Modellbasierte Entwicklung von Flugreglern vollaktuierter Multikopter durch Codeexport von Simulink nach ROS

Model-based Development Process of Flight Controllers for Fully Actuated Multicopters by Code Export from Simulink to ROS

Micha Schuster, TU Dresden, Professur für Dynamik und Mechanismentechnik, micha.schuster@tu-dresden.de

David Bernstein,TU Dresden, Professur für Dynamik und Mechanismentechnik, david.bernstein@tu-dresden.de

Willy Reichert, TU Dresden, Professur für Dynamik und Mechanismentechnik

Klaus Janschek, TU Dresden, Professur für Automatisierungstechnik, klaus.janschek@tu-dresden.de

Michael Beitelschmidt, TU Dresden, Professur für Dynamik und Mechanismentechnik, michael.beitelschmidt@tu-dresden.de

Kurzfassung

In diesem Beitrag wird ein geschlossener Entwicklungsprozess für Flugregler vollaktuierter Multikopter auf Basis von Simulink und ROS vorgestellt. Dieser Entwicklungsprozess, der sich am V-Modell nach VDI/VDE 2206 orientiert, ermöglicht einerseits eine schnelle, effiziente und detailierte Modellierung und Simulation durch die Nutzung von MATLAB/Simulink als Simulationsumgebung. Andererseits wird durch einen automatisierten Export des Simulationsmodells in ROS/C++ Code und anschließende Kompilierung ein hoch performanter Code erzeugt, der im Vergleich zu Simulink sehr geringe Leistungsanforderungen an die Zielhardware stellt und eine zeitaufwändige und fehleranfällige manuelle Übertragung nach C++ erspart. ROS bietet eine breite Hardwareunterstützung und kann durch seine Node-Struktur und standardisierte Kommunikationsschnittstellen leicht in bestehende Softwarestrukturen eingebunden werden. Der vorgestelle Entwicklungsprozess wird beispielhaft auf die Entwicklung eines Beobachters für Interaktionskräfte angewendet und im realen Flugexperiment validiert.

Abstract

In this paper, a closed-loop development process for flight controllers of fully actuated multicopters based on Simulink and ROS is presented. The development process, which is based on the V-model according to the standard VDI/VDE 2206, enables fast, efficient and detailed modeling and simulation by using MATLAB/Simulink as the simulation environment. In addition to this, an automated export of the simulation model to ROS/C++ code and following compilation generates a high-performance code. This has, compared to Simulink, very low performance requirements on the target hardware and avoids a time-consuming and error-prone manual transfer to C++. ROS offers broad hardware support and can be easily integrated into existing software structures thanks to its node structure and standardized communication interfaces. The presented development process is exemplarily applied to the development of an observer for interaction forces and validated in a real flight experiment.

1 Introduction

1.1 Motivation

Small, unmanned multicopters have been established on the market for many years and cover a wide range from small remote-controlled toy drones to large, semi-autonomous, commercially deployed multicopters. Thereby, the field of application of these conventional multicopters is mainly limited to camera-based applications, in which the multicopter only acts as a flying sensor carrier. In such applications, contact with the environment is undesirable and represents a major safety risk. In addition to this large commercially available application area, the development of multicopters for manipulation tasks is subject of recent research. In contrast to the camera applications mentioned above, physical contact with the environment is part of the operational scenario here. For these novel applications, the development of new multicopter concepts is required. One of these concepts relies on multicopters with tilted rotors that have the ability to apply forces and torques independently in all spatial directions without the need to change the orientation. Furthermore, they are able to maintain (within certain limits) an arbitrary static orientation. This special class is called fully actuated multicopters and requires special fully actuated flight controllers. Interaction tasks also bring a variety of new challenges, such as the application and determination of working and contact forces and torques, or the exact positioning relative to the target object. In order to cope with these tasks with novel multicopters, increased requirement profile and increased risk, new control concepts, good system modeling and intensive testing are necessary. The development process can be divided into three main stages: (1) Modeling, controller development and simulation, (2) the generation and integration of high-performance code for the respective target hardware and (3) extensive testing in simulation and flight experiment. The development is an interactive process with frequent changes between the mentioned phases. In order to enable an efficient development process, a simple transition between the stages plays an important role in addition to the optimization within the respective stages. In this paper, a methodology is presented, how this process can be designed efficiently for the development of fully actuated multicopters for tactile tasks.

1.2 State of the Art

The development of multicopters for new operational fields often includes adaptions up to redevelopment of existing flight controllers. For this purpose, different practices and tools for controller development and code execution have become established. One common way is to adapt the widely used PX4 autopilot, which originally was developed for conventional multicopters, to make it capable for the control of interaction flight maneuvers [1, 2, 6, 14]. Also, MATLAB®/Simulink® with its provided hardware interfaces is often used for development and execution of flight controllers [4, 5, 10]. Simulink in particular is very well suited to efficiently implement complex control algorithms thanks to its graphical programmability and numerous powerful toolboxes. One drawback is the requirement of higher performance target hardware, which must be able to run a Simulink model sufficiently fast compared to running fully compiled control algorithms. Those compiled control algorithms can either be realized as fully in-house developments or by using existing frameworks such as the Robot Operating System (ROS), which also provides libraries for multicopter control development [11, 12]. In recent projects [2–4, 6], ROS is mostly used as middleware between several guidance, navigation, control and peripheral components, benefiting from its broad hardware support for robotics components. As of now, ROS has not been widely used as a stand-alone tool for flight control implementation and execution for physical interactive tasks. A downside is the time-consuming control development in C++. Furthermore the GAZEBO simulator [7] that is commonly used in combination with ROS, is specialized on software-in-the-loop simulations for controller verification but is less suitable for precise physical modeling, which is of high importance for physical interactive tasks.
To realize a rapid development process, in addition to a suitable simulation environment and fast code execution, it is necessary to easily switch between simulation and real flight experiment. In [9] a development process using code export from Simulink is shown. However it does not use a standardized communication interface, which increases the effort for integrating new modules into the framework. The work presented in [10] shows a comprehensive development process based on a main controller running in Simulink and a PX4 backup controller with a strong focus on safety for outdoor experiments. This enormous initial effort will probably not pay off for projects that will remain in a flight lab environment.

1.3 Aim and Outline of this Paper

This paper presents a new methodology for rapid development of advanced multicopter flight control systems. Chapter 2 deals with the general development process based on the V-model from VDI standard 2206 [15]. The development process is exemplarily applied to the development of a wrench observer for aerial mobile manipulation in chapter 3.

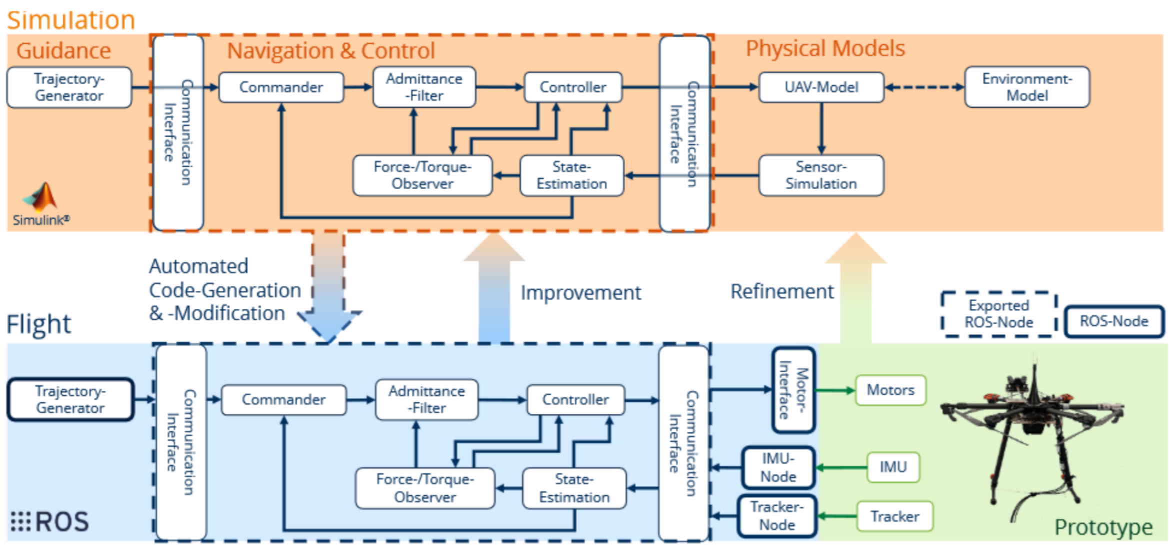

Figure 1 Overview of the code architecture in Simulink and ROS. Orange: Simulink, blue: ROS, green: multicopter prototype

Originally published at: Fachtagung VDI MECHATRONIK 2022, Darmstadt, 23.-24.03.2022, DOI: 10.26083/tuprints-00020963

2 Development Process

2.1 Choice of the Development Environment

Rapid and efficient control development benefits from a powerful simulation environment, fast code execution in the flight experiment, fast and easy switching between simulation and flight experiment and a wide hardware support for sensors and actors. As mentioned in section 1.2, MATLAB/Simulink is one of the top simulation environments in system modeling and control design and is widely used. ROS primarily benefits from its broad hardware support and the very fast execution of the compiled code by default. In addition, its standardized node architecture and messaging framework enables clear and transparent code structure and efficient communication between multiple nodes. As of release R2019b, Simulink provides a ROS interface for coupling both of the platforms. As a downside this ROS interface has a low performance and is to slow for real-time control with high sampling rates. Thus, the presented development process will build on Simulink and ROS and overcome the low speed of the ROS interface in Simulink with an automated code export as a stand alone ROS node.

2.2 Simulation Model Architecture

The overall architecture of the simulation model is shown in Fig. 1 (top). It can generally be divided into three main components:

1) **Physical Models**
This part of the simulation model contains the submodels that represent physical processes. This includes a model of the multicopter with its inertia, the modeling of the motors as well as the thrust generation. Besides this, the environment is modeled to simulate physical interaction tasks. The effect of the environment on the multicopter is realized by external forces or torques. As another physical submodel, the sensors such as the IMU or an external tracking system are implemented with their respective characteristics.

2) **Guidance, Navigation, Control (GNC)**
This module is the main subject of the development process. It contains the navigation and control functionalities such as the flight controller, a state estimator and a commander that manages the respective flight modes (lift off, hovering, trajectory tracking, contact, landing, error, ...). For tactile interaction tasks it is extended by a force-torque observer to determine the external loads applied to the multicopter from the environment. An admittance filter adapts the desired pose based on the external loads to achieve the desired mechanical compliance of the system. Depending on the system design, a trajectory generator as guidance functionality can also be integrated into this module. As in figure 1, the module without trajectory generator is named Navigation and Control (NC) module within this paper due to not containing the complete guidance functionality.

3) **Communication Interface**
The communication interface contains all ROS specific functionalities to keep the navigation and control (NC) module general. When the model is used for simulation in Simulink, the communication interface is bypassed by the signals of the physical models and the guidance. For the code export the NC module is disconnected from physical models and guidance and connected to the communication interface instead. The communication interface contains the blocks *publish* and *subscribe* from the Simulink-ROS library for all necessary topics for enabling the automatic embedded code generation for the communication between the NC module and the ROS nodes of motors, sensors, and trajectory generator.

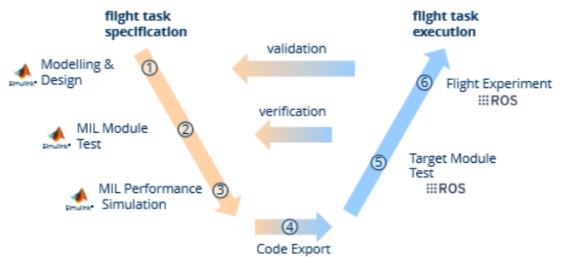

Figure 2 Development process based on the V-model

2.3 Steps of the Development Process

The development process presented here is based on the V-model as defined by standard VDI/VDE 2206 [15] and is shown schematically in Fig. 2.
After specification and theoretical conception, the development process of new functionalities or modules consists of six steps:

1) **Modeling and Design**
Starting from a theoretical concept for a new functionality (e.g. a controller or filter) or physical effect, first of all a model is implemented in the simulation environment.

2) **MIL Module Test**
The module implemented in step 1) is tested in an open loop test.
This can be done in three stages: As first stage, synthetically generated ideal test data is used to verify the module behaviour in principle. As second stage, the robustness of the module e.g. against sensor noise is tested with simulated sensor data (incl. noise, latency, etc.). As third stage, recorded sensor data is fed into the simulation model to further optimize the model.

3) **MIL Performance Simulation**
After completing the open loop module test, the newly developed module is included into the overall simulation model and the functionality performance, stabil-

ity and robustness is tested in a closed model in the loop (MIL) simulation.

4) **Code Export**
The tested simulation model (here: NC) is exported to the target language (here: C++, ROS). For this purpose, the automatic code export of Simulink is used and processed by a automated code modification, which is described in chapter 2.4.

5) **Target Module Test**
The exported code is executed as a stand-alone ROS node on the target hardware and can be tested separately in an open loop. Here, the correct operation of the communication interfaces is tested primarily.

6) **Flight Experiment**
Once all previous tests have been successfully completed, the newly developed functionality can be tested and validated in a real flight experiment. The development process can be repeated iteratively for further module improvements or refinements of the environment model.

2.4 Automated Code Export and Modification

Due to the high computation time of the ROS interface within the Simulink environment shown in Table 1, it is necessary to convert the simulation model entirely into compiled code. With the Simulink Coder, Simulink provides an export functionality to export ROS Nodes in C++ from simulation models. The blocks of the Simulink model including the communication interface blocks are exported as separate C++ functions. The communication interface blocks, which are mentioned in section 2.2 part 3), implement the publishing and subscribing of ROS messages. Incoming data is buffered to be used in the next time step. Outgoing data is published immediately when the respective function is called within the current time step. A master function realizes the coupling of the exported functions for the correct execution sequence within a time step or control cycle. A superordinate operator realizes the time step simulation with constant step size by calling the master function periodically. This default execution method is useful when the exported model itself is the trigger of the whole control loop. Since conventional sensors have a fixed sampling rate and cannot be synchronized with the exported module, additional latencies result between publishing and processing of the sensor data. Using a very high clock rate of the exported module, this latency can be minimized, but results in a high system load. For both low latency and low system load, it is useful to make the sensor with the highest sampling rate the trigger for the exported module. Accordingly, in the automated code modification of the exported code, the aforementioned master function is removed from the fixed time step simulation and placed in a callback structure. For this, the operator is deactivated and a callback function to the corresponding sensor message is generated. This callback function calls the master function after making the sensor message data accessible to it. Polling of the corresponding sensor data is removed from the master function sequence.

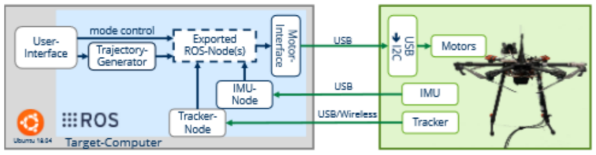

Figure 3 Hard- and software setup during flight experiments

Table 1 Comparison of the real time factor of a Simulink model with and without the ROS interface (I/F)

Environment	*with ROS I/F*	*without ROS I/F*
Simulink	0.033	2.5
Exported to Target	10 (median)	-

3 Case Study

3.1 Multicopter and Model Setup

The used multicopter prototype is a hexarotor with rotors arranged in a regular star shape. The rotor axes are each rotated about the longitudinal axis of the arms to achieve full actuation. The fixed rotation angle for each rotor is 20°. For the design of the multicopter see [8]. The total mass is 4.9 kg. The sensor system consists of an inertial measurement unit (IMU) for measuring translational acceleration and angular velocity, as well as the HTC Vive tracking system, which originally comes from the virtual reality field and provides a cost-effective alternative to camera-based motion capture systems for determining absolute position. To estimate the full pose and twist of the multicopter, an Extended Kalman Filter (EKF) with 16 states is used for sensor fusion of the IMU and tracker data. A 6-DOF PD controller is used to control the state of the multicopter. To prevent static control errors, the estimated external wrench is used for feed forward control. Additionally, an inte-

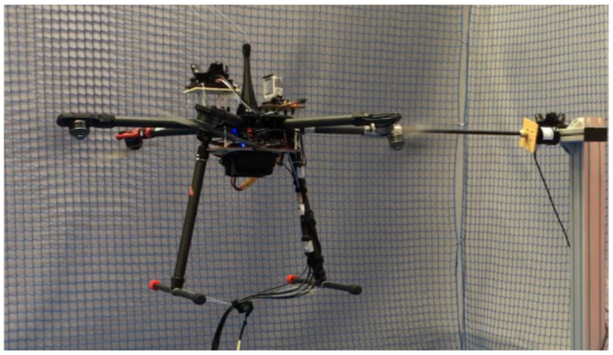

Figure 4 Multicopter with rod-shaped manipulator in contact with the sensor surface.

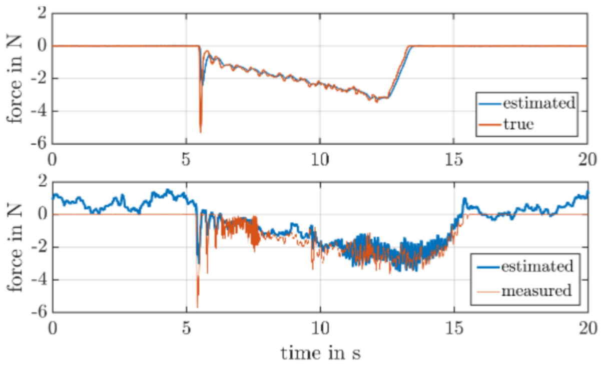

Figure 5 Estimated contact force and true resp. measured contact force for the module test (step 2) with idealized simulation data (top) and with real sensor data (bottom)

grating controller is used for all DOF except the contact direction. The 6-DOF force-torque observer corresponds to [13]. The admittance filter applies a compliance to the system in the contact direction. Separate ROS nodes for each sensor provide communication interfaces and preprocessing of sensor data. As target hardware either a ground computer connected by cable to the multicopter or a single-board on-board computer can be used. For the data presented here, as target the desktop computer specified in section 3.1 is used. Power supply as well as transmission of sensor data and actuator commands are cable based. The setup is shown schematically in Fig. 3.

3.2 Simulation Performance

As mentioned in chapter 2.1, the performance of the ROS interface within Simulink is very low. In table 1 the real time factor of the simulation model in Simulink with and without the ROS interface is compared with the real time factor of the code exported as a stand alone ROS node to the target system. The real time factor denotes the ratio of simulated time to required computation time. The test was carried out on a desktop computer running Ubuntu 18.04 and an Intel Xeon® 2.67 GHz CPU. In total 18 publish and 8 subscribe blocks from the ROS toolbox were used within the model. The used simulation time step of 4 ms corresponds to the sampling rate of the IMU. One can see that the Simulink export to standalone ROS nodes on the target is the only powerful way to run a Simulink controller with ROS interfaces. The ROS node export also enables the controller execution without running Simulink at all and saves resources at the target hardware.

The code generation for the full NC module currently takes 402.5 sec, of which 243 s stem from the compilation of the Simulink model, 143 sec from the code generation, 1.5 sec from the code modification and 15 sec from the final compilation of the C++ code.

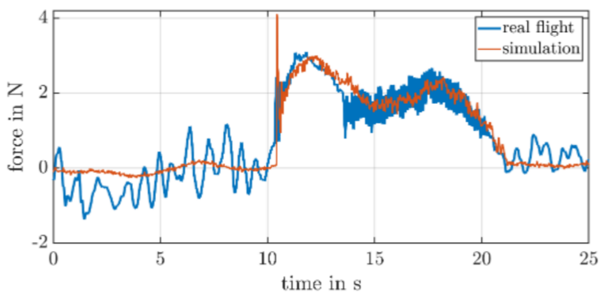

Figure 6 Comparison of the estimated contact force in simulation (step 3) and in real flight (step 6).

3.3 Application of the Development Process

The development process presented in chapter 2.3 is demonstrated using the example of the development of a force-torque observer for tactile interaction described in [13]. The flight task is to apply a force to a horizontal surface equipped with a force sensor. In Fig. 4 the multicopter with rod-shaped manipulator is shown at the moment of contact with the sensor surface. The cable feed can be seen at the bottom of the multicopter.

Fig. 5 shows step 2 of the development process, the MIL module test. The wrench observer is tested in the simulation without feedback (open loop). The result of an ideal MIL simulation is shown at the top. The input data for the wrench observer was generated with a closed position control loop without any sensor noise. A separate module test with recorded sensor data from a real flight as input to the wrench observer is shown below. The true contact force determined in the simulation is shown in red. The comparison verifies the observer design and shows the general performance of the observer. The recorded sensor data originates from a flight experiment in which the pose of the multicopter was controlled by a simple PID controller.

In Fig. 6 the estimated contact force in a flight experiment with closed control loop via wrench observer and admittance filter is shown. The result of the simulation is shown in red (step 3), the result of the flight experiment in blue (step 6). The comparison of the two results already shows a high correspondence between simulation and real flight experiment. The nonzero estimated force before contact in real flight mainly results from the influence of the power and data cables hanging on the multicopter, whose influence is not taken into account in the simulation.

To validate the observer, the comparison of estimated contact force and contact force measured with an external force sensor is shown in Fig. 7. The data from the force sensor and observer were manually synchronized based on the acceleration measurements of the IMU. The measured values from the force sensor are low-pass filtered. Again, there is good correspondence between the observer's estimate and the reference measurement by the force sensor.

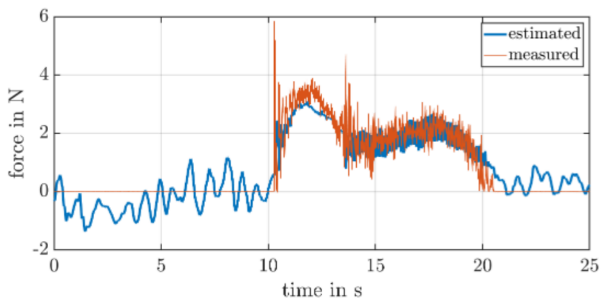

Figure 7 Force measured with an external force sensor and estimated by the observer during the flight experiment to validate step 6

4 Conclusion

This paper presents a closed-loop development process for flight controllers with a focus on tactile interaction tasks. In the presented process, the advantages of Simulink as a modeling and development environment are combined with the advantages of ROS which offers a versatile and powerful interface with low system requirements for fast code execution. The automatic code export from Simulink to ROS avoids time-consuming and error-prone manual implementation in a high-performance programming language. The presented development process is applied to the example of an observer for the determination of contact forces and shows a good transferability of the simulation results to reality. The remaining deviations can be minimized in the future by refining the environment model.

5 References

[1] Bodie, K.; Brunner, M.; Pantic, M.; Walser, S.; Pfändler, P.; Angst, U.; Siegwart, R.; Nieto, J.: *Active Interaction Force Control for Contact-Based Inspection With a Fully Actuated Aerial Vehicle*. IEEE Transactions on Robotics, Vol. 37, No. 3, June 2021

[2] Rashad, R.; Bicego, D.; Jiao, R.; Sanchez-Escalonilla, S.; Stramigioli, S.: *Towards Vision-Based Impedance Control for the Contact Inspection of Unknown Generically-Shaped Surfaces with a Fully-Actuated UAV*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) October 25-29, 2020, Las Vegas, NV, USA (Virtual)

[3] Montufar, D. I.; Muñoz, F.; Espinoza,E. S.; Garcia O.; Salazar S.: *Multi-UAV Testbed for Aerial Manipulation Applications*. 2014 International Conference on Unmanned Aircraft Systems (ICUAS) May 27-30, 2014. Orlando, FL, USA

[4] Offermann, A.; Castillo, P.; De Miras, J.: *Nonlinear model and control validation of a tilting quadcopter*. 28th Mediterranean Conference on Control and Automation (MED) 15-18 September, 2020 - Saint-Raphaël, Frankreich.

[5] Franchi, A.; Carli, R.; Bicego, B.; Ryll, M.: *Full-Pose Tracking Control for Aerial Robotic Systems With Laterally Bounded Input Force*. IEEE Transactions on Robotics, Vol. 34, No. 2, April 2018

[6] Hamaza, S.; Georgilas, I.; Fernandez, M.; Sanchez, P.; Richardson, T.; Heredia, G.; Ollero, A. *Sensor Installation and Retrieval Operations Using an Unmanned Aerial Manipulator*. IEEE Robotics and Automation Letters Vol. 4, No, 3, July 2019, pp. 2793 - 2800

[7] Meyer, J.; Sendobry, A.; Kohlbrecher, S.; Klingauf, U.; Stryk, O. v. *Comprehensive Simulation of Quadrotor UAVs Using ROS and Gazebo*. Proceedings of the Third international conference on Simulation, Modeling, and Programming for Autonomous Robots, 2012

[8] Schuster, M.; Bernstein, D.; Yao, C.; Janschek, K.; Beitelschmidt, M. *Lastraumbasierte Auslegung vollaktuierter Flugroboter*. VDI Mechatroniktagung, Paderborn, März 2019. Tagungsband Fachtagung Mechatronik 2019, ISBN 978-3-945437-05-6, S. 208-213

[9] Santamaría, D.; Alarcón, F.; Jiménez, A.; Viguria, A.; Béjar, M.; Ollero, A. *Model-based design, development and validation for UAS critical software*. J. Intell. Robotic Syst., vol. 65, pp. 103-114, 2012.

[10] De Cos,C. R.; Fernandez, M. J.; Sanchez-Cuevas, P. J.; Acosta, J. Á.; Ollero, A. *Model-based design, development and validation for UAS critical software*. IEEE Access, vol. 8, pp. 223827-223836, 2020, doi: 10.1109/ACCESS.2020.3044098

[11] Jackson, J.; Koch, D.; Henrichsen, T.; McLai, T. *ROSflight: A Lean Open-Source Research Autopilot*. 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) October 25-29, 2020, Las Vegas, NV, USA (Virtual)

[12] ROScopter *https://github.com/byu-magicc/roscopter*.

[13] Ryll, M.; Muscio, G.; Pierri, F.; Cataldi, E.; Antonelli, G.; Caccavale, F.; Bicego, D.; Franchi, A. *6D interaction control with aerial robots: The flying end-effector paradigm*. in The International Journal of Robotics Research, Vol 38, Issue 9, 2019

[14] Meier, L.; Honegger, D.; Pollefeys, M. *PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms*. in Proc. IEEE Int. Conf. Robot. Automat. (ICRA), May 2015, pp. 6235-6240

[15] VDI – The Association of German Engineers *Standard VDI/VDE 2207: Development of mechatronic and cyber-physical systems*