Software for Model Helicopter Flight Control

Markus Kottmann

kottmann@inf.ethz.ch

March 1999

Abstract

An embedded system for the flight control of an autonomous helicopter has been developed at the Institute for Computer Systems (ICS) at ETH Zurich. The technical reports [1]-[3] describe its hardware core, the programming language used, and the software core, respectively. This report covers the following topics: communication, data logging, navigation (state estimation), modelling of the helicopter, state space control, and trajectory planning.

Contents

1 Introduction 1
2 Configuration 2
  2.1 Inputs and Outputs of the Board Computer 3
  2.2 Scheduling 4
  2.3 Serial Communication: Messages 5
3 Data Flow and Access 7
4 Navigation 9
  4.1 Attitude Filter 10
  4.2 Position Filter 12
5 Control 14
  5.1 General Remarks 14
  5.2 Model 15
  5.3 Design of the Controller 16
  5.4 Implementation of the Controller 17
6 Trajectory Generator 19
7 Finite State Machines 20
8 Further Development 21
  8.1 General Ideas 21
  8.2 Varying Number and Quality of Sensors Used 21
  8.3 Geographical Information Systems 22
9 References 22

Technical Report Nr 314

1 Introduction

The model helicopter project originated at the Measurement and Control Laboratory (IMRT) at ETHZ. Starting in the mid-eighties, first experiences were gained using a prototype of a helicopter mounted on a frame. While the frame limits the range of possible movements, it allows a fast and precise calculation of the state[1] of the helicopter (position, velocity, attitude, and angular rotation rates) based on the measurements of an appropriate set of angles.

In 1996, a first aerial model helicopter was developed. Of course, not only the capabilities and the range of possible missions of the helicopter were thereby multiplied, but also the problems of state estimation and control, of communication and of power supply. In that year, the Swiss team also achieved second place at the International Aerial Robotics Competition organized by the International Association for Unmanned Vehicle Systems (AUVSI).

Since the first flying prototype is equipped with two standard PC 104 systems (Intel 486), the consumption of electrical energy is relatively high (~50 W). This leads to a heavy battery pack and thus to a short maximal duration of missions. To overcome this drawback, an embedded system[2] has been developed at the Institute for Computer Systems (ICS). It consists of the hardware arranged around a low-power 200 MHz StrongARM processor [1], the programming language Oberon SA including a compiler [2], and a small operating system customized to our needs [3]. The current on-board system consumes about 15 W, 60% thereof being needed for the GPS card.

In this report, the overall configuration is shown and the on-board tasks are described, including the scheduling scheme. In detail, these tasks are

- *IO/Communication*: Signals are read from and written to peripheral devices.
- *Logging*: Data is logged on board and simultaneously transmitted to ground.
- *Navigation*: The state of the helicopter is continuously estimated by updating the previous state estimation based on new measurements.
- *Control*: The control output is calculated based on the current state estimation and on a reference signal being tracked.

As the helicopter represents a very complex and broad field of research, it offers a large variety of directions for further investigation. Therefore we conclude the report with an outlook into future developments.

1. In the context of a physical system - such as the helicopter - the notion ‘state’ means a collection of continuously changing variables. Often, these variables correspond to some kind of energy storage, e.g. velocity corresponds to kinetic energy, position to potential energy etc. In this report ‘state’ is also used in the sense of a logical or discrete state e.g. describing finite state machines. The intended meaning should be clear from the context.
2. The system has been named OLGA: Oberon Language Goes Airborne.

2 Configuration

Only a brief overview on the overall configuration is given in this report (Fig. 2.1). A more detailed description can be found in [4].

There are two connections between the ground and the on-board system. First, we have the standard remote control system operated by a human pilot. While it serves as a backup in case the autopilot fails, it is also very useful in the process of controller design. Numerical identification of the helicopter model can be performed by logging the sensor signals as well as the control signals originating from remote control. Secondly, a datalink is used to upload the software to the on-board system, to control some logical states and controller settings from ground, to send segments of the desired trajectory to board, to get online data from the on-board system, and to send GPS correction messages on-board.

Note that in the final state of development the helicopter is to operate autonomously, except for the GPS correction messages which are necessary to make the GPS work in differential mode.

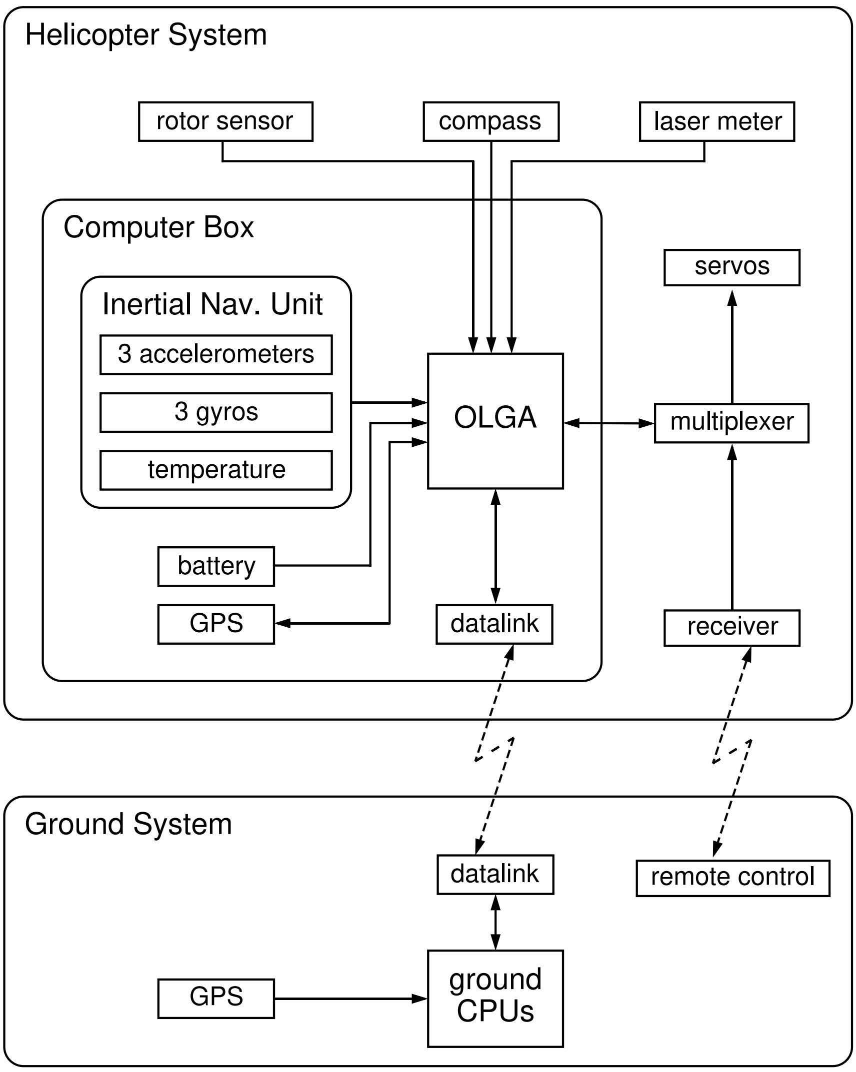

Fig. 2.1. Configuration of the system

2.1 Inputs and Outputs of the Board Computer

In the following two tables we show the characteristics of input and output signals which must be handled by the board computer. The TYPE field indicates whether a signal is read using an analog-to-digital converter (A/D), received in pulse-width or pulse-frequency-modulated form (PWM/PFM), or whether it arrives over an UART (serial)[1].

All A/D, PWM, and PFM signals may be regarded as typical signals in a sampled data control system, i.e. a procedure `GetInput(input)` returns an update of the input signal almost instantaneously[2]. Note that we are using two different sampling times: whereas 20 ms is regarded to be fast enough for the control loop, the navigation part is running four times faster.

Table 1: Characteristics of input signals

SENSOR / INPUT CHANNEL	MEASURED VALUE / TYPE OF MESSAGE	TYPE	SAMPLING TIME [ms]
3 accelerometers	acceleration	AD	5
3 gyros	rate of angular rotation	AD	5
temperature sensor	temperature of inertial navigation unit	AD	5
battery	current voltage	AD	20
rotor sensor	frequency of rotor	PFM	20
6 servos	servo signals from remote control	PWM	20
GPS	position message (msg) velocity msg	serial	—
compass	heading msg (track over ground)	serial	—
laser meter	height-above-ground msg	serial	—
datalink	command msg controller msg segment msg GPS-correction msg (to be forwarded)	serial	—

The serial inputs are handled differently. Typically, about five messages arrive per channel and second. The messages consist of an id-dependent number of characters. As the single characters arrive, they are collected to form messages. The complete messages then are either forwarded to other devices or used locally.

Therefore, a serial input cannot be sampled in the sense of actively requesting an update of the signal. Instead, it is checked whether an update is available and, if there is, the new value is used in the calculations.

1. Details on the implementation of drivers are found in [3].
2. With delay times of less than 5µs the A/D converter is the slowest of these components.

Similar statements hold for the outputs: the control signal is sent to the servos instantaneously, whereas the transfer of messages may suffer some delay.

Table 2: Characteristics of output signals

ACTUATOR / OUTPUT CHANNEL	OUTPUT SIGNAL / TYPE OF MESSAGE	TYPE	SAMPLING TIME [ms]
6 servos	control signal	PWM	20
GPS	GPS-correction msg (forwarded)	serial	—
datalink	logger msg config msg	serial	—

2.2 Scheduling

In its simplest form, the implementation of a discrete-time controller contains the following steps which are performed periodically, i.e. with a given rate:

1. read input signals from sensors
2. calculate control signal
3. write control signals to actuators.

There is a hard real-time condition for the sampling times; no jitter is desired. Besides that, the time consumed for one execution of the control algorithm must fit into the time slot provided.

In our example we introduced a second sampling time. Since the ratio between the two times is integer-valued (20 ms = 4 * 5 ms) the scheduling can be kept relatively simple. We just need a counter to distinguish the four phases. The tasks corresponding to the three steps mentioned above all are controlled by the same interrupt (IRQ, 200 Hz). Every time the interrupt occurs the input task is performed while output and calculation (navigation and control) tasks are triggered only every fourth time (Fig. 2.2).

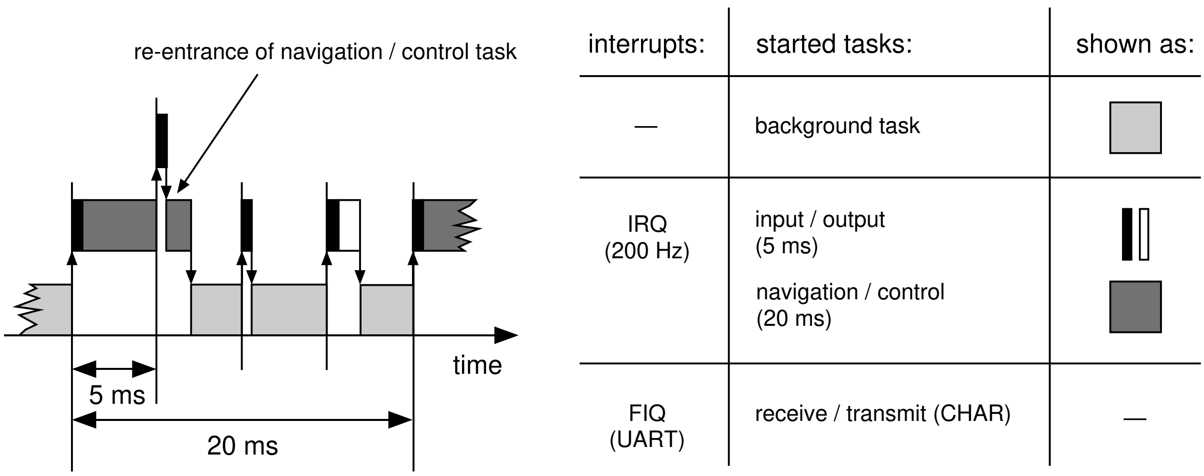

Fig. 2.2. Scheduling scheme

Note that

- the calculations may last longer than 5 ms, therefore support of re-entrance is necessary
- the calculations make use of a set of input data sampled in the previous 20 ms (while the new samples are collected in a new set)
- the output phase is not directly attached after the calculation phase; this causes an additional delay, but it ensures that a fixed controller leads to identical closed-loop behaviour under changing timing conditions (e.g. if some ‘overhead’ is introduced)

A second interrupt (FIQ, fast interrupt) handles the receiver part of the UARTs. Arriving characters are written to buffers. During an interval of 20 ms about 5-20 FIQ interrupts occur, each taking less than 2.5 µs. The corresponding elements have been omitted in Fig. 2.2.

In the idle phases control is given to the background tasks.

2.3 Serial Communication: Messages

The background tasks deal with serial communication on the level of messages. They are organised in a round-robin scheme, and we have three types of them:

- receiver tasks building up messages from incoming characters,
- transmitter tasks sending complete messages,
- logger tasks which formulate messages containing data demanded by the ground station.

A *receiver task* implements a state machine which reflects the structure of the incoming messages: typically three synchronization characters are followed by the checksum (a bitwise XOR of all other characters of the message), the identification number, the length, and the ‘real content’ (data) of the message (Fig. 2.3).

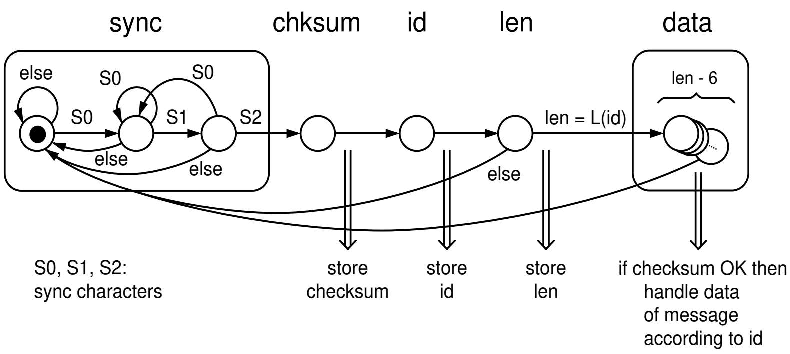

Fig. 2.3. State machine reflecting a message protocol

As sketched in the following excerpt of `MODULE Communication`, the receiver task `RxGRD` operates on the global record variable `rxGRDinfo`. That variable encapsulates all data which must be persistent between successive activations of `RxGRD`.

```
MODULE Communication;
IMPORT ComBase, Scheduler, HKernel;
CONST
   GRDchannel = 0;

   GRDsync = 0;
   GRDchksum = 1;
   GRDid = 2;
   GRDlen = 3;
   GRDdata = 4;

TYPE
   RxGRDinfo = RECORD
      state, syncState: INTEGER;
      checksum: CHAR;
      id, len: INTEGER;
      data: ComBase.Buffer;
      msgCount, CECount: INTEGER(* used for (error) statistics *)
   END;
VAR
   rxGRDinfo: RxGRDinfo;

PROCEDURE RxGRD(me: Scheduler.Task);
VAR ch: CHAR;
BEGIN
   IF HKernel.Available(GRDchannel) > 0 THEN
      HKernel.Receive(GRDchannel, ch);
      IF rxGRDinfo.state = GRDsync THEN
         (* ... *)
      ELSIF rxGRDinfo.state = GRDchksum THEN
         (* ... *)
      ELSIF rxGRDinfo.state = GRDid THEN
         (* ... *)
      ELSIF rxGRDinfo.state = GRDlen THEN
         (* ... *)
      ELSIF rxGRDinfo.state = GRDdata THEN
         (*
           - test checksum
           - copy (rxGRDinfo.len-6) chars into rxGRDinfo.data
           - perform appropriate action (depending on rxGRDinfo.id)
           - rxGRDinfo.state := GRDsync
         *)
      END
   END
END RxGRD;

BEGIN
   rxGRDinfo.state := GRDsync; rxGRDinfo.syncState := 0;
   rxGRDinfo.msgCount := 0; rxGRDinfo.CECount := 0
END Communication.
```

Once a message has been completed its contents are either written into a global data storage (see also Fig. 3.1) or the message is forwarded, i.e. put into a mailbox.

The *transmitter tasks* handling the serial output are less complex. The messages to be sent are taken from a mailbox and sent ‘en bloc’. That means that the transmitter task — although running in the background — blocks (defers) the other tasks. This is justified because the typical message length lies in the range of 20-30 characters.

In a *logger task* messages are formulated and put into a mailbox. The data to be logged to ground is a subset of a collection of input signals, filtered data, and output signals. This subset may be changed during a flight.

The complete documentation of the message types defined and their formats is given in [5].

3 Data Flow and Access

This section contains a closer look at the data handling in our system. Especially those cases are of interest where data items are shared by tasks of distinct priorities. Fig. 3.1 shows two global

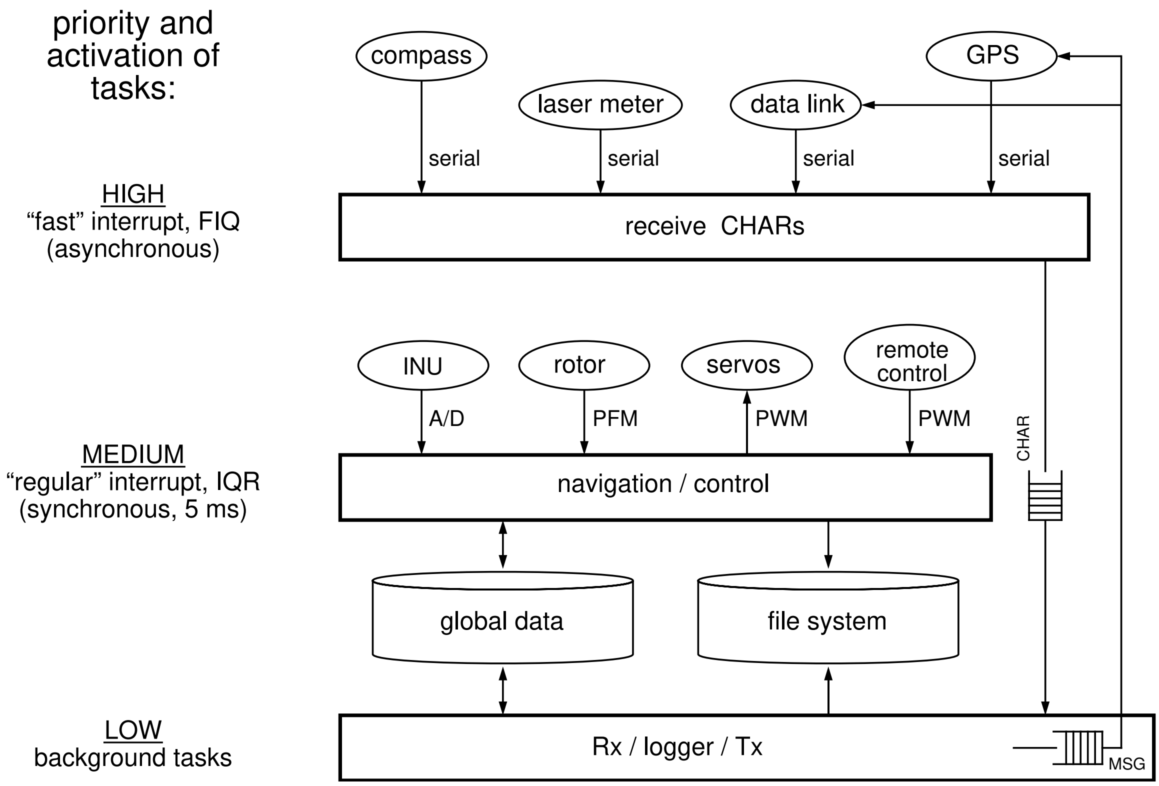

Fig. 3.1. Data flow on board

data storages in which data originating from different modules and data used by various modules is stored. Data which is used only by single modules normally is stored in variables belonging to those modules.

One of the global data storages is the file system on the RAM disk: this is a collection of signals which are stored on board during a mission. The set of signals to be collected is determined at the beginning of a mission. During the mission it is possible to (re-)start and stop the logging process at arbitrarily chosen times. Since time stamps are stored as well, it is possible to recover the times when logging is switched on and off. During the mission the file system is used only for writing, and distinctly prioritized tasks write to different files. This ensures that no problems occur concerning the access.

The second global data storage contains data which must be shared among various modules in a more flexible way. This concerns mostly the modules `Communication`, `Navigation`, `Control`, and `Logger`. The data storage is implemented in the module `Data` which also contains methods to access the data items. The data items normally are organised in arrays and records, as the following excerpt of module `Data` shows.

```
MODULE Data;

TYPE
   Vector3* = ARRAY 3 OF REAL;
   Vector3TLU = RECORD
                   vector3: Vector3;
                   time: INTEGER;
                   logged, used: BOOLEAN
                END;
   Vector6L = RECORD
                   vector6: ARRAY 6 OF INTEGER;
                   logged: BOOLEAN
                END;

VAR
   posEG: Vector3TLU;
   pwmIn: Vector6L;

(* posEG *)

PROCEDURE PUTposEG*(posX, posY, posZ: REAL; time: INTEGER);
PROCEDURE GETposEG*(VAR posEGvector3: Vector3; VAR time: INTEGER);
PROCEDURE posEGlogged*(): BOOLEAN;
PROCEDURE SETposEGlogged*;
PROCEDURE posEGused*(): BOOLEAN;
PROCEDURE SETposEGused*;

(* pwmIn *)
PROCEDURE PUTpwmIn*(pwm1, pwm2, pwm3, pwm4, pwm5, pwm6: INTEGER);
PROCEDURE GETpwmIn*(channel: INTEGER; VAR pwm: INTEGER);
PROCEDURE GETallPwmIn*(VAR pwm1, pwm2, pwm3, pwm4, pwm5, pwm6: INTEGER);
PROCEDURE pwmInLogged*(): BOOLEAN;
PROCEDURE SETpwmInLogged*();

BEGIN
   posEG.logged := TRUE; posEG.used := TRUE;
   pwmIn.logged := TRUE
END Data.
```

If required the following fields are added to a record:

- `logged` is used to keep track of whether a data item already has been logged to ground
- `used` indicates whether a data item has already been processed e.g. in the navigation algorithm
- `time` indicates the arrival time[1]

The problem of shared access is only of significance if the accessing procedures have different priorities. The following case is used as an illustration:

1. Some data, e.g. the Euler angles, are calculated by the navigation algorithm and written to `Data.`
2. The data then is read as input to the control algorithm.
3. The data is also read by a background task to be logged to ground.

The task invoking the last operation has a lower priority than the previous two. The following situations may ensue:

- The first two operations do not conflict. Because of the identical priorities their execution happens sequentially.

1. Time stamps have a resolution of 5 ms

- Since the last two operations are 'read' operations they do not conflict. However, one of them may be interrupted by the other one.
- Between operations #1 and #3 there is a conflict: during reading the data may be overwritten.

Note that the writing/reading of scalar values[1] are atomic operations. In the last situation it is therefore justified to allow the overwriting: the update of the angles for control is more important than the logging to ground. Of course, this means that some of the triples of Euler angles sent to the monitoring system are not consistent, i.e. at times the three angles may not belong to the same time instant.

One last situation which does not occur in our example is a writing operation interrupted by a reading operation. This may be the case if the heading angle delivered by the compass is used in the navigation algorithm. The reading operation is then cancelled, and the data is read next time the navigation is executed.

Reference [6] contains further details about the data management.

4 Navigation

To control the dynamics of a helicopter it is necessary to always know its state which consists of position and attitude relative to an earth frame, and their derivatives with respect to time. It might be possible to design a controller based on a subset of the chosen state. On the other hand it is also possible to augment the state vector and improve the controller by considering the extended model. The inclusion of certain dynamics of the rotor, the engine, the surrounding air etc. are possible extensions. Our choice is based on mechanical insights and on experience and has proven to lead to a working controller.

The problem of measuring the state leads to an estimation problem because

- some of the desired variables cannot be measured directly
- the measurements are noisy / biased / delayed
- the sensors / measurements have limited bandwidths.

In the most general filter structure all available sensor data is fed synchronously into the filter which then delivers an estimation of the whole state at once, including estimations of sensor bias etc. Such a filter may take into account all conceivable effects of interaction between all the signals involved. Unfortunately, such a general approach leads to a multitude of parameters for which it is very hard to find meaningful values.

For the time being, we concentrate on a simpler filter with the following structure[2]:

- In a first step, we estimate the attitude, based on acceleration and angular rotation rate data delivered by the inertial navigation unit and the heading sent by the compass.

1. In Oberon SA, both INTEGER & REAL types consist of 4 bytes.
2. Some of the necessary 'small' steps (e.g. coordinate transformations for rotations) are omitted.

- In a second step, velocity and position of the helicopter are estimated taking GPS data into account.

Some sensor data is sampled synchronously with the call of the navigation filters. Others, such as compass or GPS data, are delivered asynchronously with smaller update rates. In such cases, the navigation algorithm only makes use of refreshed data.

4.1 Attitude Filter

The attitude is given by the three Euler angles Φ (roll), Θ (pitch) and Ψ (yaw). To estimate the Euler angles we make use of three types of sensors: compass, accelerometers, and gyros. All sensors are mounted in a strapdown way (as opposed to using a gimballed platform). We are using triples of accelerometers and gyros which are arranged orthogonally.

One method of calculating the Euler angles is an integration of the rates provided by the gyros. Using suitable transformations, numerical integration propagates the estimation of Euler angles through time[1]:

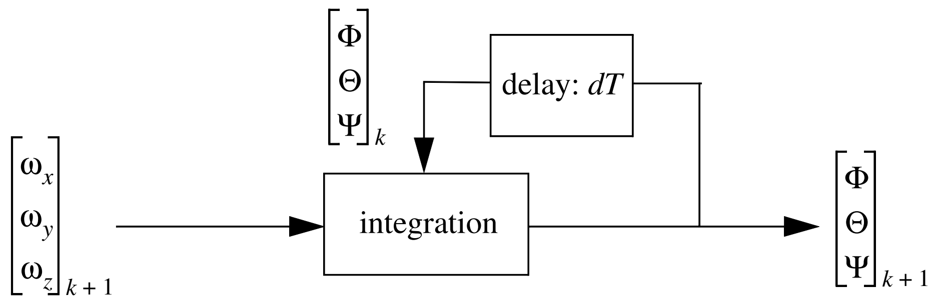

Fig. 4.1. Euler angles calculated using rates of rotation

An alternative way to estimate the Euler angles is based on the assumption that the helicopter is not accelerated with respect to an earth frame. Then the three accelerometers measure the components of the gravitational vector with respect to a helicopter frame.

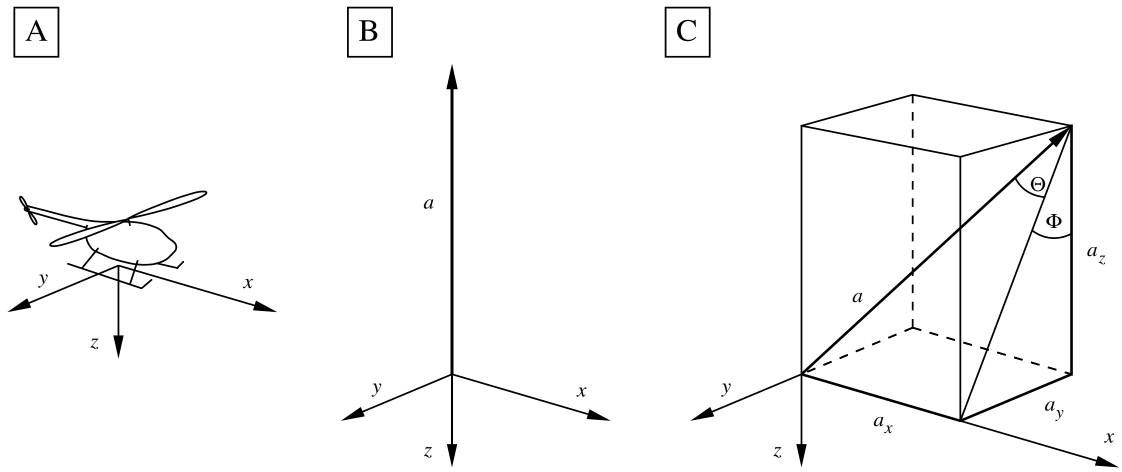

Fig. 4.2. Measuring the vector of gravitation

1. The indices k, k+1, ... correspond to the points of an equally spaced time axis. The time increment dT is normally 5 ms in navigation algorithms and 20 ms in control algorithms.

The axes of the helicopter frame are shown in Fig. 4.2, A. In the case of horizontal hovering the x- and y-components of the measured acceleration are equal to zero: $a_z = -g$ (Fig. 4.2, B). If the helicopter is inclined, the measured — gravitational — acceleration is also inclined with respect to the helicopter frame. Some trigonometry may then be used to transform the acceleration vector to roll and pitch angles[1]:

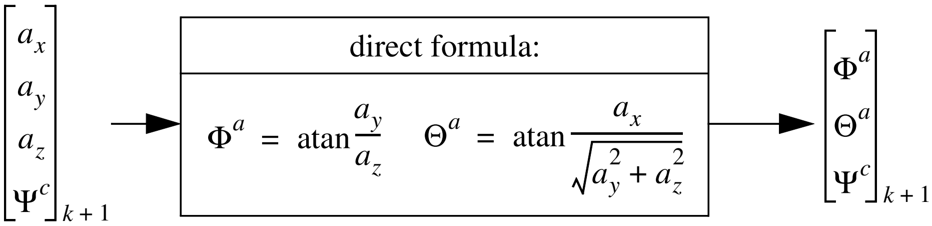

Fig. 4.3. Euler angles calculated using track over ground and rates of acceleration

Both of the methods sketched have their advantages and drawbacks. While the integration method is also usable during real flight, it is prone to errors caused by sensor drift and sensor bias. The direct (i.e. static) method yields wrong results if the helicopter is accelerated. Biased sensor data, on the other hand, does not lead to drifts in the estimated angles.

There are several approaches toward combining the two methods. The extended Kalman filter, a sophisticated filter with a theoretic background in stochastic signal theory, is described in detail in [7] and [8].

A simpler way of fusing the data sets is mixing them: support the Euler angles obtained by integration by building a weighted sum with the other set of angles:

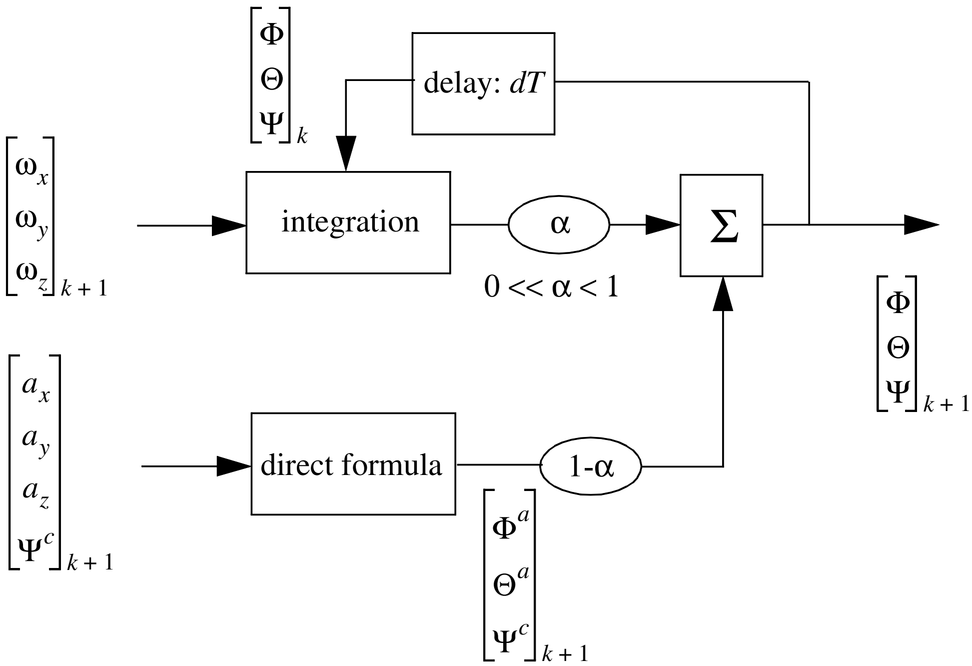

Fig. 4.4. Euler angles obtained by the mixing technique

1. See also Fig. 5.2 for details concerning the orientation of the rotation angles.

4.2 Position Filter

The basic situation is the same as in the case of the attitude estimation: we have redundant information for the determination of position and velocity. The acceleration vector, integrated twice over time, delivers velocity and position whereas GPS messages yield velocity and position directly[1].

The advantages and drawbacks of the sensors are comparable to those mentioned above in the case of attitude estimation: If the acceleration signal is biased by a constant, then the estimations of velocity and position drift away (Fig. 4.5). The drift terms are linear or quadratic with time, respectively. GPS data on the other hand is obtained absolutely, but with lower frequency, with more delay, and with the values possibly being temporarily unreliable.

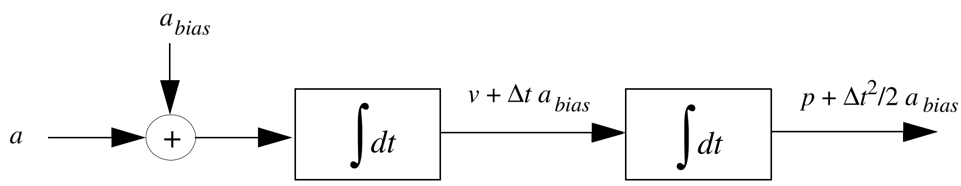

Fig. 4.5. Biased measurement of acceleration

The data fusion for the position estimation is performed by a Kalman filter. The dynamic system shown in Fig. 4.5 may be formulated in a linear discrete-time time-invariant state-space form

$$x_{k+1} = Fx_k + G(u_k + w_k) \qquad (1)$$
$$y_k = Hx_k + r_k \qquad (2)$$

where the vector-sized variables x_k, u_k, w_k, y_k and r_k are state, input, input noise, output and output (sensor) noise, respectively, of the system at time k; the matrices F, G, and H are its system matrix, input matrix, and output matrix. In our concrete example the first equation results in

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ v_x \\ v_y \\ v_z \end{bmatrix}_{k+1} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & dT & 0 & 0 \\ 0 & 1 & 0 & 0 & dT & 0 \\ 0 & 0 & 1 & 0 & 0 & dT \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{F} \begin{bmatrix} p_x \\ p_y \\ p_z \\ v_x \\ v_y \\ v_z \end{bmatrix}_k + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ dT & 0 & 0 \\ 0 & dT & 0 \\ 0 & 0 & dT \end{bmatrix}}_{G} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_k \qquad \text{(1a)}$$

1. Note that this argumentation only holds if the helicopter does not rotate or if the effects due to rotation are eliminated by suited transformations.

If the position p is taken as the output of the dynamic system, then the second equation — without noise term — yields

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}_k = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}}_{H} \begin{bmatrix} p_x \\ p_y \\ p_z \\ v_x \\ v_y \\ v_z \end{bmatrix}_k \tag{2a}$$

As shown previously, the estimated state x_k drifts away from its real value if calculated by the 'open-loop' equation (1a). The idea behind an observer is to let the estimated state converge to the real one by introducing a correction term

$$x_{k+1,\,corrected} = x_{k+1} + L_{k+1}(y_{k+1,\,measured} - y_{k+1}) \tag{3}$$

The scheme of the state estimation then is the following:

- In every time step dT the measured acceleration leads to an extrapolation step according to equation (1a), yielding a new state x_{k+1}.
- In case the GPS system delivers a new position, the state x_{k+1} is corrected according to equations (2a) and (3). The position measured by the GPS is taken as $y_{k+1,\ measured}$. Finally, x_{k+1} is replaced by $x_{k+1,\ corrected}$.
- If the GPS system delivers a velocity message, then the update is similar. A modified output equation — i.e. modified matrices H and L_k — is used.

There is a number of ways for choosing the observer or, in other words, for deciding how L_k is updated. The Kalman filter, as a special case of an observer, is based on the stochastic properties of our process. We assume that the input noise w_k and the output noise r_k are white, that both types of noise as well as the initial state x_0 are unbiased, i.e. $E\{w_k\}=0$, $E\{r_k\}=0$, $E\{x_0\}=0$, and that the respective covariances are given by the symmetric matrices $E\{w_k w_k^T\}=Q_k>=0$, $E\{r_k r_k^T\}=R_k>0$, and $E\{x_0 x_0^T\}=\Sigma_0>=0$. For practical purposes it is often sufficient to reduce these matrices to (block) diagonal form, thereby neglecting cross correlation terms and weighting only the relative levels of noisiness of different signals.

A full description of the Kalman filter for the position is given in [9]. Note that this is a simpler filter than the *extended* Kalman filter used for the estimation of attitude. The reason is the underlying model of the system which is *non*linear in the case of the attitude filter.

5 Control

5.1 General Remarks

Briefly listed, the aims of the controller design are:

- stabilization of the open-loop system
- robust stability performance
- robust tracking performance

Since the helicopter is unstable by nature, the first task to be solved is the design of a controller which stabilizes the 'nominal' model of the helicopter in hover flight. However, the controller must be robust and include some stability margin to cope with model uncertainty, sensor noise, and other imperfections.

When it comes to dynamic flight, i.e. the tracking of trajectories, we have to find a satisfying balance between the tracking abilities of the controller on one hand and the limitation on control signals, the noise suppression, and the robust stability on the other hand.

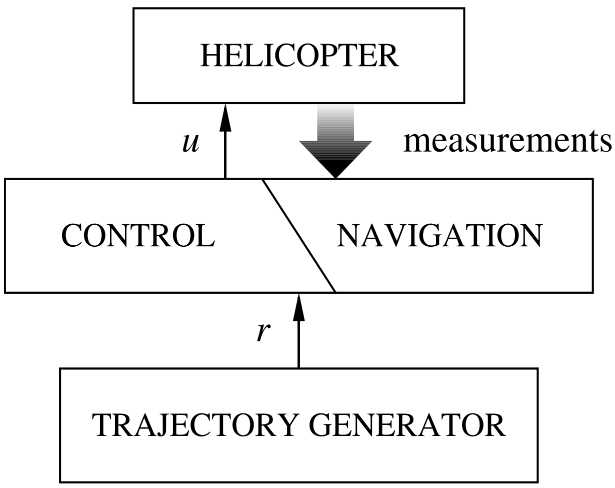

Fig. 5.1. Block diagram of control loop

The signal flow in the block diagram of the control loop (Fig. 5.1) shows the reference signal r and the control output u as 'wired' signals. In some sense, these types of signals are 'trackable' in the system, i.e. they may be measured on a wire or read from a memory address. The input to the controller, on the other hand, is the result of the navigation, a filtering process which takes as input all kinds of sensor data[1].

In order to concentrate on the control aspect, we now introduce 'wired' feedback paths, keeping in mind that this is an idealized situation, and that we actually do not have direct access to these physical variables.

1. Note that most of the sensors are mounted on the helicopter. In our setup, the only exception is the ground antenna of the GPS.

5.2 Model

The controllers we are considering are model-based controllers, i.e. we are using an explicit mathematical description as a model of the helicopter. The model has to cover the essential parts of the dynamic behaviour of the helicopter.

The controller has five outputs: one for the engine, one for the tail rotor, and the remaining three for the swash-plate of the main rotor[1]. The three degrees of freedom of the swash-plate correspond to the three control signals for roll and pitch rotations and for the collective pitch which controls the height. The model thus has five inputs. Building a model basically means gathering the knowledge about relations between the inputs and the internal state, and to formulate them, for instance, in terms of differential equations.

The model on which our first controller is based is quite simple. We consider the dynamics in the various directions[2] to be decoupled (Fig. 5.2):

- u_Φ determines the roll angle of the swash-plate and leads only to rotation around the x axis (Φ, roll) and to translation along the y axis
- u_Θ determines the pitch angle of the swash-plate and causes rotation around the y axis (Θ, pitch) and translation along the x axis
- u_C determines the collective of the swash-plate and leads to translation along the z axis
- u_Ψ — used for the tail rotor — only causes rotation around the z axis (Ψ)
- u_E controls the engine and influences only the rotor frequency

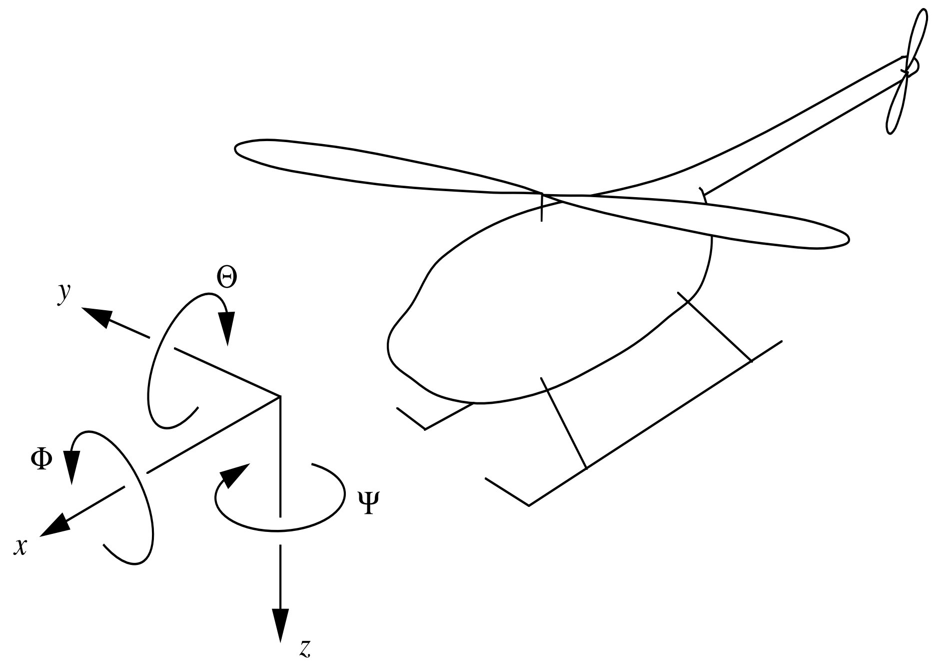

Fig. 5.2. Axes of the helicopter frame

1. Actually we have six outputs in terms of controlled servos; having some redundancy, the swash-plate is controlled by four of them. The four servo signals are calculated by applying a linear transformation to the controller output (roll, pitch, collective).
2. All directions are given in the helicopter frame, therefore the subsequent statements are also valid if the helicopter rotates.

The model described so far leads to five separate submodels. In the one shown in Fig. 5.3 u_Φ controls both lateral rotation and translation. The submodel of the helicopter contains a number of integrators and two unknown blocks, G_1 and G_2. This structure has been found by physical insight and inspection of logged data. It turns out that in frequency domain G_1 may be approximated by a PT_1 transfer function, whereas G_2 is approximated by a constant[1]:

$$G_1 = \frac{K_1}{1 + sT} \qquad\qquad G_2 = K_2$$

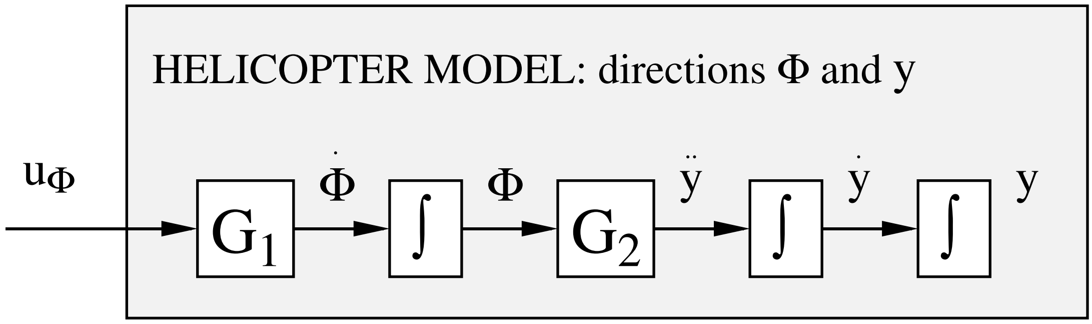

Fig. 5.3. Submodel of lateral movement: rotation and translation

The model built thus far is valid at least in a region around the equilibrium point of hovering. We can easily think of situations in which it becomes invalid, e.g. when due to fast rotation decoupling is no longer allowed.

5.3 Design of the Controller

The decomposition of the helicopter model into a number of decoupled submodels has its counterpart in the design of the control algorithm (Fig. 5.4). It is reasonable to choose the dependencies symmetrically: e.g. since u_Φ is assumed not to affect Ψ, the control law for calculating u_Φ does not make use of the estimated value of Ψ. However, as soon as coupling terms are introduced into the model, it is useful to reflect this change of structure in the controller. The control signal u_Φ is calculated by adding $u_{\Phi 0}$ — originating from operating point conditions — and a weighted sum of signals which represent the errors in lateral motion[2]. The introduction of reference signals in the velocity and position error allows the tracking of trajectories. It is assumed that velocity and position are given consistently. The aim of the controller is to stabilize hovering and ‘slow’ flight. Therefore no references are introduced for the roll angle; the helicopter is flying with a horizontally aligned attitude.

---

1. At this stage of modelling, the MATLAB identification toolbox has been used. G_1 and G_2 are assumed to be linear transfer functions of a certain order containing some unknown parameters. Then the parameters are chosen such that the measured time series fit best. Inspecting the resulting transfer function gives some clues as to whether its order has been chosen too high, as e.g. in the cases of small time constants or pole/zero-cancellations.
2. The acceleration is not included because it is assumed to be proportional to the roll angle.

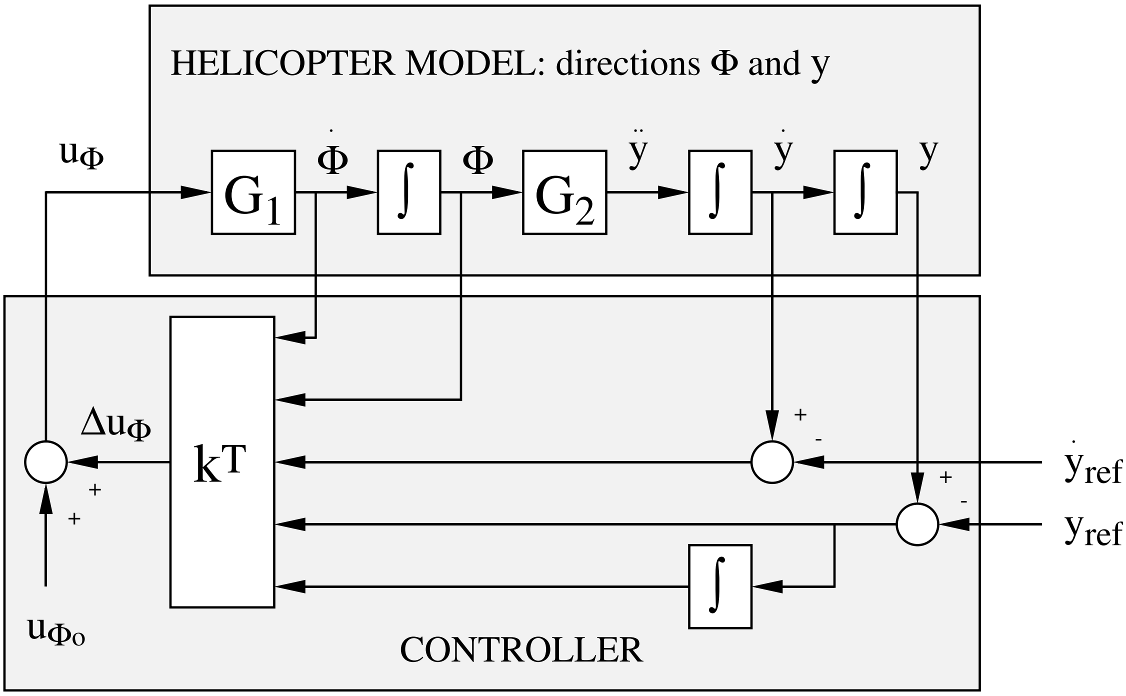

Fig. 5.4. Control of lateral movements

The structure for the other control outputs is similar. The following table lists the error and reference signals that contribute to the various control signals:

Table 3: Connections between reference, error and control signals

REFERENCE SIGNALS	WEIGHTED ERROR SIGNALS	CONTROL SIGNAL
$\dot{y}, y$	$\dot{\Phi}, \Phi, \Delta\dot{y}, \Delta y, \int\Delta y$	u_Φ
$\dot{x}, x$	$\dot{\Theta}, \Theta, \Delta\dot{x}, \Delta x, \int\Delta x$	u_Θ
$\dot{z}, z$	$\Delta\dot{z}, \Delta z, \int\Delta z$	u_C
Ψ	$\dot{\Psi}, \Delta\Psi, \int\Delta\Psi$	u_Ψ
Ω	$\Delta\Omega, \int\Delta\Omega$	u_E

5.4 Implementation of the Controller

The minimal structure of the five subcontrollers is not transferred to the implementation directly. Instead, a full version of the feedback equation

$$\Delta u = K\Delta e$$

with $\Delta u \in \Re^5$, $\Delta e \in \Re^{18}$, and $K \in \Re^{5x18}$ is implemented offering support for more general controllers. Using this structure for the decoupled controller means that the *K* matrix just contains 18 relevant elements, all other a priori being equal to zero.

Matrix multiplications are widely used not only in the control but also — and even more often — in the navigation algorithms. For the sake of efficiency the following facilities have been integrated into the compiler [2]:

1. Leaf procedures

   A procedure that does not contain any procedure calls may be declared as a leaf procedure. Parameters of leaf procedures are kept in their registers.

2. Register variables

   Variables used in the scope of a leaf procedure may be declared as register variables. Such variables are allocated in registers rather than in memory.

3. Riders

   Access to array elements is often realized by indexing. The rider concept makes use of the fact that often the index is increased or decreased by a constant value between consecutive accesses. Riders may be set to certain positions. After each access they are moved a fixed stride. There is a test to check whether a given index limit has been reached.

Examples of the all those concepts are given in [2]. Of course, matrix multiplication is not the only task deriving benefits from them.

Another aspect of the implementation, the numerical correctness, is not as critical in the control algorithm as it is in the filter algorithms [8]. Beyond that, the errors due to the control algorithm are outnumbered by the errors already contained in its input data.

As a last aspect of the implementation we have a quick look at some subtleties which are necessary to overcome the drawbacks of the given algorithm. As soon as an integral part is present in a controller we have to deal with the so called wind-up effect. This effect may occur in the following way: a long lasting error — due to a crosswind for example, which prevents the helicopter from moving to its reference position — 'fills' the integrator and brings the control signal in a range which is beyond the actuator's capabilities (Fig. 5.5). Once the position error

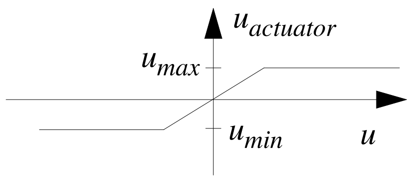

Fig. 5.5. Reaction of actuator to control signal

disappears, the controller tends to compensate the previously accumulated error: the integrator first must be depleted before the control signal returns to its range of operation. To prevent this effect a patch (anti-wind-up) is applied to the control algorithm: the state (i.e. the value) of the controller's integrator is always clipped such that the resulting control output lies within a predefined range[1].

1. Note that a pure clipping of the control *output* does not change the situation.

As a side effect, this anti-windup setup offers an elegant method to continually increase the effect of the controller in the control loop. In the phase of testing a new set of control coefficients, the range of allowed control signals may first be chosen rather small, and then gradually be opened.

6 Trajectory Generator

Whereas in the case of hover flight the generation of reference signals is quite a trivial task, we also need to create reference signals for 'real' flight. For instance, we wish to formulate flight manoeuvres on an abstract level, such as

- start, i.e. take off to a certain height above ground
- fly from point A to point B
- search a given field (e.g. defined by four points as corners).

The process of converting the manoeuvres to reference points includes the following steps:

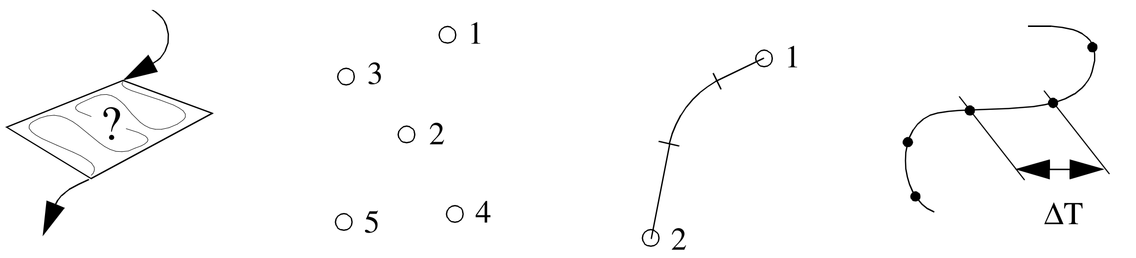

Fig. 6.1. Levels of abstractions for trajectories

- Waypoints are points which must be contained in the trajectory.
- Segments are parametric descriptions of parts of a trajectory. For example, we defined a segment of constant velocity, a segment of constant acceleration etc.
- Reference points contain the reference input to the controller (position, velocity, heading).

By passing the transformation steps from manoeuvres down to segments, a number of parameters and constraints are involved:

For instance, in a search manoeuvre we want to have a certain grid width, height over ground, maximal velocity, maximal acceleration, maximal rotational rate of the heading angle, and maximal time to accomplish. Some of these items may even be refined further, such as acceleration in different directions: horizontal vs. vertical, radial vs. tangential. Other requirements concern the grid itself: do we want to fly exactly on the grid or do we just want to somehow cover the whole field?

The current state of development is the following: sequences of segments are calculated on the ground, then sent to the helicopter. On board, segments are collected in a flightplan; reference points are extracted from the current segment. This allows a very careful testing of the strategies involved in the generation of segments and reference points.

The calculation of segments happens in an environment written in MATLAB. For the time being we are gaining experience with different types of manoeuvres and with various types of algorithms for transforming them to segments. Strategies which prove to be useful are then integrated into the board system.

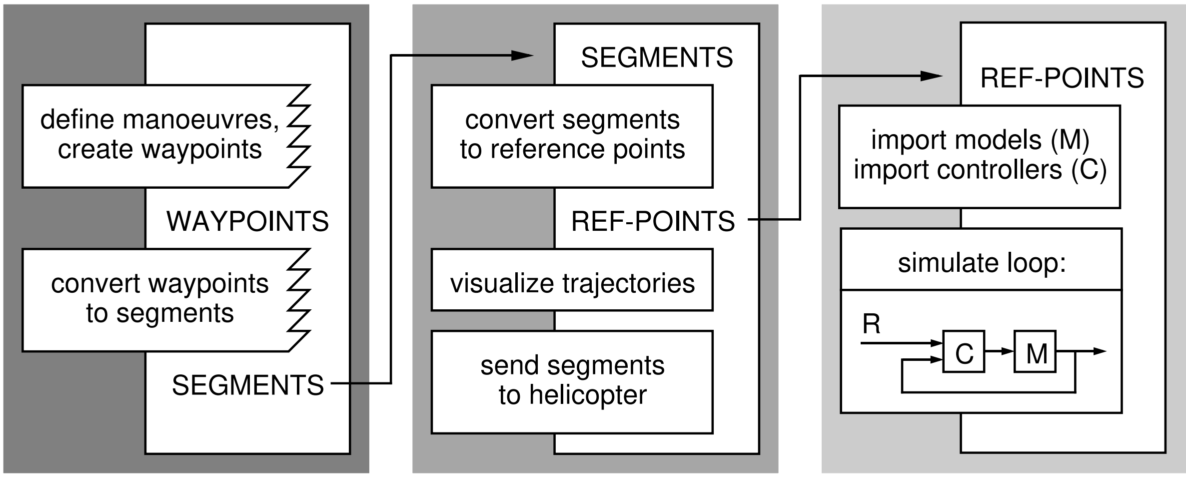

Fig. 6.2. MATLAB environment for the development of trajectories

The MATLAB environment consists of three main blocks, each of which is realized in a separate graphical user interface.

1. The purpose of the first block is the creation of segments. Both the definition of manoeuvres and the conversion of waypoints to segments are possible in a kind of plug-in style. This results in a very flexible way to develop and test new strategies.
2. The second block is mainly for the verification of resulting sequences of reference points. The algorithm used to calculate the reference points is the same as the one used on-board. The calculated reference points are visualized, and the consistency of position, velocity and heading angle is clearly visible. Segments can also be transferred to the helicopter.
3. The last block is used for the simultaneous simulation of trajectories, models of the helicopter, and controllers. For instance, a controller may be tuned to certain types of trajectories, or vice versa. One possibility not yet exhausted is the improved handling of the knowledge of future reference points.

7 Finite State Machines

The navigation and control part described so far deals only with the situation when the loop is up and running. To have a refined control over various partial aspects of the helicopter, we want to introduce some special control actions, as e.g:

- start align phase of navigation (calibration of accelerometers)
- start take-off (reference for rotor frequency is increased)
- activate / deactivate engine controller

- activate / deactivate control of main rotor and tail rotor

Some of the actions need more work than seems to be necessary at first sight. Switching a controller on and off, for example, does not just mean to enable and disable the feedthrough from the controller output to the actuators. Since the control algorithm normally contains an integral part, the control signal is likely to saturate. Therefore, at the time of reactivating the controller, a reinitialization of the controller is needed to ensure a bumpless control signal.

To support the handling of these logical states, finite state machines have been introduced.

8 Further Development

8.1 General Ideas

The system described so far of course offers a certain potential of improvements in the area of ‘fine-tuning’ the existing navigation and control algorithms. But beyond that a large number of challenging problems are waiting to be solved. From a control engineer’s point of view, they include:

- fast and precise flight along ‘demanding’ trajectories
- flexibility in initiating and interrupting manoeuvres
- ability to land with autorotation (with engine turned off)

This leads to a number of problems which must be solved first, e.g.:

- an improved model of the helicopter must be identified which includes the coupling effects among the various directions
- the navigation algorithms must be robust against the effects of dynamic flight, especially the fact that the acceleration no longer is ‘constant’
- model predictive (or a least reference predictive) features must be included
- a combined control of engine and trajectory tracking also is advantageous in the sense of timely reactions.

8.2 Varying Number and Quality of Sensors Used

The system described in the previous chapters relies on a large set of expensive sensors. At the moment we are assembling a simpler system which makes use of a reduced set of sensors, namely a low-cost inertial navigation unit consisting of three gyros and three accelerometers. Using an adapted version of the attitude filter, an estimation of the Euler angles is possible. This allows to control the attitude but not the position. Therefore, such a controller cannot prevent the helicopter from drifting away over time.

Such a system is ideally suited for the inexperienced (human) pilot. Pilot and autopilot complement each other in that their control signals are mixed together: while the pilot perhaps has trouble stabilizing the attitude, it is easy for him to control the position.

In a current term paper a small differential GPS system is being developed. The correction signal used is a radio signal. Compared to the 'local' high quality correction signal this 'global'[1] lower quality signal leads to a precision in the range of just meters.

8.3 Geographical Information Systems

The current helicopter system works in environments for which certain assumptions hold. For instance, the flight field is more or less flat. This simplifies manoeuvring.

To make the helicopter system more versatile, some kind of geographical information system needs to be integrated into the existing trajectory generation software. As a first approach this may be a grid model (e.g. on a 25-meter base), while in more sophisticated versions we are interested in objects such as power lines or ropeways.

9 References

[1] Wirth, N. (1998): *A Computer System for Model Helicopter Flight Control, Technical Memo Nr 1: The Hardware Core.* Technical Report # 284, Department of Computer Science, ETHZ.

[2] Wirth, N. (1998): *A Computer System for Model Helicopter Flight Control, Technical Memo Nr 2: The Programming Language Oberon SA.* Technical Report # 285, Department of Computer Science, ETHZ.

[3] Sanvido, M. (1999): *A Computer System for Model Helicopter Flight Control, Technical Memo Nr 3: The Software Core.* Technical Report, Department of Computer Science, ETHZ.

[4] Chapuis, J., Eck, C., and Kottmann, M., (1998) *Modellhelikopter wird zum High-Tech-Werkzeug.* SEV Bulletin, 3/1998, S.17-20.

[5] Chapuis, J. and Kottmann, M., *Specifications of Communication Protocol.* Internal Documentation, IMRT/ICS, v1.8, ETHZ.

[6] Kottmann, M. (1998): *Datenverwaltung auf dem Helikopter.* Internes Memo am ICS, v2.0, ETHZ.

[7] Eck, C. (1998): *Kalman Filter.* Internal Report at IMRT, ETHZ.

[8] Kottmann, M. (1998): *Implementation of Navigation Algorithms.* Internal Report at ICS, v1.0, ETHZ.

[9] Geering, H. P. (1996): *Kalman-Filterung.* Interner Bericht am IMRT, v2.0, ETHZ.

[10] Kottmann, M. (1999): *Trajektoriengenerierung für den Helikopter.* Interner Report am ICS, v1.0, ETHZ.

The technical reports [1]-[3] are available in electronic form at

http://www.inf.ethz.ch/publications/tech-reports/

1. 'Global' at least within the frontiers of Switzerland. The VHF signal used is provided by the Federal Office of Topography (Bundesamt für Landestopographie).