Research project:

Multirotor terrain following using laser range finders

2018-2019 academic year

Eden Shazar

Supervisor: Moshe Idan

Contents

1. Overview

This research aims to examine the feasibility of terrain following guidance and control for a multirotor over non-planar three-dimensional terrain using laser range finder (LRF) measurements alone. As a starting point, it is assumed the vehicle's attitude is stabilized, and its horizontal velocity is controlled about a constant velocity trim point. The guidance algorithm to be implemented is a simplified version of the one proposed by Ratnoo, Shima et al. [1], and expanded upon by Ratnoo et al [2]. The work is based on previous research done by A. Shender and M. Idan [3], in which the two-dimensional case was studied.

The work was carried out over the course of two semesters. The first semester was focused on basic infrastructure:

- A dynamic model of a quadcopter.
- Random terrain generation
- LRF simulation.

The second semester expanded upon the first to complete the simulation and analysis software needed for future work:

- The Ratnoo guidance algorithm was simplified in order to test for its feasibility with limited data or conditions and integrated in the simulation.
- The control has been discretized and simulation made hybrid (continuous plant; discrete hardware).
- Measurement noise was added.
- All existing software was wrapped with graphical applications to be used for statistical and parametric analysis.

As per the above, the work is presented in chronological order with the first semester consisting of sections 2-5 and the second of sections 6-8. At the end of the project, this simulation environment was handed off to serve as the basis of a future research project.

2. Quadcopter model

The dynamic model of the quadcopter was implemented using the equations and parameters described by Luukkonnen T. [4], where the only aerodynamic effect on the quadcopter taken into account (other than the basic rotor lift and drag) is a drag force scaling linearly with velocity. The body axes were rotated about the z_B axis by $-45°$ such that the quadcopter flies in an X configuration rather than a + configuration. The parameters used are listed in Table 1.

The following are the derived 6DOF equations. The positional dynamics are given by:

$$(1) \qquad \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{T}{m} \begin{bmatrix} \mathrm{C}(\psi)\,\mathrm{S}(\theta)\,\mathrm{C}(\phi) + \mathrm{S}(\psi)\,\mathrm{S}(\phi) \\ \mathrm{S}(\psi)\,\mathrm{S}(\theta)\,\mathrm{C}(\phi) - \mathrm{C}(\psi)\,\mathrm{S}(\phi) \\ \mathrm{C}(\theta)\,\mathrm{C}(\phi) \end{bmatrix} - \frac{1}{m} \begin{bmatrix} A_x & 0 & 0 \\ 0 & A_y & 0 \\ 0 & 0 & A_z \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$$

where x, y and z are the vehicle's position coordinates in an inertial frame, T is the total rotor thrust, ϕ, θ and ψ Euler angles (roll, pitch and yaw respectively), and $\mathrm{C}(\cdot)$ and $\mathrm{S}(\cdot)$ are abbreviated cosine and sine functions respectively. The angular dynamics are given by:

$$(2) \qquad \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} {}^{1}/_{I_{xx}} & 0 & 0 \\ 0 & {}^{1}/_{I_{yy}} & 0 \\ 0 & 0 & {}^{1}/_{I_{zz}} \end{bmatrix} \left(\begin{bmatrix} (I_{yy} - I_{zz})qr \\ (I_{zz} - I_{xx})pq \\ (I_{xx} - I_{yy})pq \end{bmatrix} - I_{zz,R} \begin{bmatrix} q \\ -p \\ 0 \end{bmatrix} \omega + \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \right)$$

where p, q, and r are the vehicle's angular rates in body coordinates, $\omega = \omega_1 - \omega_2 + \omega_3 - \omega_4$ is the sum of the rotors' signed angular rates and τ_ϕ, τ_θ and τ_ψ are the rotors' moments about the x_B, y_B and z_B axes respectively. All other parameters appearing in equations (1) and (2) are described in Table 1.

Table 1: Physical parameters used in the dynamic model

Parameter	Value	Units	Description
g	9.81	${}^{m}/_{s^2}$	Gravitational acceleration
m	0.468	kg	Quadcopter mass
l	0.225	m	Rotor arm length
I_{xx}	4.856×10^{-3}	$kg \cdot m^2$	Moment of inertia about x_B
I_{yy}	4.856×10^{-3}	$kg \cdot m^2$	Moment of inertia about y_B
I_{zz}	8.801×10^{-3}	$kg \cdot m^2$	Moment of inertia about z_B
A_x, A_y, A_z	0.25	${}^{kg}/_{s}$	Drag coefficients (velocity damping)
τ	0.1	s	Rotor time constant
k	2.980×10^{-6}	–	Rotor lift coefficient
b	1.140×10^{-7}	–	Rotor drag coefficient
$I_{zz,R}$	3.357×10^{-5}	$kg \cdot m^2$	Rotor moment of inertia about z_B

3. Control

3.1. Trim point

As the flight mode of interest is maneuvering about some nominal velocity, as opposed to takeoff, hovering, etc., a trim point at which the vehicle is in such a steady state was found. The trim point was used to construct a linear model with which linear controllers were designed, and as the initial conditions for the simulation.

The trim point was chosen such that the quadcopter is only rotated with positive pitch θ_{trim} and is moving at a constant velocity of $V_{trim} = 4\ \left[\frac{m}{s}\right]$ along the inertial x axis. As such, the equilibrium equations in the inertial xz plane are given by equation (3).

$$(3) \qquad \begin{cases} T_{trim}\sin(\theta_{trim}) = D_{trim} = A_x V_{trim} \\ T_{trim}\cos(\theta_{trim}) = mg \end{cases}$$

where T_{trim} is the total thrust supplied by the four rotors and D_{trim} is the drag force. The pitch is thus given by equation (4), total thrust by equation (5) and angular velocity of all four rotors by equation (6). Table 2 contains the results for the chosen velocity.

$$(4) \qquad \theta_{trim} = \arctan\left(\frac{A_x V_{trim}}{mg}\right)$$

$$(5) \qquad T_{trim} = \frac{mg}{\cos(\theta_{trim})}$$

$$(6) \qquad \omega_{trim} = \sqrt{\frac{T_{trim}}{4k}}$$

Table 2: Trim values for $V_{trim} = 4\ [^m/_s]$

Parameter	Value	Units	Description
θ_{trim}	12.3°	deg	Pitch angle
T_{trim}	4.7	N	Total thrust
ω_{trim}	6000	RPM	Rotor angular velocity

3.2. Controller design

In the following subsections several control loops are closed one after the other, treating the problem as a successive set of SISO problems. In each subsection, the linearization of the open loop was calculated using Simulink's linearization tool. The controllers chosen, closed loop systems and their time and frequency domain characteristics are detailed. Afterwards, validation of the closed loop against the full non-linear simulation is presented as well as an examination of the control effort needed after each step. All linear systems' results match well with the non-linear simulation and require reasonable control efforts. At this stage, it was decided arbitrarily that the following results are satisfactory, and that more complex design could be performed to gain better performance.

The loops were closed successively in the order of appearance in the following subsections (visualized in the following figure), such that each open loop given takes previous controllers into account. The root loci used to design the controllers are presented in Appendix A.

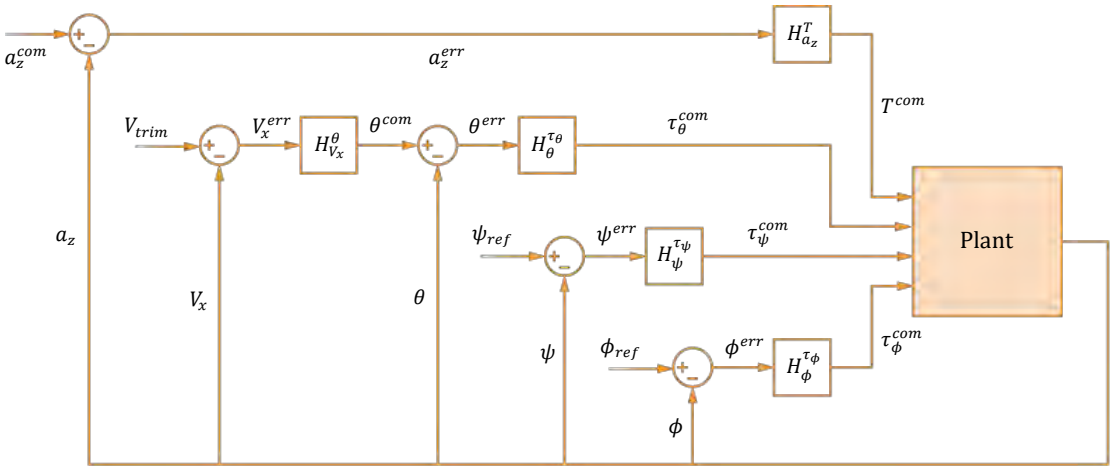


Figure 1: Series of designed SISO controllers

3.2.1. Roll controller

Open loop:

$$H_{\tau_\phi}^{\phi}(s) = \frac{2060}{s^2(s+10)}$$

Chosen controller – lead compensator:

$$H_{\phi_{error}}^{\tau_\phi}(s) = 0.06\frac{s+0.7}{s+7}$$

Closed loop:

$$H_{\phi_{error}}^{\phi}(s) = 126\frac{s+0.7}{(s+12)(s+1.8)(s^2+3.2s+4.1)}$$

Closed loop characteristics in the frequency domain:

$$GM = 17.9\,[dB] \quad \left(\omega = 7.62\,\left[\frac{rad}{s}\right]\right) \;,\; PM = 44° \quad \left(\omega_c = 1.83\,\left[\frac{rad}{s}\right]\right)$$

In the time domain:

$$Overshoot = 34.2\% \;,\; t_s^{2\%} = 3.55\,[s] \;,\; type\ II$$

Validation using step input:

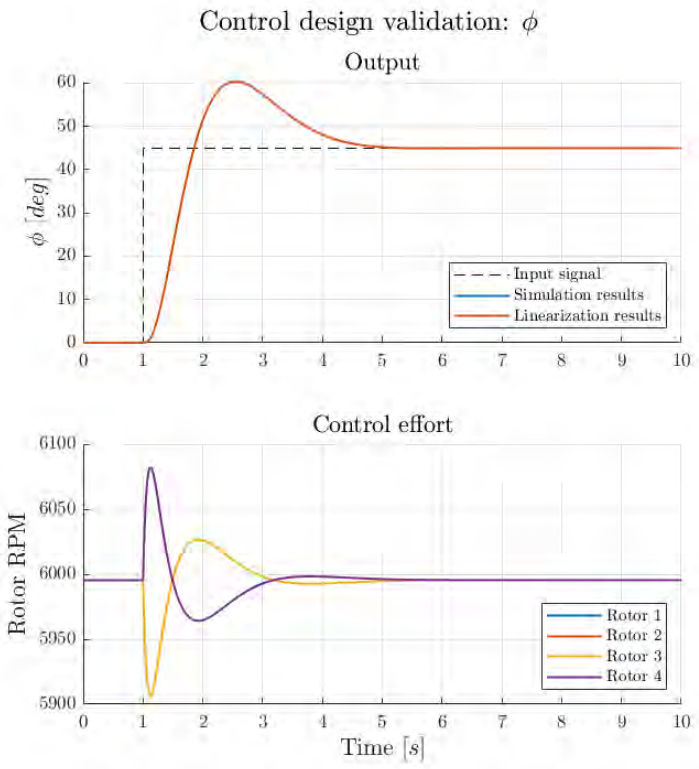


Figure 2: Roll controller validation

The above results, though exhibiting high overshoot, represent the best performance that was achieved using a simple lead compensator.

3.2.2. Yaw controller

Open loop:

$$H_{\tau_\psi}^{\psi}(s) = \frac{1160}{s^2(s+10)}$$

Chosen controller – lead compensator:

$$H_{\psi_{error}}^{\tau_\psi}(s) = 0.11\frac{s+0.7}{s+7}$$

Closed loop:

$$H_{\psi_{error}}^{\psi}(s) = 130\frac{s+0.7}{(s+12)(s+1.6)(s^2+3.4s+4.7)}$$

Closed loop characteristics in the frequency domain:

$$GM = 17.7\ [dB] \quad \left(\omega = 7.62\ \left[\frac{rad}{s}\right]\right)\ ,\ PM = 43.9° \quad \left(\omega_c = 1.87\ \left[\frac{rad}{s}\right]\right)$$

In the time domain:

$$Overshoot = 34.3\%\ ,\ t_s^{2\%} = 3.5\ [s]\ ,\ type\ II$$

Validation using step input:

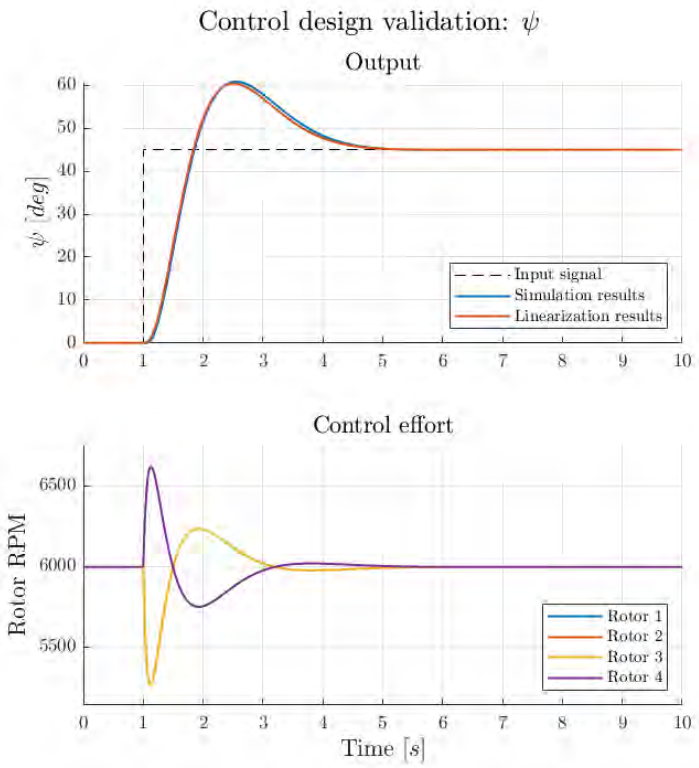


Figure 3: Yaw controller validation

The above results, though exhibiting high overshoot, represent the best performance that was achieved using a simple lead compensator.

3.2.3. Pitch controller

Open loop:

$$H_{\tau_\theta}^{\theta}(s) = \frac{2060}{s^2(s+10)}$$

Chosen controller – lead network:

$$H_{\theta_{error}}^{\tau_\theta}(s) = 5.8\frac{(s+10)(s+3)}{(s+50)(s+30)}$$

Closed loop:

$$H_{\theta_{error}}^{\theta}(s) = 11900\frac{(s+10)(s+3)}{(s+57)(s+10)(s+7.9)(s^2+15s+79)}$$

Closed loop characteristics in the frequency domain:

$$GM = 18.6\,[dB] \quad \left(\omega = 35.5\left[\frac{rad}{s}\right]\right) \;,\; PM = 45.4° \quad \left(\omega_c = 8.04\left[\frac{rad}{s}\right]\right)$$

In the time domain:

$$Overshoot = 32.5\% \;,\; t_s^{2\%} = 0.839\,[s] \;,\; type\ II$$

Validation using step input, smoothed with a prefilter to avoid unrealistic control effort as a pure step input leads to a large initial spike in rotor angular velocity:

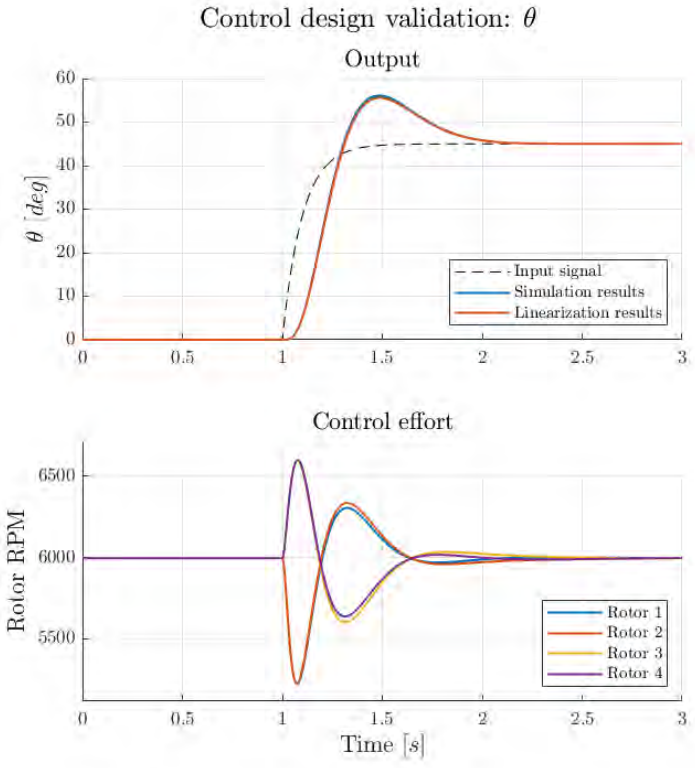


Figure 4: Pitch controller validation

A second order controller was used at this stage, as opposed to the first order roll controller, to achieve faster response – initial testing was limited to the straight trajectory (2D) case.

3.2.4. Horizontal velocity controller

Open loop:

$$H_{\theta_{com}}^{V_x}(s) = \frac{116000(s+3)}{(s+57)(s+7.9)(s+0.5)(s^2+15s+79)}$$

Chosen controller – PI:

$$H_{V_{x,error}}^{\theta_{com}}(s) = 0.22\frac{s+2}{s}$$

Closed loop:

$$H_{V_{x,error}}^{V_x}(s) = 25700\frac{(s+3)(s+2)}{(s+57)(s+14)(s^2+3.1s+4.2)(s^2+6.3s+45)}$$

Closed loop characteristics in the frequency domain:

$$GM = 11.2\ [dB] \quad \left(\omega = 9.43\ \left[\frac{rad}{s}\right]\right)\ ,\ PM = 54.1° \quad \left(\omega_c = 3.22\ \left[\frac{rad}{s}\right]\right)$$

In the time domain:

$$PO = 19.75\%\ ,\ t_s^{2\%} = 2.272\ [s]\ ,\ type\ I$$

Validation using step input, smoothed with a prefilter to avoid unrealistic control effort as a pure step input leads to a large initial spike in rotor angular velocity:

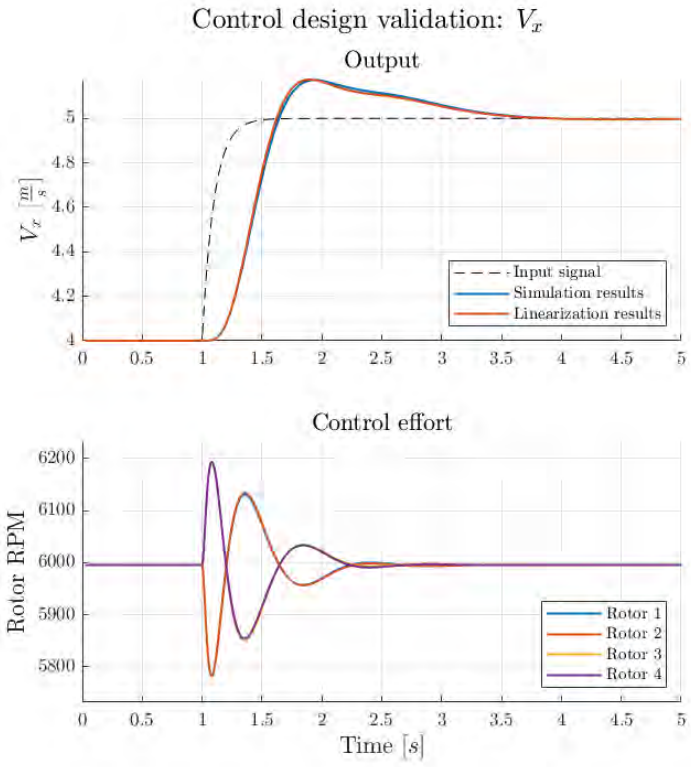


Figure 5: Horizontal velocity controller validation

3.2.5. Vertical acceleration controller

Open loop:

$$H_T^{a_z}(s) = \frac{21s}{(s+10)(s+0.5)}$$

Chosen controller – double PI:

$$H_{a_{z,error}}^{T}(s) = 0.4\frac{(s+1)(s+4)}{s^2}$$

Closed loop:

$$H_{a_{z,error}}^{a_z}(s) = 8.4\frac{(s+1)(s+4)}{(s+16)(s^2+2.8s+2.1)}$$

Closed loop characteristics in the frequency domain:

$$GM = \infty \ , \ PM = 109° \quad \left(\omega_c = 4.76\left[\frac{rad}{s}\right]\right)$$

In the time domain:

$$Overshoot = 3.26\% \ , \ t_s^{2\%} = 2.67\ [s] \ , \ type\ I$$

Validation using step input, where the rotors do not reach steady state as it takes an increasing amount of thrust to overcome drag at higher speeds:

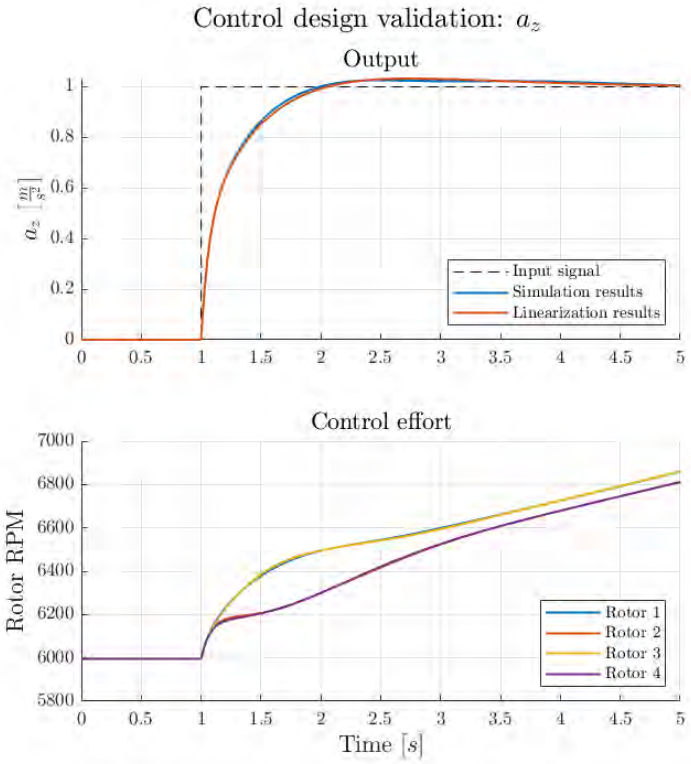


Figure 6: Vertical acceleration controller validation

As this controller is to serve the guidance algorithm for height tracking, the overshoot was made sure to be kept low, and rise time short.

3.3. Final validation

After having closed all loops, system response and control effort were tested first for a hand-built, irregular vertical acceleration command. Afterwards, random vertical acceleration and forward velocity signals were input. In both cases the signals were tracked well and the control efforts exerted were reasonable.

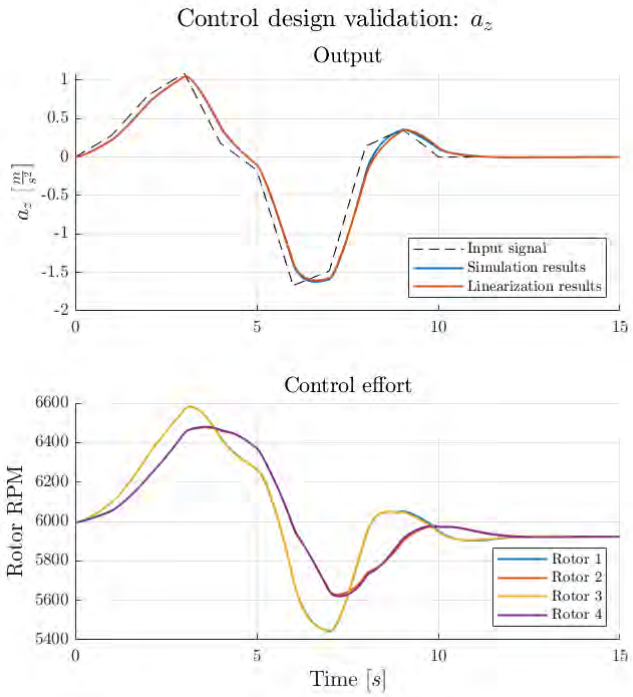


Figure 7: Final validation – hand-built signal

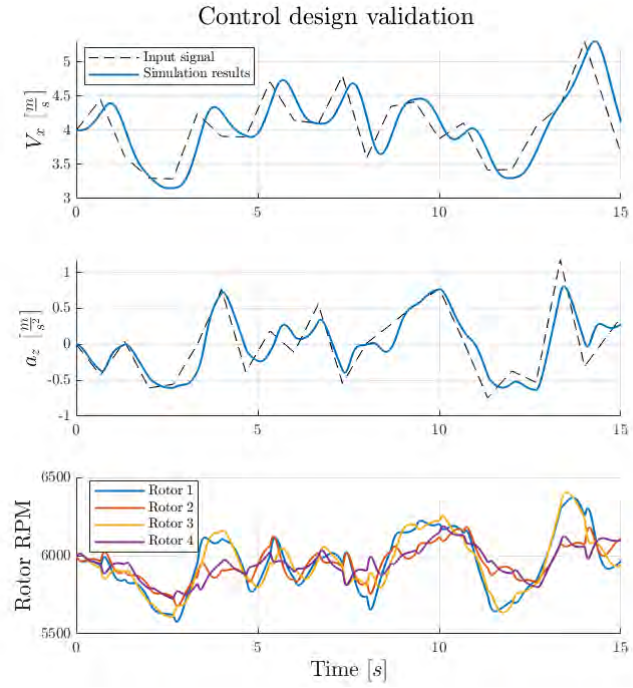


Figure 8: Final validation – random signal

4. Terrain generation

The improved Perlin noise algorithm [5] was chosen to be used for random generation of the terrain, as it is easy to implement, customize, and it is widely used to generate organic-seeming noise. The algorithm can theoretically be generalized to generate infinite terrain of the same characteristics as needed during simulation, though this feature was not implemented.

In this work Perlin noise is used to generate a matrix of height (z) values corresponding to evenly spaced xy coordinates. A small shortcoming of this method is the inability to model overhangs, though this was deemed to be of little consequence for the purpose of this research.

In order to achieve complex terrain, several noise maps are generated. Those maps differ by the scale of the noise, often called the frequency of the noise: lower frequency noise results in larger structures with lower detail, and vice versa. Variation of the frequencies used, as well as simple mathematical operations on the resulting heightmap, allow customization of the generated terrain. A sample generated heightmap can be seen in Figure 9 (colors chosen for dramatic effect).

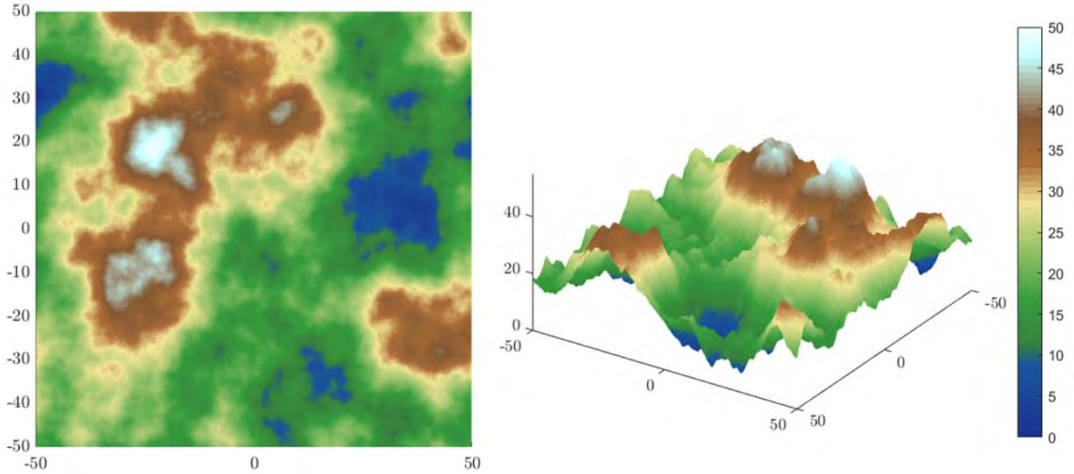


Figure 9: Example of generated heightmap

5. LRF simulation

In order to simulate the measurement of an LRF, whose laser ray is modeled as a vector that originates at the LRF (temporal) location and points in a given direction relative to the inertial frame, a ray-mesh intersection calculation algorithm was implemented based on the work of Sjöstrand T. [6]. The calculation steps are outlined in the following subsections.

5.1. LRF model

The position of an LRF is defined in the body frame as $[x, y, z]_{LRF,B}{}^{T}$, and its pointing direction by two Euler angles θ_{LRF} and ψ_{LRF} (elevation and azimuth respectively). The inertial position of the LRF at a given time step t is then given by:

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}_{LRF,E} = \boldsymbol{D}_E^B(t) \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{LRF,B} + \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}_{v,E} \tag{7}$$

where $\boldsymbol{D}_E^B(t)$ is the inertial-to-body rotation matrix at time t, calculated using the vehicle's own Euler angles, subscripts $(\cdot)_{\cdot,B}$ and $(\cdot)_{\cdot,E}$ denote body and inertial frames respectively, and subscript $(\cdot)_{v,\cdot}$ denotes the vehicle. The LRF's pointing direction in the body frame is the forward unit vector $\hat{x}_{B,B}$ rotated by θ_{LRF} (about the y_B axis) and ψ_{LRF} (about the z_B axis) in a z-y order, and is given by:

$$\hat{x}_{LRF,B} = \boldsymbol{R}_y(\theta_{LRF})\boldsymbol{R}_z(\psi_{LRF})\hat{x}_{B,B} \tag{8}$$

where $\boldsymbol{R}_{(\cdot)}$ is the matrix corresponding to a rotation about the subscripted body axis by the given angle. The LRF's pointing direction in the inertial frame is thus:

$$\hat{x}_{LRF,E}(t) = \boldsymbol{D}_E^B(t)\hat{x}_{LRF,B} \tag{9}$$

5.2. Quad tree search

Assuming the mesh to be intersected is defined as a heightmap, meaning it is represented by height values corresponding to a linearly spaced xy grid, a quad tree may be defined as follows. Note, the above definition allows for heightmaps to be generated in any fashion, meaning the Perlin noise algorithm may be changed if need be. It is even possible to incorporate this algorithm to be used on an actual digital terrain map (DTM) readily available on line for various areas on the globe and at different levels of accuracy.

- A quad is defined as a square bounding a subset (or the entirety) of the xy heightmap grid, where its vertices lie on points of the grid. Its size is the length of its side in units of the distance between two consecutive grid points.
- The first quad is defined to bound the entire xy grid.
- Four sub-quads are defined such that they do no overlap and their union bounds the entirety of the previous quad.
- Repeat the above bullet for every quad defined, until reaching a quad of size one or a user-determined limit size.

For every quad in the quad tree, a corresponding bounding box for the section of the terrain encased in it is defined, essentially extruding the quad to reach the minimum and maximum heights in its domain. An example of the above recursive definition can be seen in Figure 10, where an LRF measurement is fully simulated and only the quads through which it passes are drawn. As the sides of these bounding boxes are parallel to the heightmap axes, it is easy to determine whether a given ray passes through it or not. Simple math allows efficient pruning of the quad tree, marking the smallest quads defined through which the ray passes, which are also easy to sort by the order in which the ray enters them.

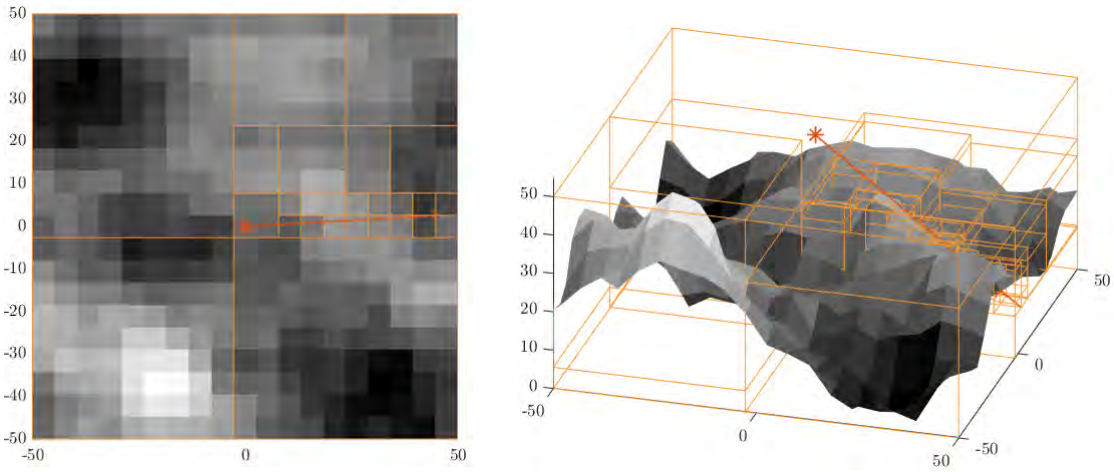


Figure 10: Example of recursive definition of quads

The above steps allow for a great reduction in low-level inspections of intersections with the terrain; the quad tree search is an analog of a binary search tree for the two-dimensional search case.

5.3. Ray-triangle intersections

After having found the minimal set of quads within which an intersection with the mesh might occur, low-level checks are performed. Considering the mesh defined by the heightmap is built of – and rendered graphically with – triangles whose vertices match the grid points (marked pink and turquoise in the following figure), testing whether the ray passes through each triangle is geometrically simple. Marching along the quads in the order the ray intersects them, and along the triangles within them similarly, guarantees the first ray-terrain intersection found is indeed the first point where the ray reaches the mesh. Thus, the marching process begins at an LRF's location as given by equation (7) and continues in its pointing direction as given by equation (9).

Figure 11 presents an example of a ray marching along sorted quads of size one, where the pink and turquoise triangles are the two triangles forming the surface within each quad that is checked until reaching the first intersection point.

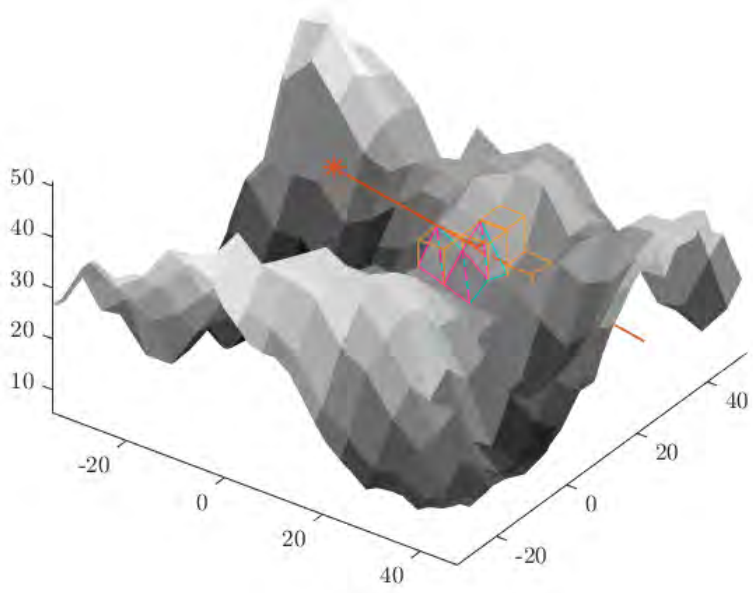


Figure 11: Example of quad and triangle marching

5.4. Measured range

The range measured by the LRF, assuming an ideal measurement, is calculated as the distance between its origin point and the intersection point. After having calculated the LRF's laser's intersection point with the mesh, denoted $[x(t), y(t), z(t)]_{g,E}{}^{T}$, the range measurement is given by:

$$d = \left\| \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}_{LRF,E} - \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}_{g,E} \right\| \tag{10}$$

In Figure 12, a simulation of 15 seconds of constant altitude flight (relative to the inertial frame, not to the ground beneath) is visualized at several points in time, where three LRFs are installed with Euler angles $\psi = -10°, 0, 10°$ and $\theta = 45°$. The quadcopter is indicated by black lines and the laser-heightmap intersection points by red x marks.

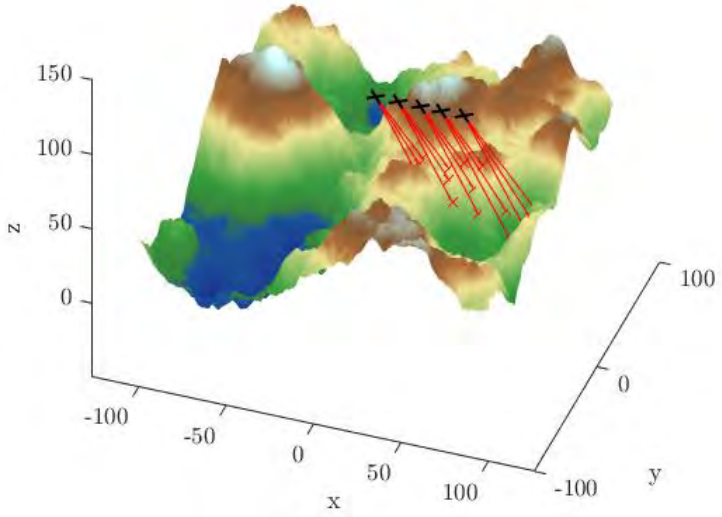


Figure 12: Example multiple LRF simulation

6. Guidance

The implemented guidance algorithm is a simplified version of the trajectory shaping method proposed by Ratnoo et al. [2].

The 3D trajectory supplied to the algorithm is a predefined 2D trajectory (in the horizontal, inertial plane) projected onto the terrain and raised to a constant desired altitude H above it. In practice, the trajectory used by the vehicle is recalculated in this manner in real-time with every new measurement, to simplify the design and avoid the requirement of onboard memory storage and processing. This is done to test the assumption that real-time measurements alone are sufficient for ground estimation, and thus trajectory estimation, in the relevant area ahead of the vehicle.

For the sake of simplification of real-world implementation, an attempt was made to rely on as partial of a navigation solution as possible for the guidance itself. Where the original algorithm calls for the position of a virtual target relative to the vehicle, which is integrated from its velocity along the required trajectory, and both the vehicle's and target's velocity directions, it was eventually decided to place the target at a constant, nominal distance from the vehicle and not integrate its position.

6.1. Ground and trajectory estimation

The LRF measurements are used to estimate ground height in the vicinity of the measured points, for calculation of the virtual target's 3D position and velocity to be fed to the guidance algorithm. Several surface fitting methods were tested against one another, of which the following were the most prominent:

- Cubic interpolation.
- X and Y polynomials.
- Biharmonic splines.
- Thin-plate splines.

The following attributes constituted the main considerations in choosing a method:

- Interpolation accuracy, in which no one method seemed significantly more suitable in tested scenarios.
- Extrapolation ability and accuracy, so that the estimation of the ground could be robust in case the desired 2D trajectory does not intersect with the (2D, top-view) convex hull of measured ground points.
- Minimal number of points required, to allow for guidance without storing previously measured points in memory (thus always working with a small dataset), and to preserve the ability to test the performance of few LRFs.

The following table compares the four methods according to the above considerations, omitting interpolation as the methods exhibited similar behavior in this regard.

Table 3: Surface fitting method comparison for ground estimation

	Cubic interpolation	**x and y polynomials**	**Biharmonic splines**	**Thin-plate splines**
Extrapolation	Triangulation-based*	Diverges faster as with polynomial degrees increase	Tends to diverge aggressively	Tends to diverge slowly
Number of points	3 points	Increases with polynomial degrees	2 points	3 points

As per the above, thin-plate splines were chosen.

After having estimated the ground surface, the desired 2D trajectory is projected onto it and raised to height H above it. The raised, 3D trajectory is fed into the guidance algorithm, where a virtual target is placed on it to be followed.

The following figure visualizes the above process. The red lines represent LRF lasers, terminating with red dots at the measured ground points. The white, gridded surface represents the estimated ground in vicinity of the measured points (some of which is extrapolated), above which are shown the desired 3D trajectory in a black dashed line, and the estimated 3D trajectory in a purple dashed line.

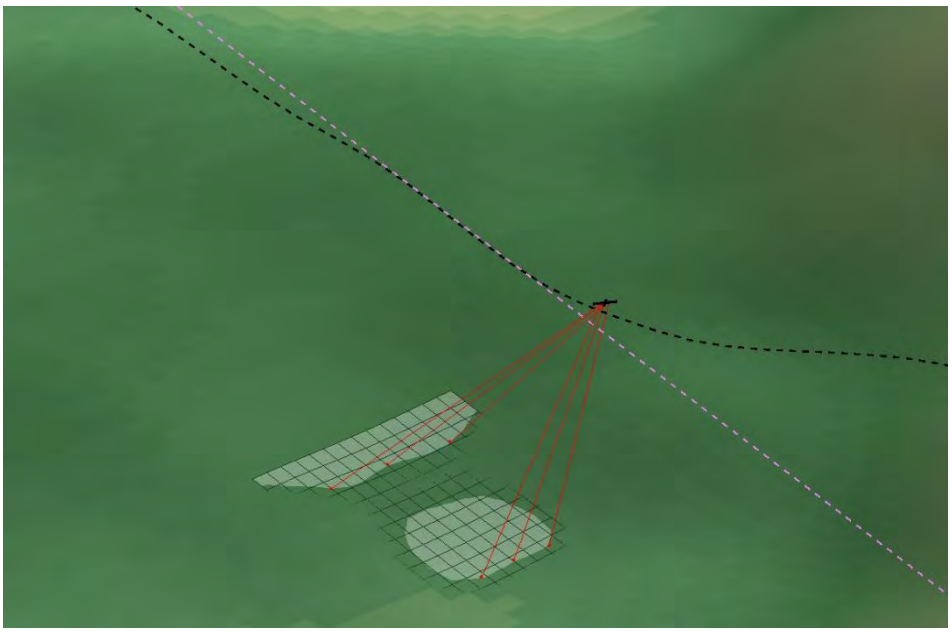


Figure 13: Example of ground and trajectory estimation

* Triangulation can cause poor results near the edge of the convex hull of the data in cases where some of the data points are inside the convex hull, and not on its edge. This method does not extrapolate.

6.2. Guidance algorithm

The original trajectory shaping guidance algorithm makes use of the following equation for the lateral acceleration command:

$$a_{cmd} = \frac{V_d^2}{R}\left(4(\lambda - \gamma_v) + 2(\lambda - \gamma_t)\right) \tag{11}$$

where a_{cmd} is the acceleration command, V_d the desired vehicle velocity, R is the distance between the virtual target and the vehicle, and λ, γ_v and γ_t the line-of-sight (LOS) angle, vehicle heading angle and target heading angle relative to some arbitrary inertial coordinate frame, respectively. The command is computed in the plane containing the vehicle velocity and LOS vectors, and is perpendicular to the vehicle velocity.

The above calculation is performed in the horizontal and vertical planes separately to receive the xy and z components of the acceleration command. This yields the following vertical command:

$$a_v^v = \frac{V_d^2}{R}\left(4(\theta_l - \theta_v) + 2(\theta_l - \theta_v)\right) \tag{12}$$

$$\theta_l = \arctan\left(\frac{z_t - z_v}{R_{xy}}\right) \;,\; \theta_v = \arctan\left(\frac{V_{v_z}}{V_{v_{xy}}}\right) \;,\; \theta_t = \arctan\left(\frac{V_{t_z}}{V_{t_{xy}}}\right)$$
$$R = \sqrt{(x_t - x_v)^2 + (y_t - y_v)^2 + (z_t - z_v)^2} \tag{13}$$

where $\theta_{(\cdot)}$ signifies an elevation angle, $x_{(\cdot)}$, $y_{(\cdot)}$ and $z_{(\cdot)}$ inertial coordinates, and $V_{(\cdot)}$ velocity, and subscripts l, v, and t are LOS, vehicle and target, respectively. Similarly, the vertical command is:

$$a_v^h = \frac{V_d^2}{R_{xy}}\left(4(\psi_l - \psi_v) + 2(\psi_l - \psi_t)\right) \tag{14}$$

$$\psi_l = \arctan\left(\frac{y_t - y_v}{x_t - x_v}\right) \;,\; \psi_v = \arctan\left(\frac{V_{v_y}}{V_{v_x}}\right) \;,\; \psi_t = \arctan\left(\frac{V_{t_y}}{V_{t_x}}\right)$$
$$R_{xy} = \sqrt{(x_t - x_v)^2 + (y_t - y_v)^2} \tag{15}$$

where $\psi_{(\cdot)}$ signifies an azimuth angle. Note, the calculation is presented in an inertial frame but is not limited to it.

The algorithm requires the following data:

- Target position relative to vehicle.
- Vehicle and target velocity directions.

The following attempts were made at simplifying the algorithm:

- No memory: the supplied trajectory estimation is recalculated with every new measurement, discarding previously measured points. This negates the need to store and process a large amount of points, at the cost of estimation accuracy. As it did not seem to impact the guidance performance significantly, this method was used.

- Limited acceleration control: acceleration in the y_B and z_B axes is controlled, but in the x_B direction velocity is stabilized, and acceleration not controlled. Assuming the $(xy)_B$ plane velocity is always in the x_B direction, the 3D acceleration command calculated by the original algorithm will require no x_B acceleration component, as it would be parallel to the velocity. This method was indeed implemented, as the vehicle remained stable and tracked tested trajectories with only these two controlled acceleration components.
    - In order to assume the above, another control loop is used to drive V_y^B to zero using ψ_{com}, meaning the multirotor yaws such that its velocity in the $(xy)_B$ plane is kept predominantly in the x_B direction.
- Target movement: rather than setting the target's velocity and integrating its position along the estimated trajectory, it is placed at a predefined nominal distance ahead of the vehicle on the estimated 3D trajectory in each time step. Assuming no memory as in the first bullet, this also helps avoid positional ambiguities of the target when the estimated trajectory is changed in every time step. This method was used as well, as stability and tracking were achieved with it.
- Partial data: initially, only gyroscope measurements (angular rates and the angles integrated from them) were to be used. As explained further on under *Guidance implementation*, testing revealed vehicle velocity information is also necessary for stability. This method was thus the only discarded simplification. Note, the control loops require a full navigation solution, regardless of the implementation of this simplification.

6.3. Added control

Two new controllers were designed to support the guidance algorithm: one for lateral velocity V_y^B controlled using ψ_{com}, as explained in subsection 6.2, and one for lateral acceleration a_y^B controlled using ϕ_{com}, serving a similar purpose to the vertical acceleration controller. The controllers were designed in conjunction with the implementation and testing of the guidance algorithms, and were ultimately validated via correct behavior of the system as a whole. As in chapter 3, the control loops were closed consecutively, such that each open loop given in the following subsections takes previous controllers into account.

6.3.1. Lateral velocity controller

Open loop:

$$H_{\psi_{com}}^{V_y^B}(s) =$$

$$= \frac{-99.78 \times 10^{-3} s(s+5621)(s+3.665)(s^2+0.7132s+1.891)}{(s+12.1)(s+2.237)(s+1.369)(s^2+2.824s+3.169)(s^2+3.545s+5.795)}$$

Note: the negative sign arises from the body frame definition; if V_y^B is positive, increasing ψ will yaw the quad into the direction of the $(xy)_B$ velocity, thus decreasing its y_B component.

Chosen controller – double PI with lead compensator:

$$H_{V_{y,error}^B}^{\psi_{com}}(s) = -0.3\frac{(s+1.369)(s+2.237)(s+1.5)}{s^2(s+15)}$$

Closed loop:

$$H_{\psi_{com}}^{V_y^B}(s) =$$

$$= \frac{29.93\times10^{-3}(s+5621)(s+3.665)(s+1.369)(s^2+0.7132s+1.891)}{(s+17.02)(s+9.118)(s+0.7495)(s^2+1.596s+1.244)(s^2+4.988s+12.09)}$$

Closed loop characteristics in the frequency domain:

$$GM = 29.9\,[dB] \quad \left(\omega = 14.1\left[\frac{rad}{s}\right]\right) \;,\; PM = 80.5° \quad \left(\omega_c = 0.489\left[\frac{rad}{s}\right]\right)$$

In the time domain:

$$Overshoot = 0 \;,\; t_s^{2\%} = 5.284\,[s] \;,\; type\ I$$

6.3.2. Lateral acceleration controller

Open loop:

$$H_{\phi_{com}}^{a_y^B}(s) =$$

$$= \frac{-1272.9(s+0.7)(s^2+0.5831s+0.3076)(s^2+3.424s+5.033)}{(s+11.8)(s+2.25)(s^2+1.171s+0.5492)(s^2+1.785s+1.205)(s^2+4.884s+11.81)}$$

Note: the negative sign arises from the body frame definition; if a_y^B is positive, increasing ϕ will roll the quad away the direction of the body xy acceleration, thus decreasing its y component.

Chosen controller – double PI with lead compensator:

$$H_{a_{y\,error}^B}^{\phi_{com}}(s) = -1.3\frac{(s+2.25)(s+3)}{s(s+30)}$$

Closed loop:

$$H_{\phi_{com}}^{a_y^B}(s) = \frac{1654.8(s+3)}{(s+32.46)(s+4.618)(s^2+8.447s+53.52)}$$

Closed loop characteristics in the frequency domain:

$$GM = 19.1\,[dB] \quad \left(\omega = 19\left[\frac{rad}{s}\right]\right) \;,\; PM = 70.4° \quad \left(\omega_c = 4.65\left[\frac{rad}{s}\right]\right)$$

In the time domain:

$$Overshoot = 1.170\% \;,\; t_s^{2\%} = 4.528\,[s] \;,\; type\ I$$

6.4. Guidance implementation

The implemented algorithm makes use of equations (12)-(15), adjusted for use in body axes rather than an inertial frame, such that the coordinates of the vehicle are $x_v^B = y_v^B = z_v^B = 0$. As a result, the elevation and azimuth angles – $\theta_{(\cdot)}$ and $\psi_{(\cdot)}$, respectively – need to be replaced with analogous ones expressed in relation to the body frame, set apart using superscript $(\cdot)^B$. The required angles are thus:

$$\psi_l^B = \arctan\left(\frac{y_t^B}{x_t^B}\right) \ , \ \psi_v^B = \arctan\left(\frac{V_{v_y}^B}{V_{v_x}^B}\right) \ , \ \psi_t^B = \arctan\left(\frac{V_{t_x}^B}{V_{t_y}^B}\right)$$

$$\theta_l^B = \arctan\left(\frac{z_t^B}{R_{xy}^B}\right) \ , \ \theta_v^B = \arctan\left(\frac{V_{v_z}^B}{V_{v_{xy}}^B}\right) \ , \ \theta_t^B = \arctan\left(\frac{V_{t,z}^B}{V_{t_{xy}}^B}\right) \tag{16}$$

$$R_{xy}^B = \sqrt{{x_t^B}^2 + {y_t^B}^2} \ , \ V_{v_{xy}}^B = \sqrt{{V_{v_x}^B}^2 + {V_{v_y}^B}^2} \ , \ V_{t_{xy}}^B = \sqrt{{V_{t_x}^B}^2 + {V_{t_y}^B}^2}$$

An attempt was made to reduce the measured data required to angular data only, by setting the vehicle velocity direction to the nominal steady-state direction when following a straight and level trajectory:

$$\psi_v^B = 0 \ , \ \theta_v^B = -\theta_{trim} \tag{17}$$

The result was a slowly diverging oscillation in both the vertical and horizontal directions for very straight and level trajectories, and a quickly diverging one for any practical trajectories. It was concluded that the vehicle velocity terms provided necessary damping to the resulting dynamics. Consequently, the velocity data cannot be discarded and is assumed to be measured for guidance purposes.

The desired 2D trajectory is assumed to be known in body coordinates at every time frame, such that its conversion from inertial to body frames need not happen onboard. This assumption was made as another attempt (along with those mentioned in subsection 6.2) to use as partial a navigation solution as possible. Should the trajectory be provided in inertial coordinates alone, not only would a full navigation solution be needed, but it would likely require GPS or vision aid.

7. Preparation for analysis

In order to simulate a real-world environment, the sensor, estimation, guidance and control components of the simulation were discrete with a sample time of 0.01 $[s]$ (meaning the simulated hardware is operated at $100\ [Hz]$), and the plant was simulated as continuous. The controllers were discretized accordingly, and Gaussian measurement noise was added to all measurements and LRF positions and angles, the latter representing installation errors. Additionally, in preparation for statistical and parametric analysis of the guidance algorithm's performance, several applications were built to wrap the existing software:

- A simulation execution tool, in which chosen parameters and settings can be set, and single or Monte Carlo simulations run.
- A single simulation analysis tool.
- A Monte Carlo simulation analysis tool.

The applications were built with future modification in mind, and the simulation and codebase adjusted accordingly.

7.1. Control discretization

The continuous controllers presented in subsections 3.2 and 6.3 were converted to the following discrete controllers, operating at $100\ [Hz]$:

- Roll:

$$H^{\tau_\phi}_{\phi_{error}}(z) = 0.061\frac{z-0.993}{z-0.932}$$

- Yaw:

$$H^{\tau_\psi}_{\psi_{error}}(z) = 5.627\frac{(z-0.993)(z-0.958)}{(z-0.861)(z-0.779)}$$

- Pitch:

$$H^{\tau_\theta}_{\theta_{error}}(z) = 5.755\frac{(z-0.984)(z-0.869)}{(z-0.741)(z-0.607)}$$

- Forward velocity:

$$H^{\theta_{com}}_{V_{x,error}}(z) = 0.221\frac{z-0.980}{z-1}$$

- Vertical acceleration:

$$H^{T}_{a_{z,error}}(z) = 0.4\frac{(z-0.990)(z-0.960)}{(z-1)^2}$$

- Sideways velocity:

$$H^{\psi_{com}}_{V^B_{y,error}}(z) = -0.3\frac{(z-0.989)(z^2-1.963z+0.963)}{(z-1)^2(z-0.861)}$$

- Sideways acceleration:

$$H_{a_{y,error}^{B}}^{\phi_{com}}(z) = -1.3\frac{z^2 - 1.954z + 0.955}{(z-1)(z-0.741)}$$

7.2. Measurement noise and installation error

In order to perform statistical analysis further on, Gaussian noise was added to the following measurements at each discrete measurement instant, some of which are unused at the current stage and were included to allow for future changes more easily:

- Body acceleration
- Body velocity
- Inertial position
- Euler rates
- Euler angles
- LRF measurements

Normally distributed error was also added to the installation position and angles of the LRFs at the beginning of each simulation run.

7.3. Simulation and analysis tools

In preparation for future analytical work, several applications were constructed:

- A simulation tool to simplify the process of simulation execution, including the setting of important parameters.
- A single simulation analysis tool, providing plots relating to the vehicle's state, control signals and responses, and guidance performance.
- A Monte Carlo simulation analysis tool, providing plots of raw or statistical simulation data to do with guidance performance.

7.3.1. Simulation tool

The simulation tool is built to allow for simple execution of Monte Carlo or single simulations.

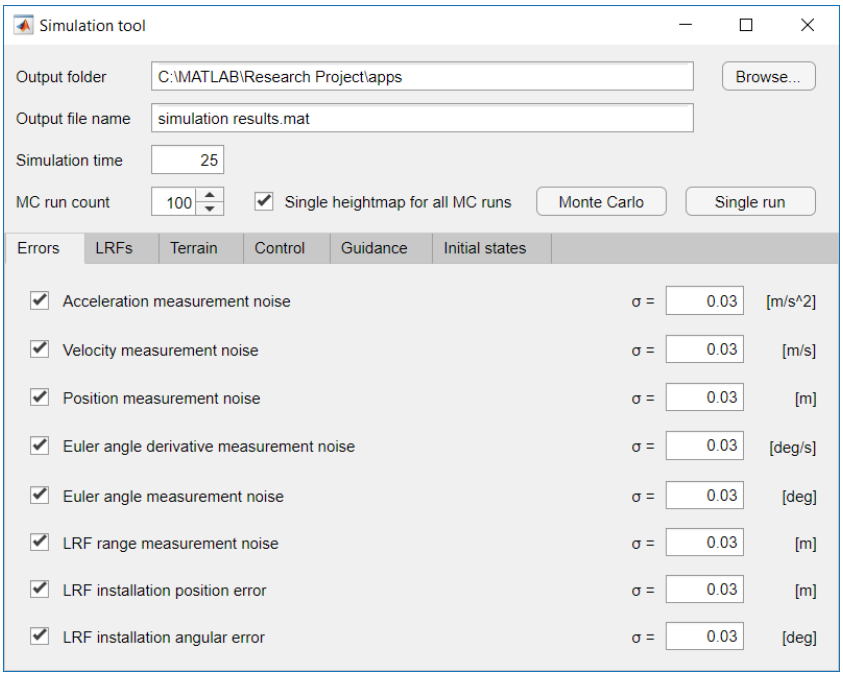


Figure 14: Simulation tool startup view

The following parameters and settings can be set prior to running a simulation using the application:

- General
  - The output folder and name of the file in which to save the results.
  - The maximum time to be simulated in each run.
  - The amount of runs of which a Monte Carlo simulation will consist.
  - Whether one heightmap will be generated for all Monte Carlo runs or a new one created for each run.
- Errors
  - Noise for each measurement or installation error can be toggled on or off.
  - The standard deviation of the Gaussian noise for each measurement or installation error can be set separately.
- LRFs
  - The number of LRFs.
  - The position of each LRF in body coordinates, and its pitch and yaw relative to the body x axis.
- Terrain
  - The side length of the generated terrain.
  - The number of mesh points along the x and y axes.
  - The maximal height of the terrain.
  - The base frequency and number of octaves for the generation of Perlin noise.
- Control
  - All controllers can be redefined.

- Guidance
  - The desired altitude to be kept above the ground.
  - The distance at which the virtual target will be placed on the desired trajectory.
  - The desired 2D trajectory, specified as a series of (x, y) waypoints in inertial coordinates.
- Initial states
  - Inertial position.
    - The z value can be specified either in inertial coordinates or in altitude above the generated ground in the initial (x, y) position.
  - Inertial velocity.
  - Euler angles.
  - Angular rates (pqr).
  - Rotor velocities.

7.3.2. Single simulation analysis tool

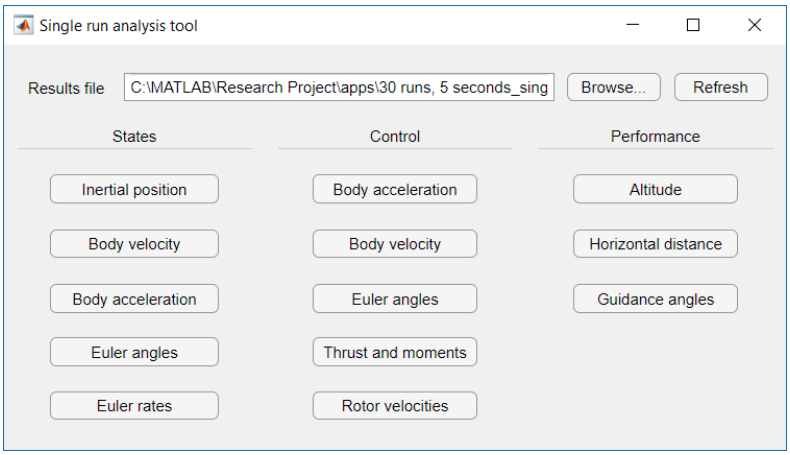


Figure 15: Single simulation analysis tool

The analysis tool provides the following plots for a specified results file from a single simulation:

- States as a function of time:
  - Inertial position.
  - Body velocity.
  - Body acceleration.
  - Euler angles.
  - Euler rates.
- Control commands vs. responses as a function of time:
  - Body acceleration.
  - Body velocity.
  - Euler angles.
  - Thrust and moments.
  - Rotor velocities.

- Performance
    - Altitude above ground, altitude error relative to the desired height and a section view of the vehicle altitude and ground height as a function of time.
    - Horizontal error from the desired trajectory as a function of time and a top view of the quad's horizontal position relative to the desired 2D trajectory.
        - Horizontal error is currently defined as the shortest distance from the vehicle to the desired 2D trajectory on the inertial xy plane, at a given time step. This definition results in a non-negative error term as only absolute distance is considered, and is likely to change in future work.
    - The LOS, vehicle velocity and virtual target velocity's pitch and yaw angles, as presented under *Guidance implementation*, as a function of time.

7.3.3. Monte Carlo simulation analysis tool

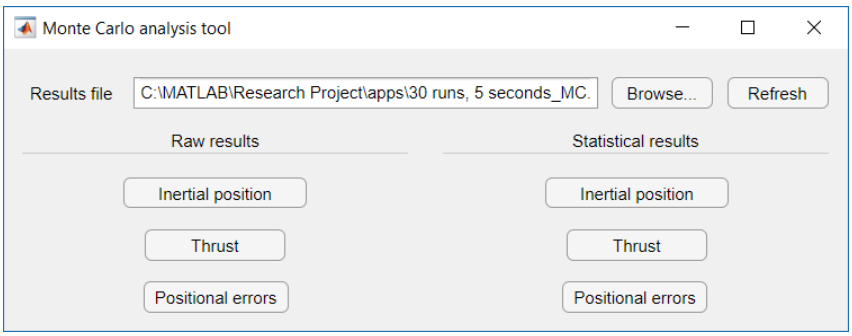


Figure 16: Monte Carlo simulation analysis tool

For a specified results file from a Monte Carlo simulation, the analysis tool provides plots of data from all runs overlaid on one another, and plots of the mean and one standard deviation surrounding it, for the following:

- Inertial position.
- Total thrust.
- Vertical and horizontal errors relative to the desired trajectory.
    - The horizontal error cannot be analyzed as normally distributed as it is non-negative, meaning the mean and standard deviation values currently displayed for it are not useful for future work and will need to be replaced.

8. Example results

The results presented in this chapter are examples only, and were not used for any rigorous analytical work. They correspond to the following arbitrary parameters set in the simulation execution tool:

- Simulation time of $20\ [s]$.
- 30 single runs within the Monte Carlo simulation.
- Single heightmap generated for all Monte Carlo runs.
- Errors:
    - All measurement noises and installation errors are enabled.
    - The standard deviation for each noise and error is arbitrarily equal and is as follows:

$$\sigma_{a_B^{meas}} = 0.005\ \left[\frac{m}{s^2}\right]\ ,\ \sigma_{V_B^{meas}} = 0.005\ \left[\frac{m}{s}\right]\ ,\ \sigma_{(x,y,z)_E^{meas}} = 0.005\ [m]$$

$$\sigma_{\dot{\phi},\dot{\theta},\dot{\psi}^{meas}} = 0.005\ \left[\frac{deg}{s}\right]\ ,\ \sigma_{\phi,\theta,\psi^{meas}} = 0.005\ [deg]\ ,\ \sigma_{LRF_R^{meas}} = 0.005\ [m]$$

$$,\ \sigma_{LRF_{(x,y,z)_B}} = 0.005\ [m]\ ,\ \sigma_{LRF_{\theta,\psi}} = 0.005\ [deg]$$

- LRFs:
    - 6 LRFs are installed near the center of the quadcopter.
    - The LRF angles are spread evenly on a grid:

$$\psi_{LRF} \in [-10{,}10]^\circ\ ,\ \theta_{LRF} \in [30{,}55]^\circ$$

- Terrain:
    - $100\ [m]$ side length.
    - 200 mesh nodes along each horizontal axis.
    - $1\ [m]$ maximum ground height (relatively flat ground).
    - Base frequency 2 (approximately 2 large peaks and troughs along each horizontal axis).
    - 4 octaves.
- Control:
    - Default controllers, as described in section 3 and subsection 6.3.
- Guidance:
    - Virtual target distance from vehicle of $12\ [m]$.
    - Desired altitude of $15\ [m]$.
    - A custom trajectory, deviating only slightly from $x \in [-50{,}50]$, $y = 0$ in its first half or so, as is visualized in subsection 8.1.
- Initial states:
    - Initial position (edge of terrain, z measured above ground):

$$x = -50\ [m]\ ,\ y = 0\ ,\ z_{above\ ground} = 17\ [m]$$

    - Other initial states are all trim values.

8.1. Single simulation results

The following are examples of all plots supplied by the *Single simulation analysis tool*.

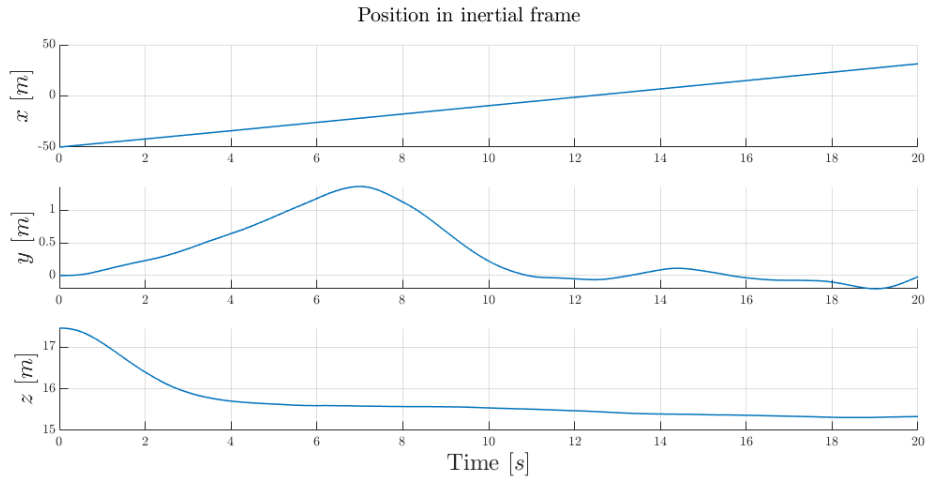


Figure 17: Single simulation results – inertial position

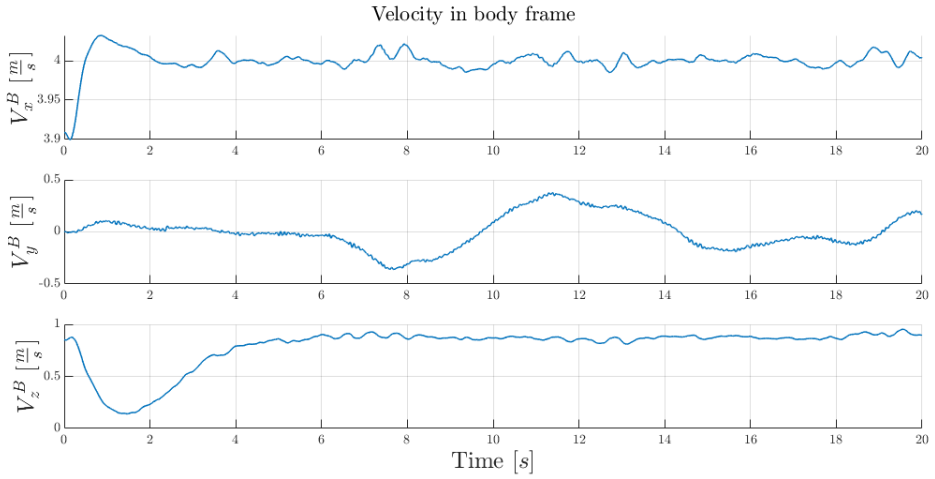


Figure 18: Single simulation results – body velocity

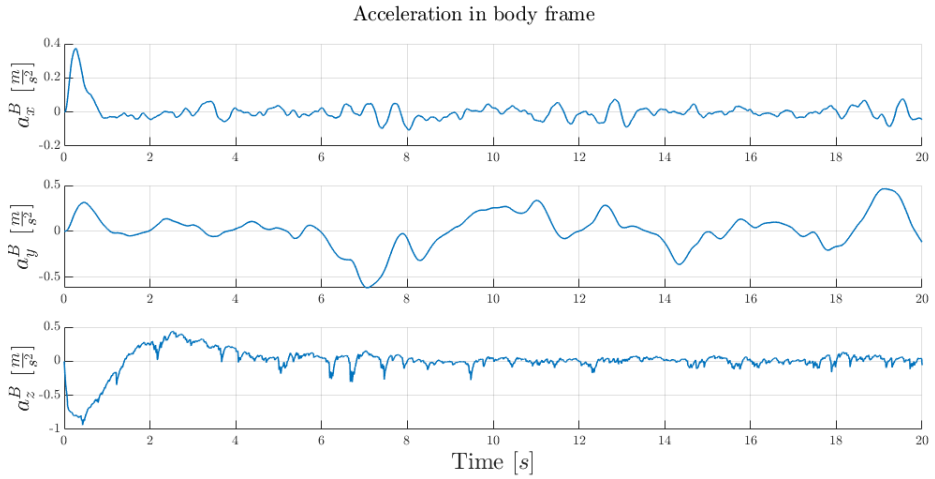


Figure 19: Single simulation results – body acceleration

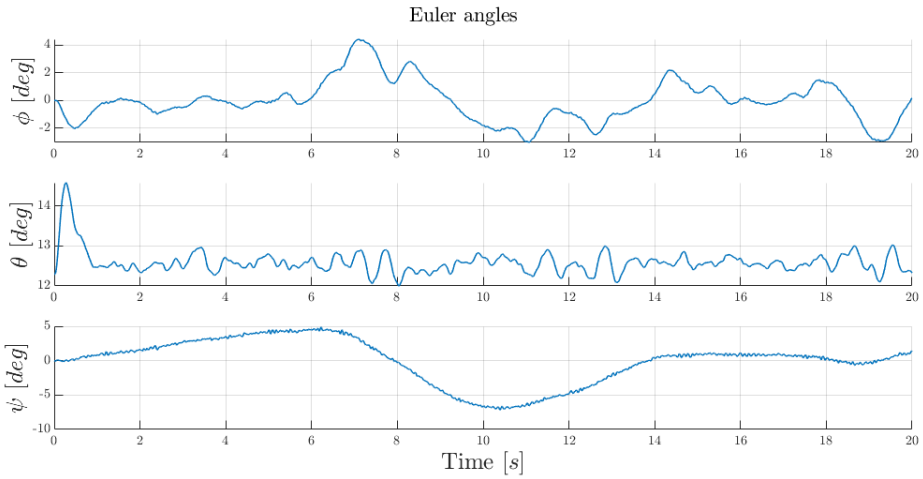


Figure 20: Single simulation results – Euler angles

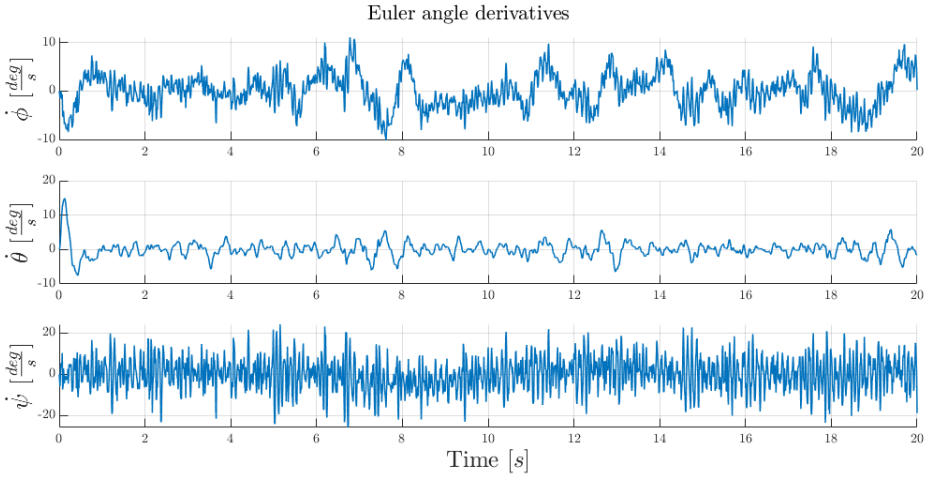


Figure 21: Single simulation results – Euler rates

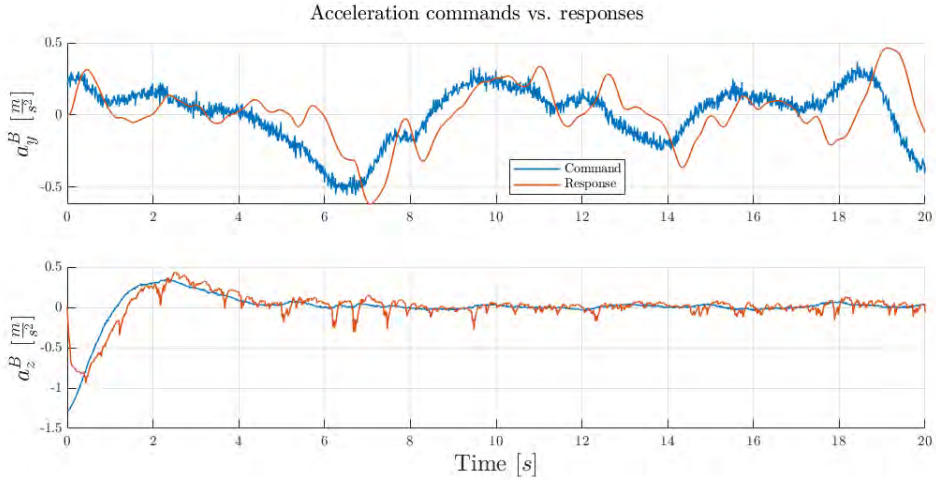


Figure 22: Single simulation results – acceleration commands and responses

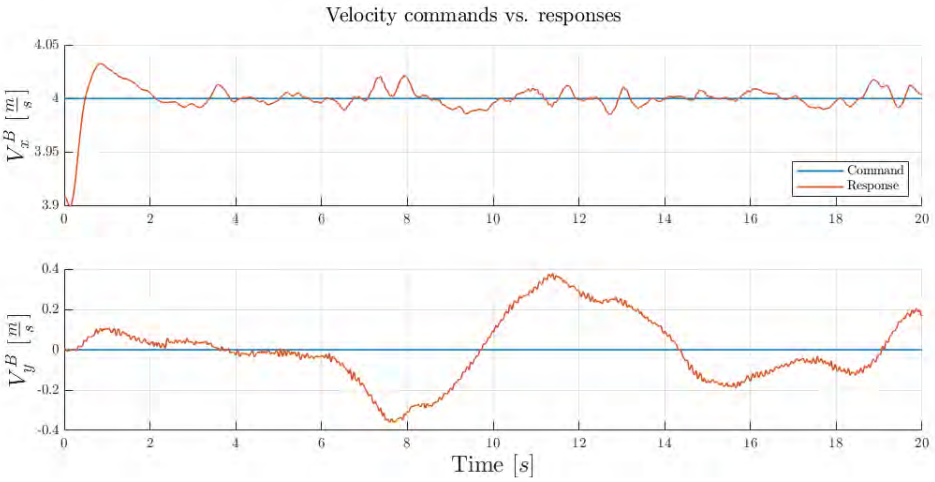


Figure 23: Single simulation results – velocity commands and responses

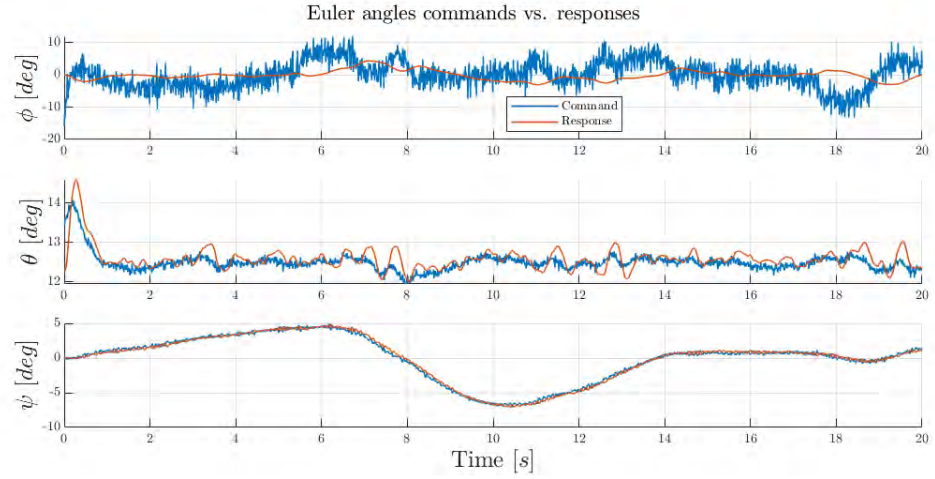


Figure 24: Single simulation results – Euler angle commands and responses

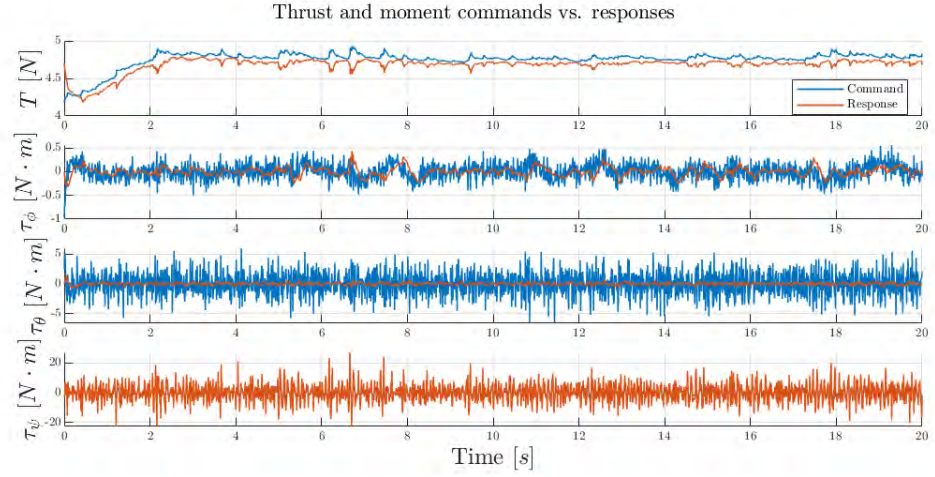


Figure 25: Single simulation results – thrust and moment commands and responses

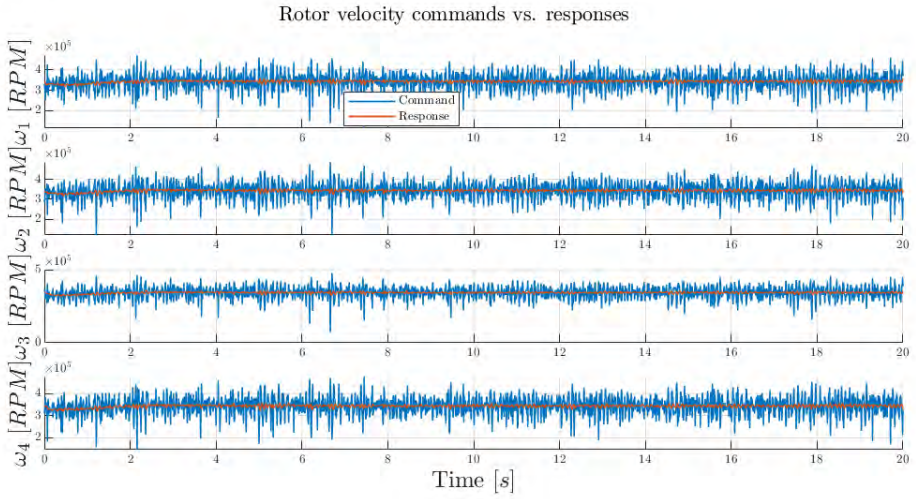


Figure 26: Single simulation results – rotor velocity commands and responses

8.2. Monte Carlo simulation results

The following are examples of all plots supplied by the *Monte Carlo simulation analysis tool*.

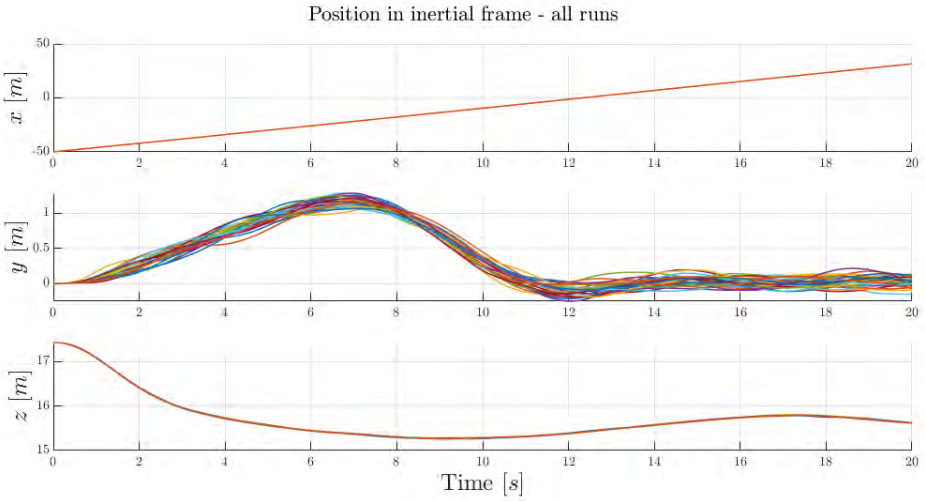


Figure 27: Monte Carlo simulation results – all run positions

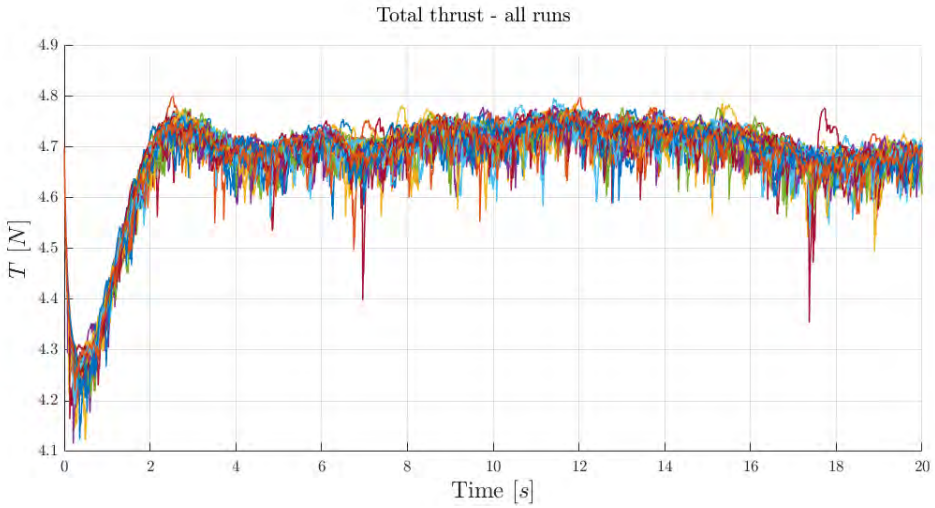


Figure 28: Monte Carlo simulation results – all run total thrusts

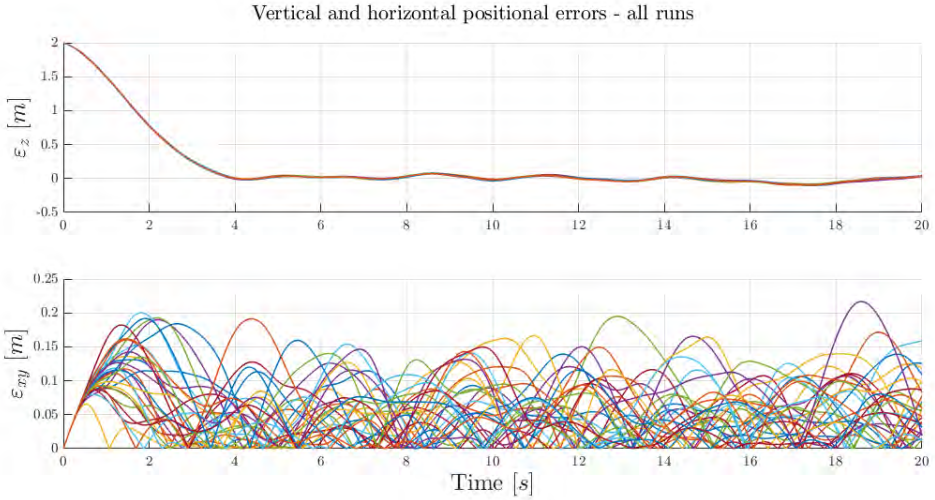


Figure 29: Monte Carlo simulation results – all run positional errors

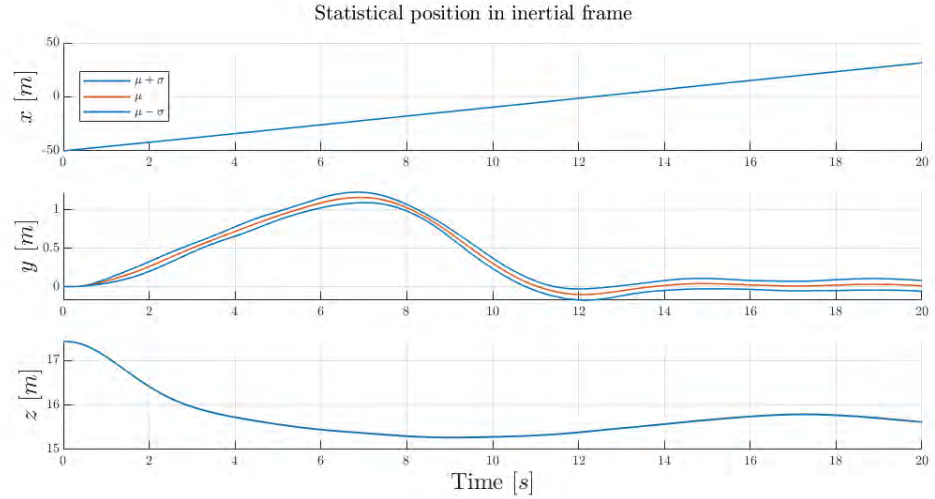


Figure 30: Monte Carlo simulation results – statistical position

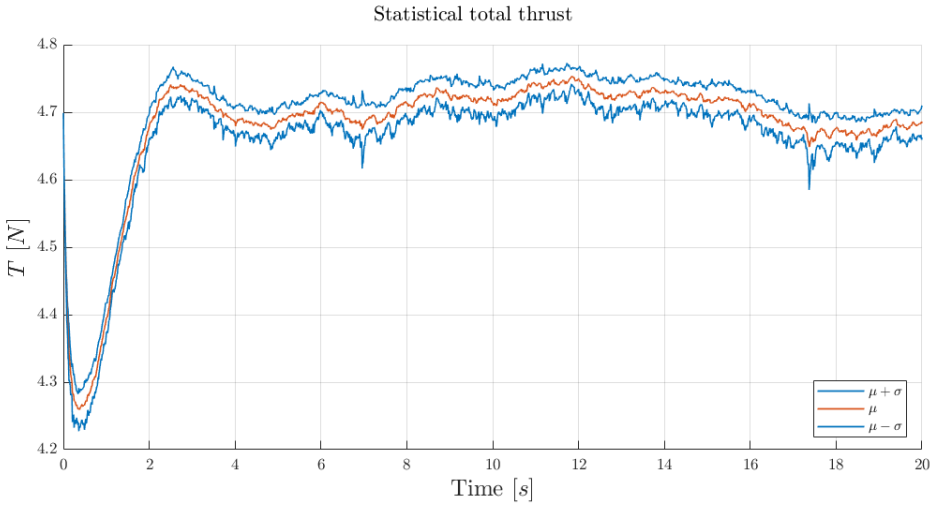


Figure 31: Monte Carlo simulation results – statistical total thrust

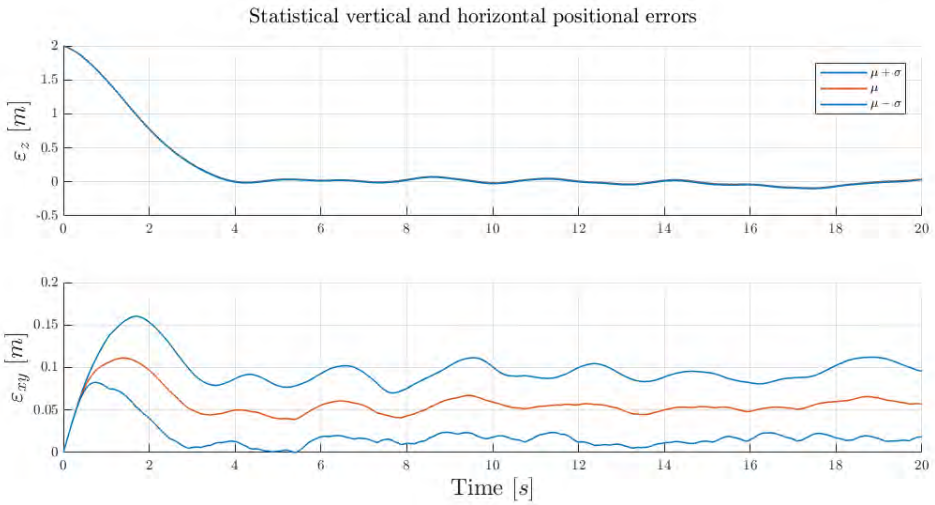


Figure 32: Monte Carlo simulation results – statistical positional errors

9. Conclusion

Over the course of two semesters a full simulation environment was developed, including:

- A quadcopter 6DOF model.
- Control design, consisting of:
    - A constant forward velocity controller, supplying commands to an elevation angle controller.
    - A controller driving sideways velocity to zero, supplying commands to an azimuth angle controller.
    - A horizontal acceleration controller, supplying commands to a roll angle controller.
    - A vertical acceleration controller, making use of all four rotors simultaneously.
- Customizable random terrain generation using Perlin noise.
- Processing of terrain into a quad tree for efficient LRF simulation.
- LRF measurement simulation.
- Simplified Ratnoo guidance algorithm.
- A wrapper application capable of setting parameters and running single of Monte Carlo simulations.
- Two analysis applications: one for single simulations, and another for Monte Carlo simulations.

This project was focused on the construction of solid infrastructure with which to carry out future work, which is a parametric and statistical analysis of the guidance algorithm's performance in terrain following. Several points were identified for potential improvement ahead of this future work:

- Better method of statistical analysis: all results are currently displayed as a function of time, although vehicles in different simulations in a Monte Carlo run may reach a given section of the trajectory at different times. Should this difference be large enough, comparing values between simulations at given times may provide little useful information.
- Alternative calculation of horizontal error, which is currently calculated as an absolute value and as such cannot be analyzed as normally distributed.

References

[1] A. Ratnoo, S. Hayoun, Granot A. and Shima T., "Path following using trajectory shaping guidance," *AIAA Journal of Guidance, Control and Dynamics,* vol. 38(1), 2014.

[2] A. Ratnoo, A. Manjunath, P. Mehrok and R. Sharma, "Application of Virtual Target based Guidance Laws to Path Following of a Quadrotor UAV," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, Arlington, VA USA, 2016.

[3] A. Shender and M. Idan, "Flight at a constant distance from ground for a multirotor," 2018.

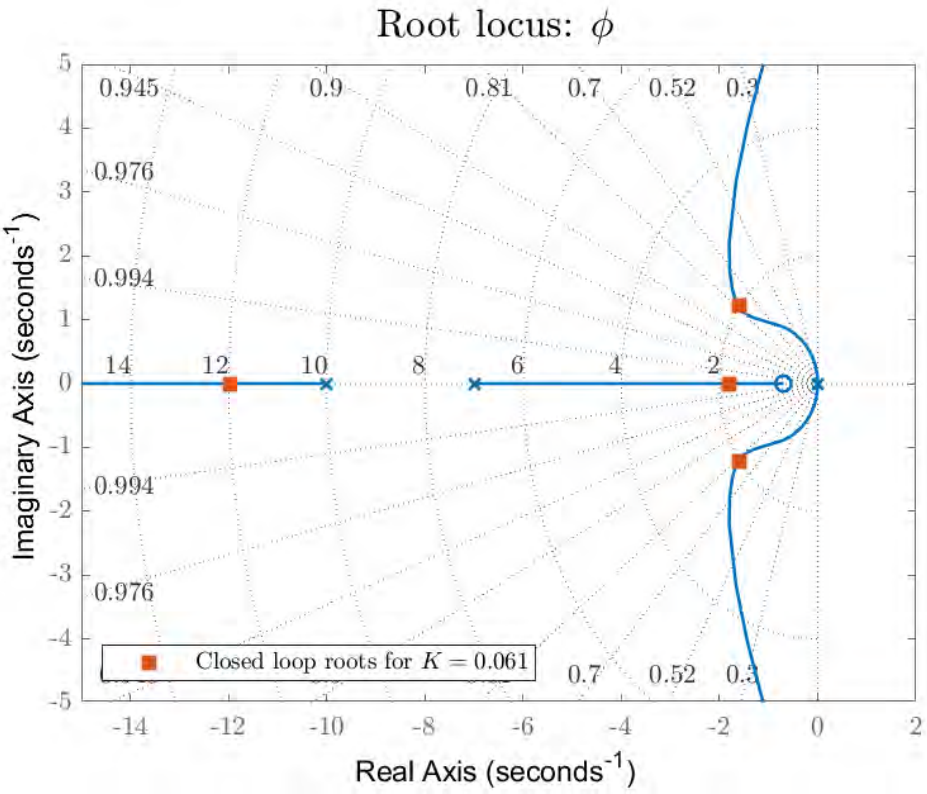
[4] T. Luukkonen, "Modelling and control of quadcopter," 2011.

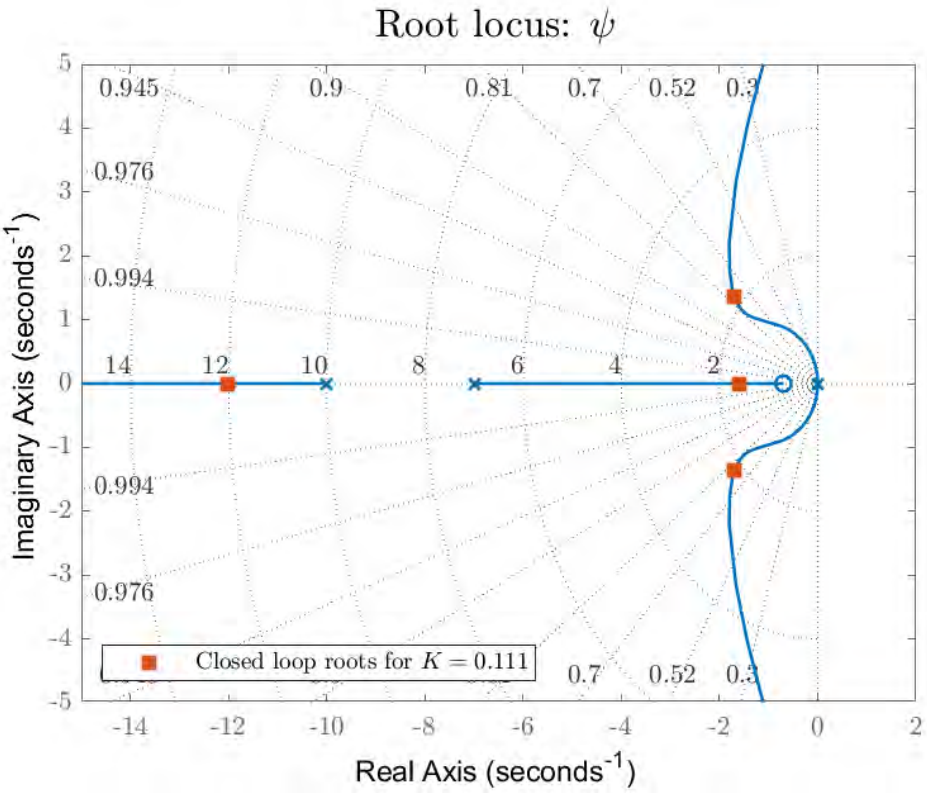
[5] K. Perlin, "Improving noise," *ACM SIGGRAPH,* pp. 681-682, 2002.

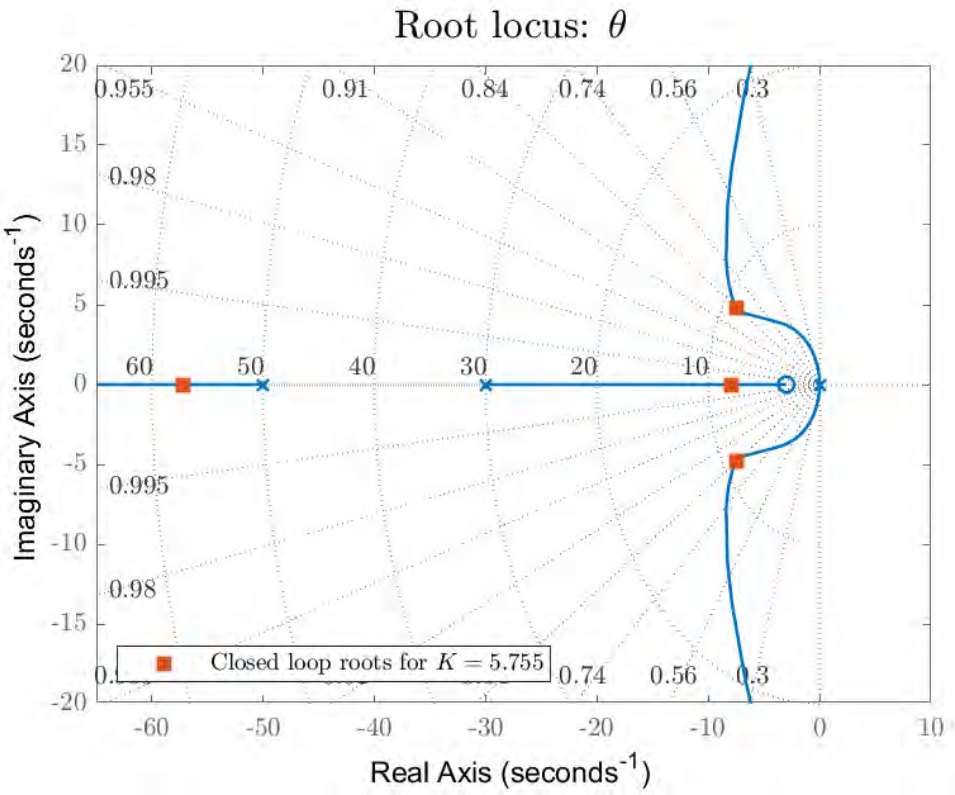
[6] T. Sjöstrand, "Efficient intersection of tterrain geometry in real-time," 2017.

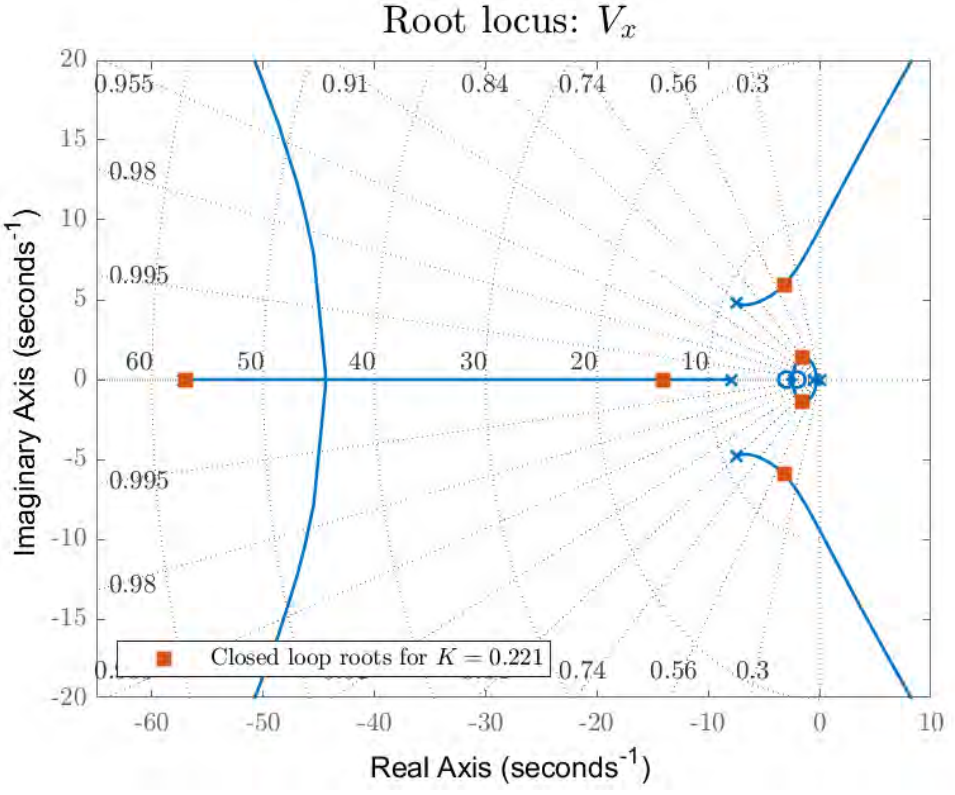
Appendix A

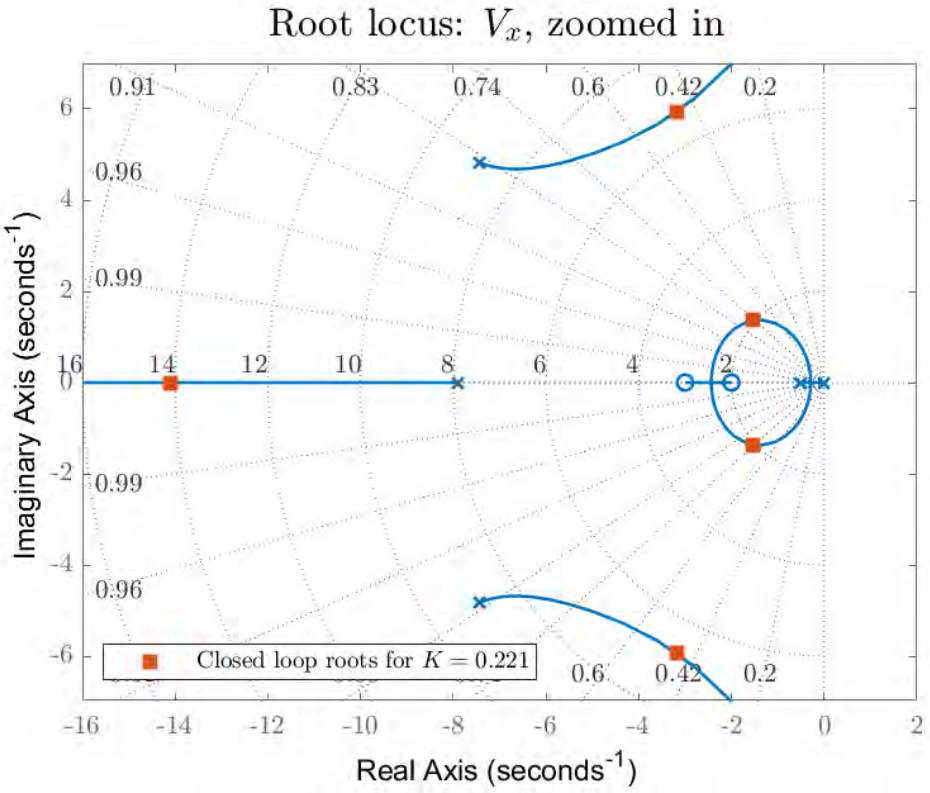
The following are the root loci used for the design detailed under section 3.

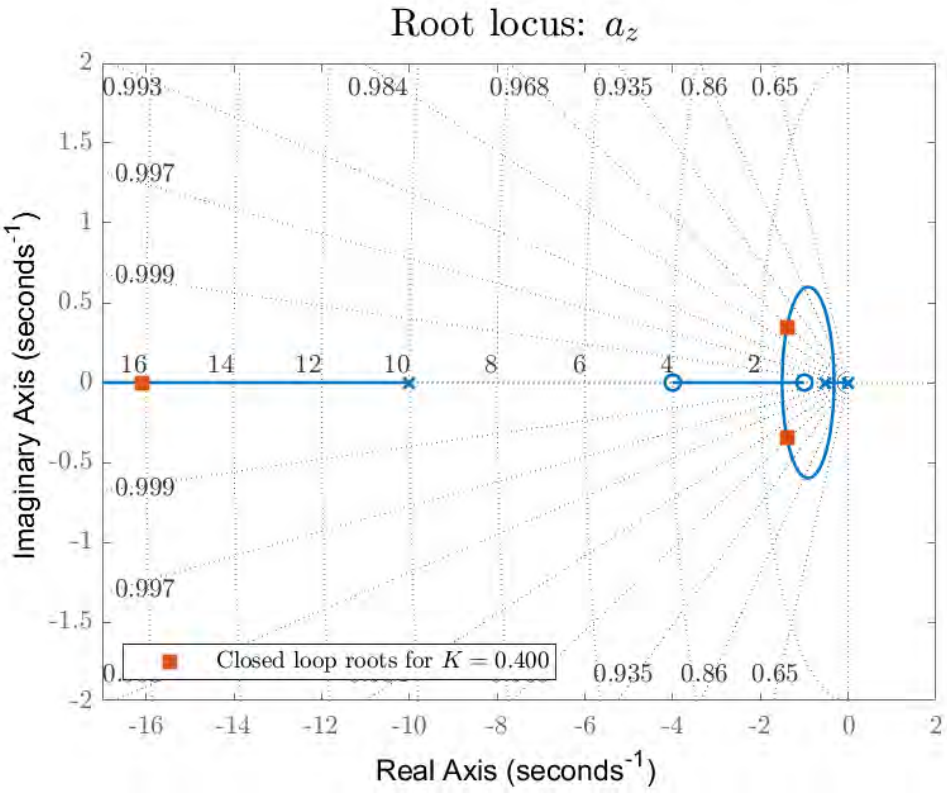





Root locus: θ
Imaginary Axis (seconds^{-1})
Real Axis (seconds^{-1})
Closed loop roots for $K = 5.755$
0.955
0.91
0.84
0.74
0.56
0.3
0.98
0.995
60
50
40
30
20
10


Root locus: V_x
Imaginary Axis (seconds^{-1})
Real Axis (seconds^{-1})
Closed loop roots for $K = 0.221$
0.955
0.91
0.84
0.74
0.56
0.3
0.98
0.995
60
50
40
30
20
10


Root locus: V_x, zoomed in
Imaginary Axis (seconds^{-1})
Real Axis (seconds^{-1})
Closed loop roots for $K = 0.221$
0.91
0.83
0.74
0.6
0.42
0.2
0.96
0.99
0.99
0.96
0.6
0.42
0.2
16
14
12
10
8
6
4
2
6
4
2
0
-2
-4
-6
-16
-14
-12
-10
-8
-6
-4
-2
0
2


Root locus: a_z
Imaginary Axis (seconds^{-1})
Real Axis (seconds^{-1})
Closed loop roots for $K = 0.400$
0.993
0.984
0.968
0.935
0.86
0.65
0.997
0.999
0.999
0.997
0.935
0.86
0.65
16
14
12
10
8
6
4
2
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-16
-14
-12
-10
-8
-6
-4
-2
0
2