2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

A Simulator Environment for Aerial Service Robot Prototypes

Roberto Naldi, Alessandro Macchelli, Dario Mengoli and Lorenzo Marconi

Abstract— **This paper provides an architectural description from the software point of view of the simulator environment developed for the AIRobots project. The scope of the project is the realization of an aerial service robotic prototype, a sort of robotic hand to be employed in inspection-by-contact tasks. The simulator is then crucial in both the training of the human operator, and as a support tool for the development and validation of low- and high-level control algorithms. The tasks that can be performed are not limited to free-flight missions, but include also to the cases in which the robot has to actively interact with the environment. The simulator relies on Simulink and Blender, and has been designed with a modular structure that makes software-in-the-loop and hardware-in-the-loop simulations possible by simply replacing the different control modules with the real controllers on the prototypes.**

I. Introduction

The development of the aerial service robot prototypes covers situations in which using directly the real vehicle would be risky, unpractical or simply time-consuming. In these scenarios, which include the training of the operator and the validation of new low-level and high-level functionalities, an advanced simulation environment has to be taken into account, [1]. To be effective in both the design and validation of control algorithms, the simulator environment should appear transparent to the rest of the control architecture and to the human operator so that switching from simulations to real flight operations should be achieved as seamlessly as possible. This fact implies that, on one side, the simulation environment should have the same software interface as the real prototype and, on the other, that the dynamical model of the aerial system and of the environment should match as far as possible the real ones.

The simulator presented in this paper has been designed to address a number of relevant scenarios, ranging from simple free-flight operations, to physical interaction with a realistic 3D reconstruction of the environment. Free-flight operations with some prototypes of aerial service robots (e.g., a quadrotor, a ducted fan, or a helicopter), are the simplest experiments possible. In this case, the simulator is helpful for validating the dynamical model of the system, and the performances of the low-level control law. Human Interface Devices (HIDs), e.g. a simple joystick, are employed to generate the reference signals for the low-level free-flight controllers to pilot the vehicle in a virtual 3D environment, in the same way as they are employed for real flight tests.

R. Naldi, A. Macchelli, D. Mengoli and L. Marconi are with the Department of Electrical, Electronic, and Information Engineering (DEI) "Guglielmo Marconi," University of Bologna, viale del Risorgimento 2, 40136 Bologna, Italy, email: {dario.mengoli, roberto.naldi, alessandro.macchelli, lorenzo.marconi}@unibo.it

Fig. 1. Typical inspection-by-contact task performed by the "double" ducted fan.

The second and more advanced kind of experiments deals with the validation of the high-level control and sensor fusion algorithms. In particular, by means of an accurate 3D model representation of the operational scenario and the correct implementation of the sensors mounted onboard, the effectiveness of high-level vision based algorithms is validated by considering some of the characteristics of the real environment (e.g., textures, illuminations, and so on). The high-level sensor fusion, together with the high level supervisor, is employed to test mission planning, obstacle avoidance and re-planning in the selected virtual world. Finally, telemanipulation algorithms are validated by integrating also haptic devices in the simulation. This is the case presented in Fig. 1. Here, through an accurate modelling of the environment in which the robot has to operate, the simulator is also used to validate low-level control algorithms able to handle physical interaction with the environment itself. More precisely, the aerial vehicle is equipped with a manipulator, and several tests on position and force control algorithms for the overall multi-body system can be easily performed. Clearly, force and contact sensors have to be implemented in the simulation environment. Finally, the operator can be successfully trained in particular end-user environments. This is achieved by integrating all the advanced functionalities discussed above in a single application, and by paying particular attention to the description of the environment in which the aerial service robot is required to operate.

To be really effective in validating both the low-level and high-level control algorithms, the simulator has been designed with a modular structure [2], in which the dynamical models of the aerial robots and of the control algorithms can be easily integrated as separated modules. This choice makes

978-1-4673-6357-0/13/$31.00 ©2013 IEEE

software-in-the-loop and hardware-in-the-loop simulations possible by simply replacing the different control modules with the real controllers installed on the prototypes. This feature is of paramount importance to actually keep the simulation environment updated with the latest version of the control software at no additional development cost. Moreover, it also allows to keep the control logic separated from visual and graphics components in case of improvements or future upgrades.

II. Software requirements

Aerial service robotics is a new field of research, and complete off-the-shelf tools able to accomplish advanced simulations are still not available. However, at the moment a relatively large number of open-source and commercial softwares allows to perform advanced simulations for other different robotic applications, such as aerial robots [3], mobile robots [3]–[11], and robotic arms [12], [13]. The simulator presented in this paper takes advantage from some functionalities already available in some open-source solutions, with the goal of actively contribute within the international open-source robotic community. So, in this section, the main guidelines that motivated the design of the simulation environment, and the relation with existing open-source tools are discussed.

A. Mandatory requirements

If compared with standard simulation tools for aerial systems, where only free-flight is taken into account, the proposed simulator is required to directly manage the interaction between vehicle and environment. Moreover, the simulator integrates the advanced control architecture of the real system, including also all the components that are necessary for advanced human-machine interaction and high-level, vision-based control algorithms. The main and mandatory requirements are then summarized as follows:

1) Physics: Physical simulation is crucial because one of the main goals is to study the UAV in different operative conditions, such as in free-flight or during the interaction with the environment. In the latter case, it is necessary to determine collisions in the 3D space, and calculate the reaction forces to apply to the robot. In this respect, realistic friction models and advanced contact models have to be available. Moreover, an accurate multi-body kinematic and dynamic description of the manipulator attached to the vehicle is mandatory. Finally, to implement and validate the different feedback control strategies, low-level sensors (e.g., IMU), and high-level sensors (e.g., vision), should be available.

2) Haptic feedback: During inspection-by-contact tasks, the vehicle approaches the surface to be inspected, gets in contact, and then slides along it, acting thus as a sort of operator "virtual hand." To have a realistic feeling, the operator will drive the vehicle using a haptic device able to provide force feedback. Since on the physical system force sensors are installed in specific points of the vehicle's body, the same kind of information have to be computed also by simulator and sent to the haptic device to provide the operator with the same feeling of real operation scenarios. As before, a realistic compliant contact model of the environment must be implemented, together with force and contact sensors able to detect the contact forces exchanged during the interaction.

3) Video feedback: Stereoscopic cameras are installed on the real UAV, whose output is sent both to the operator, and to high-level vision algorithms able to estimate the speed and the pose of the vehicle. Such cameras are present also in the simulation environment, not only for training purposes by sending the images to the operator console, but also to speed up the development and testing of the vision algorithms.

4) Frame rate: The dynamical model of the vehicle requires that the simulator is able read inputs and send outputs at a frequency of about 100 Hz. The above sample time is compatible with the dynamical properties of the closed-loop attitude and position control subsystems that stabilize the real prototype. Video output is allowed to have a lower sample-rate, in fact the onboard physical cameras are able to stream data at approximately 10-20 Hz.

B. Simulator engine limitations

Because of its development process, the control logic is available in Matlab/Simulink model. In this respect, the output of the control logic is an array of six values, namely forces and torques along the UAV's axes. This is clearly an input of the simulator as far as the UAV subsystem is concerned. Moreover, since aerodynamics can be easily modelled in MATLAB, the dynamics of the vehicle is simulated within Matlab/Simulink in a block that receives the resultant forces and torques generated by the fan and the aerodynamic control surfaces. In order to close the loop, information about the position and pose of the UAV must be returned to the control logic, but also to the visualization subsystem, which is implemented in Blender and is responsible for the visual feedback to the operator. Moreover, it is the Blender module that takes care of the collision detections, and together with Matlab/Simulink, is able to generate the contact forces on the basis of a specific contact model.

The simulation engine, then, must be able to communicate with the different components using a networked computer environment or serial communication in order to exchange data in a fast and reliable way. Since the whole main simulation loop has to be performed in less than 10 ms to attain a frequency of 100 Hz, the communication protocol must be kept simple enough so that it doesn't require time-consuming parsing and elaboration. It must nevertheless be flexible enough to allow the input of different data, for driving the UAV, controlling appliances, starting and stopping the simulation and so on. Moreover, all potentially time-consuming calculations (e.g., image data acquisition and elaboration) must be kept outside of the simulator.

C. Choice of the tools

With all the requirements in mind, the simulation environment has been developed in two main modules. The first one requires Matlab/Simulink and is responsible of the physical simulation of the aerial device, i.e. UAV and manipulator

dynamics, aerodynamics and contact dynamics. As far as the graphical part and environment interaction are concerned, our choice has been the Blender 3D content creation suite, [14]–[16]. Blender provides a 3D rendering framework and a game engine that allows to interact with the 3D scene. This feature is necessary to determine collisions and is fundamental in the simulation of the contact. Moreover, the Blender architecture can be expanded using the *Python* programming language.

In this context, the MORSE (Modular OpenRobots Simulation Engine) library [8] has come in aid for our "robotics" purposes. MORSE defines a neat architecture of components, tailored specifically for robotics, providing abstractions for robots, sensors and actuators. It also defines a middleware layer that can be implemented according to the requirements of the deployment. With this architecture, the same simulation can interact with ROS nodes or use proprietary communication protocols without changing the simulation itself, but just by switching middle-wares. Different middlewares can be used simultaneously to communicate with different clients.

III. Simulation environment

The simulation environment aims at emulating the physical parts of the real UAV in the most realistic way possible, while using all the same control algorithms. In particular, the simulation environment takes care of all physics simulation aspects e.g., rigid body dynamics, collisions, aerodynamics and sensors output. The main components have been implemented either in Simulink or in Blender, using the Python language. Simulink and Blender communicate through UDP packets, so communication has to be kept as lean as possible. Generally speaking, the simulation of the 3D physics and the integration with the low-level control has been developed in Simulink, while Blender is responsible for the graphical modelling and the discrete collision detection. The information on the contact between manipulator and virtual environment that Blender computes are properly used in Simulink to generate a realistic contact dynamics.

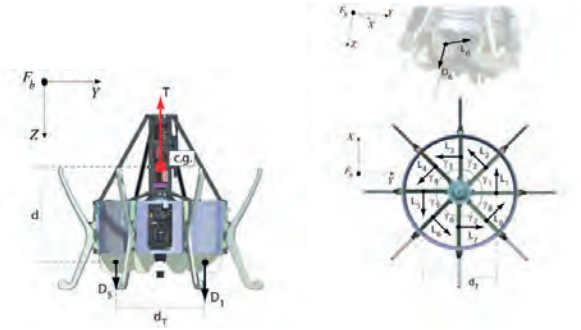
A. Aerial vehicle

Simulink is used to compute the rigid body dynamics and the aerodynamic forces for the virtual UAV. On the other hand, the Bullet Physics Library in Blender is devoted to the discrete collision detection. Such combination let the simulated UAV to fly and interact with the virtual environment in a realistic way, and the retrieved data can be exported to the controller, executed in Simulink.

A mathematical model for the rigid body dynamics can be given by means of the so-called Newtown-Euler equations:

$$\begin{aligned} M\ddot{p} &= Rf^b \\ J\dot{\omega} &= -\omega \times J\omega + \tau^b \end{aligned} \tag{1}$$

where f^b and τ^b represent respectively the vector of forces and torques applied to the vehicle expressed in the body frame, M the vehicle total mass, J the diagonal inertia matrix, $p = (x,\, y,\, z)^{\mathrm{T}}$ the position of the center of mass, ω the angular velocity expressed in the body frame and R the rotation matrix relating the body frame with the inertial frame.



(a) (b)

Fig. 2. Force generation scheme in the Ducted-Fan MAV: (a) drag forces, (b) lift forces.

To expose an abstract and generic torque/force layer to be used with the different prototypes, the aerodynamic laws that apply to the individual prototypes are implemented using a separate Simulink block that computes the vector of applied forces and toques as

$$\begin{pmatrix} f^b \\ \tau^b \end{pmatrix} = \mathcal{F}^b(u_1, u_2, \dots) \tag{2}$$

i.e., as function of further manipulable control inputs u_i. In case of the ducted fan, such control inputs are the vane's angle of attack α_i. With reference to Fig. 2, the vector of forces f^b in the body frame can be computed as

$$f^b(\underline{\alpha}) = \begin{pmatrix} 0 \\ 0 \\ -T \end{pmatrix} + \begin{pmatrix} \sum_{i=1}^{8} \mathbf{L}_i(\alpha_i)^{\mathrm{T}} i^b + \sum_{i=1}^{8} \mathbf{D}_i(\alpha_i)^{\mathrm{T}} i^b \\ \sum_{i=1}^{8} \mathbf{L}_i(\alpha_i)^{\mathrm{T}} j^b + \sum_{i=1}^{8} \mathbf{D}_i(\alpha_i)^{\mathrm{T}} j^b \\ \sum_{i=1}^{8} \mathbf{L}_i(\alpha_i)^{\mathrm{T}} k^b + \sum_{i=1}^{8} \mathbf{D}_i(\alpha_i)^{\mathrm{T}} k^b \end{pmatrix}$$

where T denotes the thrust force produced by means of the propeller, while the resultant torque vector is given by

$$\tau^b(\underline{\alpha}) = \sum_{i=1}^{8} \mathbf{r}_i \times \mathbf{L}_i(\alpha_i) + \sum_{i=1}^{8} \mathbf{r}_i \times \mathbf{D}_i(\alpha_i)$$

where, for each $i \in \{1,\, 2,\, \dots,\, 8\}$,

$$\mathbf{r}_i = \begin{pmatrix} \frac{1}{2} d_T \sin(\gamma_i - \pi/4) \\ \frac{1}{2} d_T \cos(\gamma_i - \pi/4) \\ d \end{pmatrix}$$

denotes the point of application of each aerodynamic pair of lift (L) and drag (D) forces with respect to the center of gravity of the system which coincides precisely with the origin of the body fixed reference frame. Clearly, the function $\mathcal{F}^b$ in (2) depends on the particular configuration of the UAV. In the simulator presented in this paper, beside the ducted fan, the aerodynamic forces of a modular ducted fan in double configuration and of the quadrotor have been implemented. Other model can be easily added.

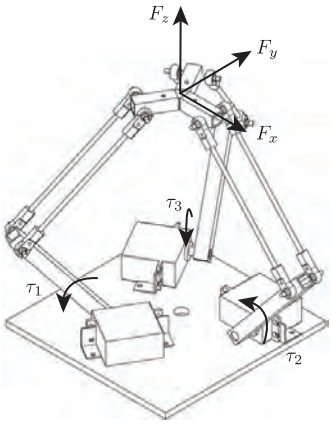


Fig. 3. Delta robot: forces at the base joints and forces at the end-effector.

B. Manipulator

The UAV is equipped with a manipulator to interact with the environment, namely a Delta Robot, whose schematic model with the indication of the joint torques and corresponding end-effector forces is reported in Fig. 3. The dynamic model is implemented as a Simulink block, and the control action is computed in such a way to have a desired position and/or force at the end effector. The "kinematic" information are then necessary in Blender to render a 3D model of the robot. The position of the end-effector is *not* directly used in Blender to manage the contact/no contact configuration with the environment. This point is discussed later on in Sect. III-C.3.

C. Sensors

The simulation environment substitutes the UAV's sensors in order to give feedback both to the operator and to the low-level and high-level control laws. Force information computed during the interaction are also transmitted beside to the dynamical models of the UAV and of the manipulator, also to the haptic devices used by the operator.

1) Inertial sensor: Information regarding the position of objects, can be easily retrieved in Blender. The MORSE framework provides out-of the box sensors that will retrieve attitude, location, speed and acceleration information, thus emulating both the IMU (Inertial Measurement Unit) and Optitrack motion tracking information. Only attitude and location are sent back to the control law in the Optitrack packet so as to provide the same amount of information used in free-flight indoor experiments. Additional speed information can be added to simulate IMU gyroscopic information, validating also the case in which an IMU is available on the vehicle.

2) Streaming camera output: Two camera sensors, provided by MORSE, are installed on the 3D model of the vehicle. Each sensor provides a continuous image stream, at a frequency which can be obtained as a fraction of the one used to send telemetry information, such as Optitrack and IMU. Namely, attitude and location data are sent at 100 Hz, while camera output is streamed at 10 Hz or 20 Hz. Cameras can be controlled in depth of field and focal length. Other non-streaming cameras can be used in the simulation environment providing different perspectives for piloting and monitoring purposes.

Fig. 4. "Virtual" force sensor.

3) Force sensor: Force sensors play a crucial role when the manipulator is in contact with the environment. Unfortunately, even if the physic engine in Blender is able to detect and simulate contacts, it is quite difficult to extract the contact forces, that have to be used in the Simulink model as inputs for the dynamical model of both the UAV and the manipulator. Moreover, it is not completely clear the contact model that Blender implements. For these reasons, a simple "virtual" force sensor has been implemented, and its rationale can be better understood by referring to Fig. 4.

The Blender model is equipped with a small mass (i.e., the grey small sphere in the picture), that is moving under the effect of a force. Such force is computed in Simulink as the result of a simple PD control law that, when there is no contact, is able to send to zero (in steady state) the error between position of the mass and position of the end-effector of the manipulator. As a consequence, when the manipulator is not in contact with the environment, the position of the end-effector (in Simulink), and the position of this mass (in Blender) are *almost* the same. Maximum error and transient response are determined by the operator behavior and can be properly changed by acting on the PD parameters.

In case the manipulator end-effector gets in contact with the environment, the motion of the mass is constrained, since Blender is able to detect the contact. The error between position of the mass and desired set-point for the end-effect increases, and it can be used to compute the contact forces on the basis of a specified contact model. Such contact forces are computed in Simulink and are a further input for the blocks that model the manipulator and the UAV. In this way, we have freed Blender from the computation of the contact forces. This task has been left to Simulink, with a clear advantage in terms of numerical stability and "control" of the contact dynamics.

D. Model of the environment

Blender provides a very complete GUI that allows modelling of complex 3D scenarios. Meshes (3D objects shapes) can be imported from different standard CAD formats, and realistic textures can be applied to the surfaces. Complex objects, like the UAV with all the sensors and actuators, can be easily shared among different scenes. Most of the

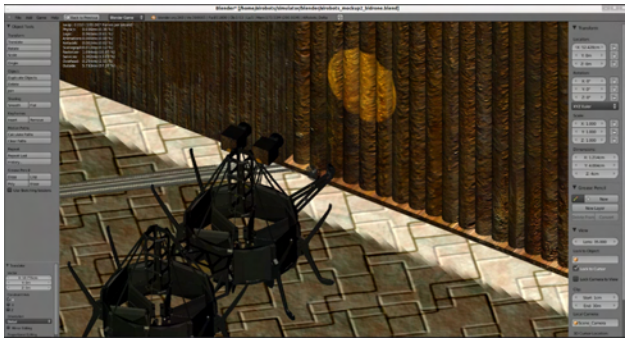


Fig. 5. A typical operative environment.

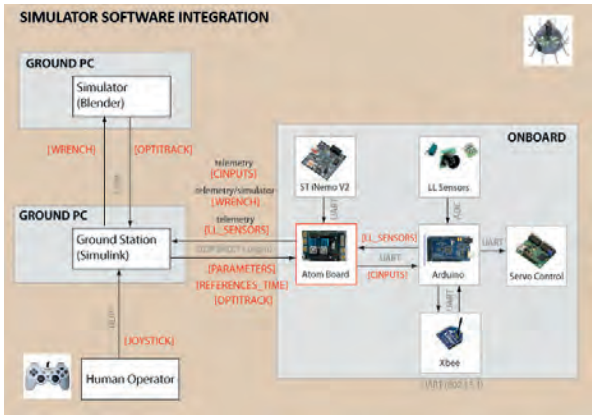


Fig. 6. Components involved in the simulation.

3D modelling is done using the 3D GUI editor, but it can also be fully programmed in Python, allowing run-time modifications of the environment or other parameters during the simulation. An example is reported in Fig. 5.

IV. Software integration

The simulation environment is designed to substitute parts of the real vehicle and its sensors, while maintaining other software and hardware components (e.g., the low-level control and Human-Machine-Interface), that are exactly the same used in the real flights. As such, the simulation environment must communicate with these components in the same way that the real vehicle does. The main integration points are with the low-level control, with the supervisory control and with other possible ROS nodes. Each component must communicate to the others using preferably UDP datagrams over the network. By the nature of this schema, also reported in Fig. 6, it is possible to distribute the computation and the control laws over multiple computational units, up to achieving a true remote-controlled application over the internet.

Since low-level control algorithms relay on Simulink, the same data packets commonly used for the laboratory Optitrack installation are used to export attitude and position values of the virtual UAV. This lead to a 1-to-1 replication of the physical testing environment that speeds up the development phase. Torques and forces from the environment and the control laws are applied to the rigid body and then new values for attitude and position are computed via direct integration of the dynamical equation.

As far as the integration with the supervisory control is concerned, the simulator allows the operator to set a number of way-points in order for the high-level control law to compute a path for the vehicle. In this situation, the environment model should be similar to the one of the end-user application, so that the operator is able also to employ the simulator to plan the trajectory for the real inspection. The simulator can also be used as a benchmark to have a 3D view of real-flight trajectories, so that the operator can evaluate the actual mission with respect to the one planned before.

Finally, additional high-level functionalities can be integrated using an off-the-shelf middleware, and in particular ROS, able to provide a publish-subscribe communication paradigm. The integration with ROS happens by defining suitable ROS nodes having on one side a public ROS interface and, on the other, the ability to communicate over the UDP network with the simulator interface.

V. Force control experiment

To test the effectiveness of the proposed simulator environment, hereafter we detail an experiment in which the ducted fan prototype is required to enter into contact with a vertical surface and to apply a certain force. To succeed in this complex task, the goal of the control design is twofold. On one hand, the controller should be able to stabilize a given free-flight configuration maintaining the end-effector in a desired vertical and lateral position. This feature can be employed, for instance, to move the tool installed on the end-effector close to the area to be inspected before entering into contact with the surface. On the other hand, the control law should be designed to perform docking maneuvers, namely it should be able to stabilize the aerial manipulator while applying certain forces to the vertical surface by means of the end-effector. This last feature is required to actually perform the inspection-by-contact operations.

Motivated by the effectiveness of the energy-based approaches, in applications pertaining physical interaction between robots and the environment, an impedance controller [17] is proposed to meet the two above control goals at once. In particular, the stability of a desired equilibrium point is obtained by shaping the energy function of the system to have a desired minimum (energy-shaping), and then by dissipating energy to asymptotically converge to it (damping-injection). More details on this point can be found in [18].

A crucial point to implement this controller in the simulation environment is to measure the contact forces with the vertical surface so as the dynamics of the vehicle can be actually influenced by the physical interaction with the environment. To achieve this goal, the force sensor described in Section III-C.3 has been employed. The forces measured in Simulink during the experiment has been depicted in Figure 7. When the system reaches the equilibrium, this force represents both the force applied to the environment and the reaction force applied back to the manipulator and then to

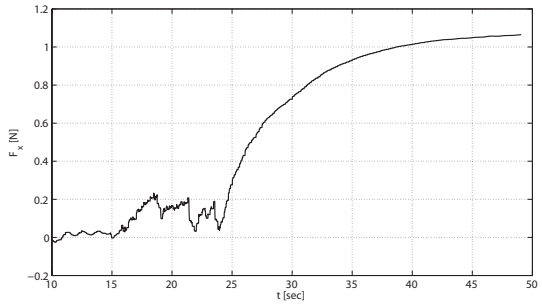


Fig. 7. The force measured by the force sensor during a physical interaction with a vertical surface.

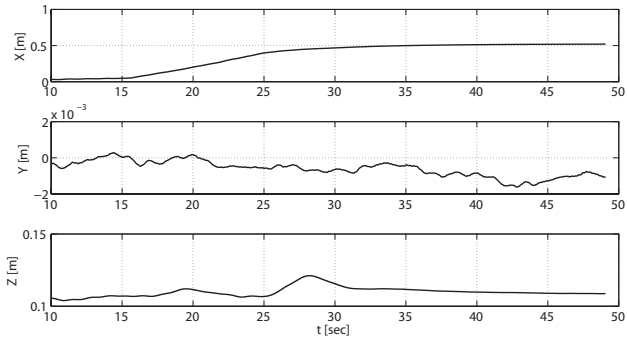


Fig. 8. The position of the vehicle during a physical interaction with a vertical surface.

the vehicle. The position of the system during the overall simulation has been depicted in Figure 8 while the attitude of the vehicle is given in Figure 9. As a consequence of the under-actuated nature of the ducted-fan dynamics, the pitch angle of the system has to tilt in order to apply the desired force to the surface.

VI. Conclusion

In this work a simulator environment suitable to model aerial robots physically interacting with the environment has been presented. The simulator has been designed in order both to train a human operator and to test and validate the control algorithms before real flight experiments. Both the software architecture and the main components allowing to obtain accurate mathematical modeling of the specific scenario have been presented. Future improvements will be focused primarily on an improved aerodynamic model of the vehicles to capture phenomena such as ground effect or other aerodynamic disturbances arising when the system is flying close to obstacles or the surfaces to be inspected.

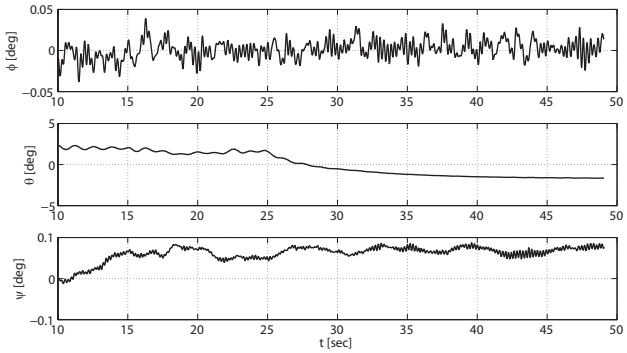


Fig. 9. The attitude of the vehicle during a physical interaction with a vertical surface.

References

[1] E. Johonson and S. Mishra, "Flight simulation for the development of an experimental UAV," in *AIAA Modeling and Simulation Technologies Conference and Exhibit, Proceedings of the*, Monterei, California, Aug. 5-8 2002.

[2] W. Dixon, D. Moses, I. Walker, and D. Dawson, "A simulink-based robotic toolkit for simulation and control of the puma 560 robot manipulator," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 4, 2001, pp. 2202 –2207.

[3] "Webots website," http://www.cyberbotics.com/overview.

[4] "Gazebo Simulator website," http://gazebosim.org/.

[5] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, sept.-2 oct. 2004, pp. 2149 – 2154.

[6] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: Morse," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 46 –51.

[7] H. . Bruyninckx, "Open robot control software: the orocos project," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3, 2001, pp. 2523 – 2528.

[8] "MORSE website," http://www.openrobots.org/wiki/morse/.

[9] "Microsoft Robotics Development Studio website," http://www.microsoft.com/robotics/.

[10] "Virtual Robot Simulator website," http://robotica.isa.upv.es/virtualrobot/.

[11] "Simbad website," http://simbad.sourceforge.net/.

[12] "Robosim website," http://robotics.ee.uwa.edu.au/robosim/.

[13] "3D Robot Arm website," http://sourceforge.net/projects/srom/.

[14] "Blender website," http://www.blender.org/.

[15] K. Buys, T. D. Laet, R. Smits, and H. Bruyninckx, *Simulation, Modeling, and Programming for Autonomous Robots*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6472, ch. Blender for robotics: Integration into the Leuven paradigm for robot task specification and human motion estimation, pp. 15–25.

[16] H. Bruyninckx, "Blender for robotics and robotics for Blender," 2004.

[17] N. Hogan, "Impedance control: an approach to manipulation: parts I-III," *ASME Journal on Dynamic Systems, Measurement and Control*, vol. 107, no. 1, pp. 1–24, 1985.

[18] F. Forte, R. Naldi, A. Macchelli, and L. Marconi, "Impedance control of an aerial manipulator," in *American Control Conference (ACC 2012). Proceedings of the*, Montréal, Canada, Jun. 27-29 2012.