Neuromorphic computation of optic flow data

Bio-inspired landing using biomorphic vision sensors

Final Report

Authors: Vasco Medici[1], Garrick Orchard[2], Stefan Ammann[3], Giacomo Indiveri[1] and Steven N. Fry[4]
Affiliation: [1] Institute of Neuroinformatics, University of Zürich and ETH Zürich, Switzerland. [2] Computational Sensory-Motor Systems Lab, Johns Hopkins University, Baltimore, MD. [3] Institute for Dynamic Systems and Control, ETH Zürich, Switzerland. [4] Rhine-Waal University of Applied Sciences, Emmerich, Germany.
ACT researcher(s): Tobias Seidl

Date: 23.09.2010

Contacts: Steven N. Fry, Professor for Bionics
Rhine-Waal University of Applied Sciences
Nollenburger Weg 115
D-46446 Emmerich, Germany
Tel: +49 2821 80673 - 610
Fax: +49 2821 80673 - 160
e-mail: steven.fry@hochschule-rhein-waal.de

Advanced Concepts Team
Fax: +31(0)715658018
e-mail: act@esa.int

Available on the ACT website
http://www.esa.int/act

Ariadna ID: 08/6303a
Ariadna study type: Standard
Contract Number: 21951/08/NL/CBI

Abstract

In this project, we demonstrated the feasibility of purely vision based landing strategies using analog VLSI sensors, in simulation.
We simulated the operation of a neuromorphic vision sensor in realistic planetary landing scenarios, at four different levels. At the lowest level we carried out SPICE simulations to characterize the response properties of the sensor's individual blocks; at the intermediate level, we used transistor equations to derive analytically the individual circuit block behavior and carry out full-chip simulations, by providing realistic inputs from planetary landing scenarios and at third level we implemented a simulink model of the sensor which reproduce its working principles. Our results are useful for evaluating the overall performance of the chip in this specific task.
In parallel we explored different control strategies for landing and analyzed their performances at different levels of realism for the optic flow extraction, starting from a purely geometrically calculated one to a realistic planetary images based one.
The simulations achieved their aim of enabling us to verify bio-inspired control strategies for planetary landing using different configurations of sensors.

Contents

Chapter 1

Introduction

1.1 General introduction / What was the call about

1.1.1 Vision based Landing

Landing is a critical stage of any planetary exploration mission. As smooth a landing as possible is desired to protect on-board equipment. In comparison to landing on Earth, landing on other planets is made more challenging by the constraints under which it must be achieved. No existing infrastructure is available on the surface for guidance, and the atmosphere (or lack thereof) limits the sensing modalities available. Furthermore, as with any space mission, cost rises rapidly with weight. Therefore custom lightweight vision sensors with low power consumption to limit battery weight are an attractive alternative to standard machine-vision systems, especially in the case of vision sensors capable of performing on-chip computation. Vision based navigation represents a challenging task; taking the inspiration from nature and especially from insects could provide the required design principles for both sensory computation and flight control.

1.1.2 The aim of the project

We carried out an assessment study on neuromorphic computation of optic flow data by combining top expertise in custom neuromorphic VLSI motion sensor design with real-time insect flight behavior and control analysis. The long term goal of this project is to reverse engineer the vision based flight control strategies of flies and combine it with the design of neuromorphic VLSI motion sensors and compact low-power neuromorphic controllers. In this work we carried out a systematic study to identify the principles used by flying insects to trigger landing responses; defined the specifications for designing compact, low-power analog VLSI motion sensors; integrated the VLSI chip design and the principles derived from the fly experiments into a neuromorphic controller; and validate the neuromorphic controller to assess the potential of neuromorphic systems for space applications. We developed an integrated simulation of a landing spacecraft whose thrusters were controlled with signals derived from optical flow sensors.

1.1.3 Why insects?

Flying insects are capable of impressive aerobatic flight maneuvers, despite their small size and hence limited neural resources. Their efficient neuro-motor control strategies emerge from millions years of evolution and high specialization. For this reason, the study of insects' sensory system and flight control strategies is of great interest for the development of unmanned flying vehicles (UAVs).
Insects rely strongly on vision for flight control (e.g. speed and altitude control, obstacle avoidance, landing, etc.). Therefore, it is essential to understand how the necessary information is extracted from the visual environment and how it is fused with the other proprioceptive sensory inputs. Vision based flight control in insects is an important research field that has been investigated for over half a century, leading to a large base of knowledge on which one can draw for the implementation of biomorphic sensors and controllers.

1.1.4 Reverse engineering of the biological sensing and control strategies

Recently, the use of engineering tools, such as control theory, and rigorous system identification techniques allowed gaining a deeper understanding of the biological flight control strategies. The Fry group at the Institute of Neuroinformatics (ETH/UZH) has applied a reverse engineering approach to quantitatively identify the control laws governing the free- and tethered-flight vision based speed responses in the fruit fly (Fry et al., 2009; Rohrseitz and Fry, 2010; Graetzel et al., 2010). The consistency of the results and the rigorousness of the approach suggest that the identified visual control principles are generic and can be adapted to the control of robotic hardware.

1.1.5 Neuromorphic sensors for optic flow computation

Analog neuromorphic VLSI design offers an alternative approach to standard digital computation methods for sensing. Programmable digital processors have the advantage of being reconfigurable, but for specific well defined applications an application specific integrated circuit (ASIC) can be both smaller and lighter. One may argue that digital electronics are already small and lightweight, but with the target of implementation in space applications, both size and weight become important factors. VLSI allows for the photoreceptors and computation circuitry to be fabricated on a single piece of silicon and is therefore particularly well suited to optical sensing.
Digital computation is power hungry, as is the process of digitizing incoming analog signals, specifically in imaging applications when multiple pixel values must be digitised in parallel. Subthreshold analog design provides a low power alternative by removing the need for digitization and digital computation. Full chip power consumption below 1mW is typical in such designs.
The nature of analog computation also ensures that results are available almost instantaneously because computation is distributed and performed in parallel.
Designing an ASIC using neuromorphic principles can therefore result in compact, lightweight,

low-power and extremely fast sensors.
Neuromorphic computation of optic flow data has been implemented on VLSI chips and investigated for many years. VLSI models of fly large-field neurons have already been proposed over ten years ago (Liu, 1996; Kramer and Koch, 1997; Harrison and Koch, 1999).
The Indiveri group at INI has significantly contributed to the field with optic flow chips for computing time-to-contact, focus-of-expansion and target position (Indiveri et al., 1996a,b; Indiveri, 1997). Recently, this work has been extended to single- and multi-chip vision systems for robotic applications (Indiveri, 2008) and to investigate control strategies that combine the output of multiple vision chips.

1.2 Approach

In this project, we investigated the applicability of bio-inspired vision sensing and flight control strategies for the autonomous landing of space vehicles under different environmental conditions. To this end, we designed a complete simulation environment comprehensive of spacecraft dynamics, visual and physical environmental conditions, vision sensor and flight controllers. This approach allows combining the knowledge and competences in the field of reverse engineering based biomorphic flight control and in the field of neuromorphic vision sensing, while providing a direct testbed for the arising hypotheses.

Chapter 2

Results

2.1 Simulation framework

The simulation framework consists of:

- Model of the spacecraft
- Environment providing visual input to the sensors and the planetary conditions
- Model of the sensor for optic flow computation
- Model of ego-motion computation from sensory data
- Controller

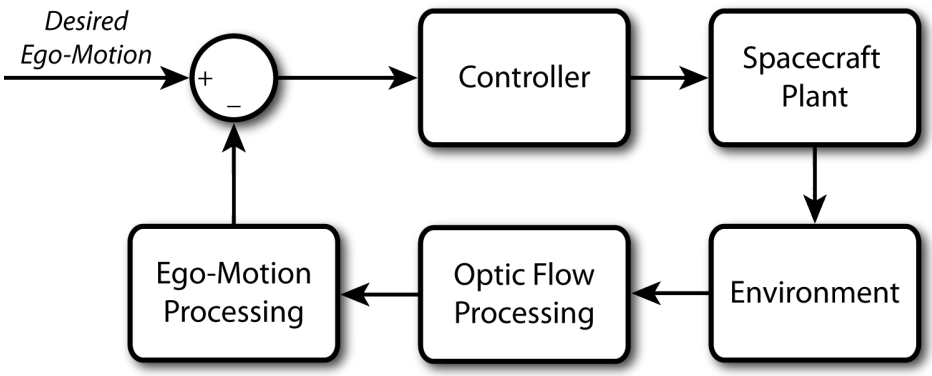


Figure 2.1: Block diagram of the simulation environment

Using this very general environment, different strategies for each part in the loop can be tested. It is also possible to use simple models for some parts at first and already close the loop, and then gradually update and extend all the parts as soon as new versions become available. This allows to get preliminary results early in the process.

The simulation framework was implemented in Simulink (the Mathworks). A socket was used to interface the visual environment simulator PANGU (planet and asteroid natural scene generation utility; Space Systems Research Group, University of Dundee, c.f. 2.4) with the Simulink model.

2.2 Vehicle model

2.2.1 The Spacecraft Model and Thruster Configuration

In order to keep the simulation environment as general and unconstrained as possible, the spacecraft model was chosen to be fully actuated and to have 6 degrees of freedom.
The spacecraft is assumed to have a spherical shape with radius r and initial mass m.
The mass is going to change due to the fuel consumption. In fact, most of the initial mass is fuel, such that the change in mass is significant and cannot be neglected.
For the simulations during the project, a radius of 0.5 meters and an initial mass of 10kg has been chosen. Those numbers are arbitrary, but seem to be somewhat reasonable, because the goal of the project was to control a small and lightweight vehicle. It is, however, easy to change those parameters, as they only need to be changed in the simulink file. Of course, the controller parameters might need to be be adapted too in such a case.
There are six thrusters distributed on the spacecraft to control its movement. It is assumed that the thrusters can generate a positive or negative force, so in reality, this setup would probably need two thrusters that are aligned but facing opposite directions for each thruster in the model.

The coordinate frame as well as the roll, pitch and yaw angles have been chosen according to the NASA standard airplane convention: x 'forward', y 'right'and z 'down'. The roll angle (ϕ) is measured around the x-axis, the pitch angle (θ) around the y-axis and the yaw angle (ψ) around the z-axis. The rotations are applied in the order of roll-pitch-yaw.
Figure 2.2 shows the spacecraft, the body fixed coordinate frame and the six thrusters. Thrusters 5 and 6 point in the direction of the x-axis. They can therefore be used to control the position in x-direction and the yaw angle. Thrusters 3 and 4 point in the direction of the z-axis and control the movement in z-direction and the pitch angle, while thrusters 1 and 2 point in the direction of the y-axis and control the movement in y-direction and the roll angle.

2.2.2 Coordinate Frames and Transformations

Two coordinate frames are placed in the scene. One is a fixed inertial frame that is assumed to lie on the planet's surface. The other is the spacecraft body fixed frame shown in figure 2.2. The inertial frame on the planet's surface is defined to have its center on the surface, its x and y axes lying in the tangential plane at that point and the z-axis pointing downwards towards the center of the (spherical) planet (see figure 2.3).

It is assumed that for roll, pitch and yaw angles each equal to 0, the axes of the inertial frame and the body fixed frame are aligned.

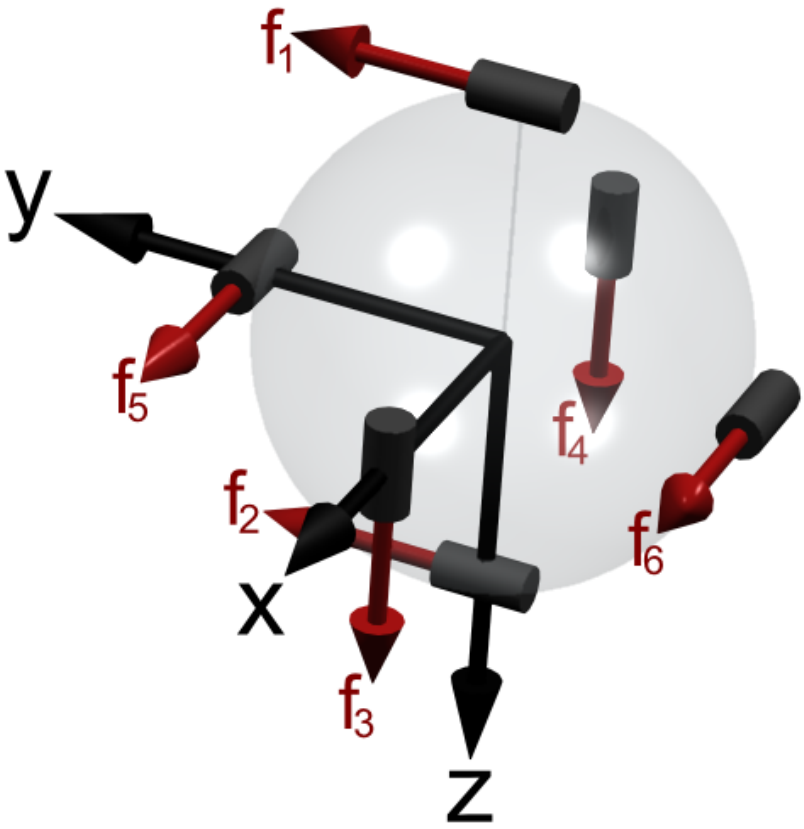


Figure 2.2: thruster configuration.

The coordinate transformation from the body fixed frame to the inertial frame can therefore be written as:

$$R_{BI}(\phi,\theta,\psi) = \begin{pmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \sin\phi\cos\psi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix} \tag{2.1}$$

2.2.3 The Differential Equations of the Spacecraft

There are two things that need to be known about the spacecraft:

1. Location of the center of the body fixed frame with respect to the inertial frame.
2. Rotation of the body fixed frame with respect to the inertial frame.

This information is provided by the six differential equations of motion. But before they can be written down, the coordinate transformation from the body fixed frame to the inertial

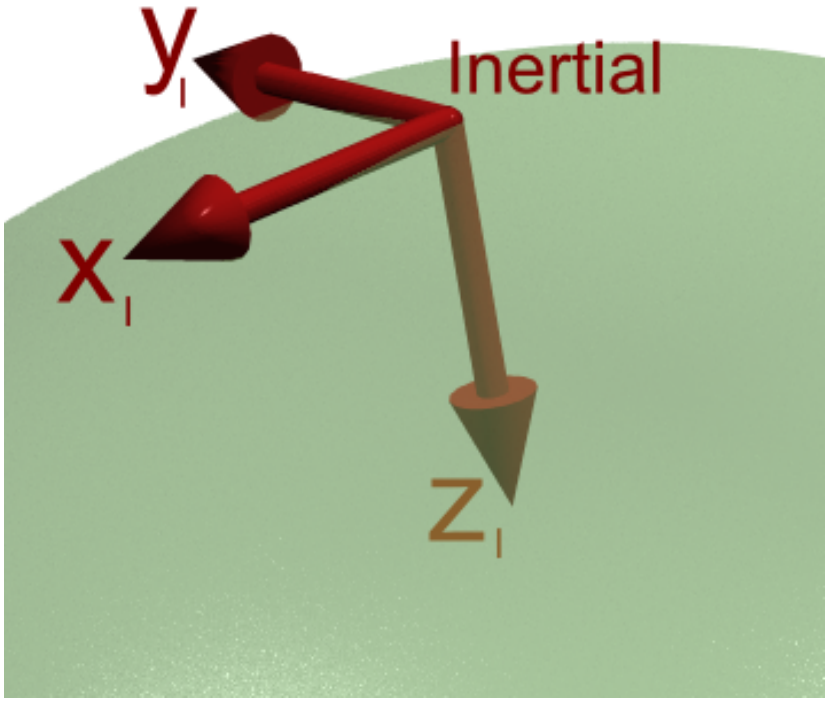


Figure 2.3: Inertial Coordinate Frame on the Planet's Surface.

frame has to be considered.
According to the way the thrusters are placed on the spacecraft, they generate the following force vector in the body fixed frame:

$$F_B = \begin{pmatrix} f_5 + f_6 \\ f_1 + f_2 \\ f_3 + f_4 \end{pmatrix} \tag{2.2}$$

with f_x being the force generated by thruster x.
Since the translation of the body fixed frame has to be known with respect to the inertial frame, the force vector is transformed into the inertial frame before calculating the equations of motion:

$$F_I = R_{BI} \cdot F_B \tag{2.3}$$

Considering the transformed forces, the six differential equations of motion can easily be found using newton's law of forces and moments:
with

$$\begin{aligned} t_x &= (f_5 + f_6) \\ t_y &= (f_1 + f_2) \\ t_z &= (f_3 + f_4) \end{aligned} \tag{2.4}$$

and

$$I = \frac{2}{5}mr^2 \tag{2.5}$$

The equations of motion are written as:

$$\begin{aligned} m\ddot{x} &= \cos\psi\cos\theta\cdot t_x + (\cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi)\cdot t_y + (\cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi)\cdot t_z \\ m\ddot{y} &= \sin\psi\cos\theta\cdot t_x + (\sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi)\cdot t_y + (\sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi)\cdot t_z \\ m\ddot{z} &= mg - \sin\theta\cdot t_x + \cos\theta\sin\phi\cdot t_y + \cos\theta\cos\phi\cdot t_z \end{aligned} \tag{2.6}$$

for the three translational degrees of freedom, and

$$\begin{aligned} I\ddot{\phi} &= (f_1 - f_2)r \\ I\ddot{\theta} &= (f_4 - f_3)r \\ I\ddot{\psi} &= (f_6 - f_5)r \end{aligned} \tag{2.7}$$

for the three rotational degrees of freedom [1].

2.2.4 Model of the Nonlinear Spacecraft Dynamics (The Plant)

The model (figure 2.5) has been implemented exactly according to the differential equations shown in the last section. Due to the page width limitation, the above image is probably too small to be readable for most readers and it is only meant to give the reader an overview over the model by referring to the coloration of the blocks. Please look at the actual simulink model provided with this text on a CD for a detailed view of the blocks.
There are 12 integrators at the heart of the model (dark blue and cyan/turquoise blocks). They integrate, for example, from $\ddot{x}$ to $\dot{x}$ and then from $\dot{x}$ to x. This is necessary, because the differential equations only show an influence on the second derivative of each state variable. This simulink model therefore has 12 states (more on that in section 7.5).
All the other blocks are now used to feed the right input to the second derivatives of each state, according to the differential equations.
The inputs (green on the left) to the model are the six thruster forces, while the output (green on the right) is all 12 states ('x') on the one hand, which would not be available on the real spacecraft, but is used to simulate the sensor. On the other hand, the second output ('y') is configured using a matrix C to only show certain states, which can be measured in reality. On the real spacecraft, only 'y'will be known.

[1] In this model, no atmospheric drag is considered, additional terms will be introduced to the differential equations of translation in section 2.6.1

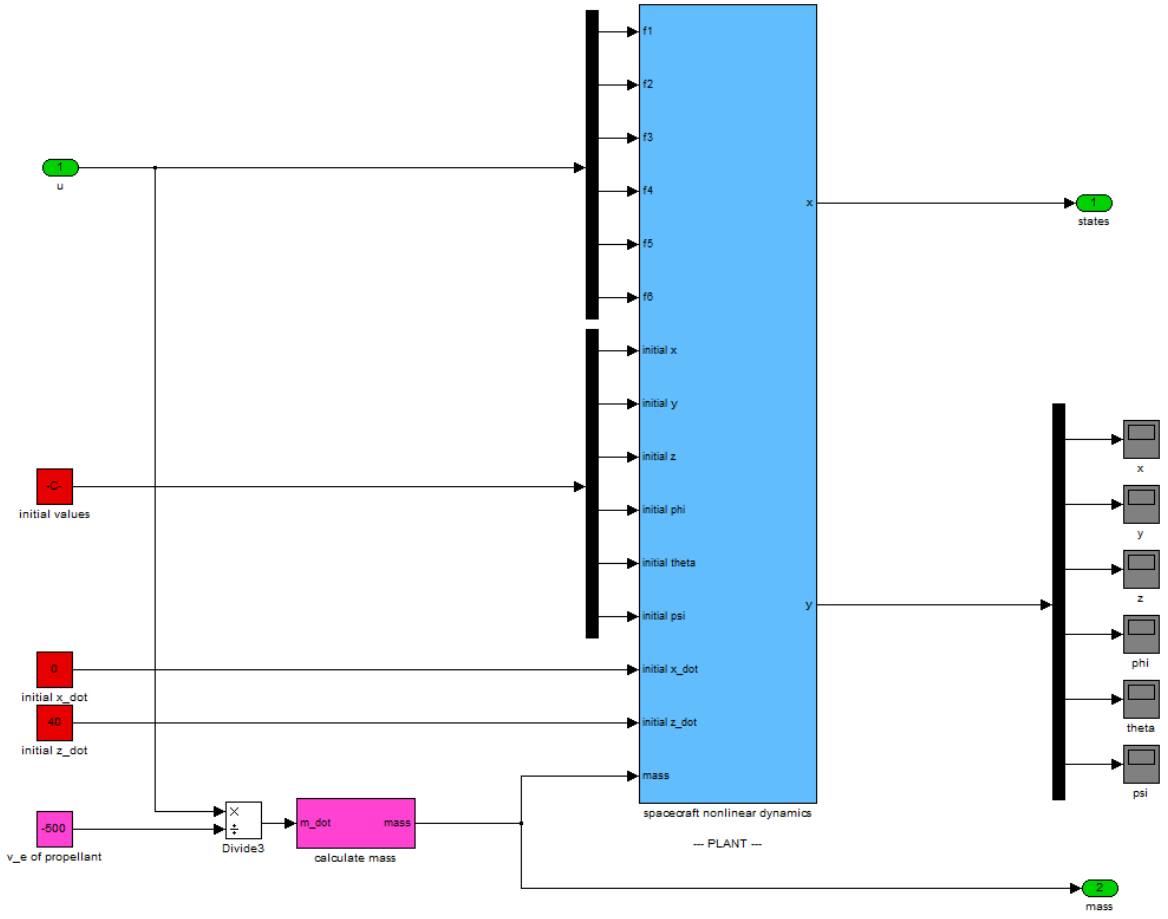


Figure 2.4: Spacecraft model Simulink block

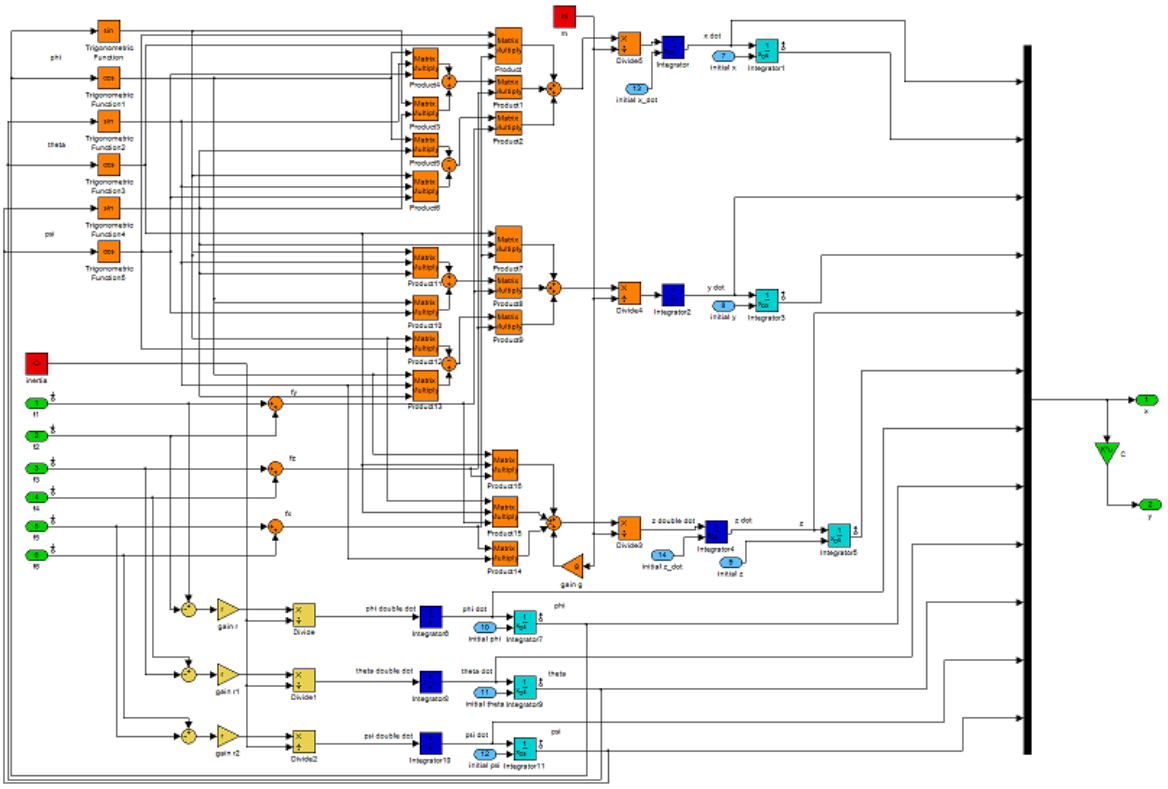


Figure 2.5: Simulink model of the spacecraft dynamics

2.2.5 Linearized System

The (nonlinear) system discussed in the last section has been linearized in order to find linear control laws. For the linearization, the following states have been chosen:

$$\begin{aligned} x_1 &= \dot{x} \\ x_2 &= x \\ x_3 &= \dot{y} \\ x_4 &= y \\ x_5 &= \dot{z} \\ x_6 &= z \\ x_7 &= \dot{\phi} \\ x_8 &= \phi \\ x_9 &= \dot{\theta} \\ x_{10} &= \theta \\ x_{11} &= \dot{\psi} \\ x_{12} &= \psi \end{aligned} \tag{2.8}$$

and

$$\begin{aligned} u_1 &= f_1 \\ u_2 &= f_2 \\ u_3 &= f_3 \\ u_4 &= f_4 \\ u_5 &= f_5 \\ u_6 &= f_6 \end{aligned} \tag{2.9}$$

The system was linearized around the setpoint:

$$\begin{aligned} x = y = z = \dot{x} = \dot{y} = \dot{z} = \phi = \theta = \psi = \dot{\phi} = \dot{\theta} = \dot{\psi} = 0 \\ u_1 = u_2 = u_3 = u_4 = u_5 = u_6 = 0 \end{aligned} \tag{2.10}$$

the linearized system matrices are found to be:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & \frac{1}{m} & \frac{1}{m} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{m} & \frac{1}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{m} & \frac{1}{m} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{r}{I} & -\frac{r}{I} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{r}{I} & \frac{r}{I} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{r}{I} & \frac{r}{I} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.11}$$

The output matrix C has to be chosen according to the states that are known/measured.

2.3 Physical Environment

Two different environmental conditions have being tested, one moon-like and one mars-like. The mars-like environment was modeled with a gravity of $g = 3.654\frac{m}{s^2}$ and an atmosphere modeled according to the equations provided by NASA (http://www.grc.nasa.gov/WWW/K-12/airplane/atmosmrm.html). The density of the atmosphere ρ can be modeled as:

$$\rho = \frac{p}{0.1921(T + 273.1)} \tag{2.12}$$

$$p = 0.699e^{-0.00009h} \tag{2.13}$$

$$T = -31 - 0.00998h \tag{2.14}$$

where h is the height from the ground. The atmospheric drag was then computed according to the equation:

$$Fd = 0.5C_d A_f \rho v \left|v\right| \tag{2.15}$$

with C_d being the drag coefficient ($C_d = 0.47$ for a sphere), A_f being the frontal area ($A_f = \pi r^2$ for a sphere of radius r), v being the airspeed.
In the moon case we only simulated lunar gravity $g = 1.627\frac{m}{s^2}$, drag is inexistent it the absence of an atmosphere.

2.4 Visual Environment

In order to have a fully functional simulation environment, the sensor described in the last section has to get an input representing what the real sensor on the real spacecraft would perceive. We created an artificial environment according to the orientation and position of the sensor and rendered an image using a program called PANGU (Planet and Asteroid Natural scene Generation Utility; Space Systems Research Group, University of Dundee).

2.4.1 PANGU

PANGU can be used to artificially create 3 dimensional models of a planet's surface. To do this, one can specify a variety of parameters, the most important ones being the surface size, the average number of craters per square kilometer and their size distribution, as well as the addition of boulders, dust or fog, if the represented planet features any of those. For detailed information about PANGU, please refer to the manual. It is possible to specify a camera position, orientation and field of view within the PANGU model and with that one can render images corresponding to what a sensor would see within its field of view.
PANGU is easily integrated into the simulation environment using a socket, such that the current spacecraft position and orientation as well as the distribution of the sensors on the spacecraft are sent to PANGU directly from simulink. Given those pieces of information, the corresponding images for each sensor are returned to simulink by accessing the PANGU program over an internal TCP/IP connection. Unfortunately, there were some problems regarding PANGU: the program is still in development, and some important features are not yet available, or did not work correctly. For example, it is not yet possible to see the sun

within the 3D environment. Since it is planned to have sensors all around the spacecraft, it would be desirable to be able to use the information of where the sun is. Another very big problem was the planet surface itself. Unfortunately, it was not possible to create very large surface patches. Since the spacecraft will be at an altitude of about 2km at the beginning of the mission, and the field of view of the sensors can be up to 90°, quite a large surface patch is needed to avoid seeing the edges within the field of view. Despite considerable efforts we failed at generating large enough surfaces. PANGU in the current version (2.7) is probably not well suited. Because of those difficulties, we combine PANGU and the Matlab Virtual Reality toolbox. The first to precisely render the landing site region, the second to render the rest of the 360 degrees field of view.

2.4.2 Matlab virtual reality toolbox

Simulink offers the possibility to include VRML (Virtual Reality Modeling Language) files in a specific block. It is then possible to send commands to that virtual reality block and get signals (or images) back from it. We created a virtual reality consisting of a planet, the sun and a movable camera, in order to have a 3D environment including the sun and a full-scale planet. In VRML, it was possible to create the planet and the sun at their real scale. Of course, this introduces enormous diameters and distances (diameter of the sun and the distance of the sun to moon or mars) but due to VRML's usage of vector graphics, this did not cause any problems. This virtual reality can therefore be used to get any desired view on the planet, including a 360° environment featuring some random star pattern, and the sun which is visible as a yellow circle emitting light. Unfortunately, there were problems once again. Since for the project optic flow is used to control the whole landing procedure, the planet's surface obviously needs a texture representing the real planet's surface. The problem in the VR environment is that although there is a possibility to add a texture, the resolution of this texture was way too low, such that the image of the surface was not usable anymore at some point during the approach. Because of that, we decided to combined the VR and PANGU to create usable images.

2.4.3 Combination of PANGU and Matlab virtual reality toolbox

This approach now combines the previous two separate ways of creating a virtual planet. The virtual reality is used to create a raw picture of the environment (picture A in figure 5). A detailed PANGU surface patch, viewed from a camera that is positioned in the same way as it was in the VR (fig.2.6B), is then superimposed on the VR picture, such that it is possible to extract the optic flow (fig.2.6C).

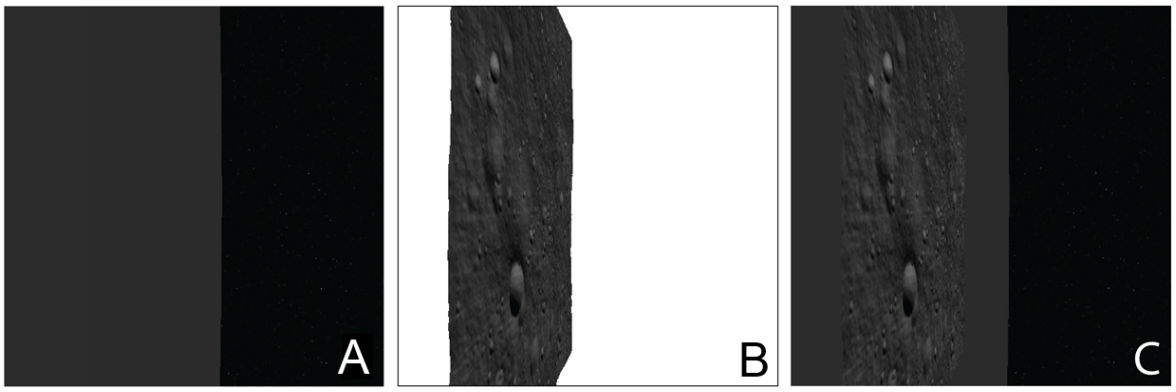


Figure 2.6: A: VR image; B: PANGU surface patch; C: final image (combination of the previous two)

2.5 The neuromorphic vision sensor

We approached the task of detecting optic flow by considering a recently fabricated subthreshold VLSI optical flow sensor. Using Matlab (Natick, MA), we simulated the sensor at four different levels.

1. Critical circuit blocks such as the photoreceptor are simulated at a transistor level to gain a better understanding of their operation and to predict their response to known signals. These simulations are typically run with a microsecond time-step and span only a few milliseconds.

2. A low level full chip simulation that simulates each part of the chip using approximate equations is used to verify the operation of the chip and investigate the sensitivity to transistor mismatch. The optical inputs for the low level simulations are extracted from images generated by Pangu, a simulator capable of generating images of a planet surface. These simulations are also typically run with a microsecond timestep, but the entire chip can be simulated for up to half a second.

3. A high level simulation of the chip was written based on the low level simulations. This high level simulation is used to provide approximate sensor outputs for use in the planetary landing simulations. The timestep for this simulation is arbitrary, the sensor simply provides an output when queried, but the simulation can span minutes or hours.

4. A faster high level simulation, based on the working principles of the chip, but not on its electronic implementation. The model is implemented in Matlab Simulink and uses realistic planetary images from Pangu (cf. 2.4) to compute the perceived optic flow.

The Neuromorphic Vision Sensor used in this work is a recently developed asynchronous event-driven motion chip, named Tracker Motion Sensor (TMS). The TMS extracts the spatial derivative and the temporal derivative of the temporal contrast change detected by each photoreceptor and extracts edges of moving objects and their velocity. The chip readout can

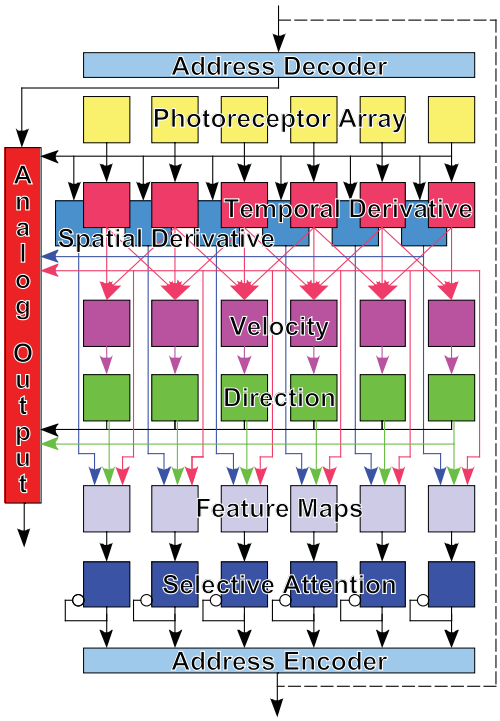


Figure 2.7: Tracker Motion Sensor Block Diagram: the photo-receptor's output is spatially and temporally derived, fast transitions of the photo-current triggers an event and the computation of velocity and direction of the event movement over the array. The temporal and space derivative and the speed are summed to compute the saliency map. A Winner-Take-All (WTA) network with self inhibition scans the saliency map in order of decreasing saliency. The identity of the winning pixel is a digital output of the chip and can be used to steer the readout of the analog values corresponding to the winning pixel.

be either "traditional" frame-based scanning of the pixels output, or asynchronous. In the latter case the output is read in order of decreasing pixel saliency, implementing a form of stimulus-driven attention.
Figure 2.7 shows the block diagram of the TMS. The input is an adaptive logarithmic photoreceptor (Liu, 1999), that responds to temporal variations of the logarithm of light intensity impinging on the photo sensitive area; this makes the photoreceptor sensitive to the relative contrast change, rather than to absolute illumination. Furthermore the photoreceptor is capable of adapting its response to the global scene illumination level, accomplishing a wide dynamic range. The temporal derivative circuit detects fast variations of the photoreceptor output, typically corresponding to edges of objects moving in front of the sensor. Such an event triggers the computation of velocity as time to travel of the event across neighboring pixels, using "Facilitate-and-sample" circuits based on (Kramer and Koch, 1997). The attentional read-out is implemented by means of a Winner-Take-All circuit (WTA) that selects the pixel with highest activity in the saliency map (Itti and Koch, 2001), computed by summing with tunable weights the outputs of the spatial and temporal derivative and of the velocity circuits. The attentional system comprises also a self-inhibition mechanism that implements shifts of attention and allows to scan the pixels output in order of decreasing saliency (Bartolozzi and Indiveri, 2009). The output of the chip indicates the location of the object moving with highest saliency and can be used to select the analog outputs of the corresponding pixel. In addition, it is possible to read out the analog outputs (spatial and temporal derivative, and velocity) of each individual pixel.

2.5.1 Transistor Level Simulation

The photoreceptor, temporal derivative, slow pulse and inhibition of return blocks were simulated using transistor current/voltage equations. Differential equations for each circuit were derived and solved using the Matlab "ODE15S" function. The slow pulse and inhibition of return blocks consist of the same circuitry just with different transistor sizes and capacitor values. A discussion of the transistor level simulations for each block follows.

Photoreceptor

The photoreceptor circuit (see Fig. 2.8) is designed to detect only changes in brightness. The feedback loop ensures that low frequencies are removed by adapting the operating point of the photoreceptor to new lighting conditions.
The photodiode was modelled as a capacitance in parallel with a current source that acts as the input signal. The capacitance can be approximated from the layout as

$$Cdiode = WLCSub + 2W + LCjsw \tag{2.16}$$

where W and L are the width and length respectively of the photodiode. The values used are: W=16μm, L =16μm, Csub=0.08fF, and Cjsw=0.51fF.
Knowing the circuit currents and capacitances we can approximate the derivative of the

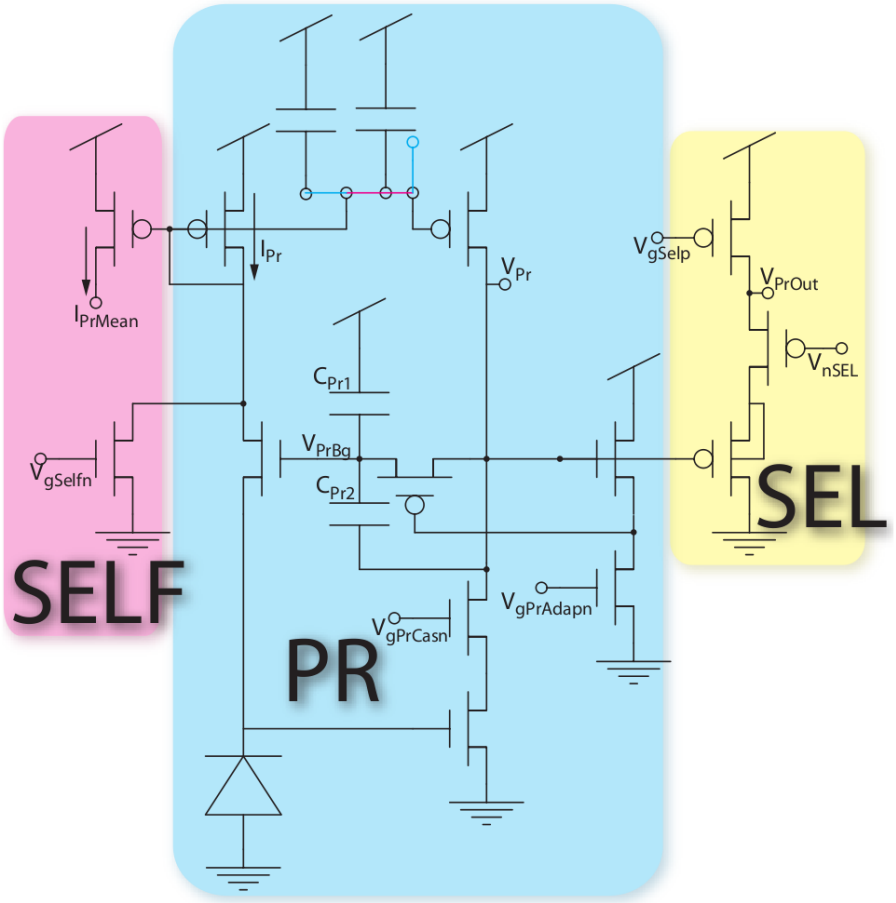


Figure 2.8: Adaptive and Self-biasing Photo-Receptor. The gate of the output PMOS is the maximum of the pixel's photocurrent. Such gate is the source of the NMOS reading the photocurrent. Its value is set by the maximum photocurrent in the pixels, so all of the photo-receptors will have the same power supply, even if this power supply is not at V_{dd}. The transistors NMOS of the photoreceptor, comprising the inverting amplifier have to be LONG, the M_{bg} to minimize the Early effect, the others to increase the gain of the amplifier. Also the biasing PMOS of the amplifier should be long, to increase the gain. In the SELECTION circuit the PMOS connected to V_{pr} doesn't have an isolated well, to save space (its function was to remove the κ dependence. Instead of this readout we could use a switch whose ds is connected to all the other pixels and to an output amplifier, but in this case one has a big capacitive load in the input node of the amplifier (+). The amplifier in in unity gain follower configuration. In the case of the I-V converter this is not a problem because the node where all the pixels are connected is the inverting (-) input, and the non-inverting input is connected to a reference voltage, that clamps the inverting input canceling (reducing) the capacitive load effect and the early effect.)

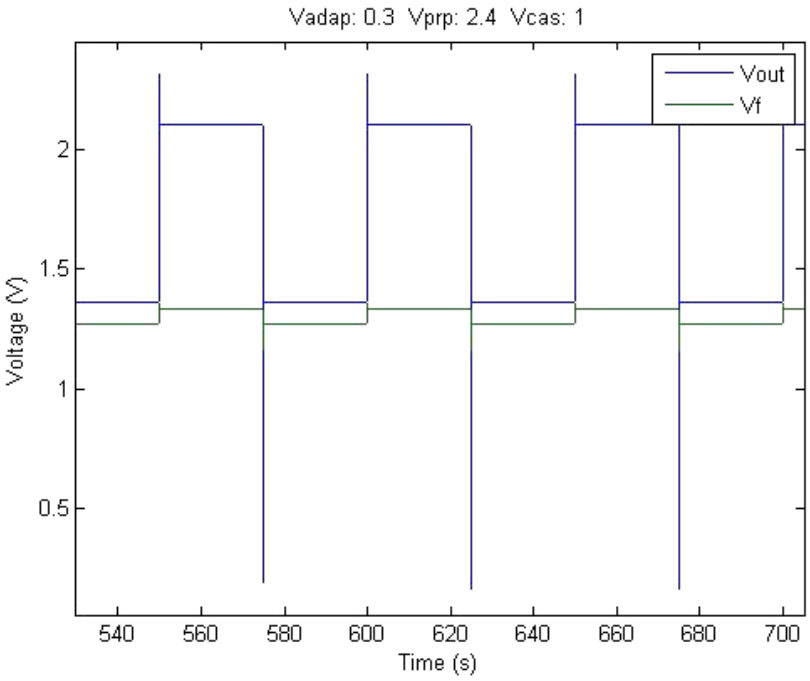


Figure 2.9: **Simulation results with neuromorphic chip**. In this first case the adaptation bias is very low and therefore adaptation is not noticeable. It is interesting to note that on negative edges, the output briefly goes below the feedback voltage, but then the circuit overcompensates, immediately pulling it back above the feedback voltage.

voltages at each time-step as:

$$dVr = IN1 - IdiodeCdiode \tag{2.17}$$

$$dVout = Ibias - IN3C1C2C1 + C2 \tag{2.18}$$

$$dVf = dVoutC2C1 + C2 + IfeedbackC1 + C2 \tag{2.19}$$

Simulations show an asymmetrical adaptation. The adaptation after negative edges is much faster than after positive edges. This sometimes causes the circuit to over-adapt as seen in the examples below. These behaviors were also seen in measurements made on the actual chip when focused on a flashing LED. The simulation results of Fig. 2.9 and Fig. 2.10 are generated with a square wave input to the photoreceptor.
The photoreceptor block acts to amplify the input photocurrent and adapt to new lighting conditions. The adaptation is much faster when adjusting to darker lighting than brighter lighting. In some cases, the adaptation is so strong that immediately after a decrease in lighting the circuit over-adapts resulting in a higher output than before. This has only been seen for sharp changes in lighting (a square wave), which will not be present in the images we are using for simulation. The simulations show that decreasing the bias current can actually result in a larger signal for high frequencies because it impairs the adaptation. After a number

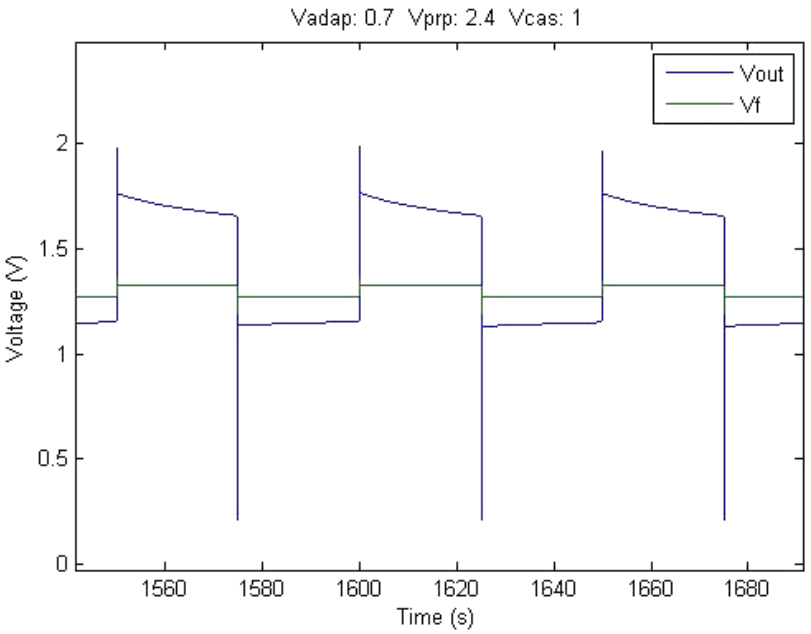


Figure 2.10: **Simulation results with neuromorphic chip**. In this case the adaptation has been increased even more, we now see adaptation during both the on and off sections, but the on section still has much more adaptation than the off. The off section adaptation is likely overestimated in simulation because the transistor is no longer sub-threshold, but is still being modelled using the subthreshold equations.

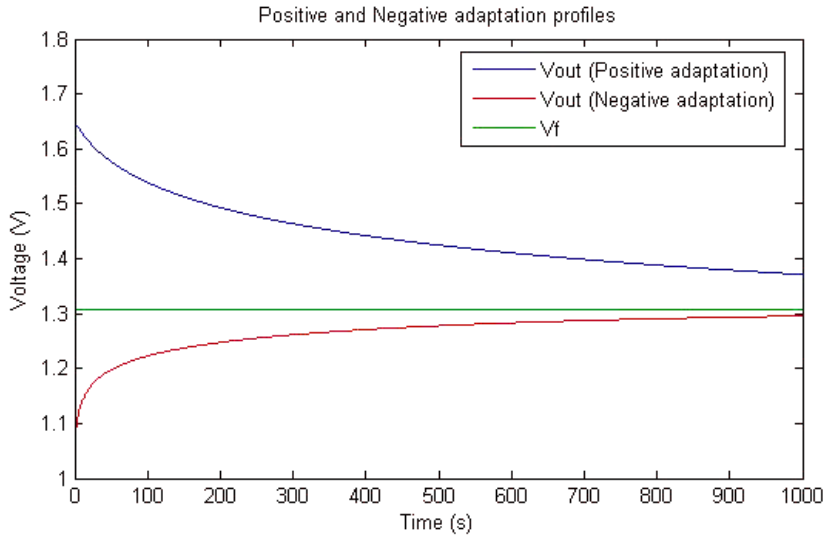


Figure 2.11: **Simulation results with neuromorphic chip**. Plot showing adaptation profiles for the same adaptation bias when output starts from an offset of +0.3V(blue) and -0.3V(red) from steady state. This clearly shows the asymmetrical adaptation.

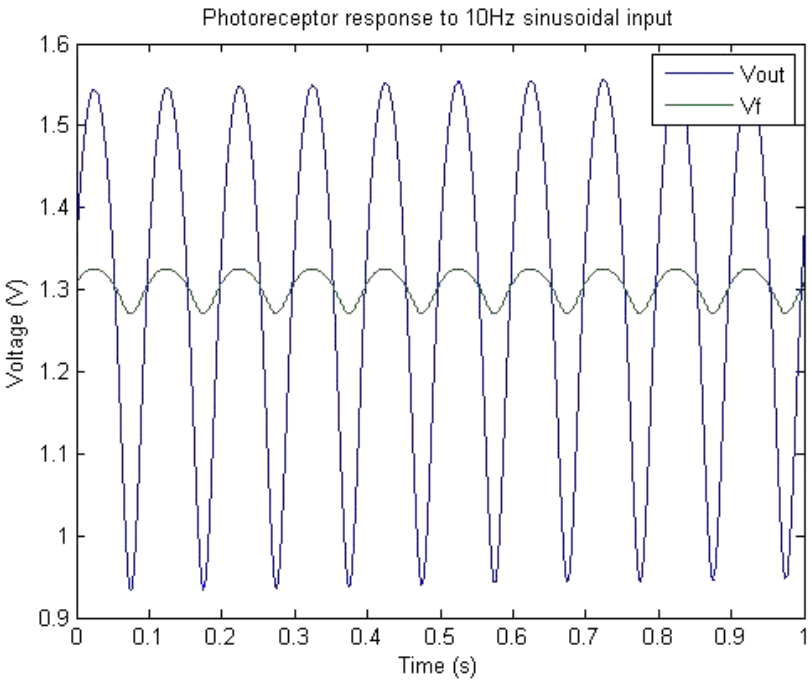


Figure 2.12: **Simulation results with neuromorphic chip**. Photoreceptor response to a 10Hz sinusoidal input. The amplitude of the output remains roughly constant for frequencies between 10 and 100 Hz.

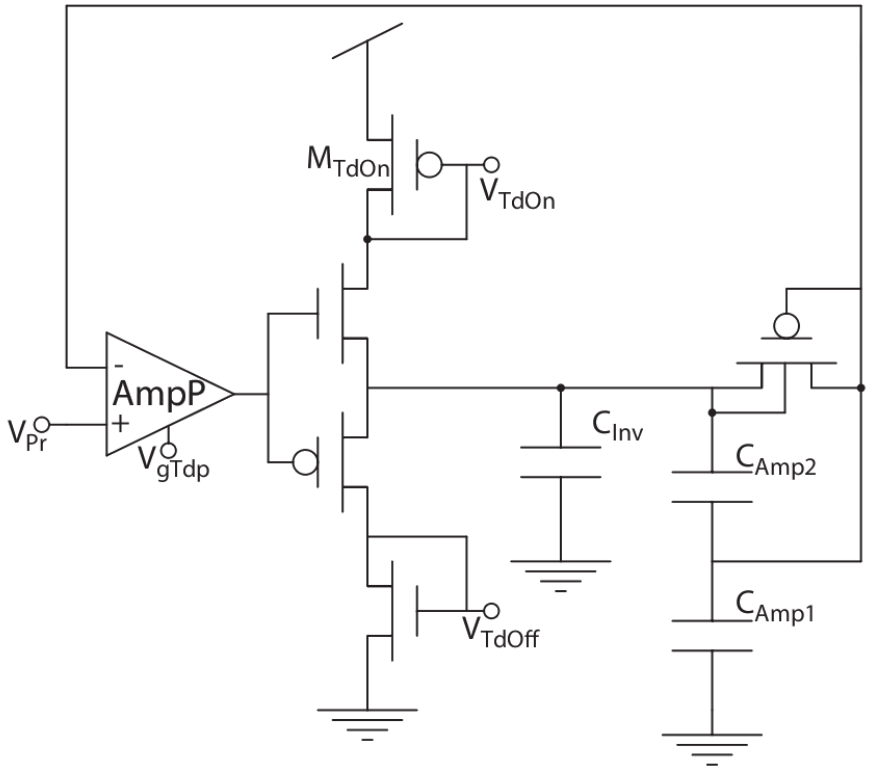


Figure 2.13: The circuit used to compute and rectify the derivative of the photoreceptor signal.

of simulations of the photoreceptor without high frequency inputs, it was observed that the output closely reflects a scaled version of the input current. As a result, the photoreceptor is modeled as a simple gain in the full chip simulations where slowly changing images are expected.

2.5.2 Temporal Differentiator

This circuit block is used to compute the derivative of the photoreceptor signal. It takes the form of a high pass active filter using an amplifier.
The circuit acts as an active high pass filter to approximate the derivative of the input voltage. Due to the configuration of the inner two transistors at the output of the amplifier, only one will conduct at a time. The outer two transistors simply act to mirror the current. The five transistors on the right rectify and combine the current output using current mirrors. The resulting output is a rectified current representing the derivative of the input voltage.
We simulated the temporal differentiator behavior in response to a step input of 50mV. The

simulations show an oscillation of the output node, but nevertheless the circuit appears to produce a feasible output current.
The NMOS and PMOS in the current branch after the amplifier are of equal size, but due to the relatively low mobility of charge carriers in the PMOS, the NMOS is likely to dominate. This means that the circuit will be responsive to positive changes in V1, but negative changes will occur much more slowly.
Simulation results indicate a possible oscillation of the circuit output, which could prove a problem when driving the edge detection thresholding circuit as it could cause the threshold to be crossed multiple times for each image edge.
This is a tricky circuit to simulate and the oscillations in simulation may be due to the unrealistically high speed of the amplifier. Simulating the amplifier in more detail by including capacitances to slow down the speed at which the output changes could remove these oscillations, but the oscillations have a period above 1ms, which means it is unlikely that the amplifier is the cause.
A more likely cause is the relative strength of the NMOS and PMOS in the current branch on the output of the amplifier. They are of equal size, but the relatively low mobility of charge carriers in the PMOS will result in it being dominated by the NMOS. The circuit will therefore respond fast to positive changes in the amplifier output, but slowly to negative changes in the output. This slow response to negative changes can cause the circuit to oscillate.
A very rough approximation of the Tobi element has been used for simulation, but this is not the cause of the oscillations. Short circuiting the Tobi element or removing it completely both still result in oscillations.
Actual on-chip measurements of the circuit output are expected to be available soon and may help in determining what needs to be changed in simulation. The transistor level simulations of the blocks described above gave us enough information to make higher-level abstractions and pass onto (much faster) behavioral simulations.

2.5.3 Behavioral Simulation

Behavioral simulations were implemented using approximate equations as described in the following section. The simulation allowed for a number of parameters to be varied. Most importantly, the angle of the array to the velocity can be changed. The angle between the array and the velocity can greatly affect the output as discussed under the high level simulation section.
Transistor mismatch of up to 20% was introduced wherever current mirrors are present to investigate the sensitivity of the design to mismatch.
As discussed in the circuit simulations section, the photoreceptor was approximated as a simple gain of its input. The input to the photoreceptors was obtained from Pangu images using five steps.

1. The mean of the image was subtracted from the original image to obtain an image with zero mean.

2. The image was upsampled using interpolation to allow for a finer temporal resolution in the input signals.

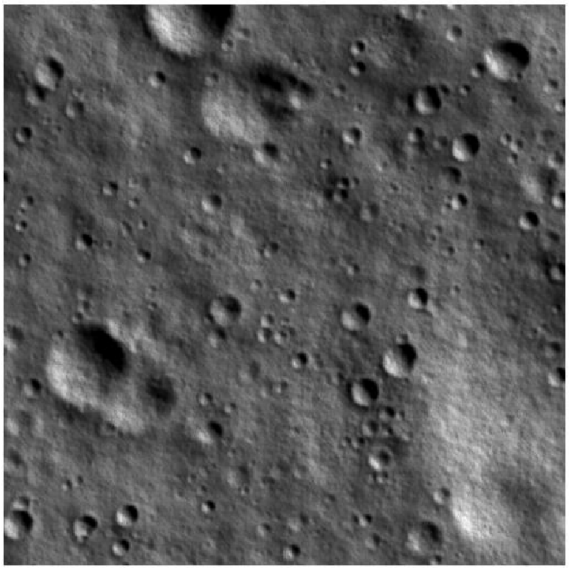


Figure 2.14: A Typical image generated by Pangu. This particular image was generated at a height of 2km with a Field of View (FOV) of 10 degrees.

3. The image was low pass filtered to model the non-negligible size of each photoreceptor
4. A line of image pixels was extracted, which corresponded to the path of the photoreceptor moving over the image.
5. The pixels were rescaled to obtain a set peak to peak output value (which is a variable in the code).

In actual simulations only the necessary portions of the image are filtered to decrease simulation time. The output peak to peak value cannot be inferred from the images because the image values themselves do not represent absolute light levels.
The spatial derivative block computes the difference between the outputs of the photoreceptors on the left and right of the current pixel, thus giving a measure of the spatial derivative. The circuit is known as the bump-antibump circuit and is described in Delbruck (1991). According to the circuit equations, a 10% change in the output current would require a difference in input current of 1.3V. Such high signal differences are unlikely, especially in the planetary landing scenario. The signal is therefore likely to be very small compared to the offset. This makes the circuit very sensitive to transistor mismatch, particularly for the bias setting transistor.
The spatial difference block is therefore unlikely to provide a useful input to the winner take all if transistor mismatch is taken into account. A one percent change in signal requires 0.4V difference between the input voltages, which is very unlikely, thus even if transistor mismatch is only at 1% it is likely to dominate the output signal.
The temporal derivative circuit block outputs a scaled and rectified derivative of the photoreceptor output voltage. The detailed simulations must still be verified against actual chip

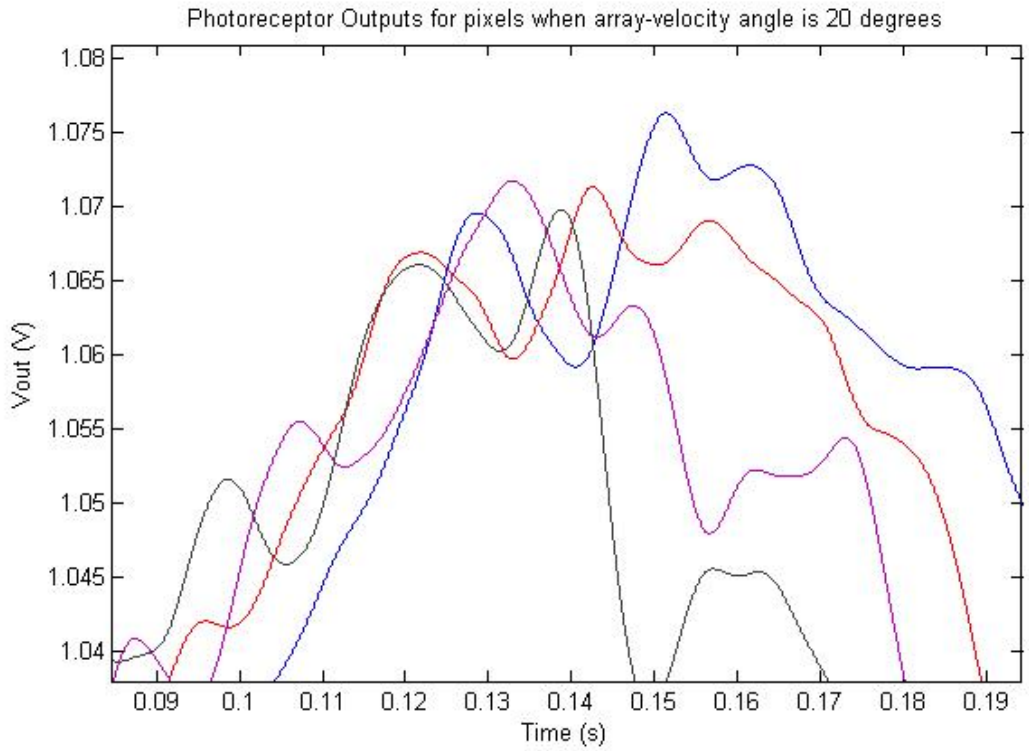


Figure 2.15: A subsection of the photoreceptor outputs for multiple photoreceptors with an array to velocity angle of 20 degrees. Each colour represents a different photoreceptor output. If the array to velocity angle were zero degrees, the outputs would just be time delayed versions of each other.

measurements, but in its most simple form the circuit output can be approximated as the scaled, rectified derivative of the input. This is the equation used in the full chip simulations.

$$Iout = C\frac{d}{dt}V_{photoreceptor} \tag{2.20}$$

The edge threshold block compares the temporal derivative to a set threshold and outputs a current when the threshold is crossed. The circuit takes the form of a winner take all circuit. When the Temporal Derivative input current is greater than the threshold input current then an output current is drawn through the top transistor. The voltage on Edge Output can be used to mirror this current.

The Fast Pulse block outputs a fixed length fixed voltage square pulse when it receives a current input from the Edge Threshold block. In simulation, this circuit simply gives a fixed length pulse when the edge circuit block output is high. This pulse length was set to 100 microseconds in simulation, but must be measured on the chip as it cannot be controlled with a bias.

The Slow Pulse block consists essentially of a Differential-Pair Integrator (DPI) low-pass filter. In the full chip simulations, the DPIs (one for each pixel) are simulated only once using differential equations to obtain the charge and discharge profiles. The full chip simulations then determine the DPI output based on the input voltage from the Fast Pulse block and the pre-generated charge and discharge profiles.

The Slow Pulse block for each pixel has the same bias voltages, but transistor mismatch causes the actual current values to differ. In simulation this is modelled as a random variation of up to 20% in the transistor W/L ratio. The random number is pulled from a uniform distribution over the range $[-20\%, +20\%]$.

When the Fast Pulse output from a pixel to the left or right of the current pixel is asserted, a Sample and Hold block samples the output of the Slow Pulse block. This gives a measure of the time taken for an edge to cross from the current pixel to an adjacent pixel (edge velocity). For each pixel there are two Sample and Hold outputs. One sampled when the Fast Pulse output of the pixel to the left is asserted and one when the output to the right is asserted.

The direction block is a three input winner take all circuit. One input is a pre-set threshold and the other two are the outputs from the Sample and Hold block. If either of the Sample and Hold outputs are above threshold, the greater of the two will be passed to the final winner take all as the velocity measurement. The circuit also outputs which sample was greater, indicating the direction of motion of the edge. If the threshold is the greatest input, then none of the outputs are asserted.

The winner take all can be modelled in more detail using similar equations to the two-input winner take all used for thresholding, but using the complete equations is unlikely to significantly alter the results. The current mirror used in the above circuit could possibly be improved by placing the PMOS switches at the drains of the mirroring transistors rather than at the source.

The output of “Temporal Derivative”, “Spatial Derivative” and “Velocity” circuits all get merged into a common feature map after passing through a current normaliser circuit. There are three current normaliser circuits per pixel. As the name suggests, the outputs of each current normalizer circuit is a scaled value of its inputs, such that the sum of the outputs is

equal to a predetermined value (set by an external bias).
Each pixel therefore contributes to creating a saliency map, which represents the sum of the pixel's normalised spatial derivative, temporal derivative, velocity measurement minus the output of the inhibition of return circuit (discussed below). The relative weighting of the inputs can be controlled by changing the biases of the current normalisers. An advantage of current mode design is that addition and subtraction can be easily implemented by connecting the current outputs to a common node.
The Saliency Map output is fed to a winner take all circuit with lateral excitation and hysteresis. The output of the winner take all determines which pixel's readings to consider. Hysteresis ensures smooth transitions between pixels, while the lateral excitation promotes transition to an adjacent pixel as an edge propagates across the photoreceptors.

2.5.4 Behavioral Simulation Results

The simulations show that the chip can accurately detect optical flow when transistor mismatch is ignored.
With the introduction of transistor mismatch of 20% the chip has a tendency to overestimate the optical flow by roughly 15% because when multiple pixels detect the optical flow it is biased to choose the largest value.
Simulation results so far appear to be in line with the results predicted by the equations for the high level simulation. Complexity continues to be added to the simulation framework in an attempt to more accurately approximate the operation of the chip.
Full details of the transistor-level and behavioral level simulations are provided in a separate report titled: "Simulation of a Neuromorphic VLSI Optical Flow Sensor" written by Garrick Orchard.

2.5.5 High-level Simulink model

Circuit level simulations of the AVLSI chip revealed being extremely slow, for this reason it would have been impossible to include a circuit level model of the sensor in the overall simulation. A high level model of the sensor was implemented instead; a model which reconstructed the ideal structure of the optic flow given the linear and angular position and speed of the vehicle with respect to the ground. This simple model appeared to be useful for the controller design, helping to identify the basic control principles that can be used to land given the optic flow as an input. However a model of the chip that computes speed from the actual visual environment and that can be used directly in the simulation would provide us with more realistic results. For this reason, we decided to implement a simulink model of the sensor that mimics its functioning at the level of the single pixels.
The model provides with the output of the single pixels. A winner take all for the selection of the fastest point on the image was designed but we preferred to use the mean of all pixels since the whole image is moving and the mean gives a more robust readout.

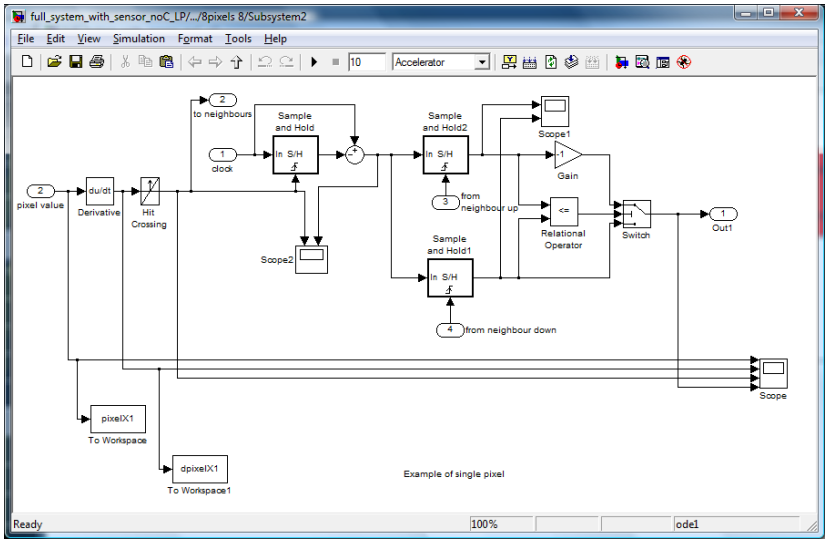


Figure 2.16: Single pixel Simulink design

Limitations

Discretization

The sensor is analog, which means that it works in a continuous mode, so discretization is an issue one needs to confront for its implementation in simulation. In particular, one needs to choose the adequate minimum timestep in order to still be able to measure high speeds. The sensor measures the time that an edge takes to move from a pixel to its next neighbor. The inverse of this time gives the retinal slip speed omega in pixels/s. Due to this inversion, the resolution in speed is not constant but varies exponentially. Figure 2.20 shows an example of resolution in speed at timestep=1ms (i.e. time resolution for the time to travel measurement = 1ms). To give a realistic comparison, assuming a 5 degrees field of view, a height of 1000 meters and an horizontal speed of 100 m/s on the 64 pixels linear sensor, omega would be 73.3 (pixels/s).

Simulation time

With a chosen timestep of 1ms (1000 frames/s per couple of chips), the simulation is still quite slow. For example the simultaneous simulation of two chips that use the same PANGU images to extract their pixel values takes 3 minutes of real time per second of simulation with PANGU 2.70 and 1 minute with PANGU 3.0pre (on Intel Q9550, 4GB ram, Vista 32-bit, Nvidia GeForce 9500GT).

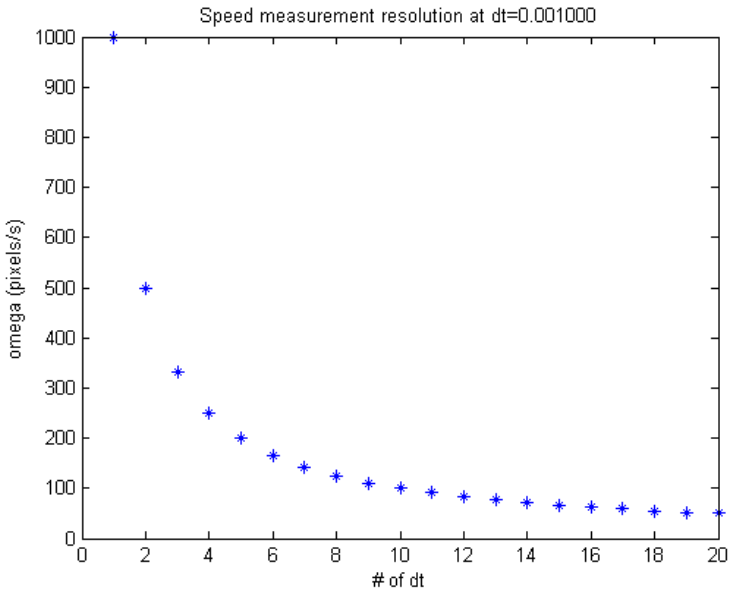


Figure 2.17: Resolution in speed: Discretization in the measurable speeds due to finite timestep.

Characterization

Speed response

With the sensor perfectly aligned with the direction of motion and within a speed range measurable with the current timestep choice, the sensor shows its ability to measure the predicted retinal slip speed as shown in figure 2.19. The first second of simulation gives strange results, it is an artifact due to the initialization of the sensor. These data are therefore discarded.
In this example, the noise level is very low and the output of the chip is optimal. It is important to underline that the choice of the appropriate threshold value for the trigger of a pulse in the sensor (see Garrick Orchard's report) needs to be set accurately. In fact, there is a trade-off between sensitivity and noise. The higher the threshold, the lower the noise, but the lower the sensitivity, which results in the sensor only being sensitive to sharp edges and not sensitive in smooth gradients. This choice needs to be made depending on the image structure.

Angle response / Aperture problem

The output of the sensor at different angles with respect the direction of motion was tested and the results are shown in figure 2.20. One can easily notice that as soon as the angle is not perfectly zero, the output of the single pixels gets very noisy, thus suggesting that a winner-take-all network for the extraction of the velocity signal is not really suited in this

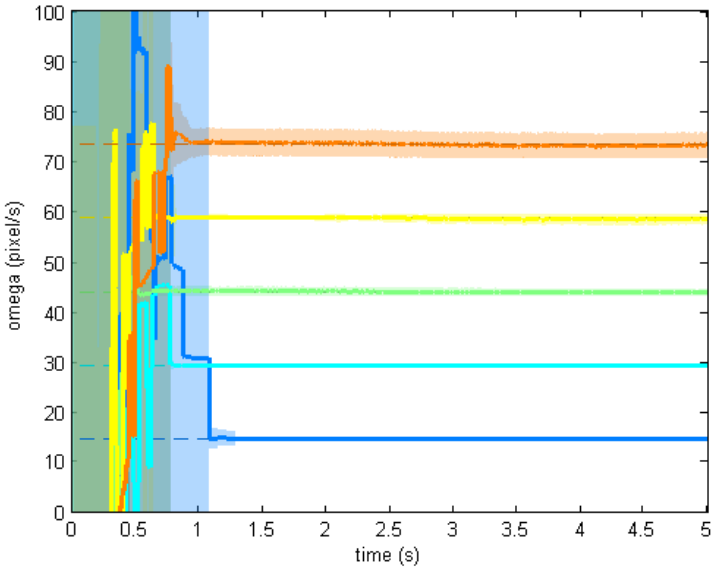


Figure 2.18: Speed response mean and std for the 64 pixels ($z = 1000m$ and $v_x = 20, 40, 60, 80, 100m/s$, $5s$ simulations). Dashed lines represent omega predicted from geometrical calculation

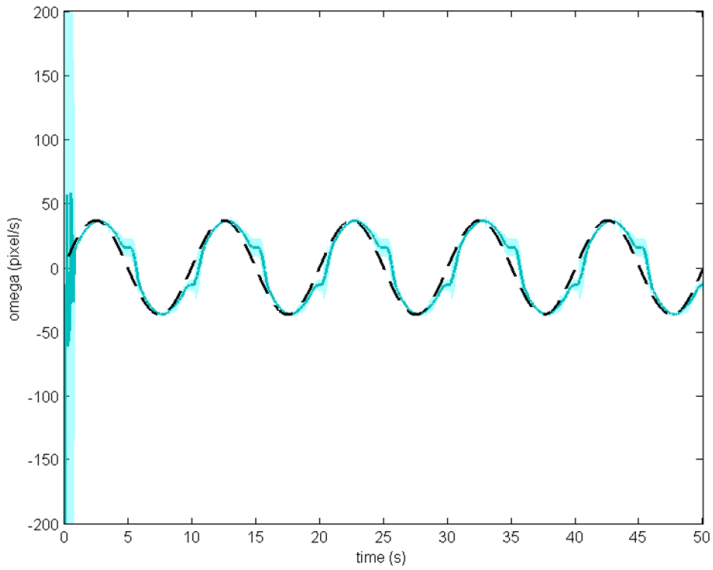


Figure 2.19: Speed response mean and std for the 64 pixels to a sinusoidally varying speed. Dashed line represents omega predicted from geometrical calculation

case. A winner-take-all network shows optimal results if there are objects moving in front of a static background. In the case of a moving sensor the whole image is expected to move more or less at the same speed, in which case one should consider if this solution is still usable. An average of the output of the single pixels may instead give better results.

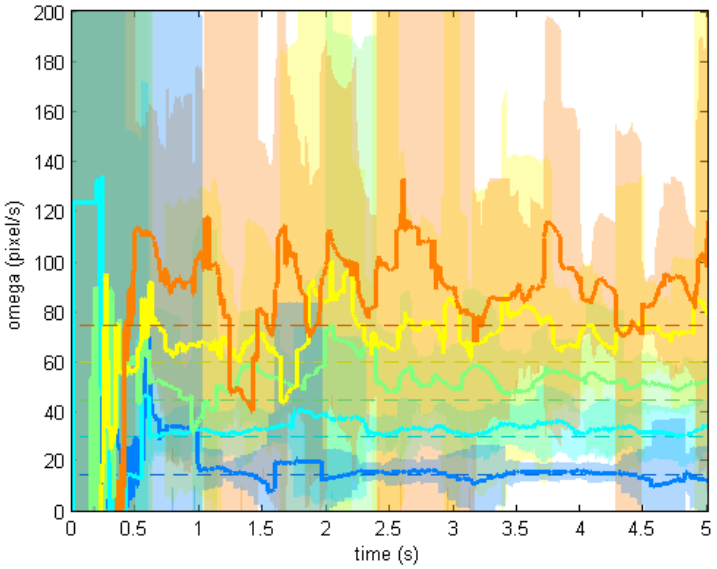


Figure 2.20: Speed response mean and standard deviation for the 64 pixels ($angle = 10\,\text{deg}$ $z = 1000m$ and $v_x = 20, 40, 60, 80, 100m/s$, $5s$ simulations). Dashed lines represent the optic flow ω predicted from geometrical calculation.

2.6 Landing Control

To test possible landing control strategies, we first decided to work with a geometrical estimate of the optic flow reconstructed from the spacecraft state, that is from the translational and rotational speed with respect to the ground. Afterwards, the most suited control was selected and used in simulation with the simulink model of the sensor.
The following sections will present different controllers that have been tested and implemented to fulfill the landing task.
The Problem was split into two parts:

1. Control of the attitude of the spacecraft to match a given setpoint.

2. Control of the altitude (1D) or the position (3D) of the spacecraft such that it lands smoothly on the planet.

Of course, those tasks are at least partially coupled, since the thrusters are all used for some translational and rotational control at the same time. Nevertheless, it is possible to split the problem, and then fuse the two outputs to get the final input that must be sent to the six thrusters.
Generally, the idea of this project is to use a bio-inspired controller for the landing task. Franceschini et al. (Franceschini et al., 2007) and Srinivasan et al. (Srinivasan et al., 2001) have given some insight on how insects control their flight behavior, namely that they use optic flow to do so. They have also shown that the control strategies can be successfully used to control small helicopters. Those control strategies have now been adapted to control the spacecraft.
The most important result is that insects want to keep the optic flow they perceive constant at a certain setpoint.
The optic flow an insect perceives is due to the fact that the image of the ground sweeps backward across the field of view of their compound eyes when the insect is flying forward. The optic flow can be defined as $\omega = \frac{groundspeed}{groundheight}$ if we assume that the only image considered here is the image of the ground. Any other object within the field of view of the insect will also result in an optic flow, but since we are interested in the landing behavior, we only consider the ground here. In this case, the optic flow is given by $\omega = \frac{speed\ in\ x\ direction}{altitude} = \frac{v_x}{z}$ if we assume the insect flies in the direction of the x-axis of our coordinate frame.
It is also important to note that insects do not measure absolute distances. Optic flow depends on the velocity divided by the distance, and therefore insects never know their current absolute altitude over ground.
Keeping the optic flow constant therefore couples the forward speed and the altitude. In order to perform a landing maneuver (meaning that the altitude z has to go to zero), the forward flight speed must be reduced. Switching action and reaction is this relation leads to the idea that gradually reducing the forward speed will automatically lead to a landing maneuver if a controller keeping ω constant is active.
Such a controller would adjust the lift force the insect produces such that its altitude changes to match a change in groundspeed (and therefore the change in optic flow).
This behavior ensures that a smooth landing can be achieved by simply gradually reducing

the forward flight speed. If the forward speed is reduced, the altitude will also automatically decrease, due to the control law. It also has the advantage that at the point where the altitude will reach very low values (nearly zero), the forward speed is very small. This connection allows the insect to land safely.
This basic control law of keeping the measured optic flow constant can be used to control the altitude of the spacecraft. The following sections describe how this was done for two different scenarios:

1. Vertical landing (straight down).

2. Grazing landing (with horizontal speed).

In the first case, the horizontal speed was set to zero and the spacecraft would therefore land by moving straight down. This is a simplified landing trajectory for landing on mars, where the real landing trajectory typically does not feature large horizontal speeds. Because of that, the environmental parameters of mars were used for this simulation. Since there is no horizontal speed anymore, the optic flow controller has to be adapted, but the general idea remains the same.
In the second case horizontal movement was introduced again, leading back to the case which was originally discussed in the previously cited papers. Such a grazing landing is usually performed when approaching moon, and therefore the environmental parameters of the moon were used for this simulation.

2.6.1 Vertical Landing

In this section, a simplified landing approach on mars is discussed, analyzed and simulated. The landing trajectory for this scenario is strictly vertical (1D).
The most important environmental factor on mars is the presence of an atmosphere. This will change the differential equations of the spacecraft and affect the whole system. The martian atmosphere was modeled according to the equations provided by NASA (cf. 2.3) The simulink model of the spacecraft was extended to include the atmospheric drag of the spacecraft for the simulation of the mars landing mission. This was simply done by adding another force term to each of the three translational degrees of freedom.
The new translational differential equations of motion for the spacecraft therefore are the following:

$$\begin{aligned} m\ddot{x} &= \cos\psi\cos\theta\cdot t_x + (\cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi)\cdot t_y \\ &\quad +(\cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi)\cdot t_z - D_x \qquad (2.21) \\ m\ddot{y} &= \sin\psi\cos\theta\cdot t_x + (\sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi)\cdot t_y \\ &\quad +(\sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi)\cdot t_z - D_y \qquad (2.22) \\ m\ddot{z} &= mg - \sin\theta\cdot t_x + \cos\theta\sin\phi\cdot t_y \\ &\quad +\cos\theta\cos\phi\cdot t_z - D_z \qquad (2.23) \end{aligned}$$

With:

$$D_x = \frac{1}{2} \cdot c_d \cdot A_f \cdot \rho \cdot \dot{x} \cdot |\dot{x}| \tag{2.24}$$

$$D_y = \frac{1}{2} \cdot c_d \cdot A_f \cdot \rho \cdot \dot{y} \cdot |\dot{y}| \tag{2.25}$$

$$D_z = \frac{1}{2} \cdot c_d \cdot A_f \cdot \rho \cdot \dot{z} \cdot |\dot{z}| \tag{2.26}$$

Also note that ρ depends on the altitude z. This introduces some cross-coupling to the differential equations.
The simulink model of the spacecraft as it was presented in section 2.2.1 has been adapted to include the atmospheric drag force. In figure 2.21 the three pink blocks have been added. They have the inputs $\dot{x}, \dot{y}$ and $\dot{z}$ respectively, as well as the altitude z, and have their output connected to the sum that represents $\ddot{x}, \ddot{y}$ and $\ddot{z}$

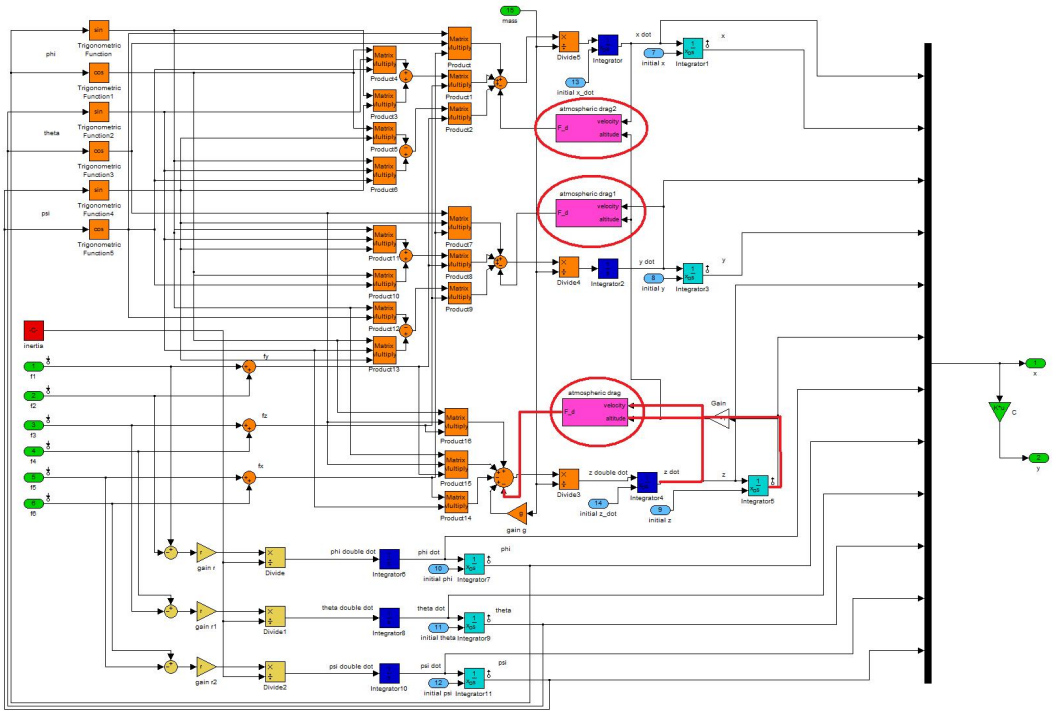


Figure 2.21: The simulink model of the spacecraft dynamics including the drag force

The atmospheric drag force is rather small, but still large enough to have quite a visible and measurable impact on the spacecraft behavior and therefore cannot be neglected.

Control

With the spacecraft model updated to the environment of mars, the main goal of the project, namely to land the spacecraft on the surface, can now be investigated and solved.

As mentioned above, the bio-inspired control approach has to be slightly modified from the models based on observations of insects (Franceschini et al., 2007; Srinivasan et al., 2001), because the spacecraft only moves downwards.
When the Spacecraft flies straight down, the flow field of an ideal facing down camera will appear as shown in figure 2.22.

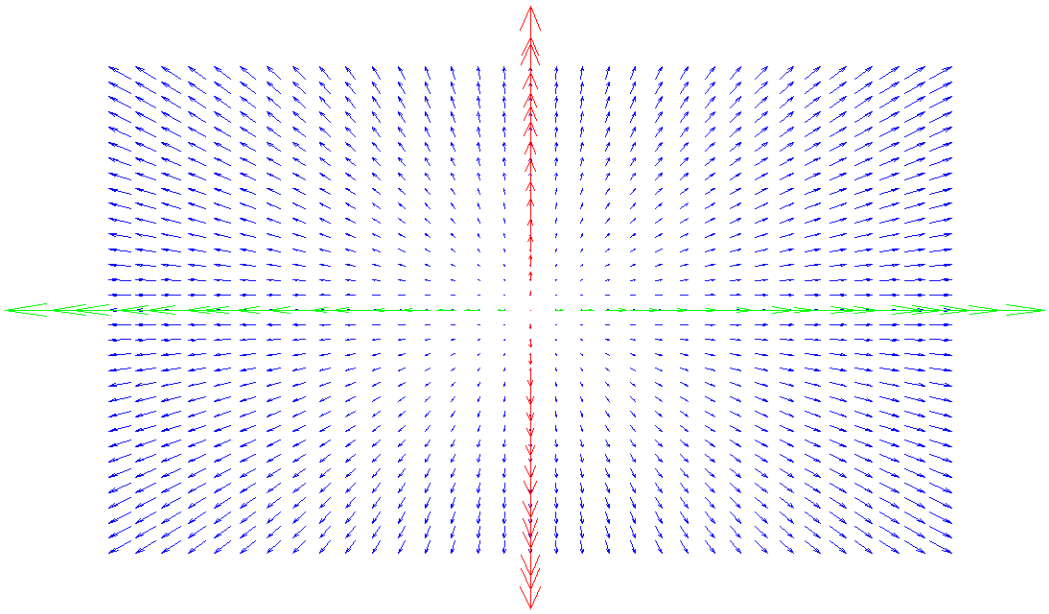


Figure 2.22: The (qualitative) sensor output for straight down fight

A downward looking down sensor inclined at an angle from the center of expansion of the optic flow, should theoretically be sufficient to get a meaningful measurement of the relative descent speed. The underlying assumption is that the terrain is perfectly flat.
The measurement obtained by this has the following form:

$$\omega = K \cdot \frac{\dot{z}}{z} \tag{2.27}$$

where K is a constant that depends on the number of pixels of the sensor and the sensor characteristics. In the simulated case presented later in this chapter, $K = 32$.

With this sensor measurement, the closed loop system will now look as shown in figure 2.23.
As one can see, this system only includes movement in z-direction. As mentioned before, for a vertical landing maneuver, all other degrees of freedom are assumed to be zero and remain at zero. The task for this problem is therefore to find a controller that can stabilize and land the spacecraft. To do so, the controller has to keep the measured ω constant at the given setpoint $\omega_{setpoint}$ by controlling the thruster force in z-direction, while only knowing the measured ω.

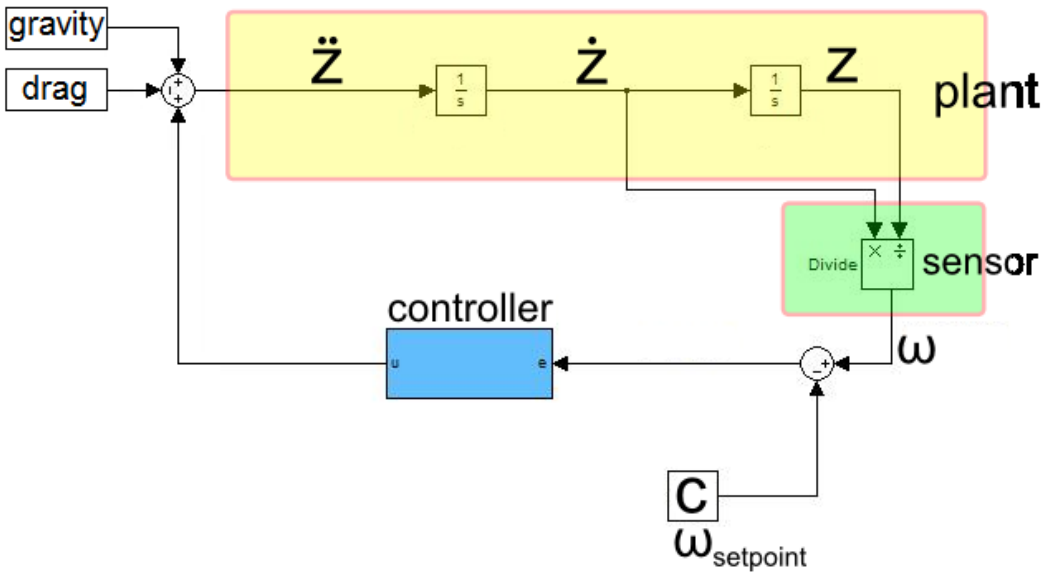


Figure 2.23: The closed loop System for vertical landing

If a nonzero setpoint for ω is chosen, the controller will always try to accelerate or decelerate the spacecraft such that the measured ω is equal to the setpoint. This means, that apart from possible zero crossings, the downward velocity will never be equal to zero. It will, most importantly, not stay equal to zero for more than one timestep.
If there is a nonzero downward velocity, and the controller must keep the measured ω at a constant setpoint, it can be seen from the formula that the altitude will decrease and therefore also the velocity itself has to decrease. This leads back to a smooth landing trajectory.
The reader might have noticed that there are infinitely many pairs of z and $\dot{z}$ which fulfill the condition that ω must be equal to $\omega_{setpoint}$.
This problem is solved by choosing a nonzero $\omega_{setpoint}$, which ensures that the spacecraft must always be moving. With that, only two possibilities remain: descending and slow down, or increase the altitude and accelerate. The sign of the controller now determines which way the spacecraft is going to move. For example, consider an altitude of 2km and a desired setpoint of $\omega_{setpoint} = 0.02$ (including the constant). This would therefore require a speed of magnitude $40\frac{m}{s}$ at this altitude to match the setpoint. If we assume zero speed, the measured ω will, according to the formula, be equal to zero, too. Therefore the controller will get an input of $(0 - \omega_{setpoint})$ and the sign of the controller then determines in what direction the spacecraft will start to move and therefore which of the two cases will be chosen.
Finding the appropriate control strategy is not a trivial problem, sections from 2.6.1 to 2.6.1 deal with this problem and present the simulated results for the different kinds of tested controllers.
The resulting plant for the system, as seen in figure 2.23, can easily be identified as a simple double integrator. $\ddot{z}$ can be manipulated directly by the controller, and the output is given

by the already known formula $\omega = \frac{\dot{z}}{z}$.
This system can be represented by:

$$x_1 = z \tag{2.28}$$
$$x_2 = \dot{z} \tag{2.29}$$

and therefore:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} x_2 \\ \ddot{z} \end{bmatrix} \tag{2.30}$$
$$y = \frac{x_2}{x_1} \tag{2.31}$$

The differential equations of the double integrator are linear and well studied (O'Brien et al. (2003), Rao and Bernstein (2001)). The Problem here is the nonlinearity on the output y, which makes the whole controller design a lot more complicated. Systems with this structure are called 'Wiener Systems'. Again, those systems have been well studied in the past (Neäsiãc (1999), AL-Sunni and Shafiq (2003), Nesic (2000), Nesic (1997), Wang and Wang (2006)).
Unfortunately, the literature on control of Wiener typed systems usually only consider Wiener systems of a certain type, for example nonlinearities of a square type, or invertible nonlinearities etc.
In this case, however, the nonlinearity is a fraction of two states as seen in the equation above. It is impossible to estimate both states x_1 and x_2 by only measuring y and not having any additional information.
Within the time frame of this project, we could not find a controller that was able deal with such a fraction of states.
A future goal might be to find a specific Wiener-systems controller that can deal with the specific nonlinearity present in this system.
For the present project general (nonlinear) control theory was used in order to find a controller. There are several options, which will be presented on the following pages.

Linearization The easiest option to find a controller is to linearize the system and use linear control theory. It is straightforward to see the A and B matrices, since the double integrator system is already a linear system. Linearizing the system's output causes some problems though.
First, the system is supposed to always be in motion and the equilibrium at $z = 0$ and $\dot{z} = 0$ is never fully reached because of the singularity of ω at that point.
Therefore, the system cannot be linearized around that point. The question of how to choose the setpoints for z and $\dot{z}$ thus remains open. Different choices especially for z have a huge impact, since the linearized C matrix looks as follows:

$$C = \begin{bmatrix} -\frac{x_2}{x_1^2} & \frac{1}{x_1} \end{bmatrix} \tag{2.32}$$

With the *square* of x_1 appearing in the denominator.
Different setpoints have been tested, for example $(40\frac{m}{s}, 2000m)$ or $(0.2\frac{m}{s}, 10m)$, because the first pair represents the initial condition, and the second pair represents the simulation end condition at 10 meters above ground.
Unfortunately, neither of those linearized systems was useful to design a controller. In fact, a controller that failed on the linearized system worked well on the nonlinear simulink model. The controller used in the end (see the end of this section) for example works very well for the second pair of linearization setpoints, but fails for the first pair (figure 2.24).

Nonlinear Model Predictive Control (MPC) One of the most powerful tools for nonlinear control is nonlinear model predictive control (Findeisen and Allgöwer (2002), Patwardhan et al. (1997), Findeisen et al. (2003), Jeong et al. (2001)). However, it can not be used to begin with, because the main principle of MPC is to minimize a cost function that penalizes the states and the controller action. In this project though, the guideline to build a bio-inspired controller prevents the knowledge of the states. Only ω is known from the sensor, while z and $\dot{z}$ are unknown.
In addition, it is not possible to build an observer to estimate the states, because ω is the fraction of two states and therefore there is no way to calculate back and estimate both states without any additional information.

The problem could be solved easily by adding any kind of altitude sensor or any similar sensor providing additional information. This however is explicitly not the goal of this project, and therefore NMPC cannot be used.

Output Feedback Linearization Output feedback linearization is a tool to transform a nonlinear system into a new form, in which the nonlinearities disappear and the resulting equivalent linear system can be controlled using linear control theory (Khalil, 2002).

The Approach is applied easily to the system and the new system will have the state:

$$a_1 = \frac{x_2}{x_1} = \frac{\dot{z}}{z} (= \omega) \tag{2.33}$$

and therefore:

$$\dot{a_1} = -\frac{\dot{z}^2}{z^2} + \frac{u}{z} := v \tag{2.34}$$

with v being the new system input and the system output simply given by:

$$y = a_1 = \omega \tag{2.35}$$

The new system is linear and has the desired output ω.
One can now close the loop and build a controller that takes y as its input and v as its output.
While this result looks promising so far, unfortunately there are some problems again: in

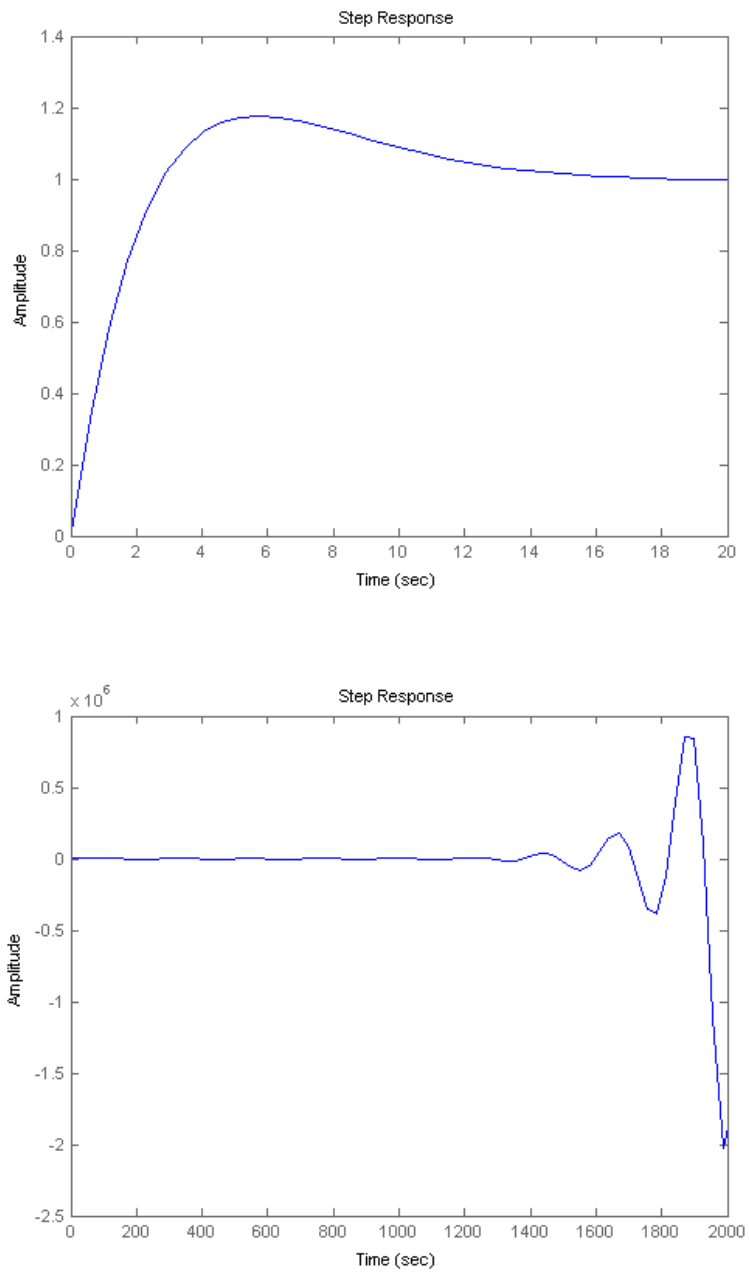


Figure 2.24: Closed loop step response of the two linearized systems and the final PID controller for the linearization setpoints ($\dot{z} = 0.2\frac{m}{s}; z = 10m$) at the top and ($\dot{z} = 40\frac{m}{s}; z = 2000m$) at the bottom.

order to calculate the input u to the original system from the new input v, the following equation has to be used:

$$u = (\omega^2 + v) \cdot z \tag{2.36}$$

As can be seen, the current altitude z needs to be known in order to calculate back from v to u. As mentioned in the NMPC section, this is not the case in this project though, and therefore output feedback linearization cannot be used either.

Sliding Mode Control

The so-called sliding mode control is a control strategy that uses a switching function to switch between two modes of a controller. Hebisch et al. (1995) (in German) presents the basics of this control strategy. The most important result is that sliding mode control is extremely robust. Of course, that is a very nice property, and especially helpful since we only consider a very simple spacecraft model in this project.
Also, sliding-mode has proven to yield very good results in nonlinear control of a quadrotor - a system that is similar to the spacecraft.
The main problem when designing a switching mode controller is always to find the switching function. Usually, the switching function will have the form $S(x) = ax_1 + x_2$, under the condition that $\dot{x_1} = -ax_1$ is stable. Of course, for the system considered here, this is not possible, since once again we do not know x_1 and x_2 and therefore the switching function S cannot be chosen according to this rule.
For the vertical landing system, the following alternative switching function has been chosen:

$$S(x) = \frac{x_2}{x_1} - \omega_{setpoint} \tag{2.37}$$

Note that this function is basically the error of the measured ω from the setpoint $\omega_{setpoint}$.
With this switching function, one can define a Lyapunov function $V = \frac{1}{2}S^2$, due to the square is guaranteed to be positive semidefinite.
With this selection

$$\dot{V} = S\dot{S} = \frac{ux_2}{x_1^2} - \frac{x_2^3}{x_1^3} - \frac{u\omega_{setpoint}}{x_1} + \frac{x_2^2\omega_{setpoint}}{x_1^2} \tag{2.38}$$

and with

$$u = -\beta \cdot sign(S) \tag{2.39}$$

$$\dot{V} = -\frac{\beta sign(S)x_2}{x_1^2} - \frac{x_2^3}{x_1^3} + \frac{\beta sign(S)\omega_{setpoint}}{x_1} + \frac{x_2^2\omega_{setpoint}}{x_1^2} \tag{2.40}$$

Three cases can be distinguished for this function:

- If $S = 0$ then $\omega = \omega_{setpoint}$ and therefore $\dot{V} = 0$.

- If $S > 0$ then $\omega > \omega_{setpoint}$ and by inserting it can be seen that $-\frac{\beta\omega}{x_1} + \frac{\beta\omega_{setpoint}}{x_1}$ is negative overall, and $-\omega^3 + \omega^2\omega_{setpoint}$ is negative too. Therefore, $\dot{V}$ is smaller than zero.

- If on the other hand $S < 0$ then $\omega < \omega_{setpoint}$. Inserting again yields $\frac{\beta\omega}{x_1} + -\frac{\beta\omega_{setpoint}}{x_1}$ is negative, but $-\omega^3 + \omega^2\omega_{setpoint}$ is positive.
  Rearranging the terms yields the condition that $\beta > \frac{x_2^2}{x_1}$ in order to ensure that $\dot{V}$ is negative.

This allows two conclusions:
1) sliding mode control with the chosen switching function basically leads to a simple proportional controller which feeds back the weighted negative value of the error $(\omega - \omega_{setpoint})$
2) To guarantee Lyapunov stability, additional constraints on the states x_1 and x_2 are needed, but given that the thrusters are powerful enough, it should always be possible to find a value β large enough to satisfy the condition.

Overall, because the states are unknown, the usual sliding mode approach cannot be used for the landing task, but using a different switching function has revealed some interesting results.

Simple Transformation Another idea was to simply transform the system output.
The original output $\omega = \frac{\dot{z}}{z}$ can be rewritten as:

$$y_{new} = x_2 - \omega_{setpoint} \cdot x_1 \tag{2.41}$$

With this new formulation, if the measured ω is close to the desired $\omega_{setpoint}$, the new output y_{new} will be close to zero.
Therefore, the controller for this system has to keep y_{new} equal to zero in order to keep the system output ω at the setpoint $\omega_{setpoint}$.
In theory, this very simple transformation solves the problem and yields a linear system that can now be controlled using linear control theory.
The problem though is, that in reality we really just have the sensor measuring ω. The output can therefore not be remodeled as proposed and therefore, this idea again does not help to solve the overall problem.

Final Solution and Simulation Results

Because the nonlinear approaches did not work, and because of the result shown in the sliding mode section, the system is now to be controlled by a simple PID controller, which is tuned to match the needs for this system.

The final, closed loop system including the PID controller can be seen in figure 2.25.
The output of the controller, which is the desired thruster force, is manipulated before actually being sent to the spacecraft. First, there is a lowpass filter, because the thruster cannot adjust the force it generates infinitely fast. Later in this section, when noise is introduced to the system, very fast changes in the controller output will appear. Those fast changes have to be filtered out in order to guarantee that the real thruster can provide the desired force.

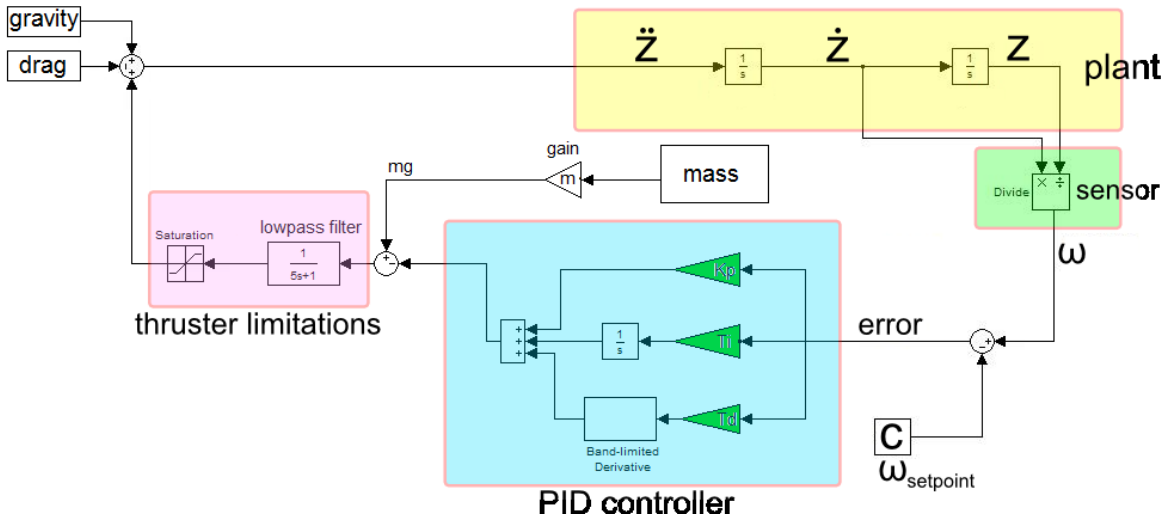


Figure 2.25: Closed loop system for straight down landing

After the lowpass filter, there is a saturation, which limits the thrust to be smaller than zero. Recalling the definition of the coordinate frames this means, that the thruster cannot be used to accelerate downwards, but only to break. The acceleration down towards the planet is only due to gravity, which makes sense in terms of fuel economy - we only want to use fuel if we really have to.
Lastly, there is another addition: right after the controller, the control signal is subtracted from a value equal to the current gravity force $m \cdot g$.
This means, that if the controller output is zero, the thrusters will still counter the gravity force and therefore keep the spacecraft at the current altitude. The controller is then used to add or subtract from this counter-gravity-force and by that, move the spacecraft.
This special arrangement is not necessary, it is possible to close the loop by directly using the controller output without any addition of the gravity force. The problem in that case is that at the beginning, the thruster output is zero, and therefore during the first timestep only the gravitational force acts on the spacecraft. Only after the first timestep in the simulation, the controller has a non-zero output. This can cause problems, if for example the acceleration due to gravity accelerates the spacecraft so much during the first timestep, that the controller is unable to slow the spacecraft down before it hits the planet. In order to avoid this problem, the addition of the gravity force has been implemented.
The resulting simulink model for this section is depicted in figure 2.26.

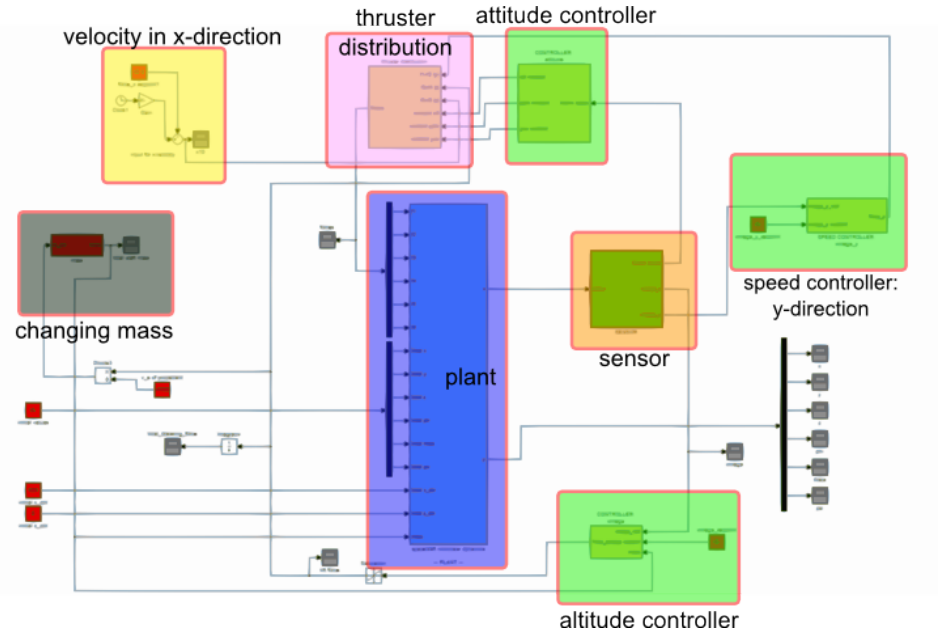


Figure 2.26: Simulink model for the landing controller with given x-velocity

In a first step, the PID parameters k_p, T_i and T_d have been determined using the Matlab Optimization Toolbox, which was set to choose those three parameters such that the altitude z would stay below a curve. In fact, the curve was a linear slope from 2000m at time 0s to 10m at time 200s, and after that, the altitude had to constantly stay below 10m for the rest of the simulation.
The parameters resulting from this optimization have then been tuned by hand to enhance the performance and the final values are:

$K_p = 20$
$T_i = 0.4$
$T_d = 0.02$

Using the linearized system introduced at the beginning of this section, and using the linearization setpoint of $z_0 = 10m$ and $\dot{z}_0 = 0.02\frac{m}{s}$, the following results are obtained:

The closed loop step response (as seen before) is shown in figure 2.27. The system has a

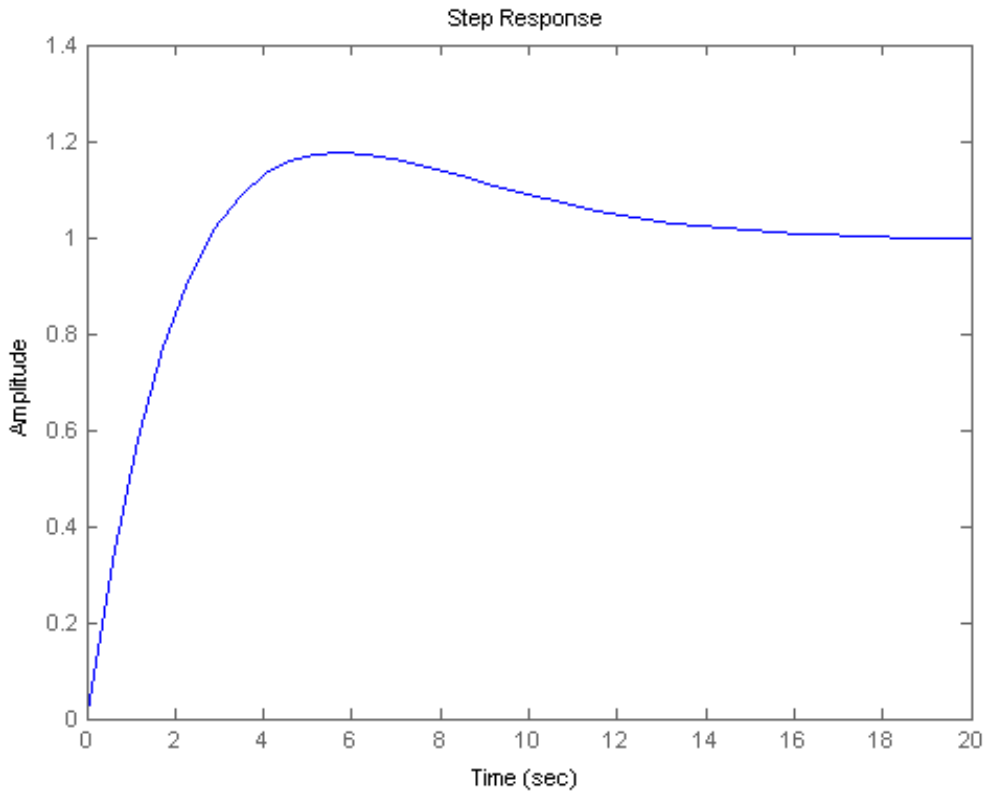
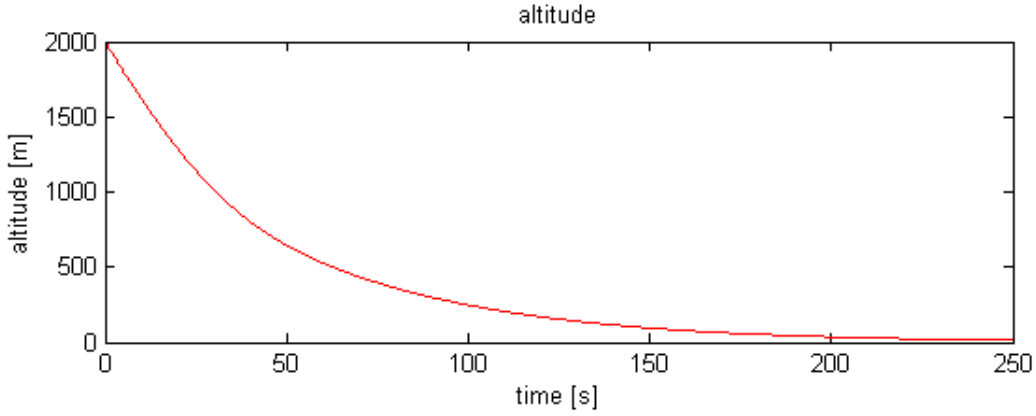
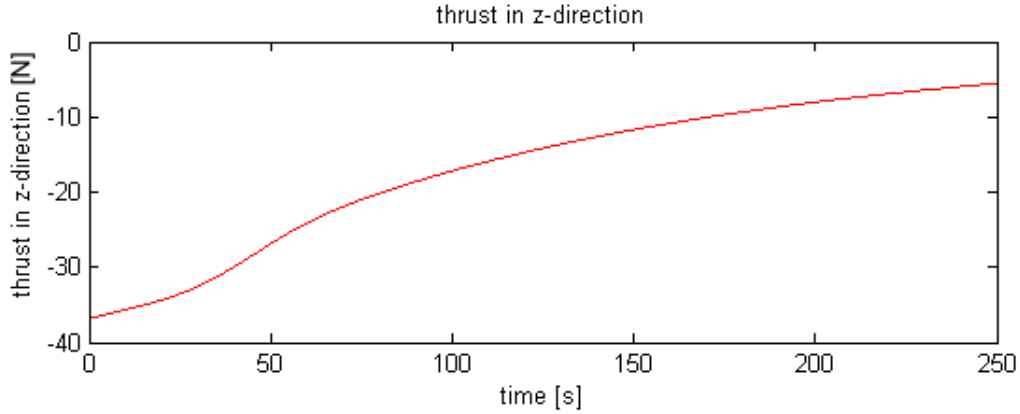
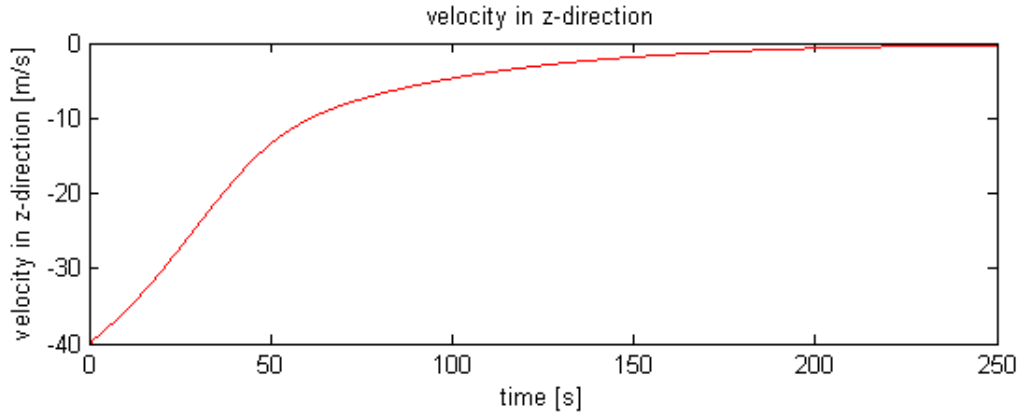


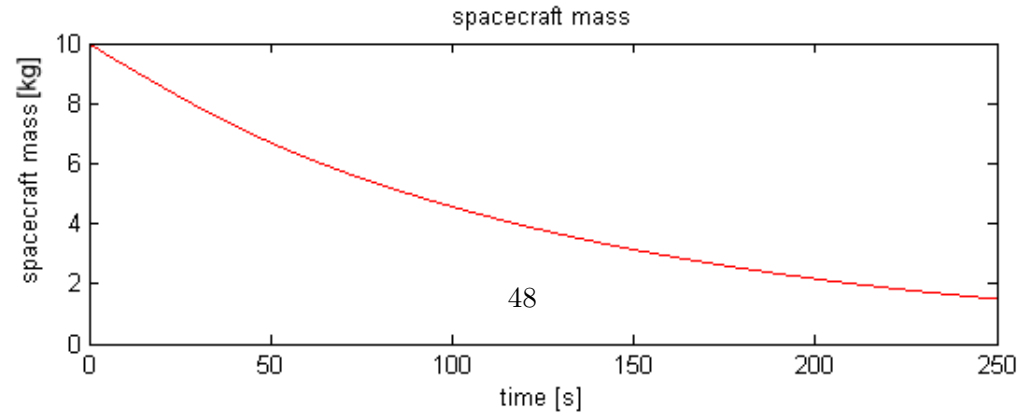
Figure 2.27: Closed loop Step response using PID control

phase margin of 71.6690 degrees, which would indicate a high robustness. The poles of the linearized system including drag both have a negative real part - the linearized system is thus stable.
As mentioned before though, the results for the linear system cannot be trusted, since the results depend largely on the linearization setpoints. It is therefore a lot more interesting to look at the nonlinear simulink model and investigate how this system reacts and behaves.
The following results (figure 2.28 and 2.29) are obtained by applying the previously introduced PID controller to the system using the initial conditions $z = 2000m$ and $\dot{z} = -40\frac{m}{s}$. For this example, $\omega_{setpoint}$ was chosen to be equal to 0.64, which corresponds to the sensor output at the given initial conditions.
The top plot shows the altitude of the spacecraft. As specified before, it starts at 2000m and drops down close to the surface. The next plot shows the thruster force that is needed for this maneuver. As explained before, it starts at the gravity compensating value of 36.9N. The third plot shows the velocity in z-direction. Again, it starts at the specified $-40\frac{m}{s}$. Since $\omega_{setpoint}$ has been chosen such that the condition is fulfilled right from the beginning, the spacecraft never accelerates downward, but only slows down. The final plot shows the change in mass due to the fuel consumption needed to break the spacecraft.
Considering the nice shape of both the altitude and the velocity curve, this very simple controller is capable of landing the spacecraft in a smooth way. At an altitude of about 10m the spacecraft is nearly standstill.
Figure 2.29 shows the measured ω over time in red, and the setpoint in black.


altitude
altitude [m]
2000
1500
1000
500
0
0
50
100
150
200
250
time [s]


thrust in z-direction
thrust in z-direction [N]
0
-10
-20
-30
-40
0
50
100
150
200
250
time [s]


velocity in z-direction
velocity in z-direction [m/s]
0
-10
-20
-30
-40
0
50
100
150
200
250
time [s]


spacecraft mass
spacecraft mass [kg]
10
8
6
4
2
0
0
50
100
150
200
250
time [s]

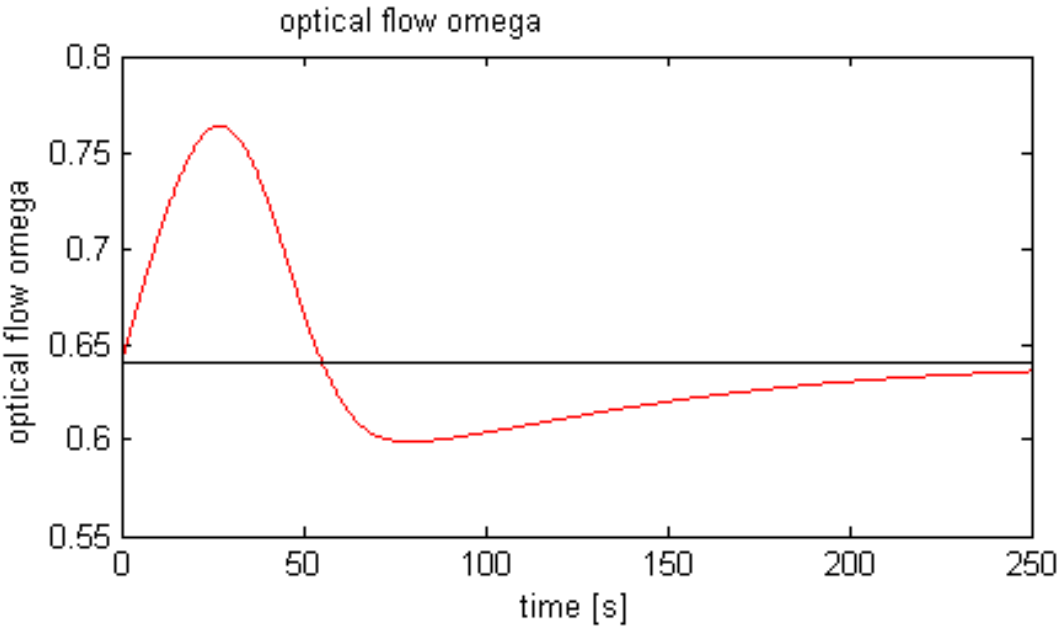


Figure 2.29: red: measured ω; black: $\omega_{setpoint}$

Choosing $\omega_{setpoint}$ In the previous example, $\omega_{setpoint}$ was chosen to match the sensor output for the initial conditions. Of course, other choices are possible. Previously in this text, the condition that $\omega_{setpoint}$ must not be smaller than the value given by the initial conditions was set. It thus remains to either choose the setpoint equal to that value, or larger.
Figures 2.30 and 2.31 show the same plots as before, but now $\omega_{setpoint}$ has been chosen to be equal to 1.5 and thus larger than the value corresponding to the initial conditions.
The altitude and velocity result is nearly the same, still resulting in a smooth landing approach. The difference is, that now the spacecraft actually accelerates at the beginning, getting faster than the initial -40$\frac{m}{s}$. Also, the initial thrust does not fully counter the gravity anymore. This is expected, because otherwise the spacecraft would not accelerate. It also makes sense, that with this setup, the spacecraft reaches the desired final altitude of about 10m faster than before.
$\omega_{setpoint}$ can be used for a numerical optimization (e.g. to minimize fuel consumption). Other limitations, such as velocity acceleration constraints might also be relevant to choose the setpoint.

Introducing Disturbances: Noise Until now, the system was assumed to be perfect, without any measurement uncertainties or similar problems. In reality, this will of course not be the case. It is therefore important to know how the closed loop system reacts to disturbances. The most likely signal to be perturbed by uncertainties is the measured ω because the sensor is not able to perfectly measure ω. To see how such measurement noise affects the system, we added white noise to the sensor output (figure 2.32). For this simulation, $\omega_{setpoint}$ was set back to 0.64 again.
The top plot shows the measured ω in its original shape. Note the quite large changes in ω at the end, between 180 and 200 seconds. At first, it looks as if the system would start to get unstable, but that is not the case. The behavior can instead be explained because between 180 and 200 seconds the spacecraft was already very close to the ground, $\omega = \frac{\dot{z}}{z}$ approached the singularity at $z = 0$ which explains why the introduced noise has such a strong effect and on the measured ω. The middle plot shows the white noise added to the signal. Notice that the magnitude of the noise is quite large compared to the actual signal value! While the signal has a maximum value of about 1.1 and minimum value of about 0.4, the noise magnitude mostly varies between +0.5 and -0.5 with peaks up to +1 and -1. The resulting signal-to-noise ratio for this configuration is $SNR = 17.898$ dB, which is a rather low value as expected due to the large noise. The bottom plot shows the new ω that is now perturbed by noise. This imperfect signal is then used and fed forward into the control loop.
The simulation results for this noisy ω are shown in figure 2.33
The plots show that the closed loop results are still very similar to the perfect case. The control loop therefore seems to be reasonably robust against measurement disturbances. With this result, the chosen PID controller seems to be a reasonable choice, and it has been shown to fulfill the task of smooth landing.

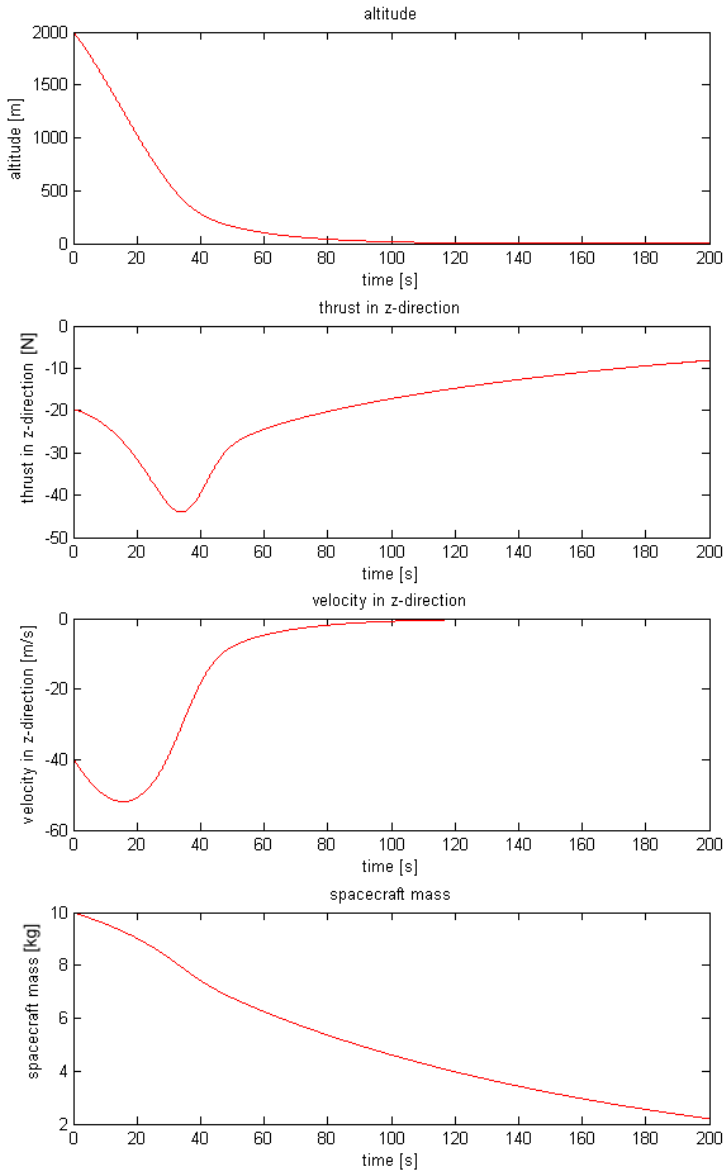


Figure 2.30: Simulation results for $\omega_{setpoint}$=1.5

2.6.2 Grazing Landing

As a second problem was investigated, landing on moon. With respect to the previous section, the main differences are:

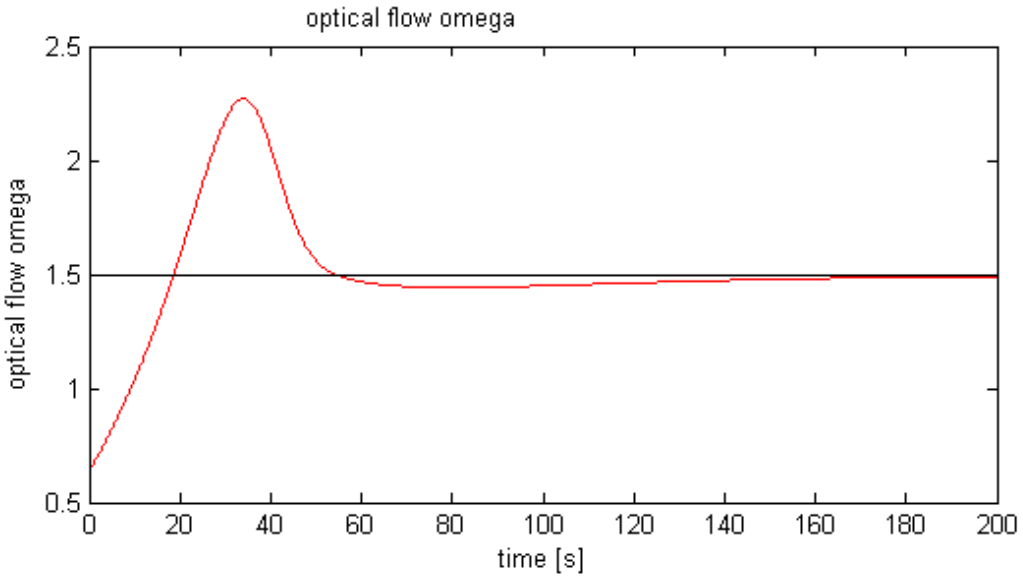


Figure 2.31: ω for $\omega_{setpoint}$=1.5

1. The absence of an atmosphere.
2. A grazing landing approach: horizontal speed is allowed and in some cases even necessary.

Not having an atmosphere allows us to use the differential equations derived in section 2.2.1 with lunar gravity, which is approximately $1.627\frac{m}{s^2}$. One can estimate the optic flow according to the definition:

$$\omega_x = \frac{v_x}{z} = \frac{\dot{x}}{z} \tag{2.42}$$

$$\omega_y = \frac{v_y}{z} = \frac{\dot{y}}{z} \tag{2.43}$$

$$\omega_z = \frac{v_z}{z} = \frac{\dot{z}}{z} \tag{2.44}$$

Prerequisites and Assumptions

For the landing control on the moon, we integrated the information from sensors distributed over the whole spacecraft and estimated:

- optic flow $\omega_x = \frac{v_x}{z}$, $\omega_y = \frac{v_y}{z}$ and $\omega_z = \frac{v_z}{z}$
- angular speeds $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$

It has been shown by Dahmen et al. (2001) that these are realistic assumptions. Note that in this section, all six degrees of freedom are controlled, even though the problem can be reduced to 2DOF (x-z).

Control

This section discusses and presents a controller structure to control the attitude of the spacecraft, and two approaches to reach the goal of landing the spacecraft on the moon.
The first approach to land is a PID controller using the same structure as for the straight down landing. The implementation corresponds to the original idea by Franceschini (Franceschini et al., 2007). In this approach, the velocity in x-direction is set externally to perform a braking maneuver. The controller is used to keep ω_x at the given setpoint, and therefore lands the spacecraft for decreasing values of v_x.
For the second approach, this idea has been expanded. The external influence of a given breaking maneuver in x-direction was removed, and the whole landing procedure was performed only using the optic flow measurements. To do so, the vertical and horizontal components of the optic flow are extracted and used perform a smooth landing maneuver.
Although in both cases only movements in x-direction and in z-direction are needed, all the degrees of freedom are controlled. This allows us to demonstrate how the outputs of the attitude and translation controllers can be combined to generate thruster commands.
The problem was therefore split into attitude control and translation control. Translation control includes the movement in x-direction, y-direction and z-direction. Attitude control controls the three rotational degrees of freedom. This leads to the problem of combining those controller requests in the end, because obviously the force in one direction is always coupled with the torque around one axis.
The problem has been solved as follows:

1. Six separate controllers determine the necessary forces in x-,y- and z-directions as well as the roll, pitch and yaw moments which are needed to perform the intended maneuver. The forces in x-, y- and z-direction are determined by the landing controllers, while the moments are determined by the attitude controllers.

2. An s-function block takes those six requests as its input and fuses all those requested forces and moments to determine the six individual thruster forces needed to meet the requests.

Fusing the controller outputs While the next few sections show more precisely how the different controller requests are generated, we first have to investigate how those requests are fused in order to obtain the commands for the six thrusters.
The thruster distribution block has six inputs, which are the requested forces in x-, y- and z-direction and the requested roll, pitch and yaw moments. The output is a vector containing the six individual thruster forces.
Inside the block, the corresponding force and moment pairs are each fed into the s-function, which then determines the thruster forces needed.
According to figure 2.2, the force in y-direction and the rolling moment are generated by

thrusters 1 and 2, the force in z-direction and the pitching moment are generated by thrusters 3 and 4 and the force in x-direction and the yaw moment are generated by thrusters 5 and 6. The s-function, which is used three times in the distribution block, therefore has two inputs, namely a force and a moment and two outputs, which are the two individual thruster forces. First, it is checked if the requested input force is zero. If so, the thrusters must only provide a moment. Therefore the requested moment is created by letting one thruster force be positive, the other negative. Both forces must be equally large, in order not to create any force along the corresponding axis.
The thruster forces can then be computed according to:

$$M_{requested} = f_+ \cdot r - f_- \cdot r \tag{2.45}$$

and the condition that $f_+ = f_-$
If the requested force is not equal to zero, both thrusters will need to generate a force pointing in the same direction, but one of the two is larger than the other, such that the requested moment is generated, too.
If the force is positive, the following formula distributes the forces to the two thrusters:

$$f_- = \frac{f_{tot}}{2} - \frac{M}{2r} \tag{2.46}$$

$$f_+ = \frac{f_{tot}}{2} + \frac{M}{2r} \tag{2.47}$$

If the force is negative the two values have to be interchanged.
With this function, a given requested force and a given requested moment can be converted into the thruster forces of the two corresponding thrusters.

Attitude Control Since the optic flow can be used to estimate the angular speeds, those values can be used to at least partially control the attitude of the spacecraft.
To control the attitude, a smaller system can be used, that consists of only six states:

$$\begin{aligned} x_1 &= \phi \\ x_2 &= \dot{\phi} \\ x_3 &= \theta \\ x_4 &= \dot{\theta} \\ x_5 &= \psi \\ x_6 &= \dot{\psi} \end{aligned} \tag{2.48}$$

As previously mentioned, only three of those (x_2, x_4 and x_6) are known, however. Unfortunately, this does not yield a fully observable system. Therefore, it is not possible to estimate the complete state vector of the system. It is thus also not possible to use optimal control or any similar control strategy, which needs access to all the states, or at least estimates of all the states. And it is generally not possible to control ϕ, θ and ψ to be equal to zero, but only possible to control the derivatives, using a simple PI controller for each of them.
Doing so will certainly help to keep the spacecraft at the desired attitude, but once there is

an error in one of the angles, this control scheme does not have any means to drive those states back to zero. Any error in the attitude is therefore permanent, and we can only try to prevent further errors.
In reality, similar to the mars landing case, additional sensors would be needed to fully control the attitude of the spacecraft. However, with the additional information in this case, it is possible to at least control the derivatives to zero.
Simulations have shown that this works well. Disturbances in $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$ can be countered, while the position controllers are active at the same time. Figure 2.34 shows a plot of a typical landing maneuver. The bottom subgraph shows that an initial angular velocity in the pitch angle of $\frac{\frac{\pi}{360}}{sec} = \frac{0.5°}{sec}$ is correctly controlled back to zero, while the top three graphs show the landing maneuver in z-direction, the breaking maneuver in x-direction and keeping the velocity in y-direction at zero.

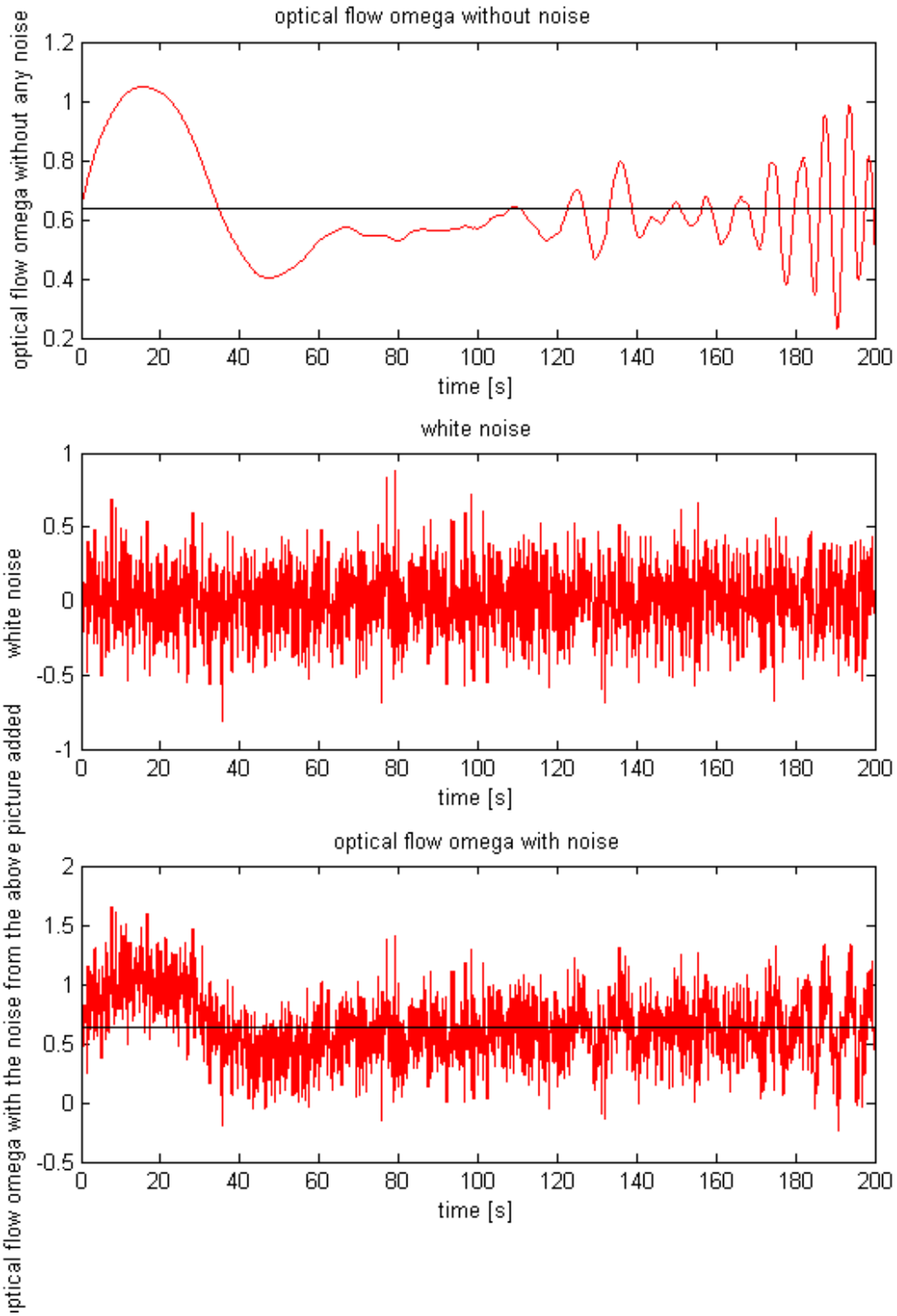


Figure 2.32: ω with additional white noise

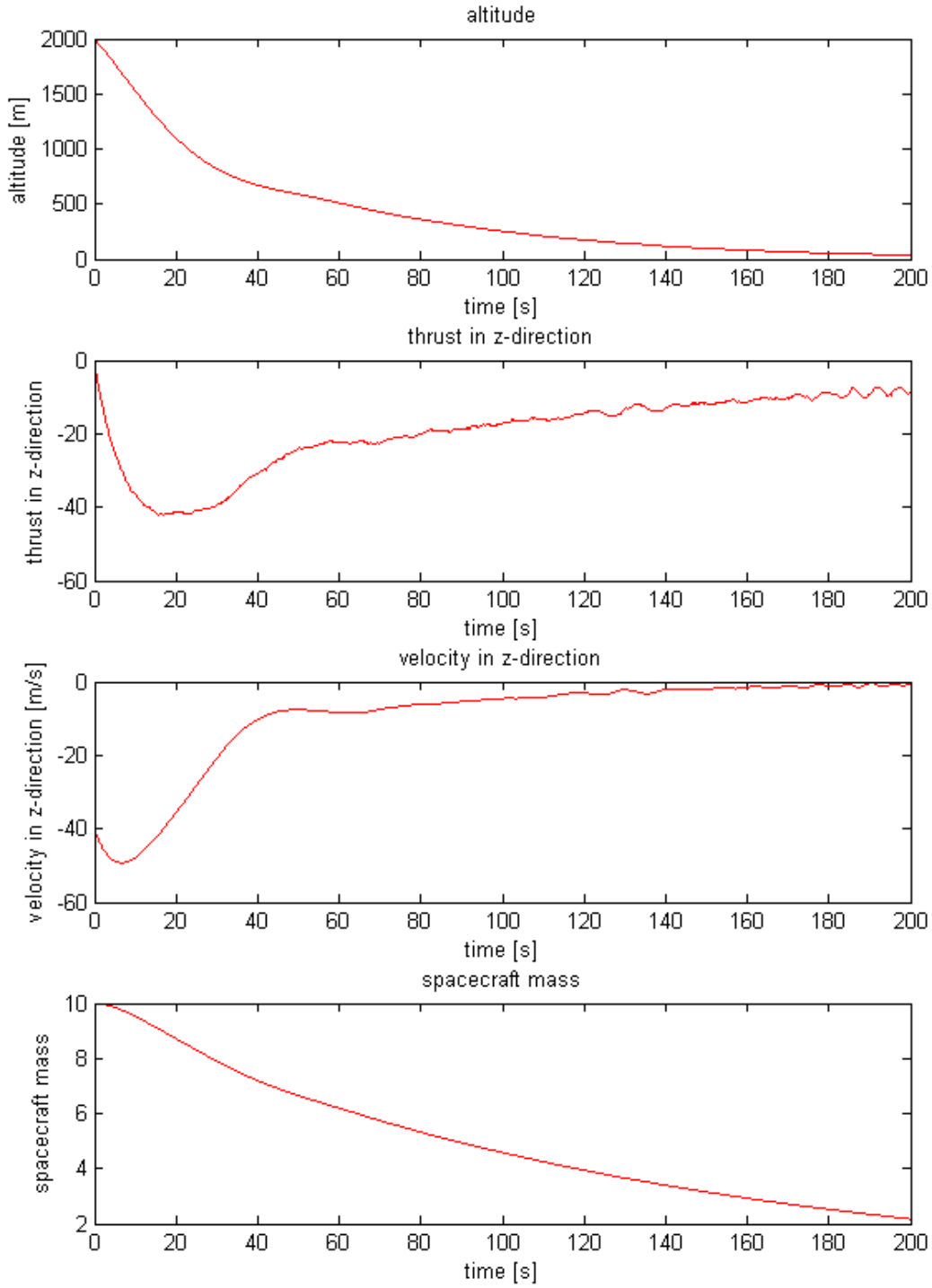


Figure 2.33: Results for the closed loop system with additional white noise

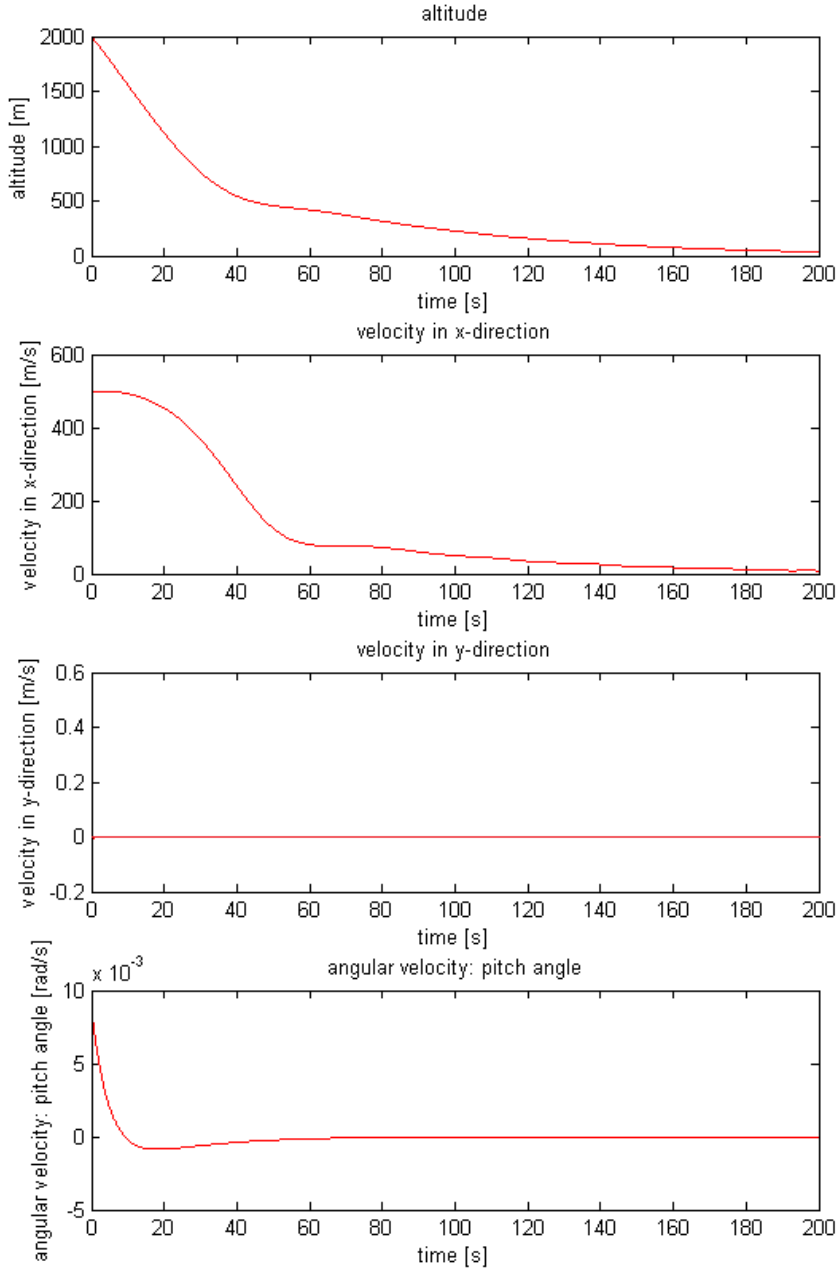


Figure 2.34: Example of a simulation with angular velocity control

Landing Control In this section, the previously introduced two approaches to land the spacecraft are presented and analyzed. Both of them have the same objective, which is to bring the spacecraft close to the ground, and break the spacecraft's horizontal velocity close to zero. In both approaches, the attitude controller is active, and keeps the angular velocities at zero, and both approaches use a simple ω_y controller to control the velocity in y-direction. This y-controller works similar to the other optic flow controllers used in the two approaches. The input to the controller is $\omega_y = \frac{v_y}{z}$ and since the velocity should be kept at zero, the setpoint for this controller is zero. A PID controller is used, such that any deviation form zero of ω_y is countered and the velocity in y-direction goes to zero again.

PD Landing Control with given velocity in x-direction
This landing procedure has two key elements:

1. The measured optic flow $\omega_x = \frac{v_x}{z}$ must be held constant at a setpoint $\omega_{setpoint,x}$.
2. In order to land the spacecraft, the horizontal speed v_x must be reduced.

With condition 1), this will lead to a reduction of the altitude and therefore land the spacecraft. The controller schematic is shown in figure 2.35.

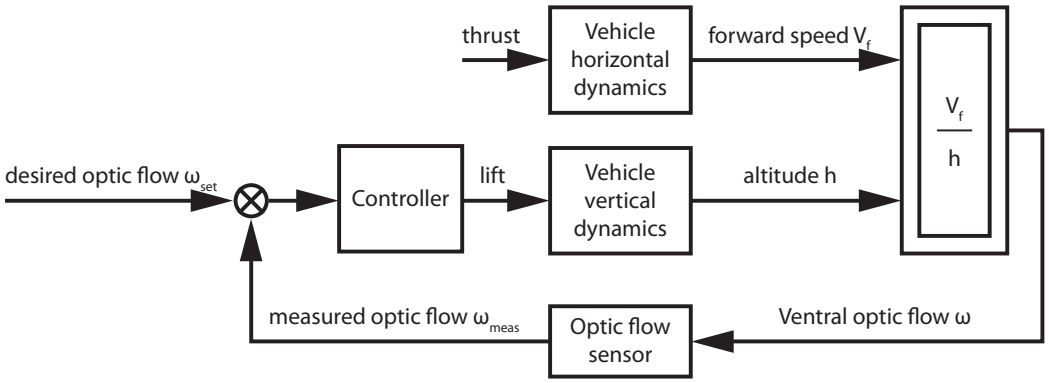


Figure 2.35: Schematic of the controller with active braking.

We first needed to control the velocity in x-direction. The spacecraft starts with a horizontal speed of $500\frac{m}{s}$. The speed is then reduced to nearly zero within a given time frame for the landing maneuver. The mission ends before the speed reaches zero, because at that point the altitude would be zero too, and this would lead to the well known singularity in ω.
For the simulation, a time frame of 200 seconds was chosen. Figure 2.36 shows the velocity in x-direction during those 200 seconds.
To ensure a smooth landing, the breaking force in x-direction was chosen to be a linear function of time, such that the velocity in x-direction decreases exponentially. However, to be able to do this one needs to keep track of the absolute speed, or at least one needs to precisely

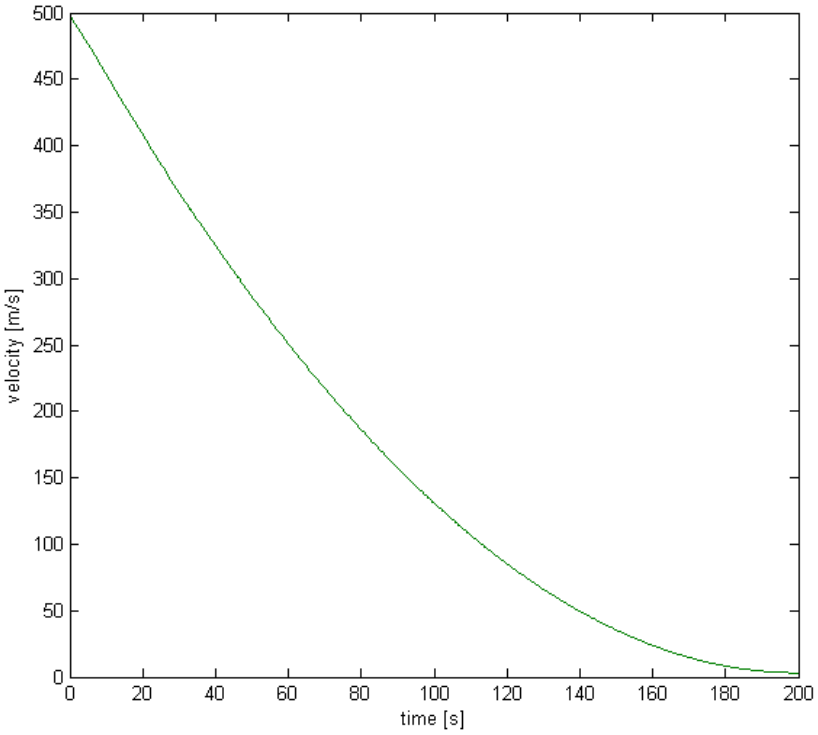


Figure 2.36: Velocity in x-direction

know its initial conditions. Notice that the final value is around $2\frac{m}{s}$ and therefore does not hit the singularity at $0\frac{m}{s}$.
Such a linear change in breaking force should be realizable on a real spacecraft, but it would require additional sensors. A constant braking force is not is not an applicable solution, because it would lead to a constant vertical speed landing trajectory, without any braking.
The actual landing controller works similarly to the one used for mars landing.
For the reasons explained in section 2.6.1, a P[I]D controller is used again for this task. An integrator is not necessarily needed, therefore in the current version, only a PD controller was used.
Unlike before, in this case the assumption is that the absolute speed is known, so it would be possible to use the measured optic flow ω and the known velocity in x-direction to explicitly estimate the current altitude z. This was done as an exercise and a MPC controller was implemented (not presented here, please refer to: Ammann, 2009). Since the original task of this project was to use a bio-inspired control approach, we decided to focus on a controller which does not estimate the altitude. $K_p = 500000$ and $T_d = 100000$ were found to yield a satisfying control result. It is again assumed, that the spacecraft can only break, such that any downward acceleration is only due to gravity. $\omega_{setpoint}$ was chosen to fit the initial conditions, which are $\dot{x} = 500\frac{m}{s}$ and $z = 2000m$, and therefore $\omega_{setpoint,x} = \frac{500}{2000} = 0.25$.

With the PD controller introduced above and this setpoint, the following result was obtained (fig.2.37).

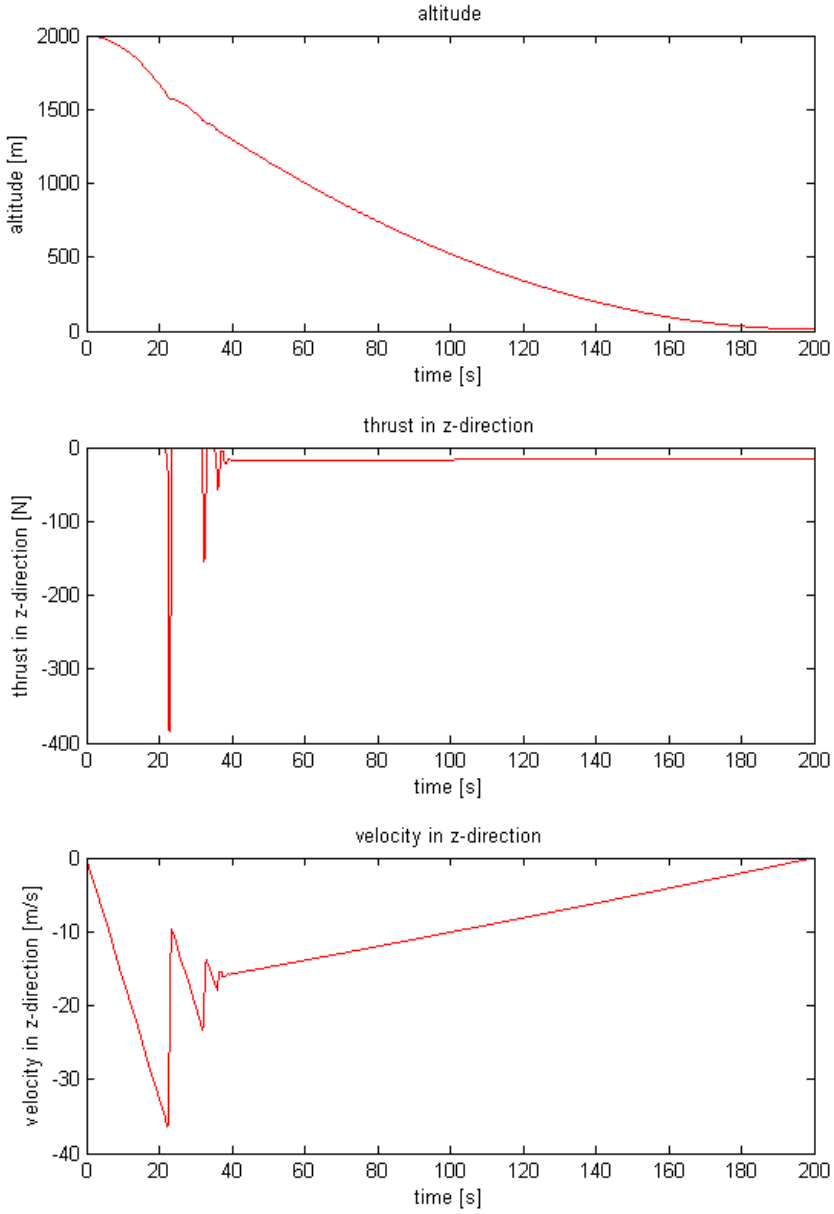


Figure 2.37: Results for PD controlled landing

The controller fulfills the task of smoothly landing the spacecraft. It is visible that for the first 21 seconds, the spacecraft is in free fall, and only after that, the controller takes over and starts breaking.
With this result, we show that this approach can be applied to the spacecraft. However, this controller still uses the given change in velocity in x-direction. To achieve this, some external input has to be present, and the velocity has to be controlled to follow a given curve.
We felt that it would be desirable to find a landing controller that does not require any external inputs, described below.

Landing Control without feed-forward external braking In this section, we present a landing controller that does not require any external sensory input to set the velocity in x-direction. The idea consists in extracting both the vertical and horizontal components of the optic flow and use the first one to control the descent speed and the second one to control the horizontal speed. Figure 2.38 depicts the schematic structure of the controller.

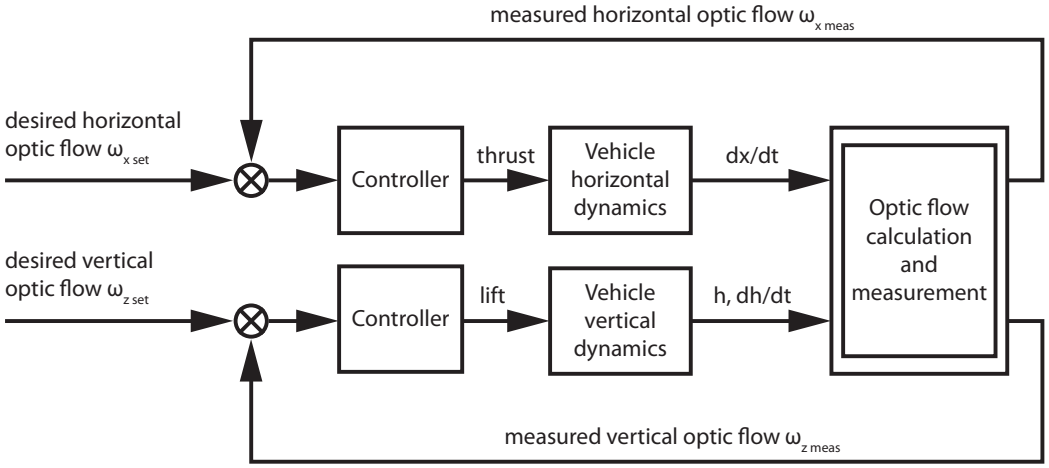


Figure 2.38: Schematic of the controller without active braking.

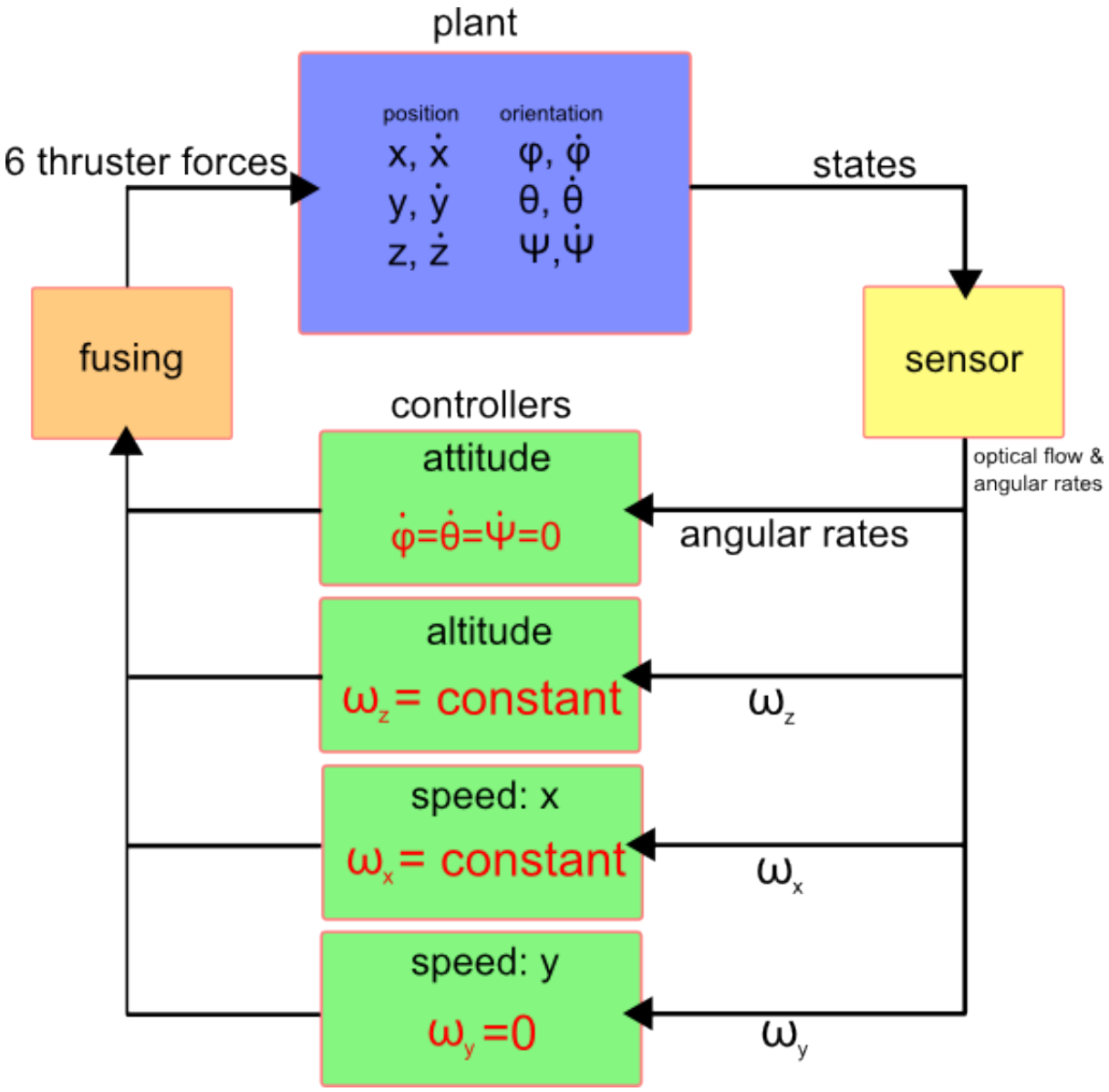


Figure 2.39: Schematic structure of the position and landing controller

Defining the optic flow for the x-direction as

$\omega_x = \frac{v_x}{z}$

suggests that controlling ω_x to be constant at a given setpoint will yield the desired behavior, if the altitude z is controlled to go to zero at the same time.
It is therefore possible to split the position and landing control into three separate parts:

1. Control $\omega_z = \frac{v_z}{z}$ to be constant according to section 2.6.1.
2. Control ω_x to be constant in order to decrease the velocity in x-direction with decreasing altitude.
3. Control ω_y to be equal to zero in order to avoid any velocity in y-direction.

The PID controller for the z-direction is the same as in section 2.6.1, but the parameters had to be adjusted to the new sensor approximation. All the values had to be multiplied by 32, because the new sensor results in overall smaller values. The new parameters are found to be:

$\omega_{setpoint,z} = 0.02$
$K_p = 640$
$T_i = 12.8$
$T_d = 0.64$

The PID speed controllers for the x- and y-direction have the same parameters, since the spacecraft was assumed to by symmetric (z-direction is an exception because of the gravity force). A very simple linearized model of the velocity in x-direction or y-direction can be used to check the resulting PID controller, even though once again one cannot trust those linear results since the output of the real system is still nonlinear.
The linearized system has only one state ($\dot{x}$ or $\dot{y}$) and can be represented by the following matrices:

$A = 0$
$B = \frac{1}{m}$
$C = \frac{1}{1000}$

with 1000 being the middle value between the starting altitude of 2000m and the final altitude of roughly 0m.
The resulting PID parameters are based on the results of the z-controller, though slightly smaller because the measurement values in x-direction are larger than those in z-direction. The values have been found by hand tuning starting from the altitude controller.

$\omega_{setpoint,x} = 0.25$
$\omega_{setpoint,y} = 0$
$K_p = 500$

$T_i = 4$
$T_d = 0.45$

The setpoint in x-direction was chosen to match the initial conditions of $z = 2000m$ above ground and $v_x = 500\frac{m}{s}$, while the velocity in y-direction has to stay at zero as mentioned before. With this setting, the controllers can now be tested in the simulink framework. The schematic structure presented in figure 2.39 was implemented in the Simulink model, shown in figure 2.40.
Note that the structure of this model can easily be transferred into the general framework. It misses only the PANGU connection as we are not using images; instead we approximate the optic flow directly form the states.

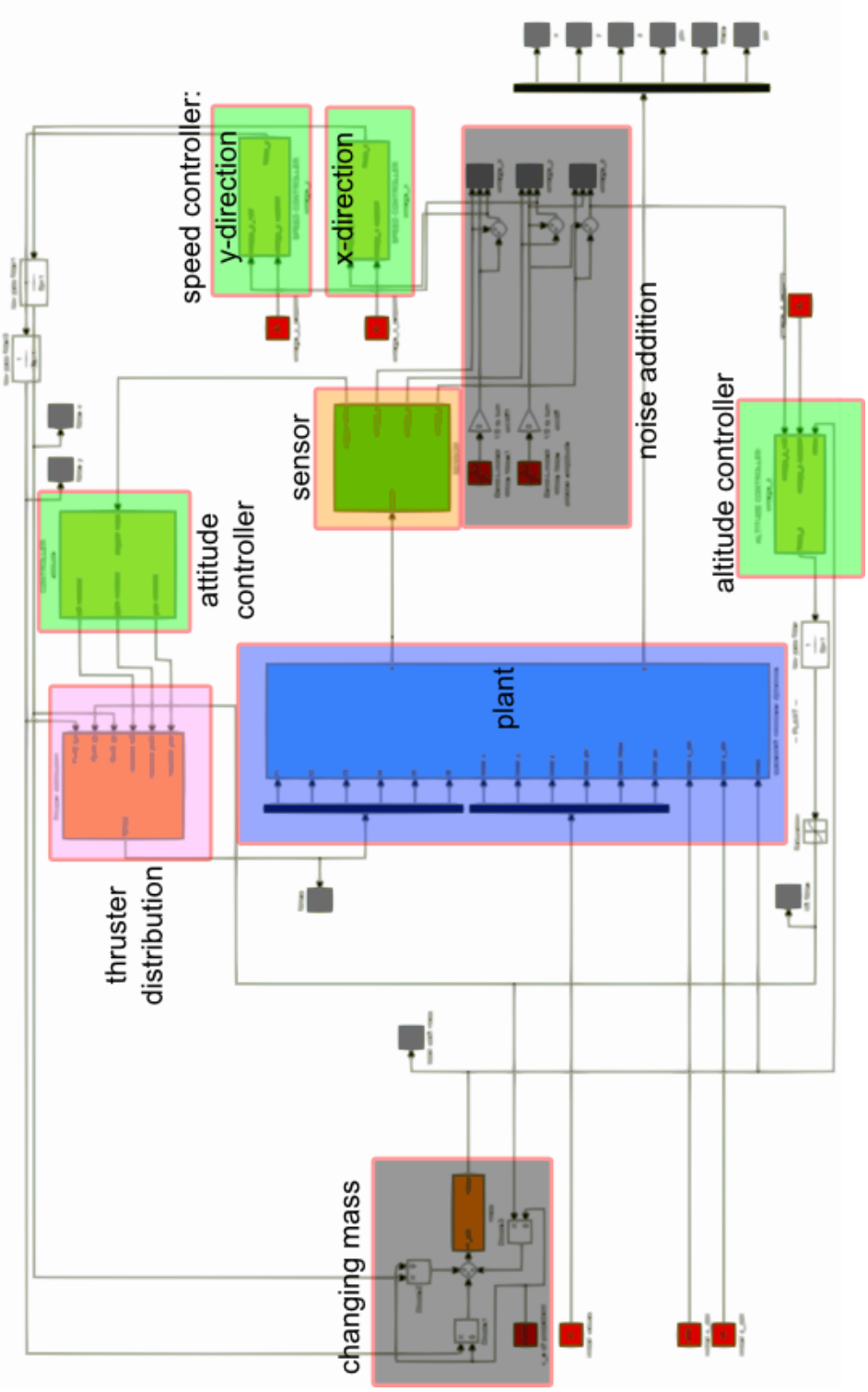


Figure 2.40: Simulink model for the position and landing controller

Simulating this model for 200 seconds without any addition of noise results in the behavior shown in figure 2.41. For this simulation, the initial conditions $z = 2000m$,$\dot{z} = -40\frac{m}{s}$ and $\dot{x} = 500\frac{m}{s}$ were chosen. Those are typical values for the landing trajectory on the moon at this altitude.
No disturbances are active in y-direction, as well as on the angular states to keep the plots clean. Nevertheless, the controllers in those directions are active during the simulations and could counter any disturbances (as will be shown later in this section when noise is added). Figure 2.34 shows how the system reacts to disturbances on the angular states.
Without any noise or disturbances, two controllers (altitude and velocity in x-direction) are active. Figure 2.42 plots ω_x and ω_z over time, while the black line represents the respective setpoint.
From these results, one can notice that the controllers act as intended: the spacecraft descends and slows down its downward and forward velocity. By measuring the optic flow only, and without any additional external input, the proposed controller structure is able to bring the spacecraft from the given initial conditions to the desired final state, in which the spacecraft is close to the ground and has slowed down its velocity to near standstill.
In a second step, the optic flow measurement was not assumed to be perfect anymore. To simulate the imperfect measurement, white noise has been added to the three measured $\omega_{z,y,x}$ as depicted in figures 2.43, 2.44 and 2.45 respectively.
The system was simulated again using noisy measurements of the optic flow as inputs to the controllers. The results are shown in figure 2.46. The most important result of this experiment is that the relevant behavior of the spacecraft is very robust to noise. The spacecraft still descends and slows down such that the overall result still satisfies the goals of the project. The only big difference is that now there is some movement in y-direction, which is due to the noise added to the measured ω_y. Because of the active speed controller, which has the purpose of controlling the velocity in y-direction to zero, the noise has an influence on the spacecraft, because the controller now engages to counter the noise. It can be seen in the plot in the middle right of figure 2.46, that the magnitude of the velocity in y-direction stays rather small with a maximum of about $0.25\frac{m}{s}$, and that the controller indeed stabilizes the velocity around zero.
The noise signal used in this simulation is rather large compared to the measurement signal: the noise has peaks that nearly equal the biggest amplitude of both the ω_x and ω_z curves. In terms of signal-to-noise ratio, the noise can be seen to be quite large too as we find the two signal-to-noise ratios to be $SNR_x = 21.417$ dB and $SNR_z = 16.8647$ dB. In y-direction, the measured ω_y is caused by the noise, and we see that the controller actually reduces the magnitude and therefore works as intended.
Overall, the proposed controller structure fulfills its task and shows good robustness to measurement noise, which is an important property given the fact that the optic flow measurement from the real sensors is not going to be perfect.

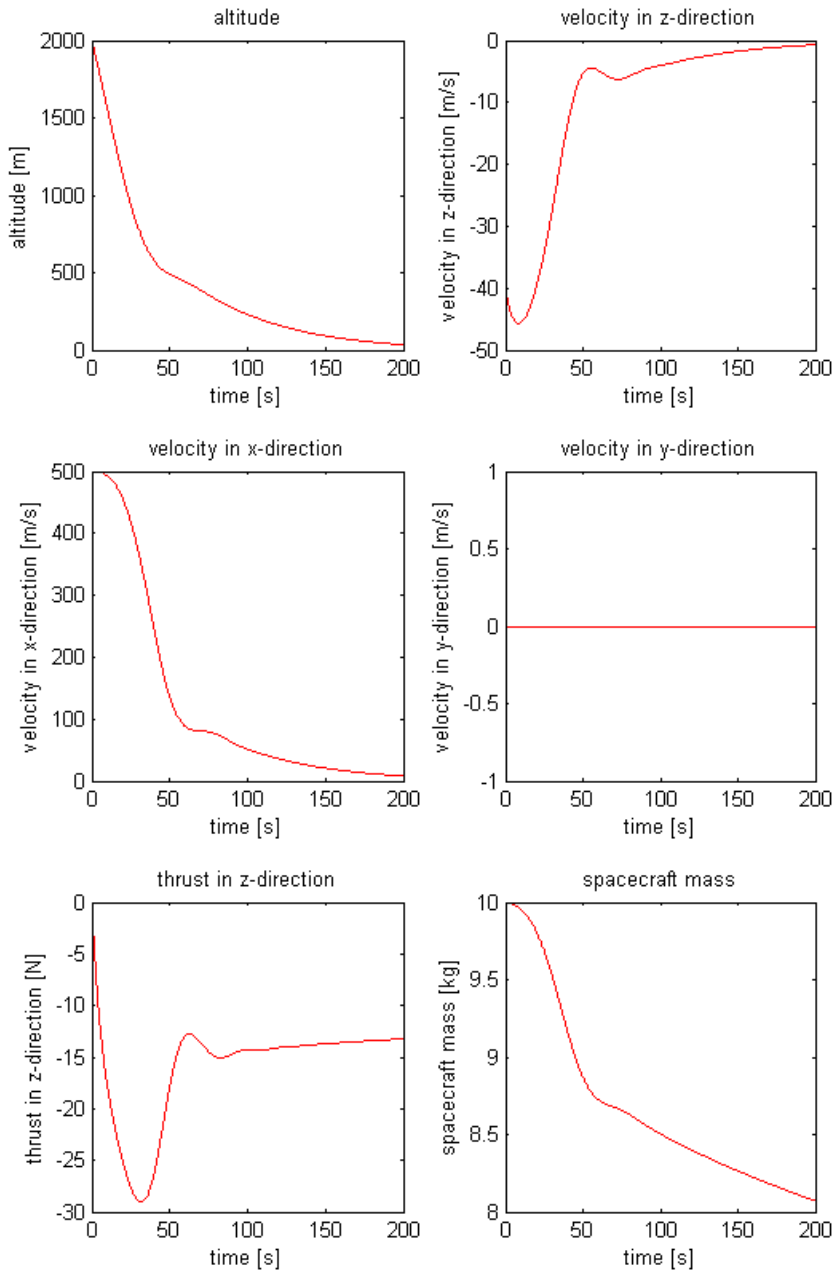


Figure 2.41: Results of the simulation without any noise.

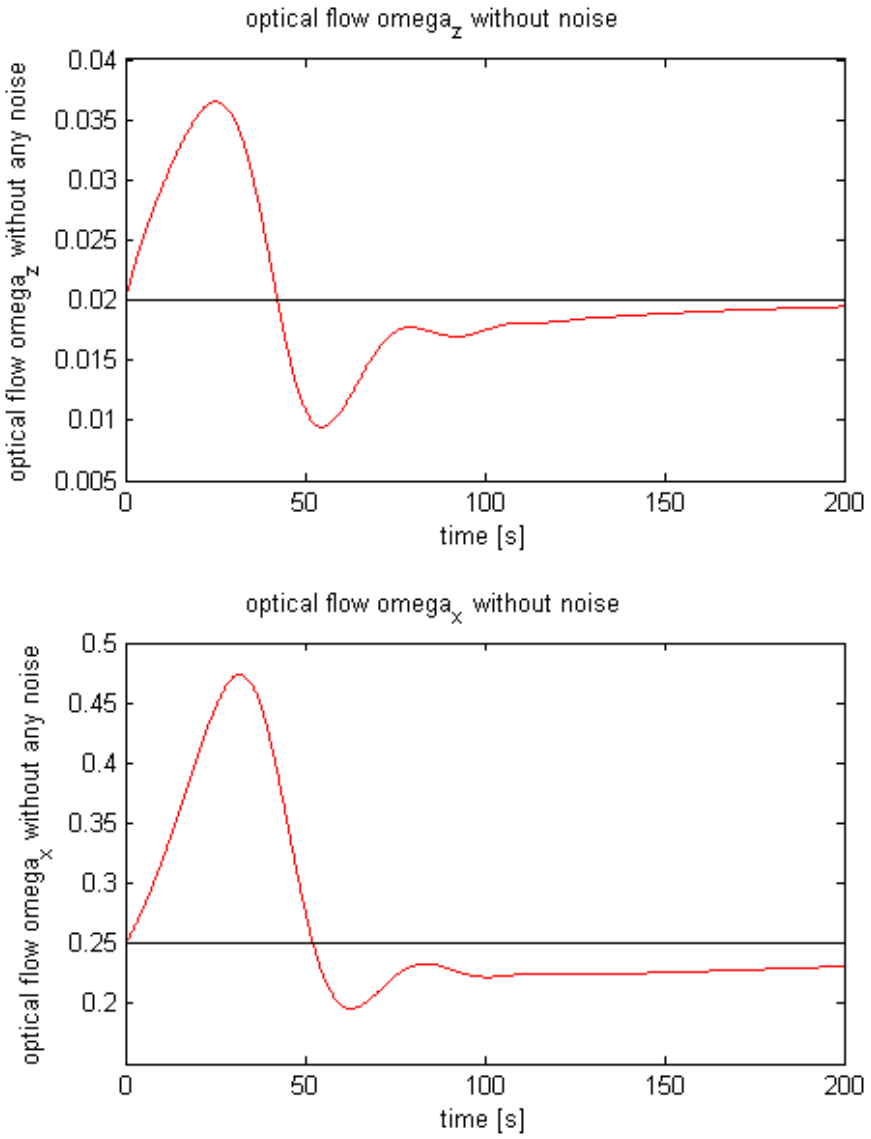


Figure 2.42: ω_z (top) and ω_x (bottom)

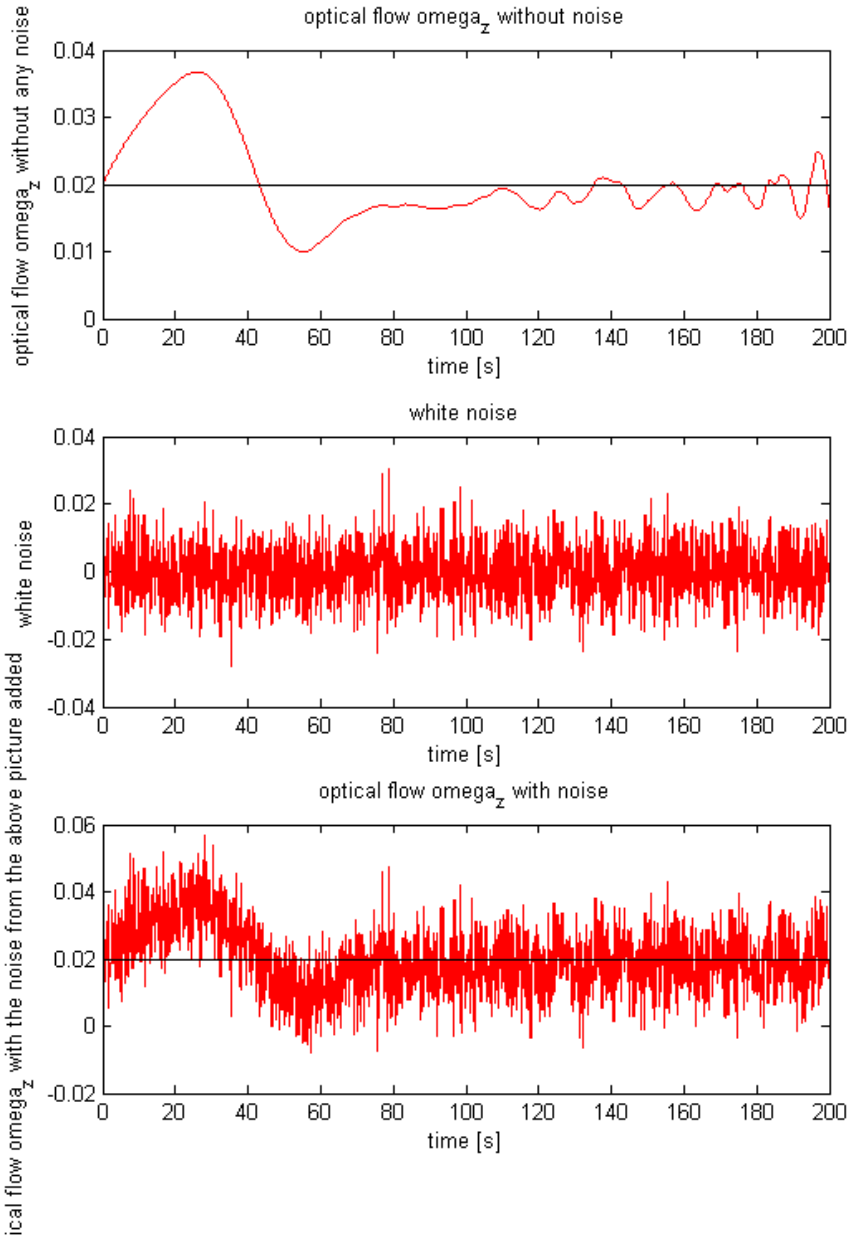


Figure 2.43: measured ω_z (top); white noise (middle); final signal: addition of ω_z and the noise (bottom)

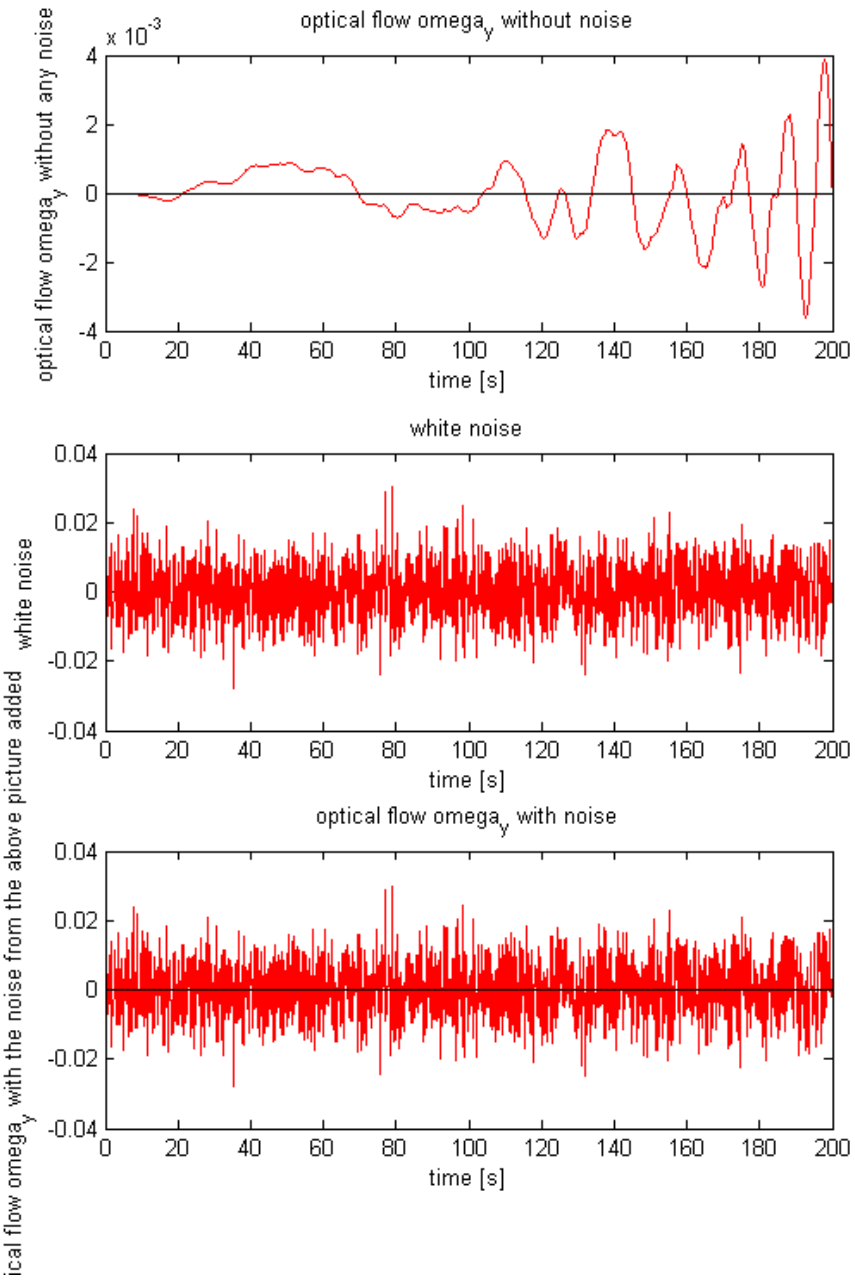


Figure 2.44: measured ω_y (top); white noise (middle); final signal: addition of ω_y and the noise (bottom)

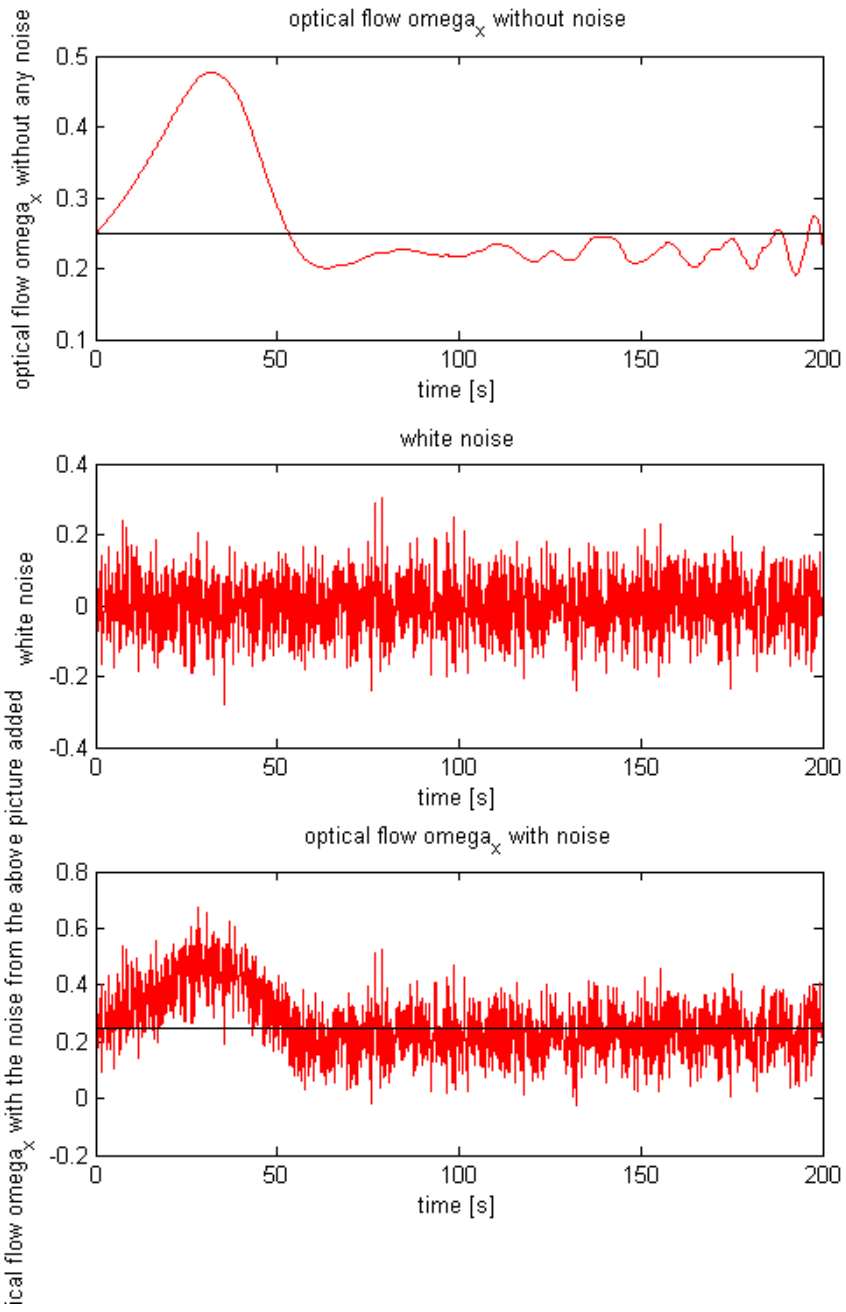


Figure 2.45: measured ω_x (top); white noise (middle); final signal: addition of ω_x and the noise (bottom)

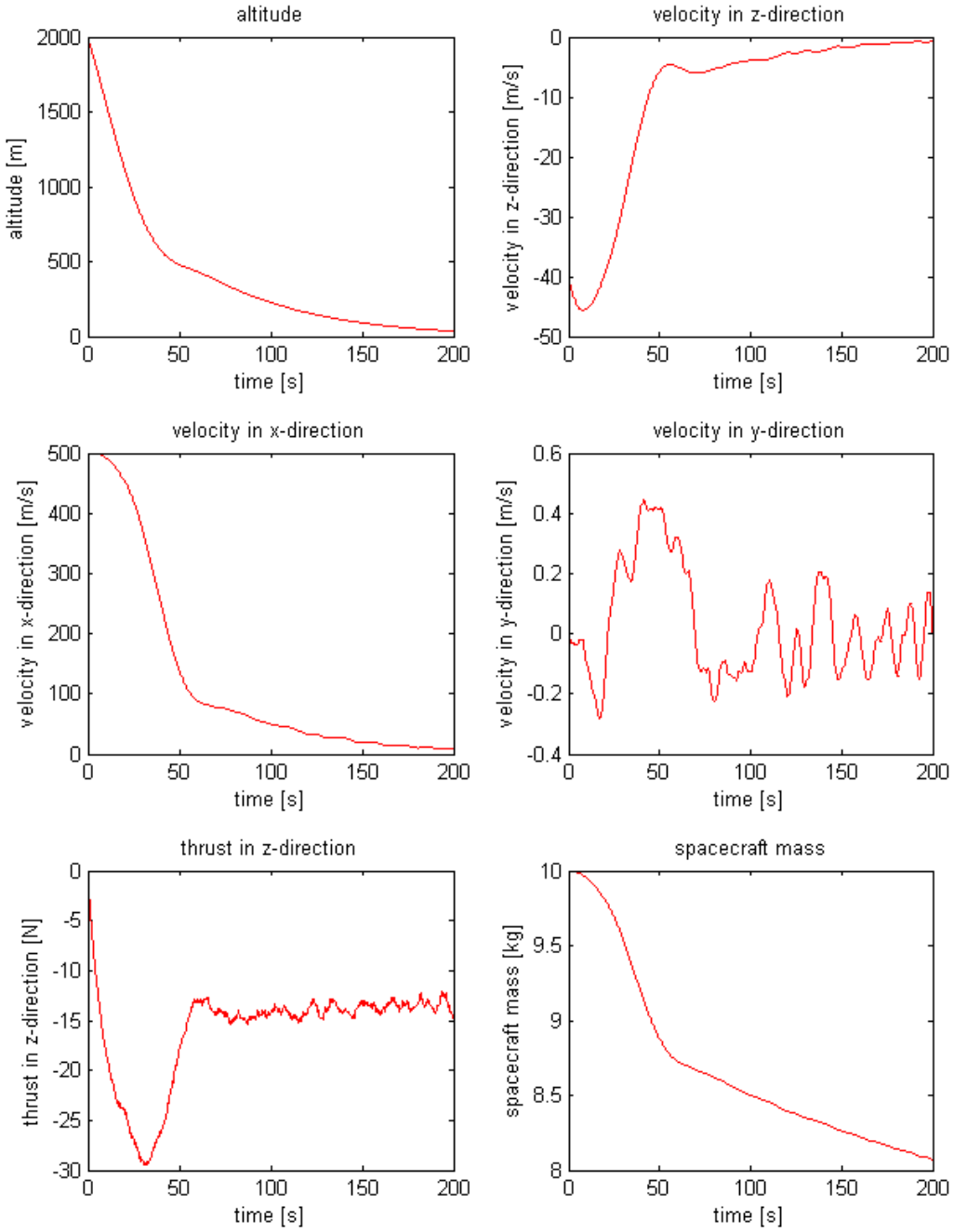


Figure 2.46: System behavior with noisy measurements of the optic flow.

2.6.3 Simulations with realistic visual feedback

In this section, the simulink model is used in combination with PANGU to test the proposed control strategies using realistic images. Only the grazing landing case is considered, since the second strategy proposed in 2.6.2 already includes a vertical speed controller.
Since the simulations were extremely slow, only one example per control strategy is shown. However, since the model of the sensor appears to work properly (cf.2.5.5) and the control strategies succeed with ideal (and even noisy) optic flow (cf.2.6.1,2.6.2), one can expect that the simulations of the complete system with realistic visual feed-back would work properly, as observed.
Two different sensor configurations are used. In the case in which the horizontal speed is not automatically controlled, a single downward facing sensor aligned with the direction of motion is used (fig:2.47A). In the case in which both the horizontal and vertical speeds are controlled automatically, two sensors aligned with the direction of motion and looking down at 45°are used (fig:2.47B).

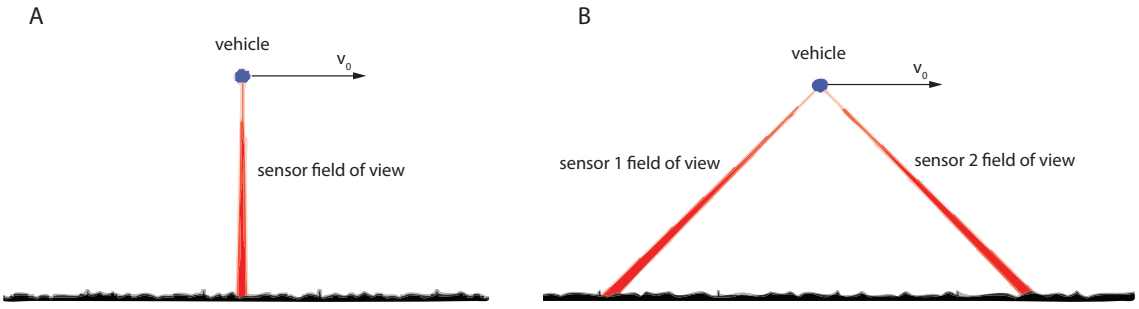


Figure 2.47: Sensor configurations used in the simulations with realistic visual feedback. *A*: one looking down sensor aligned with the direction of motion. *B*: two sensors aligned with the direction of motion and looking down at 45°. v_0: initial speed.

One sensor simulations

In this first case, assuming unknown initial conditions and information about the absolute speeds and distances, as in the case of a purely optic flow based navigation, it has been shown in 2.6.2 that it would be impossible to use this strategy in absence of an atmosphere. Instead, if an atmosphere were present, one could make use of the atmospheric drag to slow down the spacecraft without having to actively break it. Only in this case this strategy is applicable even without any a priori information about the vehicle initial state. That is we don't actively brake the vehicle, but instead it is braked by the atmospheric drag. Here atmospheric drag is simplified and considered as being a linear function of the speed (and not quadratic). If the drag is too small to brake down the vehicle in a reasonable time, one could assume that a parachute is deployed to increase the braking force. The results of the simulation with realistic visual feedback are shown in figure 2.48

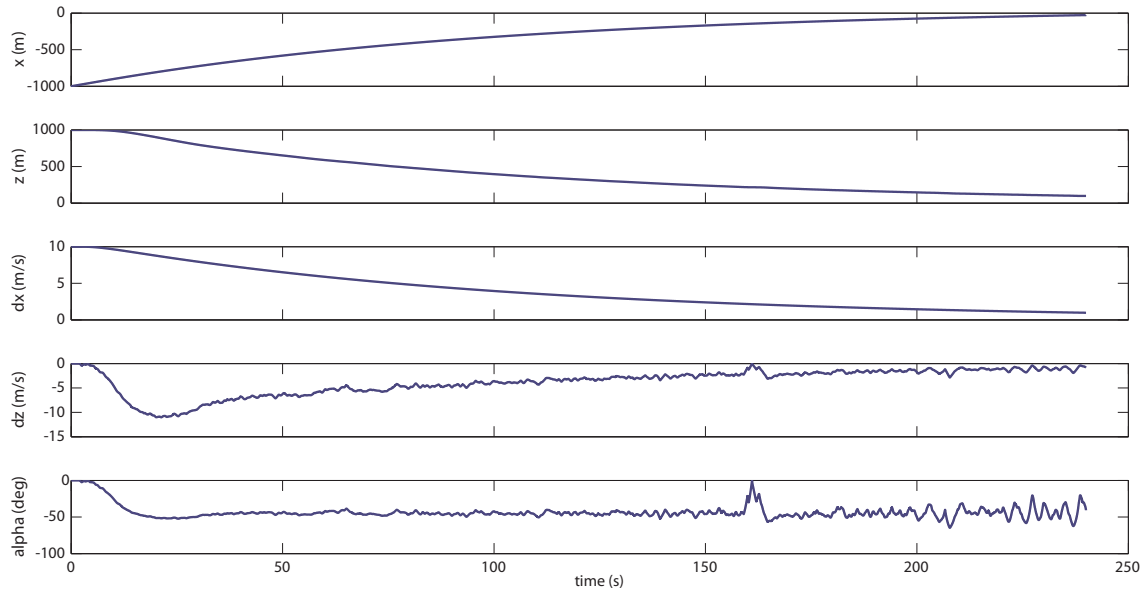


Figure 2.48: Simulation of landing trajectory using one sensor and atmospheric drag braking force. *x,z*: horizontal and vertical positions. *dx,dz*: horizontal and vertical speeds, *alpha*: descent angle

As expected, the vehicle is able to land safely. More precisely it safely approaches the ground, in fact due to the exponentially decaying horizontal speed, theoretically the spacecraft never touches the ground, because its vertical speed also decays exponentially. In practice, when the vehicle is very close to the ground the controller becomes unstable, because a small change in vertical position results in a huge change in ω.

Drag vs. descent angle A linear drag guarantees a constant descent angle (fig.2.48). This is certainly an interesting property of the controller, because in principle it could allow the autopilot to first locate the landing site at a high altitude and then simply fix the descent angle to approach it by fixing the desired optic flow set point ω_{set}. Unfortunately, while linear drag well approximates the slowing down trajectory of a helicopter of which the pitch angle is reset to zero (Franceschini et al., 2007), it does not describe the forces acting on a parachutes, which instead is more closely quadratic. A quadratic drag does not predict a constant descent angle.

In the linear drag case on can deduce the descent angle from the following equations:

$$\frac{dv_x(t)}{dt} = -K_d v_x(t) \tag{2.49}$$

$$v_x(t') = v'_0 e^{-K_d t'} \tag{2.50}$$

$$v_z(t') = \frac{dh}{dt'} = \frac{dv_x}{dt'}\frac{1}{\omega_{set}} = -\frac{K_d v'_0}{\omega_{set}} e^{-K_d t'} \tag{2.51}$$

$$v_z(t') = -\frac{K_d}{\omega_{set}} v_x(t') \tag{2.52}$$

$$\frac{v_z(t')}{v_x(t')} = -\frac{K_d}{\omega_{set}} \tag{2.53}$$

$$\alpha = -atan\left(\frac{K_d}{\omega_{set}}\right) \tag{2.54}$$

where K_d is the drag constant and α is the descent angle. Here we assum a perfect controller that always keeps ω at its desired value ω_{set}. As one can notice α is constant.
In the case of a quadratic drag, the same train of thoughts leads to a different conclusion:

$$\frac{dv_x(t)}{dt} = -K_d v_x(t)^2 \tag{2.55}$$

$$v_x(t') = \frac{1}{K_d t - v'_0} \tag{2.56}$$

$$v_z(t') = \frac{dh}{dt'} = \frac{dv_x}{dt'}\frac{1}{\omega_{set}} = -\frac{K_d}{\omega_{set}\left(K_d t - v'_0\right)^2} \tag{2.57}$$

$$v_z(t') = -\frac{K_d}{\omega_{set}} v_x(t')^2 \tag{2.58}$$

$$\frac{v_z(t')}{v_x(t')} = -\frac{K_d}{\omega_{set}} v_x(t') \tag{2.59}$$

$$\alpha = -atan\left(\frac{K_d}{\omega_{set} K_d t - v'_0}\right) \tag{2.60}$$

In this case, the descent angle is not constant, which precludes a precise selection of the landing site position at the beginning of the maneuver, without knowledge of the initial height or speed.

Two sensors simulations

The second model, summarized in figure 2.38, requires two sensors. If one wants to extract the vertical and horizontal components of the optic flow separately, a single linear sensor giving a single output is not sufficient. Instead, two linear sensors looking down at an angle as shown in figure 2.47B allow performing this calculation. The sum (or the mean) of the output of the two sensors provides the information about the horizontal component of the speed, while their difference provides the vertical one.

$$\omega_x = K_x(\omega_1 + \omega_2) \quad \text{and} \quad \omega_z = K_z(\omega_1 - \omega_2) \tag{2.61}$$

K_x and K_z depend on the angle at which the two sensors are placed.
With this configuration the vehicle is perfectly able to land safely, as shown in figure 2.49. Moreover, the presence of drag or even wind is counter-balanced by the controllers, which are purely vision based. On the other hand, this model would also work in the absence of an atmosphere.

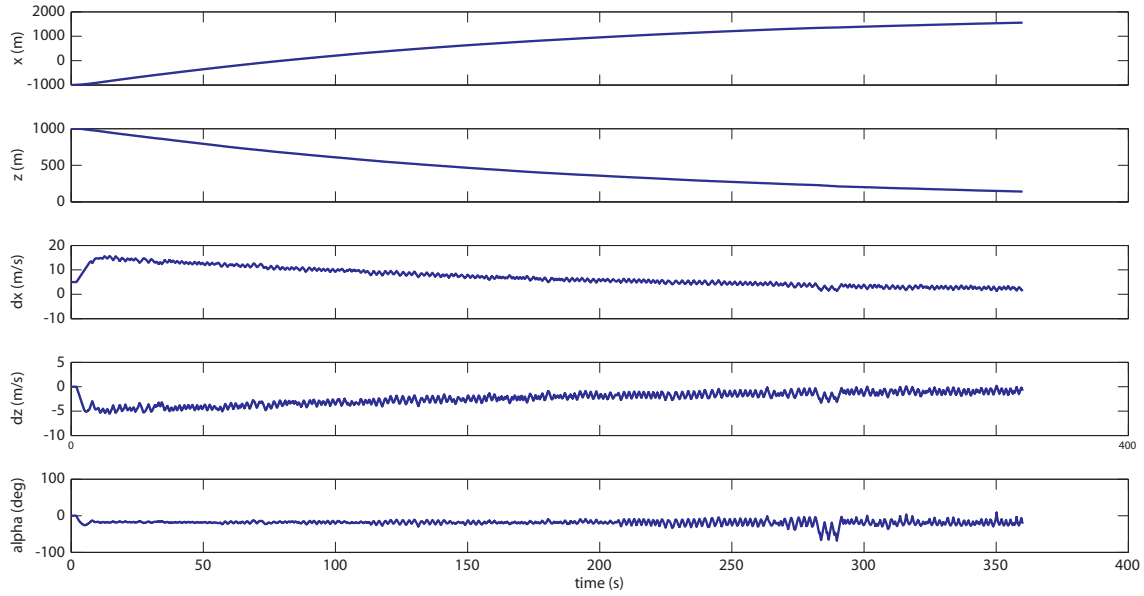


Figure 2.49: Simulation of landing trajectory using two sensors. *x,z*: horizontal and vertical positions. *dx,dz*: horizontal and vertical speeds, *alpha*: descent angle

In this case, the descent angle is constant and can easily be set by the autopilot. Moreover the relative landing speed, or more precisely the time constant of the speed exponential decay, can be set easily. The following equations give the mathematical proof of these statements.

$$\omega_{setx} = \frac{v_x}{h} \tag{2.62}$$

$$\omega_{setz} = -\frac{v_z}{h} \tag{2.63}$$

$$\frac{v_z}{v_x} = -\frac{\omega_{setz}}{\omega_{setx}} \tag{2.64}$$

$$\alpha = -\arctan\left(\frac{\omega_{setz}}{\omega_{setx}}\right) \tag{2.65}$$

$$\frac{dh(t)}{dt} = -\omega_{setz} h(t) \tag{2.66}$$

$$h(t') = h_0 e^{-\omega_{setz} t'} \tag{2.67}$$

$$\tag{2.68}$$

Chapter 3

Conclusions and outlook

In the previous chapters, we demonstrated the feasibility of optic-flow based landing strategies using biomorphic sensors. More precisely, we designed the full layout of a bio-mimetic aVLSI vision/motion sensor. We then performed simulations of the neuromorphic vision chip at different levels (cf.2.5) and we tested different optic-flow based landing strategies using realistic planetary scenarios (cf.2.6).
Many questions are still open, such as the performance of the chips in space mission like conditions. Nevertheless, these results provide a good starting point on which to base for the future developments of both the technology and the control strategies.

3.1 Neuromorphic analog VLSI technology for the computation of optic flow

Compared to standard methods, neuromorphic VLSI technology offers the advantages of allowing to build smaller, lighter and lower power vision sensors. On the sensor presented here (the Tracker Motion Sensor, TMS), the computation is analog and is performed on chip without the need for a digital processor, such as a microcontroller or a FPGA. This insures almost instantaneous and very low-power data processing in a compact housing. Indeed, the TMS power consumption is predicted to be < 1mW and the chip size is 2146x2147 μm^2 [1].
All the simulations we carried out (cf.2.5) demonstrated that the TMS is able to compute the true angular velocity even for the low contrast images generated using PANGU (fig.2.6). Moreover, the real sensor would not suffer from the temporal aliasing problem encountered when running a discrete time simulation (cf.2.5.5).
Different strategies can be used to extract the angular velocity from the pixel values measured by the chip. In our high-level simulations, we computed the angular speed by calculating the average value of the speed measured by all the pixels. The TMS also offers the possibility to use a WTA network to select the pixel with the highest activity (i.e. the fastest one). This solution, however, it most suited to detect the movement of objects in a static field of view rather than computing the overall angular speed, because this kind of measurement is

[1] The chip size depends on the manufacturing technology (in this case AMS $0.35\mu m$ VLSI technology). Newer technologies, like for instance Intel®45nm, would allow building even smaller chips.

more subject to noise and would tend to overestimate speed. On the other hand, taking the average of all the pixels introduces a longer latency in the system. To get a precise measure of angular speed from the overall average all the pixels must have seen a moving edge. This introduces a delay in the measurement that depends on both the sparseness of the image and the angular speed itself. Different intermediate angular speed measurement solutions can be explored in the future to try to find a good compromise between noise and latency.
The use of a 1D sensor, as the one presented here, is restricted to the longitudinal flight condition, in which the sensor is perfectly aligned with the direction of motion. Indeed, the direction of motion cannot be measured with a linear sensor and, moreover, linear sensors are very sensitive to the aperture problem (cf.2.5.5,fig.2.20). A more suitable solution for a fully autonomous flying vehicle would be the use of 2D sensors with which it would be possible to extract both the magnitude and the direction of the optic flow. Neuromorphic VLSI technology offers the possibility to build 2D sensors, therefore their future design has to be considered if we want to be able to operate under more realistic conditions.

3.2 Optic-flow based landing strategies: advantages and limitations

By measuring the optic flow, it is possible to navigate without having explicit knowledge about absolute position and speed with respect to the ground. This is particularly suited for flying and swimming animals, as well as UAVs, which operate in a drifting and sometimes turbulent media. Their movement cannot easily be measured directly.
We presented two control strategies for optic-flow based landing. The first one uses a single downward facing sensor aligned with the direction of motion (cf.2.6.3). The sensor allows measuring the horizontal component of the optic flow. A controller keeps it constant by setting the altitude, while the horizontal speed is passively reduced thanks to the atmospheric drag. The second strategy we proposed is based on two sensors aligned with the direction of motions, looking down at an angle (cf.2.6.3). This solution offers the possibility two extract both the horizontal and vertical components of the optic flow and have full control over both the horizontal and vertical speed. The two-sensors solution revealed being more flexible and independent from the presence of an atmosphere.
From a control point of view, optic flow based navigation is challenging. Although we demonstrated that optic-flow based landing is achievable, the relativity of the measured quantities complicates the control task. Based on the optic flow only, the state of the vehicle in the inertial frame of reference cannot be measured, because the translational optic flow is always a function of both the speed and the distance from the imaged objects (e.g. Dahmen et al., 2001). The majority of the more modern and efficient control strategies are based on the estimate of the system's state vector and the inability to do it precludes their usage (cf.2.6.1). We showed that simple PID control strategies are able to smoothly land the vehicle up to a certain altitude after which the system becomes unstable. By choosing suitable parameters for the controller and by filtering and saturating its outputs it is possible to the avoid running into instability before the low-gate condition is reached.
Nonetheless, the integration of the optic flow sensors with other sensory modalities that allow

estimating the system's state would be convenient, because it would considerably increase the robustness of the control strategies. Information about the distances could for instance be inferred by using stereo vision or LIDAR, gravity sensors and compasses could be used for the estimate of the absolute orientation.
On the other hand, insects do not seem to use absolute sensing (Taylor and Krapp, 2007) and still they are able of robust flight control. For this reason it is important to continue the effort of trying to identify the working principles of the sensory systems and the multimodal sensory integration in insects.

3.3 Towards a robotic implementation

We are now designing a hybrid reality/simulation framework to test the presented sensing and control strategies on a flying robot, in collaboration with the D'Andrea lab at the Institute for Dynamic Systems and Control, ETH Zürich. An X3D quadrotor (Ascending Technologies, Germany, fig.3.1A-B) will be flown in a room equipped with a motion tracking system that is able to provide information about the position and orientation of the helicopter in real-time at 200Hz (the Flying Machine Arena, www.flyingmachinearena.org). This positional information will then be used to render a scene in a virtual visual environment to feed a model of the sensor that will compute the optic-flow from the rendered images (fig.3.1C-D). The extracted optic-flow will then be used to control the robot. All this procedure will happen in real-time with small latencies.
In the meanwhile, the real AVLSI chip will be sent out for production and will subsequently be mounted on the quadrotor and tested in the flying machine arena. The results of the tests with the real chip and those of the simulations will then be compared.
This particular framework allows developing new sensing and control strategies without having to simulate the dynamics of the helicopter and thus it simplifies the transfer from the simulations to the real world applications.

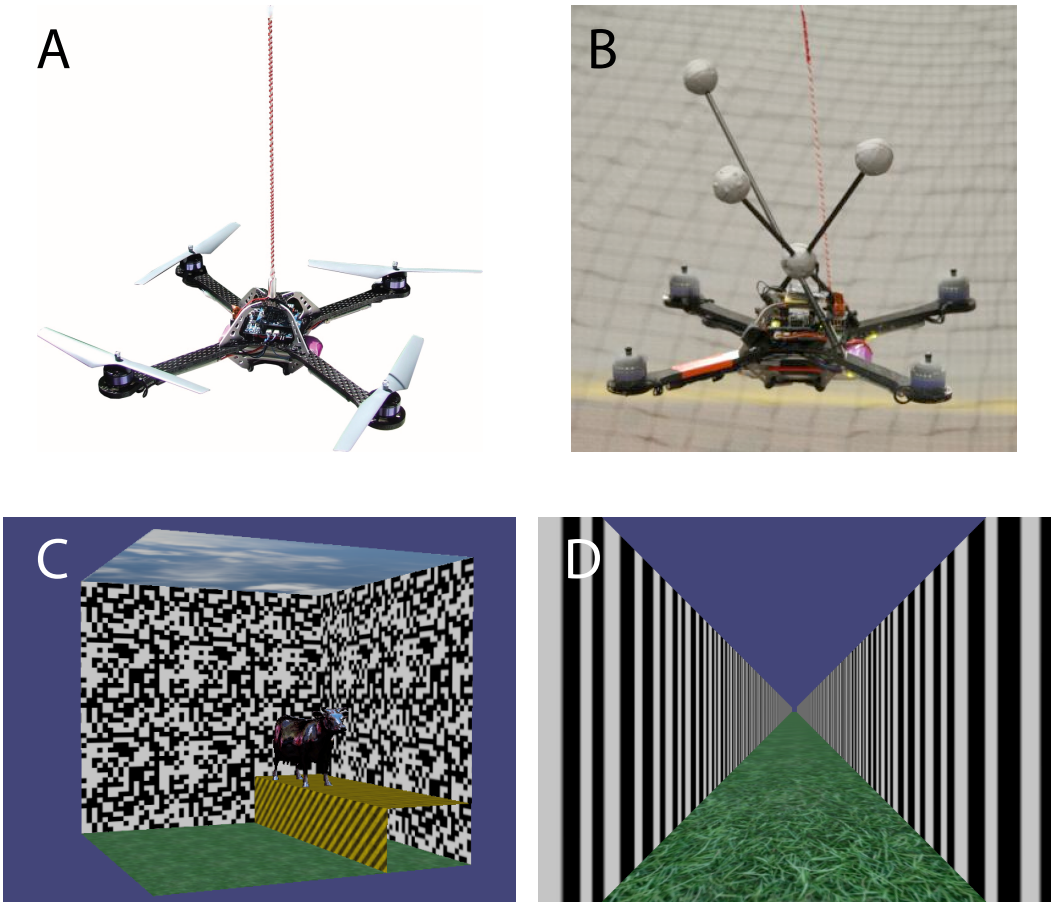


Figure 3.1: X3D quadrotor and 3D virtual environment. *A*: X3D quadrotor. *B*: Flying X3D quadrotor equipped with balls markers used by the motion tracking system. *C*,*D*: Examples of virtual environments generated using OpenSceneGraph (www.openscenegraph.org).

Bibliography

AL-Sunni, F. and Shafiq, M. (2003). Adaptive Control Schemes for a class of Hammerstein-Wiener nonlinear systems. *PROCEEDINGS OF THE 2003 10TH IEEE INTERNATIONAL*.

Ammann, S. (2009). Design of a biomorphic controller to land a vtol spacecraft. Technical report, ETH Zürich.

Bartolozzi, C. and Indiveri, G. (2009). Selective attention in multi-chip address-event systems. *Sensors*, 9(7):5076–5098.

Dahmen, H., Franz, M., and Krapp, H. (2001). Extracting egomotion from optic flow: limits of accuracy and neural matched filters. *Motion Vision: Computational, Neural, and Ecological Constraints*, page 143.

Delbruck, T. (1991). Bump'circuits for computing similarity and dissimilarity ofanalog voltages. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 1.

Findeisen, R. and Allgöwer, F. (2002). An introduction to nonlinear model predictive control. In *21st Benelux Meeting on Systems and Control*, volume 11. Citeseer.

Findeisen, R., Imsland, L., Allgower, F., and Foss, B. (2003). State and output feedback nonlinear model predictive control: An overview. *European Journal of Control*, 9(2-3):190–206.

Franceschini, N., Ruffier, F., and Serres, J. (2007). A bio-inspired flying robot sheds light on insect piloting abilities. *Curr Biol*, 17(4):329–335.

Fry, S. N., Rohrseitz, N., Straw, A. D., and Dickinson, M. H. (2009). Visual control of flight speed in Drosophila melanogaster. *J Exp Biol*, 212(8):1120–1130.

Graetzel, C. F., Nelson, B. J., and Fry, S. N. (2010). Frequency response of lift control in Drosophila. *Journal of The Royal Society Interface*, in print.

Harrison, R. and Koch, C. (1999). A robust analog VLSI motion sensor based on the visual system of the fly. *Autonomous Robots*, 7(3):211–224.

Hebisch, H., Nr, F., and Schwarz, I. (1995). Grundlagen der Sliding-Mode-Regelung. *ForschungsberichtNr*, 15:95.

Indiveri, G. (1997). Analog VLSI model of locust DCMD neuron response for computation of object approach. *Neuromorphic Systems: Engineering silicon from neurobiology*.

Indiveri, G. (2008). Neuromorphic VLSI models of selective attention: from single chip vision sensors to multi-chip systems. *Sensors*, 8(9):5352.

Indiveri, G., Kramer, J., and Koch, C. (1996a). Parallel analog VLSI architectures for computation of heading direction and time-to-contact. *Advances in Neural Information Processing Systems*, pages 720–728.

Indiveri, G., Kramer, J., and Koch, C. (1996b). System implementations of analog VLSI velocity sensors. *IEEE Micro*, 16(5):40–49.

Itti, L. and Koch, C. (2001). Computational modelling of visual attention. *Nature Reviews Neuroscience*, 2(3):194–203.

Jeong, B., Yoo, K., and Rhee, H. (2001). Nonlinear model predictive control using a Wiener model of a continuous methyl methacrylate polymerization reactor. *Ind. Eng. Chem. Res*, 40(25):5968–5977.

Khalil, H. (2002). *Nonlinear systems.* Upper Saddle River, NJ: Prentice Hall.

Kramer, J. and Koch, C. (1997). Pulse-based analog VLSI velocity sensors. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 44(2):86–101.

Liu, S. (1996). Silicon model of motion adaptation in the fly visual system. In *Proc. of Third Joint Caltech/UCSD Symposium*. Citeseer.

Liu, S. (1999). Silicon retina with adaptive filtering properties. *Analog Integrated Circuits and Signal Processing*, 18(2):243–254.

Nesic, D. (1997). A note on dead-beat controllability of generalised Hammerstein systems. *Systems and Control Letters*, 29(4):223–231.

Neäsiãc, D. (1999). Controllability for a class of simple Wiener–Hammerstein systems. *Systems & Control Letters*, 36:51–59.

Nesic, D. (2000). Output feedback stabilization of a class of Wiener systems. *IEEE Transactions on Automatic Control*, 45(9):1727–1731.

O'Brien, R., Boernke, E., and Gorsky, L. (2003). Sampled-data control of double integrator systems. In *System Theory, 2003. Proceedings of the 35th Southeastern Symposium on*, pages 413–416.

Patwardhan, R., Lakshminarayanan, S., and Shah, S. (1997). Nonlinear Model Predictive Control using PLS based Hammerstein and Wiener Models. In *AI Ch. E. Meeting, LA, Session*, volume 217.

Rao, V. and Bernstein, D. (2001). Naive control of the double integrator. *IEEE Control Systems Magazine*, 21(5):86–97.

Rohrseitz, N. and Fry, S. N. (2010). Behavioral system identification of visual flight speed control in *Drosophila melanogaster*. *Journal of The Royal Society Interface*, in print.

Srinivasan, M., Zhang, S., and Chahl, J. (2001). Landing strategies in honeybees, and possible applications to autonomous airborne vehicles. *The Biological Bulletin*, 200(2):216.

Taylor, G. K. and Krapp, H. G. (2007). Sensory systems and flight stability: What do insects measure and why? In Casas, J. and Simpson, S., editors, *Insect Mechanics and Control*, volume 34 of *Advances in Insect Physiology*, pages 231 – 316. Academic Press.

Wang, H. and Wang, H. (2006). Wiener-Typed Nonlinear Systems' Output Feedback Control Algorithm Based on Dual-Mode Method. In *Decision and Control, 2006 45th IEEE Conference on*, pages 2698–2703.