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

Robotics, Vision and Control

Report on the assignments given during the 2021/2022 a.y.

Author: Lorenzo Busellato, VR472249
email: lorenzo.busellato_02@studenti.univr.it

UNIVERSITÀ di VERONA
Dipartimento di INFORMATICA

Contents

Chapter 1

Robotics

1.1 Assignment 1

1.1.1 Implement in MATLAB 3rd-, 5th-, 7th-order polynomials for $q_i > q_f$ and $q_i < q_f$ and for $t \in [t_i, t_f]$ and $t \in [0, \Delta T]$

All polynomial trajectories can be expressed as:

$$
\begin{aligned}
q(t) &= a_7(t-t_i)^7 + a_6(t-t_i)^6 + a_5(t-t_i)^5 + a_4(t-t_i)^4 + a_3(t-t_i)^3 + a_2(t-t_i)^2 + a_1(t-t_i) + a_0 \\
\dot{q}(t) &= 7a_7(t-t_i)^6 + 6a_6(t-t_i)^5 + 5a_5(t-t_i)^4 + 4a_4(t-t_i)^3 + 3a_3(t-t_i)^2 + 2a_2(t-t_i) + a_1 \\
\ddot{q}(t) &= 42a_7(t-t_i)^5 + 30a_6(t-t_i)^4 + 20a_5(t-t_i)^3 + 12a_4(t-t_i)^2 + 6a_3(t-t_i) + 2a_2 \\
\dddot{q}(t) &= 210a_7(t-t_i)^4 + 120a_6(t-t_i)^3 + 60a_5(t-t_i)^2 + 24a_4(t-t_i) + 6a_3 \\
\ddddot{q}(t) &= 840a_7(t-t_i)^3 + 360a_6(t-t_i)^2 + 120a_5(t-t_i) + 24a_4
\end{aligned}
$$

For the 3rd-order polynomial $a_7 = a_6 = a_5 = a_4 = 0$ while for the 5th-order polynomial $a_7 = a_6 = 0$.

The problem of determining the a_i coefficients of the polynomials is solved by setting up a system of equations using initial and final conditions on velocity (3rd-order), velocity and acceleration (5th-order), velocity, acceleration and jerk (7th-order).

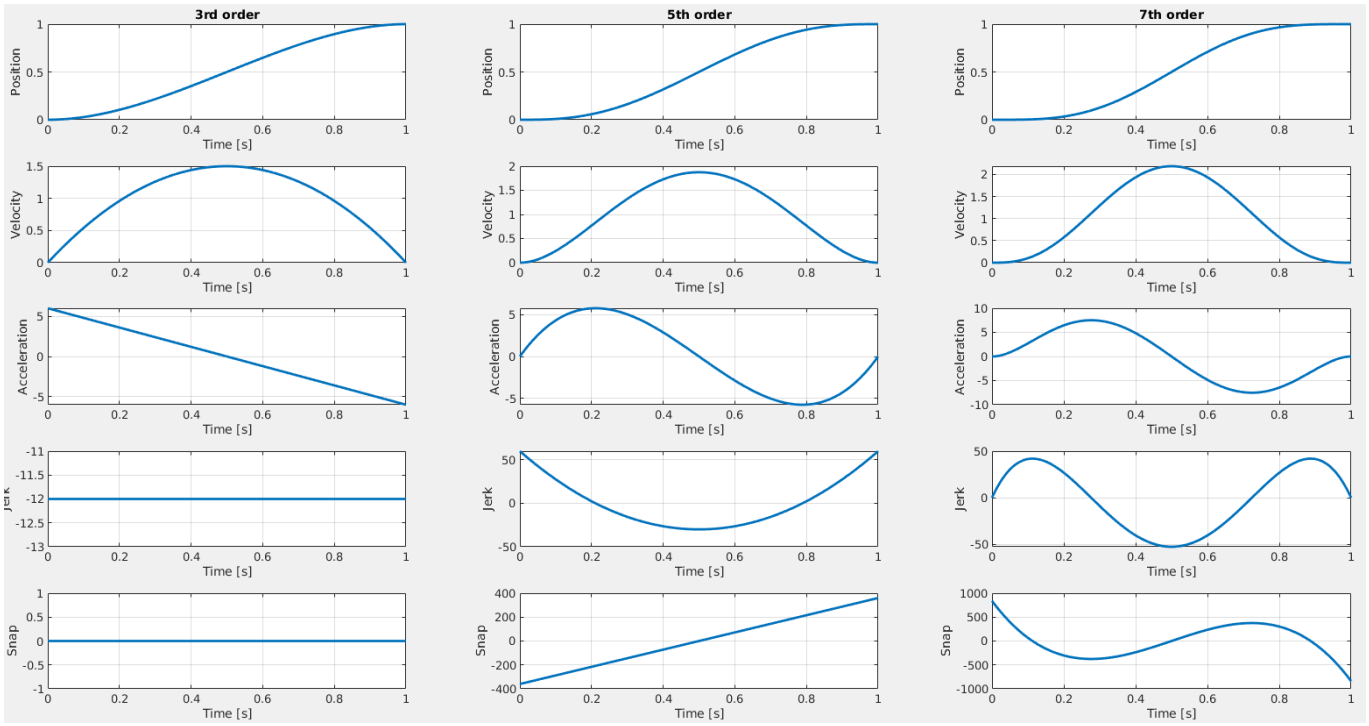


Figure 1.1: 3rd-, 5th-, 7th-order polynomial trajectories with $q_i < q_f$ and $t \in [0, \Delta T]$, $q_i = 0, q_f = 1, \Delta T = 1, v_i = v_f = a_i = a_f = j_i = j_f = 0$.

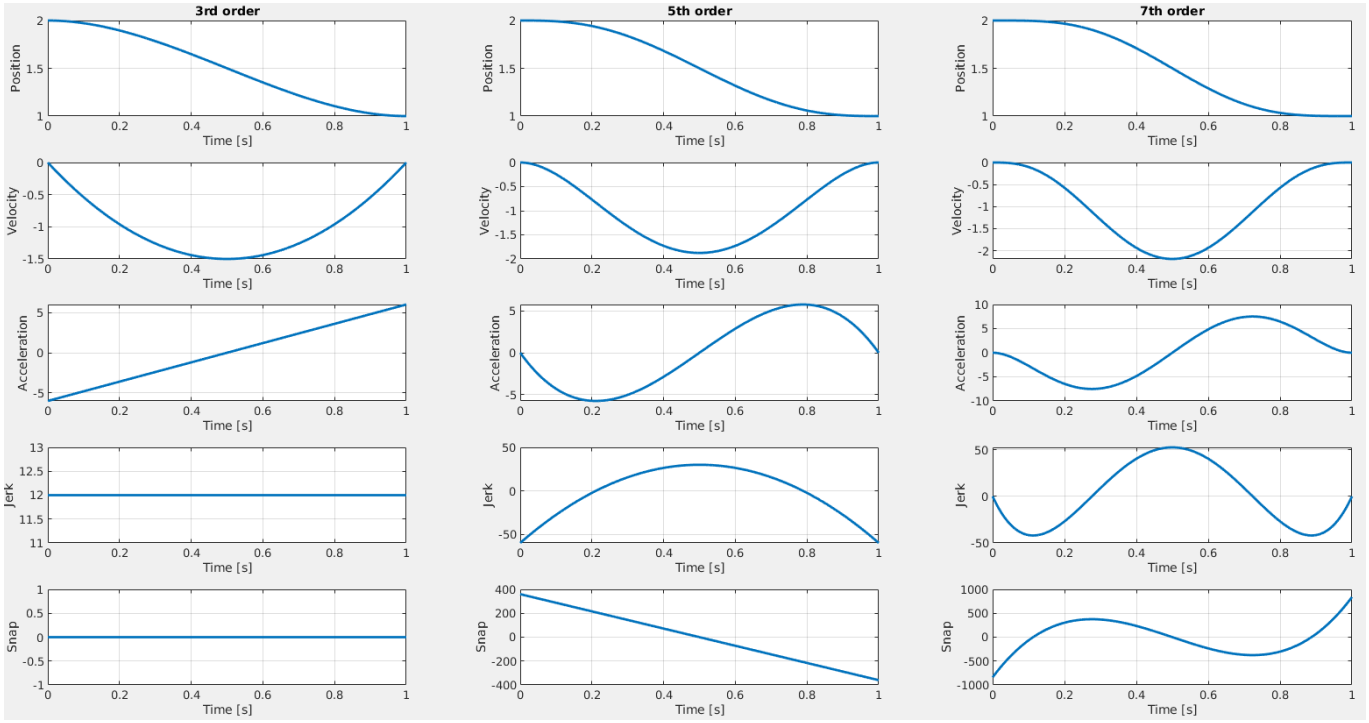


Figure 1.2: 3rd-, 5th-, 7th-order polynomial trajectories with $q_i > q_f$ and $t \in [0, \Delta T]$, $q_i = 2, q_f = 1, \Delta T = 1, v_i = v_f = a_i = a_f = j_i = j_f = 0$.

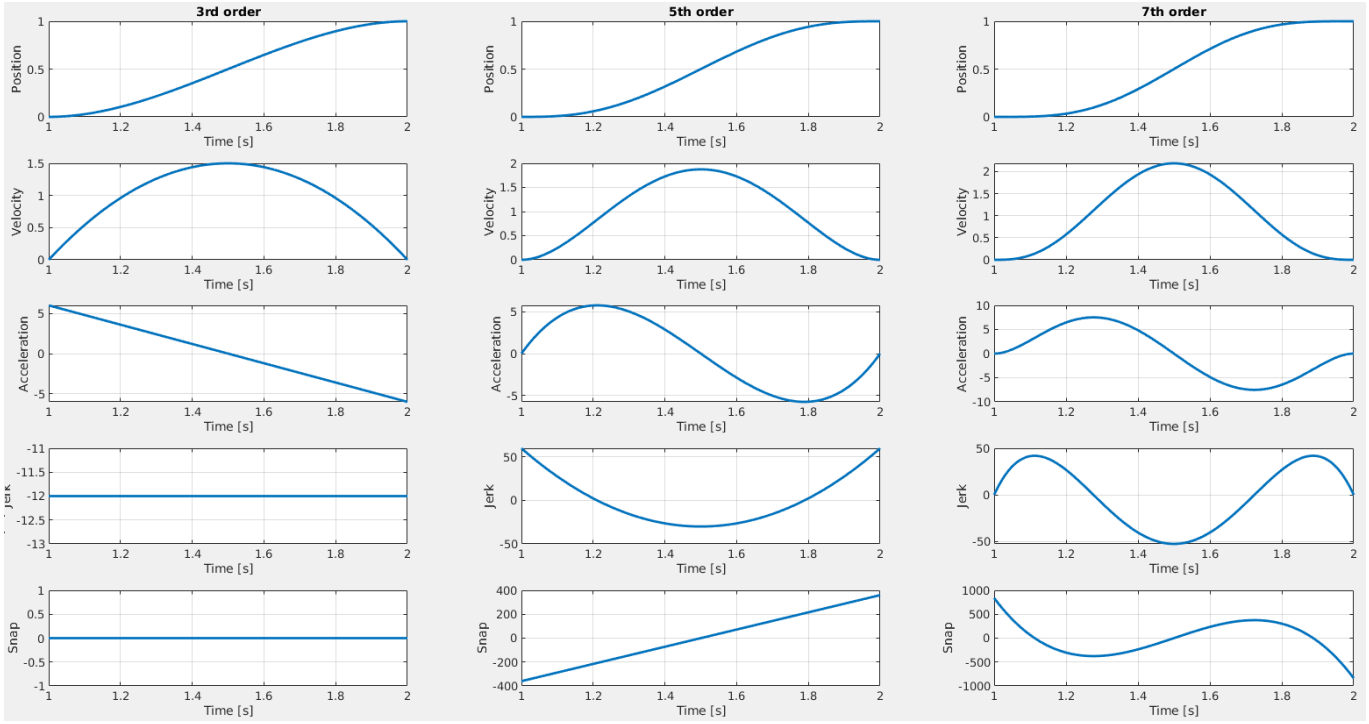


Figure 1.3: 3rd-, 5th-, 7th-order polynomial trajectories with $q_i < q_f$ and $t \in [t_i, t_f]$, $q_i = 0, q_f = 1, t_i = 1, t_f = 2, v_i = v_f = a_i = a_f = j_i = j_f = 0$.

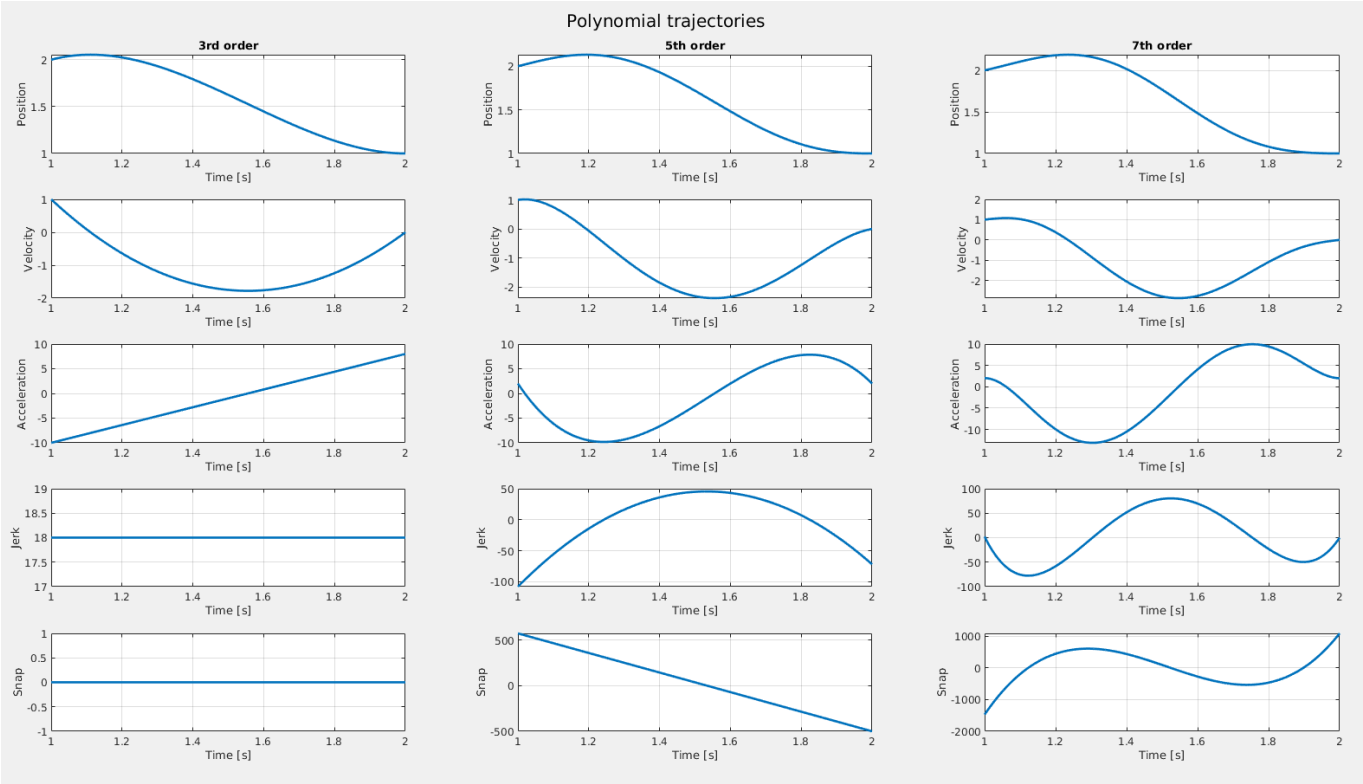


Figure 1.4: 3rd-, 5th-, 7th-order polynomial trajectories with $q_i > q_f$ and $t \in [t_i, t_f]$, $q_i = 2, q_f = 1, t_i = 1, t_f = 2, v_i = 1, v_f = 0, a_i = 2, a_f = 2, j_i = 2, j_f = 0$.

1.2 Assignment 2

1.2.1 Implement in MATLAB the trapezoidal trajectory taking into account the different constraints.

The general expression for a trapezoidal velocity profile is:

$$q(t) = \begin{cases} q_i + \dot{q}_i(t-t_i) + \frac{\dot{q}_c - \dot{q}_i}{2t_a}(t-t_i)^2 & t_i \le t \le t_a + t_i \\ q_i + \dot{q}_i\frac{t_a}{2} + \dot{q}_c(t - t_i - \frac{t_a}{2})^2 & t_a + t_i \le t \le t_f - t_d \\ q_f - \dot{q}_f(t_f - t) - \frac{\dot{q}_c - \dot{q}_f}{2t_d}(t_f - t)^2 & t_f - t_d \le t \le t_f \end{cases}$$

If the initial and final velocities are null, then $t_a = t_d = t_c$.
If $\dot{q}_i = \dot{q}_f = 0$ then the possible constraints are:

- t_c:

$$\ddot{q}_c = \frac{q_f - q_i}{t_c t_f - t_c^2} \quad \dot{q}_c = \ddot{q}_c t_c$$

- $\ddot{q}_c$:

$$t_c = \frac{t_f}{2} - \frac{1}{2}\sqrt{\frac{t_f^2\ddot{q}_c - 4(q_f - q_i)}{\ddot{q}_c}} \quad \dot{q}_c = \ddot{q}_c t_c$$

- $\dot{q}_c$:

$$t_c = \frac{q_i - q_f + \dot{q}_c t_f}{\dot{q}_c} \quad \ddot{q}_c = \frac{\dot{q}_c^2}{q_i - q_f + \dot{q}_c t_f}$$

- $\ddot{q}_c, \dot{q}_c$:

$$t_c = \frac{\dot{q}_c}{\ddot{q}_c} \quad t_f = \frac{\dot{q}_c^2 + \ddot{q}_c(q_f - q_i)}{\dot{q}_c\ddot{q}_c}$$

In all cases the feasibility condition is that $2t_c \le t_f - t_i$.
If $\dot{q}_i, \dot{q}_f \neq 0$ the possible constraints are:

- $\ddot{q}_{c,max}$:

$$\dot{q}_c = \frac{1}{2}(\dot{q}_i + \dot{q}_f + \ddot{q}_{c,max}\Delta T + \sqrt{\ddot{q}_{c,max}^2\Delta T^2 - 4\ddot{q}_{c,max}\Delta q + 2\ddot{q}_{c,max}(\dot{q}_i + \dot{q}_f)\Delta T - (\dot{q}_i - \dot{q}_f)^2} \quad t_a = \frac{\dot{q}_c - \dot{q}_i}{\ddot{q}_{c,max}}\ t_d = \frac{\dot{q}_c - \dot{q}_f}{\ddot{q}_{c,max}}$$

The trajectory is feasible when the argument of the square root is positive, and when the maximum acceleration satisfies:

$$\ddot{q}_{c,max}\Delta q > \frac{\left|\dot{q}_i^2 - \dot{q}_f^2\right|}{2} \quad \ddot{q}_{c,max} \ge \ddot{q}_{c,lim} = \frac{2\Delta q - (\dot{q}_i - \dot{q}_f)\Delta T + \sqrt{4\Delta q^2 - 4\Delta q(\dot{q}_i + \dot{q}_f)\Delta T + 2(\dot{q}_i^2 + \dot{q}_f^2)\Delta T^2}}{\Delta T^2}$$

when $\ddot{q}_{c,max} = \ddot{q}_{c,lim}$ there is no constant velocity phase.

- $\ddot{q}_{max}, \dot{q}_{max}$: First we compute the condition:

$$\ddot{q}_{c,max}\Delta q \gtreqless \dot{q}_{c,max}^2 - \frac{\dot{q}_i^2 + \dot{q}_f^2}{2}$$

If the above is $>$ then:

$$\dot{q}_c = \dot{q}_{c,max} \quad t_a = \frac{\dot{q}_{c,max} - \dot{q}_i}{\ddot{q}_{c,max}} \quad t_d = \frac{\dot{q}_{c,max} - \dot{q}_f}{\ddot{q}_{c,max}} \quad \Delta T = \frac{\Delta q\ddot{q}_{c,max} + \dot{q}_{c,max}^2}{\ddot{q}_{c,max}\dot{q}_{c,max}}$$

If the above is $\le$ then:

$$\dot{q}_c = \dot{q}_{c,lim} = \sqrt{\ddot{q}_{c,max}\Delta q + \frac{\dot{q}_i^2 + \dot{q}_f^2}{2}} < \dot{q}_{c,max} \quad t_a = \frac{\dot{q}_{c,lim} - \dot{q}_i}{\ddot{q}_{c,max}} \quad t_d = \frac{\dot{q}_{c,lim} - \dot{q}_f}{\ddot{q}_{c,max}}$$

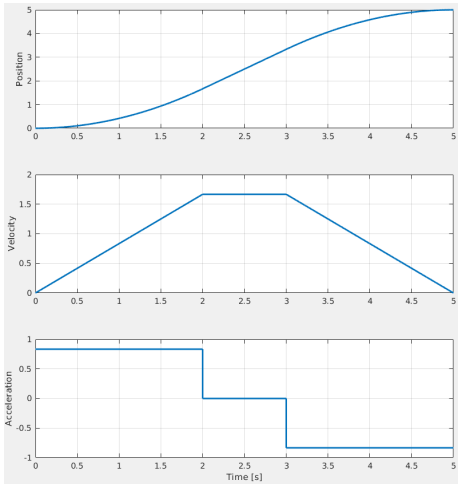


Figure 1.5: $t_c = 2s$

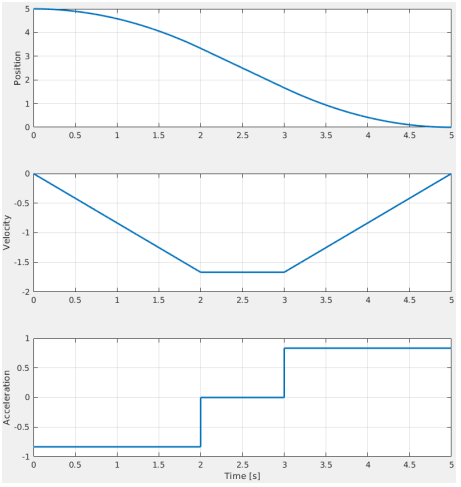


Figure 1.6: $t_c = 2s, q_i > q_f$

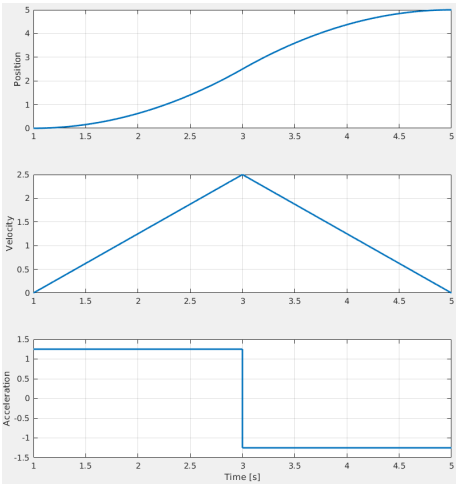


Figure 1.7: $t_c = 2s, t_i \neq 0$

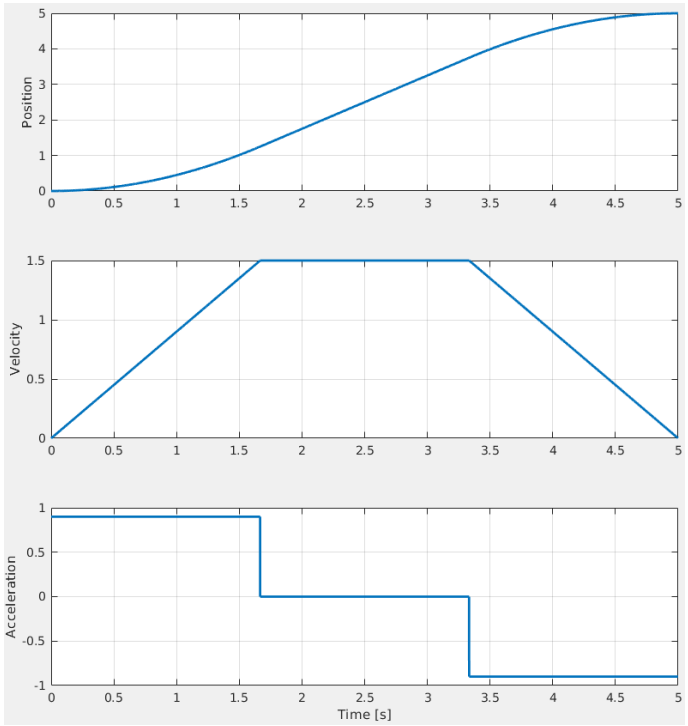


Figure 1.8: $v_c = 1.5$

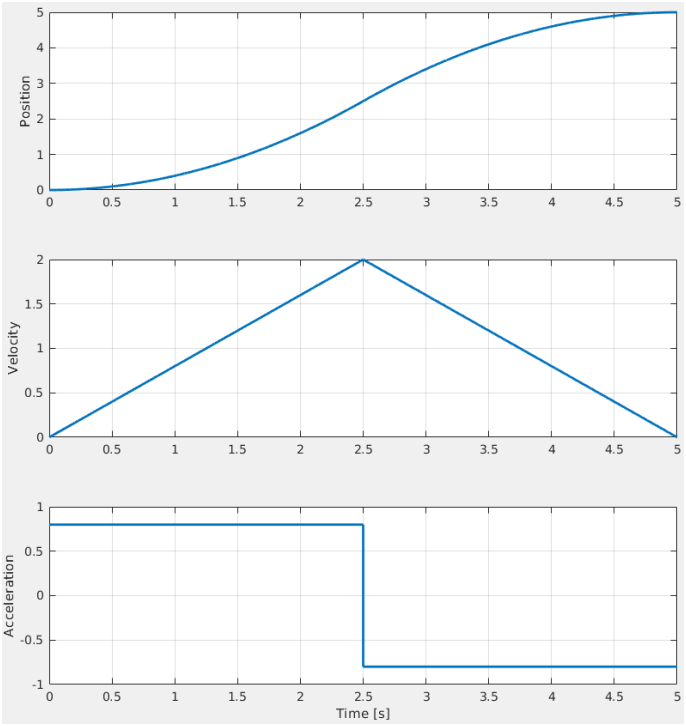


Figure 1.9: $v_c = v_{c,lim} = 2$

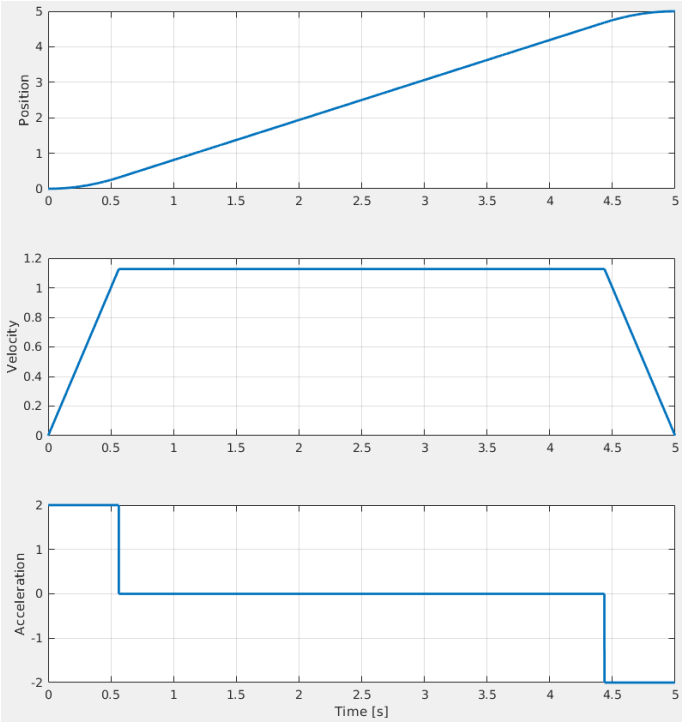


Figure 1.10: $a_c = 2$

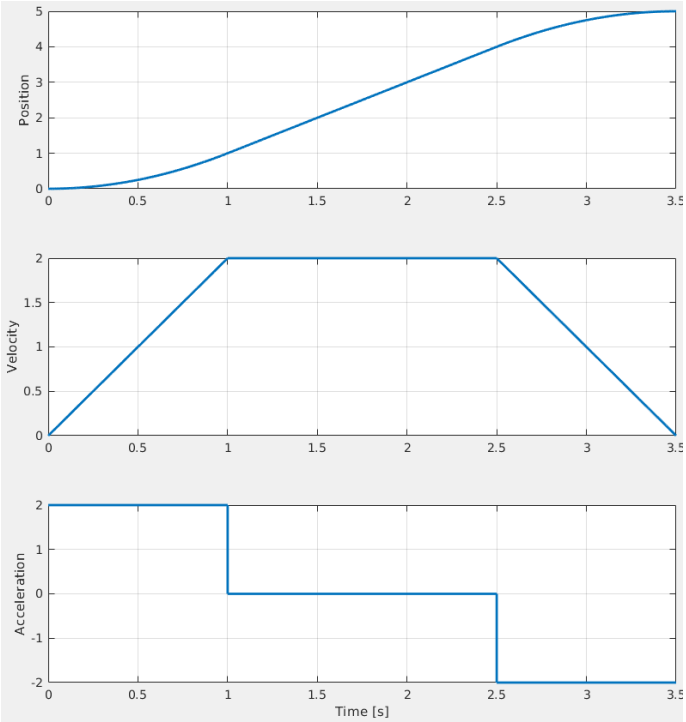


Figure 1.11: $a_c, v_c = 2$

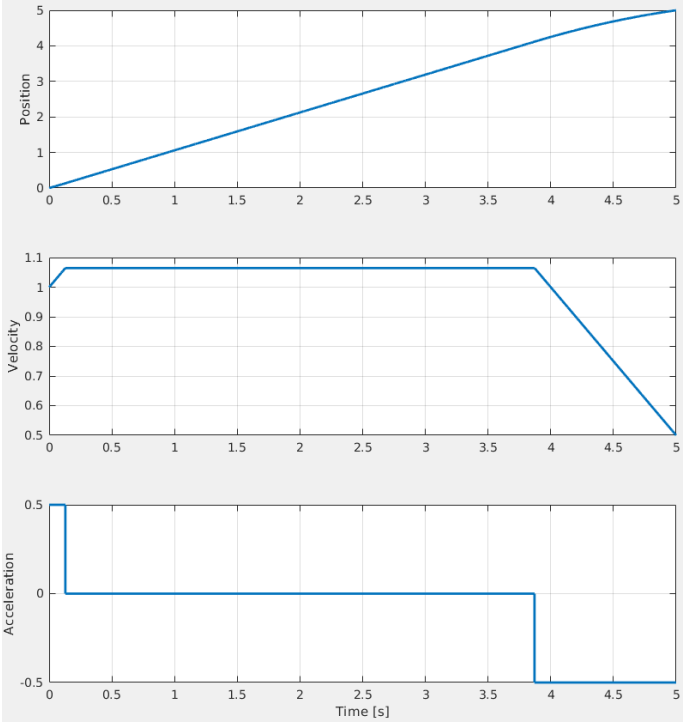


Figure 1.12: $\dot{q}_i = 1, \dot{q}_f = 0.5, \ddot{q}_{max} = 0.5$

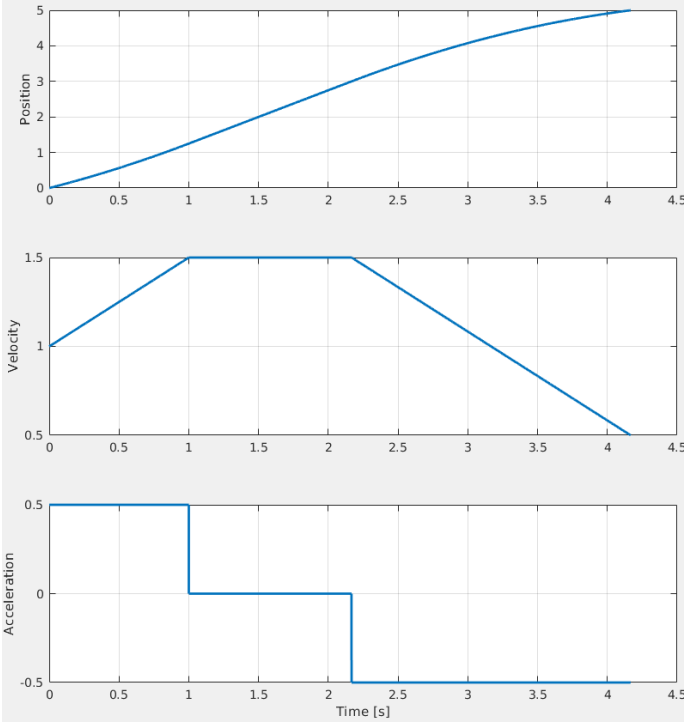


Figure 1.13: $\dot{q}_i = 1, \dot{q}_f = 0.5, \dot{q}_{max} = 1.5, \ddot{q}_{max} = 0.5$

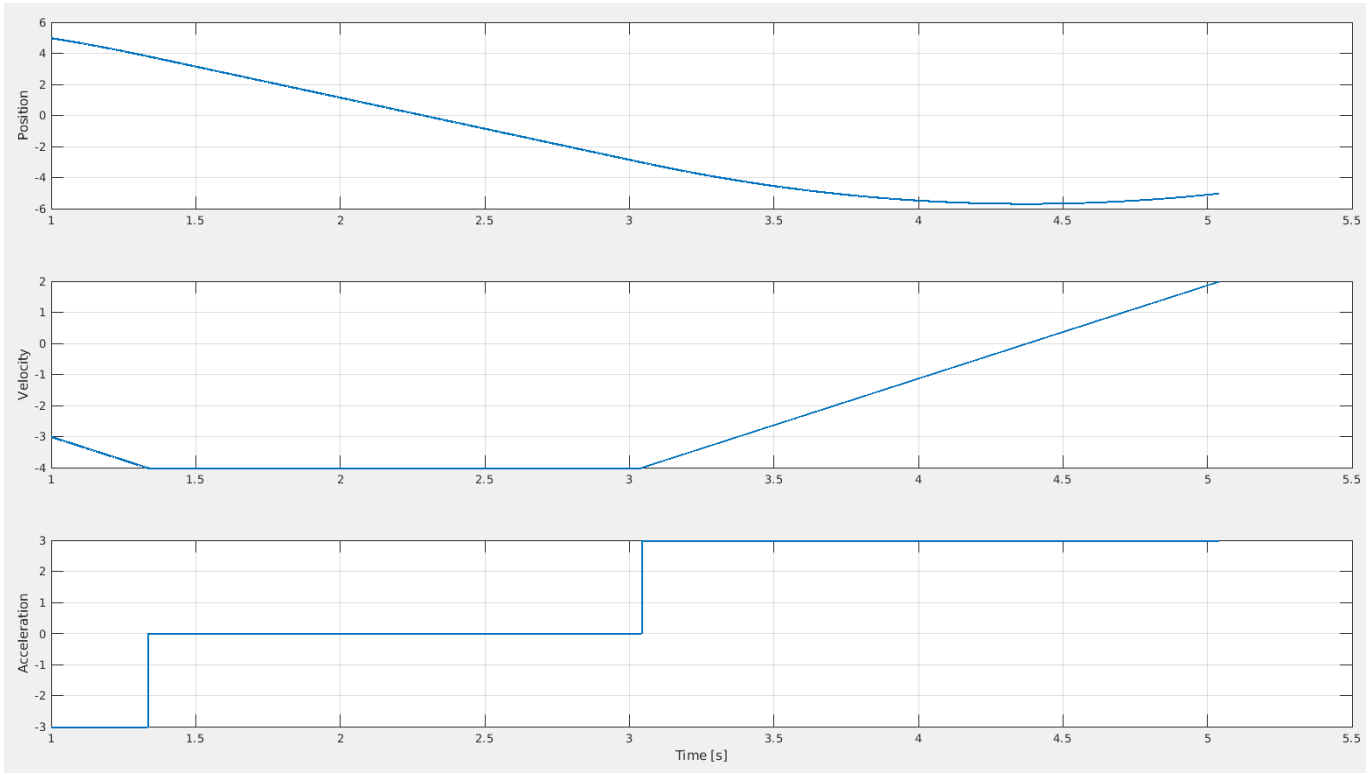


Figure 1.14: $q_i = 5, q_f = -5, \dot{q}_i = -3, \dot{q}_f = 2, \dot{q}_{max} = 4, \ddot{q}_{max} = 3$

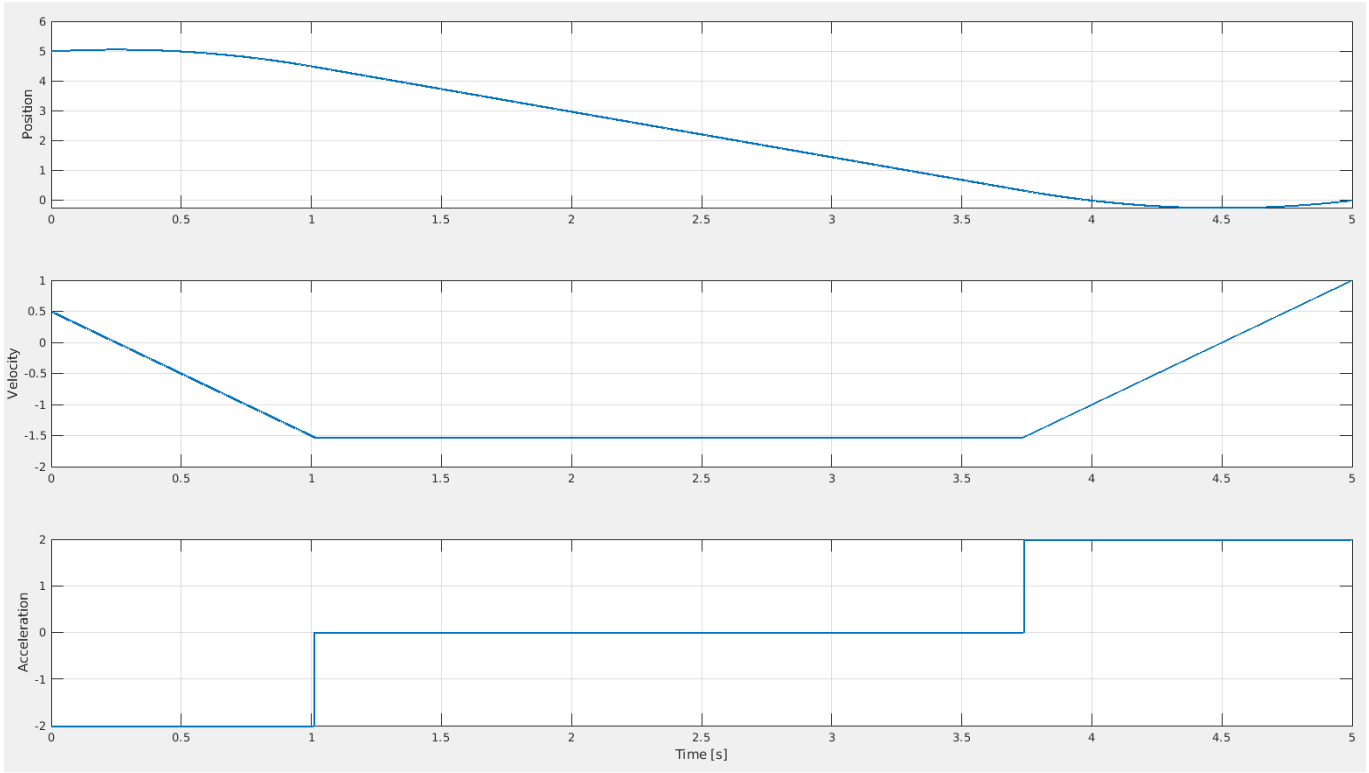


Figure 1.15: $q_i = 5, q_f = 0, \dot{q}_i = 0.5, \dot{q}_f = 1, \ddot{q}_{max} = 2$

To generate a multipoint trajectory we simply repeat the computation for each consecutive point pair. To improve the resulting velocity and acceleration profiles we can apply an heuristic to assign velocities to each waypoint, so that a less jagged profile is obtained:

$$
\begin{aligned}
&\dot{q}(t_i) = \dot{q}_i \\
&\dot{q}(t_k) = \begin{cases} 0 & \text{if } sign(\Delta Q_k) \neq sign(\Delta Q_{k+1}) \\ sign(\Delta Q_k)\dot{q}_{max} & \text{if } sign(\Delta Q_k) = sign(\Delta Q_{k+1}) \end{cases} \\
&\dot{q}(t_f) = \dot{q}_f
\end{aligned}
$$

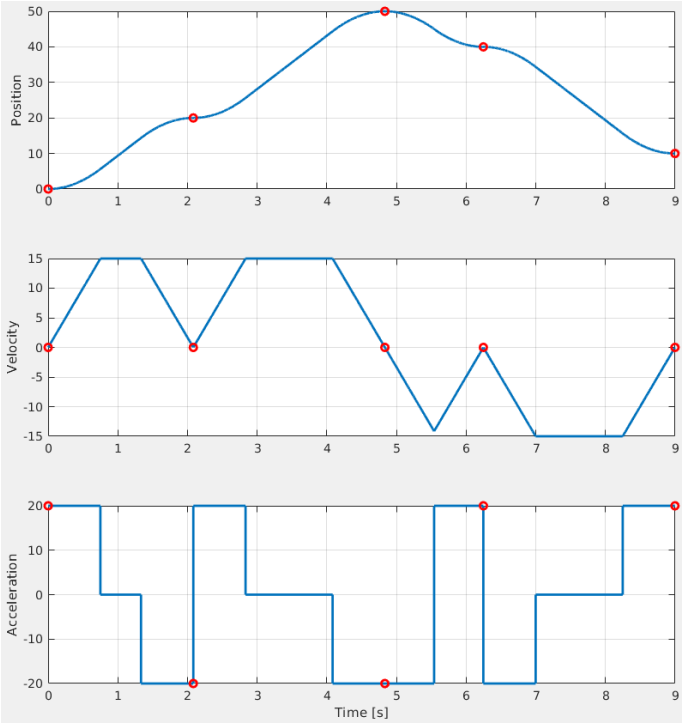


Figure 1.16: Multipoint trajectory without velocity heuristic

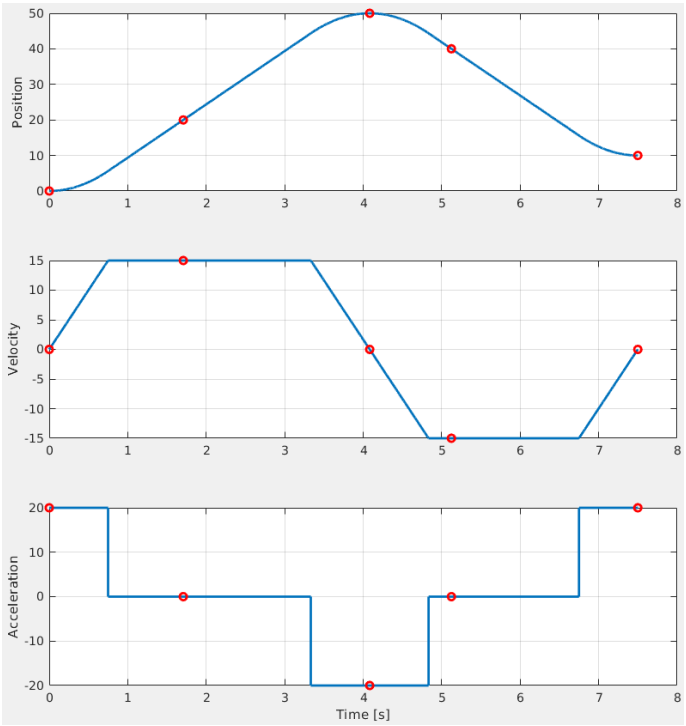


Figure 1.17: Multipoint trajectory with velocity heuristic

1.3 Assignment 3

1.3.1 Interpolating polynomials with computed velocities at path points and imposed velocity at initial/final points.

To implement multipoint trajectories we concatenate cubic splines. The interpolating trajectory is then:

$$q(t) \coloneqq \{\Pi_k(t), t \in [t_k, t_{k+1}], k = 0, \dots, n-1\}$$

where:

$$\Pi(t) = a_3^k(t-t_k)^3 + a_2^k(t-t_k)^2 + a_1^k(t-t_k) + a_0^k$$

Fixing velocities on all path points yields:

$$\begin{cases} a_0^k &= q_k \\ a_1^k &= \dot{q}_k \\ a_2^k &= \frac{1}{T_k}\left(\frac{3(q_{k+1}-q_k)}{T_k} - 2\dot{q}_k - \dot{q}_{k+1}\right) \\ a_3^k &= \frac{1}{T_k^2}\left(\frac{2(q_k-q_{k+1})}{T_k} + \dot{q}_k + \dot{q}_{k+1}\right) \end{cases}$$

where $T_k = t_{k+1} - t_k$.

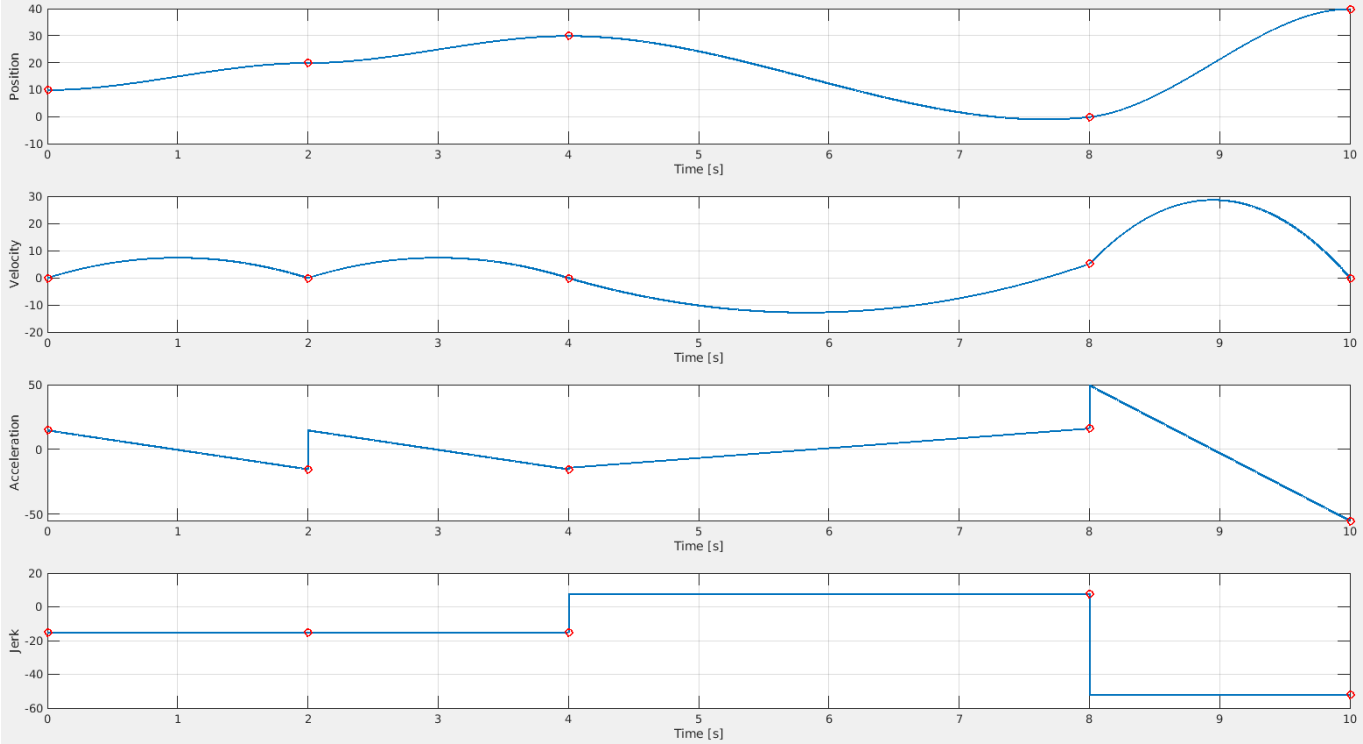


Figure 1.18: Trajectory through $q_k = \begin{bmatrix}10 & 20 & 30 & 0 & 40\end{bmatrix}$ at times $t_k = \begin{bmatrix}0 & 2 & 4 & 8 & 10\end{bmatrix}$ with velocities $\dot{q}_k = \begin{bmatrix}0 & 0 & 0 & 5.2 & 0\end{bmatrix}$

Path point velocities are not generally known. We can estimate them with Euler's approximation:

$$v_k = \frac{q_k - q_{k-1}}{t_k - t_{k-1}} \implies \begin{cases} \dot{q}(t_0) &= \dot{q}_0 \\ \dot{q}(t_k) &= \begin{cases} 0 & \text{if } sign(v_k) \neq sign(v_{k+1}) \\ \frac{v_k + v_{k+1}}{2} & \text{if } sign(v_k) = sign(v_{k+1}) \end{cases} \\ \dot{q}(t_n) &= \dot{q}_n \end{cases}$$

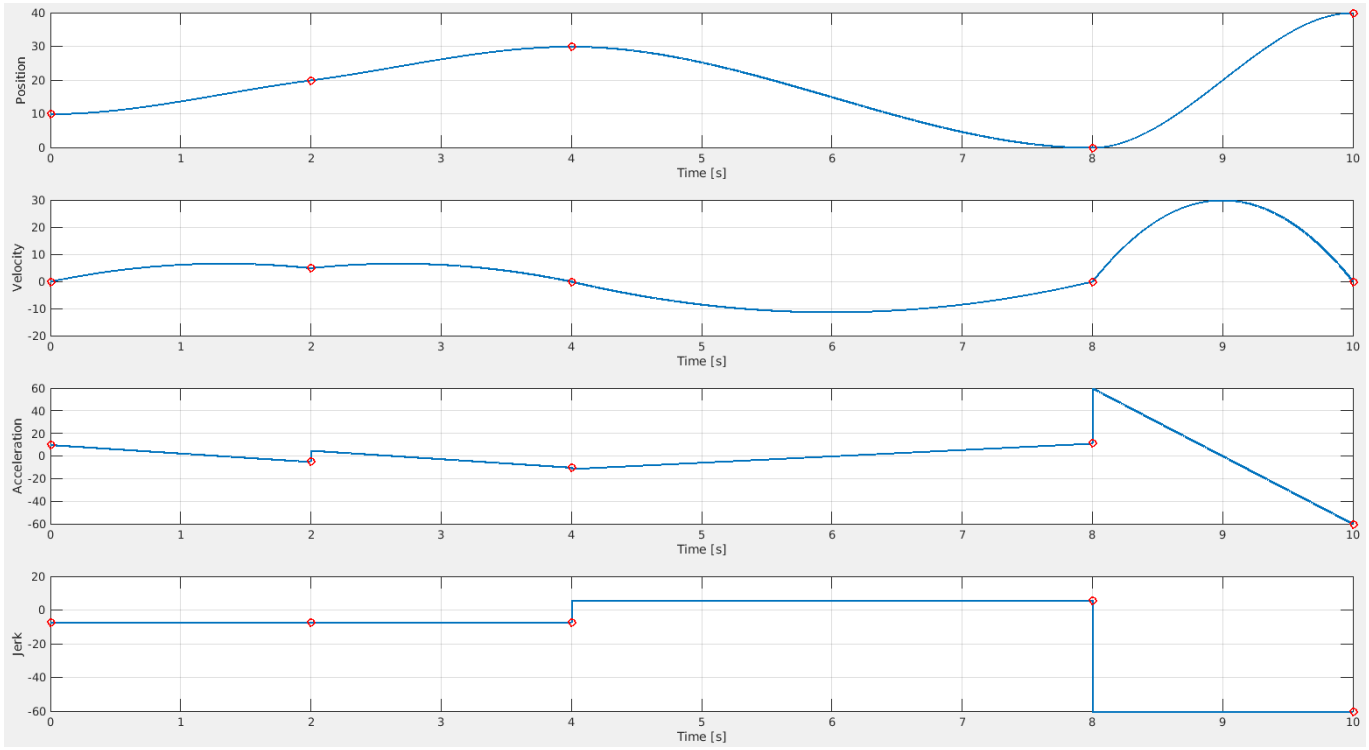


Figure 1.19: Trajectory interpolation with Euler's approximation.

1.3.2 Interpolating polynomials with continuous accelerations at path points and imposed velocity at initial/final points (+ Thomas algorithm)

Imposing velocity and acceleration continuity at path points results in the definition of a linear system $A\dot{q} = c$, where A is a tridiagonal matrix. Thanks to this property the system can be solved for $\dot{q}$ efficiently using Thomas' algorithm. Given:

$$\begin{bmatrix} 2(T_0+T_1) & T_0 & & & & & \\ T_2 & 2(T_1+T_2) & T_1 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & T_{k+1} & 2(T_k+T_{k+1}) & T_k & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & T_{n-1} & 2(T_{n-2}+T_{n-1}) \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_k \\ \vdots \\ \dot{q}_{n-1} \end{bmatrix} = \begin{bmatrix} c_0 - T_1\dot{q}_0 \\ \vdots \\ c_k \\ \vdots \\ c_{n-2} - T_{n-2}\dot{q}_n \end{bmatrix}$$

where $T_k = t_{k+1} - t_k$ and:

$$c_k = 3\frac{T_{k+1}}{T_k}(q_{k+1} - q_k) + 3\frac{T_k}{T_{k+1}}(q_{k+2} - q_{k+1})$$

Thomas' algorithm is as follows:

Forward elimination:

for $k = 2 : 1 : n$ **do**
  $m \leftarrow \frac{a_k}{b_{k-1}}$
  $b_k \leftarrow b_k - mc_{k-1}$
  $d_k \leftarrow d_k - md_{k-1}$
end for

Backward substitution:

$x_n \leftarrow \frac{d_n}{b_n}$
for $k = 2 : 1 : n$ **do**
  $x_k \leftarrow \frac{d_k - c_k x_{k+1}}{b_k}$
end for

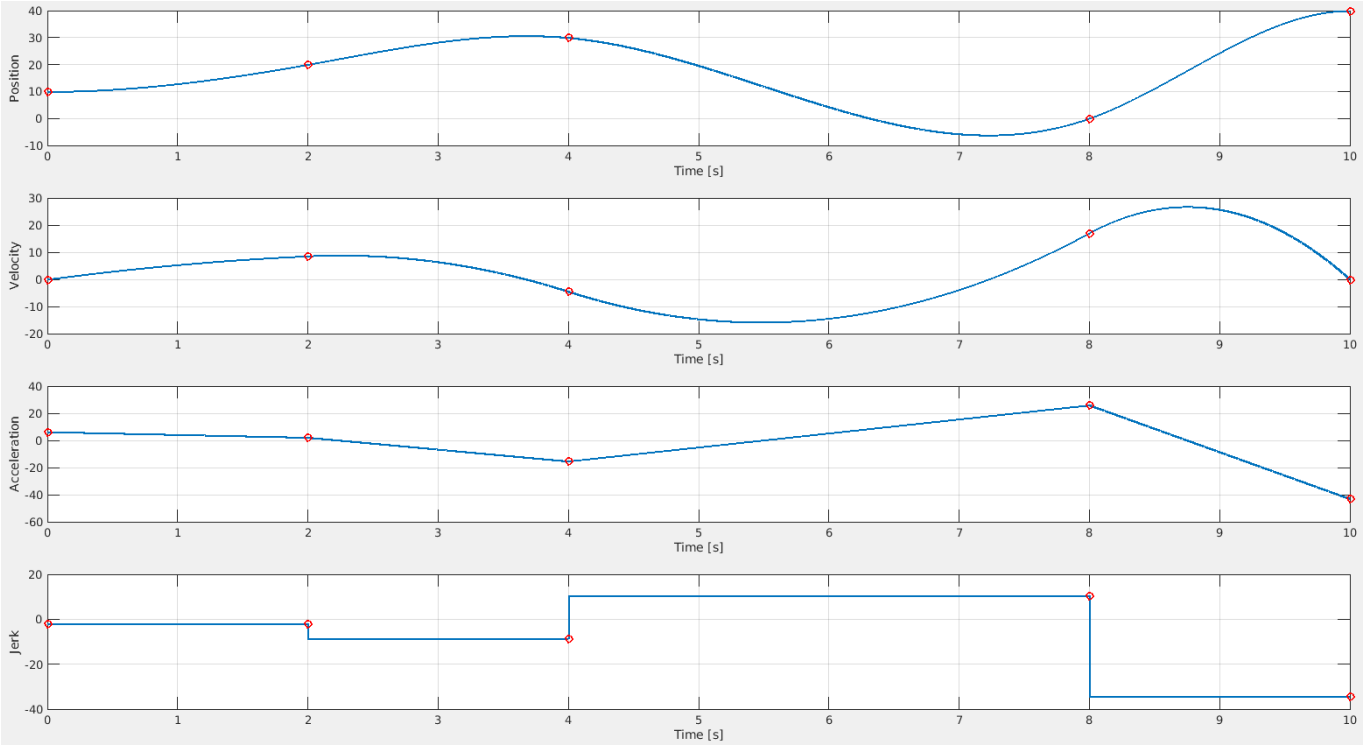


Figure 1.20: Trajectory interpolation with continuous accelerations.

1.4 Assignment 4

1.4.1 Compute cubic splines based on the accelerations with assigned initial and final velocities.

The interpolation based on accelerations is centred around the solution of the system of linear equations:

$$A\ddot{q} = c$$

$$\begin{bmatrix} 2T_0 & T_0 & 0 & & & & & \\ T_0 & 2(T_0+T_1) & T_0 & 0 & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & T_k & 2(T_k+T_{k+1}) & T_{k+1} & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & T_{n-2} & 2(T_{n-2}+T_{n-1}) & T_{n-1} & \\ & & & & & T_{n-1} & 2T_{n-1} \end{bmatrix} \begin{bmatrix} \ddot{q}_0 \\ \vdots \\ \ddot{q}_k \\ \vdots \\ \ddot{q}_n \end{bmatrix} = \begin{bmatrix} 6\left(\frac{q_1-q_0}{T_0} - \dot{q}_0\right) \\ \vdots \\ 6\left(\frac{q_k-q_{k-1}}{T_{k-1}} - \frac{q_{k-1}-q_{k-2}}{T_{k-2}}\right) \\ \vdots \\ 6\left(\dot{q}_n - \frac{q_n-q_{n-1}}{T_{n-1}}\right) \end{bmatrix}$$

Thanks to the tridiagonal form of matrix A, the system can be solved with Thomas' algorithm.

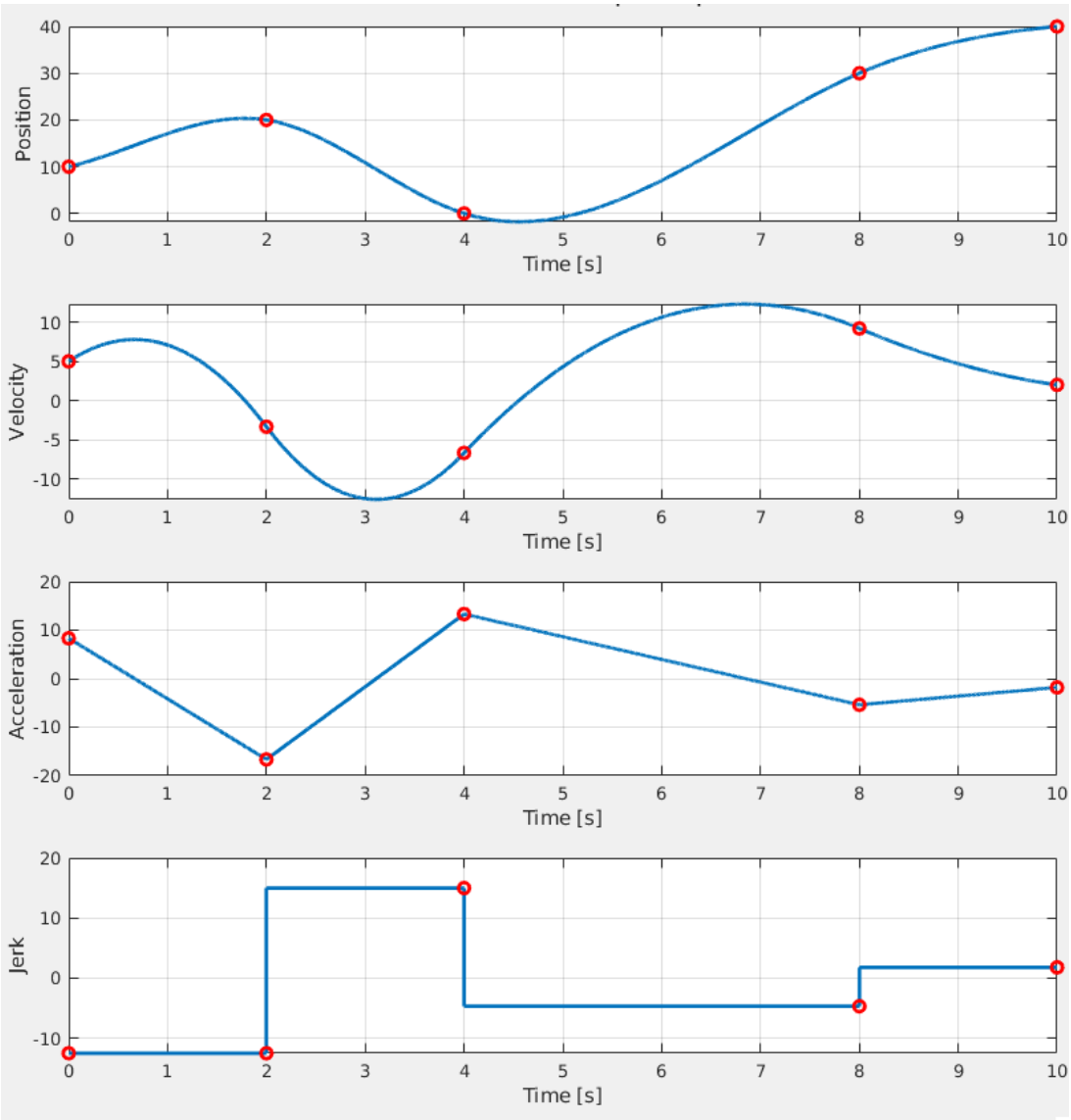


Figure 1.21: Trajectory based on accelerations through $q_k = \begin{bmatrix} 10 & 20 & 0 & 30 & 40 \end{bmatrix}$ at times $t_k = \begin{bmatrix} 0 & 2 & 4 & 8 & 10 \end{bmatrix}$ with $\dot{q}_i = 5$ and $\dot{q}_f = 2$.

1.4.2 Compute the smoothing cubic splines.

The smoothing cubic splines express a trade-off between the fitting of the given points and the minimization of the curvature and acceleration of the trajectory, through a parameter μ. The closeness of the approximation is expressed by a weight vector w, which weighs every point in the trajectory. The closer w_k is to zero, the closer the resulting trajectory is to the interpolation of the point.

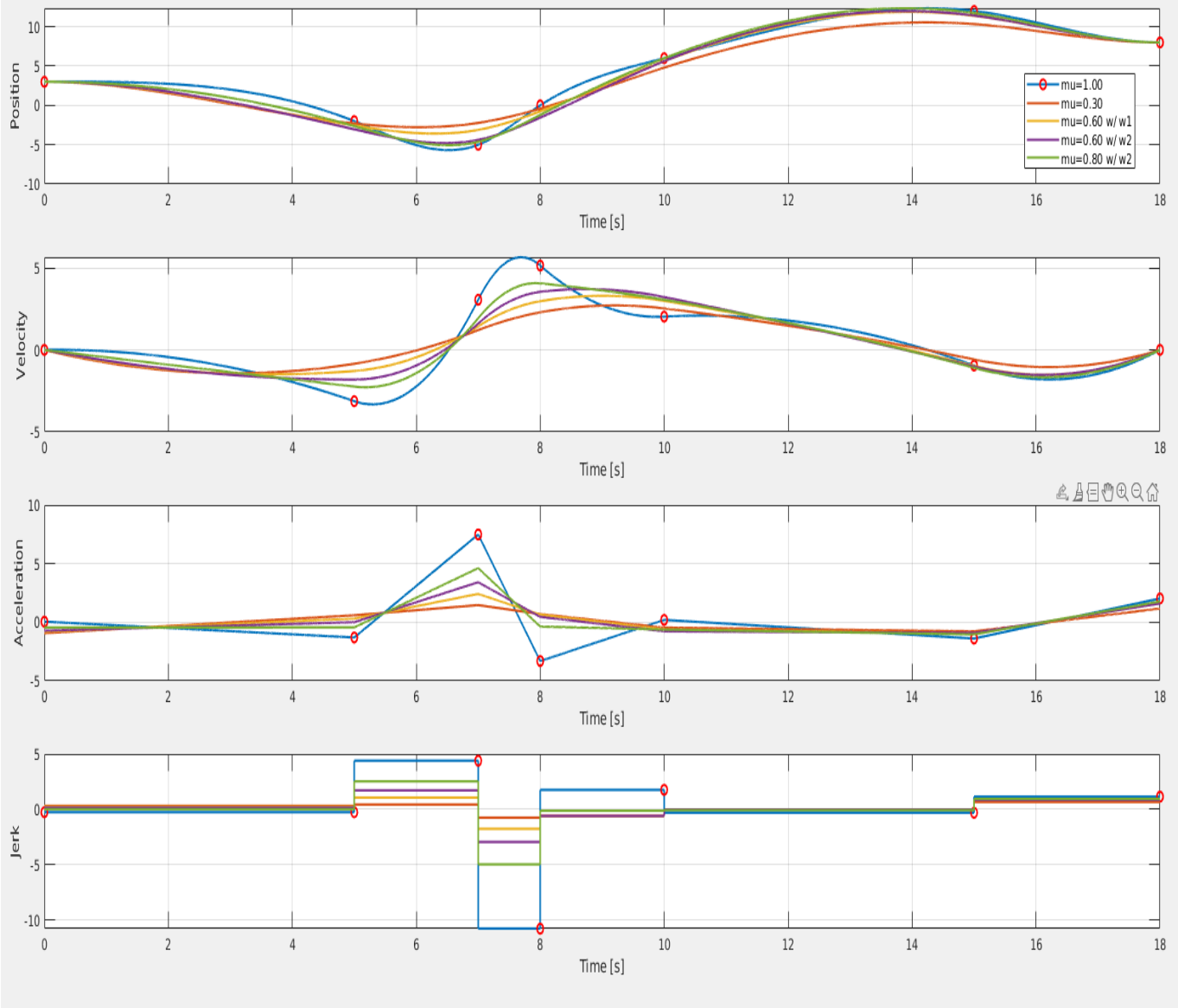


Figure 1.22: Comparison of smoothing cubic splines. $w1 = [\infty \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad \infty]$,$w2 = [\infty \quad 1 \quad 5 \quad 1 \quad 1 \quad 1 \quad \infty]$. Note that the actual used value is $1/w_k$.

1.5 Assignment 5

1.5.1 Compute the 3D trajectory (position, velocity, acceleration and jerk) in the picture as a combination of linear and circular motion primitives and compare it with the trajectory obtained using one of the multi-point methods.

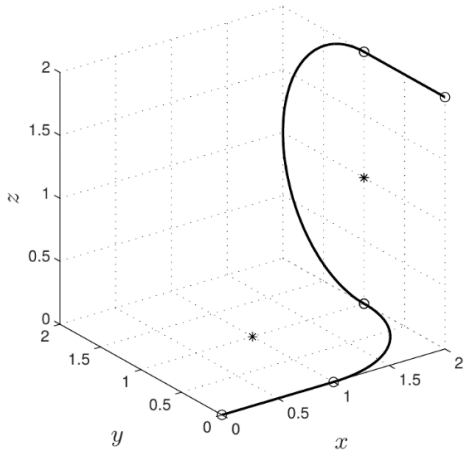


Figure 1.23: 3D trajectory - reference.

Operational space trajectories can be computed by composing multiple motion primitives. The motion primitives used here are:

- Rectilinear path:

$$p(u) = p_i + u(p_f - p_i) \qquad u \in [0, 1]$$

- Circular path:

$$p(u) = c + Rp'(u) = c + R \begin{bmatrix} \rho\cos(u) \\ \rho\sin(u) \\ 0 \end{bmatrix} = c + \begin{bmatrix} (P-c)' & e_3' \times (P-c)' & e_3' \end{bmatrix} \begin{bmatrix} \rho\cos(u) \\ \rho\sin(u) \\ 0 \end{bmatrix} \qquad u \in [0, \theta]$$

  where c is the position of the center of the circular path, P is the position of the starting point on the circle and $e_3 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$.

For comparison, a 3D trajectory is computed by computing the smoothing cubic splines along the three directions. The smoothing spline trajectory is also computed with added waypoints to improve the tracking of the position.

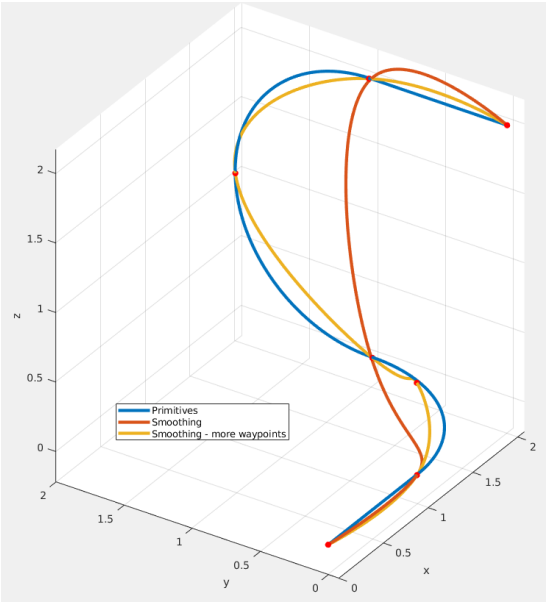


Figure 1.24: 3D trajectory - motion primitives vs smoothing splines ($w_k = \infty \forall k$, $\mu = 1$).

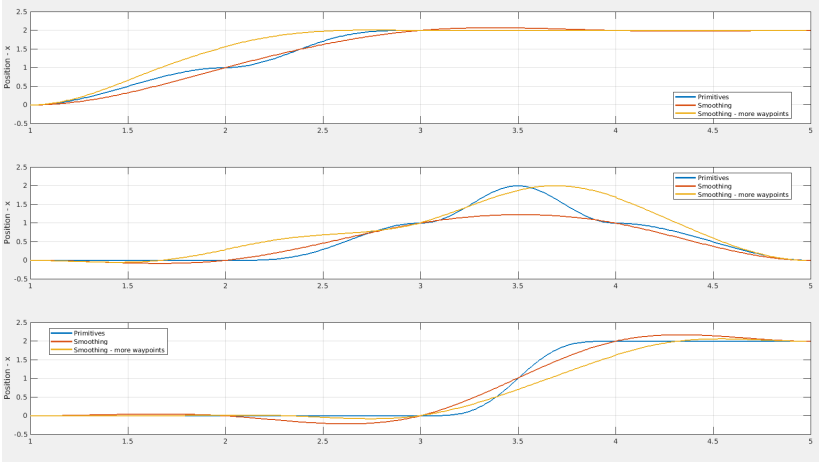


Figure 1.25: 3D trajectory - Position comparison.

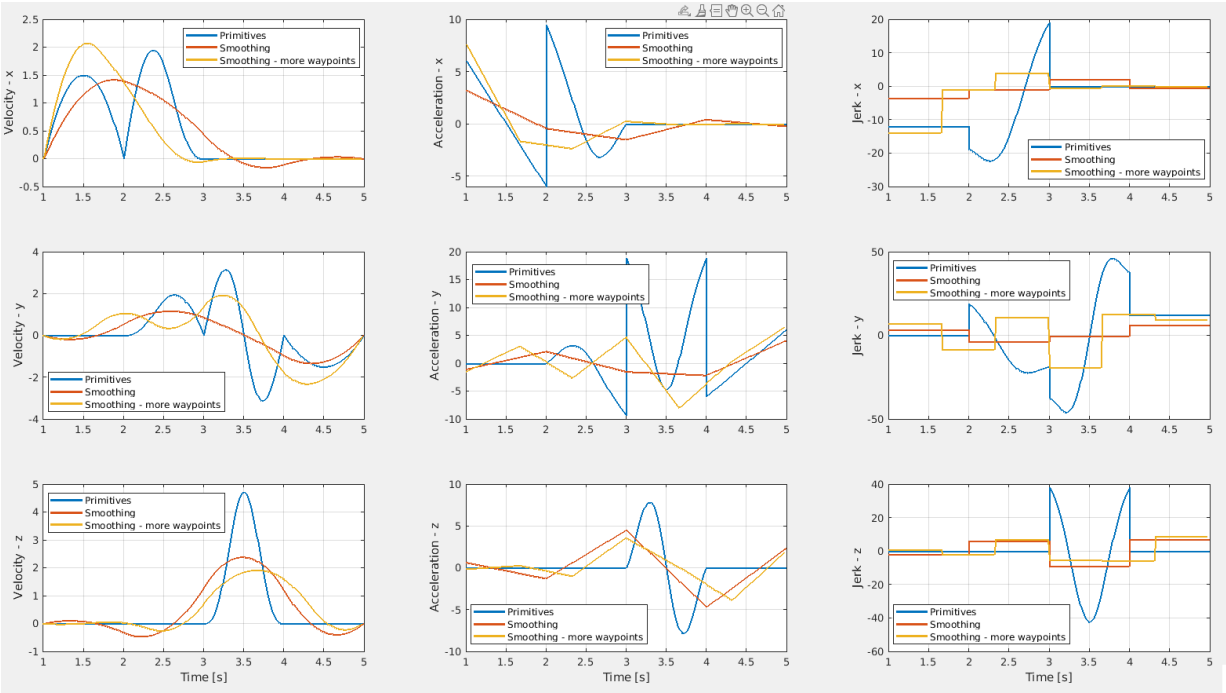


Figure 1.26: 3D trajectory - Velocity, acceleration, jerk comparison.

1.6 Assignment 6

1.6.1 Let p_1, p_2, p_3 be three points on a sphere of center P_0 and radius R. Design the trajectory such that (1) the EE will pass through the three points along the shortest path, and (2) the z axis of the EE is always orthogonal to the sphere.

The shortest path γ on a spherical surface lies on a great circle, which is the biggest circle that belongs to the spherical surface. Such a path is called geodesic.

First of all, the radius vectors of the starting and ending points, p_1 and p_2, of the geodesic are used to compute its direction:

$$\begin{matrix} r_1 = \frac{(p_1 - p_0)}{R} \\ r_2 = \frac{(p_2 - p_0)}{R} \end{matrix} \implies r_g = r_1 \times r_2$$

The rotation matrix that rotates the great circle with $z = 0$ to align it with the geodesic is then:

$$R_\gamma = \begin{bmatrix} r_1 & r_g \times r_1 & r_g \end{bmatrix}$$

where the cross product between r_g and r_1 was arbitrarily chosen (r_2 would have worked as well) to obtain a third orthogonal direction.

Using the rotation matrix we can rotate the parametric form of the great circle as follows:

$$\gamma = R_\gamma \begin{bmatrix} R\cos(u) \\ R\sin(u) \\ 0 \end{bmatrix} + p_0$$

From the parametrization, we obtain the axes of the Frenet frame associated to each point:

$$t = \frac{d\gamma}{du} = R_\gamma \begin{bmatrix} -R\sin(u) \\ R\cos(u) \\ 0 \end{bmatrix} \qquad n = \frac{d^2\gamma}{du^2} = R_\gamma \begin{bmatrix} -R\cos(u) \\ -R\sin(u) \\ 0 \end{bmatrix} \qquad b = t \times n$$

The computed normal vector n is always perpendicular to the sphere surface. In fact it lays on the same direction of the corresponding radius vector of the parametric geodesic:

$$r_\gamma = \frac{\gamma - p_0}{R} = R_\gamma \begin{bmatrix} \cos(u) \\ \sin(u) \\ 0 \end{bmatrix} \implies r_\gamma \times n = R_\gamma \left(\begin{bmatrix} \cos(u) \\ \sin(u) \\ 0 \end{bmatrix} \times \begin{bmatrix} -\cos(u) \\ -\sin(u) \\ 0 \end{bmatrix} \right)$$

The cross product in the parentheses is null for any value of u, therefore n is always parallel to the corresponding radius vector. Since the radius vector's direction is perpendicular to the sphere by definition and the normal vector n starts from a point belonging to the sphere's surface, the vector n is always perpendicular to the sphere's surface.

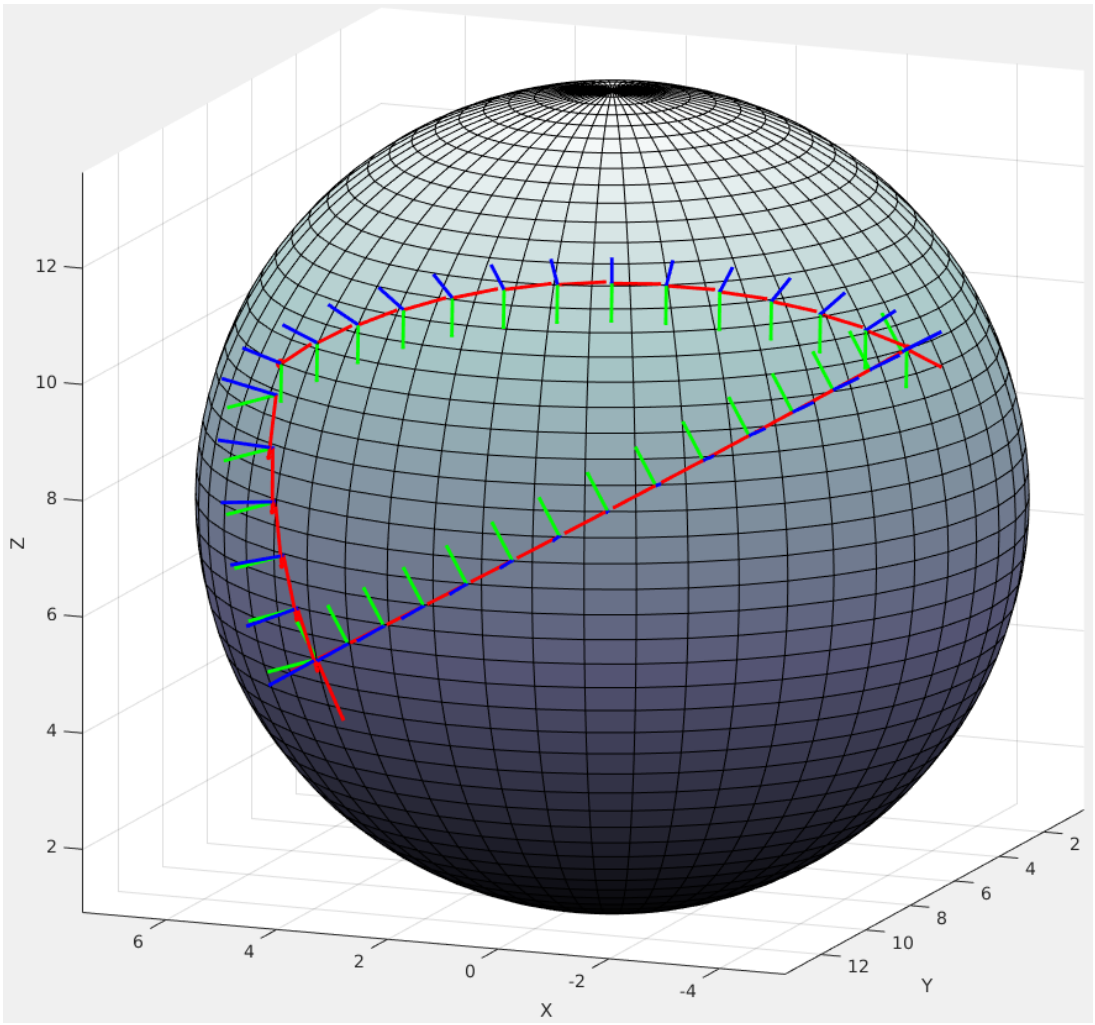


Figure 1.27: Geodesics between three random sphere points with the associated Frenet frames (n in blue, t in green, b in red).

1.7 Assignment 7

1.7.1 Plan the pick-and-place task for the UR5 robot in ROS for four cubes using three different orientations for the end-effector.

The pick-and-place task is divided into two phases: scanning and pick-and-place.

The scanning consists of a composition of rectilinear and circular motion primitives that make sure that the camera mounted on the end-effector of the robot covers the whole workspace, detecting all cubes and their destination.

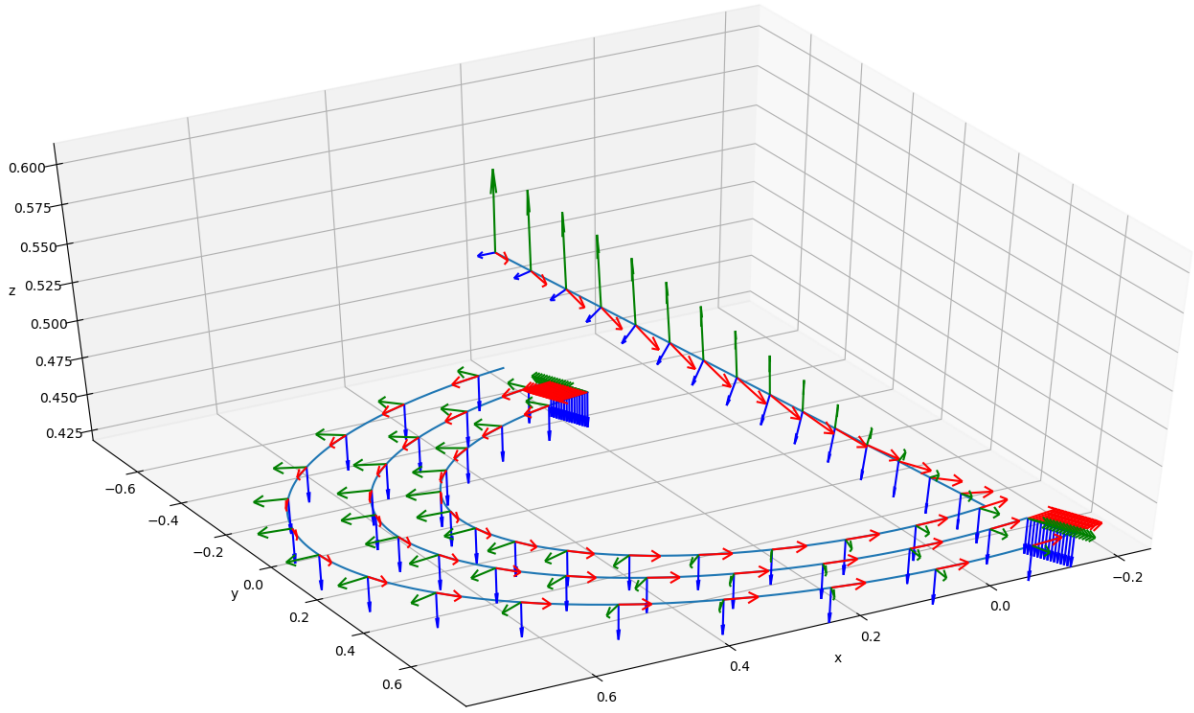


Figure 1.28: End-effector path and associated Frenet frames for the scanning phase.

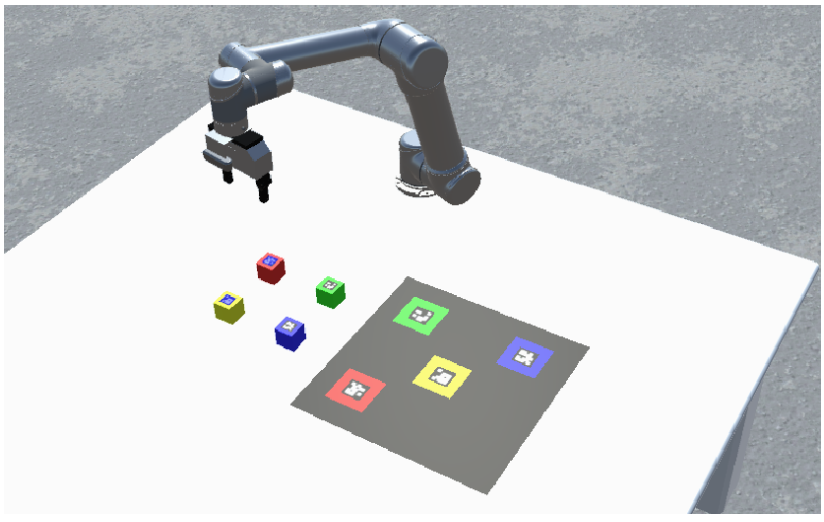


Figure 1.29: Scanning task in Unity. The highlighted ARUCO markers are the ones currently detected by the end-effector camera.

After the cubes and their destinations have been found, the pick-and-place phase is carried out as follows:

- Move linearly in Cartesian space towards the cube location.
- Lower the end-effector onto the cube, close the gripper and then raise the end-effector back up.
- Move linearly to the home position.
- Move linearly to the destination, open the gripper and then raise the end-effector back up.
- Move linearly to the home position.

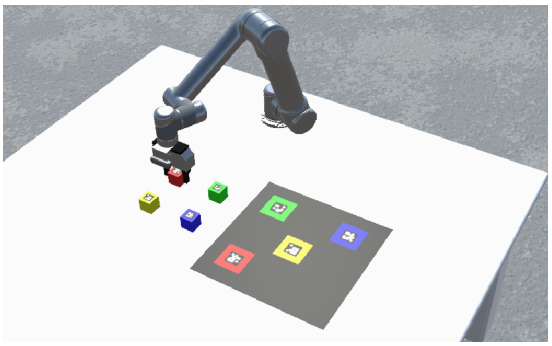


Figure 1.30: Pick

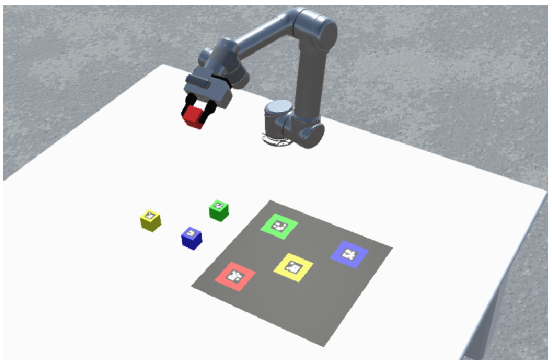


Figure 1.31: Homing

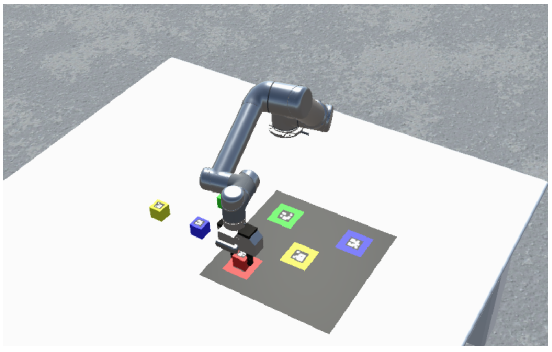


Figure 1.32: Place

Chapter 2

Vision

2.1 Assignment 1

2.1.1 3D model creation using Zephyr

The object chosen for the assignment was a plastic statue of an angel.

119 photos of it were collected, all around it and at three different heights. The 3D model was obtained with Zephyr after the following steps:

- Sparse point cloud generation, using the 'Human body' and 'High details' presets.
- Dense point cloud generation, using the 'Human body' and 'High details' presets.
- Mesh generation, using the 'Default' and 'High details' presets.
- Textured mesh generation, using the 'Default' and 'High details' presets.

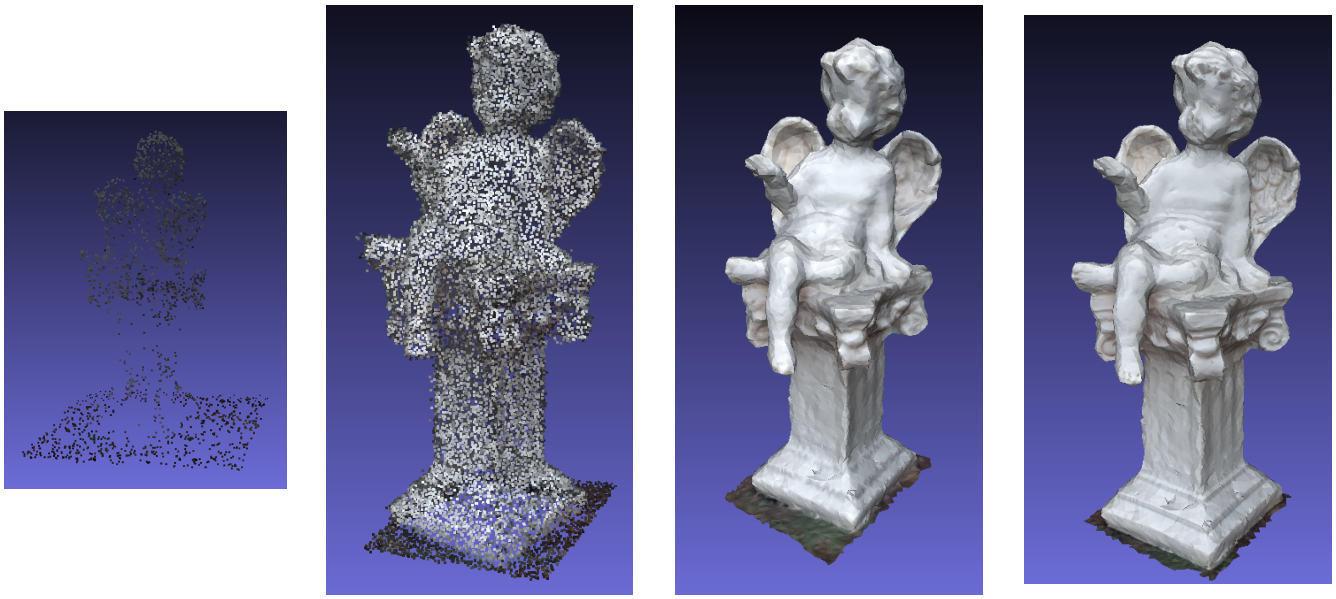


Figure 2.1: From left to right: sparse point cloud, dense point cloud, mesh and textured mesh.

2.2 Assignment 2

2.2.1 3D point cloud from range image

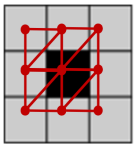
Using publicly available RGB-D images (http://redwood-data.org/3dscan/dataset.html), reconstructing the corresponding point cloud is easily achieved by applying the projection equations:

$$x = (u - u_0)\frac{z}{f_u} \quad y = (v - v_0)\frac{z}{f_v} \quad z = RGBD(u, v)$$

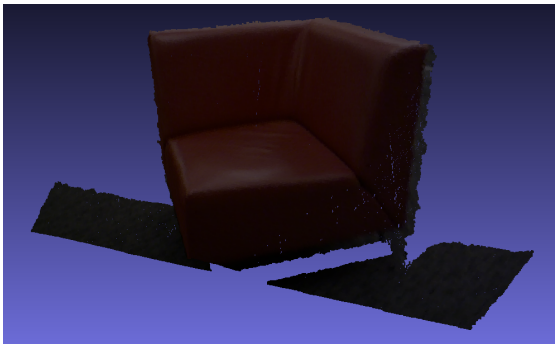
where $u_0, v_0, f_u = k_u f$ and $f_v = k_v f$ are the intrinsic camera parameters. To isolate the object of interest, a depth-thresholding step is introduced, which simply means to consider only z values between predetermined limits.

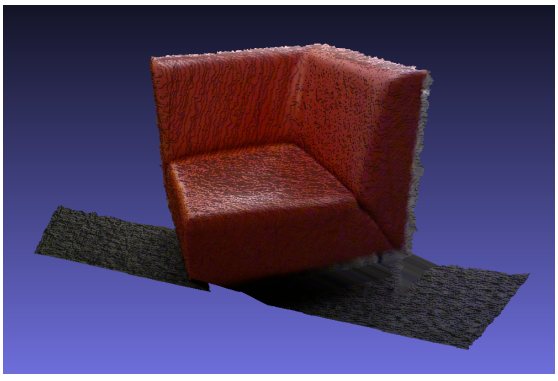
2.2.2 Mesh from point cloud

Once a cloud of points is obtained, a mesh can be computed in a number of ways. A simple approach is to loop over every vertex and explore it's 3x3 neighborhood. If the vertex's neighbour is a valid vertex (i.e. if it is within the depth threshold), it can be used along another valid vertex to construct a triangular face.



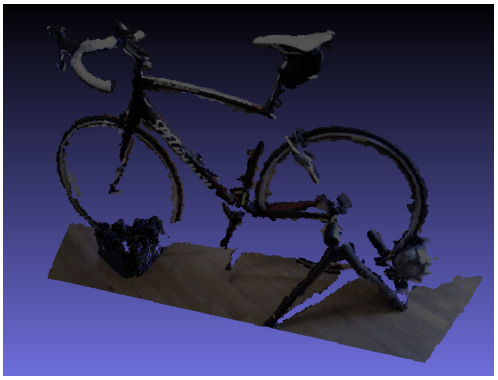
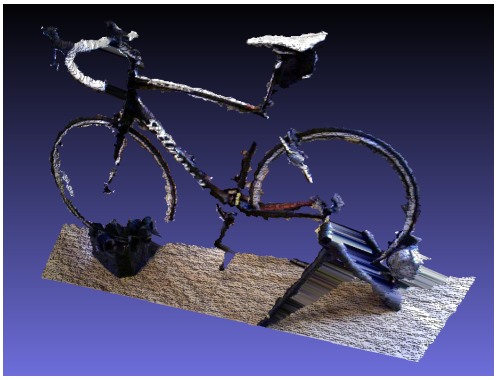
Example 1





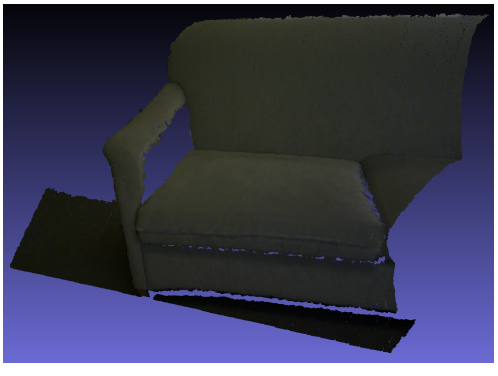
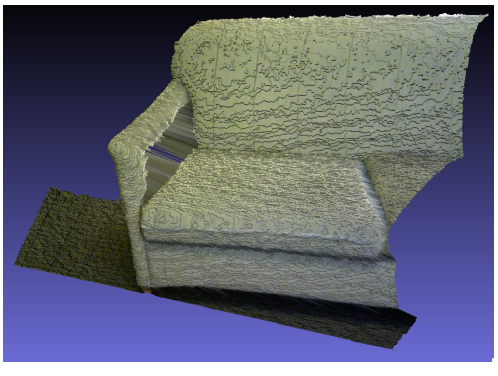
Example 2



Example 3



2.3 Assignment 3

2.3.1 Compute the canonical orientation of the 3D model

The orientation of the point clouds and meshes produced by Zephyr is arbitrary. Since some 3D registration algorithms (e.g. ICP) require a strong initialization, a canonical orientation is introduced. The canonical orientation is derived from the covariance matrix of the point cloud, from which three principal directions are extracted using Principal Component Analysis (PCA).

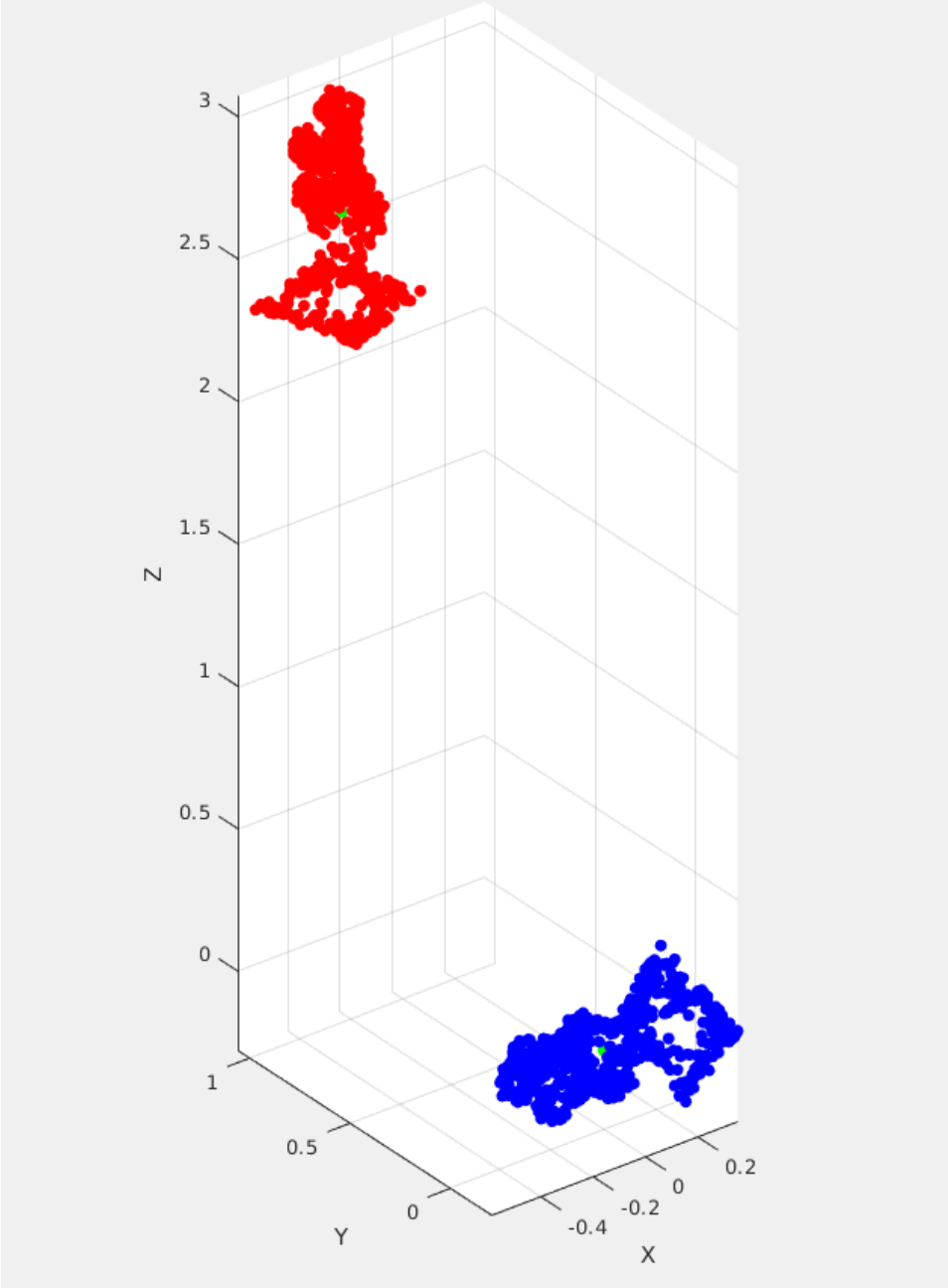


Figure 2.2: Zephyr orientation (red) vs canonical orientation (blue), centroids (green)

2.4 Assignment 4

2.4.1 Image analysis using morphological operators

2D image analysis can be carried out with the use of morphological operators. Image closing and opening let us remove the background of pictures and isolate the objects within them. More complex operations can then be carried out, inferring for example geometric information on the shapes produced.

Example 1

Background removed by opening with a structuring element in the shape of a disk of size 65. Shapes improved by image closing with a structuring element in the shape of a disk of size 25 and by filling holes. Noise removed after image binarization with area opening, removing all shapes with fewer than 50 pixels.

Classification used the 'Circularity' property to distinguish between coins and USB stick, as well as the 'MajorAxis' and 'MinorAxis' properties, which were averaged to compute the radius of the coins to detect which was smaller and which was bigger.

Figure 2.3: Original image (left) vs extracted shapes (right).

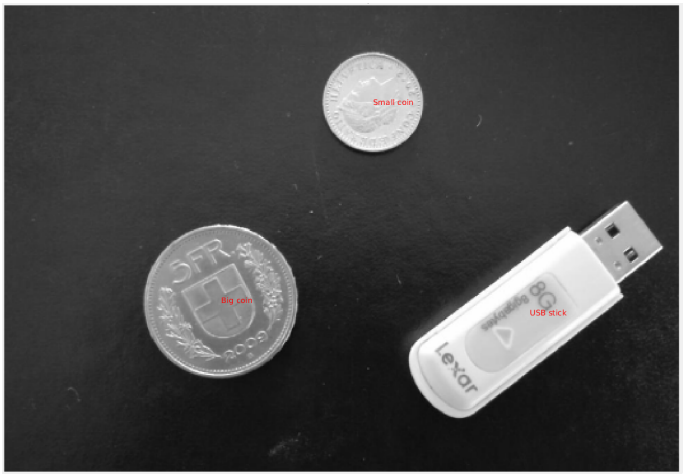


Figure 2.4: Labeled image.

Example 2

Background removed by opening with a structuring element in the shape of a disk of size 65. Shapes improved by image closing with a structuring element in the shape of a disk of size 25 and by filling holes. Noise removed after image binarization with area opening, removing all shapes with fewer than 50 pixels.

Classification used the 'Circularity' property to distinguish between washers and bolts. The 'MajorAxis' and 'MinorAxis' properties were averaged to compute the outer radius of the washers, on which a quintic polynomial was fit to determine the relation with the actual inner diameter. For the bolts, the 'Orientation' property was used to align the shapes vertically, so that the bottom part of them, which contains the threaded part, could be extracted. On the extracted shapes, the 'MajorAxis' and 'MinorAxis' properties were extracted and used to fit cubic polynomials to determine, respectively, the bolt length and diameter.

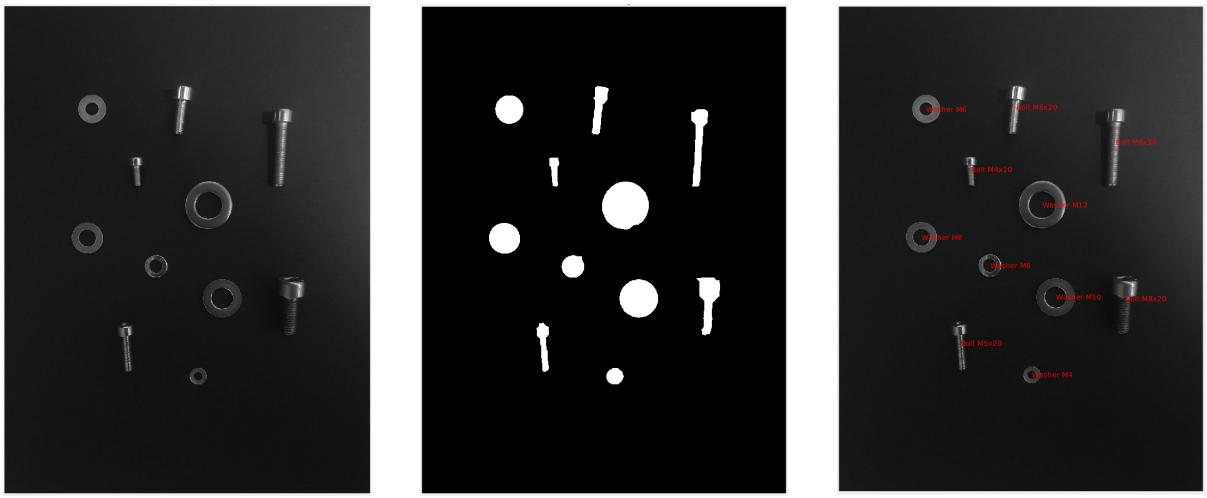


Figure 2.5: Original image (left) vs extracted shapes (middle) vs labels (right).

2.5 Assignment 5

2.5.1 3D analysis pipeline

Using 3D geometry, interesting information can be reconstructed from point clouds. This can be done in combination with the 2D analysis with morphological operators.

Example 1

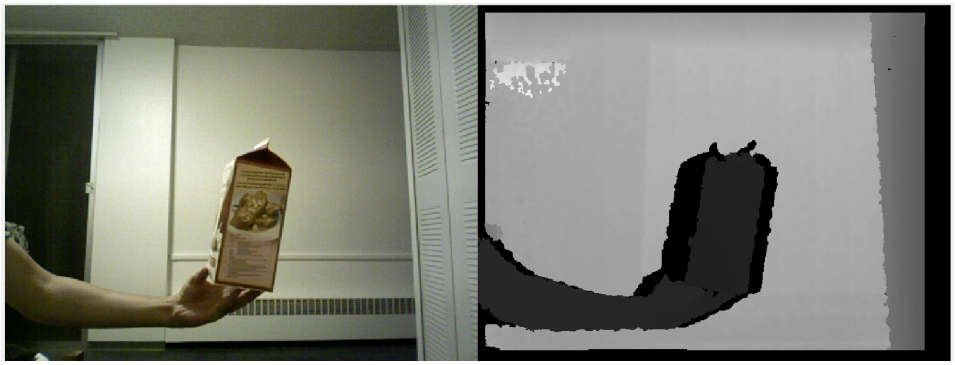


Figure 2.6: Rgb image (left) vs range image (right)

Figure 2.7: Extracted shapes (left) vs masked image (middle) vs extracted boundary (right)

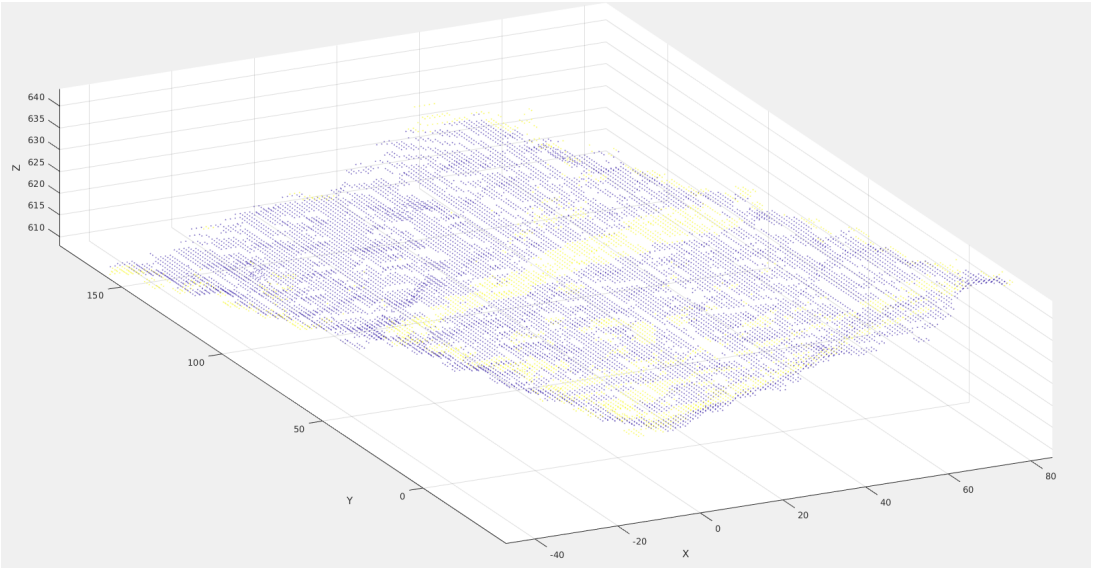


Figure 2.8: Extracted point cloud

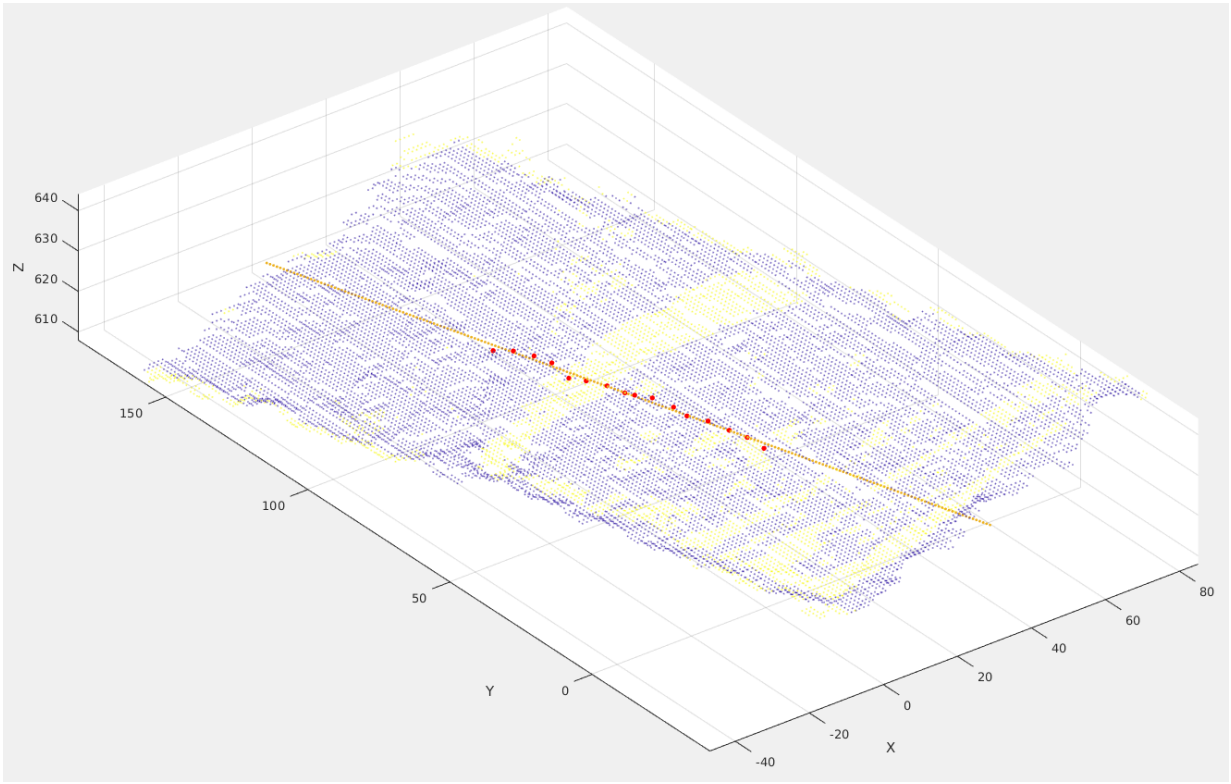


Figure 2.9: Main orientation - 3D line fitting

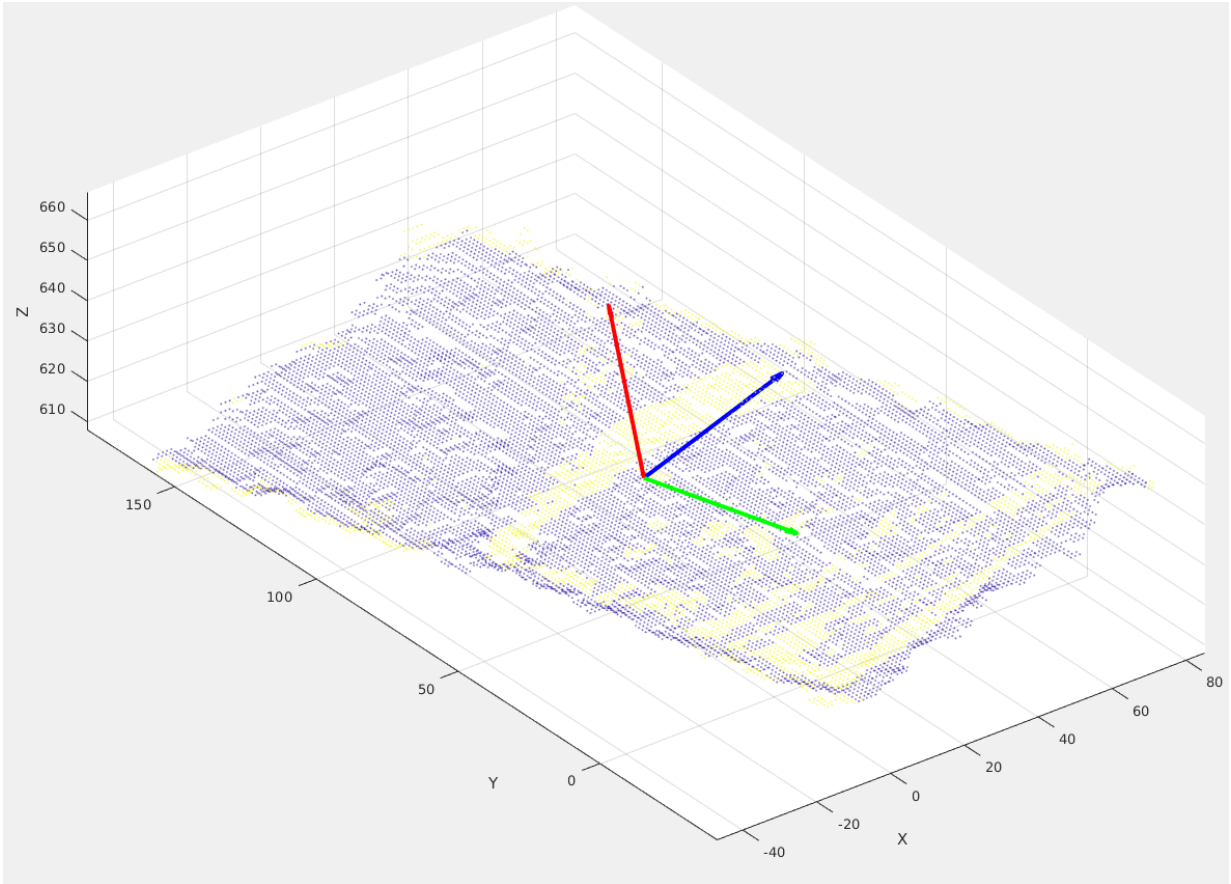


Figure 2.10: Orientation - PCA

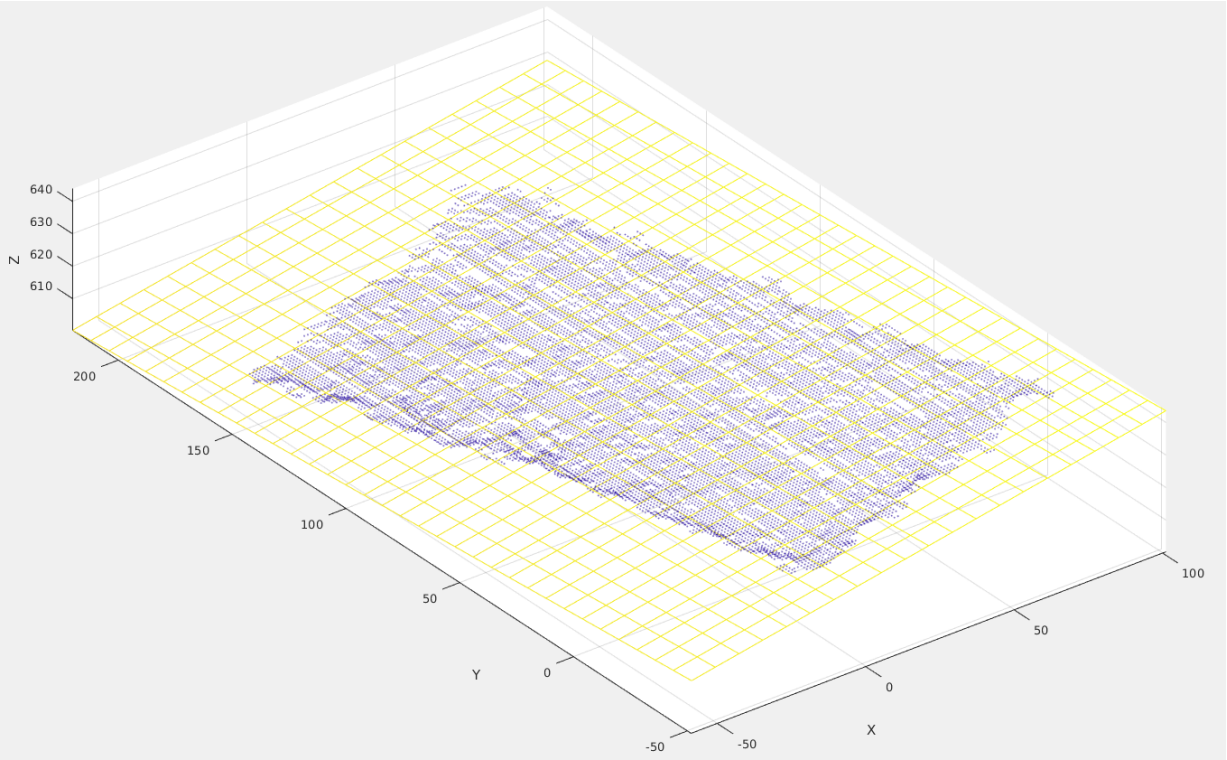


Figure 2.11: Plane fitting

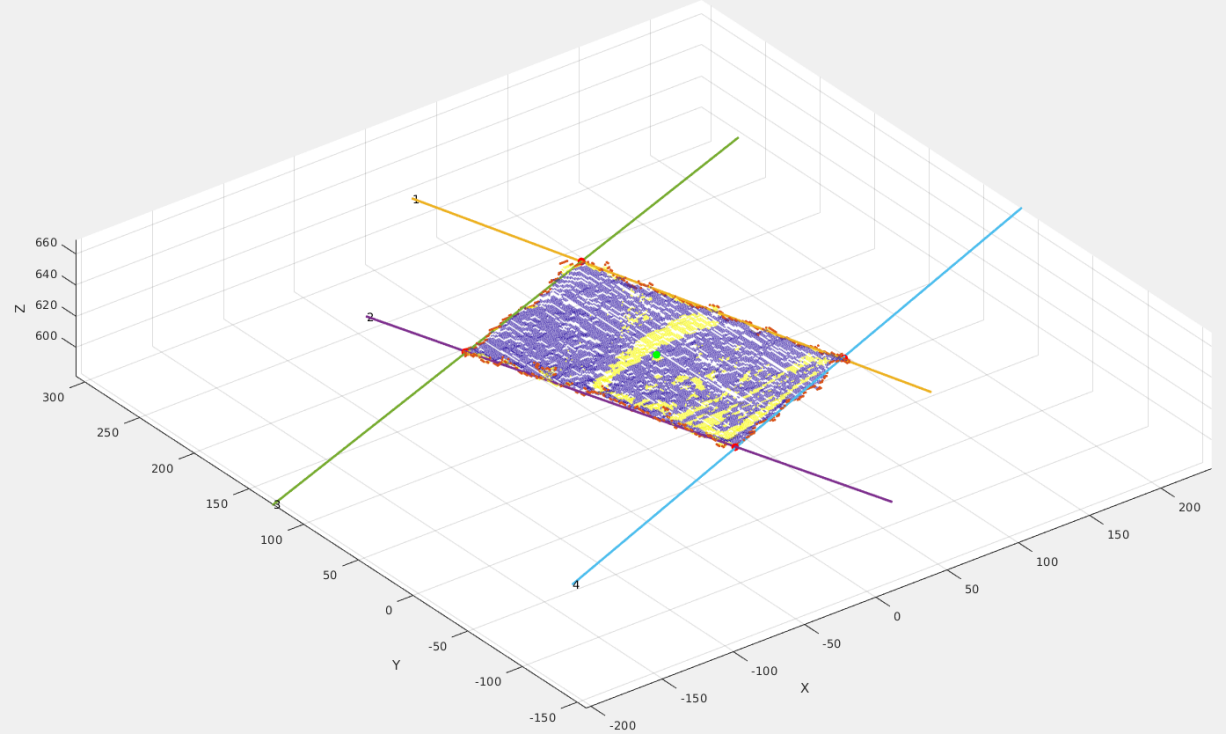


Figure 2.12: Boundary with 3D line fitting and corner detection

Example 2

Figure 2.13: Rgb image (left) vs range image (right)

Figure 2.14: Extracted shapes (left) vs masked image (middle) vs extracted boundary (right)

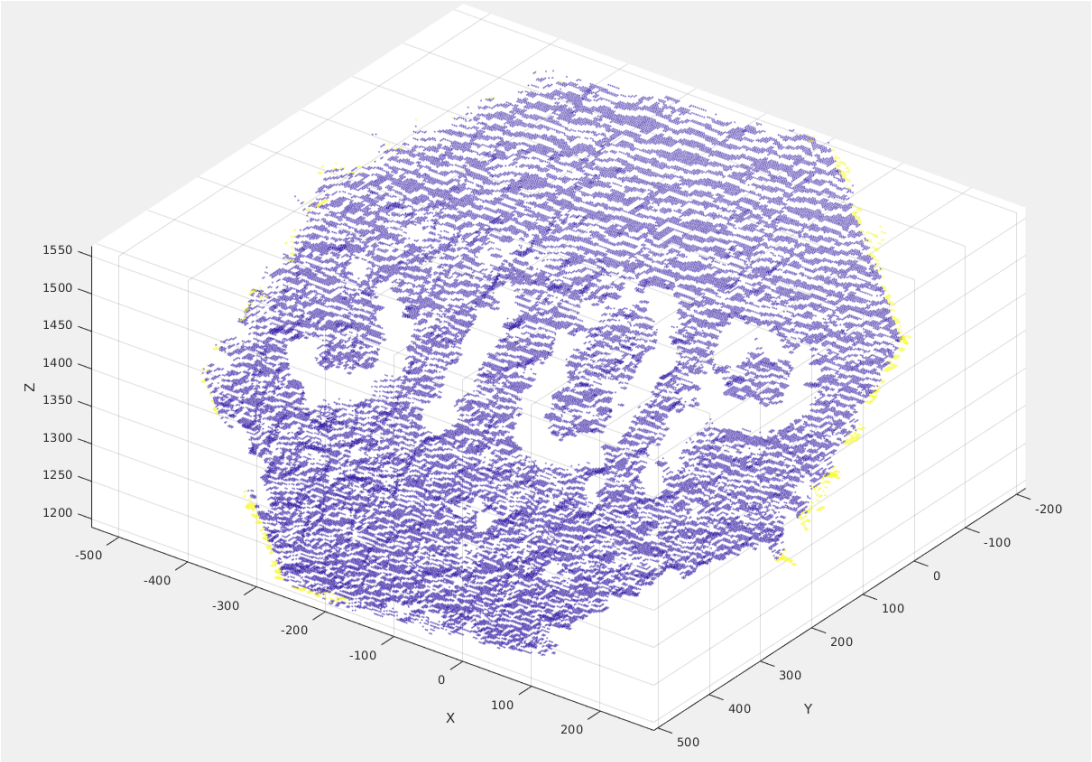


Figure 2.15: Extracted point cloud

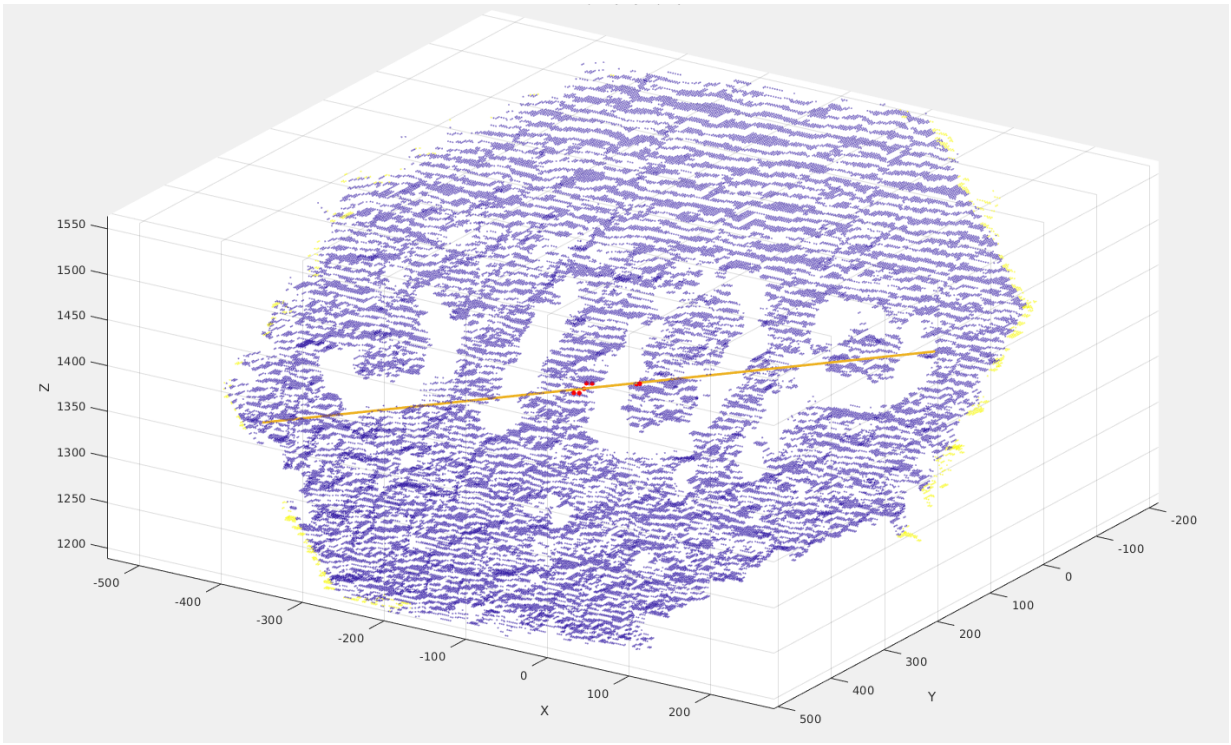


Figure 2.16: Main orientation - 3D line fitting

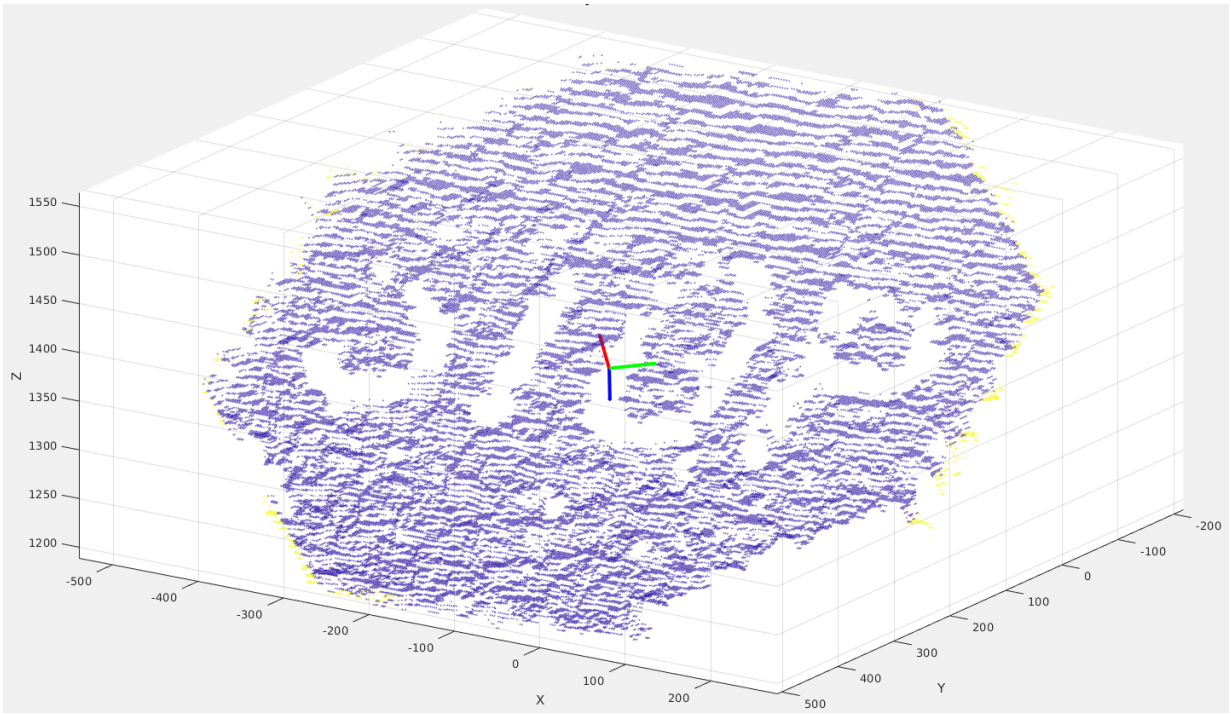


Figure 2.17: Orientation - PCA

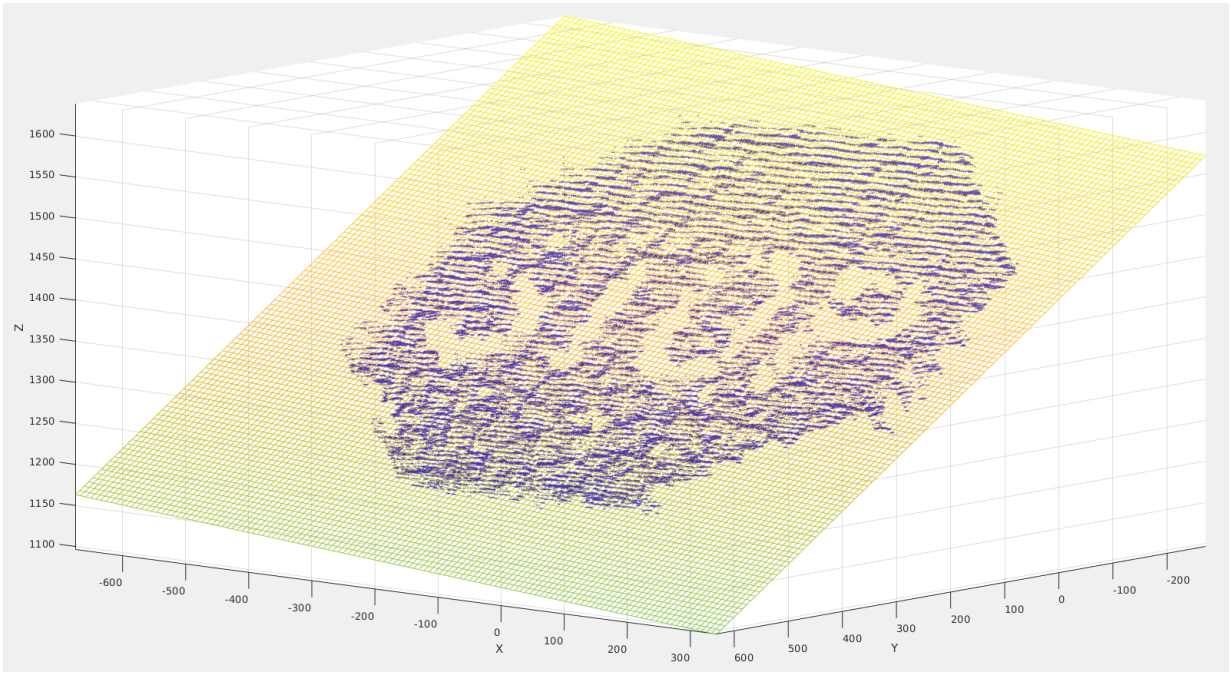


Figure 2.18: Plane fitting

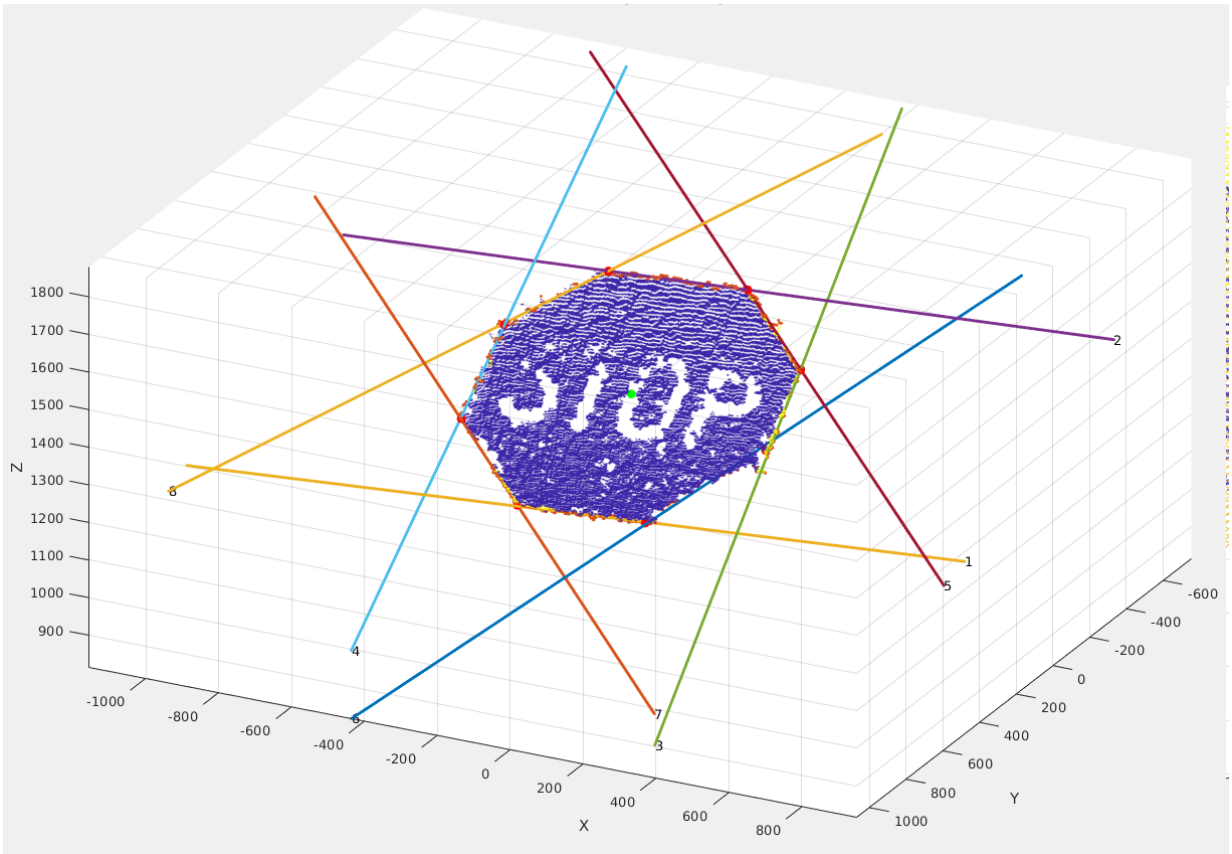


Figure 2.19: Boundary with 3D line fitting and corner detection

2.6 Assignment 6

2.6.1 Object to camera pose framework

The model computed in the first assignment was used to test an object to camera pose framework (https://github.com/creminem94/CameraPoseEstimation).

The pipeline is as follows:

- Choosing the reference image among the ones used in Zephyr.
- Exporting the reference camera's intrinsic parameters and point visibility from Zephyr.
- Creating the reference model.
- Choosing some new images of the object.
- Iteratively applying pose estimation using Fiore's method to them to compute the poses.

Figure 2.20: Reference image

Some results: in red the actual 2D feature points, in blue the reprojection using the computed camera poses.

Reconstructed camera poses: