UAV Flight Experiments with a RT-Linux Autocode Environment including a Navigation Filter and a Spline Controller.

Karl Kufieta[1], Christian Wickbold[1] and Prof. Vörsmann[1]

Institute of Aerospace Systems, TU Braunshschweig, Braunschweig, Germany
`karl.kufieta@tu-braunschweig.de`

Abstract

A common method for navigation in Unmanned Aerial Vehicles is the fusion of the GPS-position and an Inertial Measuring Unit. For this data fusion a Kalman Filter is used. To guide and control the Unmanned Aerial Vehicle, spline paths in combination with linear or nonlinear controllers are suitable. The development of these algorithms is iterated cyclically beginning with a simulation and then testing the algorithms on the hardware. To be able to repeat fast development cycles an autocode environment in combination with a real-time Linux has been developed. This paper describes the performance of the used autopilot architecture in the scope of algorithm development. Finally the recorded flight test data is compared to the simulation data.

1 Introduction

Unmanned aerial vehicles (UAV) as depicted in fig. 1 are a testbed for a wide field of applications. Research at the ILR (institute of aerospace systems) ranges from navigation and control algorithms, over modeling and parameter identification to flight missions, e.g. meteorological [11] campaigns.

Figure 1: Left: Arctic mission with Unmanned Aerial Vehicle, Right: experimental Multiplex Twinstar with mounted autopilot

Today's microprocessors are powerful and suitable for complex algorithms. As those processors are small and energy efficient they can be used in small UAV's. Usually a processor is programmed in a high level language like C or C++. Complex algorithms for navigation and control of UAV's or robots in general are often programmed in object oriented simulation environments like Matlab Simulink. In these environments signals are represented by lines between operators (e.g. plus, minus, transfer function), sources (e.g. constant, sensor) and sinks (display, actuator). To implement an algorithm from the simulation world, the vendors (like Mathworks) have designed code generators. With a code generator it is possible to translate the object oriented simulation into C-code. This process can be automated so far that it is possible to program the processor with only one mouse click directly from the simulation. This way the implementation cycle times for different types of algorithms are reduced dramatically. As different algorithms can run in different sample times a real-time Linux with the ability of multitasking has been considered. Following a closer look at the hardware the main points of the real-time Linux are briefly explained. The execution times of the Kalman navigation filter and the spline controller are elaborated and the necessity of multitasking is explained in this scope. Starting with a simulation of the airplane with navigation and control the complete simulation can be tested as software in the loop on the autopilot hardware.

A result of the symbiosis of all components is that complex algorithms from a high-level GUI environment can be elaborated in flight tests, as the programming from the object oriented high-level GUI interface to the ready to flight UAV takes around 30 seconds. At the end of this paper, results of the navigation solution and control algorithms are shown in experimental flight test.

2 The Autopilot System

The autopilot hardware (see fig. 2) is based on a two processor concept. As the main computing processor an OMAP3530 [8] is used, which offers a good computing performance with 720 Mhz and a floating point unit. This main computing processor is connected to the data acquisition processor STM32 [14] with 72 Mhz and the capability of hardware interfaces like the CAN-Bus.

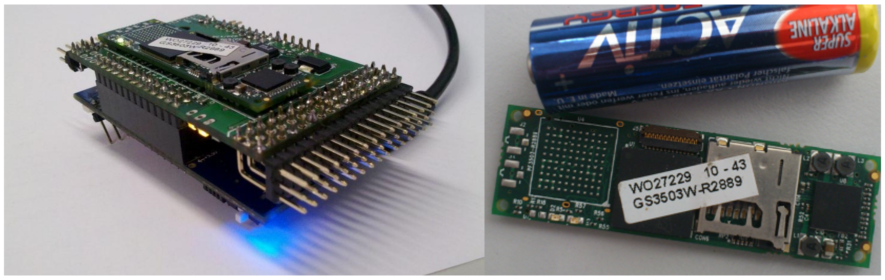


Figure 2: Left: Autopilot Hardware, Right: OMAP3530 mounted on a gumstix board

As the STM32 uses no operating system, its software uses only one thread. Hardware interrupts can be programmed directly beneath the main processor loops and no high level driver need to be used as the hardware can be programmed directly through the processor registers. Thus the STM32 software setup is relative simple to the OMAP3530 which uses Linux as operating system with multithreading for complex algorithms and high level drivers for the peripheral interfaces. In the case of an OMAP3530 failure, e.g. an experimental algorithm crashes the system, the STM32 will always be able to take control over the airplane. As all actuators and sensors are connected to the STM32 (see fig. 3) and as a high update rate for algorithms is preferable, also a high data bandwidth between the STM32 and the OMAP3530 is of benefit. For this reason an SPI (serial peripheral interface) Bus has been chosen which allows bidirectional 20 MBit/s which is sufficient for the transmission of 50 signals for sensors and actuators in both directions within 300 μs.

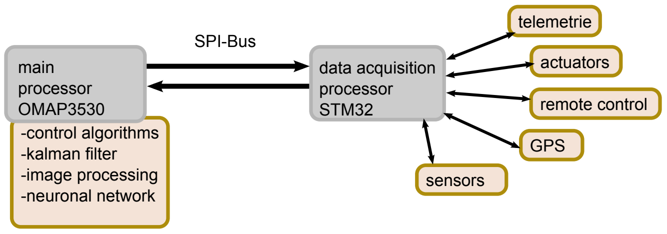
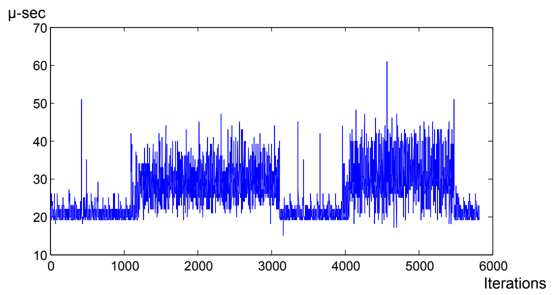


Figure 3: Connection scheme between processors

2.1 Operating System

The OMAP 3530 processor is running a real-time Linux [1] [13] which allows the usage of multithreading and complex drivers like a TCP stack or WiFi drivers. The fastest thread, i.e. the navigation and control algorithms, are running with an update rate of 100 Hz. This means that this thread must be executed deterministically within 10 ms, which leads to the most important variable of the real-time Linux, the execution delay: A standard Linux kernel behaves non deterministically [2] and schedules the given tasks, respectively the threads in a fair manner. Although the tasks are executed as fast as possible, it is not predictable when or how long they are executed. To overcome these problems a real-time layer has been integrated by the kernel developers which gives the ability to run high priority tasks with a deterministic execution behavior. Figure 4 shows the bounded the thread execution without real-time priority that can be very high whereas with real-time the delay is bounded below $< 65\mu$.

The STM32 runs its own independent 100 Hz loop. If the OMAP3530 provides a SPI-connection, the STM32 will transfer the sensor data to the OMAP3530 and use the computed control commands for the actuators (compare 4). The OMAP3530 runs several threads for different frequencies, e.g. the Kalman filter for navigation with 100 Hz. Non real-time threads can be executed in parallel e.g. to run a terminal via WiFi.



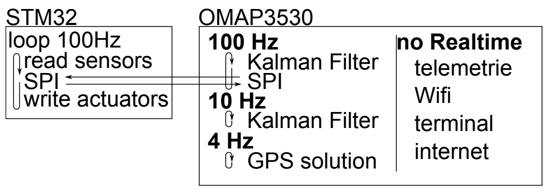


Figure 4: Left: The execution delay is bounded below $< 65\mu$, Right: Different threads running on the two processors

2.2 From Simulation to Flight Experiment

Coming from the hardware and software description, the concept of the autocode [12] design is elaborated. The algorithms for navigation and control are designed in the object oriented simulation environment Matlab/Simulink. Figure 5 depicts a simple simulation where a sinus signal is added to a constant matrix and the result is plotted in a "scope". To run this simulation on the autopilot system a so called tool-chain is used, where the simulation is translated into C-Code and compiled into machine executable binaries. To this purpose for example an "add-block" (compare fig. 5) is translated by Matlab's Target Language Compiler (TLC) with its corresponding tlc file into a C-Code function. In this way the simulation with all its blocks

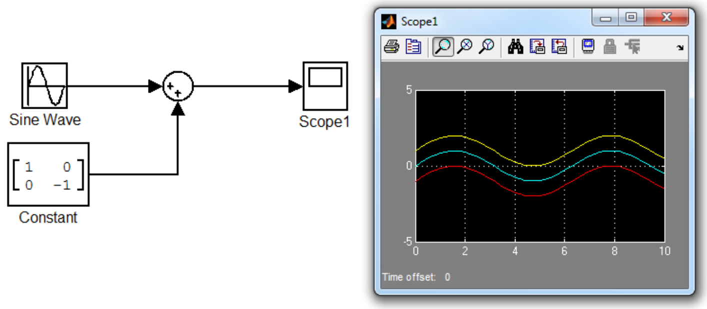


Figure 5: A simple Simulink model

creates a C-Code program, e.g. airplane.c (compare fig. 6). In a last step the compiler tool-chain for the specific processor translates the C-Code into a machine readable binary. Hardware like actuators or sensors can be modeled as a simulink block and after programming the autopilot this hardware is directly used. With the so called "external mode" it is possible to link and monitor the simulation on the real hardware via WiFi in the "scopes" of the developement hardware.

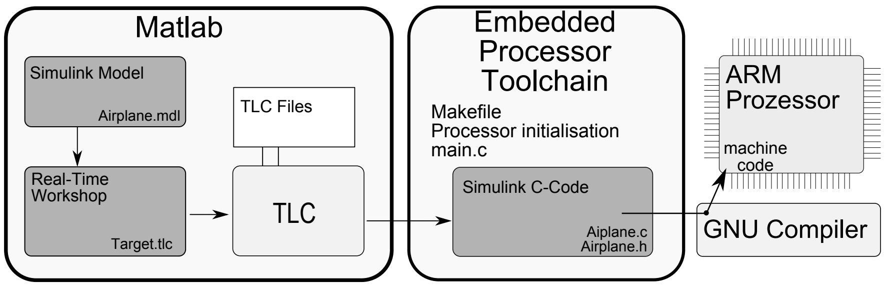


Figure 6: Autocode Environment: from model to machine code

2.3 Simulation

Before going into the flight experiment the navigation and control algorithms are tested in the simulation. The simulation is built out of five main parts (fig: 7): airplane, sensors, navigation, control and actuators. The main part is the airplane, where actuator signals and wind simulation influence the airplane's forces and moments, which are integrated twice into position and attitude [3]. The airplane variables (position (4 Hz), rotational velocities, accelerations) are measured by the sensors. This sensor simulation includes sensor noise, temperature drifts or position errors. With this data the navigation reconstructs an attitude signal and a 100 Hz position for the controller cascade. The controller compares the actual position with the desired flight path and generates control signals for the actuators. The actuators are usually servos or the engine, which includes delays and nonlinearities from the input signal to the outgoing force. The connection to the airplane closes the loop.

In the development process the simulation is tested on the development environment: Running the simulation here takes less than a second and the simulation is running faster than real-time. If the algorithms perform good in the simulation, they are compiled and programmed onto the autopilot and tested in real-time, which is usually called software in the loop.

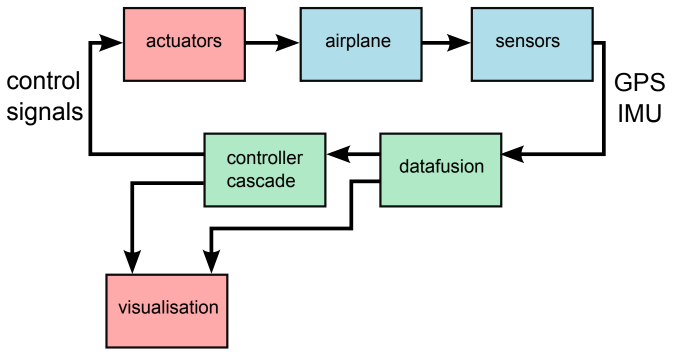


Figure 7: Airplane simulation with navigation and control

2.4 Kalman Filter

To control the airplane a precise position and attitude signal is necessary. As the dimensions of the small UAV require small and lightweight hardware, Microelectromechanical (MEMS) sensors are used in the Inertial Measuring Unit (IMU). The used low-cost acceleration and gyro sensors operate with a frequency of 100 Hz but suffer from drifting signals.
To compensate this drifting, long-term stable 4 Hz position measurements of the the GPS receiver are used. In this way position, velocity and attitude can be determined at the rate of the IMU with a better precision than with a stand-alone GPS receiver (fig: 8).

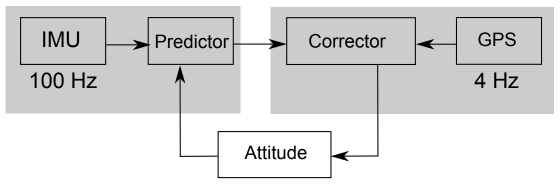


Figure 8: Kalman Filter

Different concepts for the fusion of sensor data as from GPS and IMU measurements exist (see [6]). The simplest and therefore most common integration is the so called loosely coupled system. These systems use the position and velocity information of the GPS to bind the IMU measurement errors [5].

The Kalman filter, introduced in [10], has become a quasi-standard for accomplishing the data fusion of inertial and satellite navigation. The navigation system of the new autopilot system is based on a time discrete, loosely coupled extended Kalman filter, see [4]. The error state architecture allows the estimation of a non-linear process with a linear Kalman filter.

The utilized state vector $\vec{x}$ consists of 15 states for errors in position, velocity, attitude, gyro bias, and accelerometer bias.

$$\vec{x} = \begin{bmatrix} \delta r_g^{eb} & \text{... 3 position errors} \\ \delta v_g^{eb} & \text{... 3 velocity errors} \\ \delta \phi^{gb} & \text{... 3 attitude angle errors} \\ \delta \omega_b & \text{... 3 errors in gyro bias} \\ \delta a_b & \text{... 3 errors in accelerometer bias} \end{bmatrix}$$

The indices g and b show that the vector is described in the geodetic- and the body-frame respectively. The superscripts and stand for the body- with respect to earth-centred earth-fixed (index e) (ECEF)-frame and b -with respect to -frame, respectively.

The Kalman filter works in two phases - propagation and estimation, which justifies the need of multithreading. The propagation is executed at the IMU's measurement frequency of 100 Hz. Parallel to the propagation process, the navigation solution is calculated using the IMU measurements. These are processed via a so-called strapdown algorithm, which allows the computation of navigation data from body-fixed inertial sensors, see [9].

The estimation process is started when new GPS measurements have arrived. During this update, the measurements $\vec{z}_k$ which are received at time t_k are processed. This vector consists of the differences between predicted and measured values of the position.

A hat $\hat{}$ indicates values which are re-estimated, measured values are denoted by a tilde $\tilde{}$ on top of each variable. The enhancement of the navigation data quality is performed by the calculation of the so called Kalman Gain matrix and the following update of the state vector and the covariance matrix of the state estimation uncertainty.

$$K_k = P_k^{-1} \cdot H_k^T \cdot (H_k \cdot P_K^{-1} \cdot H_k^T + R_k)^{-1}$$

$$\hat{\vec{x}}_k^+ = \vec{x}_k^- + K_k \cdot (\tilde{\vec{z}}_k - H_k \cdot \vec{x}_k^-)$$

$$P_k^+ = (I - K_k \cdot H_k) \cdot P_k^-$$

The Kalman Gain matrix weights the difference between measurements $\vec{z}_k$ and the expected measurement, which is calculated by the a priori state vector and the measuremnt matrix $\underline{H}_R$. For this it includes the autocovariance matrix of the measurement noise $\underline{R}_k$. The measurement matrix maps the state vector onto the measurement vector. Matrix $\underline{I}$ is a 15 $\times$ 15 identity matrix.

2.4.1 Sensor Calibration

Some attention needs to be given to the sensor calibration. The actual autopilot uses an IMU3000 Gyro from Invensense and an Analog Devices ADXL345 acceleration sensor. Those sensors deliver an unsigned binary number proportional to the measured variable. The sensor output $w_{i,meas}$ of the gyroscopes has been modeled as follows:

$$w_{i,meas} = w_i * a_{w,i} + b_{w,i} + c \cdot T \tag{1}$$

where w_i denotes the angular rate for an axis i , which is multiplied by a scaling factor $a_{w,i}$. An static bias $b_{w,i}$ is added to a temperature depended bias $c \cdot T$. The used accelerometers have been modeled without the temperature dependent bias with the static bias $b_{a,i}$ and the scaling $a_{a,i}$ to the equation:

$$a_{i,meas} = a_i * a + b. \tag{2}$$

Simulation shows that this sensor model is sufficient for navigation and control of the UAV (compare fig. 9). The sensor calibration followed a simple scheme:The autopilot is rotated on

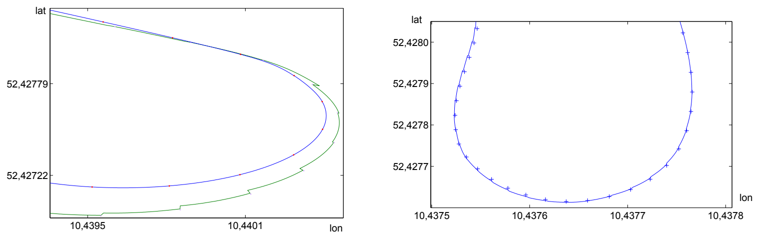


Figure 9: Left: navigation simulation with no sensor errors (blue), navigation with 30% error in bias value (green); Right: navigation in real flight (line), GPS measurements (dots)

the Desk around every axis with 90°. In a first step the gyro signal is out-biased, which means, the end value of a forward and back rotation on the Desk is known as zero. The signal is scaled to fit 90°, compare figure 10. As there a barely visible temperature effect in the scaling parameter, only the gyro bias is considered. This was done by putting the autopilot into a fridge while the bias drift was recorded.

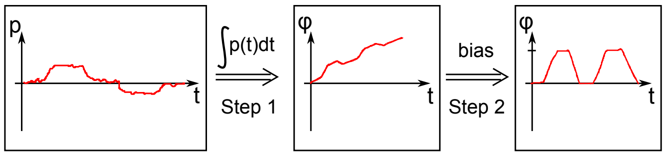


Figure 10: Step 1: Summarize gyro rate signal into angle, Step 2: Calculate bias, Step 3: Scaling

2.5 Spline Control

To be able to follow a predefined path a spline controller is used. To this purpose two-dimensional cubic bezier splines are used. Thus the curves follow a polynome of third degree [7].

The spline can be described with the Point $A_n = [x_n, y_n]$ as follows:

$$x(t) = a_3t^3 + a_2t^2 + a_1^t + x_0 \tag{3}$$

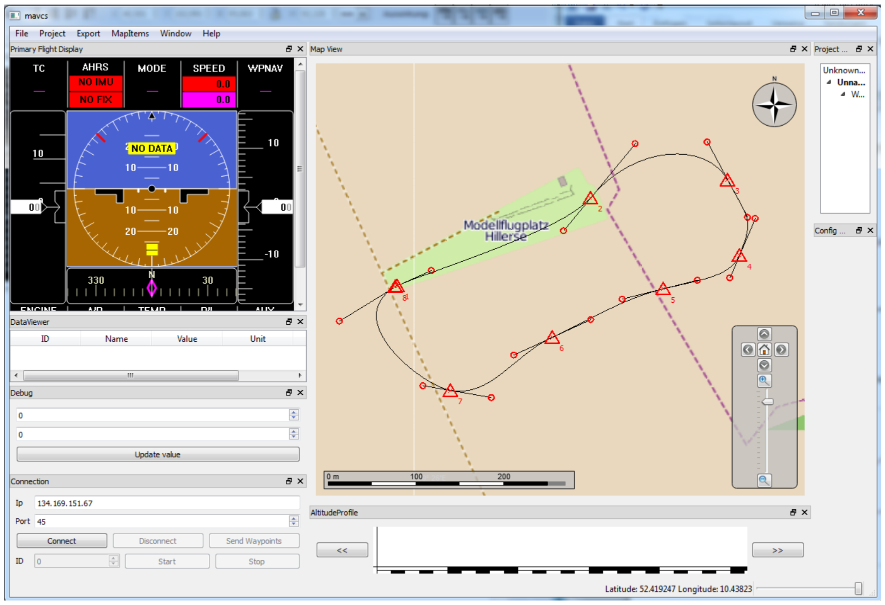
$$y(t) = b_3t^3 + b_2t^2 + b_1^t + y_0 \tag{4}$$

The run parameter t lies in the region between $0 \leq t \leq 1$. The spline parameters are defined as follows:

$$a_1 = 3(x_1 - x_0), a_2 = 3(x_0 - 2x_1 + x_2), a_3 = (-x_0 + 3x_1 - 3x_2 + x_3) \tag{5}$$

$$b_1 = 3(y_1 - y_0), b_2 = 3(y_0 - 2y_1 + y_2), b_3 = (-y_0 + 3y_1 - 3y_2 + y_3) \tag{6}$$

The spline path is preset in the ground-station software (see fig. 11) and transmitted via WiFi or radio to the UAV.



Ip 134.169.151.67
Port 45
Latitude: 52.419247 Longitude: 10.43823

Figure 11: Ground-station GUI with flight display and waypoint navigation map

2.6 Flight experiments

As experimental setup a Multiplex Twinstar airplane model (see fig.1 right) was used. This model has a wingspan of 1.4 meters, and a take off weight of 1.5 kg. The airplane speed ranges from 35km/h to 70km/h. The experiments were made on a moderate gusty day with a wind speed of 15km/h.

2.6.1 Attitude Controller Flight Result

Usually the airplane model operator controls the angular rates of the model with the remote control and thus the elevator and aileron position. In the attitude controller mode the remote control stick positions for the aileron u_{aile} and the elevator u_{ele} are used as reference signals for the roll Φ and pitch Θ angle of the airplane model, such that the aileron and the elevator position $(\varphi_{aile}, \varphi_{ele})$ results directly from this difference:

$$\varphi_{ele} = \Phi - u_{ele} \tag{7}$$

$$\varphi_{aile} = \theta - u_{aile} \tag{8}$$

In the Attitude controller mode taking off and landing the airplane is possible even for novice pilots. Figure 12 plots the aileron signal u_{aile} and the measured roll angle Φ. As the angle is used directly as control feedback, no damping terms are considered which leads to visible overshoots in the angle.

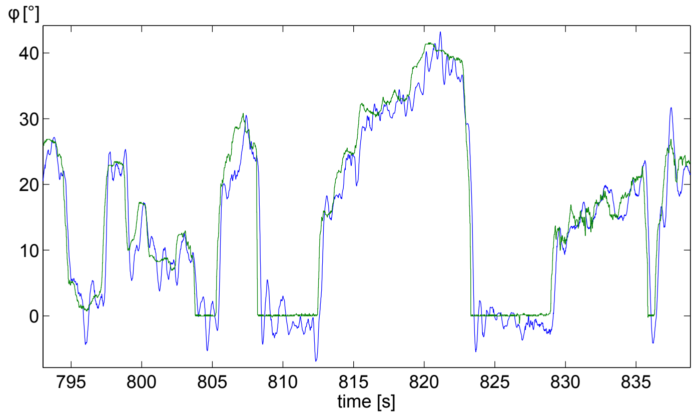


Figure 12: Attitude controller: roll reference signal u_{aile} (green), roll Φ estimated from navigation (blue)

2.6.2 Spline controller flight result

After taking off the airplane in the attitude controller mode, the spline controller is turned on. In this mode the UAV is flying automatically the predefined path. The navigation and control algorithms consume ~ 25% of the processor time on the OMAP3530. Terminal via WiFi or UMTS is always accessible and the flight data is recorded in a non real-time thread. Figure 13 depicts the flight path of the UAV in this experiment. Usually the spline controller is started after takeoff in some distance to the pre-programmed flight path. If the distance to this flight-path (respectively spline-path) is high, the spline controller cascade is going into saturation such that an additional start spline is calcualated automatically in the autopilot from the actual UAV position to the beginning of the predefined flight path.

2.7 Conclusion and Outlook

The combination of Linux and auto-code systems leads to a very effective experimental platform. The development cycles from simulation to the flight experiment are reduced to less than a few minutes. A spline controller in combination with a Kalman filter is running easily on the modern 720 Mhz cell phone processor OMAP3530 and leaves enough resources for e.g. running neural networks. It is expected that actual cell phones with multicore processors and a real-time

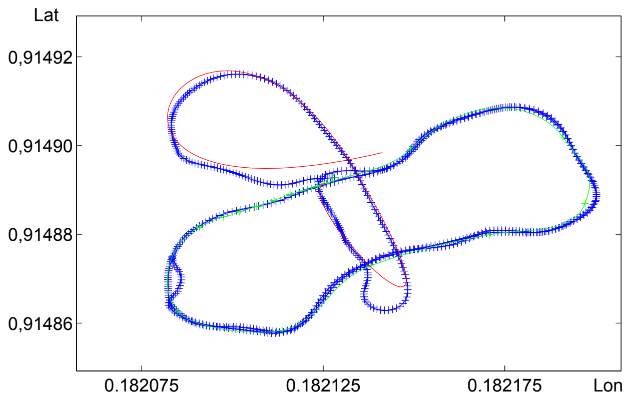


Figure 13: Flight experiment: flight path with GPS measurements (blue line and blue marks), spline (green line), transition spline (red line)

Linux kernel have the capabilities to control UAV's with enough resources for more complex and intelligent algorithms.

References

[1] Stefan Agner. *Linux Realtime-Faehigkeiten.* Hochschule Luzern, $http://www.agner.ch/linuxrealtime/Linux-Realtime-Faehigkeiten.pdf$, 2009.

[2] Cesati M. Bovet P. *Understanding the Linux Kernel.* Oreilly, $http://oreilly.com/catalog/linuxkernel/chapter/ch10.html$, 2000.

[3] W. Alles Brockhaus, R. *Flugregelung.* Springer Verlag, Berlin, 2011.

[4] J. Crassidis. *Sigma Point Filtering for Integrated GPS and Inertial Navigation.* University at Buffalo, New York, U.S.A, -.

[5] J. Weston D. Titterton. *Strapdown Inertial Navigation Technology.* The Institution of Engeneering and Technology, London, 2004.

[6] J. L. Farrell and Barth. *The Global Positioning System and Inertial Navigation.* McGraw-Hill, New York, U.S.A, 1999.

[7] Hans-Georg-Buesing. *Ein Flugregler fuer die Startphase autonomer Kleinstflugzeuge.* TU Braunschweig, Braunschweig, 2006.

[8] Texas Instruments. *OMAP 3530 Application processor.* Texas Instruments, Silicon Valley, 2013.

[9] Fried Kayton, M. *Avionics Navigation Systems.* John Wiley and Sons, New York, U.S.A., 1997.

[10] R. E. Klmn. *A New Approach to Linear Filtering and Prediction Problems.* Journal of Basic Engineering, Baltimore, U.S.A, 1960.

[11] A. Kroonberg. *Airborne Measurement of Small-Scale Turbulence with special regard to the Polar Boundary Layer.* Zentrum fr Luft- und Raumfahrt, Braunschweig, 2009.

[12] Mathworks. *Embedded Coder User Guide.* Mathworks, $http://www.weizmann.ac.il/matlab/pdf_doc/ecoder/ecoder_ug.pdf$, 2013.

[13] OSADL. *Real-Time Linux Wiki.* Open Source, $https://rt.wiki.kernel.org/index.php/Main_Page$, 2013.

[14] STMicroelectronics. *STM32 Mainstream Performance line, ARM Cortex-M3 MCU.* STMicroelectronics, Silicon Valley, 2013.