I. Troch, F. Breitenecker, eds. ISBN 978-3-901608-35-3

KNOWLEDGE-BASED SIMULATION MODEL GENERATION FOR CONTROL LAW DESIGN APPLIED TO A QUADROTOR UAV

M.J. Foeken, M. Voskuijl
Delft University of Technology, Delft, The Netherlands

Corresponding author: M.J. Foeken, Delft University of Technology, Faculty of Aerospace Engineering
Kluyverweg 1, 2629HS Delft, The Netherlands, m.j.foeken@tudelft.nl

Abstract. Like for all mechatronic systems, the role of control software in unmanned aerial vehicle design is becoming more important. As part of an automated control software development framework, this paper discusses the development of a simulation model generation method. As a basis, the application of knowledge-based engineering is suggested, requiring the definition of an ontology to capture the various domain concepts and relationships. The initial knowledge base represents concepts and relations to create models with Modelica, the object oriented modelling language used to construct the simulation model. The need for high-fidelity simulation models using the latest design parameters is illustrated by investigating the model of a quadrotor UAV. Further work will extend the knowledge base to include the link between system and simulation model components, enabling the inclusion of behavior in various physical domains.

1 Introduction

According to [1], "mechatronic design deals with the integrated design of a mechanical system and its embedded control system." With the advances in computing capabilities, the role of the embedded control system in the mechatronic system design is becoming more and more prominent. The 'traditional' sequential design approach, in which mechanical, electrical, electronics and software components are designed and optimized sequentially, is therefore becoming less suitable.

The increasingly important role of control software is prominent in the aerospace field, where both aircraft and spacecraft operations benefit from the application of software. Apart from passenger jets, using software with millions of lines of code, and modern fighter jets that require computers to be able to fly in the first place, software has enabled the use of unmanned aerial vehicles (UAVs). UAVs come in a range of sizes, forms, and purposes, from the fixed-wing Global Hawk, with a wingspan of 35 meters and with a primary focus on military surveillance and intelligence gathering, to the X-Ufo quadrotor, a remotely controlled toy of 50 centimeters wide weighing less than 350 grams. However, all UAVs share the characteristic that flight is either remotely controlled by a pilot on the ground, or performed autonomously. In the first case the pilot uses the signals from on-board visual sensors to fly the aircraft. In the second case, a combination of visual, inertial and positioning sensors is used to guide the aircraft along its predefined path, in the mean time performing additional tasks when required. Autonomous flight is then obtained by a combination of control software acting like an auto-pilot, and 'external input' that defines the flight profile and the tasks to be performed during the mission.

(a) Northrop Grumman RQ-4 Global Hawk [2].

(b) Silverlit X-Ufo.

Figure 1: Examples of unmanned aerial vehicles.

As controller design can often not be performed without a suitable dynamic model of the aircraft, these models should be available from early on in the design process. Using control theory, the control algorithms are subsequently developed and implemented as software. To test and verify the control software a mix of hardware and software-based test setups can be applied, ranging from full software-based simulations to prototype hardware tests. A common practice in aircraft design is the use of 'Iron birds', which combine the real aircraft systems hardware with a virtual flight environment. In this way, the control of the electric, hydraulic and other systems can be tested in flight condition without having to perform an actual flight test. Due to the large amount of systems,

it is not possible to model and simulate the behaviour of all of them, therefore, a combination of hardware and software is used.

In Section 2, the research and its relation to a control software generation framework will be introduced. Section 3 discusses the simulation model generation process, followed by an overview of related research. In Section 5 the model for the quadrotor UAV case study is discussed and the results are presented. Finally, in Section 6 the conclusions are drawn.

2 Control software generation framework

To support the development of control software and to cope with the multidisciplinary nature of mechatronic systems, an automatic control software generation framework has been proposed previously [3, 4]. Depicted in Figure 2, the framework contains a combination of to be further developed and existing commercial tools, supported by a high-level model description providing both a functional view on the system, as well as an integration platform.

A design process with not only simultaneous multidisciplinary involvement, but also focussing on multidisciplinary architecture design, will provide possibilities to increase the synergetic effects characterising mechatronic systems. Starting the design process with the requirements specification, the initial system can subsequently be build-up from basic, domain independent system components, satisfying the high-level function requirements. Further requirement and system decomposition results in a system specification serving as the backbone for detailed design and analysis. Implemented in software, this backbone also acts as a data exchange facility.

Labeled as 'Control Model Generation' in Figure 2, part of the supported development process consists of creating models used for control algorithm and control software design and verification. Often referred to as plant or simulation models, these come in the form of system dynamics models. Though an integrated finite element simulation, taking into account structural, aerodynamical, and electrical physical phenomena, might be attractive in terms of accuracy, the computational effort and time required for large scale systems makes them unsuitable for controller design and verification. Instead, these different types of simulations can be used in a sequential order. For example lift, drag and moment coefficients and stability derivatives determined by using computational fluid dynamics (CFD) methods can be used in flight mechanics models during controller design. In the current framework, this analysis data is directly tied to the system components it is related to. For the implementation, the Systems Modelling Language (SysML) has been proposed [3]. Being a visual language, SysML provides a way to give an overview of the systems architecture, while at the same time the XML-based XMI representation makes it possible to process the model data.

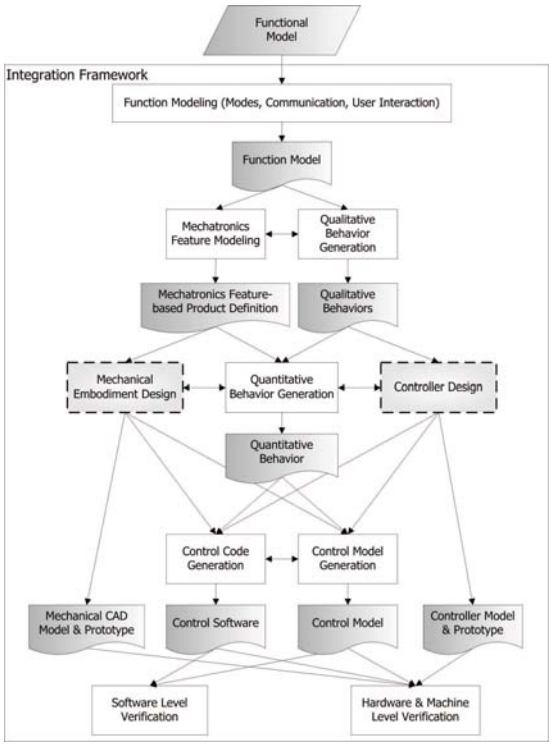


Figure 2: Architecture of the integration framework. The white blocks represent tools to be further developed. Dashed-line blocks correspond to existing commercial software tools.

3 Control model generation

As introduced in the previous section, in the framework the multi-disciplinary system architecture is build-up from mechatronic system components. However, in control design the system is often represented as a set of (linear) transfer functions, either in the time or in the frequency domain. The link between the parameters in this representation and those of the physical system is small, in the sense that the former have no direct physical meaning, but are based in the mathematical domain. For a simple mass-spring-damper system these can still be traced back to parameters of the original system, but for more complex systems, this becomes troublesome, if not impossible.

The Modelica language has been designed to model large, complex and hybrid physical systems and is based on differential and algebraic equations. It supports noncausal and object-oriented modelling techniques, and as such stimulates the reuse of modelling knowledge. In contrast to the use of transfer functions, with this 'physical modelling' paradigm a component-based model corresponding to physical elements, using parameters directly related the real world, can be obtained. These parameters can then be obtained via the integration framework, which binds the model data and analysis results directly to the system components. As often control design relies on linear models at certain design conditions, the obtained component-based non-linear model must be linearised.

The relation between elements on system component, control and mathematical model can be represented as in Figure 3, adapted from [12].

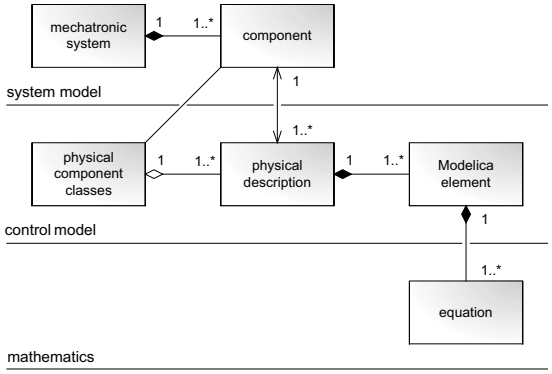


Figure 3: Relationships between components and elements at different viewpoints.

Although a direct, one-to-one mapping from system component to control model component is relatively straightforward, this kind of mapping ignores the possible interaction occurring in or between elements of different domains. As a single system component can show behaviour in different physics domains, e.g. the mechanical and aerodynamical domains, a combination of elements will be required to obtain the correct interaction. For example, the main functionality of a DC-motor is to act as a torque generator. However, when attached to a rotor it will also serve as an element introducing the aerodynamic forces into the structure, for which the dimensions of the DC-motor might come into play. The relative vertical position of the rotors with respect to the overall center of gravity has a strong effect on the stability.

3.1 Requirements

To be able to generate control models in the context of the software development framework the following requirements have been recognized:

- It must be possible to build-up the system architecture from elementary technical components.
- These components must have one or multiple representations in the physical modelling world, see Figure 3.
- When connecting elements the port compatibility must be checked to prevent the coupling of incompatible elements. See e.g. [15].
- Not only the intended behaviour, but also 'secondary' or 'unintended' behaviour must be recognised and included.

To handle the third requirement, the interfaces of the elements should not be fixed to the main behaviour of the element, or those that fulfill the main, expected, functionality. Reference [5] denotes the additional interaction as 'unpredictable interaction', resulting in behaviour which occurs either within a domain or by interactions between domains.

The solution presented in [5] links physical phenomena, like e.g. gravity or thermal radiation, to system model elements in a library. When the system is build-up by instantiating these elements, a qualitative reasoner reasons out all possible behaviour [6]. In combination with a model representing the intended behavior, the unpredictable interactions are discovered. The knowledge base containing the model elements, physical phenomena and related

concepts is implemented in the Knowledge Intensive Engineering Framework (KIEF) [7]. The use of expert knowledge in engineering applications enables the automation of that part of the design and analysis process that is repetitive, non-creative and time-consuming. This can be supported by applying knowledge-based engineering techniques and tools, featuring rule based design and object-oriented programming [8]. In general, a knowledge base is build-up from concepts, attributes to these concepts, values, and the relations between concepts. Most knowledge bases can be represented as in a diagram with concepts connected via relations, see Figure 5, or with the focus on attributes, shown as a frame as in Figure 4 [9].

Although the method of [5] is based on qualitative physics, the idea of using physical phenomena linked to system elements can be extended to quantitative simulation model components linked to these system elements. Whether a certain submodel is required in a certain system depends then on e.g. orientation and relative position in case of heat radiation. The interface of a component should therefore not be fixed nor restricted to only the intended connections, but must depend on the system's implementation.

3.2 Knowledge base development

The creation of a knowledge base containing the systems components and relations starts with the identification of the applicable concepts and relations. Such an ontology defines what concepts can be found in the domain, and what kind of relations between these concepts exist. The three levels indicated in Figure 3 each form a domain by itself for which a separate ontology can be created.

As a first step for setting up the knowledge base supporting the model generation process, a language ontology for Modelica models has been created. For this, Epistemics' PCPack [10] has been used, a tool to capture, structure and re-use both procedural and conceptual knowledge. The former is related to the expertise to perform a task, while the latter is related to the expertise on concepts and the relationship between these concepts [9].

Based on the Modelica language specification, the concepts and relations in the ontology are restricted such that only 'structurally' correct models can be created with it. In general, a Modelica class represents the behaviour of a particular aspect of a physical system component. It contains a declaration of elements, which include component definitions or class instances, equations and connections, and algorithms. Similar to bond graphs, the connection between physical components have the dimension of energy or power, while the input to actuators and the output of sensors is a data stream.

Based on the ontology, a database containing Modelica libraries can be created, initially providing an insight in the relation between and the properties of classes. Figure 4 shows a model of a rotor actuator in the PCPack interface. The frame representation contains the class attributes and values as well as related class and element definitions. Due to the nature of the Modelica language, a class specialization hierarchy based on only class inheritance relations is not possible. However, using additional attributes indicating the level of specialization can circumvent this problem, which is easily managed in the knowledge base. Similarly, the applicable physical phenomena can be seen as a attribute of the class.

	QuadRotorCS.Drives.RotorActuator	
– Name	RotorActuator	
– Restriction	model	
– Attributes	Partial	false
	Encapsulated	false
	Replaceable	false
– Comment		
– Element of	QuadRotorCS.Drives	
– Elements	Extends	
	Imports	
	Class Definition	
	Component	• QuadRotorCS.Drives.RotorActuator dCMotor • QuadRotorCS.Drives.RotorActuator frame_b • QuadRotorCS.Drives.RotorActuator p • QuadRotorCS.Drives.RotorActuator rotor
	Equation	• connect(dCMotor.p, p) • connect(gear.flange_a, dCMotor.flange_b) • connect(gear.flange_b, rotor.flange_a) • connect(rotor.frame_b, frame_b)
	Algorithm	

Figure 4: Rotor actuator component in knowledge base.

The 'RotorActuator' class is part of a library of quadrotor specific components, which can be represented as a tree, see Figure 5.

To be able to relate components from the system level to the control model components, the obtained ontology and the related knowledge base should be extended or merged. Whereas the ontology should then be applicable to all types of mechatronic systems, the knowledge base will be system or application domain specific, as the type of components and their various representations can vary significantly between domains.

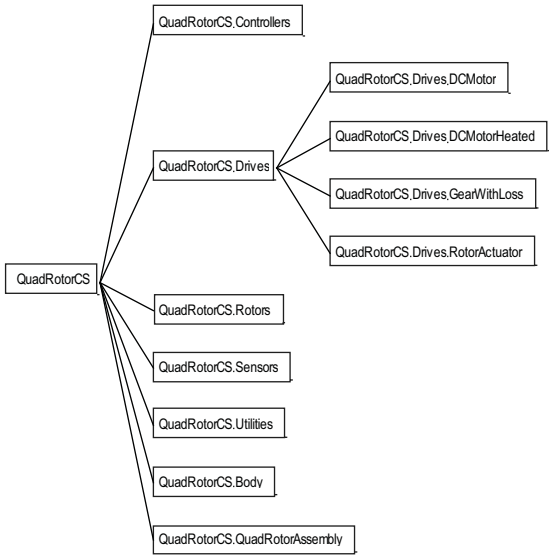


Figure 5: Quadrotor component library as part of knowledge base.

4 Literature review

The application of knowledge-based techniques in the development of simulation models focusses on various subjects. The application of knowledge engineering for the development of conceptual simulation models is discussed in [11], focussing on how to capture, represent and organize the knowledge required. A general introduction on knowledge based systems is given in [9].

Reference [12] describes the architecture of a library of reusable simulation models, and argues that there is a need for a taxonomy of component classes to handle the complexity associated with large libraries. The structure of the library is not restricted to be tree-like, instead, a tangled structure can also be obtained. To enable component reuse the application of web-based hierarchical libraries is introduced in [13]. The method is based on the concepts of modeling components, abstraction hierarchy and genericity of use.

Instead of mapping components, the Composable Object concept combines different views in a single object [14]. Using parametric descriptions the consistency between form and behaviour can be maintained. The interaction between objects is port-based, and connected they represent both a system-level design description and a virtual prototype of the system. Due to the port-based approach, views can be replaced by more elaborate models if necessary, which are possibly build-up from other objects.

Reference [15] discusses more a port ontology in light of automatic model composition, and the need to take into account the type of interaction taking place. Often, these interaction models depend on the parameters of both subsystems involved. This principle is also applied in [5] to discover unpredictable interactions in and between system components, based on qualitative physics. Similarly, a framework to capture the interaction in component-based design is presented in [16]. To prevent connecting incompatible components, the interaction type is checked.

5 Example application results

5.1 Modelling of quadrotor UAV

In the group of quad- or multirotor UAVs, the systems are highly componentised. As the four actuators are used both for propulsion and control purposes, and there are no other moving components, the dynamics are mainly determined by these rotor actuators and the supportive structure. The type and quality of the sensors strongly influences the amount of autonomomy that can be achieved.

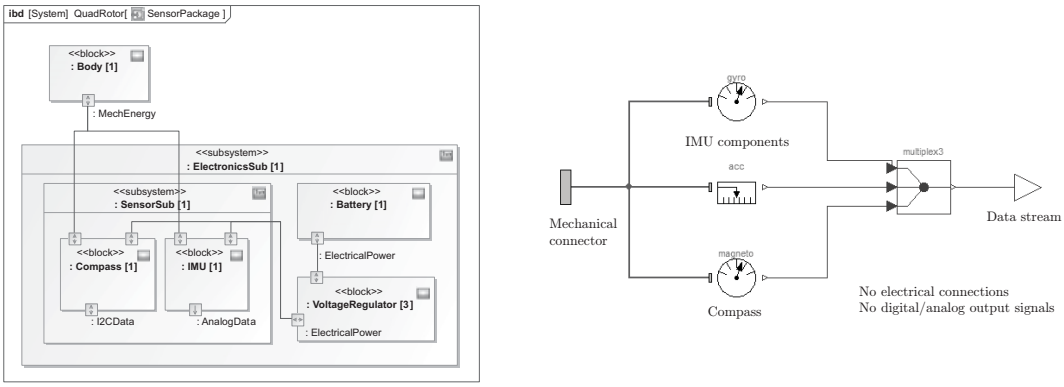
In its most basic form, the model of the quadrotor UAV can be built-up from the following components:

- Body with mass, inertia and area of total system
- 4 actuators consisting of a DC motor, gear, and rotor, taking into account gyroscopic effects, see Figure 7.
- Sensor package, including drift and noise characteristics, see Figure 6
- Environmental effects, i.e. gravity and aerodynamic effects

The effect of electrical and thermodynamical phenomena on system behavior has not been taken into account at this point, nor the electrical and control hardware subsystem. Various system components require multiple control

model elements to be represented correctly. Although the main effect of the rotor is the creation of thrust and torque, the gyroscopic effects have large influence on the dynamics of the system. Furthermore, the aerodynamic drag on the body can not be discarded despite the low flight speeds. A description of an accurate simulation model used for testing and validation purposes in the Matlab/Simulink environment is given in [17].

Although on system component level the input to the actuators and the output of sensors will be an electrical current, using either an analog or digital signal, the model represents this as a stream of data which can directly be used for control purposes, see Figure 6. The sensor package used in the example consists of an inertial measurement unit (IMU), containing three gyroscopes and a tri-axial accelerometer, and a magnetic compass. Additional sensors can include barometric or infrared altitude sensors and infrared distance sensors for indoor flight.



(a) Sensor assembly on system component level. **(b)** Modelica model of sensor assembly.

Figure 6: Sensor package on quadrotor UAV.

The simplest way to calculate the thrust and torque of a single rotor actuator is given in Eq. 1, only taking into account a variable rotor speed Ω. The constants b and d are the thrust and torque constants, respectively, determined by experiment or computational fluid dynamics (CFD) calculations.

$$\begin{aligned} F &= \Omega^2 \begin{bmatrix} 0 & 0 & b \end{bmatrix}^T \\ Q &= \Omega^2 \begin{bmatrix} 0 & 0 & d \end{bmatrix}^T \end{aligned} \tag{1}$$

However, as the effect of altitude, flight speed, and flight direction on the rotor performance is significant, these variables have to be taken into account. Eqs. 2-3 uses local airspeed and flow directions to calculate 3D forces and moments at the rotor hub. In here, λ is the dimensionless rotor inflow velocity, μ the dimensionless in-plane velocity, with subscript X and Y denoting in x and y axis direction, respectively. The thrust and torque coefficient are defined by [18]:

$$\begin{aligned} C_T &= \sigma a \left(\frac{4+6\mu^2}{24}\theta_0 - \frac{1+\mu^2}{8}\theta_{tw} - \frac{1}{4}\lambda \right) \\ C_Q &= \sigma a \left(\frac{1+\mu^2}{8a} C_d + \lambda \left(\frac{1}{6}\theta_0 - \frac{1}{8}\theta_{tw} - \frac{1}{4}\lambda \right) \right) \end{aligned} \tag{2}$$

The solidity σ is a measure for what part of the rotor disc is covered by the rotor blades, and θ_0 and θ_{tw} are the root pitch angle and twist angle, respectively. a is the derivative of the lift coefficient C_L with respect to the angle of attack α. The forces and moments are:

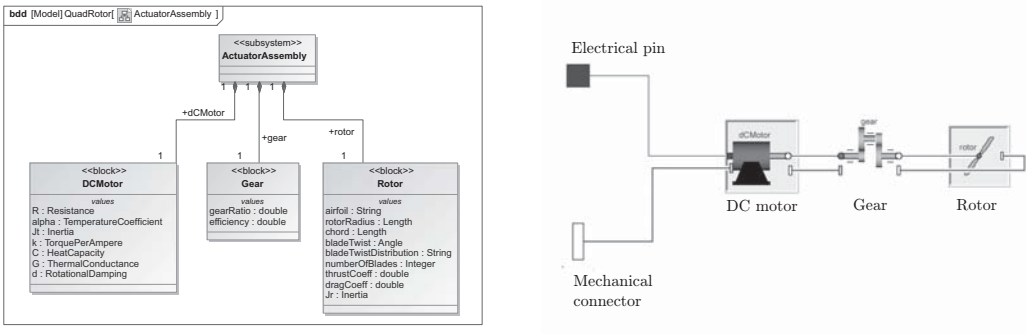
$$\begin{aligned} F &= \rho A \Omega^2 R^2 \begin{bmatrix} C_{hX} & C_{hY} & C_T \end{bmatrix}^T \\ Q &= \rho A \Omega^2 R^3 \begin{bmatrix} C_{rX} & C_{rY} & C_Q \end{bmatrix}^T \end{aligned} \tag{3}$$

where ρ is the air density, A the area of the rotor disc, and R the rotor radius.

A third way to calculate the rotor forces and moments, known as the blade element method, applies a discretisation to the rotor blades, and calculates the forces and moments on each blade segment based on the local flow field. The composition of the rotor actuator assembly is given in Figure 7.

5.2 Trim routine results

The obtained non-linear models can be further used for control law design purposes, using linearization or system identification techniques. As a first step, to find the condition in which steady flight is achieved is obtained, a

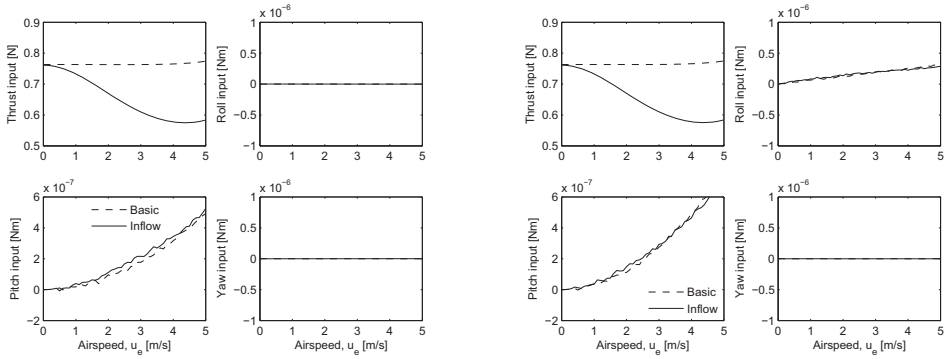


(a) Actuator assembly components. **(b)** Modelica model of actuator assembly.

Figure 7: Actuator assembly on quadrotor UAV.

generic rotorcraft trim routine is used, based on the Jacobian method, which uses numerical perturbation of the model in combination with a Newton-Raphson iteration scheme [19].

Using a trim routine the steady flight condition for a fixed turn rate, flight path angle and/or sideslip angle for a range of flight speeds is determined. In Figure 8 the control inputs for a flight speed 0-5m/s in forward flight and in a 3°$/s$ turn. The inputs are force and torques, which are transformed to motor voltages using a simplified inverse model. To balance the torque, two clockwise and two counterclockwise rotating actuators are needed.



(a) Control inputs in forward flight. **(b)** Control inputs in a turn.

Figure 8: Results of trim routine, for an airspeed 0-5m/s. Left in forward flight, right in a 3°$/s$ turn.

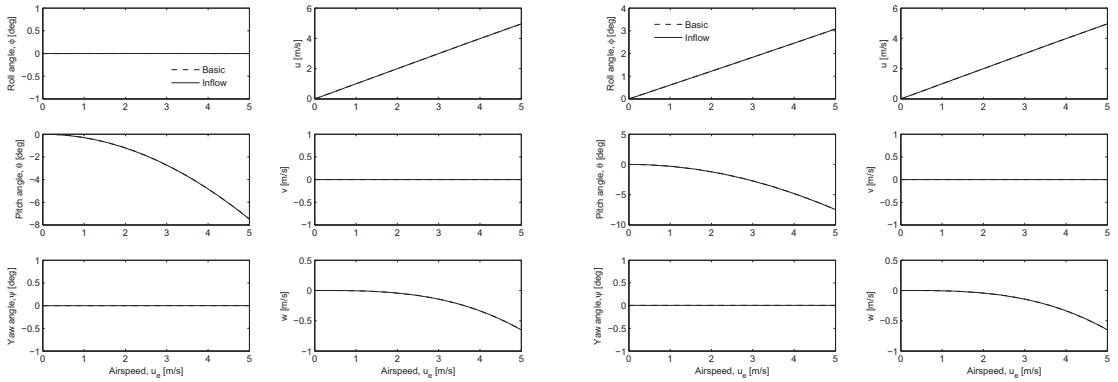
By taking into account the local flow field for each rotor, the required thrust input and rotor speeds are reduced, as the component of the flight speed perpendicular to the rotor becomes larger, and the thrust increases. The pitch angle, which is positive nose up, decreases for increasing flight speed, controlled with a small pitch input.

As seen in Figure 9, there is on the other hand no effect on the trimmed velocities in the body reference frame and the attitude of the UAV, as expected. As a quadrotor UAV has the capability to hover, a 3°$/s$ turn with zero airspeed corresponds to a turn around the top axis. In forward, level flight, when the aircraft has a small pitch angle, there is a small component of the velocity in the body z-axis (positive downward).

6 Conclusions and future work

The increasing importance of control software in mechatronic systems requires a design approach that addresses both simultaneous multi-disciplinary involvement as well as multi-disciplinary architecture design. Based on this idea, a framework supporting the development of control software for mechatronic systems was previously proposed. This paper has introduced a knowledge-based approach for generating simulation models as part of this framework, and illustrated the need for, and advantages of such approach by considering the development of non-linear simulation models of a quadrotor UAV.

With respect to the case study, the linearisation of the quadrotor model will have to performed to be able to use it for future control design. The development of an ontology underlying the knowledge base has started at the level of the modelling language, and in the next step will be extended to include the concepts and relations representing the mechatronic system components. To be able to include the representation of the behavior of a component in various domains, if required, knowledge on the applicable physical phenomena will be have to added to the knowledge base.



(a) Rotation angles and translational velocities in forward flight.

(b) Rotation angles and translational velocities in a turn.

Figure 9: Results of trim routine, for an airspeed 0-5m/s. Left in forward flight, right in a $3^\circ/s$ turn.

7 Acknowledgments

The authors gratefully acknowledge the support of the Dutch Innovation Oriented Research Program 'Integrated Product Creation and Realization (IOP-IPCR)' of the Dutch Ministry of Economic Affairs.

8 References

[1] Van Amerongen J., and Breedveld P.: *Modeling of Physical Systems for the Design and Control of Mechatronic Systems*. Annual Reviews in Control, 27 (2003), 87–117.

[2] Air Force Link: *RQ-4 Global Hawk*. `http://www.af.mil/photos/`.

[3] Foeken M. J., Voskuijl M., Alvarez Cabrera A. A., and Van Tooren M. J. L.: *Model Generation for the Verification of Automatically Generated Mechatronic Control Software*. In: Proc. IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, Beijing, China, 2008, 275–280.

[4] Alvarez Cabrera A. A., Erden M. S., Foeken M. J., and Tomiyama T.: *On High-Level Model Integration for Mechatronic Systems Control Design*. In: Proc. IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, Beijing, China, 2008, 387–392.

[5] D'Amelio V., and Tomiyama T.: *Predicting the Unpredictable Problems in Mechatronics Design*. In: Proc. International Conference on Engineering Design, Paris, France, 2007.

[6] Forbus K. D.: *Qualitative Process Theory*. Artificial Intelligence, 24 (1984), 85–168.

[7] Yoshioka M., Umeda Y., Takeda H., Shimomura Y., Nomaguchi Y., and Tomiyama T.: *Physical Concept Ontology for the Knowledge Intensive Engineering Framework*. Advanced Engineering Informatics, 18 (2004), 95–113.

[8] La Rocca G., and Van Tooren M. J. L.: *Enabling Distributed Multi-Disciplinary Design of Complex Products: a Knowledge Based Engineering Approach*. Journal of Design Research 3, 5 (2007).

[9] Milton N. R.: *Knowledge Technologies*. Polimetrica, Milan, Italy, 2008.

[10] Epistemics: *PCPack*. `http://www.epistemics.co.uk/index.htm`.

[11] Zhou M., Son Y. J., and Chen Z.: *Knowledge Representation for Conceptual Simulation Modeling*. In: Proceedings of the 36th Winter Simulation Conference, Washington D.C. USA, 2004, 450–458.

[12] Breunese A., Top J. L., Broenink J. F., and Akkermans J. M.: *Libraries of Reusable Models: Theory and Application*. Simulation 71, 1 (1998), 7–22.

[13] Bernardi F., and Santucci, J. F.: *Model design using hierarchical web-based libraries*. In: Proceedings of the 39th Conference on Design Automation, New Orleans, Louisiana, USA, 2002, 14–17.

[14] Paredis C. J. J., Diaz-Calderon A., Sinha R., and Khosla P. K.: *Composable Models for Simulation-Based Design*. Engineering with Computers, 17 (2001), 112–s128.

[15] Liang V.-C., and Paredis C. J. J.: *A Port Ontology for Automated Model Composition*. In: Proc. Winter Simulation Conference, 1, 2003, 613–622.

[16] Lee E. A., and Xiong Y.: *System-Level Types for Component-Based Design*. Lecture Notes in Computer Science, 2211 (2001).

[17] Bouabdallah S., and Siegwart R.: *Design and Control of a Miniature Quadrotor*. In: Advances in Unmanned Aerial Vehicles, (Ed.: Valavanis, K. P.), Springer, The Netherlands, 2007, 171–210.

[18] Johnson W.: *Helicopter Theory*. Courier Dover Publications, 1994.

[19] Dreier M.: *Introduction to Helicopter and Tiltrotor Flight Simulation*. AIAA Education Series, 2007.