sciendo

VOLUME 19, N° 2 2025
Journal of Automation, Mobile Robotics and Intelligent Systems

Model-Based Development of Autopilot for a Gasodynamically Controlled High-Speed Unmanned Aerial Vehicle

Submitted: 17th February 2024; accepted: 7th November 2024

Mariusz Jacewicz, Dariusz Miedziński, Grzegorz Chmaj, Robert Głębocki

DOI: 10.14313/jamris-2025-010

Abstract:
In recent years, model-based design and automatic code generation have gained popularity in various applications. However, using this approach in some specialized safety-critical applications is still challenging because the code must fulfill rigorous requirements. In this paper, the methodology of development of the software autopilot for the guided High-Speed Unmanned Aerial Vehicle (HSUAV) is presented in detail. The platform is actuated only with 32 solid propellant lateral motors, which makes the control task challenging. MATLAB and Simulink were used to develop the detailed simulation of the vehicle, together with the control software. A Model-in-the-Loop testing was evaluated to achieve an appropriate autopilot response. Embedded Coder was applied to generate production-ready C code from the model. A custom test framework was created to accelerate the design process. The numerical equivalency of the Simulink model and C code was investigated extensively using Software-in-the-Loop and Processor-in-the-Loop simulations. A developed control algorithm was implemented on real hardware with an ARM Cortex M4 microcontroller. The integrated prototype of the control system was successfully tested in laboratory conditions by Hardware-in-the-Loop simulation. The scientific significance of this paper lies in a comprehensive description of the methodology that might be used by other researchers.

Keywords: *model-based design, automatic code generation, Embedded Coder*

1. Introduction

Model-based design (MBD) is gaining popularity in various applications, such as medical, industrial, aerospace, and automotive. This methodology is also sometimes used to develop control systems for guided missiles. However, in the existing literature, finding a detailed and comprehensive description of the process is challenging. Such data are often classified due to security reasons. Moreover, the development of control systems for high-speed objects is very costly and restricted by military institutions. A lot of the existing systems are based on legacy solutions that were developed several years ago. Also, testing the developed system requires specialized test ranges because failure during real flight might have catastrophic consequences and be very dangerous.

When a malfunction occurs, the object can hit the unintended target accidentally. For this reason, using models plays a critical role in preparing flight tests. The onboard software must be developed cost-efficiently and ensure a final high-quality product. There exists a significant research gap in publications relating to the topic of automatic code generation (ACG) in the field of guided HSUAVs. The overall process workflow is generally well-known, but from existing available literature, it is hard to conclude about the details (for example, settings of the code generator or verification method).

Alpaslan [3] analyzed the challenges with the development of weapon software. The autopilot might be considered as a safety-critical system operating in real-time. The development of such a system is often more complicated than that of commercial products. Also, information security must be considered. Today, a lot of weapon functionalities are included in the software. As a result, requirements can be pretty similar to those in the space industry.

The traditional process of software development was manual coding [7]. However, such an approach is time-consuming. Up to this time, many papers on code generation have appeared in the literature. A review of recent studies on automated code generation was presented by Kshirsagar et al. [39]. ACG was used for the aerospace projects. Maroli et al. [46] adopted code generation for the motor controller for NASA's X-57 Maxwell aircraft. They also explained the role of several analysis tools available in MATLAB to ensure that the created code is bug-free. This technique was applied to develop the software for the fault protection system in the Deep Space 1 mission [47]. MBD and Simulink were successfully used to create the Guidance, Navigation, and Control (GNC) software for the Orion project, and NASA published a detailed description of the methodology in [31, 64]. Fraticelli [23], presented the example of software development for an Attitude Determination Control System (ADCS) of a satellite and investigated the numerical equivalency between the model and C code output. Carpenter [10] also reported using ACG to obtain the ready-to-use software for nanosatellite in the context of ADCS. Yahyaabadi et al. [71] studied the feasibility of using auto-code generators in future space missions.

Arm et al. [5] discussed the code generation problem for safety-critical applications and presented an example of a complete workflow for the ARM Cortex R hardware target. Erkinnen [17] also addressed the topic of software development by using ACG for such systems. Several years ago, Schwarz [59] mentioned that code generation is not often applied for safety-critical systems because such applications should obey strict requirements. However, up to now, auto-coding using automatic code generators is still not a widely accepted method of software creation in safety-critical applications. The main reason behind this is the lack of certification and qualification of the generators [15, 22, 62]. Moreover, many of the existing aerospace and defense systems were developed a couple of years ago, and there is a problem with adequately integrating the legacy codes with modern software development tools.

Autogenerated code from Simulink can have bugs that might have fatal consequences [34]. The resulting code must work in the same manner as the original model of the system. The problem of numerical equivalency between the model and autogenerated code was addressed by several researchers [6]. Lambersky [40] presented the development of the control algorithm for a legged robot and compared the results from the simulation and target platform.

Many examples of using ACG were reported for relatively simple systems that do not have to meet strict requirements [41, 48, 52, 53, 60]. For instance, Hyl and Wagnerová [28] presented the case of the controller development process for the climate unit laboratory stand. Otava [55] showed motor control development using Simulink Embedded Coder and verified the results experimentally. Fakih and Warsitz [21] reported the code generation process for two simple user case studies and compared the results of MIL and SIL simulations. Krizan et al. [38] presented a detailed description of a model-based design for control of a direct current brushless motor. Also, Wu [69] reported using MATLAB built-in tools from the Coder family for controller design for an electric motor using TI FI28379D hardware. Lixin and Liwen [11] presented the case of code generation for an autonomous underwater robot.

Existing works are related to using MATLAB tools in the MBD process. It must be mentioned that a set of other code generators exists. SCADE environment with a qualifiable KCG Code Generator is used for the development of avionics software [67]. In the automotive industry, the TargetLink generator reached significant popularity. Ajwad [2] compared the performance of Embedded Coder and TargetLink. Also, Scilab-Scicos and GeneAuto were proposed as open-source solutions [37, 58, 66]. Jacobitz and Xiaobo proposed [32, 33] LoRra generator that can be used to translate Scilab/Xcos models into usable C code. Also, a group of research papers proposes customized, much less popular generators other than widely used (both commercial and open-source) solutions.

For example, Yu et al. [72] designed a custom generator named Mercury and compared the performance with Simulink Embedded Coder. Bourbouh et al. [8] proposed a framework named CoCoSim that can be used to analyze and verify the code generated from Simulink models. Some recent studies concentrate on generating parallelized code from Simulink block diagrams [70].

Several trials by many individual researchers and various organizations were performed to standardize the flight-ready production code generation process from Simulink models. For example, Erkkinen [16] described several available guidelines and best practices for automatic code generation for flight applications to make it optimized and efficient. Some years later, in [18], he also presented the use of a qualified verification tool for the model-based designs for DO-178B applications. Fraticelli [24] described a set of valuable practices in generating C code from the Simulink model. MathWorks Automotive Advisory Board (MAAB) proposed a set of proper practices and recommendations. Motor Industry Software Reliability Association (MISRA) initially created a set of C standards only for automotive applications, but now they are widely used in other branches of industry. NASA has its standards and regulations (for example, NPR 7150.2, NASA-STD-8739.8, and NASA-GB-8719.13). One of the most important contributions and progress in that field in the context of aerospace applications is reports prepared by the Space AVionics Open Interface aRchitecture (SAVOIR) working group that can be used as guidelines in C code generation from Simulink models [19, 20].

In the field of guided UAVs, the literature related to MBD is relatively scarce. Among the detailed publications, one might mention the work of Putro and Septanto [57], who developed a real-time test environment for evaluating the missile's performance. Craft [13, 14] presented the case of using automatic code generation for the Long Range Land Attack Projectile (LRLAP). Abdelaty [1] gave a comprehensive description of the autopilot development for the anti-tank round. Tancredi [65] described the overall design methodology, and reported using code generation by MBDA company but without describing a particular case. Holliday [27] reported using MBD by Thales company and presented several user cases, however, without sufficient detail. Several authors briefly described Hardware-in-the-Loop simulators for guided high-speed objects [4, 36, 61, 76], but the results are difficult to reproduce. Waxenegger [68] presented a Hardware-in-the-Loop testing methodology for a rocket propulsion control system.

Automatic code generation from the system model offers several advantages compared to the software's traditional manual coding for the control system. The overall cost and time of the prototyping might be reduced significantly [74]. As a result, auto coding can reduce the overall effort compared to manual coding [51].

The research might concentrate on testing various variants of the algorithm instead of implementing low-level language details [45, 75]. The resulting code has a clear and repeatable structure. This technique also decreases the need for manual debugging, which is often time-consuming. Such methodology reduces the probability of introducing human-made mistakes during manual coding (for example, in signs). Furthermore, the consecutive versions of the software might be easily managed. The verification and documentation process could also be automated. Automatically generated code can be as fast or even faster than handwritten code [9, 12, 44]. Using MBD, the number of expensive and time-consuming flight trials that must be performed to validate the control system can be reduced significantly. SIMULINK diagrams are quite easy for managers and system engineers to understand, so team collaboration is simplified. On the other hand, this methodology requires the staff to have multidisciplinary competencies (flight dynamics, modeling, electronics, etc.).

The motivation for the presented study was strictly practical. There was a need to develop a control system prototype for the gasodynamically controlled HSUAV in a short time. The most common actuation method is using movable aerodynamic fins or thrust vectoring. Using multiple solid propellant lateral motors is much less prevalent.

Up to now, only a few existing solutions have used this type of actuation. Such pulse thrusters (as only actuators) were used on the M47 Dragon anti-tank and STRIX mortar rounds. Ukrainian Vilkha adopts 90 small pulse thrusters in the front of the fuselage, but also embraces the aerodynamic movable fins used in the last phase of flight. Moreover, 180 lateral motors are used on PAC-3 MSE in the initial and terminal phases of flight (this platform also uses aerodynamic control).

The main contribution of this study is the detailed description of automatic C code generation from the Simulink model for the HSUAV autopilot. The originality of this research lies in the fact that the object is actuated only by solid propellant lateral motors, which is not a standard solution. The control systems for such a configuration are not described in sufficient detail. For a tactical-grade system, the reliability of the operation is a critical factor. The solution must be carefully tested for various possible scenarios that might appear in the actual firings. The resulting code cannot include unintended functionality affecting the system's operation. This research extends the results presented in [30, 42]. The proposed approach might be helpful for other researchers in the field of external ballistics. The article also includes a description of a set of good code-generating practices.

The structure of the paper is as follows. At first, a description of the flying platform is shown. Then, a flight simulation model of the HSUAV developed in Simulink was presented. Next, the control algorithm was described. Later, the code generation process was explained.

The representative results were shown. The paper ends with a discussion of the obtained results and a summary of the main findings. Finally, some further possible research directions are suggested.

2. High-Speed Unmanned Aerial Vehicle Description

The diameter of the fuselage is 0.122 m and the total length of the object is 2.1 m. At launch, the mass is around 26 kg, and moments of inertia are 0.13 kgm^2 (longitudinal) and 10.2 kgm^2 (transversal). The mass of the propellant is 6 kg. The operation time of the main motor is a little more than 2.8 s, and the available thrust has a maximum value of about 8600 N. HSUAV is aerodynamically stabilized by four trapezoidal fins (there is no wrap-around functionality). These stabilizers are canted, and as a result, they create a rolling moment (the platform intentionally spins slowly during the flight). The platform is equipped with 32 solid propellant lateral thrusters located in front of the center of mass (no movable aerodynamic fins were used). The thrusters are arranged into four layers with eight motors in each. Each of these thrusters might be used only once. The operation time of the thruster is less than 0.05 s, and the mean thrust is more than 700 N. This kind of actuation is challenging because the object possesses very low control authority. That means the single pulse thruster can translate the impact point location by only several meters. The maximum range is approximately 11 km. In the considered version, the solution accounts for only stationary targets.

The GNC system comprises a navigation unit and an actuation unit. These subsystems are integrated into a control unit using the mechanical structure.

The main components of the navigation unit are a set of sensors, a Primary Control Computer (PCC), and three antennas. The main function of that system is to use data collected by the sensors and convert them into information about the object's flight state. The navigation data are obtained using an off-the-shelf available Inertial Measurement Unit (IMU), a custom-developed GALILEO receiver, a pressure sensor, and infrared sensors (these sensors are equally spaced around the circumference of the object body). The PCC uses an ARM Cortex A7 processor. The data obtained from all sensors is fused using a Kalman filter. One of the main difficulties of the adopted solution lies in the fact that the roll angle must be estimated very precisely (with accuracy < 1°). The low-cost micro-electromechanical systems (MEMS) gyroscopes that are available on the market have a low measurement range (typically up to 2000°/s) and are not entirely suitable for the developed system (sensor saturation could occur). Due to this, satellite signals needed to be received by the antennas mounted on the sides of the fuselage to estimate the roll angle. Moreover, the infrared sensors are used to improve the calculation accuracy. Additionally, the information about the altitude is corrected by the readings from the pressure sensor.

The actuation unit comprises a Lateral Motors Control Computer (LMCC), a set of power amplifiers, and lateral motors. The primary function of this unit is to use the data about the current platform state (obtained from the navigation module), calculate the steering commands, and activate the appropriate lateral thrusters. The LMCC is based on the ARM Cortex M4 architecture. It calculates when each motor should be fired to steer the HSUAV on target. The power amplifiers activate the igniters attached to the lateral motors. They convert the output data from the LMCC into a set of electrical signals for the 32 ignition channels. Each igniter (sometimes called a motor starter) is responsible for starting the combustion of a single small solid propellant in the lateral motor. This unit also includes an Electrostatic Discharge (ESD) protection circuit to prevent each igniter from unintentional release.

Additionally, the HSUAV is equipped with a radio telemetry downlink that transmits the data (linear accelerations, angular rates, position, etc.) to the ground control station (GCS) with a frequency of up to 500 Hz. The object state can be observed in real-time (in the form of graphs and visualization) on the GCS. During the launch phase, the electronic units are subjected to high acceleration. For that reason, triple-redundant communication channels were used to increase the system's reliability and minimize the probability of failure in flight.

The onboard battery powers all electronic subsystems described above.

3. Simulation Model Description

In the MBD approach, one of the first steps is to create a realistic plant model. To make the real prototype of the hardware, the detailed flight simulation model of the real object was developed and implemented in the MATLAB/Simulink environment. The baseline mathematical formulation can be found in "Appendix A". The detailed description of the mathematical model and its implementation can be found in [26, 29, 30, 42, 43, 56].

The vehicle subsystems (main motor, lateral thrusters, onboard sensors, etc.) were modeled based on the data of real hardware. The dynamics model parameters were obtained mainly by the experiments in laboratory conditions and firings of the existing prototypes (unguided ones). Additionally, they were confirmed using CAD models of the structure. HSUAV moments of inertia were measured using bifiliar and quadfilar torsional pendulums. The thrust curves of the main motor and lateral thrusters were acquired by measurements on the test stand at various temperatures. Aerodynamic parameters were collected by using specialized semi-empirical and Computational Fluid Dynamic (CFD) codes. Later, after initial flight trials, the aerodynamic database was corrected by the appropriate form factors. The sensors' data were found using datasheets from the manufacturers and confirmed by laboratory experiments.

Also, the launcher dynamics were included in the model to make the simulation of the initial phase of flight more realistic and include several important phenomena (for example, the tip-off effect). Environmental conditions (e.g., air temperature and humidity) are taken into account to predict the atmospheric properties.

The key element of the simulation is the model of the control system hardware and the software. The inputs to the control algorithm are current position coordinates, Tait-Bryan angles (yaw, pitch, roll), angular velocities, linear velocities, and linear accelerations. The output is the firing command of each thruster. The target position coordinates (latitude, longitude, and altitude) must be known and programmed before the launch.

The scheme of the overall control algorithm is presented in Figure 1.

The quality of the resulting C code can be affected by a lot of factors and settings. To appropriately choose the compiler parameters, the model was created using SAVOIR guidelines [19, 20]. The equations of motion were integrated numerically using a fixed-step Bogacki-Shampine (third-order) solver with a step size of 0.0001 s. Only fixed-step solvers are currently supported in MATLAB during the code generation process. The model cannot contain any blocks that are not supported by the code generator. Before performing ACG, it is crucial to ensure a well-styled Simulink model. All of the inputs and outputs were named using strict predefined conventions. The names and units were displayed on the signal wires to eliminate the probability of human error. Signals were grouped using buses, which ensured readability. Signal line crossing was eliminated. The scheme also includes a set of diagnostic outputs that can be used in later stages to analyze in detail the results on the actual hardware. Equality comparison using floating point numbers was avoided because that can lead to entirely unpredictable results.

A small team of experienced programmers developed the model, so it was necessary to perform version control. Each subsequent version of the code has a unique name with data and time of last saving included. Simulink Design Verifier was also used to detect potential problems with the block diagram (for example, dividing by zero or dead logic).

The requirements and acceptance criteria for individual subsystems were formulated at the initial phase of the control system development. These requirements were also created for the software. The code should be efficient and able to operate in real-time on the target hardware. The control algorithm was carefully optimized to increase the execution speed and save the available computational resources. The autopilot code should be integrated with other software parts created by different cooperators. It is obvious that the software should include appropriate comments and be easy to understand.

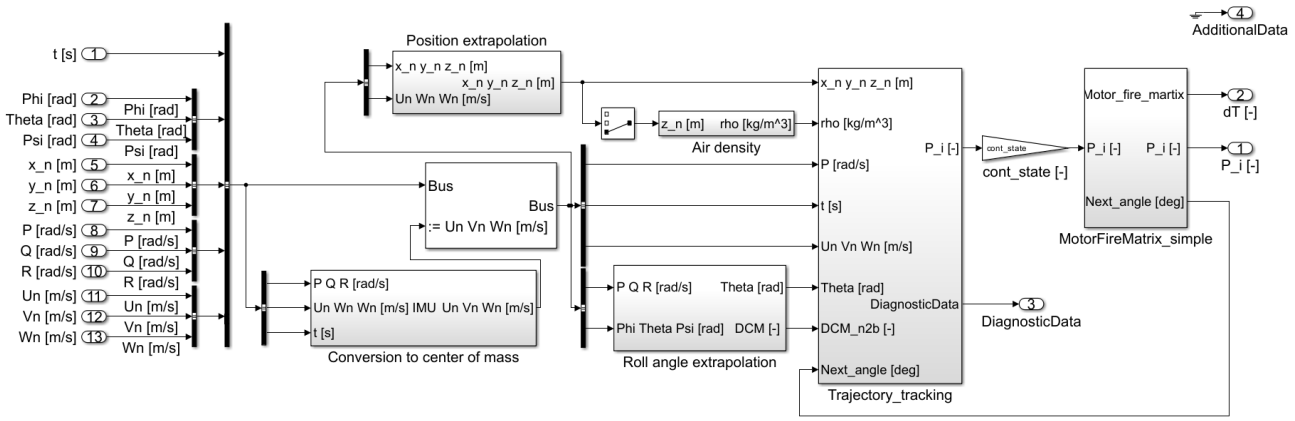

Figure 1. Top-level Simulink scheme of the control algorithm

3.1. Navigation Data Extrapolation

The GNC module can be considered as a complicated multi-rate digital system. The data about the individual HSUAV flight parameters can be delivered from the navigation system module with various (often relatively low) frequencies. For example, the position using a GALILEO receiver can be obtained with a frequency of no more than several hertz. On the other hand, the raw data from IMU outputs (angular rates and linear accelerations) could be sampled with a frequency of up to 200 Hz. These data are fused using a Kalman filter [73] to obtain a more accurate estimation of the flight parameters (mainly velocity, position, and Euler angles), but the output frequency is no more than 40 Hz. The thrusters can be fired only in certain phases of flight when the roll rate of the object is approximately 2 revolutions per second (after apogee). If the roll angle is estimated at 40 Hz, then the motors can be fired with the angular resolution 18° which is too low. That means the control loop should operate at a much higher frequency than the data can be delivered from sensors. It was estimated that the control algorithm should be recalculated with a frequency of 5000 Hz. That way, the firing conditions will be checked with an angular resolution of 0.144°.

For this reason, an upsampling algorithm should be applied to obtain the navigation data with the required frequency. On the other hand, such an algorithm cannot introduce time delays because then the object might be steered in the inappropriate direction.

A custom module (whose primary function is to extrapolate the data gathered from the various onboard sensors) was developed to overcome that difficulty. This module consists of two parts: extrapolation of roll angle and position, which were determined to be the most influential on the algorithm's accuracy. On the output of both extrapolation modules, the navigational information (roll angle and three-dimensional position in space) is delivered with the desired frequency of 5000 Hz. All other signals are obtained directly from the navigation module without the extrapolation procedure (their frequency is sufficient).

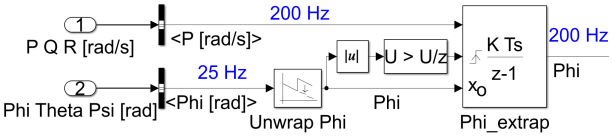

Figure 2. Roll angle extrapolation block scheme (signal frequencies denoted in blue font)

The roll angle extrapolation block scheme is presented in Figure 2.

The main idea behind this method is that the angular rate (obtained directly from gyroscopes) can be delivered with a higher frequency when compared to information about the actual roll angle. It was assumed that between two consecutive samples of the roll angle, the information on the intermediate points can be obtained using linear interpolation. The forward Euler method was used for numerical integration of the roll rate P to obtain the roll angle Φ:

$$\Phi(n) = \Phi(n-1) + [t(n) - t(n-1)]P(n-1) \quad (1)$$

where t - time and n - sample number. After delivering a new value of the roll angle from the measurement system, the value of the integral is reset, and the integration starts from the new initial condition (which is the same as the last measured value of the roll angle).

The same idea was used to extrapolate the position coordinates:

$$x_n(n) = x_n(n-1) + [t(n) - t(n-1)]U_n(n-1) \quad (2)$$

$$y_n(n) = y_n(n-1) + [t(n) - t(n-1)]V_n(n-1) \quad (3)$$

$$z_n(n) = z_n(n-1) + [t(n) - t(n-1)]W_n(n-1) \quad (4)$$

where U_n, V_n, W_n - velocity vector components in North-East-Down frame.

3.2. Guidance Algorithm

A modified Proportional Navigation Guidance (mPNG) algorithm was developed to steer the object accurately towards the target. The target location $\vec{P_T} = [P_{Tx}, P_{Ty}, P_{Tz}]$ and the HSUAV position $\vec{P_M} = [P_{Mx}, P_{My}, P_{Mz}]$ must be known. The acceleration perpendicular to the line of sight (LOS) is [49]:

$$\vec{a_c} = N\vec{V_c}\dot{\vec{\lambda}} \quad (5)$$

where N - proportionality constant, $\vec{V_c} = [V_{cxy}, V_{cxz}, V_{cyz}]$ - closing velocity and $\vec{\lambda} = [\lambda_{xy}, \lambda_{xz}, \lambda_{yz}]$ - line of sight angular rate. The projections of the closing velocity on the xy, xz and yz planes of the North-East-Down (NED) coordinate system are:

$$V_{cxy} = -\frac{P_{TMx}V_{TMx} + P_{TMy}V_{TMy}}{\sqrt{P_{TMx}^2 + P_{TMy}^2}} \tag{6}$$

$$V_{cxz} = -\frac{P_{TMx}V_{TMx} + P_{TMz}V_{TMz}}{\sqrt{P_{TMx}^2 + P_{TMz}^2}} \tag{7}$$

$$V_{cyz} = -\frac{P_{TMy}V_{TMy} + P_{TMz}V_{TMz}}{\sqrt{P_{TMy}^2 + P_{TMz}^2}} \tag{8}$$

The components of the distance between the target and the vehicle (along each axis of the NED frame) are:

$$P_{TMx} = P_{Tx} - P_{Mx} \tag{9}$$

$$P_{TMy} = P_{Ty} - P_{My} \tag{10}$$

$$P_{TMz} = P_{Tz} - P_{Mz} \tag{11}$$

In a similar way, the components of the relative velocities are:

$$V_{TMx} = V_{Tx} - V_{Mx} \tag{12}$$

$$V_{TMy} = V_{Ty} - V_{My} \tag{13}$$

$$V_{TMz} = V_{Tz} - V_{Mz} \tag{14}$$

The LOS angles in each plane are:

$$\lambda_{xy} = \arctan\frac{P_{TMy}}{P_{TMx}} \tag{15}$$

$$\lambda_{xz} = \arctan\frac{P_{TMz}}{P_{TMx}} \tag{16}$$

$$\lambda_{yz} = \arctan\frac{P_{TMz}}{P_{TMy}} \tag{17}$$

These angles must be differentiated with respect to time to obtain LOS rates.

$$a_{cxy} = NV_{cxy}\dot{\lambda}_{xy} \tag{18}$$

$$a_{cxz} = NV_{cxz}\dot{\lambda}_{xz} \tag{19}$$

$$a_{cyz} = NV_{cyz}\dot{\lambda}_{yz} \tag{20}$$

Next, the acceleration obtained from equation 5 must be converted to the body-fixed frame and compensated by the gravity:

$$\vec{a}_c^b = T_n^b(T_{LOS}^n\vec{a}_c + \vec{g}) \tag{21}$$

where

$$T_n^b = \begin{bmatrix} \cos\Theta\cos\Psi & \cos\Theta\sin\Psi & -\sin\Theta \\ \sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi & \sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi & \sin\Psi\cos\Theta \\ \cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi & \cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi & \cos\Phi\cos\Theta \end{bmatrix} \tag{22}$$

is the transformation matrix from the North-East-Down (NED) frame to the body-fixed frame and

$$T_{LOS}^n = \begin{bmatrix} -\sin\lambda_{xy} & -\sin\lambda_{xz} & 0 \\ \cos\lambda_{xy} & 0 & -\sin\lambda_{yz} \\ 0 & \cos\lambda_{xz} & \cos\lambda_{yz} \end{bmatrix} \tag{23}$$

and

$$\vec{g} = \begin{bmatrix} 0 & 0 & -g \end{bmatrix} \tag{24}$$

At the outputs, this algorithm delivers information about the commanded linear accelerations in the body-fixed frame. The three components of the acceleration vector are used as inputs to the Firing Logic (FL) submodule (please see the next section of the paper). To hit the target successfully, the vehicle must realize this commanded acceleration. The commanded direction of flight in the body-fixed frame is calculated as:

$$\gamma = \arctan\frac{a_{cy}^b}{a_{cz}^b} \tag{25}$$

The magnitude of the commanded acceleration is:

$$|\vec{a}_{xy}| = \sqrt{{a_{cx}^b}^2 + {a_{cy}^b}^2} \tag{26}$$

The details about the mPNG algorithm are presented in [30, 56]. The proposed algorithm can achieve a significant impact point dispersion reduction, which was proved by various studies conducted in e.g. [25, 26, 29, 35, 63], where very similar types of guidance method were utilized.

Additionally, it must be remembered that the navigation system estimates the velocity $\vec{V}_{IMU}^b$ at the location of IMU, so this value must be converted to the center of mass:

$$\vec{V}_{cg}^b = \vec{V}_{IMU}^b + \vec{\omega} \times (\vec{r}_{IMU}^b - \vec{r}_{cg}^b) \tag{27}$$

where $\vec{\omega}$ - vector of angular rates, $\vec{r}_{IMU}^b$ - location of the IMU (measured from aft) and $\vec{r}_{cg}^b$ - actual position of object center of mass.

3.3. Firing Logic of the Lateral Thrusters

The discrete nature of actuation makes the guidance process difficult because there is no possibility of continuously tracking the commanded lateral accelerations. Since only 32 thrusters are available, their use during the flight must be carefully planned. If the motors are fired too early, then it will be impossible to influence the trajectory in the terminal phase of flight. Moreover, if the time between two pulses is too short, the platform can achieve high angles of attack and sideslip, and as a result, disintegration can occur due to high structural loads. On the other hand, if the thrusters are fired at too large time intervals, their usage will be ineffective, and the HSUAV will not achieve its destination.

The firing logic is based on the conjunction of four conditions that could be expressed mathematically as (for details, please see [30]):

$$\begin{aligned} & t - t_{prev} \le t_t \quad \wedge \\ & |\vec{a}_{xy}| \le a_t \quad \wedge \\ & |\gamma - P\tau - \Phi_i - 180°| \le \gamma_t \quad \wedge \\ & t \ge t_g \quad \wedge \quad \Theta \le \Theta_g \end{aligned} \tag{28}$$

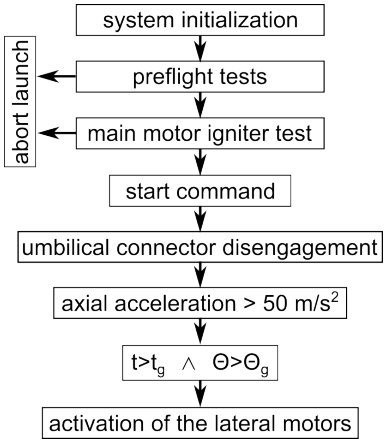

Figure 3. Simplified sequence of events before activation of the thrusters

where t - actual time, t_{prev} - time of the previous firing, t_t - the minimum allowed time between the two consecutive pulses, $\vec{a}$ - actual acceleration vector, a_t - threshold value, γ - commanded flight direction, P - measured roll rate, τ - time constant (approximately one-half of the burning time of the propellant), Φ_i - angular location of the lateral motor, γ_t - angular tolerance of the lateral motors firings, t - time, t_g - threshold time, Θ - actual pitch angle, Θ_g - pitch angle threshold.

The individual thruster is fired when the output signal from the algorithm changes from 0 (low state) to 1 (high state), which happens after all four conditions are true simultaneously.

The state machine (Figure 3) was also used to realize a step-by-step control process to prevent too-early lateral motor firings for safety reasons. The actuation unit with the control motors might be activated only after detecting the launch event (longitudinal acceleration of more than 50 m/s^2 over 0.4 s).

3.4. Pulse Thrusters Assignment

The lateral thrusters are fired in the predefined sequence prepared as a firing table (block "MotorFireMatrix_simple" in Figure 1. The system user might program this sequence before the launch. An example of such a table is presented in Table 1, where the first column is the number of the pulse, the second column is the ID of the thruster, and the last column is the angular position of the thruster on the fuselage's circumference.

Table 1. Example of the firing table.

Pulse no.	Thruster ID	Angular position
1	2	0^o
2	5	25^o
...	...	...
32	28	285^o

4. Automatic C Code Generation

The software should be developed using a strict strategy. Several management methodologies exist that can be applied, but some are unsuitable in this context. For example, the waterfall methodology was not feasible because the initial requirements were not fully formulated at the beginning of the process. The software development process diagram is presented in Figure 4.

The onboard hardware is quite limited in computational resources, and it is evident that the control algorithm prototype developed in MATLAB must be implemented in the appropriate low-level programming language. The Simulink Coder (earlier Real Time Workshop) and Embedded Coder were used for automatic code generation. However, this MATLAB extension was insufficient to automate the whole process entirely. For this reason, a custom code was developed to generate, compile, run, and test the algorithm. Microsoft Visual Studio (MS VS) 2019 was used to compile the code.

4.1. Model Build Options

To generate the code appropriately, the settings must be carefully chosen, as the default code generation settings are not optimal [50]. ISO/IEC 9899:1999 C language standard was used. The code was generated with various settings, and the results were assessed for each combination of parameters. For the generated code, a Visual C\C++ file for Embedded Coder was chosen. Also, the influence of code optimization settings on the accuracy of the generated code was investigated. After a set of trials, it was decided to set the optimization level as a minimum. The algorithm parameters (e.g., target location can be set as inlined or tunable. It was decided that they should be inlined in the code to reduce memory usage and increase code efficiency. In that way, there is no need to preallocate the memory for the parameters.

The embedded Coder Support Package for ARM Cortex-M Processors was used to ensure that the code would be effective in realizing math operations using the Cortex Microcontroller Software Interface Standard (CMSIS) library. Using this addon, it is possible to replace some functions on the more effective implementations (for example, "sin" trigonometric function is replaced on "arm_sin_f32").

Single-precision float-type numeric numbers were used to save the available memory. By default, MATLAB operates on double-precision numbers, but using them in the microcontroller is not necessary and seriously slows down the computations. For that reason, all input and output ports in the Simulink blocks of the control system model were manually set to "single". In that way, it was ensured that there would be no unintentional conversion between "single" and "double". It was detected that this issue is crucial and might lead to catastrophic consequences.

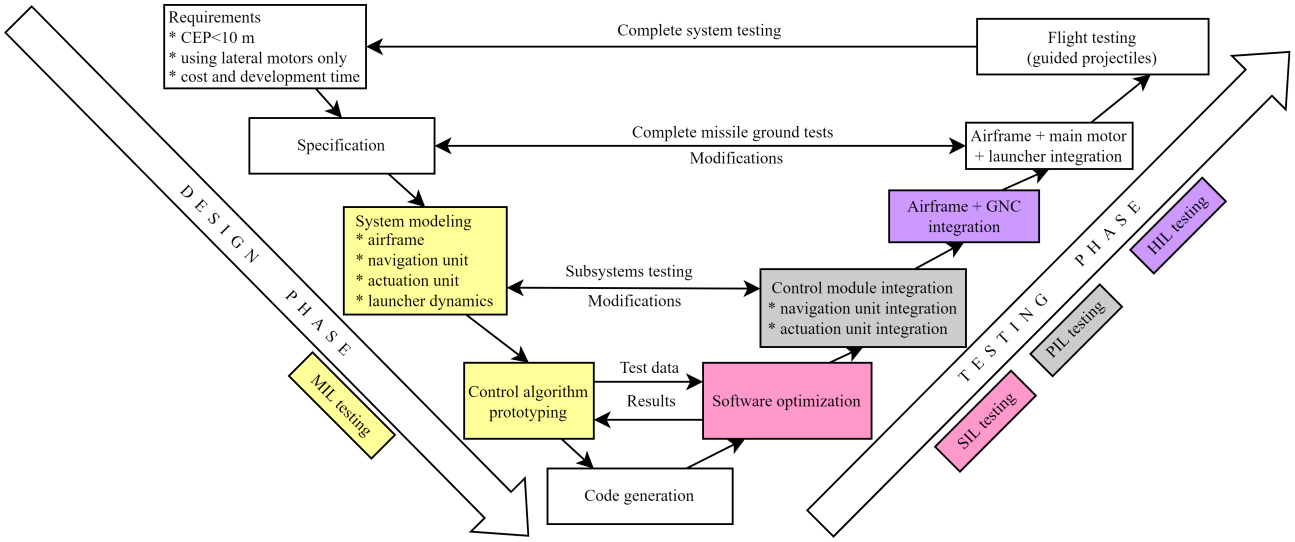

Figure 4. Software development process

4.2. Code Verification Methodology

Before being used on real hardware, the algorithm must be tested for various operating conditions. A custom test procedure was developed to verify the control algorithm's reliability and ensure high code coverage.

The code generation process started with the selection of an appropriate flight scenario. At first, a vast number of Model-in-the-Loop (MIL) simulations (for individual flight scenarios) were evaluated to detect potential problems at the early stage of control system development. At this stage, only a control module model was connected with the HSUAV model. These tests were realized for various launcher azimuth and elevation angles, various thrust curves of the lateral thrusters, etc. Next, the structure of the Simulink control system model was iteratively optimized to achieve the best possible configuration and decrease execution time. The model was carefully checked to prevent inefficient parts of the code (for example, algebraic loops). Not all functions are supported for code generation. In that case, a custom equivalent block structure replaced such an unsupported function. Simulink Profiler was used to measure the execution time of individual model components. Simulink "Accelerator mode" was set to speed up the execution time. In that way, the resulting time of a single simulation was decreased to only 7 s (for 50 s of the real flight). Then, Monte Carlo simulations were used extensively to evaluate accuracy and precision. The Parallel Computing Toolbox was used to assess the simulations on the workstation (Intel® Core™ i9 and 16 GB RAM). In that way, it was possible to simulate thousands of firings and estimate the measures of dispersion (for example, Circular Error Probable). Also, parametric analysis was conducted to investigate the influence of individual autopilot settings on the overall accuracy and to understand the system behavior. The example of results from this methodology can be found in [26]. This phase ends when the controller model meets the predefined requirements and generates appropriate outputs.

After the MIL stage, detailed documentation was created to describe the low-level functionality of the flight simulation. The documentation of the results was based on the automatic generation of structured reports. After each simulation in MATLAB, two text files were recorded and saved on the computer disc. The first file included in the tabular form time and platform flight parameters: angular rates, position, Euler angles, and linear accelerations. The second file included the array of firing commands. In the later stages of the whole testing process, the above-mentioned two files were used as reference data (in the end, the output of the final code should match the reference firing commands).

After completing the MIL phase, the next step was to perform Software-in-the-Loop (SIL) tests. At the SIL stage, the C code from only the autopilot module was generated and used in the control loop instead of the controller system model. The traceability function that is available in Embedded Coder was used extensively to navigate between the Simulink model and C code. These tests were performed on the standard desktop computer (Intel® Core™ i7, 32 GB RAM, MS Windows 11 operating system) without dedicated hardware. At this stage, checking if the autopilot software could be implemented was possible.

Additionally, the code was tested externally to ensure no problems with the generated C software. The above-mentioned custom-developed test procedure was as follows. The C code was run on a Desktop computer in MS VS, and the results (lateral thruster's firing commands) were stored in a separate text file. Then, both text files (reference data from MATLAB obtained at the MIL stage and results from the standalone C code) were compared. If the results met the acceptance criteria, the C code was ready to be implemented on the real hardware and tested again.

If any anomaly output or unintended behavior was observed, then it was necessary to return to the MIL stage, introduce modifications, confirm that after adjustments, the MIL results were still acceptable, and then again continue the SIL level using the modified software. This process was automatized using MATLAB scripts.

Next, the Processor-in-the-Loop (PIL) phase was performed. The developed algorithm in the form of C code was implemented on real hardware with an ARM Cortex M4 processor. The developed control system was tested in laboratory conditions. At first, the (PIL) simulation was evaluated not in real-time. The reference data, including flight parameters, were sent from the text file to the hardware by the interface (these data were used instead of the information from the navigation unit). The control computer calculated steering commands, and the outputs from the system were logged into text files. The main goal of that stage was to check that the results were the same as the reference data. Then, real-time calculations were performed to ensure that the control loop was able to operate with a sufficient frequency (5000 Hz, as mentioned before).

One of the last steps of the testing was the Hardware-in-the-Loop simulation. At this stage, the controller performance can be verified (at least partially) without using the complete real plant. This approach is much safer than testing the control system on the real object in flight tests. The system operated in real-time.

In the last stage, the results from MIL, SIL, PIL, and HIL were compared to ensure that the results were in agreement. If the results of HIL were unsatisfactory, the process came back to MIL, SIL, or PIL levels. Otherwise, if the code met the acceptance criteria, it was decided that the software might be implemented on the real vehicle.

5. Results and Discussion

The main goal of the experiments was to check the numerical equivalency (or eventually similarity) of the results between the model in Simulink and the resulting C code that might be directly implemented on the real hardware. The obtained C code must work like the original MATLAB program when tested with the same input data. The numerical discrepancy can be caused by various issues, for example, low-level implementations of the trigonometric functions in the two programming languages [54].

MATLAB R2023 with Update 5 was used to perform numerical simulations. The control algorithm was improved step by step, iteratively. Several iterations of MIL, SIL, and PIL simulations were evaluated to meet the acceptance criteria. The solution was acceptable if the impact point dispersion measured using Circular Error Probable 50% (CEP50%) was smaller than 25 m and the results were repeatable for various flight scenarios. Here, the selected example of a mission scenario is presented.

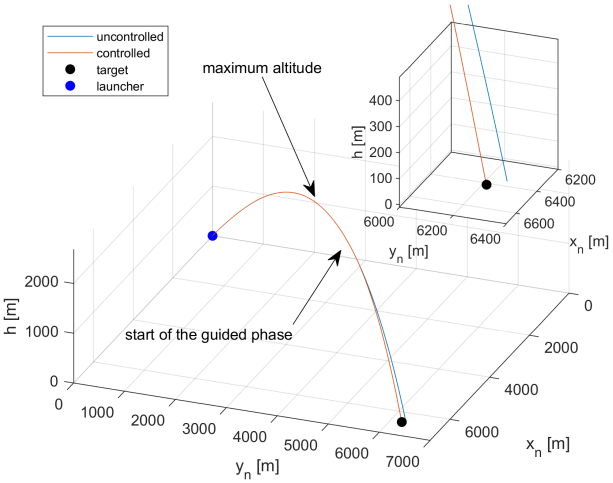

Figure 5. Three-dimensional flight trajectory

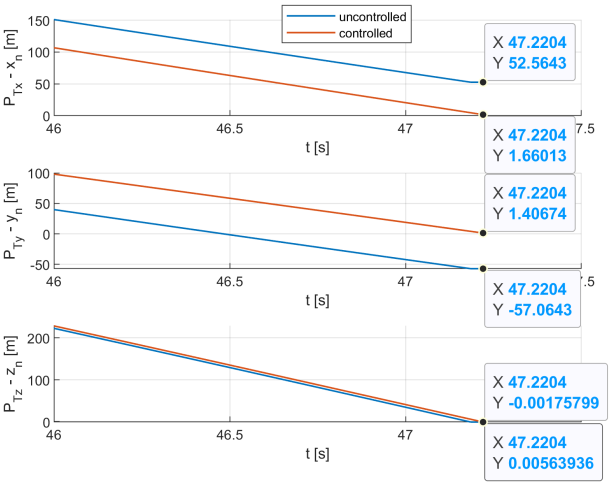

Figure 6. Position errors (in the last phase of flight)

The following initial conditions were assumed: launcher elevation angle 45.2° and azimuth 46.5°. The algorithm settings were as follows: t_t = 0.25 s (minimum time between two pulses), Θ_g = 5° (pitch angle threshold), t_g = 30 s (threshold time), γ_t = 1.5° (angular tolerance of lateral motor firings), a_t = 0.5 m/s^2 (threshold value of the lateral acceleration). The launcher was located at the origin of the NED coordinate system. It was assumed that the target is stationary, and its coordinates in the NED frame were set to (6430.7, 6222.0, 0.0) m.

At first, MIL tests were evaluated for two cases: unguided flight (the control system was intentionally deactivated) and for the guided one. The comparison of the flight paths is presented in Figure 5. In Figure 6, the errors between the target location and the actual position of the HSUAV are shown for the terminal phase of flight. Angular rates (roll, pitch, and yaw) are shown in Figure 7 and orientation angles in Figure 8.

The uncontrolled HSUAV ground range is about 8950.3 m, and the maximum achieved altitude is approximately 2680 m. For the unguided platform, the impact point coordinates are (6378.1, 6279.1) m, and the achieved miss distance was 77.58 m.

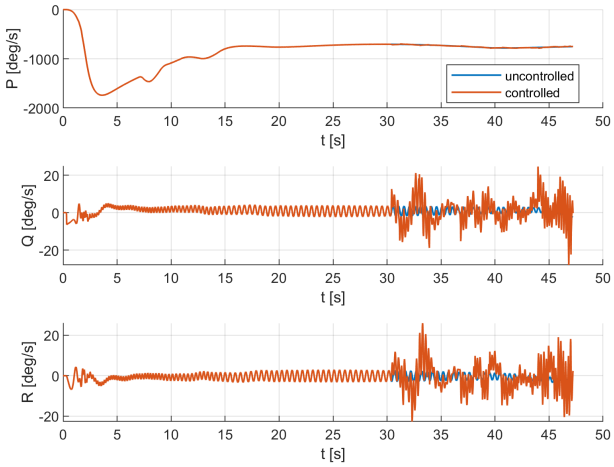

Figure 7. Angular rates

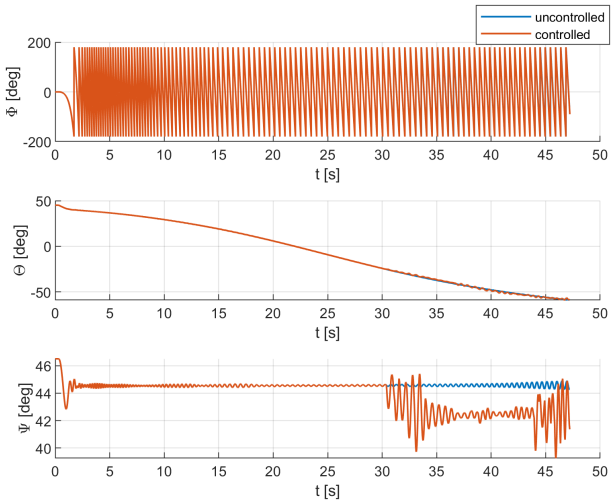

Figure 8. Euler angles (roll, pitch, and yaw)

For the controlled one, the impact point is located at (6429.0, 6220.6) m, so only 2.18 m from the target. That means the lateral motors can successfully steer the HSUAV to the target.

All three components of the position error should ideally be 0. For the unguided object, the x and y errors (in the horizontal plane) deviated significantly from the desired value. On the other hand, for the guided one they are very small (<2 m).

The absolute value of the roll rate increases in the active portion of the flight (when the main motor operates). The minus sign in the first plot in Figure 7 means that the HSUAV rotated counterclockwise when looking from aft. For the controlled flight, the oscillations of pitch and yaw rate signals after 30 s result from the firings of lateral motors.

For the guided object, the disturbances can also be observed on pitch and yaw angle time histories.

The forces generated by the lateral motors in the aeroballistic frame attached to the platform center of mass are presented in Figure 9. It can be observed that 21 lateral motors were used.

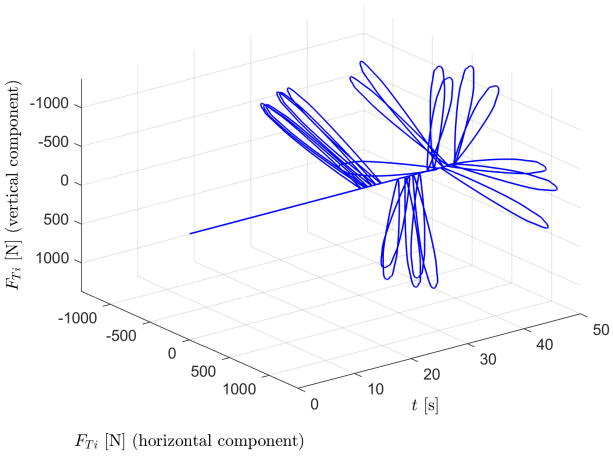

Figure 9. Forces generated by lateral motors

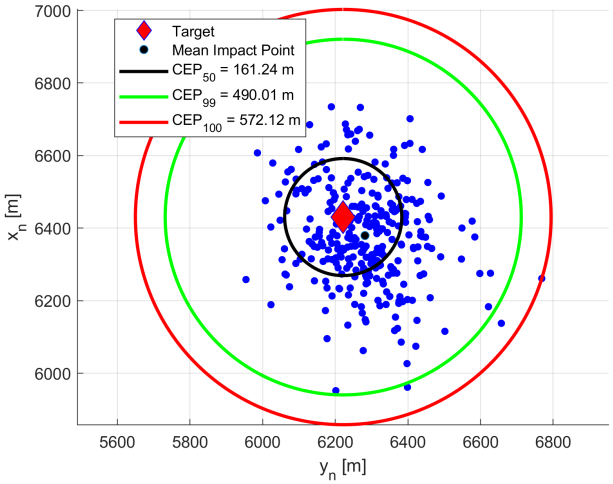

Figure 10. Impact point dispersion (uncontrolled flights)

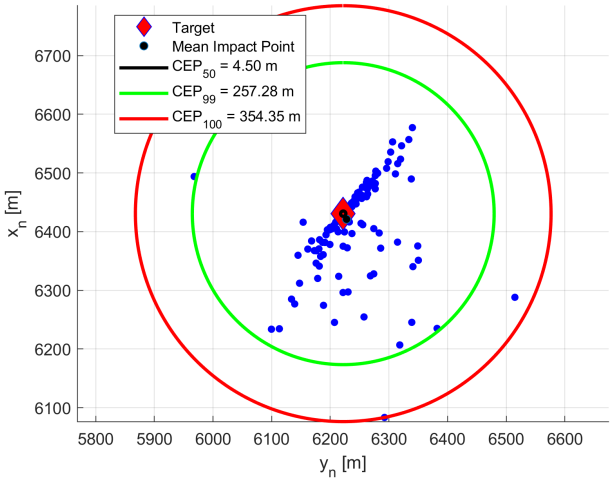

Figure 11. Impact point dispersion (controlled flights)

Later, a set of Monte-Carlo simulations were performed (300 runs for each scenario). The comparison of impact point dispersion for unguided and guided scenarios is presented in Figures 10 and 11.

The CEP50 centered on the target for uncontrolled scenario one is 161.24 m. Some of the impact points are located 572.12 m from the aiming point (please see CEP100%).

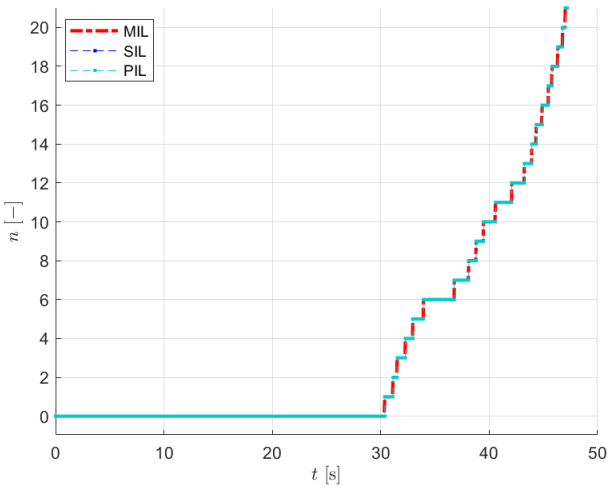

Figure 12. Number of activated lateral motors

There is a systematic offset between the mean point of impact (MPI) and the target. With the control system applied, CEP50% decreased to 4.50 m. That means using a control system reduces the impact point dispersion (measured using CEP50%) by approximately 35.83 times. The location of MPI coincides with the target. This is important from a practical point of view because the unintended collateral damage can be minimized.

Next, the C code was generated, and SIL simulations occurred. Finally, the obtained C code was implemented on the target hardware with ARM Cortex M4. The PIL simulations were evaluated, and the obtained results were compared with MIL and SIL. The comparison of outputs from the controller implemented in Simulink and using C code is presented in Figure 12. The abovementioned figure illustrates the number of firing commands obtained from the original model in Simulink and from the SIL and PIL simulations. In the ideal situation, lines should coincide with each other. Then, the original Simulink model and C code would be numerically equivalent. In practice, due to numerical rounding errors, the model and C code are numerically similar but not perfectly identical.

Up to 30.37 s, no thruster was activated because the conditions given by Equation 28 were not fulfilled. That means the HSUAV was in an unguided flight. Then, a series of 21 firings occurred. The last motor in the sequence was activated in 41.01 s.

To better visualize and understand the eventual numerical errors, in Figure 13, the plot of time differences between Simulink and C code generated in SIL in evaluating firing commands is shown. It was assumed that the value of the acceptance threshold is 0.003 s.

The dots indicate that for most of the motors, there was no time difference between results from Simulink and C code. In 4 cases, the difference was smaller than 8e-15 s. The results obtained indicate that the C code possesses functionality identical to the original Simulink model.

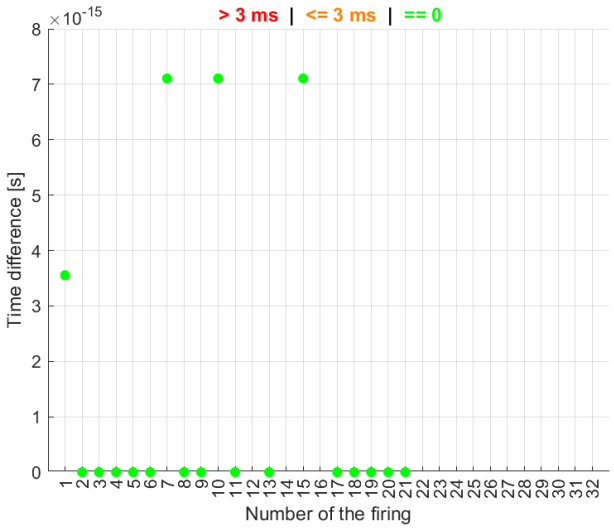

Figure 13. Time differences in firing commands (Simulink vs C code)

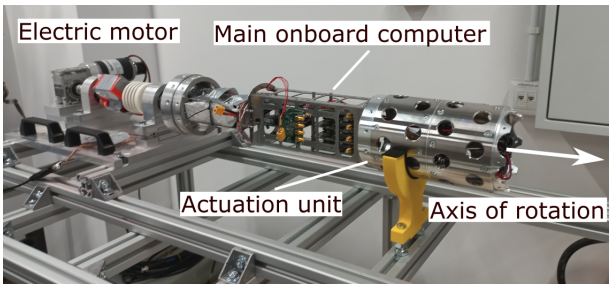

Figure 14. Control system prototype during the laboratory experiments

In the next stage, a series of laboratory Hardware-in-the-loop experiments with the actual hardware were performed. This issue is not trivial because it is quite difficult to mimic the real conditions (for example, linear accelerations) in the static tests. Here, only the basic description of these tests is presented for legal reasons. The main goal of the tests was to ensure that the lateral motors could be fired in the right direction and at appropriate time intervals. For example, if the HSUAV should turn left (when looking from aft), then the thrusters must be fired on the right side of the fuselage. To verify the control unit, a simplified laboratory test stand was developed. The apparatus (Figure 14) consisted of a mounting, electric rotor, encoders, control unit, and operator's control desk.

The main computer electronics were powered by the onboard rechargeable lithium polymer battery (12 V) integrated with the control system. The electric motor that drives the stand was connected to a different power source. The remaining elements (main solid propellant rocket motor and nose section) were removed because they are separate units (not necessary in the experiments). The developed control algorithm was implemented in the onboard computer.

The electric motor about the longitudinal axis rotated the control module with the same angular rate as obtained from the numerical simulation.

That way, the realistic input data about the roll angular rate and the roll angle from the navigation system were generated. The test stand is stationary, so the information about the position, linear speed, and accelerations cannot be delivered directly from the navigation module (values of these signals are zero). For that reason, the information about the other flight parameters was delivered directly to the onboard computer from the numerical simulation.

The firing commands were logged and compared with the data obtained from previous stages of testing (MIL, SIL). Firings of the lateral motors can be ineffective and problematic in laboratory conditions. A significant amount of time is required to prepare the test setup for the single test. After each test, the spent motors must be replaced with new ones. The lateral motors also produce a lot of smoke. To solve the abovementioned problems, instead of using real pulse thrusters, a set of LEDs (light-emitting diodes) was used to visualize the results (installed at the same locations as the original motors). The time of the operation of the single diode was set to be the same as the operation time of the real pulse thruster. The system was monitored using a high-speed camera (up to 5000 frames per second). Using this approach, the experiments can be repeated many times. That way, it was confirmed that the developed control system operates properly and is ready for flight trials. Real tests are required to prove the correctness of the system's operation because even a sophisticated model cannot predict all the physical phenomena that can occur in flight.

6. Conclusions and Future Work

Developing the software for the autopilot is complicated, challenging, and time-consuming. This process can be performed much more efficiently using a model-based design approach than traditional hand coding. Also, the integration and testing phases can be done in a structured and repeatable manner. The probability of introducing human-made errors can be minimized. It might be expected that this approach will gain more and more popularity in such applications. The main contribution of the reported research is the detailed description of model-based development software for the gasodynammically controlled guided HSUAV. Built-in MATLAB tools and custom-developed verification procedures were used to create reliable software for the autopilot. SAVOIR guidelines were applied to perform the overall workflow. MIL, SIL, PIL, and HIL tests were completed to verify the correctness of the solution. The need for manual code implementation on the target hardware was minimized in the described process. Using MBD allowed moving some testing activities to the earlier stages of the system development. The results indicate that a good numerical similarity between the Simulink prototype and the C code was achieved. The presented results partially fill the existing literature gap and extend the simulation study reported in the papers [30, 42].

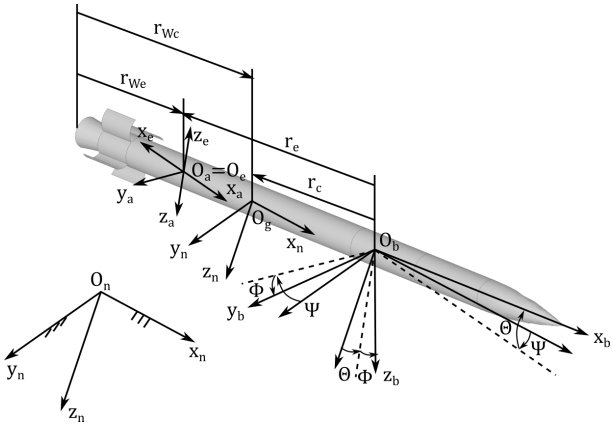

Figure 15. Coordinate systems

Future research might involve code optimization to achieve higher computational efficiency and more experiments in laboratory conditions with real equipment. Hardware-in-the-Loop testing using dedicated Speedgoat target hardware and Simulink Real-Time might be evaluated. The performance of the developed control system will be carefully checked during the real flight tests.

7. Appendix A - Mathematical model of the HSUAV

7.1. Dynamic Equations of Motion

The coordinate systems used in the mathematical model are presented in Figure 15.

The dynamic equations of motion were derived using the linear and angular momentum change theorems for the 6 degrees of freedom rigid body taking into account the mass variation in time. In the body-fixed, non-inertial coordinate system $O_b x_b y_b z_b$ (the origin O_b does not coincide with the center of mass of the vehicle), the equations are as follows:

$$\frac{\tilde{\delta}\vec{\Pi}}{\tilde{\delta}t} + \vec{\Omega} \times \vec{\Pi} = \vec{F}_b \tag{29}$$

$$\frac{\tilde{\delta}\vec{K}_0}{\tilde{\delta}t} + \vec{\Omega} \times \vec{K}_0 + \vec{V}_b \times \vec{\Pi} = \vec{M}_b \tag{30}$$

where $\vec{\Omega} = \begin{bmatrix} P & Q & R \end{bmatrix}^T$ - vector of the angular velocities, $\vec{F}_b = \begin{bmatrix} X_b & Y_b & Z_b \end{bmatrix}^T$ - the vector of forces acting on the vehicle, $\vec{M}_b = \begin{bmatrix} L_b & M_b & N_b \end{bmatrix}^T$ - the vector of torques with respect to point O_b and $\frac{\tilde{\delta}}{\tilde{\delta}t}$ - the local derivative. Linear and angular momentum for a rigid body are:

$$\vec{\Pi} = m\left(\vec{V}_b + \vec{\Omega} \times \vec{r}_C\right) \tag{31}$$

$$\vec{K}_0 = I\vec{\Omega} + \vec{r}_C \times m\vec{V}_b \tag{32}$$

where m - a mass of the object, $\vec{V}_b = \begin{bmatrix} U & V & W \end{bmatrix}^T$ - the velocity vector, $\mathbf{I}$ - a moment of inertia tensor, and $\vec{r}_C$ - the center of mass location with respect to point O_b.

The nonlinear dynamic equations of motion of the projectile are as follows:

$$m\dot{\vec{V}}_b + \dot{\vec{\Omega}} \times \vec{S} + \vec{\Omega} \times m\vec{V}_b + \vec{\Omega} \times \left(\vec{\Omega} \times \vec{S}\right) = \vec{F}_b \quad (33)$$

$$\boldsymbol{I}\dot{\vec{\Omega}} + \vec{S} \times \dot{\vec{V}}_b + \dot{\boldsymbol{I}}\vec{\Omega} + \vec{\Omega} \times \boldsymbol{I}\vec{\Omega} + \vec{\Omega} \times \left(\vec{S} \times \vec{V}_b\right) + \vec{V}_b \times \left(\vec{\Omega} \times \vec{S}\right) = \vec{M}_b \quad (34)$$

where $\vec{S} = m\vec{r}_C$ - the first moment of mass. The propulsion effects are included on the right side of the above-mentioned equations. In the moment's equations, the jet damping effect was neglected because, at low flight altitudes, it is quite small when compared to aerodynamic damping. Next, the cross-products could be replaced by the matrix multiplication, and the skew-symmetric matrix notation, $[]_x$ could be used. As a result, the equations of motion could be written as:

$$\begin{bmatrix} m\mathbf{1} & -[\boldsymbol{S}]_x \\ [\boldsymbol{S}]_x & \mathbf{I} \end{bmatrix} \begin{bmatrix} \dot{\vec{V}}_b \\ \dot{\vec{\Omega}} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \dot{\boldsymbol{I}} \end{bmatrix} \begin{bmatrix} \vec{V}_b \\ \vec{\Omega} \end{bmatrix} + \begin{bmatrix} [\Omega]_x & 0 \\ [\boldsymbol{V_b}]_x & [\Omega]_x \end{bmatrix} \begin{bmatrix} m\mathbf{1} & -[\boldsymbol{S}]_x \\ [\boldsymbol{S}]_x & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \vec{V}_b \\ \vec{\Omega} \end{bmatrix} = \begin{bmatrix} \vec{F}_b \\ \vec{M}_b \end{bmatrix} \quad (35)$$

where $\mathbf{0}$ - zero matrix and $\mathbf{1}$ - unit matrix. In the short form, this is:

$$\boldsymbol{A\dot{x}} + \boldsymbol{\dot{A}x} + \boldsymbol{\omega Ax} = \boldsymbol{F_B} \quad (36)$$

where the state vector has the form $\mathbf{x} = \begin{bmatrix} U & V & W & P & Q & R \end{bmatrix}^T$. This equation can be integrated numerically to obtain the actual values of the state vector.

7.2. Angular Orientation of the Vehicle

Quaternions were used to describe the object orientation:

$$\mathbf{e} = e_0 + e_1\mathbf{i} + e_2\mathbf{j} + e_3\mathbf{k} \quad (37)$$

where e_0, e_1, e_2, e_3 - the real numbers and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are the axes versors. The kinematic equations that connect the rate of change of the quaternion elements with the angular rates are:

$$\begin{bmatrix} \dot{e}_0 \\ \dot{e}_1 \\ \dot{e}_2 \\ \dot{e}_3 \end{bmatrix} = -\frac{1}{2} \begin{bmatrix} 0 & P & Q & R \\ -P & 0 & -R & Q \\ -Q & R & 0 & -P \\ -R & -Q & P & 0 \end{bmatrix} \begin{bmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{bmatrix} - kE \begin{bmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{bmatrix} \quad (38)$$

where k - the feedback coefficient (assumed 1) and E - the bounding equation violation coefficient $E = e_0^2 + e_1^2 + e_2^2 + e_3^2 - 1$. Quaternions are also used to calculate the transformation matrix from the body $O_b x_b y_b z_b$ to the North-East-Down coordinate system $O_n x_n y_n z_n$ as:

$$\Lambda = \begin{bmatrix} e_0^2 + e_1^2 - e_2^2 - e_3^2 & 2(e_1e_2 - e_0e_3) & 2(e_0e_2 + e_1e_3) \\ 2(e_0e_3 + e_1e_2) & e_0^2 - e_1^2 + e_2^2 - e_3^2 & 2(e_2e_3 - e_0e_1) \\ 2(e_1e_3 - e_0e_2) & 2(e_0e_1 + e_2e_3) & e_0^2 - e_1^2 - e_2^2 + e_3^2 \end{bmatrix} \quad (39)$$

The transformation matrix (39) could be used to formulate the relations between the rate of change of the position in $O_n x_n y_n z_n$ and linear velocities in $O_b x_b y_b z_b$:

$$\begin{bmatrix} \dot{x}_n \\ \dot{y}_n \\ \dot{z}_n \end{bmatrix} = \Lambda \begin{bmatrix} U \\ V \\ W \end{bmatrix} \quad (40)$$

Quaternions could be converted to orientation angles (roll, pitch, and yaw) using the relations:

$$\Phi = \arctan \frac{2(e_0e_1 + e_2e_3)}{e_0^2 - e_1^2 - e_2^2 + e_3^2} \quad (41)$$

$$\Theta = \arcsin 2(e_0e_2 - e_1e_3) \quad (42)$$

$$\Psi = \arctan \frac{2(e_0e_3 + e_1e_2)}{e_0^2 + e_1^2 - e_2^2 - e_3^2} \quad (43)$$

The initial quaternion could be calculated from the initial orientation angles in the following way:

$$e_0 = \cos\frac{\Phi}{2}\cos\frac{\Theta}{2}\cos\frac{\Psi}{2} + \sin\frac{\Phi}{2}\sin\frac{\Theta}{2}\sin\frac{\Psi}{2} \quad (44)$$

$$e_1 = \sin\frac{\Phi}{2}\cos\frac{\Theta}{2}\cos\frac{\Psi}{2} - \cos\frac{\Phi}{2}\sin\frac{\Theta}{2}\sin\frac{\Psi}{2} \quad (45)$$

$$e_2 = \cos\frac{\Phi}{2}\sin\frac{\Theta}{2}\cos\frac{\Psi}{2} + \sin\frac{\Phi}{2}\cos\frac{\Theta}{2}\sin\frac{\Psi}{2} \quad (46)$$

$$e_3 = \cos\frac{\Phi}{2}\cos\frac{\Theta}{2}\sin\frac{\Psi}{2} - \sin\frac{\Phi}{2}\sin\frac{\Theta}{2}\cos\frac{\Psi}{2} \quad (47)$$

7.3. Total External Loads

The external forces and torques were calculated as the sum of aerodynamics $\vec{F}_a$ and $\vec{M}_a$, gravity $\vec{F}_g$ and $\vec{M}_g$, thrust $\vec{F}_s$ and $\vec{M}_s$ and lateral solid propellant thrusters $\vec{F}_{sk}$ and $\vec{M}_{sk}$:

$$\vec{F}_b = \vec{F}_a + \vec{F}_g + \vec{F}_s + \vec{F}_{sk} \quad (48)$$

$$\vec{M}_b = \vec{M}_a + \vec{M}_g + \vec{M}_s + \vec{M}_{sk} \quad (49)$$

7.4. Aerodynamic Loads

The vectors of aerodynamic force and moment are calculated with respect to point O_e, so with respect to point O_b they are given as:

$$\vec{F}_a = \begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix} = \frac{1}{2}\rho|\vec{V}_b|^2 S \begin{bmatrix} C_X(\alpha, \beta, Ma) \\ C_Y(\alpha, \beta, Ma) \\ C_Z(\alpha, \beta, Ma) \end{bmatrix} \quad (50)$$

$$\vec{M}_a = \begin{bmatrix} L_a \\ M_a \\ N_a \end{bmatrix} = \frac{1}{2}\rho|\vec{V}_b|^2 Sd \begin{bmatrix} C_l(\alpha, \beta, Ma) \\ C_m(\alpha, \beta, Ma) \\ C_n(\alpha, \beta, Ma) \end{bmatrix} + \vec{r}_e \times \vec{F}_a \quad (51)$$

where ρ - the air density, S - the cross-section area of the fuselage, and d - its diameter, $\vec{r}_e = \vec{r}_{we} - \vec{r}_{wC} + \vec{r}_C$ is the vector describing the location of point O_e with respect to point O_b, $\vec{r}_{we}$ is the position of point O_e with respect to rocket's base and $\vec{r}_{wC}$ is the position of point O_b with respect to the main motor's exit nozzle. Aerodynamic angles of attack α and sideslip β as well as the Mach number Ma are:

$$\alpha = \arctan \frac{W - W_w}{U - U_w} \quad (52)$$

$$\beta = \arcsin \frac{V - V_w}{|\vec{V}_b|} \tag{53}$$

$$Ma = \frac{|\vec{V}_b|}{a} \tag{54}$$

where U_w, V_w, W_w - components of the wind velocity vector in body frame, $|\vec{V}_b| = \sqrt{(U-U_w)^2 + (V-V_w)^2 + (W-W_w)^2}$ and a - the local speed of sound. The aerodynamic coefficients are:

$$\begin{aligned} C_X &= (C_{X_{base\ 0}} + C_{X_{base\ \alpha^2}}\alpha^2 + C_{X_{base\ \beta^2}}\beta^2) \\ &\quad + (C_{X_{eng\ 0}} + C_{X_{eng\ \alpha^2}}\alpha^2 + C_{X_{eng\ \beta^2}}\beta^2)\delta_{eng} \\ C_Y &= C_{Y_0} + C_{Y_\beta}\beta \\ C_Z &= C_{Z_0} + C_{Z_\alpha}\alpha \\ C_l &= C_{l_0} + (C_{l_{p_0}} + C_{l_{p_{\alpha^2}}}\alpha^2 + C_{l_{p_{\beta^2}}}\beta^2)\frac{Pd}{2|\vec{V}_b|} \\ C_m &= C_{m_0} + C_{m_\alpha}\alpha \\ C_n &= C_{n_0} + C_{n_\beta}\beta \end{aligned} \tag{55}$$

where $C_{X_{base\ 0}}$ - axial force coefficient (for $\alpha = \beta = 0°$), $C_{X_{base\ \alpha^2}}$, $C_{X_{base\ \beta^2}}$ - yaw-axial force coefficients, C_{Y_β} - side force with the angle of sideslip derivative, C_{Z_α} - normal force with respect to the angle of attack derivative, C_{l_0} - spin driving rolling moment coefficient, $C_{l_{p_0}}$ - spin damping derivative, C_{m_α} - pitching moment with respect to the angle of attack derivative, C_{n_β} - yawing moment derivative with respect to sideslip angle, C_{m_q} - pitching moment coefficient derivative with pitch rate and C_{n_r} - yawing moment coefficient derivative with yaw rate. Additionally, the parameter δ_e describes the main motor state ($\delta_e = 0$ for the active phase of flight, and $\delta_e = 1$ after main motor burnout, for gliding flight). When the main motor operates at the beginning of the flight, the base drag is significantly lower than after the main motor burnout. The axial force coefficient was obtained for two system configurations (main motor on/off), and δ_e is used during the simulation to switch between aerodynamic data tables. The values of the coefficients can be found in [30].

7.5. Gravity Loads

Gravitational acceleration in the $O_n x_n y_n z_n$ coordinate system is given as $\vec{g} = \begin{bmatrix} 0 & 0 & g_0 \end{bmatrix}^T$. Gravitational acceleration was assumed to be constant $g_0 = 9.80665$ m/s^2. Gravitational force and torques are given as:

$$\vec{F}_g = T_g^b m \begin{bmatrix} 0 \\ 0 \\ g_0 \end{bmatrix} \tag{56}$$

$$\vec{M}_g = \vec{r}_C \times \vec{F}_g \tag{57}$$

where T_g^b - the transformation matrix from gravitational to body coordinate system (given by equation 22).

7.6. Main Solid Propellant Motor Loads

Thrust vector can deviate from the geometric longitudinal axis of the HSUAV by angle Θ_T in pitch plane and Ψ_T in the yaw plane, so the main motor loads are:

$$\vec{F}_s = F_p(t) \begin{bmatrix} \cos\Theta_T \cos\Psi_T \\ \cos\Theta_T \sin\Psi_T \\ -\sin\Psi_T \end{bmatrix} \tag{58}$$

where $F_p(t)$ is the instantaneous magnitude of the thrust force (obtained in the simulation using lookup-table procedure). Torque generated by the main motor with respect to point O_b could be obtained as:

$$\vec{M}_s = (-\vec{r}_{wC} + \vec{r}_C) \times \vec{F}_s \tag{59}$$

7.7. Lateral Thrusters

The gasodynamic control system is based on a set of identical correction lateral thrusters at the circumference of the body. The thrust and torque generated by the i-th thruster are:

$$\vec{F}_{sk_{i,j}} = F_{p_{sk}}(t) \begin{bmatrix} 0 \\ \sin\Phi_{i,j} \\ -\cos\Phi_{i,j} \end{bmatrix} \tag{60}$$

$$\vec{M}_{sk_{i,j}} = \left(\vec{r}_{sk_{i,j}} - \vec{r}_{wC} + \vec{r}_C\right) \times \vec{F}_{sk_{i,j}} \tag{61}$$

where $F_{p_{sk}}(t)$ - the instantaneous thrust force generated by the lateral thruster, $i = 1, \ldots, M$ - the layer number ($i = 1$ means the layer located closest to the rocket base), index $j = 1, \ldots, N$ - the number of an individual thruster in a particular layer, $\Phi_{i,j}$ - the azimuth angle of a thruster is a particular layer, $\vec{r}_{sk_{i,j}}$ - the vector describing the position of the layer with respect to the rocket's base. The layers are located 1.10 m, 1.15 m, 1.20 m, and 1.25 m from aft, respectively. The total force and torque generated by the gasodynamic control system is calculated as the sum of load from all the lateral motors:

$$\vec{F}_{sk} = \sum_{i=1}^{M}\sum_{j=1}^{N} \vec{F}_{sk_{i,j}} \tag{62}$$

$$\vec{M}_{sk} = \sum_{i=1}^{M}\sum_{j=1}^{N} \vec{M}_{sk_{i,j}} \tag{63}$$

7.8. Mass and inertial properties

The instantaneous mass of the vehicle is calculated as:

$$m(t) = m_0 - \frac{m_p}{I_c}\int_{t_0}^{t} F_p(t)dt \tag{64}$$

where m_0 is the initial mass of the object at time t_0, m_p is the mass of the propellant, and I_c is the total impulse given as:

$$I_c = \int_{t_0}^{t_k} F_p(t)dt \tag{65}$$

where t_k - time of propellant burnout. During the powered flight phase, the vehicle's mass center position vector $\vec{r}_{wC}$ measured from the nozzle of the main motor has the following components:

$$\vec{r}_{wC} = \left[x_{cg_0} - \frac{x_{cg_0} - x_{cg_k}}{I_c}\int_{t_0}^{t} F_p(t)dt \quad 0 \quad 0\right] \tag{66}$$

where x_{cg_0} is the center of mass position on the $O_b x_b$ axis during launch and x_{cg_k} is the center of mass position on the $O_b x_b$ axis after the propellant burnout. The change of moments of inertia also depends on time and is calculated as:

$$I_{ij}(t) = I_{ij_0} - \frac{I_{ij_0} - I_{ij_k}}{I_c} \int_{t_0}^{t} F_p(t)dt \qquad (67)$$

where I_{ij_0} is the moment of inertia tensor component during launch and I_{ij_k} is the moment of inertia tensor component after the propellant burnout.

AUTHORS

Mariusz Jacewicz* – Institute of Aeronautics and Applied Mechanics, Warsaw University of Technology, Nowowiejska 24, 00-665 Warsaw, Poland, e-mail: mariusz.jacewicz@pw.edu.pl.

Dariusz Miedziński – Institute of Aeronautics and Applied Mechanics, Warsaw University of Technology, Nowowiejska 24, 00-665 Warsaw, Poland, e-mail: dariusz.miedzinski2.dokt@pw.edu.pl.

Grzegorz Chmaj – DRI Solutions, Chełmska 21, 00-724 Warsaw, Poland, e-mail: office@drisolutions.pl.

Robert Głębocki – Institute of Aeronautics and Applied Mechanics, Warsaw University of Technology, Nowowiejska 24, 00-665 Warsaw, Poland, e-mail: robert.glebocki@pw.edu.pl.

*Corresponding author

ACKNOWLEDGEMENTS

This work was supported by The National Centre for Research and Development (grant number DOB-SZAFIR/03/B/002/01/2021).

References

[1] B. G. Abdelaty, A. Hamdy, and A. N. Ouda, "Flight vehicle autopilot system: From design to implementation", *Automation, Control and Intelligent Systems*, vol. 6, no. 6, 2019, pp. 62–72, doi: 10.11648/j.acis.20180606.11.

[2] N. Ajwad. "Evaluation of Automatic Code Generation Tools". Master's thesis, Lund University, Department of Automatic Control, 2007.

[3] K. D. Alpaslan, "Challenges of Weapon Systems Software Development", *Journal of Naval Science and Engineering*, vol. 5, no. 3, 2009, pp. 104–116.

[4] A. Alsaraj and G. Stuffle, "Investigation of hardware-in-loop simulation (HILS) for guidance system". vol. 1, 2015, pp. 704–708, doi: 10.1109/IAEAC.2015.7428646.

[5] J. Arm, Z. Bradac, P. Fiedler, and V. Kaczmarczyk, "Characterizing the Simulink-based Code Generation Toolchain for Safety-critical Applications in an ARM Cortex-R Target", *IFAC-PapersOnLine*, vol. 52, no. 27, 2019, pp. 271–276, doi: 10.1016/j.ifacol.2019.12.672.

[6] A. Arregi, F. Schriever, C. Arias, A. Jung, and G. T. D. Gmbh, "Ensuring Numerical Reproducibility for Model-Based Software Engineering". In: *8th European Conference for Aeronautics and Space Sciences*, vol. 1, 2019, pp. 1–10, doi: 10.13009/EUCASS2019-790.

[7] R. Bond, S. Bemrich, J. Connelly, G. Pendergrass, and J. Hulsey, "Missile Guidance Processor Software Development - A Case Study". In: *Proceedings. Real-Time Systems Symposium*, vol. 1, 1988, pp. 60–68, doi: 10.1109/real.1988.51101.

[8] H. Bourbouh, P.-l. Garoche, T. Loquen, É. Noulard, and C. Pagetti, "CoCoSim, a Code Generation Framework for Control/Command Applications An overview of CoCoSim for multi-periodic discrete Simulink models". In: *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, vol. 1, 2020.

[9] N. P. Brayanov and A. V. Stoynova, "Evaluation of Model-Based Code Generation for Embedded Systems–Mature Approach for Development in Evolution", *International Journal of Computer and Information Engineering*, vol. 13, no. 8, 2019, pp. 455–460.

[10] B. Carpenter, "Automatic Code Generation for Spacecraft Attitude Determination and Control". In: *2014 IEEE Aerospace Conference*, vol. 1, 2014, pp. 1–5, doi: 10.1109/AERO.2014.6836510.

[11] L. chang and L. kui, "Simulation of underwater vehicle control based on code generation technology". In: *2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, vol. 1, 2021, pp. 773–777, 10.1109/ICBAIE52039.2021.9390015.

[12] J. M. Choe, L. Arnedo, Y. Lee, Z. Sorchini, A. Mignogna, I. Agirman, and H. Kim, "Model-Based Design and DSP Code Generation using Simulink® for Power Electronics Applications", *ICPE 2019 - ECCE Asia - 10th International Conference on Power Electronics - ECCE Asia*, vol. 3, 2019, pp. 923–926, doi: 10.23919/icpe2019-ecceasia42246.2019.8797107.

[13] J. E. Craft. "A User's Experience with Model-Based Design for GNC-Based Systems". https://it.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/aerospace-defense/files/2008/LM_Craft_MWSymp.pdf.

[14] J. E. Craft and B. Rusk. "A User's Experience with Simulink and Stateflow for Real-Time Embedded Applications". https://it.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/aerospace-defense/files/2007/MADC_2007_08_Craft_TMW.pdf.

[15] E. Denney and S. Trac, "A Software Safety Certification Tool for Automatically Generated Guidance, Navigation and Control Code". In: *2008 IEEE Aerospace Conference*, vol.

1, Big Sky, MT, USA, 2008, pp. 1–11, doi: 10.1109/AERO.2008.4526576.

[16] T. Erkkinen, "Model Style Guidelines for Flight Code Generation". In: *AIAA Modeling and Simulation Technologies Conference and Exhibit*, vol. 2, 2005, pp. 708–715, doi: 10.2514/6.2005-6216.

[17] T. Erkkinen and M. Conrad, "Safety-Critical Software Development Using Automatic Production Code Generation", *SAE Technical Papers*, vol. 1, no. April, 2007, doi: 10.4271/2007-01-1493.

[18] T. Erkkinen and B. Potter, "Model-Based Design for DO-178B with Qualified Tools". In: *AIAA Modeling and Simulation Technologies Conference and Exhibit*, vol. 1, 2009, pp. 1–13, doi: 10.2514/6.2009-6233.

[19] European Space Research and Technology Centre. "Guidelines for the Automatic Code Generation for AOCS/GNC Flight SW Handbook: Volume 1 - General concepts". Technical report, SAVOIR, 2022.

[20] European Space Research and Technology Centre. "Guidelines for the Automatic Code Generation for AOCS/GNC Flight SW Handbook: Volume 2 - Mathworks specific guidelines". Technical report, SAVOIR, 2022.

[21] M. Fakih and S. Warsitz, "Automatic SDF-based Code Generation from Simulink Models for Embedded Software Development". In: *5th International Workshop on. High Performance Energy Efficient Embedded Systems*, vol. 1, 2017.

[22] I. Fey and I. Stürmer, "Code Generation for Safety-Critical Systems-Open Questions and Possible Solutions", *SAE Technical Papers*, vol. 1, 2008, doi: 10.4271/2008-01-0385.

[23] J. C. M. Fraticelli. "Auto Code Generation for Simulink-Based Attitude Determination Control System". Technical report, National Aeronautics and Space Administration, 2012.

[24] J. C. M. Fraticelli. "Simulink Code Generation. Tutorial for generating C code from Simulink Models using Simulink Coder". Technical report, National Aeronautics and Space Administration, 2012.

[25] M. Gao, Z. Yongwei, S. Yang, and D. Fang, "Trajectory Correction Capability Modeling of the Guided Projectiles with Impulse Thrusters", *Engineering Letters*, vol. 24, 2016, pp. 11–18.

[26] R. Głębocki and M. Jacewicz, "Parametric Study of Guidance of a 160-mm Projectile Steered with Lateral Thrusters", *Aerospace*, vol. 7, no. 5, 2020, doi: 10.3390/aerospace7050061.

[27] N. Holliday. "Software Development with Real-Time Workshop Embedded Coder". https://www.mathworks.com/content/dam/mathworks/tag-team/Objects/m/48884_Thales_DualCore.pdf.

[28] R. Hýl and R. Wagnerová, "Fast Development of Controllers with Simulink Coder". In: *2017 18th International Carpathian Control Conference, ICCC 2017*, vol. 1, 2017, pp. 406–411, doi: 10.1109/CarpathianCC.2017.7970434.

[29] M. Jacewicz, R. Głębocki, and R. Ożóg, "Monte-Carlo Based Lateral Thruster Parameters Optimization for 122 mm Rocket". In: R. Szewczyk, C. Zieliński, and M. Kaliczyńska, eds., *Automation 2020: Towards Industry of the Future*, vol. 1, Cham, 2020, pp. 125–134.

[30] M. Jacewicz, P. Lichota, D. Miedziński, and R. Głębocki, "Study of Model Uncertainties Influence on the Impact Point Dispersion for a Gasodynamicaly Controlled Projectile", *Sensors*, vol. 22, no. 9, 2022, doi: 10.3390/s22093257.

[31] M. C. Jackson and J. R. Henry, "Orion GN&C Model Based Development: Experience and Lessons Learned". In: *AIAA Guidance, Navigation, and Control Conference 2012*, vol. 1, 2012, pp. 1–16, doi: 10.2514/6.2012-5036.

[32] S. Jacobitz and X. Liu-Henke, "The Seamless Low-cost Development Platform LoRra for Model based Systems Engineering". In: *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development - MODELSWARD*, vol. 1, 2020, pp. 57–64, doi: 10.5220/0008993500570064.

[33] S. Jacobitz and X. Liu-Henke, "Automatic Code Generation for a Seamless Low-cost Development Platform". In: *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*, vol. 1, 2022, pp. 294–301, doi: 10.5220/0010894300003119.

[34] H. Jiang, H. Cheng, S. Guo, and X. Li, "Partition Based Differential Testing for Finding Embedded Code Generation Bugs in Simulink". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*, vol. 1, 2023, pp. 1–6, doi: 10.1109/DAC56929.2023.10247877.

[35] T. Jitpraphai and M. Costello, "Dispersion Reduction of a Direct Fire Rocket Using Lateral Pulse Jets", *Journal of Spacecraft and Rockets*, vol. 38, no. 6, 2001, pp. 929–936, doi: 10.2514/2.3765.

[36] E. H. Kapeel, H. Hendy, A. M. Kamel, and Y. Z. Elhalwagy, "Terminal Guidance Law Hardware In The Loop Simulation Against Maneuvering Targets Using FPGA Based Floating Point Approach". In: *2021 International Telecommunications Conference (ITC-Egypt)*, vol. 1, no. 1, 2021, pp. 1–6, 10.1109/ITC-Egypt52936.2021.9513941.

[37] W. Kom Fotso and X. Querol, "Evaluation of a Modeling and Automatic C Code Generation Toolset as an Open Source Alternative Solution". In: *6th European Congress on Embedded Real-Time Software and Systems*, vol. 1, 2012.

[38] J. Krizan, L. Ertl, M. Bradac, M. Jasansky, and A. Andreev, "Automatic Code Generation from Matlab/Simulink for Critical Applications". In:

Canadian Conference on Electrical and Computer Engineering, vol. 1, 2014, pp. 1–6, doi: 10.1109/CCECE.2014.6901058.

[39] K. Kshirsagar, S. Rane, P. Shah, and R. Sekhar, "Automatic Code Generation in Model Based Design and Digital Signal Processing". In: *2023 4th International Conference for Emerging Technology*, vol. 1, 2023, pp. 1–6, doi: 10.1109/INCET57972.2023.10170569.

[40] V. Lambersky, "Model Based Design and Automated Code Generation from Simulink Targeted for TMS570 MCU", *EDERC 2012 - Proceedings of the 5th European DSP in Education and Research Conference*, vol. 1, 2012, pp. 225–228, doi: 10.1109/EDERC.2012.6532260.

[41] G. Li, R. Zhou, R. Li, W. He, G. Lv, and T. J. Koo, "A Case Study on SDF-Based Code Generation for ECU Software Development". In: *2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*, vol. 1, 2011, pp. 211–217, doi: 10.1109/COMPSACW.2011.45.

[42] P. Lichota, M. Jacewicz, R. Głębocki, and D. Miedziński, "Wavelet-Based Identification for Spinning Projectile with Gasodynamic Control Aerodynamic Coefficients Determination", *Sensors*, vol. 22, no. 11, 2022, doi: 10.3390/s22114090.

[43] P. Lichota, M. Jacewicz, and J. Szulczyk, "Spinning Gasodynamic Projectile System Identification Experiment Design", *Aircraft Engineering and Aerospace Technology*, vol. 92, no. 3, 2018, pp. 452–459, doi: 10.1108/AEAT-06-2019-0124.

[44] F. Luo and Z. Huang, "Embedded C Code Generation and Embedded Target Development Based on RTW-EC", *Proceedings - 2010 3rd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2010*, vol. 5, 2010, pp. 532–536, doi: 10.1109/ICCSIT.2010.5563906.

[45] X. Lv, "Automatic Generation of Single-Phase SVPWM Embedded Code", *Proceedings - 2019 International Conference on Artificial Intelligence and Advanced Manufacturing, AIAM 2019*, vol. 1, 2019, pp. 460–463, doi: 10.1109/AIAM48774.2019.00097.

[46] J. M. Maroli, B. A. Morris, J. A. Blystone, and A. M. Oconnor, "Utilizing Code Generation from Models for Electric Aircraft Motor Controller Flight Software". In: *AIAA AVIATION 2023 Forum*, vol. 1, 2023, doi: 10.2514/6.2023-4274.

[47] MathWorks. "NASA Uses Stateflow and Simulink Coder to Generate Fault-Protection Code for Deep Space 1". https://www.mathworks.com/company/user_stories/nasa-uses-stateflow-and-simulink-coder-to-generate-fault-protection-code-for-deep-space-1.html.

[48] T.-Y. Moon, S.-H. Seo, J.-H. Kim, S.-H. Hwang, and J. W. Jeon, "Simulation with Consideration of Hardware Characteristics and Auto-generated Code Using MATLAB/Simulink". In: *2007 International Conference on Control, Automation and Systems*, vol. 1, 2007, pp. 1494–1498, doi: 10.1109/ICCAS.2007.4406575.

[49] I. Moran and D. T. Altilar, "Three Plane Approach for 3D True Proportional Navigation". In: *AIAA Guidance, Navigation, and Control Conference and Exhibit*, vol. 8, 2005, doi: 10.2514/6.2005-6457.

[50] M. Muresan and D. Pitica, "Simulating Embedded Targets for Efficient Code Implementation". In: *ISSE 2009: 32nd International Spring Seminar on Electronics Technology: Hetero System Integration, the path to New Solutions in the Modern Electronics - Conference Proceedings*, vol. 1, 2009, pp. 1–4, doi: 10.1109/ISSE.2009.5206997.

[51] S. Nadir and D. Streitferdt, "Software Code Generator in Automotive Field", *Proceedings - 2015 International Conference on Computational Science and Computational Intelligence, CSCI 2015*, vol. 1, 2016, pp. 13–17, doi: 10.1109/CSCI.2015.186.

[52] O. Netland and A. Skavhaug, "Software Module Real-Time Target: Improving Development of Embedded Control System by Including Simulink Generated Code Into Existing Code". In: *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, vol. 1, 2013, pp. 232–235, doi: 10.1109/SEAA.2013.51.

[53] J.-g. Niu, Z.-j. Wang, C.-h. Xu, P.-b. Zhang, and B. Zhang, "Research on Fuzzy Logic Control Based on Targetlink Automatic Code". In: *2019 International Conference on Communications, Information System and Computer Engineering (CISCE)*, vol. 1, 2019, pp. 148–151, doi: 10.1109/CISCE.2019.00041.

[54] D. Oddenino. "AUTOCODING WORKING GROUP Automatic Code Generation for AOCS Flight SW", 2018.

[55] L. Otava, "Simulink Model Code Generation for Motor Control Applications". In: *Proceedings of the 21th Conference STUDENT EEICT 2015*, vol. 1, no. 2, 2015, pp. 470–474.

[56] R. Ożóg, M. Jacewicz, and R. Głębocki, "Modified Trajectory Tracking Guidance for Artillery Rocket", *Journal of Theoretical and Applied Mechanics*, vol. 58, no. 3, 2020, pp. 611–622, doi: 10.15632/jtam-pl/121981.

[57] I. E. Putro and H. Septanto, "Real-Time Simulation of Embedded Controller for Missile", *Jurnal Teknologi Dirgantara*, vol. 1, 2019, pp. 129–140.

[58] A. Rugina and J. Dalbin, "Experiences with the GENE-AUTO Code Generator in the Aerospace Industry". In: *ERTS2 2010, Embedded Real Time Software & Systems*, vol. 1, 2010.

[59] M. H. Schwarz, H. Sheng, A. Sheleh, and J. Boercsoek, "Matlab® / Simulink® Generated Source Code for Safety Related Systems". In: *AICCSA 08 - 6th IEEE/ACS International*

Conference on Computer Systems and Applications, vol. 1, 2008, pp. 1058–1063, doi: 10.1109/AICCSA.2008.4493678.

[60] A. Soldati, R. Zanichelli, F. Brugnano, and C. Concari, "Implementing Discrete PID Controllers: Benchmarking Manual vs. Automatic Generation of Embedded Code". In: *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, 2016, pp. 178–183, doi: 10.1109/IECON.2016.7793334.

[61] G. Strub, V. Gassmann, S. Theodoulis, S. Dobre, and M. Basset, "Hardware-in-the-Loop Experimental Setup Development for a Guided Projectile in a Wind Tunnel". In: *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, vol. 1, no. 1, 2014, pp. 458–463, doi: 10.1109/AIM.2014.6878120.

[62] I. Stürmer, "Certification of Model-based Code Generators – Open Problems and Possible Solutions". In: *Embedded Real Time Software and Systems (ERTS2008)*, vol. 1, Toulouse, France, 2008.

[63] A. Szklarski, R. Głębocki, and M. Jacewicz, "Impact Point Prediction Guidance Parametric Study for 155 mm Rocket Assisted Artillery Projectile with Lateral Thrusters", *Archive of Mechanical Engineering*, vol. 67, no. 1, 2020, pp. 31–56, doi: 10.24425/ame.2020.131682.

[64] S. Tamblyn, J. Henry, and E. King, "A model-based design and testing approach for orion gn&c flight software ddevelopment". In: *IEEE Aerospace Conference Proceedings*, vol. 1, 2010, pp. 1–12, doi: 10.1109/AERO.2010.5446802.

[65] N. Tancredi. "DiSTERaP Distributed Simulation Test Environment for Rapid Prototyping". https://www.matlabexpo.com/content/dam/mathworks/mathworks-dot-com/images/events/matlabexpo/it/2018/distributed-simulation-test-environment-for-rapid-prototyping-disterap.pdf.

[66] A. Toom, T. Naks, M. Pantel, M. Gandriau, and Indrawati, "Gene-Auto: an Automatic Code Generator for a Safe Subset of Simulink/Stateflow and Scicos". In: *Proceeding of the 4th European Congress on Embedded Real Time Software*, vol. 1, 2008.

[67] G. Walde and R. Luckner, "Bridging the tool gap for model-based design from flight control function design in simulink to software design in scade". In: *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, vol. 1, 2016, pp. 1–10, 10.1109/DASC.2016.7778044.

[68] G. Waxenegger-Wilfing, K. Dresia, M. Oschwald, and K. Schilling, "Hardware-In-The-Loop Tests of Complex Control Software for Rocket Propulsion Systems". In: *71st International Astronautical Congress*, vol. 1, 2020.

[69] Q. Wu, J. Qiu, C. Zhu, and Y. Wang, "Automatic Fast Experiment System Design Based on Matlab Embedded Code". In: *Proceeding - 2021 China Automation Congress, CAC 2021*, vol. 1, 2021, pp. 7360–7363, doi: 10.1109/CAC53003.2021.9728568.

[70] P. Xu, M. Kondo, and M. Edahiro, "Code Generation from Simulink Models with Task and Data Parallelism", *International Journal Of Computers & Technology*, vol. 21, no. 2, 2021, pp. 1–13, doi: 10.24297/ijct.v21i.9004.

[71] A. Yahyaabadi, P. Harrison, and P. Ferguson, "Auto Code Generation for Onboard Space Object Detection and Other Flight Software Applications - A Feasibility Study". In: *CASI ASTRO 2019*, vol. 1, Quebec, Canada, 2019, pp. 1–19.

[72] Z. Yu, Z. Su, Y. Yang, J. Liang, Y. Jiang, A. Cui, W. Chang, and R. Wang, "Mercury: Instruction Pipeline Aware Code Generation for Simulink Models", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, 2022, pp. 4504–4515, doi: 10.1109/TCAD.2022.3199967.

[73] P. Zhang, J. Gu, E. Milios, and P. Huynh, "Navigation with IMU/GPS/digital Compass with Unscented Kalman Filter". In: *IEEE International Conference Mechatronics and Automation, 2005*, vol. 3, 2005, pp. 1497–1502, doi: 10.1109/ICMA.2005.1626777.

[74] Q. Zhang and W. Pei, "DSP Processer-in-the-Loop Tests Based on Automatic Code Generation", *Inventions*, vol. 7, no. 1, 2022, pp. 1–9, doi: 10.3390/inventions7010012.

[75] Y. Zhang, Y. Zhang, and Y. Zhang, "Using Automatic Code Generation to Accelerate Control Algorithm Design for FPGAs". In: *2022 2nd International Conference on Algorithms, High Performance Computing and Artificial Intelligence, AHPCAI 2022*, vol. 1, 2022, pp. 68–72, 10.1109/AHPCAI57455.2022.10087457.

[76] K. Ćosić, I. Kopriva, T. Kostić, M. Slamić, and M. Volarević, "Design and Implementation of a Hardware-in-the-Loop Simulator for a Semi-Automatic Guided Missile System", *Simulation Practice and Theory*, vol. 7, no. 2, 1999, pp. 107–123, doi: 10.1016/S0928-4869(98)00027-5.