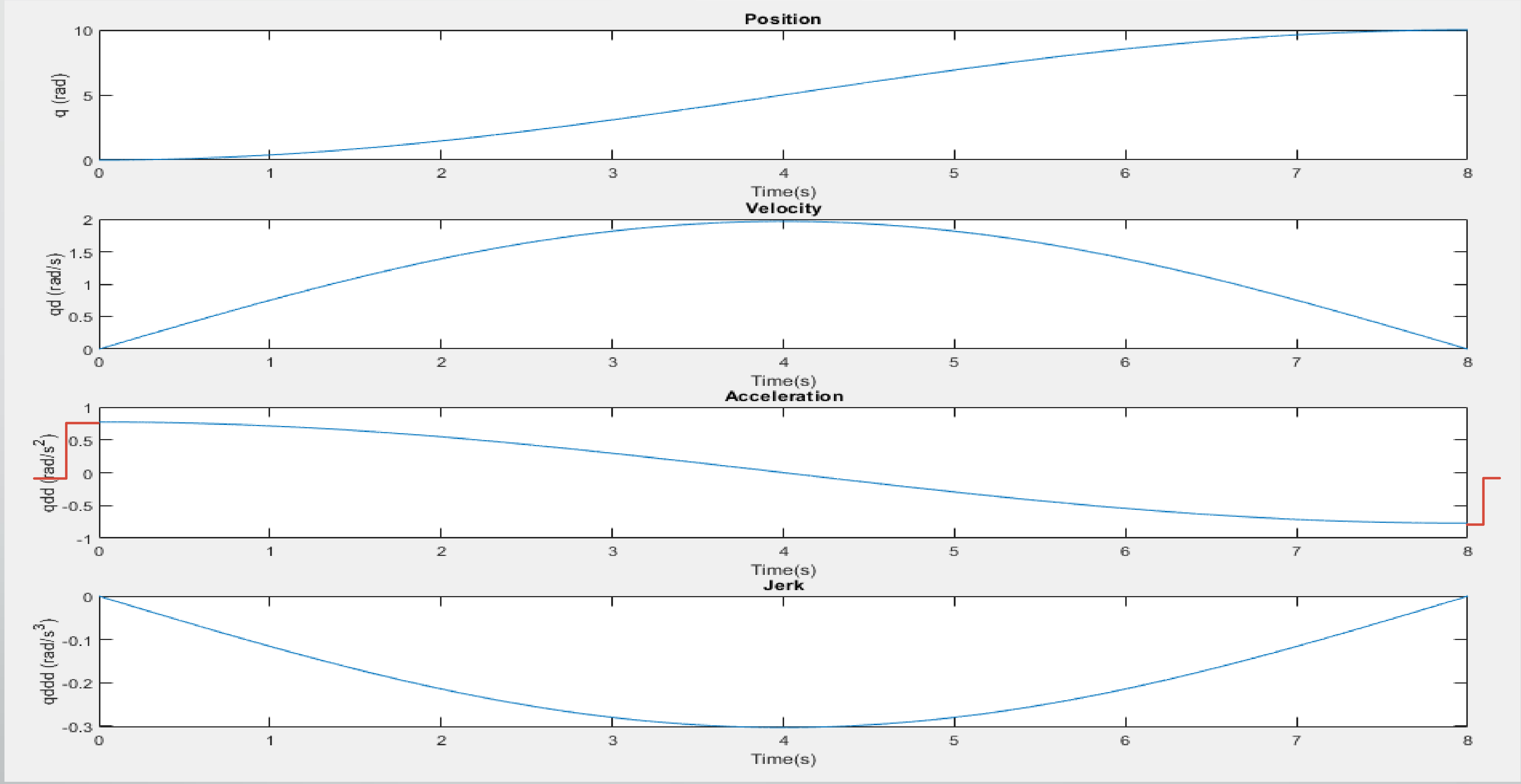
[q, qd, qdd, qddd, T] = fifth_polynomials(qi, qf, dqi, dqf, ddqi, ddqf, ti, tf, Ts);

7th-order polynomial



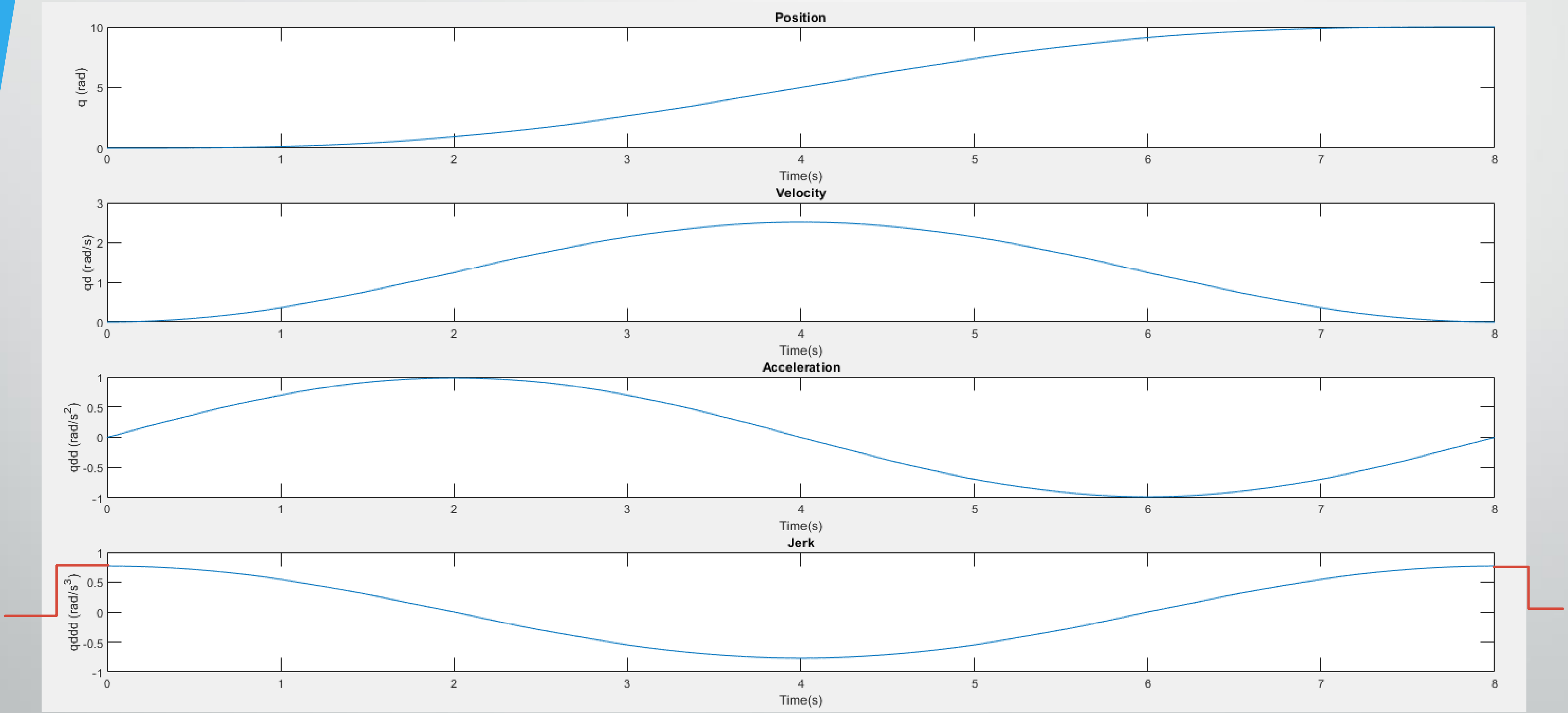
[q, qd, qdd, qddd, qdddd, T] = seventh_polynomials(qi, qf, dqi, dqf, ddqi, ddqf, dddqi, dddqf, ti, tf, Ts);

Harmonic trajectory



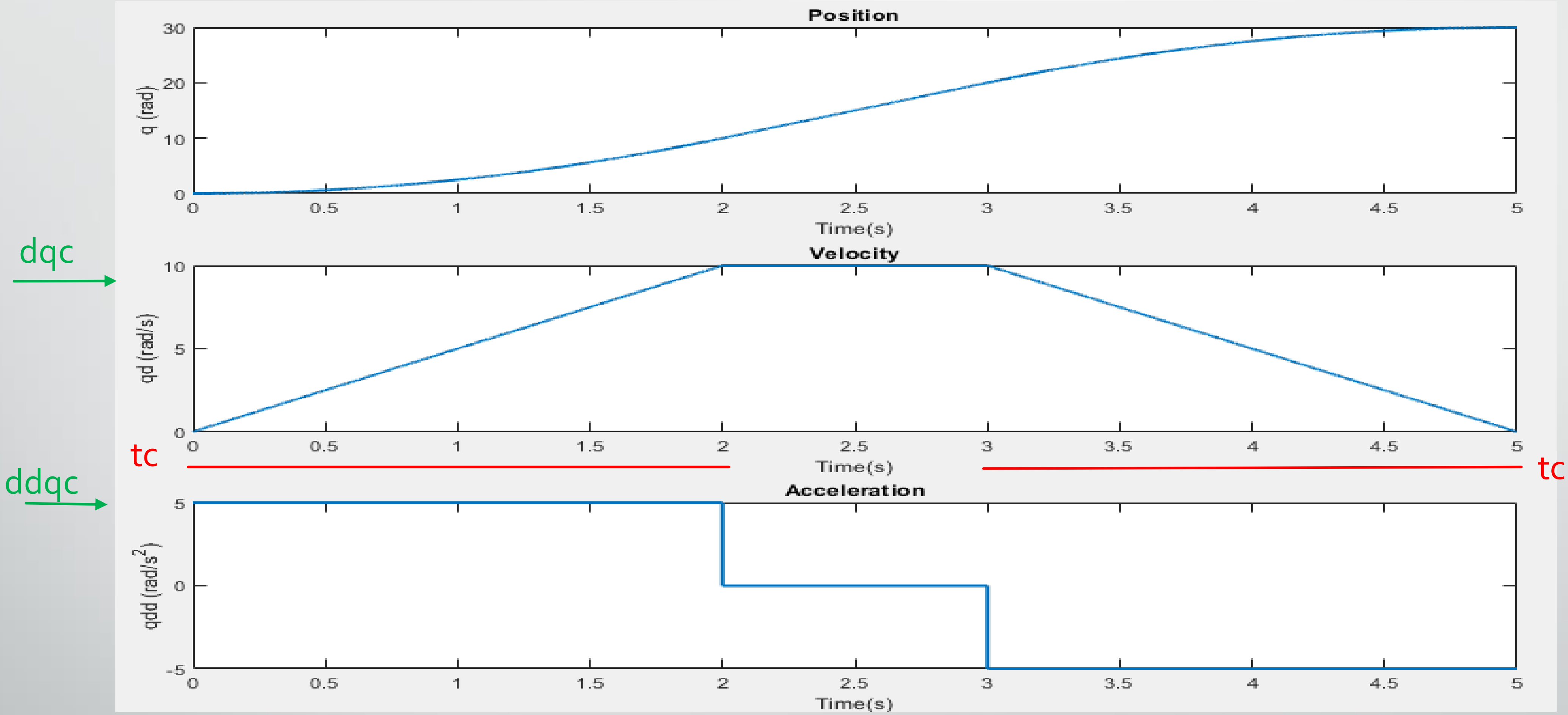
[q, qd, qdd, qddd, T] = harmonicTrajectory(qi, qf, ti, tf, Ts);

Cycloidal trajectory



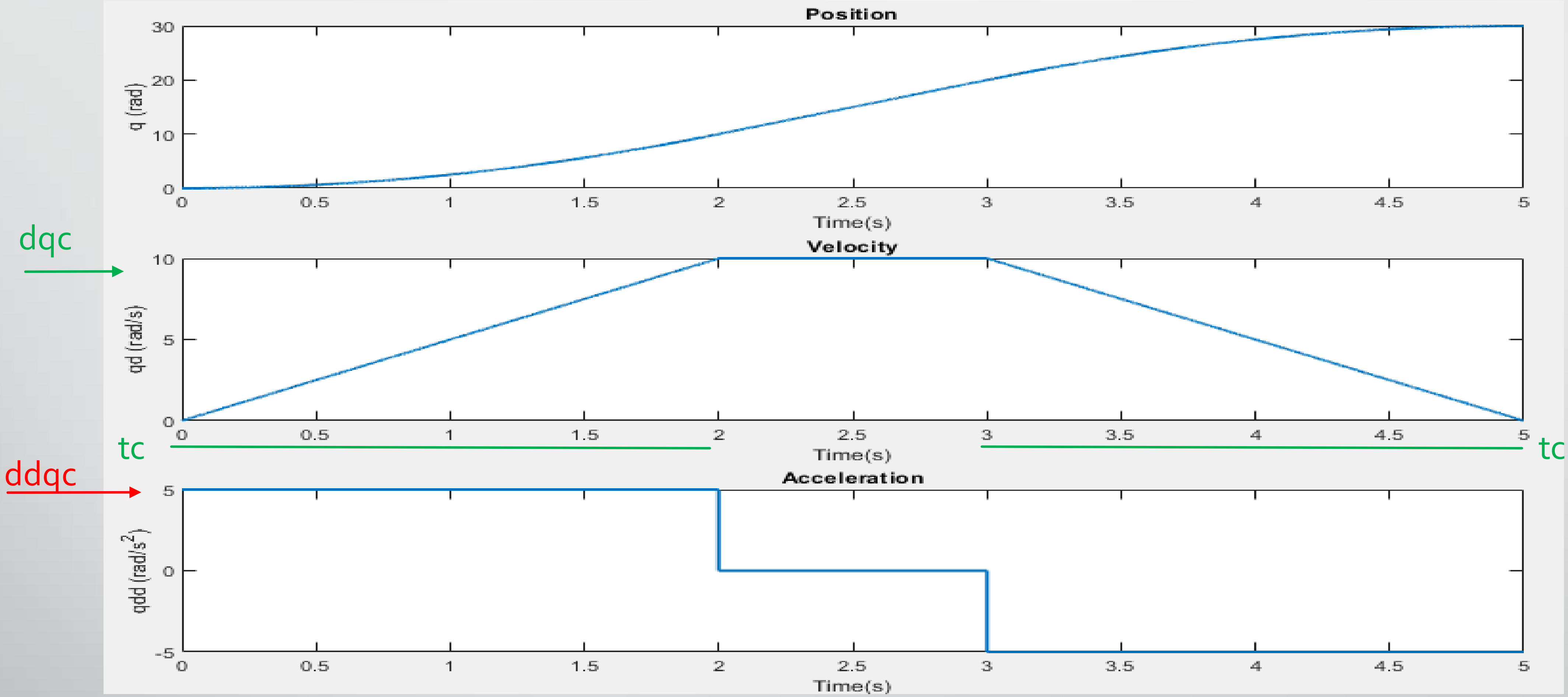
[q, qd, qdd, qddd, T] = cycloidalTrajectory(qi, qf, ti, tf, Ts);

Symmetric trapezoidal trajectory – subcase 1



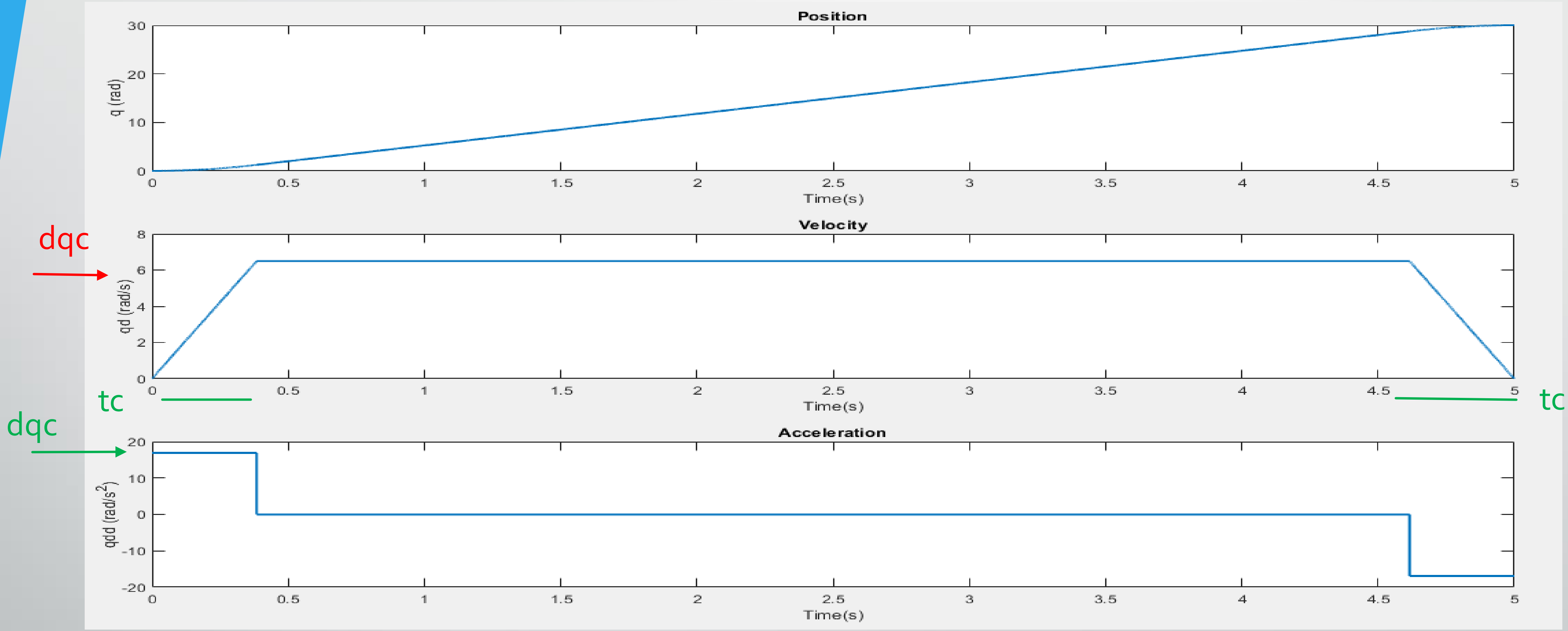
[q, qd, qdd, T]= trapezoidalProfileSymmetric(tc, dqc, ddqc, ti, tf, qf, qi, Ts, subcase);

Symmetric trapezoidal trajectory – subcase 2



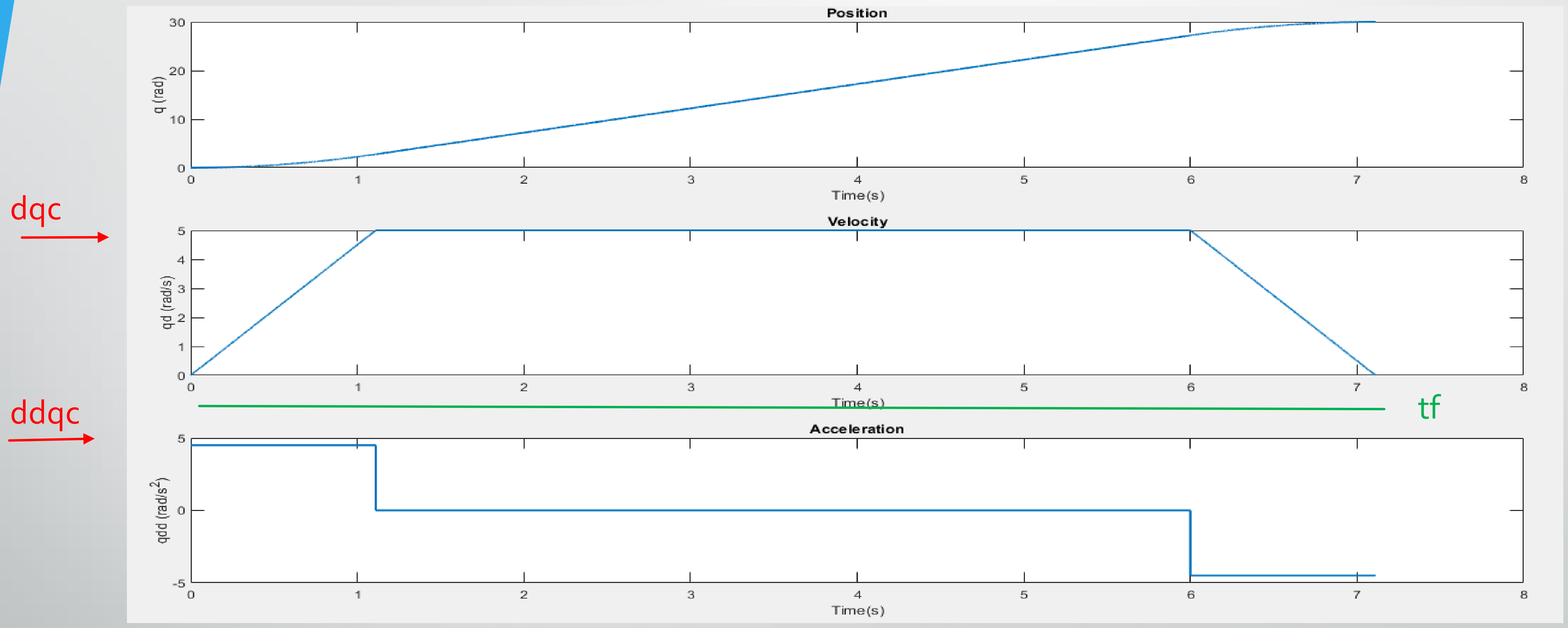
[q, qd, qdd, T]= trapezoidalProfileSymmetric(tc, dqc, ddqc, ti, tf, qf, qi, Ts, subcase);

Symmetric trapezoidal trajectory – subcase 3



[q, qd, qdd, T]= trapezoidalProfileSymmetric(tc, dqc, ddqc, ti, tf, qf, qi, Ts, subcase);

Symmetric trapezoidal trajectory – subcase 4

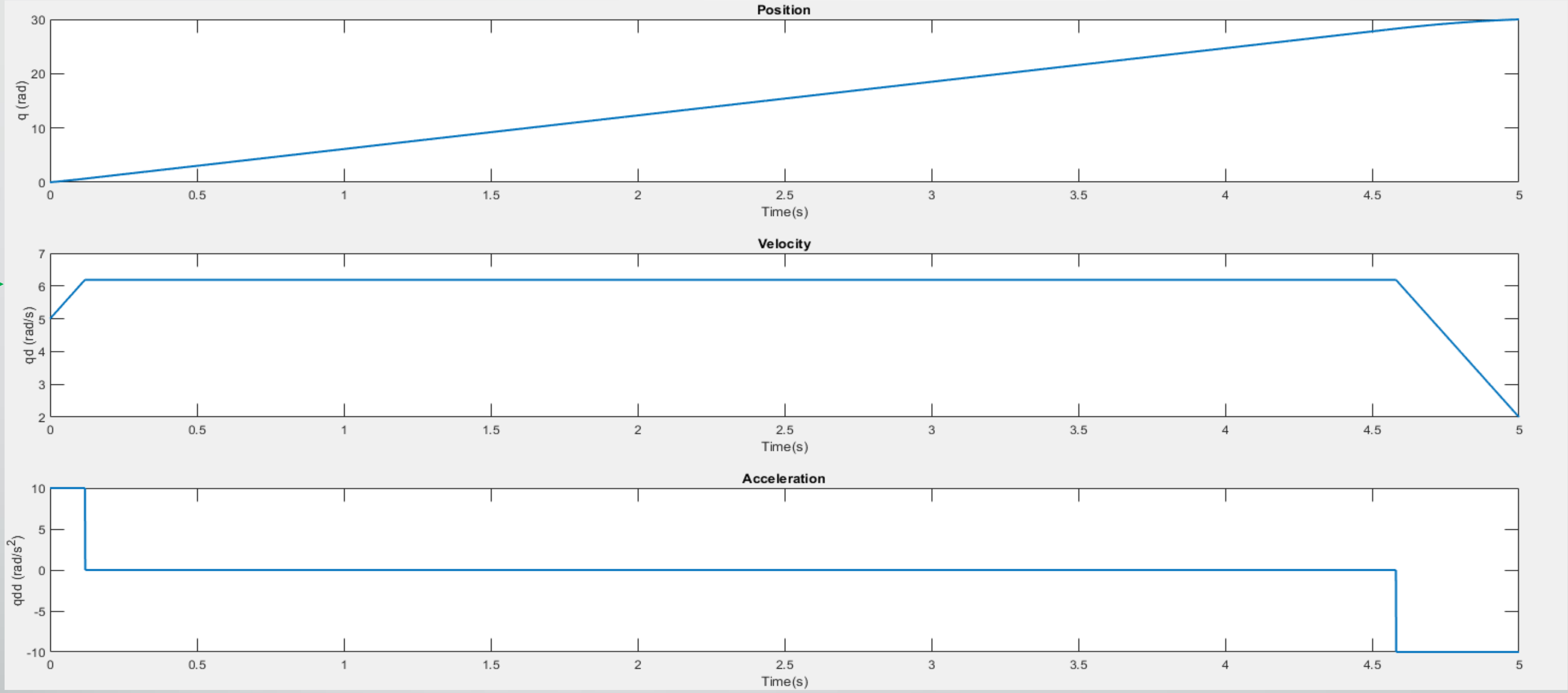


[q, qd, qdd, T]= trapezoidalProfileSymmetric(tc, dqc, ddqc, ti, tf, qf, qi, Ts, subcase);

General trapezoidal trajectory – subcase 1

dqc

ddqc

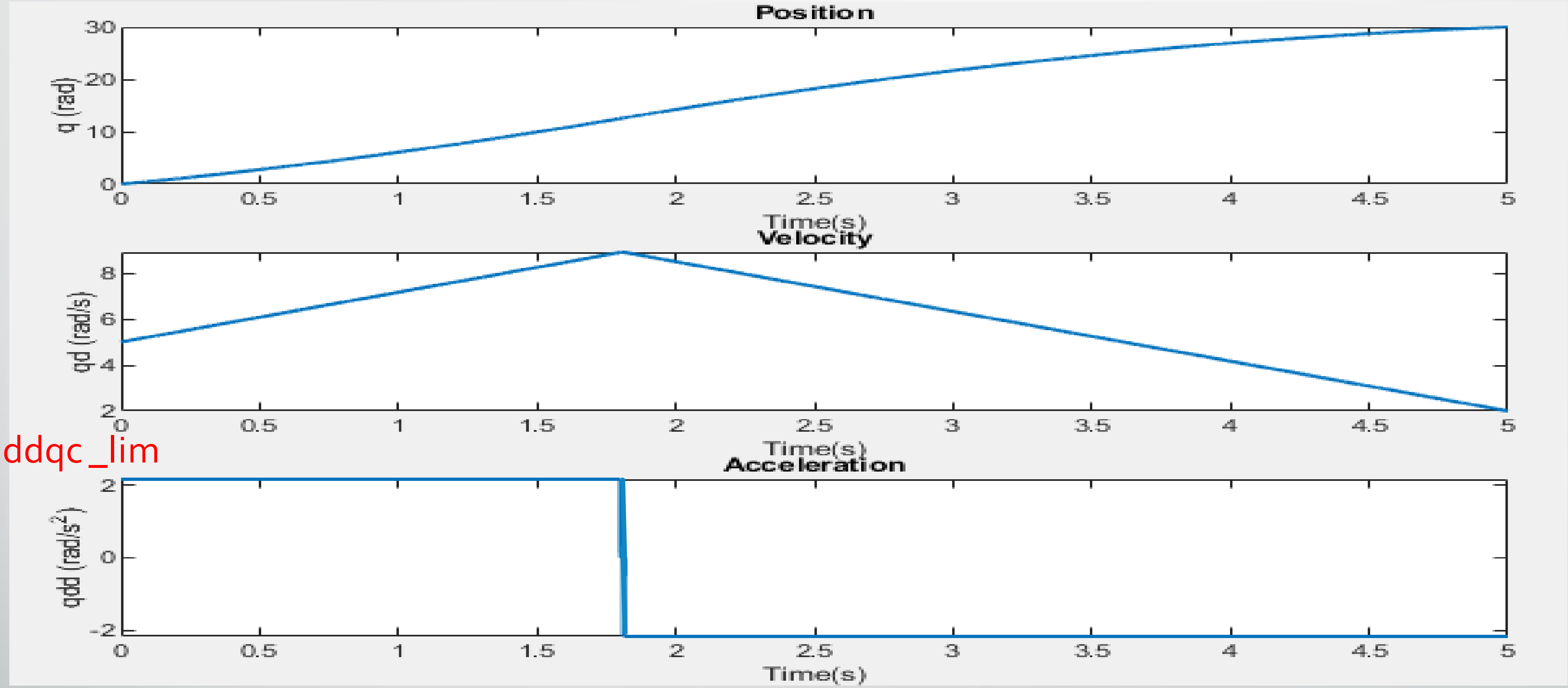


[q, qd, qdd, T]= trapezoidalProfileGeneral(dqc, ddqc, ti, tf, qf, qi, dqi, dqf, Ts, subcase);

General trapezoidal trajectory – subcase 1

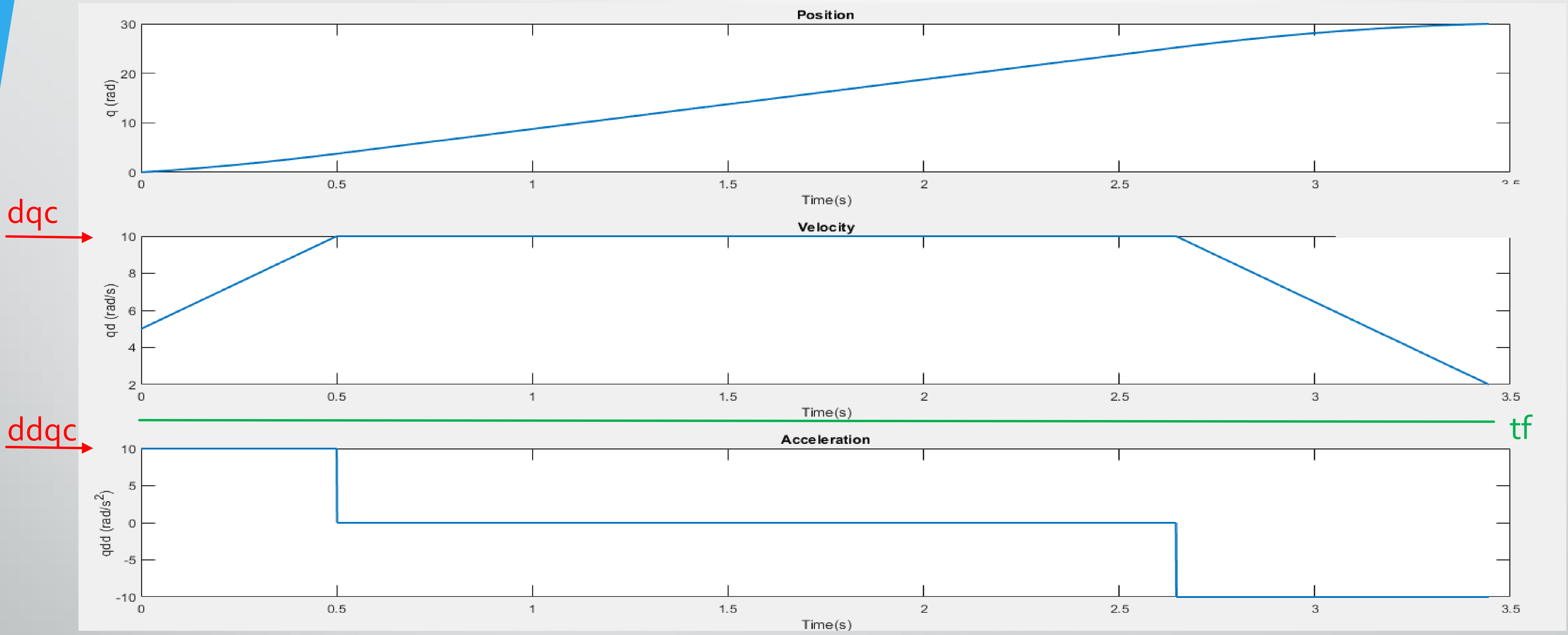
dqc

ddqc == ddqc _lim



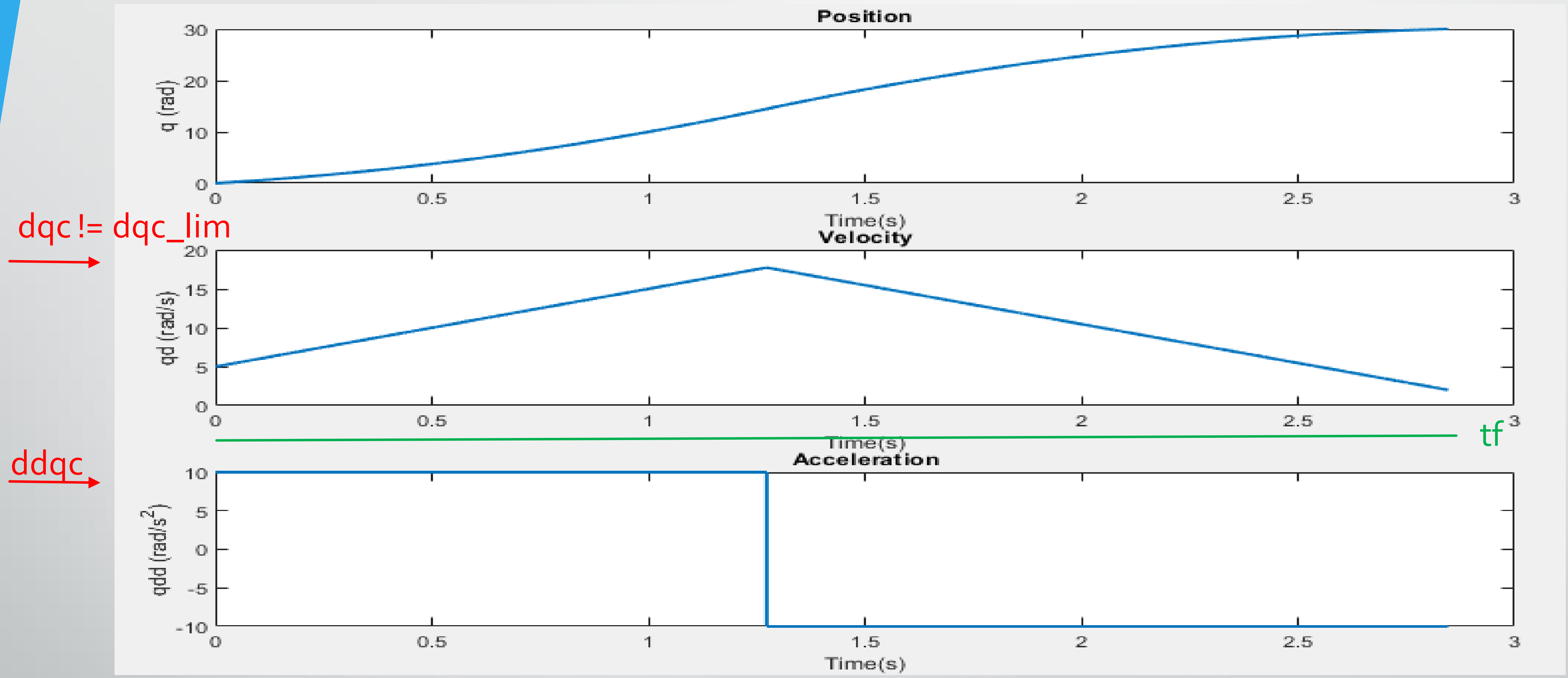
[q, qd, qdd, T]= trapezoidalProfileGeneral(dqc, ddqc, ti, tf, qf, qi, dqi, dqf, Ts, subcase);

General trapezoidal trajectory – subcase 2



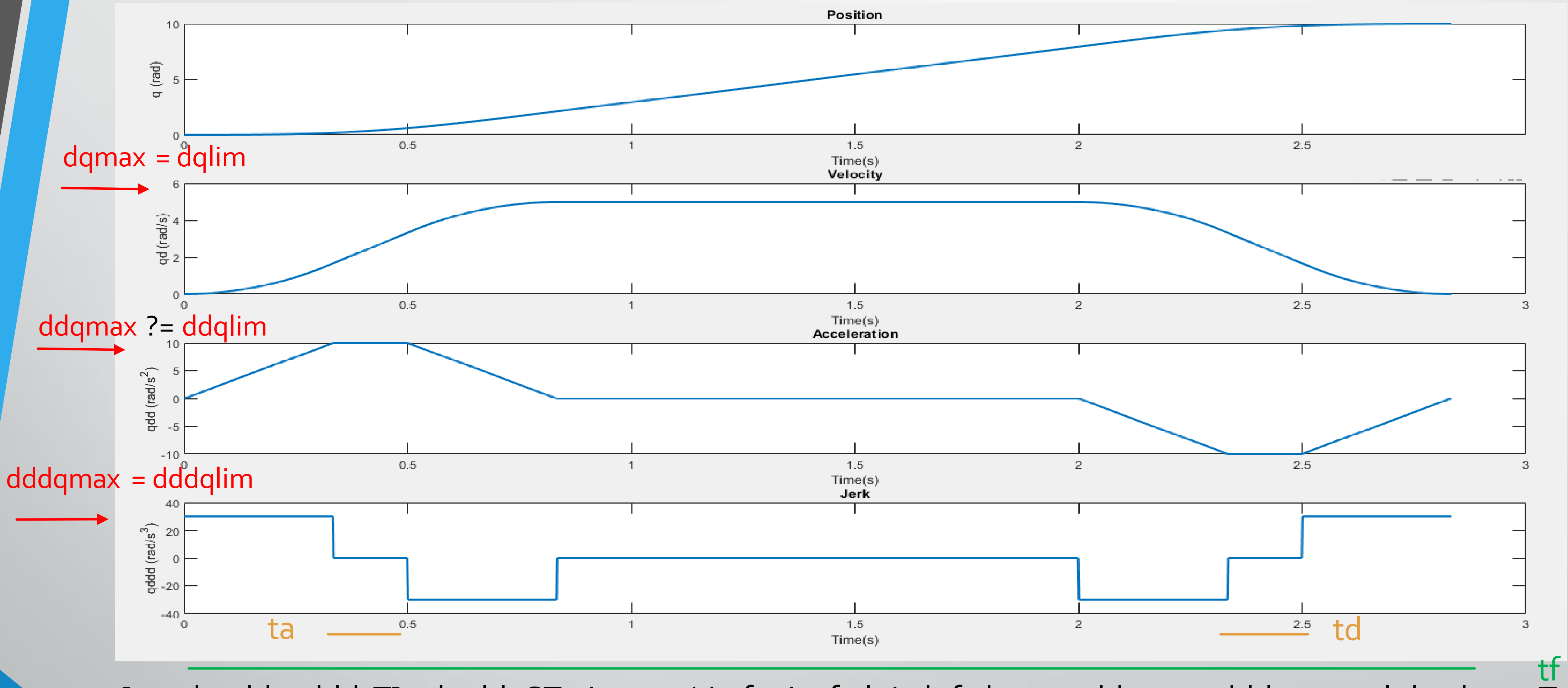
[q, qd, qdd, T]= trapezoidalProfileGeneral(dqc, ddqc, ti, tf, qf, qi, dqi, dqf, Ts, subcase);

General trapezoidal trajectory – subcase 2



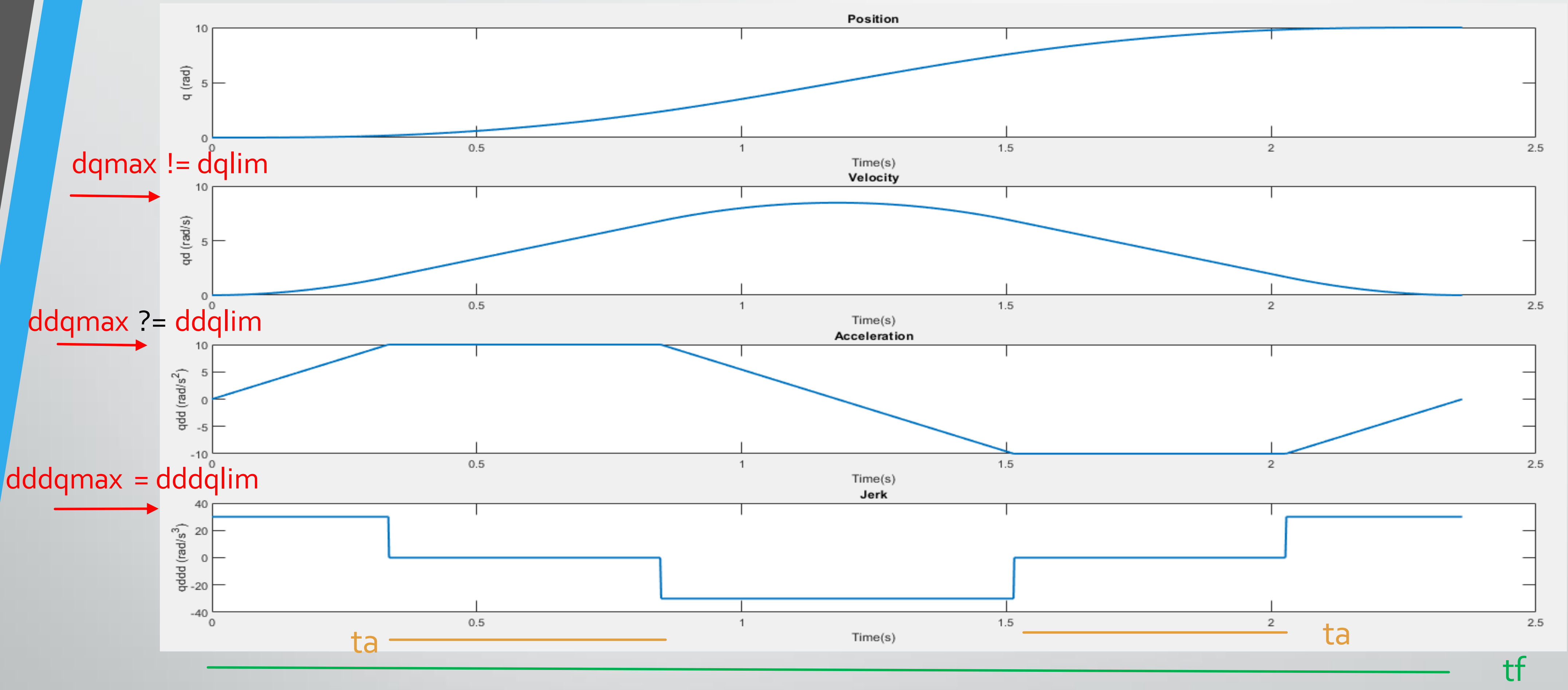
[q, qd, qdd, T]= trapezoidalProfileGeneral(dqc, ddqc, ti, tf, qf, qi, dqi, dqf, Ts, subcase);

Double S trajectory – subcase 1



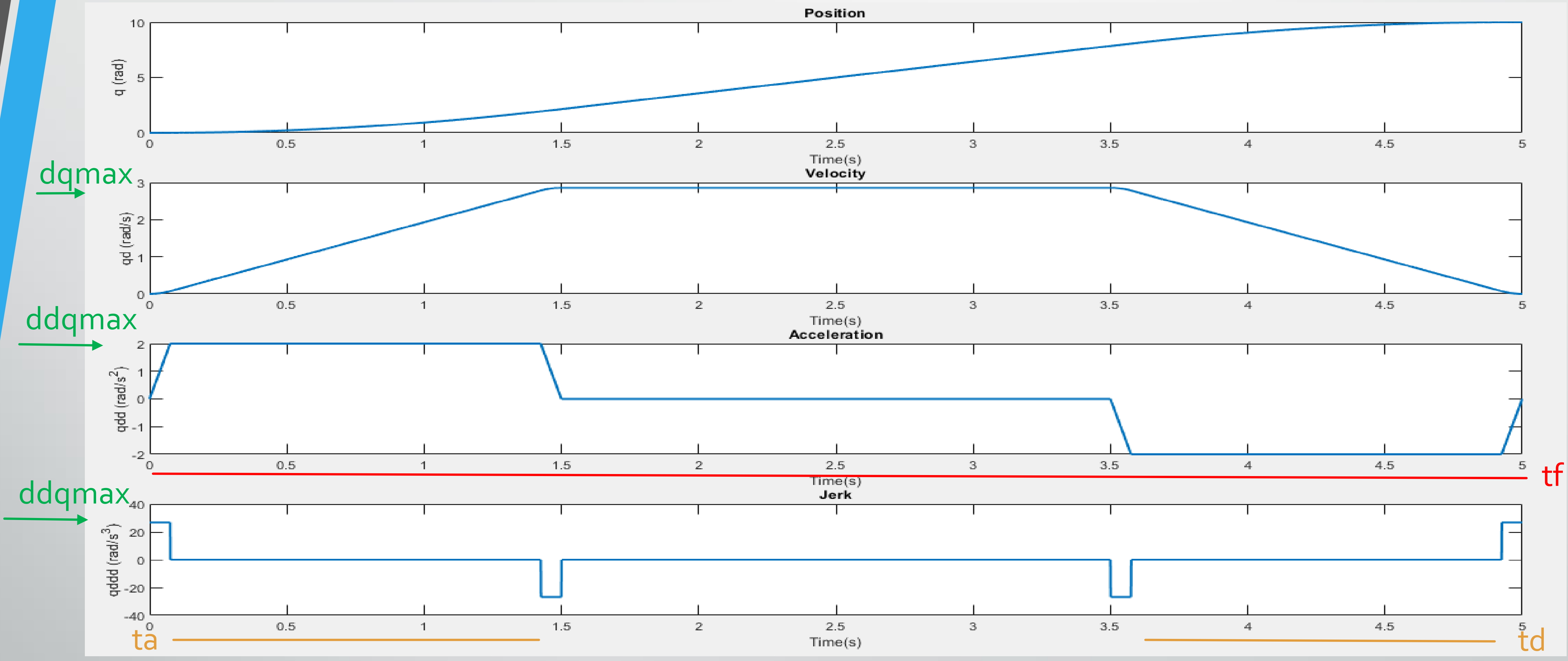
[q, qd, qdd, qddd, T]= doubleSTrajectory(ti, tf, qi, qf, dqi, dqf, dqmax, ddqmax, dddqmax, alpha, beta, Ts);

Double S trajectory – subcase 2



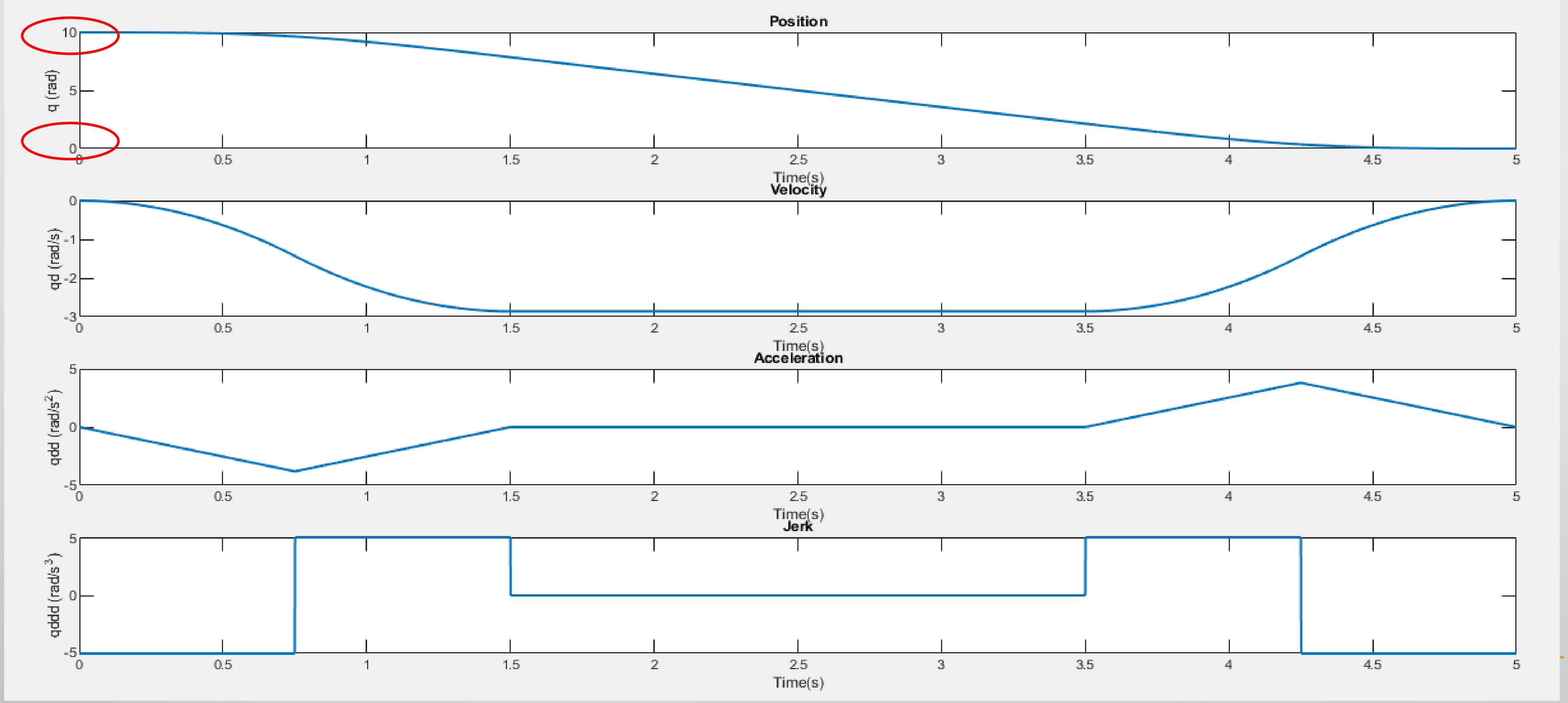
[q, qd, qdd, qddd, T]= doubleSTrajectory(ti, tf, qi, qf, dqi, dqf, dqmax, ddqmax, dddqmax, alpha, beta, Ts);

Double S trajectory – fixed duration



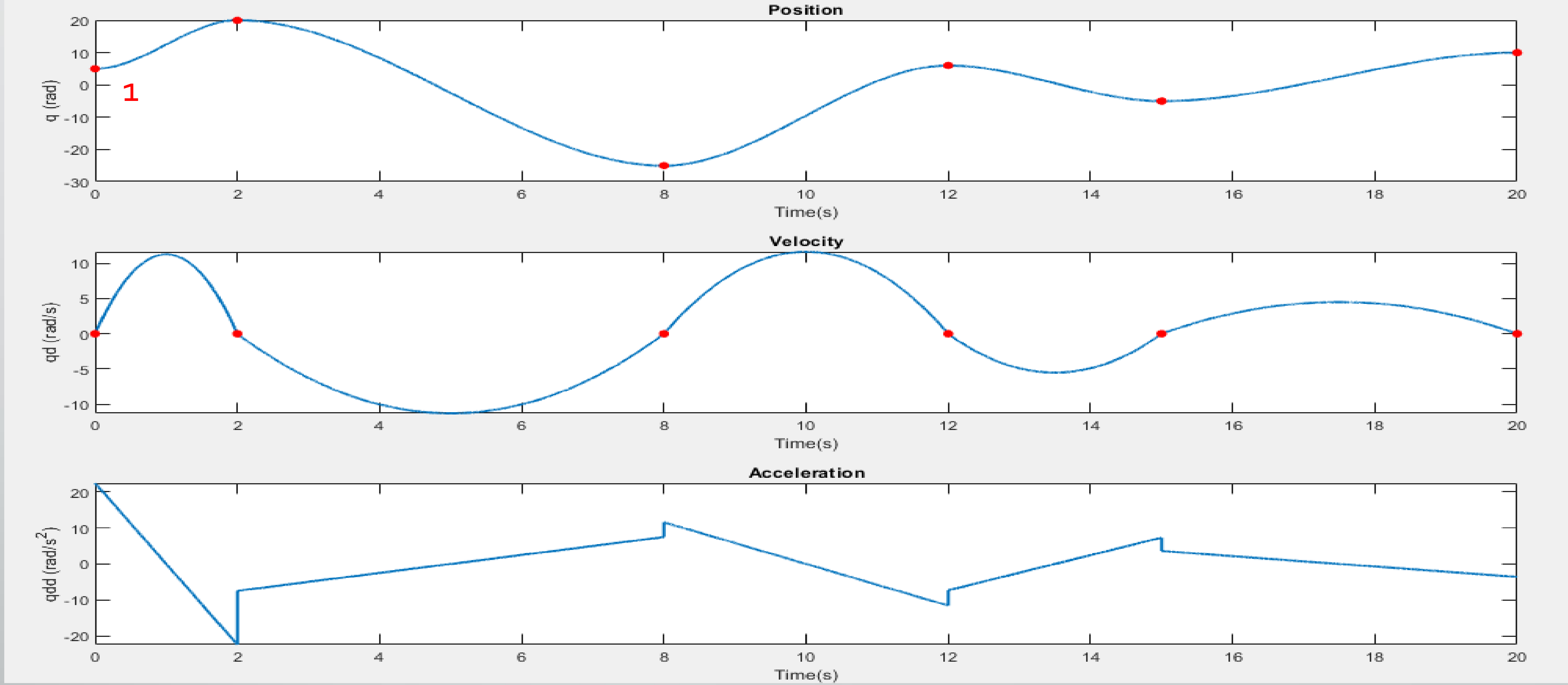
[q, qd, qdd, qddd, T]= doubleSTrajectory(ti, tf, qi, qf, dqi, dqf, dqmax, ddqmax, dddqmax, alpha, beta, Ts);

If qf < qi?



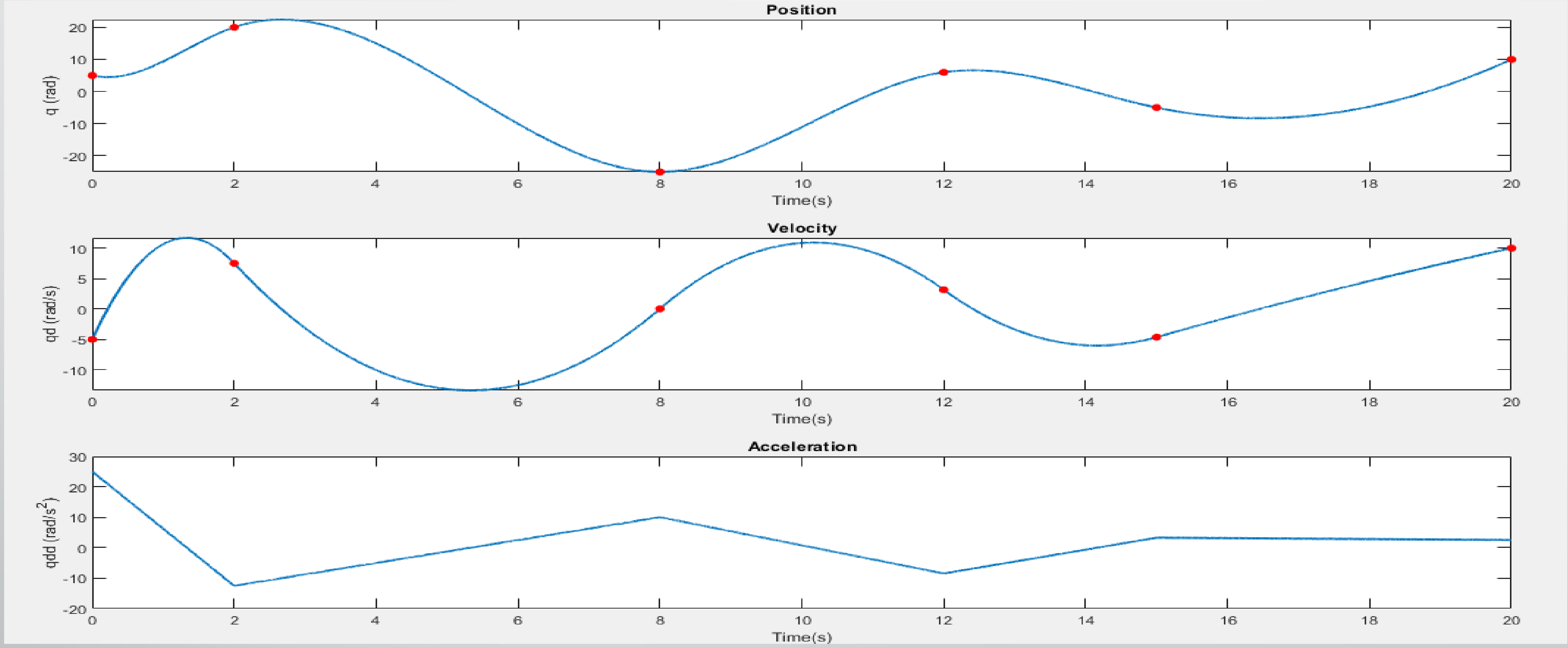
[qi, qf, dqi, dqf] = changeSign(qi, qf, dqi, dqf);

Computed velocities at path points and imposed velocity at initial/final points



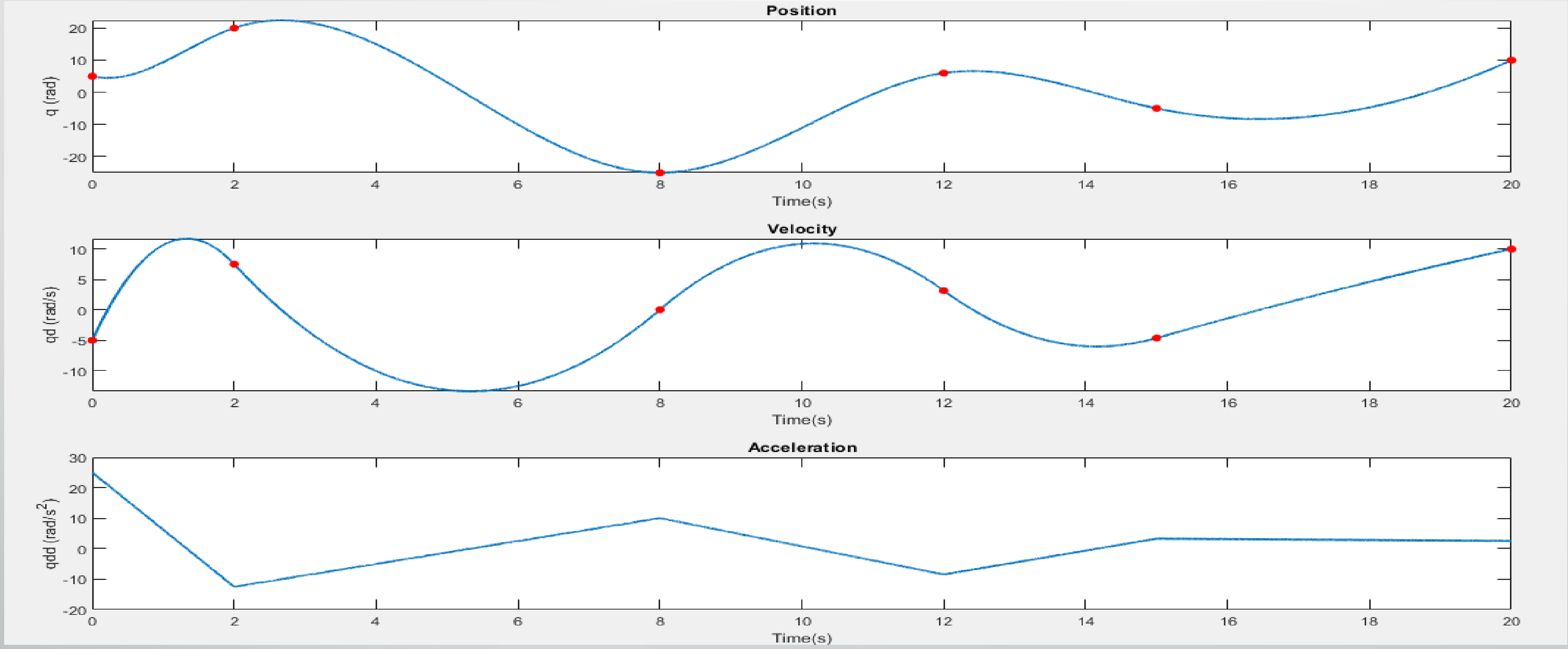
[q, qd, qdd, dqk, T] = computedVelocities(qk, tk, k, dqi, dqf, Ts);

Continuous accelerations at path points and imposed velocity at initial/final points



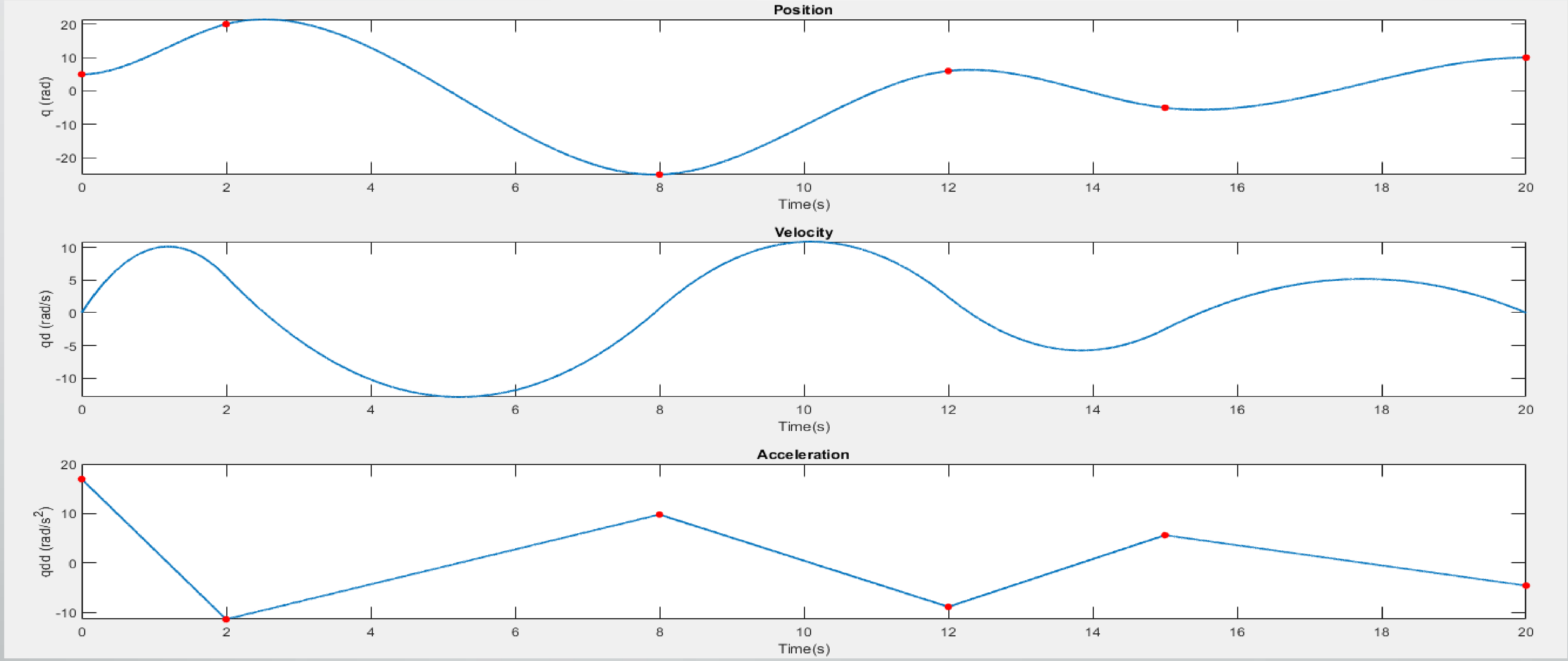
[q, qd, qdd, T,dqk] = continuousAccelerations(qk, tk, k, dqi, dqf, Ts);

Continuous accelerations at path points and imposed velocity at initial/final points



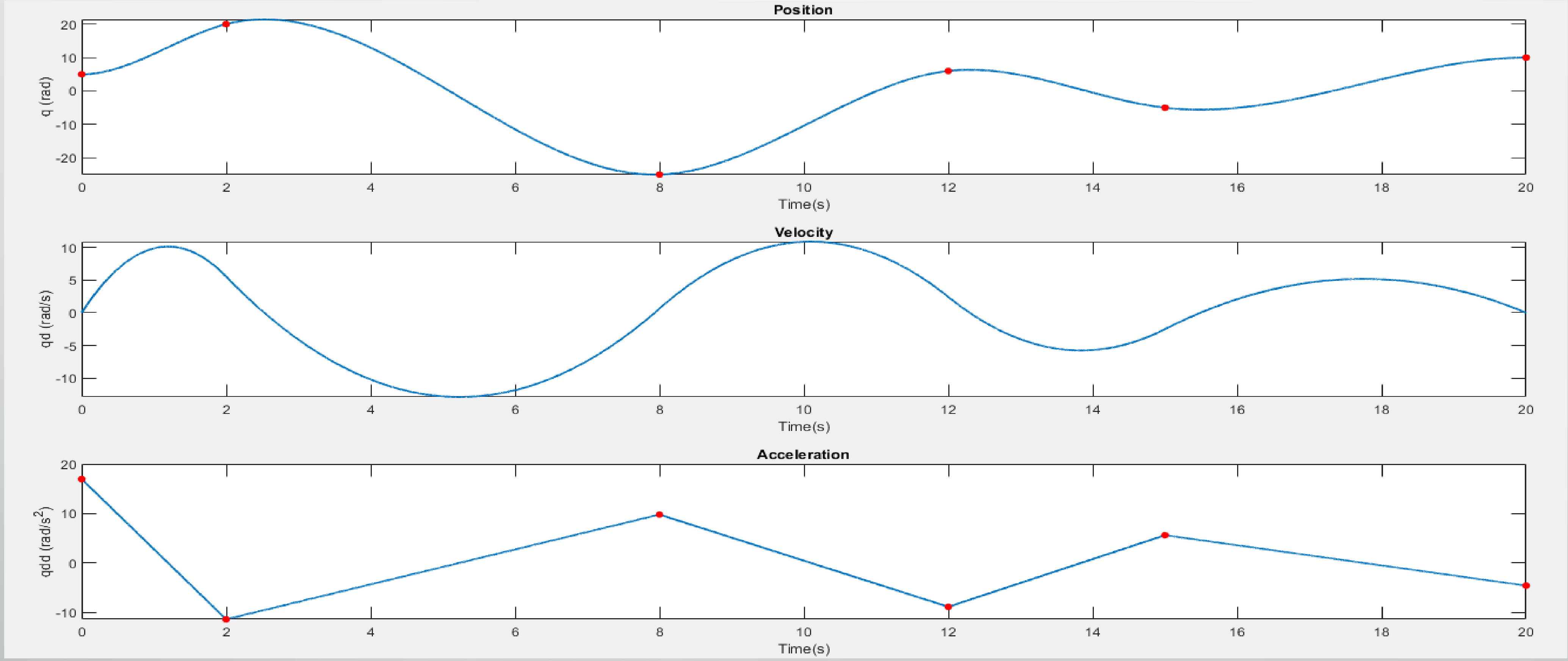
[q, qd, qdd, dqk T] = continuousAccelerationsThomas(qk, tk, k, dqi, dqf, Ts);

Cubic splines based on the accelerations with imposed initial/final velocities



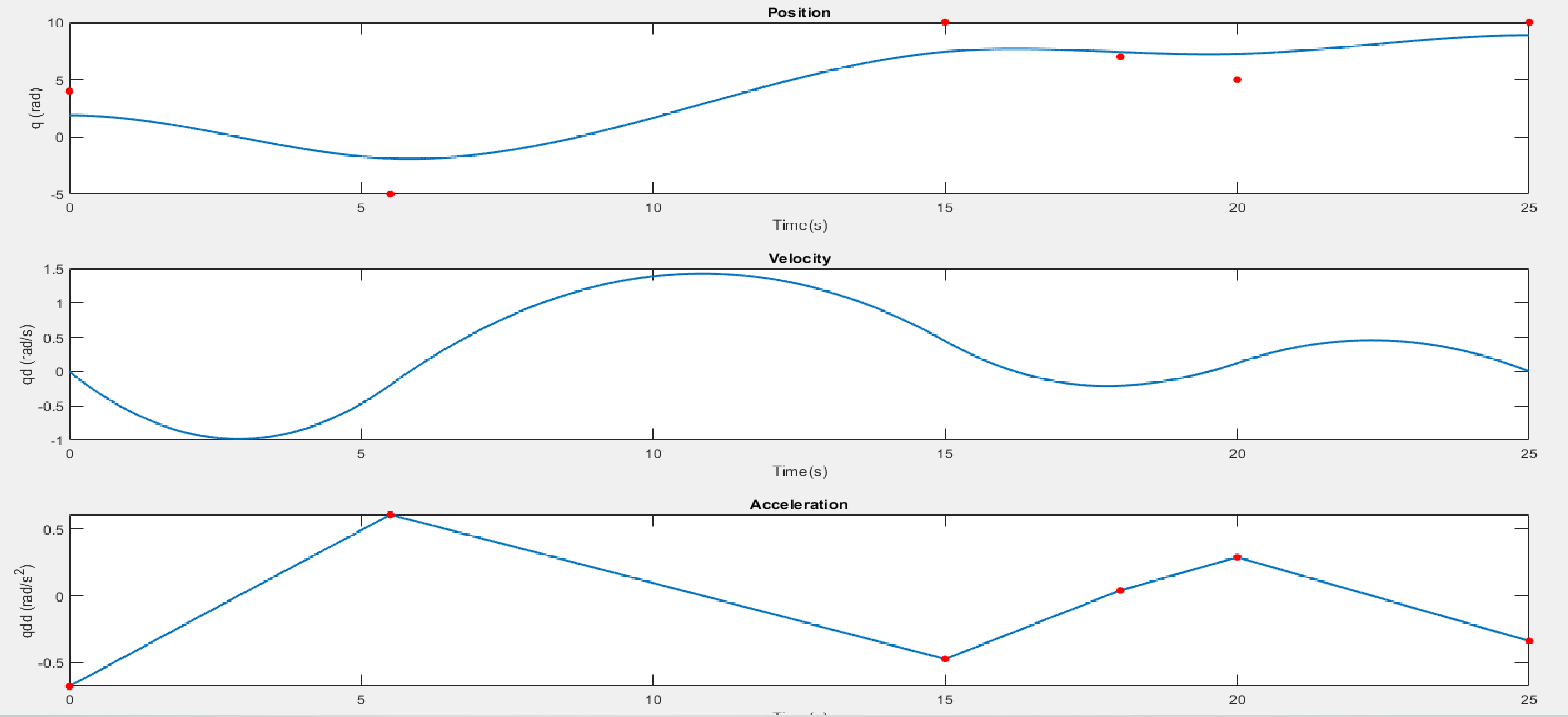
[q, qd, qdd, ddqk, T] = continuousVelocities(qk, dqi, dqf, tk, k, Ts);

Cubic splines based on the accelerations with imposed initial/final velocities



[q, qd, qdd, ddqk, T] = continuousVelocitiesThomas(qk, tk, k, dqi, dqf, Ts);

Smoothing cubic splines

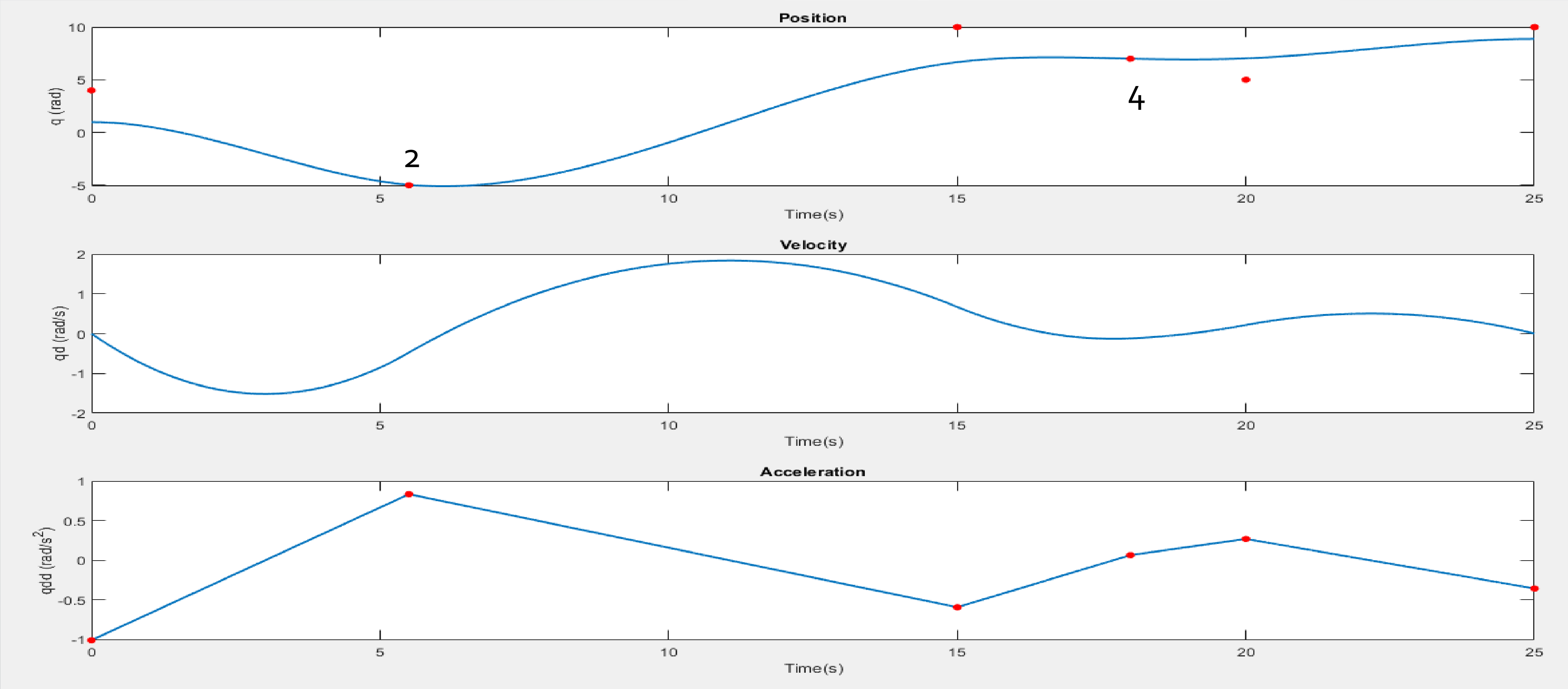


[q, qd, qdd, dds, T] = smoothingSplines(qk, tk, k, Ts, W, lambda);

W = diag([1, 1, 1, 1, 1, 1]);

mu = 0.1;

Smoothing cubic splines

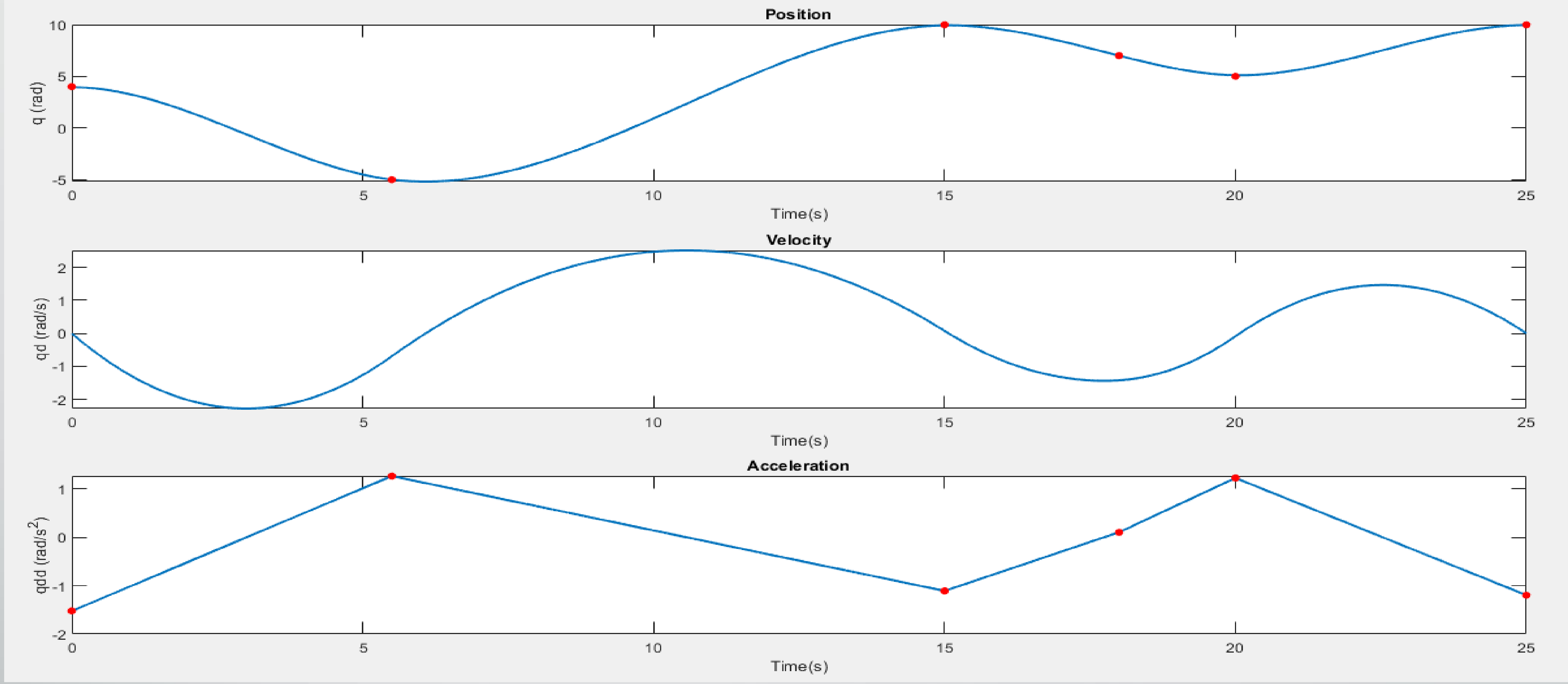


[q, qd, qdd, dds, T] = smoothingSplines(qk, tk, k, Ts, W, lambda);

W = diag([1, 100, 1, 100, 1, 1]);

mu = 0.1;

Smoothing cubic splines

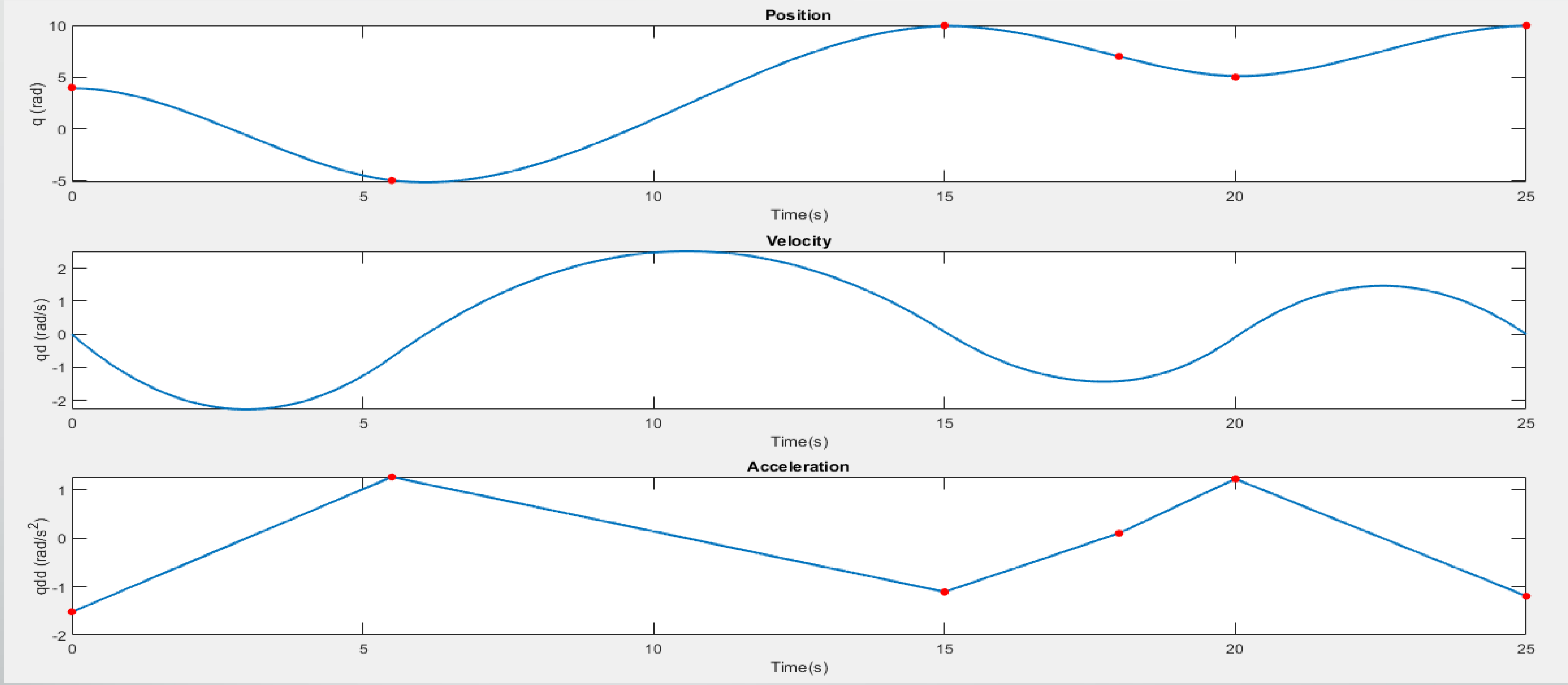


[q, qd, qdd, dds, T] = smoothingSplines(qk, tk, k, Ts, W, lambda);

W = diag([1, 100, 1, 100, 1, 1]);

mu = 0.9;

Smoothing cubic splines



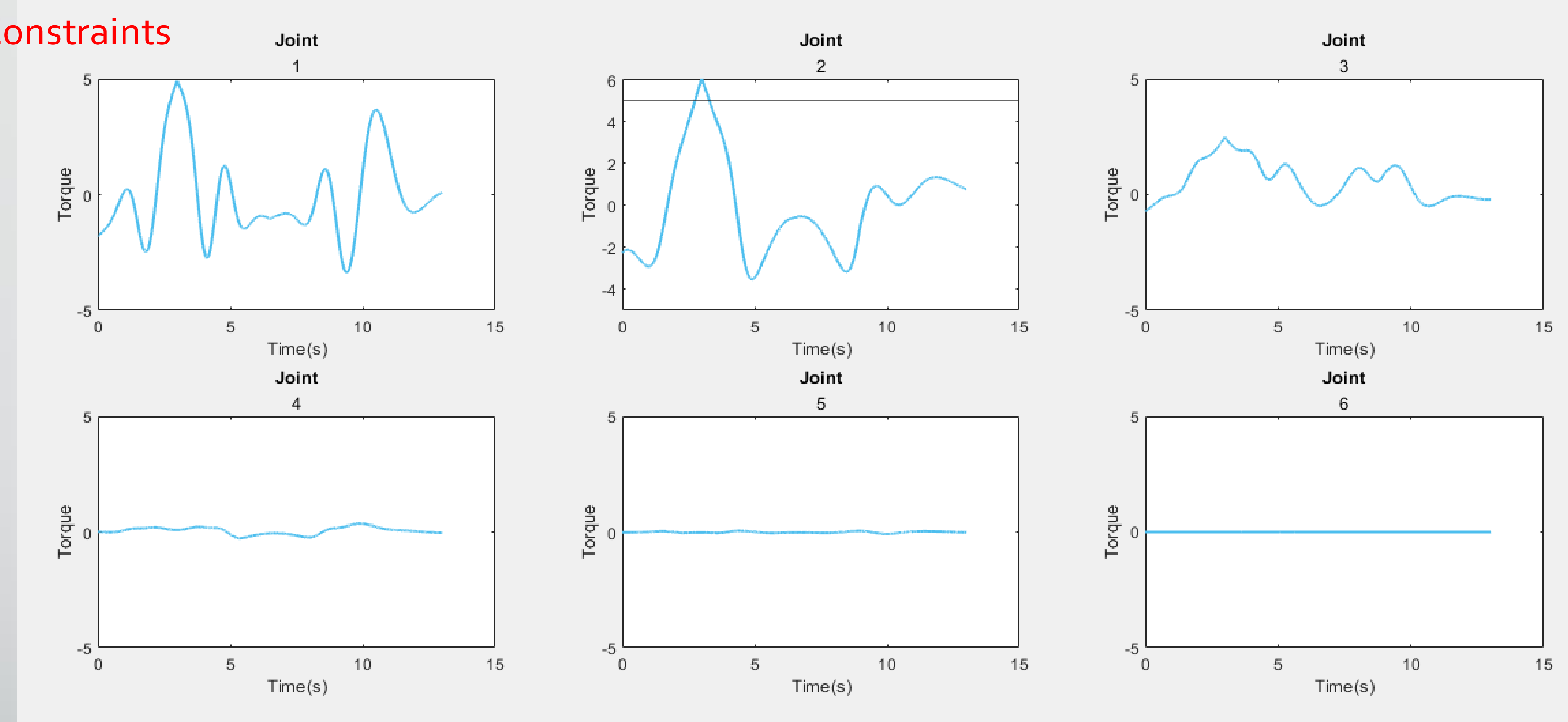
[q, qd, qdd, dds, T] = smoothingSplines(qk, tk, k, Ts, W, lambda);

W = diag([1, 1, 1, 1, 1, 1]);

mu = 0.9;

Model-based Trajectory

tauConstraints

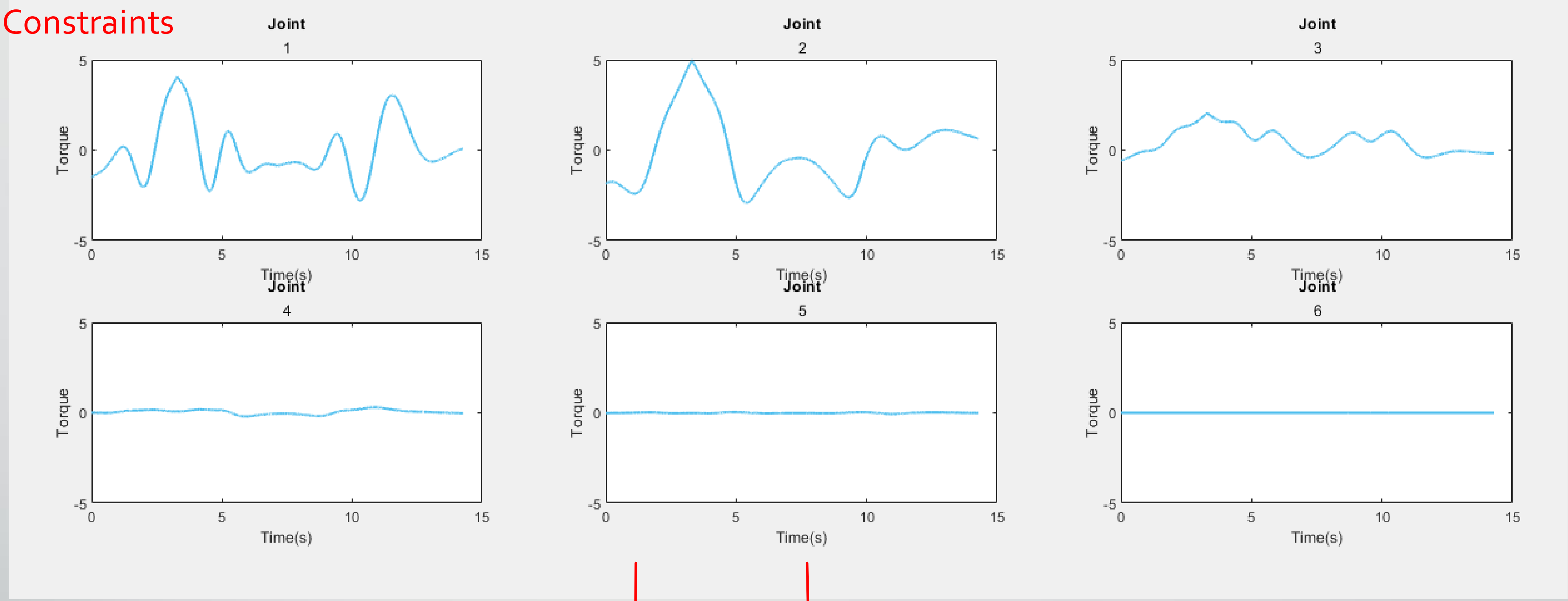


[q, dq, ddq, dqk, T_series] = continuousAccelerationsThomas(qk(i,:), tk, k(1,i), dqi(i, 1), dqf(i, 1), Ts);

tauNoG(:,i) = inv_dyn_recursive_NewtonEulero(dh, DataPositions(:,i), DataVelocities(:,i), DataAccelerations(:,i), gravity) - g(:,i);

Model-based Trajectory

tauConstraints

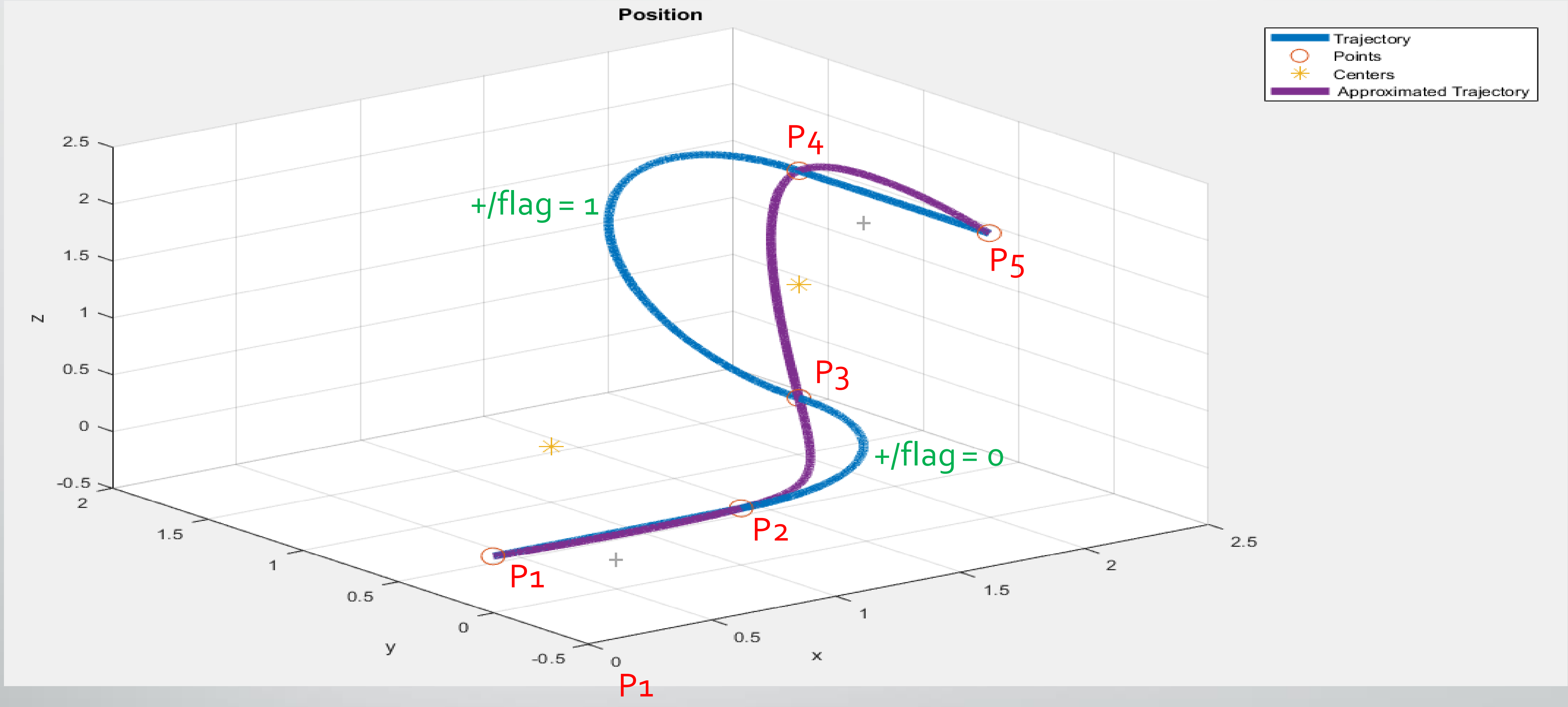


lambda = findLambda(tau, tauConstraints)

T = T/lambda; DataVelocities = DataVelocities*lambda;

DataAccelerations = DataAccelerations*lambda^2;

3D trajectory

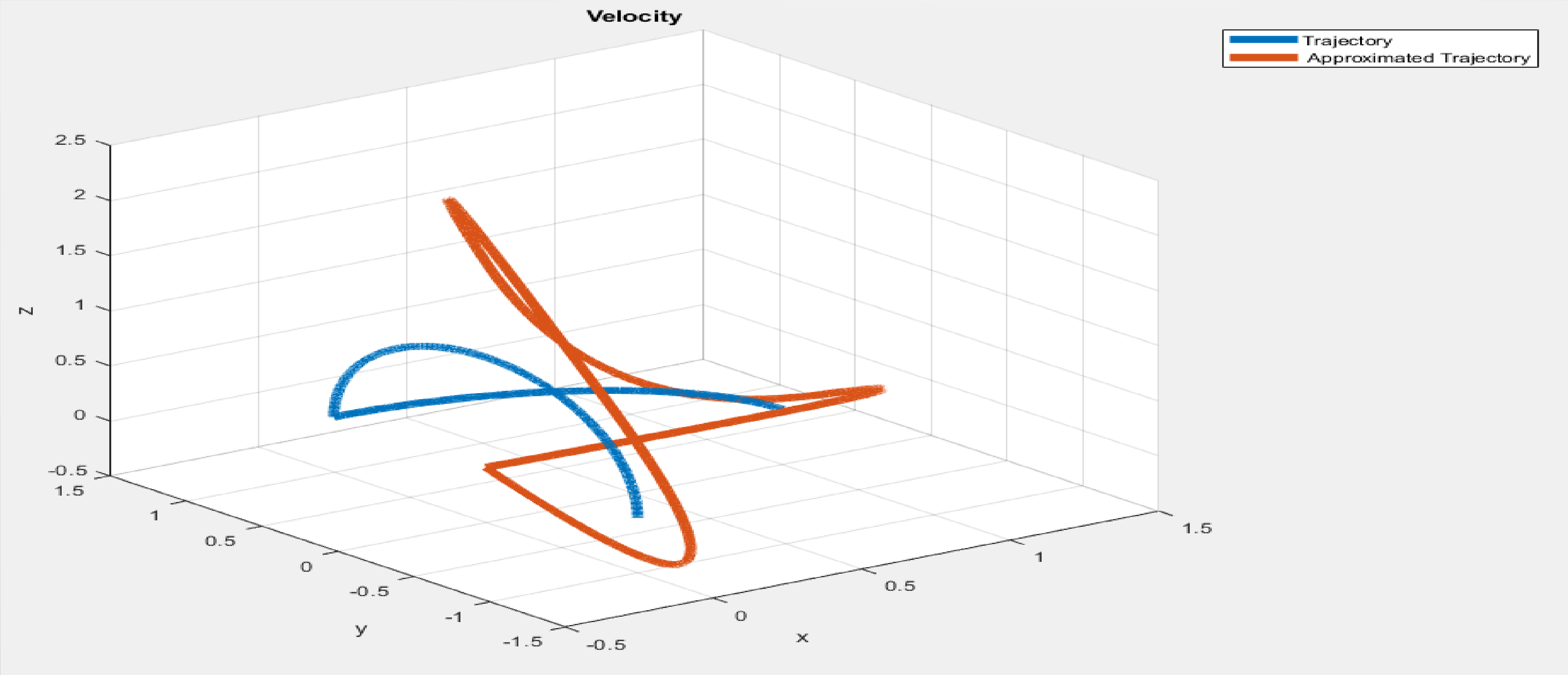


+ [P, DP, DDP, DDDP, T, p dp, ddp] = linearMotion(pi, pf, Ts, P, DP, DDP, DDDP, T);

+ [P, DP, DDP, DDDP, T, p, dp, ddp] = circularMotion(pi, pf, Ts, c, P, DP, DDP,DDDP, T, flag);

- [Q, QD, QDD, T] = cartesianApproximation(Pg)

3D trajectory

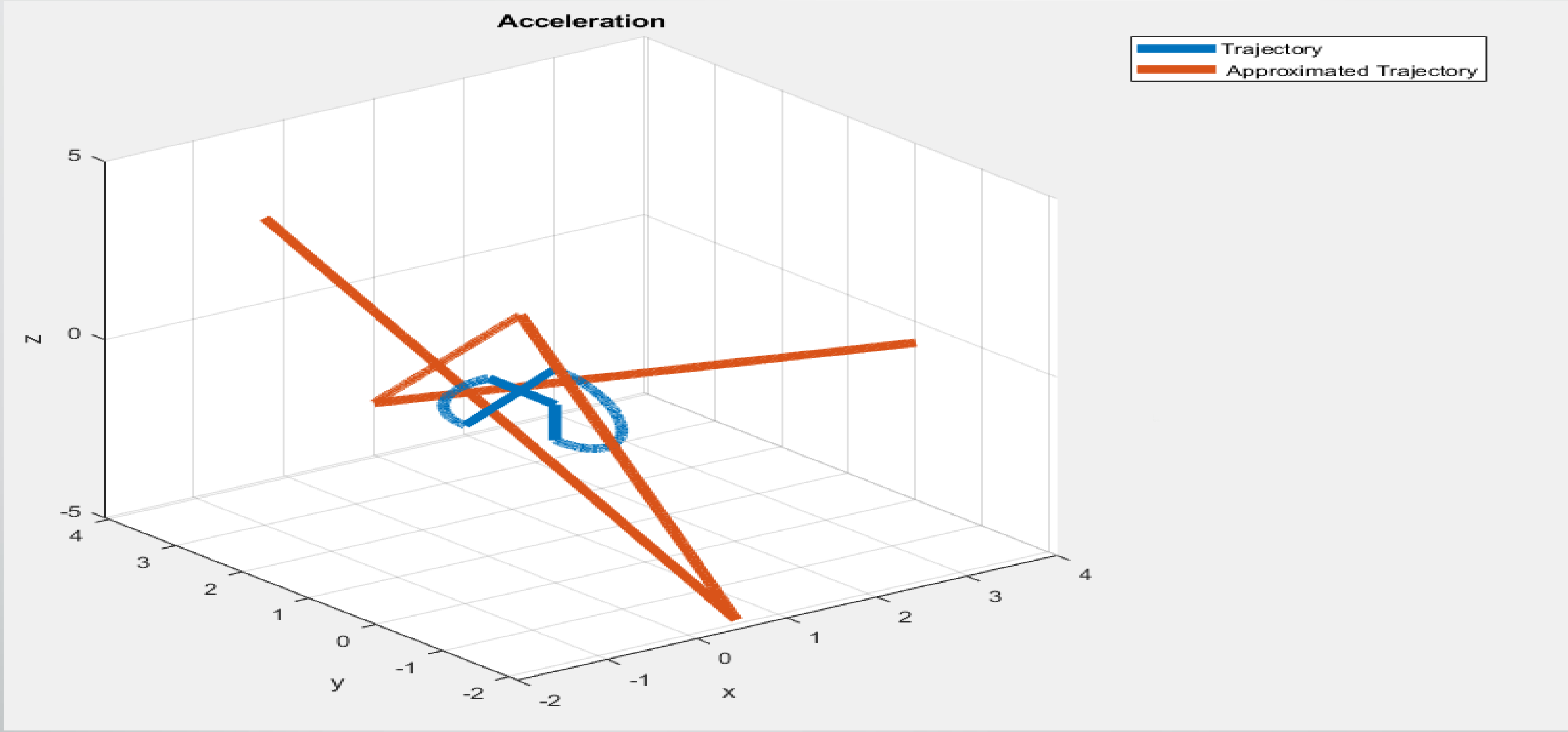


[P, DP, DDP, DDDP, T, p dp, ddp] = linearMotion(pi, pf, Ts, P, DP, DDP, DDDP, T);

[P, DP, DDP, DDDP, T, p, dp, ddp] = circularMotion(pi, pf, Ts, c, P, DP, DDP,DDDP, T, flag);

[Q, QD, QDD, T] = cartesianApproximation(Pg)

3D trajectory

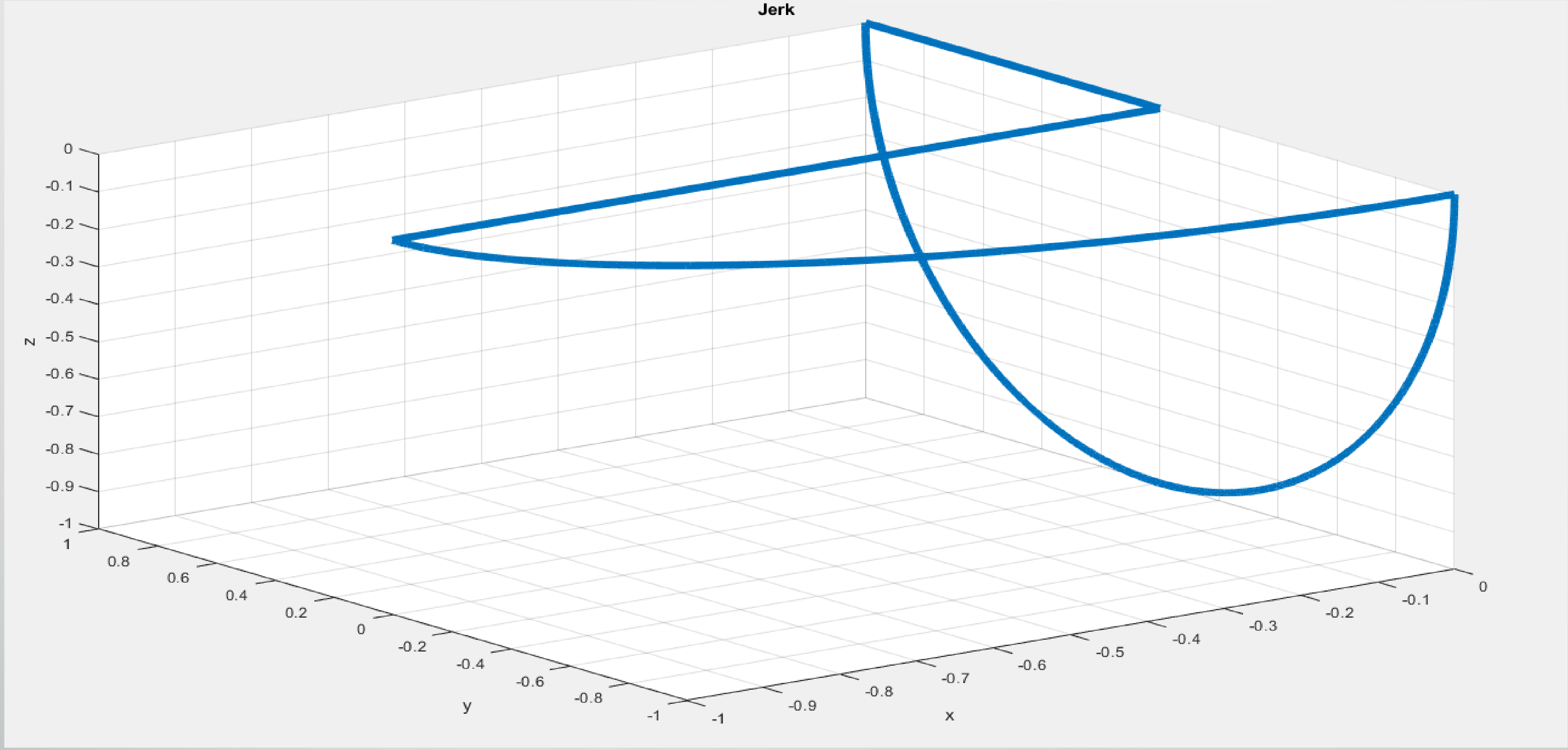


[P, DP, DDP, DDDP, T, p dp, ddp] = linearMotion(pi, pf, Ts, P, DP, DDP, DDDP, T);

[P, DP, DDP, DDDP, T, p, dp, ddp] = circularMotion(pi, pf, Ts, c, P, DP, DDP,DDDP, T, flag);

[Q, QD, QDD, T] = cartesianApproximation(Pg)

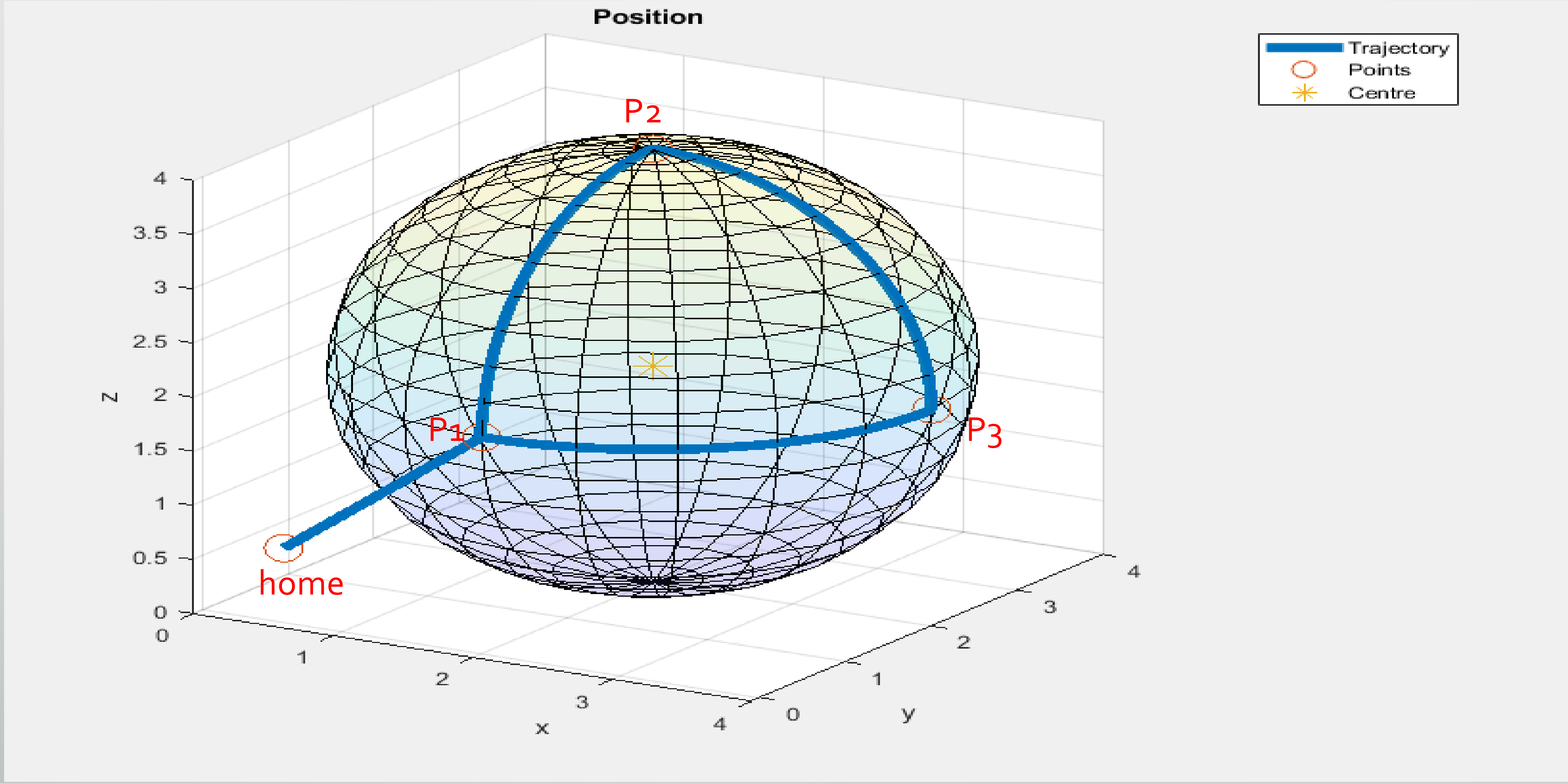
3D trajectory



[P, DP, DDP, DDDP, T, p dp, ddp] = linearMotion(pi, pf, Ts, P, DP, DDP, DDDP, T);

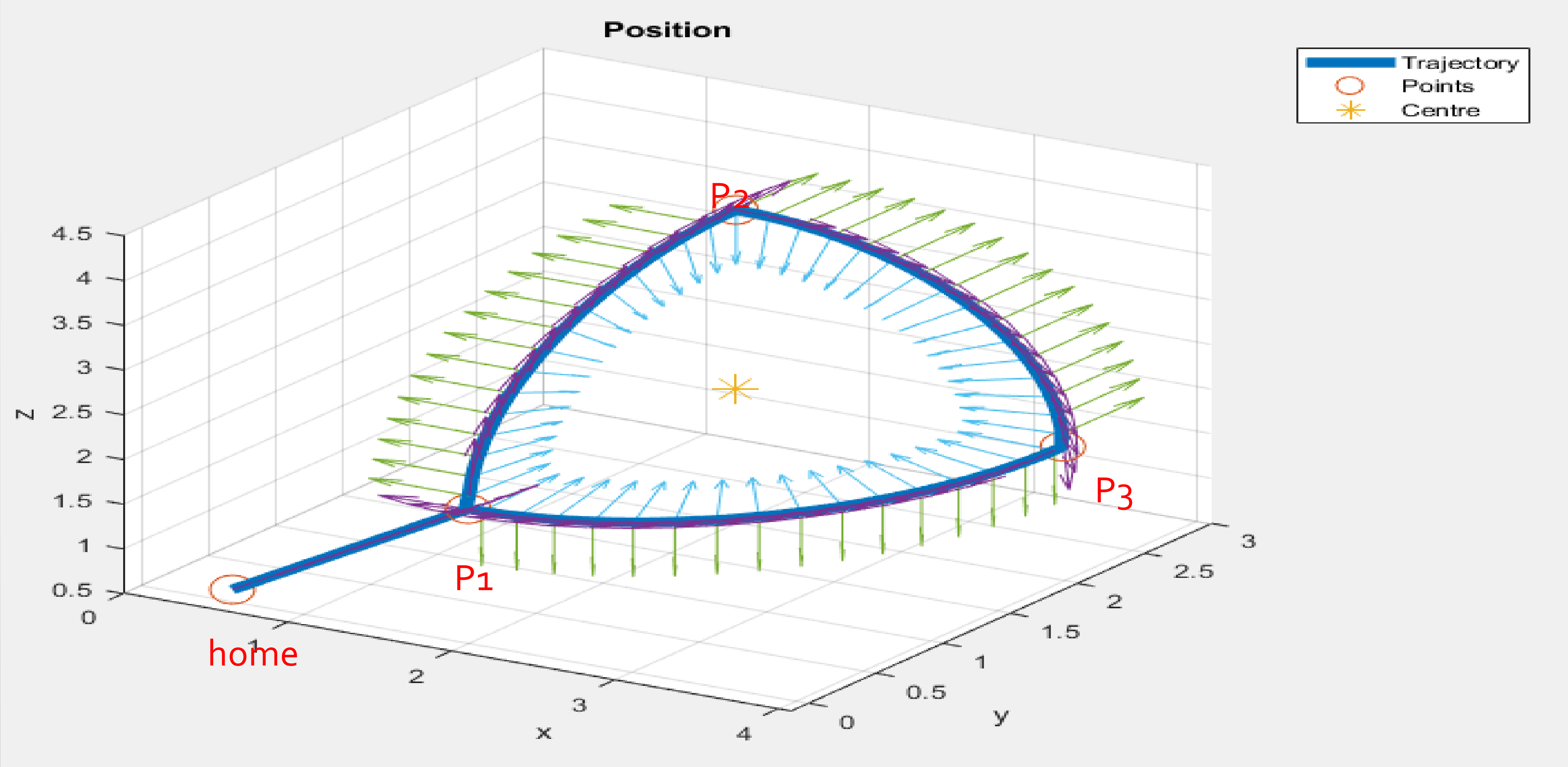
[P, DP, DDP, DDDP, T, p, dp, ddp] = circularMotion(pi, pf, Ts, c, P, DP, DDP,DDDP, T, flag);

3D trajectory – Position



[Ptilde, DPtilde, DDPtilde, T] = linearTilde(x0(1:3), p1, Ts, Ptilde, DPtilde, DDPtilde, T, ti, tf);
[Ptilde, DPtilde, DDPtilde, T] = circularTilde(p2, p3, Ts, p0', Ptilde, DPtilde, DDPtilde, T, flag, ti, tf);

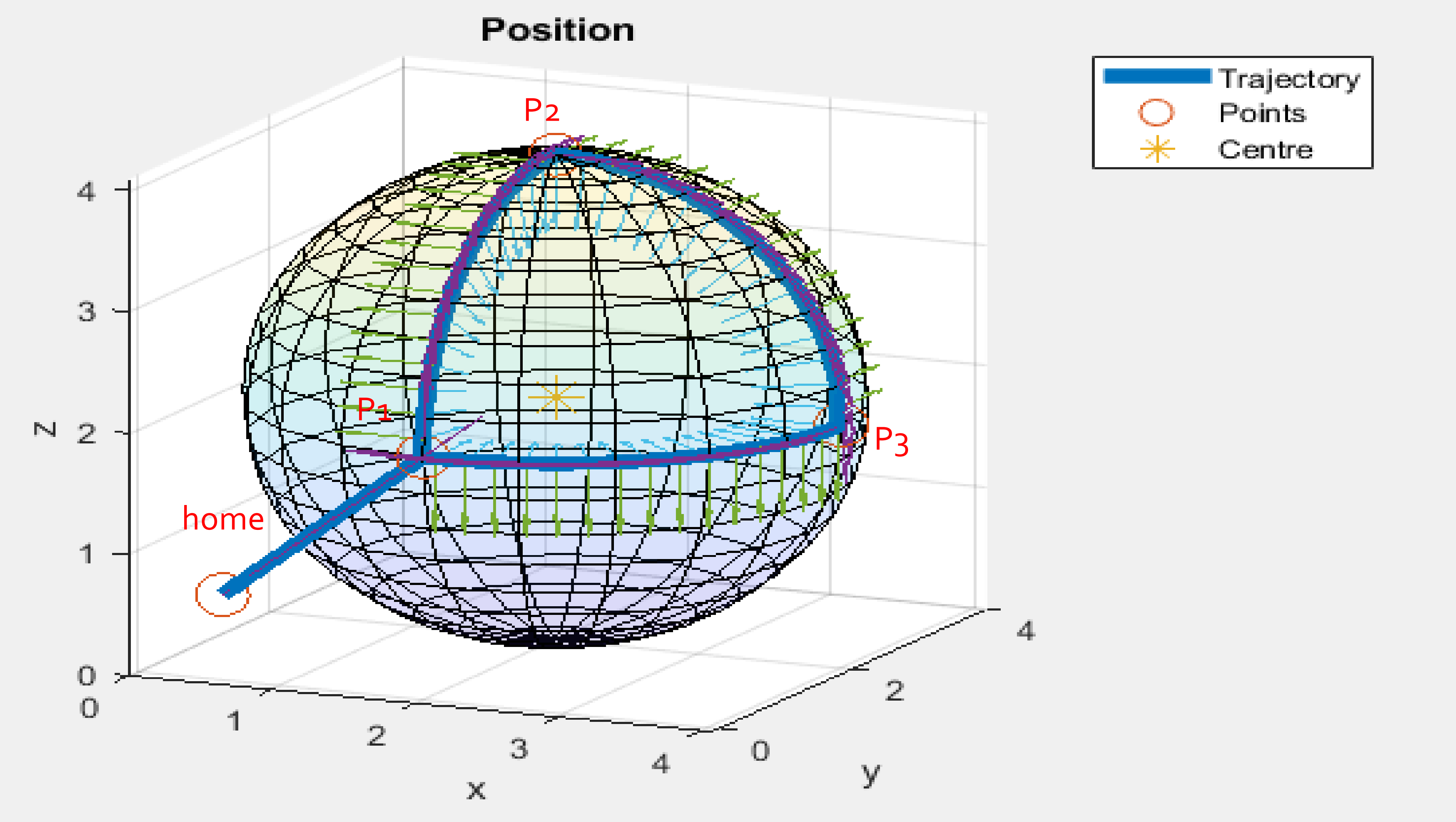
3D trajectory – EE Orientation



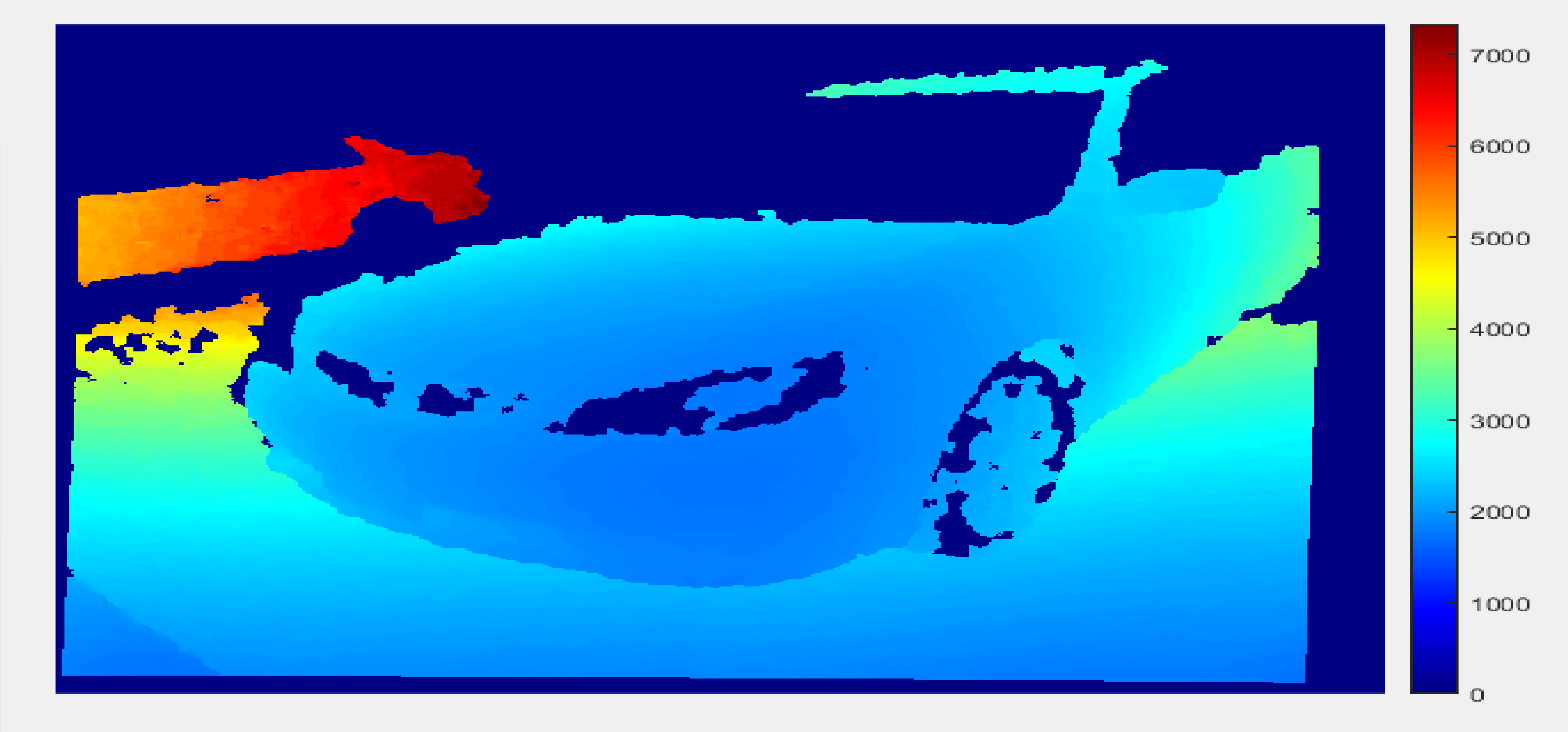
[Otilde, DOtilde, DDOtilde, TO, O_EE_t, O_EE_n, O_EE_b] = FrenetFrame(p, dp, ddp,ti, tf, p1, p2 ...
Otilde, DOtilde, DDOtilde, TO, O_EE_t, O_EE_n, O_EE_b , Ts);

[Otilde, DOtilde, DDOtilde, TO] = EEOrientation(PHI_i, PHI_f, Ts, Otilde, DOtilde, DDOtilde, TO, ti, tf...
pi, pf);

3D trajectory – Position and EE Orientation



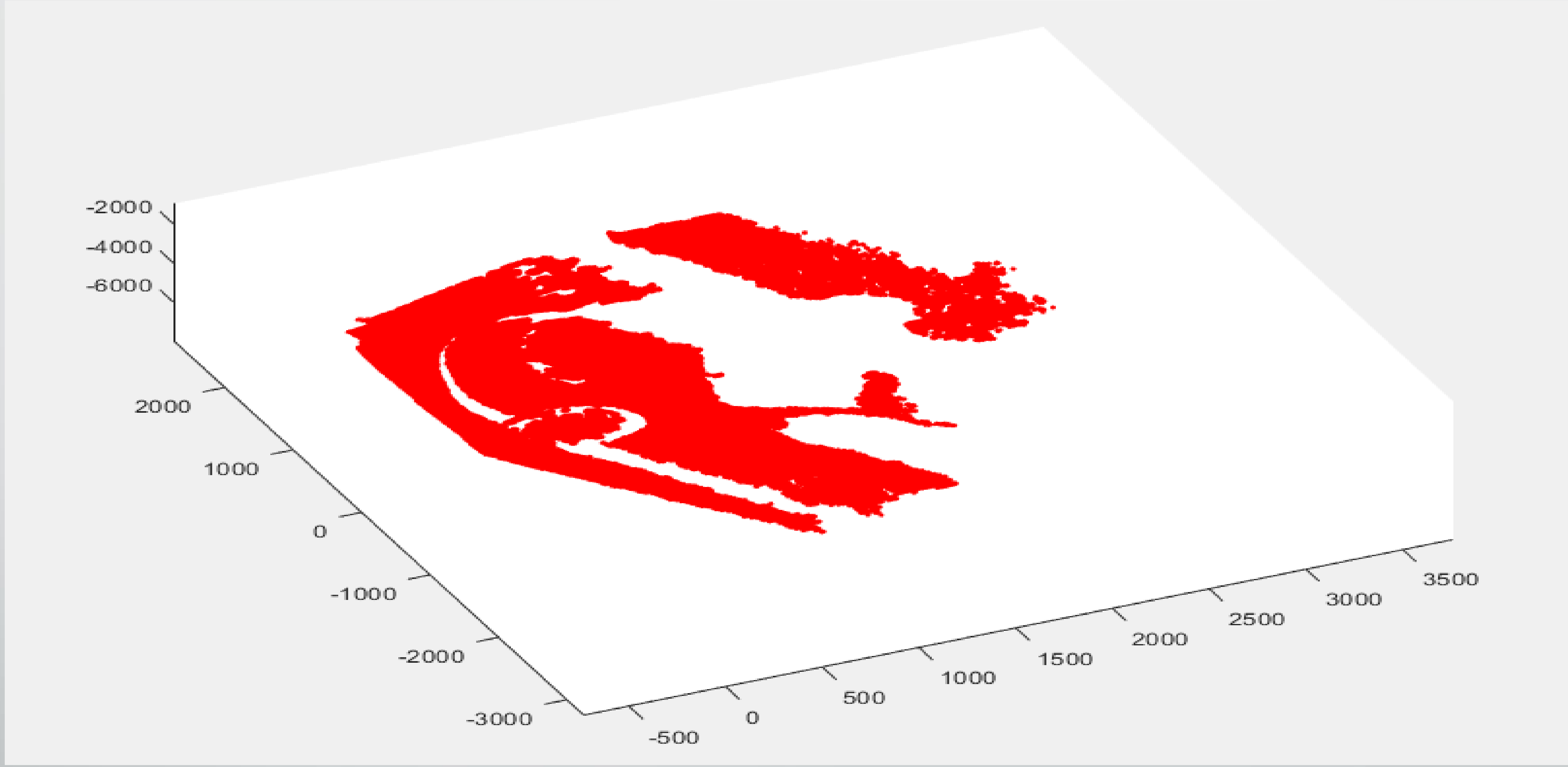
3D Acquisition – Depth image



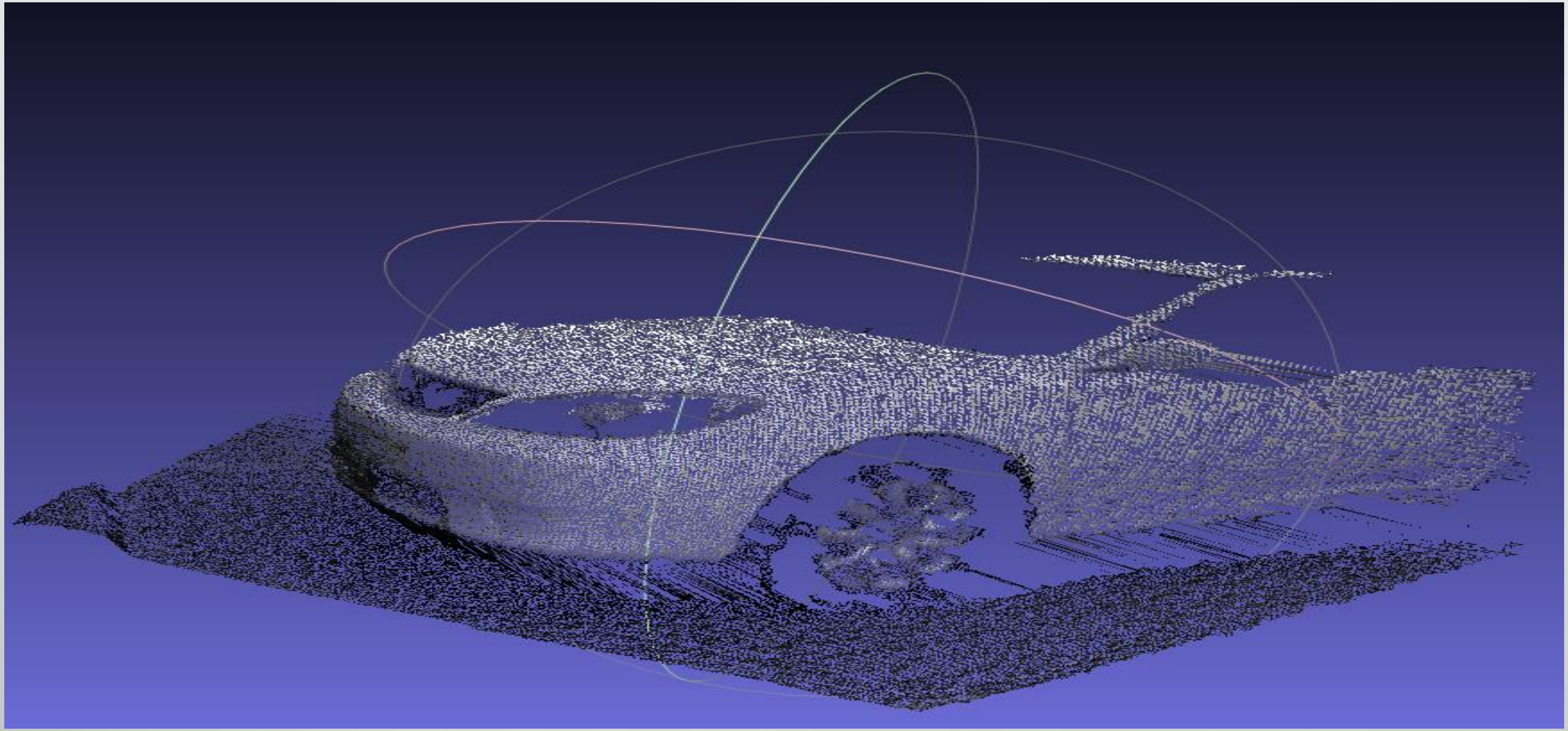
imshow(depth,[min(depth,[],'all'),max(depth,[],'all')]);

colormap jet; colorbar; colormap;

3D Acquisition – Cloud Points

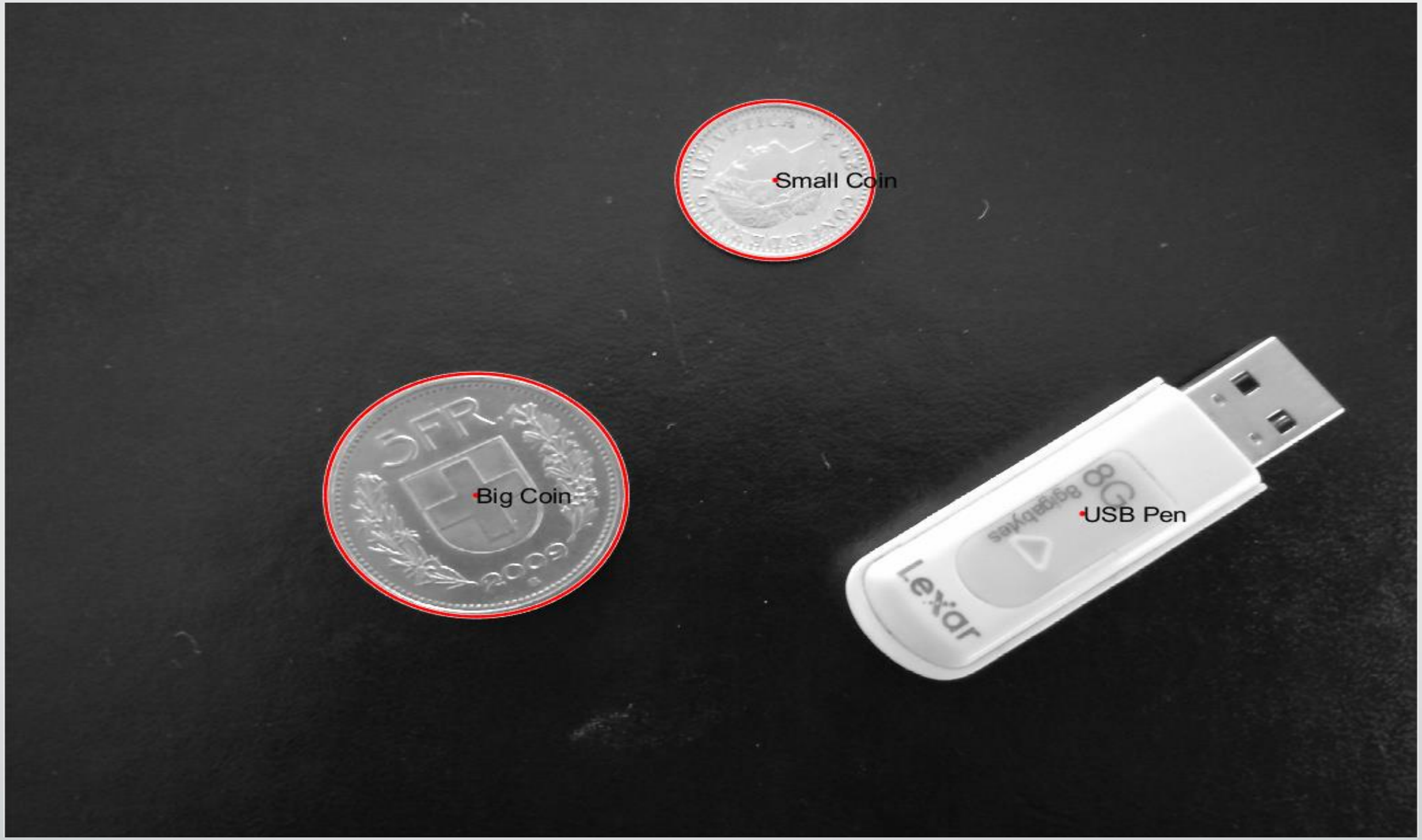


3D Acquisition – Cloud Points



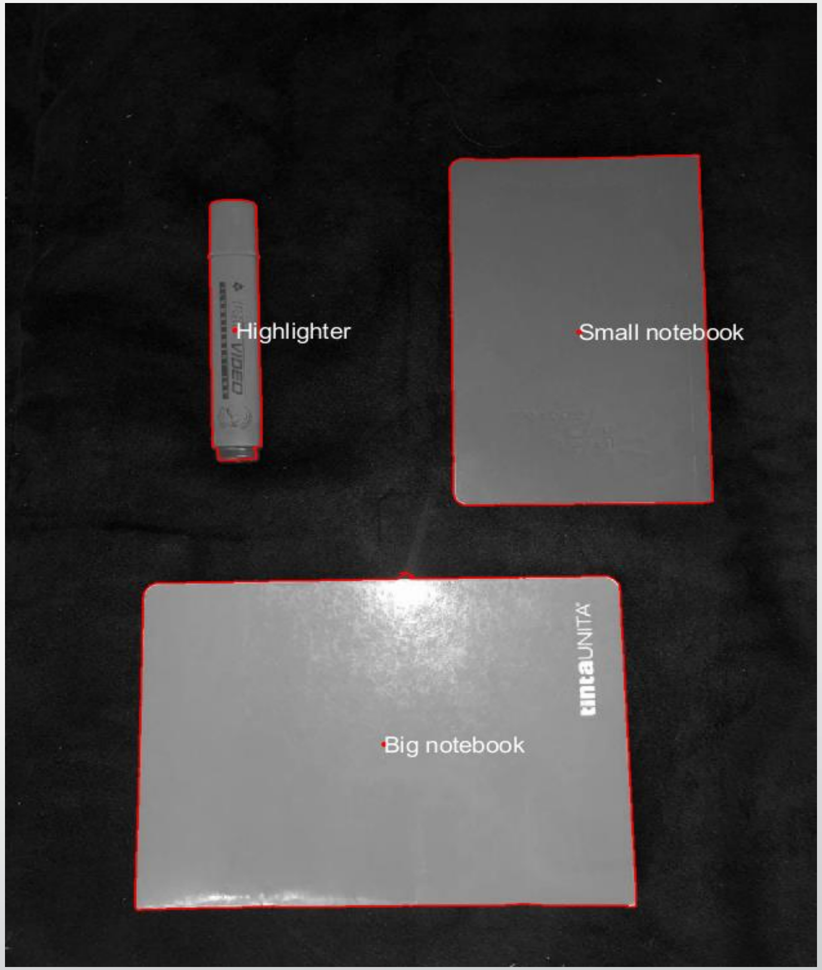
exportMeshToPly(vertices, faces, vertex_color, name);

Image Analysis



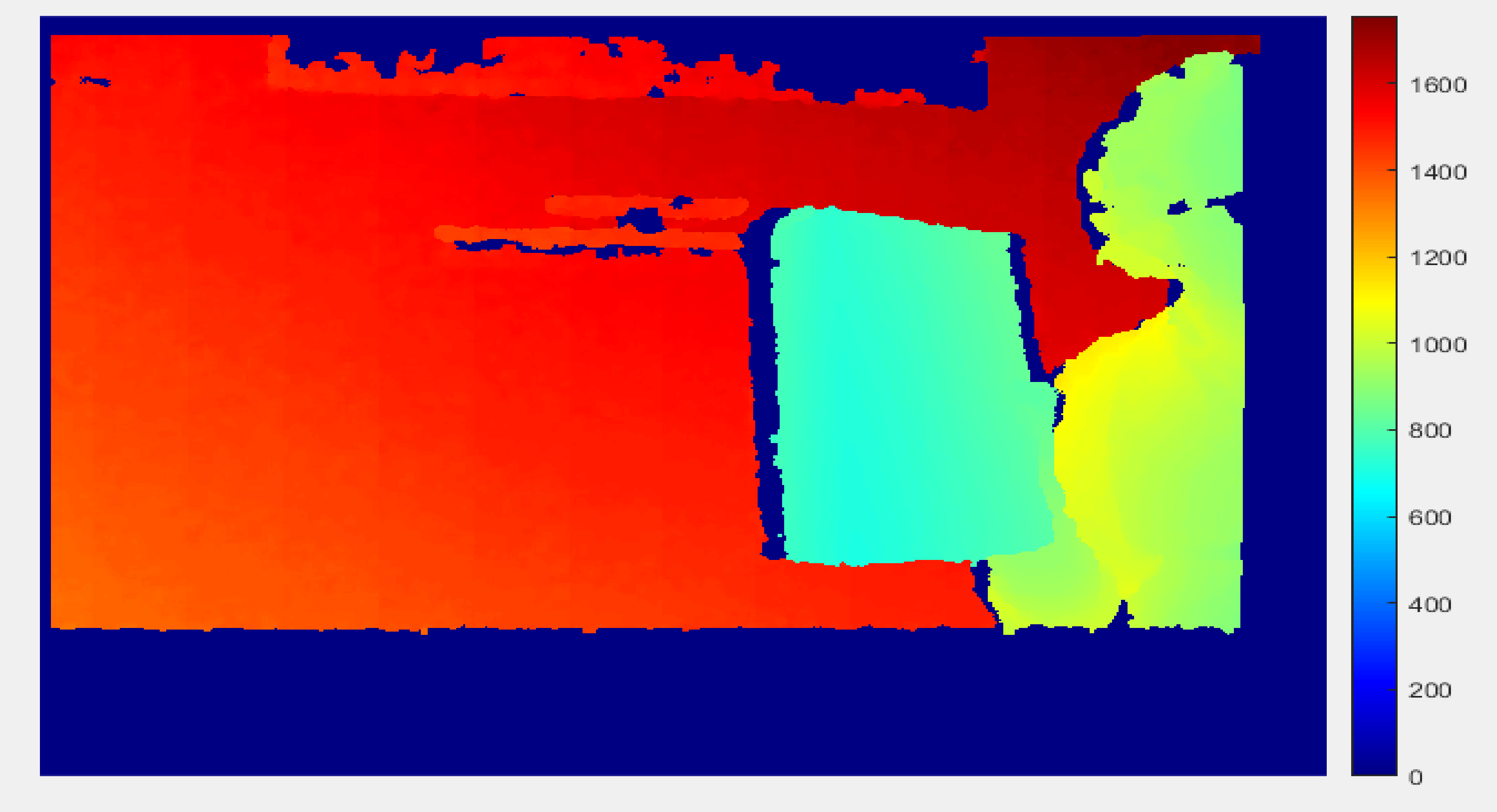

```
bw = imbinarize(I);
stats = regionprops('table',bw,'Centroid',  'MajorAxisLength','MinorAxisLength', 'Area', 'Perimeter',
'BoundingBox');
```

Image Analysis



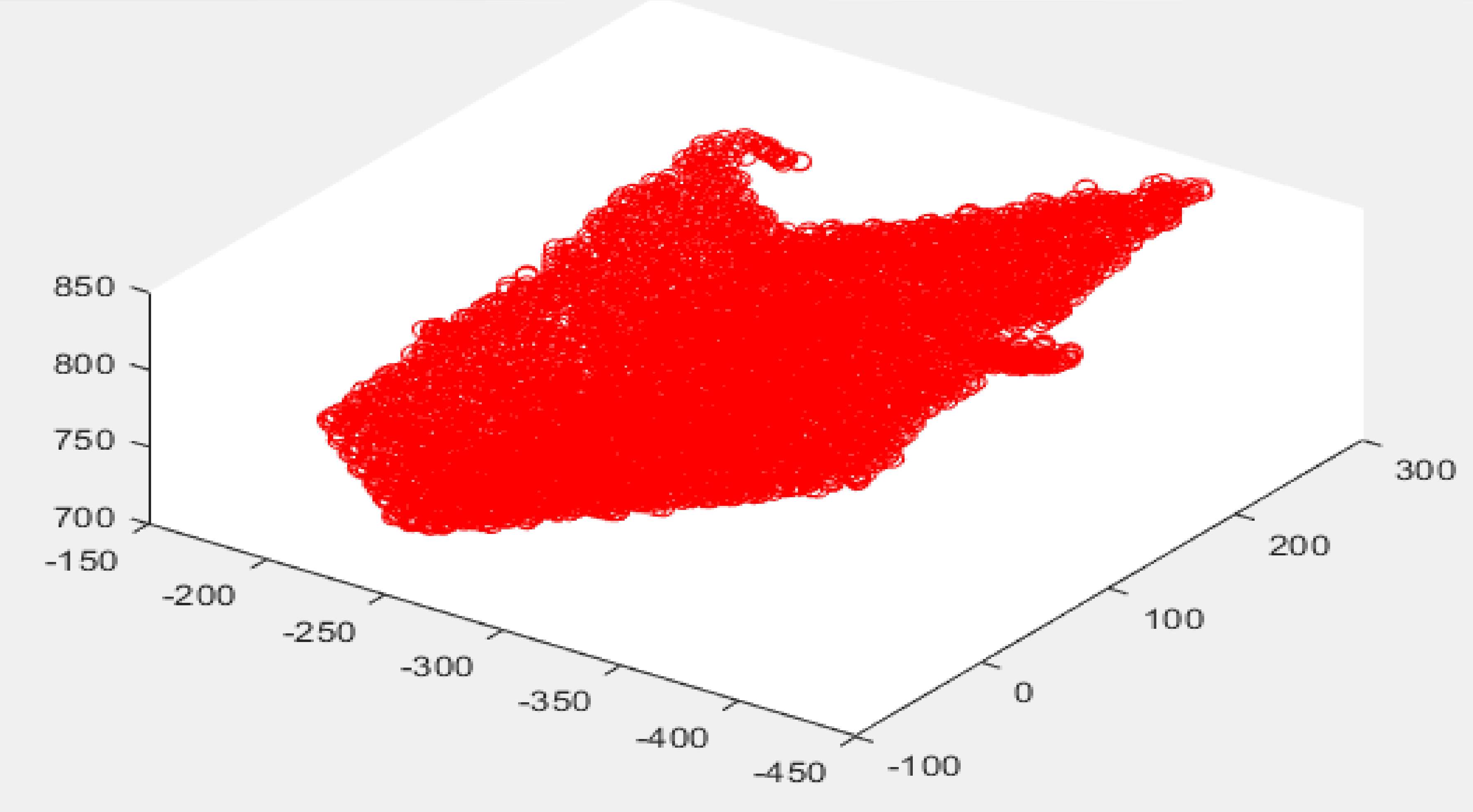

```
bw = imbinarize(I);
stats = regionprops('table',bw,'Centroid',  'MajorAxisLength','MinorAxisLength', 'Area', 'Perimeter', 'BoundingBox');
```

3D Analysis – Depth image

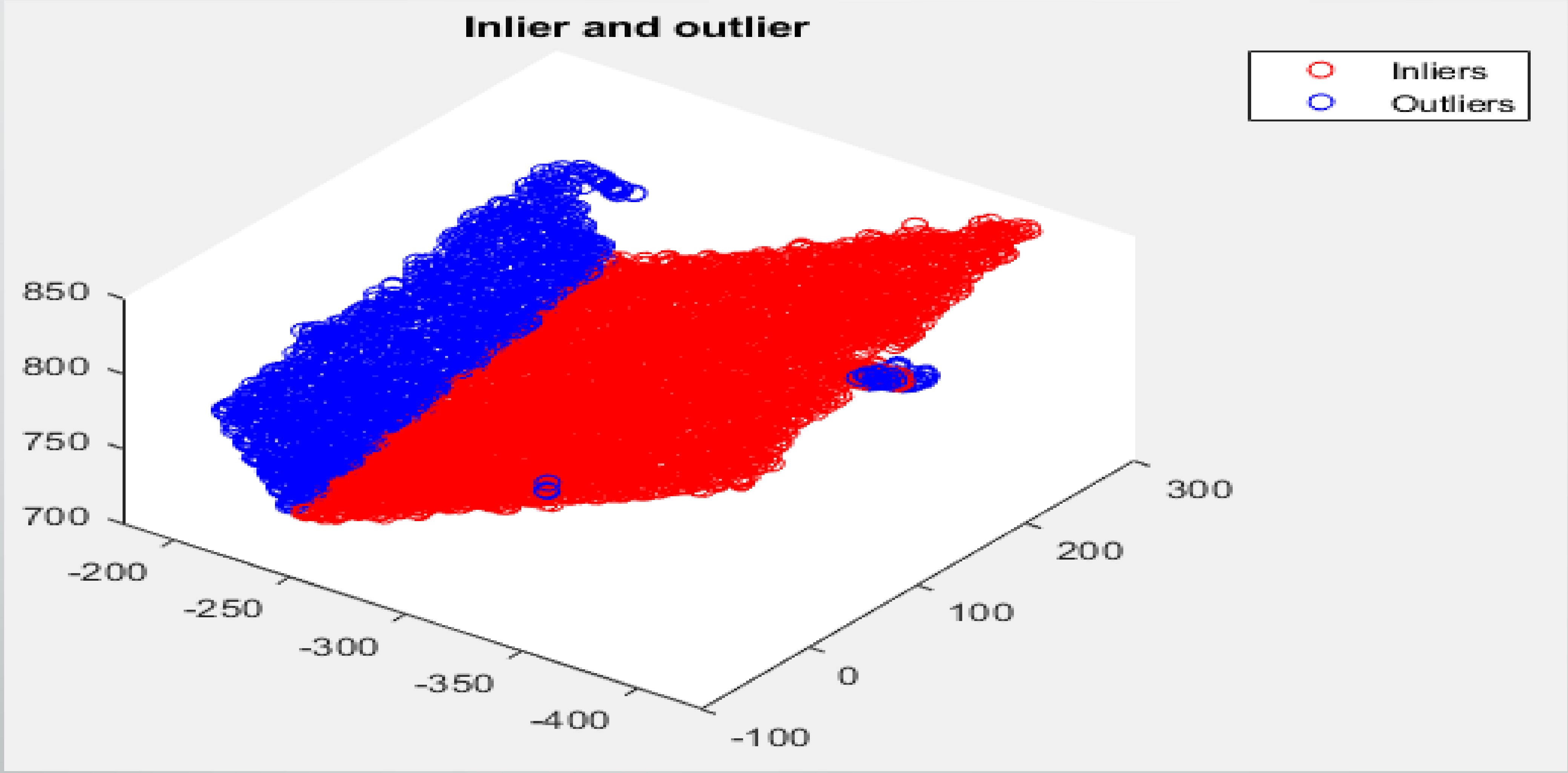



```
imshow(depth,[min(depth,[],'all'),max(depth,[],'all')]);
colormap jet; colorbar; colormap;
```

3D Analysis – Cloud Points

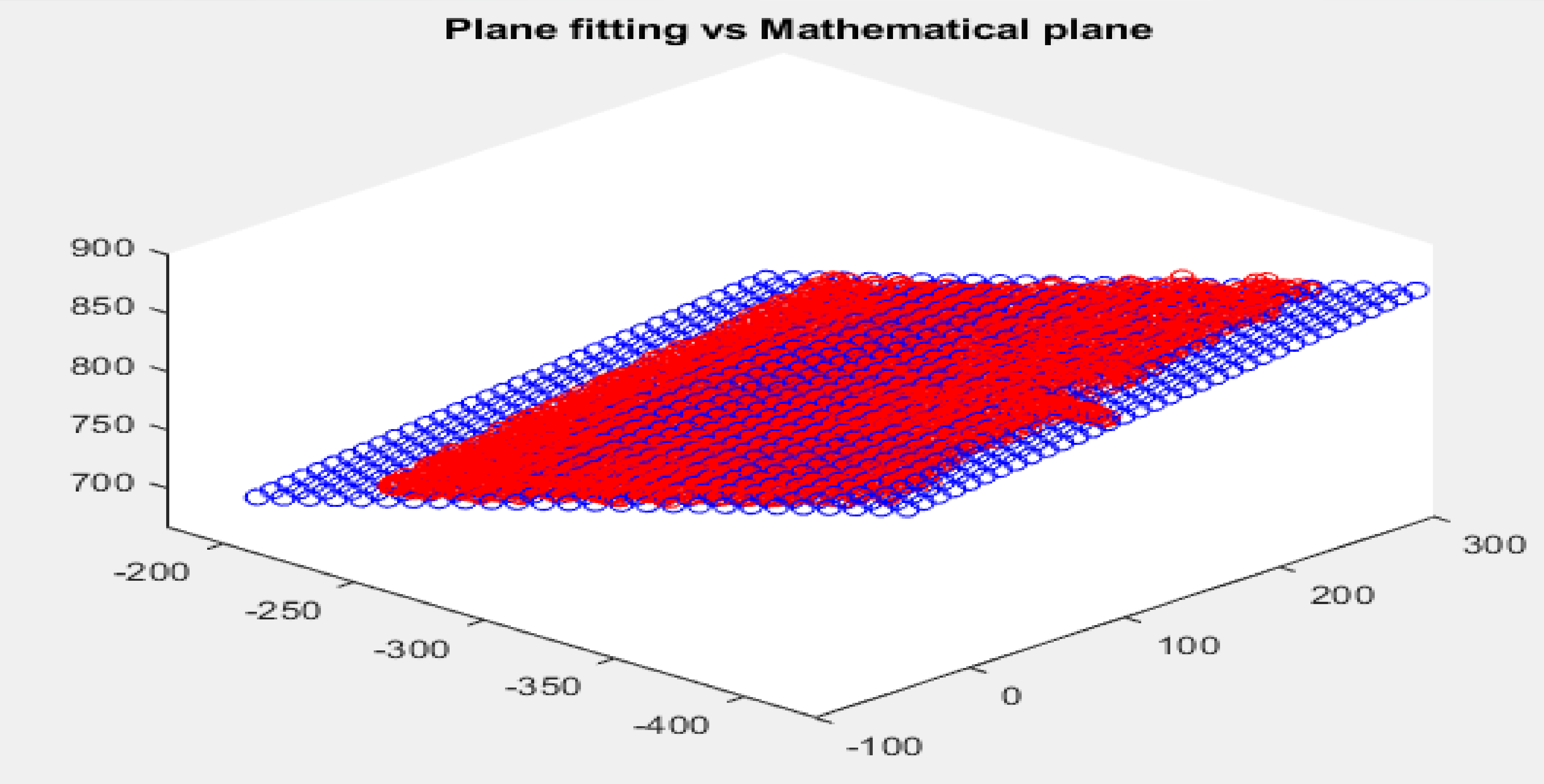


3D Analysis



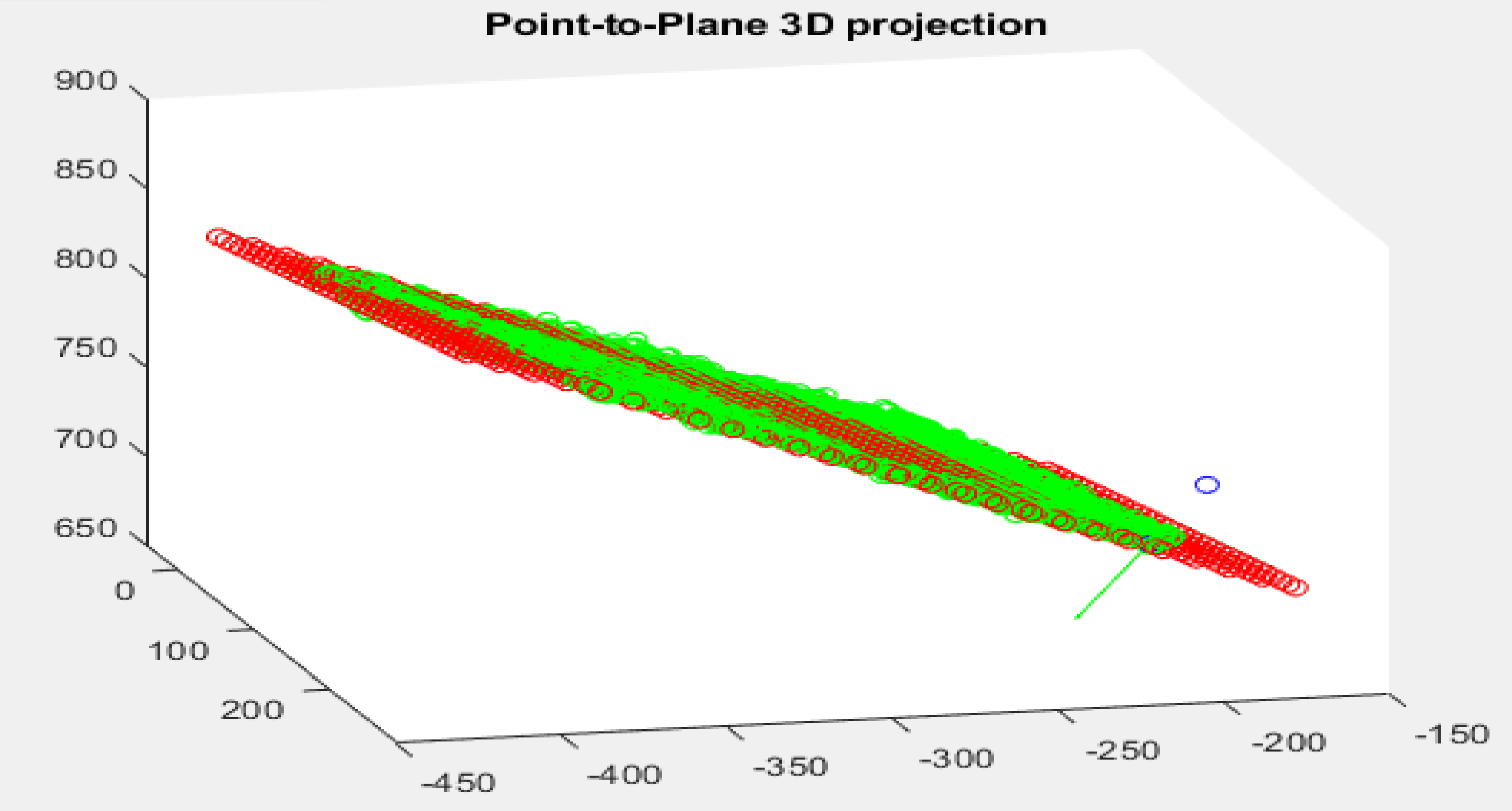
[cloudPoints_fitted, outlier] = planeFitting(cloudPoints, a,b, c,d);

3D Analysis



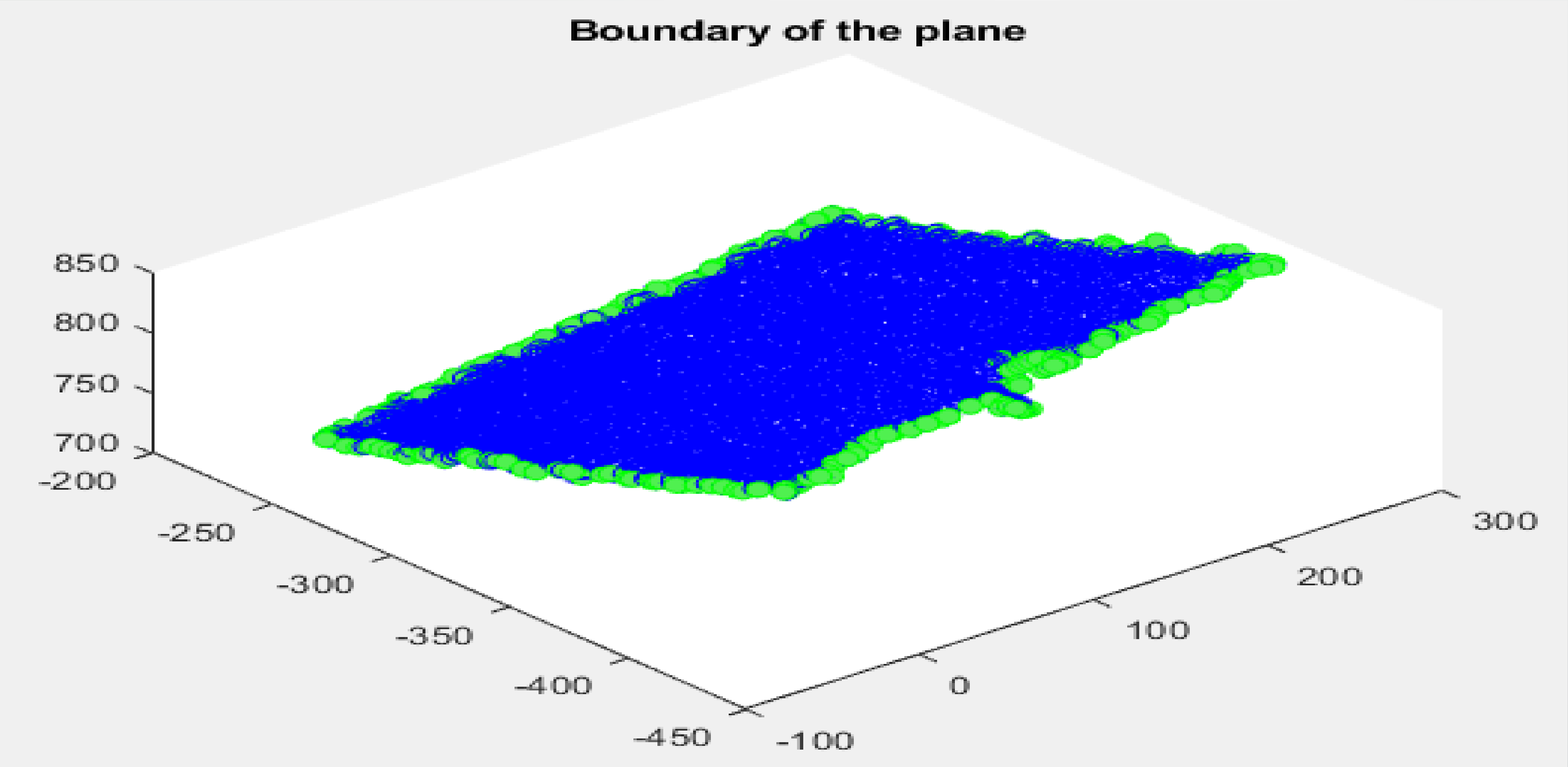
plane = pcfitplane(cloudPoint, 5);

3D Analysis



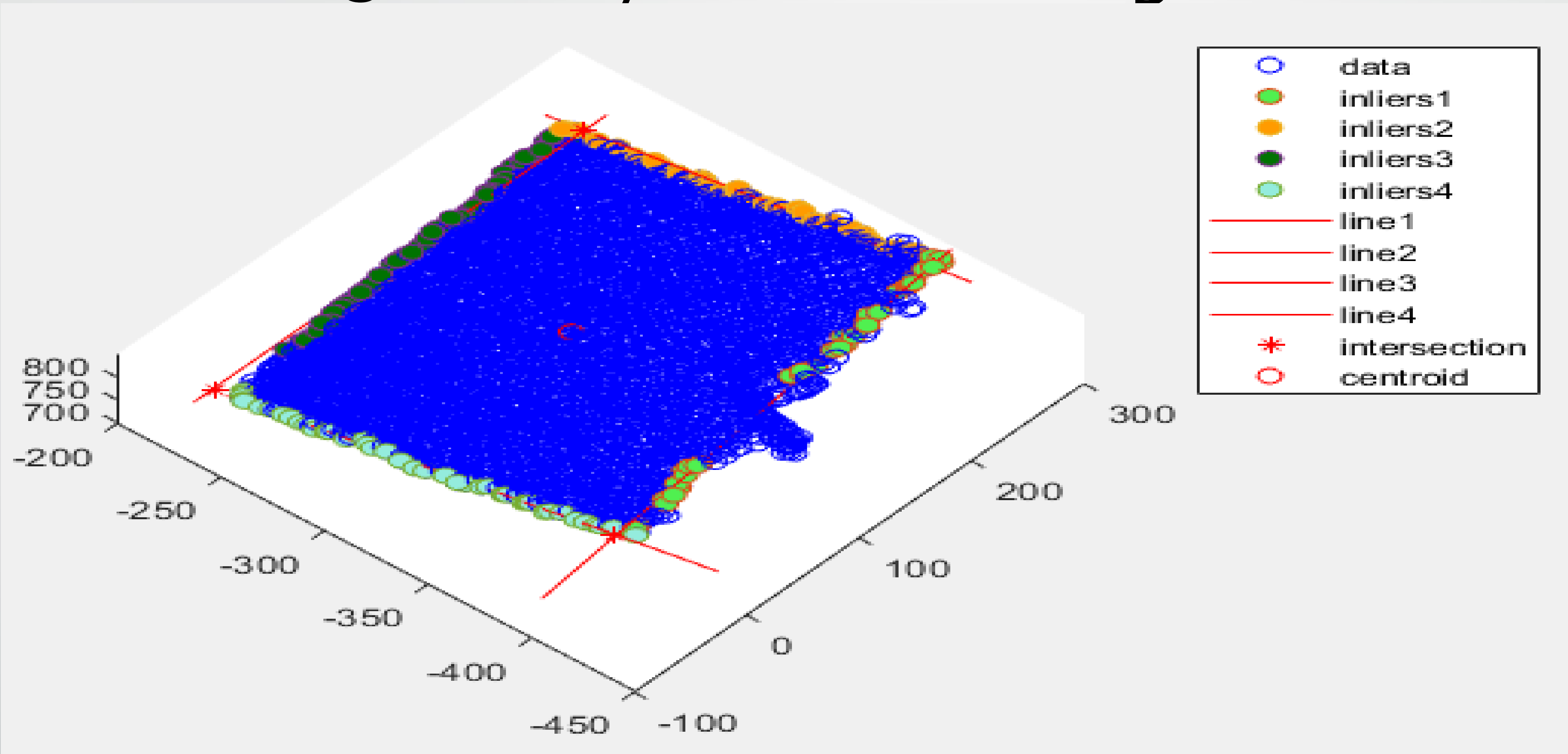
- [projection] = point2plane_projection(normal, P1, Pplane);

3D Analysis




```
boundIndex = (boundary(cloudPoints_fitted(:,1), cloudPoints_fitted(:,2), 1));
boundPoints = cloudPoints_fitted(boundIndex,:);
```

3D Analysis – line fitting



[bestFit, outliers, inliers] = Ransac(data, k, t,d);
[vertices] = findLineIntersection(bestFit1, bestFit2, bestFit3, bestFit4);
[Theta] = angle2lines(line_intersection,L1, L2);
[line_intersection] = lineIntersection(PA, PB, lines);

Thanks for your attention!