Journal of Intelligent and Robotic Systems manuscript No.
(will be inserted by the editor)

Real-time Implementation and Validation of a New Hierarchical Path Planning Scheme of UAVs via Hardware-in-the-Loop Simulation

Dongwon Jung · Jayant Ratti · Panagiotis Tsiotras

Received: date / Accepted: date

Abstract We develop a hierarchical path planning and control algorithm for a small fixed-wing UAV. Incorporating the hardware-in-the-loop (HIL) simulation environment, the hierarchical path planning and control algorithm has been validated through on-board, real-time implementation on a small autopilot. We present two distinct real-time software framework for implementation of the overall control algorithms including path planning, path smoothing, and path following. We especially emphasize the use of a real-time kernel, which shows effectiveness and robustness in accomplishing non-trivial real-time software environment. By a seamless integration of the control algorithms with a help of real-time kernel, it has been demonstrated that the UAV equipped with a small autopilot having limited computational resources manages to autonomously accomplish the mission control objective of reaching the goal while avoiding obstacles without human intervention.

Keywords Path planning and control · Hardware-in-the-loop simulation (HILS)

1 Introduction

Autonomous path planning and control for small UAVs imposes severe restrictions on control algorithm development, stemming from the limitations imposed by the on-board hardware and the requirement for real-time implementation. This is especially the case when a low-cost micro-controller is utilized as an embedded controller for a small UAV. In order to overcome these limitations, developing a computationally efficient algorithm is imperative that the vehicle makes use of the on-board computational resources wisely.

Due to the stringent operational requirements and the restrictions imposed on UAVs, a complete solution to fully automated path planning and control of UAVs is a

Dongwon Jung
E-mail: dongwon.jung@gatech.edu

Jayant Ratti
E-mail: jratti@gatech.edu

Panagiotis Tsiotras
E-mail: tsiotras@gatech.edu
Georgia Institute of Technology, Atlanta, GA, 30332-0150

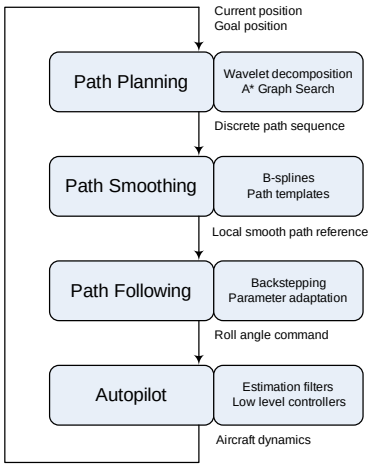


Fig. 1 Block diagram for control hierarchy of the proposed path control algorithm.

difficult undertaking. Rather, hierarchical structures have been effectively applied for many control system, in that an entire control problem can be subdivided into a set of sub-control tasks, which allows not only straightforward design of control algorithms for each modular control tasks but also simple, yet effective implementation in practice.

In this paper, a complete solution of the hierarchical path planning and control algorithm, recently developed by the authors, is experimentally validated. The control hierarchy consists of path planning, path smoothing, and path following, so that each stage provides necessary commands to the next control stage in order to accomplish a mission goal specified at the top level. The execution of the entire control algorithms is demonstrated through a realistic hardware-in-the-loop simulation environment. The overall control algorithms are coded in a micro-controller using a real-time kernel, in order to schedule each tasks effectively by taking full advantage of provided kernel services. We describe the practical issues associated with the implementation of the proposed control algorithm, when taking into consideration the actual hardware limitations.

2 Hierarchical Path Planning and Control Algorithms

In this section, we briefly describe a hierarchical path planning and control algorithm, which has been recently developed by the authors that takes into account the limited computational resources of the on-board autopilot.

Figure 1 shows the overall control hierarchy. It consists of path planning, path smoothing, path following, and the low level autopilot. At the top level of the control hierarchy, the wavelet based path planning algorithm[5] is employed to compute an optimal path from the current position of the vehicle to the goal. The path planning algorithm utilizes a multiresolution decomposition of the environment, such that a coarser resolution (initially known) is used far away from the agent, whereas fine resolution is used in the vicinity of the agent using on-board sensor data. The path planning algorithm computes the path with the highest accuracy at the current location of the vehicle, where is needed most. In conjunction with the adjacency relationship derived

from the direct use of the wavelet coefficients,[5] a discrete path sequence is solved by invoking the $\mathcal{A}^*$ graph search algorithm.

The discrete path sequence is utilized by the on-line path smoothing algorithm[6] to generate a smooth reference path, which incorporates path templates comprised of a set of B-spline curves. The path templates are obtained from an off-line optimization, so that the resulting path stays inside a prescribed cell channel. The on-line implementation of the path smoothing algorithm finds the corresponding path segments over a finite planning horizon with respect to the current position of the agent, and stitches them together, while preserving the smoothness of the composite curve.

After a local smooth path reference is obtained, a nonlinear path following control algorithm[4] is applied to asymptotically follow the reference path constructed by the path smoothing step. Assuming that the air speed and the altitude are constant, a kinematic model is utilized to design a kinematic control law for heading rate command. Subsequently, based on this kinematic control law, a roll command of the desired heading rate is derived by taking into account the inaccurate system time constant.

Finally, an autopilot with on-board sensors that provides feedback control for attitude angles, air speed, and altitude, implements the low-level inner loops for commanding each control surface to attain the roll angle steering, while keeping the altitude and the air speed constant.

As shown in Fig. 1, at each stage of the hierarchy, the corresponding control commands are obtained from the output of the previous stage, given the initial environment information (e.g., a two dimensional elevation map). With the goal position specified by the user, the hierarchical path planning and control algorithm allows the agent to accomplish the mission to reach the goal, while avoiding obstacles without human intervention.

3 Experimental Test-bed

3.1 Hardware description

A UAV platform based on the airframe of an off-the-shelf R/C model airplane has been developed, to implement the hierarchical path planning and control algorithms described above. The development of the hardware and software was done completely in-house. The on-board autopilot is equipped with a micro-controller, sensors and actuators, along with communication devices that allow full functionality for autonomous control. An 8-bit micro-controller (Rabbit RCM-3400 running at 30 MHz with 512 KB ROM and 512 KB Flash ROM) is chosen as the brain of the autopilot, which shows very limited computational throughput, as low as 7 [μsec] for floating-point multiplication and 20 [μsec] for square root, as compared to a generic high performance 32 bit micro-processor. The micro-controller provides data acquisition, processing, and communication with the ground station. It also runs not only the estimation algorithms for attitude and absolute position, but also the low-level control loops for attitude angles, air speed, and altitude control. The on-board sensors include angular rate sensors for three axes, accelerometers along all three axes, a three-axis magnetic compass, a GPS sensor, and absolute and differential pressure sensor. For detail description about the UAV platform and the autopilot can be found in Refs. [1, 2].

3.2 Hardware-in-the-loop simulation environment

A realistic hardware-in-the-loop simulation (HILS) environment has been developed to validate the UAV autopilot hardware and software development utilizing Matlab® and Simulink®. A full 6-DOF nonlinear aircraft model is used in conjunction with a linear approximation of the aerodynamic forces and moments, along with Earth gravitational (WGS-84) and magnetic field models. Detailed models of the sensors and actuators have been incorporated. Four independent computer systems are used in the hardware-in-the-loop simulation (HILS) as illustrated in Fig. 2. A 6-DOF simulator, the flight visualization computer, the autopilot micro-controller, and the ground station computer console are involved in the simulation. Further details about the HILS set-up can be found in Ref. [3].

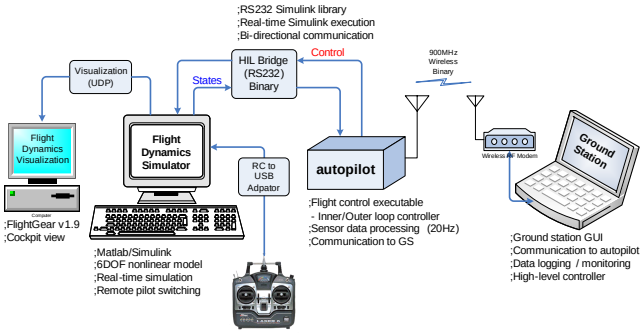


Fig. 2 High fidelity hardware-in-the-loop simulation (HILS) environment for rapid testing of the path planning and control algorithm

4 Real-time Software Environment

The software architecture of the on-board autopilot is shown in Fig. 3. It is comprised of several blocks, so called *tasks*, allotted throughout different functioning layers such as the application level, the low level control, the data processing level, and the hardware level. The tasks in the hardware level, or hardware services, interact with the actual hardware devices to collect data from sensors, to communicate to the ground station, and to issue commands to the DC servo motors. The middleware tasks on top of the hardware services provide the abstraction of the inbound and outbound data, supplying the processed data on a globally shared data bus or extracting data from the global bus to the corresponding hardware services. Utilizing the processed data on the globally shared data bus, the lower level control layer achieves the basic control functions such as estimation of attitude angles, estimation of the absolute position, and implementation of the inner loop PID controllers. Finally, three application tasks, which correspond to path planning, path generation, and path following are incorporated to implement the hierarchical path planning and control algorithm described in Section 2. It should be noted that a hierarchical control structure is implied in the application tasks, in the sense that completion of the upper level task (when an event occurs) is necessary for enabling the execution of a lower level task (event processed), by red dashed arrows

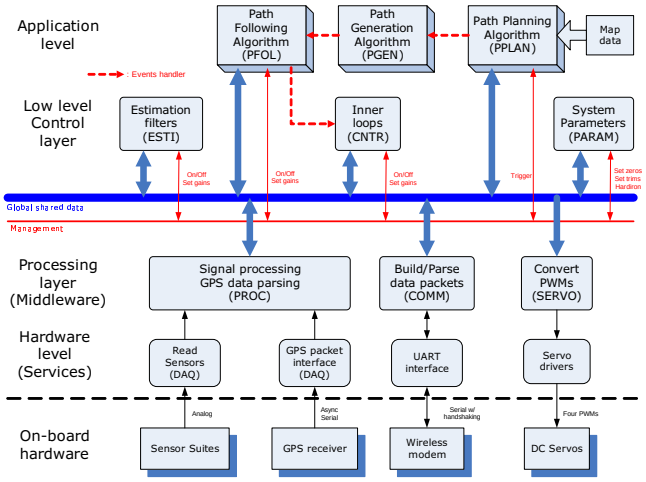


Fig. 3 Software architecture of the on-board autopilot system of the fixed-wing UAV.

between each module that represent each event handlers. In Fig. 3, besides exchanging the data via the global shared data bus, each task is managed from a global management bus for triggering execution of tasks, initializing/modifying system parameters, etc.

The task management, known as task scheduling, is an important subject to attain a seamless integration of the software, where an application contains multiple tasks to perform. In practice, a processor can only execute one instruction at a time. Thus, multitasking scheduling is necessary for embedded control system implementation where several tasks need to be executed while meeting real-time constraints. In such an embedded system, more than one task, such as control algorithm implementation, hardware device interface, and so on, can appear to be executed in parallel. Hence, the tasks need to be arranged in a timely fashion to meet the real-time criteria of the specific control application.

4.1 Cooperative scheduling methods: Initial design

For the initial implementation, we developed a real-time control software environment that is predominately based on cooperative scheduling. Cooperative scheduling is better explained by a large loop containing small fragments of codes (tasks). Each task is configured to voluntarily relinquish the execution when it is waiting, allowing other tasks to execute. This way, one big loop can execute several tasks in parallel, while no single task is busy waiting.

Like most real-time control problems, we let the big loop begin with waiting for a trigger signal from a timer, as shown by red arrows in Fig. 4. In accordance with the software framework of Fig. 3, we classify the tasks into three groups: routine tasks, application tasks, and non-periodic tasks. The routine tasks are critical tasks required for the UAV to perform minimum automatic control, which consists of the tasks of reading analog/GPS sensors (DAQ), signal processing (PROC), estimation (ESTI), inner loop control (CNTR), and servo driving (SERVO). The sampling period T_s is carefully chosen not only to ensure the necessary computation time for routine tasks, but also

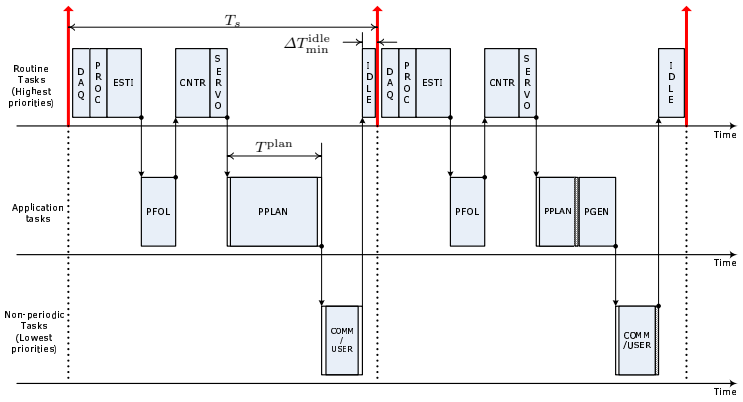


Fig. 4 A real-time scheduling method combining cooperative and naive preemptive multi-tasking.

to allow the minimum sampling period to capture the fastest dynamics of the system. In order to attain real-time scheduling over all other tasks besides the routine tasks, a sampling period of $T_s = 50$ [msec], or a sampling rate of 20 [Hz] was used. On the other hand, some of the application tasks require substantial computation time/resources, as they deal with complicated high level computational algorithms such as path planning (PPLAN), path generation (PGEN), and path following (PFOL). In particular, the path planning algorithm in Ref. 5 turns out to have a total computation time greater than the unit sampling period. As a result, in order to meet the real-time constraints, we have fragmentized the execution of the computationally intensive task, PPLAN, by several slices of code execution with a finite execution time T^{plan}. The finite execution time is selected a priori by taking into account both T_s and the (estimated) total execution time of routine tasks, such that we want to maximize the use of the CPU to complete the task PPLAN as soon as possible, while meeting real-time operation. Finally, non-periodic tasks such as communication (COMM) and user application (USER) are executed whenever the CPU becomes available, ensuring the minimum idling time $\Delta T^{\text{idle}}_{\text{min}}$ to allow the CPU to wait for another triggering signals.

Figure 5 shows a pseudo-code implementation of the proposed cooperative and the preemptive scheduling scheme. Each `costate` realizes the cooperative scheduling, while the `slice` statement realizes the preemptive scheduling in conjunction with the finite execution window ΔT^{plan}.

4.2 Preemptive scheduling methods: Final design

Given the tasks designed a priori in conjunction with an approximate knowledge of total execution time, cooperative scheduling using *costate* blocks was shown an effective implementation in the previous section. However, it will become difficult for a programmer to schedule all tasks seamlessly when an application contains several tasks that are sometimes unpredictable for their completion time. In contrast, it is possible to design a cooperative scheduler by using conservative timing estimates for corresponding tasks in a similar manner discussed in Sec. 4.1, rather resulting in poor performance with respect to the overall completion time. It can be inferred that with a conservative esti-

```
main() {
    while (1) {
        costate {
            Wait_for_timer(T_s);
            Task DAQ;
            Task PROC;
            Task ESTI;
            if (EVENT(PFOL)) Task CNTR;
            Task SERVO;
        }
        costate {
            if (EVENT(PGEN)) Task PFOL;
        }
        costate {
            if (EVENT(PPLAN)) Task PGEN;
        }
        costate {
            Task COMM;
            Task PARAM;
            Task USER;
        }
        if (ΔT^idle > ΔT^plan) {
            slice (ΔT^plan, Task PPLAN);
        }
    }
}
```

Fig. 5 Pseudo-code implementation of the combined cooperative/preemptive scheduling scheme for the hierarchical path planning and control algorithms.

mate of execution time for routine tasks, the portion of the computationally expensive tasks remain fixed regardless of the CPU being idle for the rest of the sampling period. This indicates that the CPU does not make full use of its capacity, hence delaying the execution of the overall tasks with significant amount of time. The throughput of the computationally intensive tasks may be improveed by employing a preemptive multitasking scheduler7, because the kernel will have full access of CPU time and allot the CPU to the lower level tasks whenver possible. Hence, it effectively minimizes the CPU idle time and reduces the task completion time. In this section, we present a new software framework for implementing the hierarchical path planning and control algorithm shown in Sec. 2 using a preemptive real-time kernel, MicroC/OS-II.

The MicroC/OS-II is known to be a highly portable, low memory required, scalable, preemptive real-time, multitasking kernel (RTOS) for small microcontrollers. Besides a preemptive task scheduler which can manage up to 64 tasks, the MicroC/OS-II can also provide general kernel services such as semaphores including mutual exclusion semaphores, event flags, message mailboxes, etc. These services are especially helpful for a programmer to build a complex real-time software system by integrating tasks seamlessly, simplifying the software structure using a stateflow diagram. The MicroC/OS-II is compatible with the micro-controller (Rabbit RCM-3400), while allowing small on-chip code size of the real-time kernel. The code size of the RTOS kernel is about no more than 5 to 10 kBytes,[7] which renders relatively small overhead around 5.26 % to the current total code size of 190 kBytes.

Table 1 List of tasks created in the real-time kernel.

Alias	Description	Priority	Used stack
HILS_Tx	Sending back servo commands to the simulator	11	97
HILS_Rx	Reading sensor/GPS packets from the simulator	12	153
COMM_Rx	Uplink for user command from the ground station	13	70
COMM_Proc	Parsing the user command	14	145
ESTI_Atti	Attitude estimation task	15	266
ESTI_Nav	Absolute position estimation task	16	216
CNTR	Inner loop control task	17	150
PFOL	Nonlinear path following control task	18	464
COMM_Tx	Downlink to the ground station	19	104
PGEN	Path generation task using B-spline templates	20	494
PMAN	Control coordination task	21	152
PPLAN	Multiresolution path planning task	23	445
STAT	Obtaining run-time statistics	22	–

4.2.1 Real-time software architecture

The real-time software programming begins with creating tasks. For this research we especially emphasize a real-time implementation of the path planning and control algorithms using a hardware-in-the-loop scheme, thus new tasks which deal with additional HILS communication need to be incorporated. The simulator transmits the emulated sensor data to the micro-controller via serial communication. Hence, the sensor/GPS reading task (DAQ) is substituted with the sensor data reading task (HILS_Rx), which continuously checks a serial buffer for incoming communication packets. In a same token, the servo driving task (SERVO) is replaced by the command writing task (HILS_Tx), which sends back PWM servo commands to the simulator. On the other hand, the communication task COMM is subdivided into three different tasks according to respective roles for such as a downlink for data logging (COMM_Tx), a uplink for user command (COMM_Rx), and user command parsing task (COMM_Proc). In addition, we create a PMAN task which coordinates the execution of the path planning and control algorithms, thus directly communicating with PPLAN, PGEN, and PFOL, respectively. Finally, a task STAT is created in order to obtain run-time statistics of the program such as CPU usage and the execution time of each task, as a performance measure of the real-time software. Table 1 itemizes the entire tasks created in the real-time kernel.

The MicroC/OS-II manages up to 64 distinct tasks of which the priority must be uniquely assigned. Starting from zero, increasing number imposes lower priority to be assigned to corresponding task. In particular, because the top/bottom ends of the priority are reserved for internal kernel use, application tasks are required to have priorities other than a priority level in this protected range. Following an empirical convention of priority assignment, we assign the critical tasks with high priorities because they usually involve direct hardware interface. In order to minimize degradation of the overall performance of the system, the hardware related tasks may need proper synchronization with the hardware, hence demanding for an immediate attention. It follows that routine tasks that are required for the UAV to perform minimum automatic control such as ESTI_Atti, ESTI_Nav, and CNTR are given subsequent priorities. Finally, application specific tasks such as PFOL, PGEN, and PPLAN are given lower priorities, which implies that these tasks can be made run whenever the highest priority tasks relinquish the CPU. Table 1 shows the assigned priorities for each task. Note that

the task COMM_Tx is assigned with a lower priority, because this task is less critical to the autonomous operation of the UAV.

Having the required tasks created, we proceed to design a real-time software framework by establishing relationship between tasks using available kernel services: A semaphore is utilized to control access to a globally shared object, in order to prevent it from being shared indiscriminately by several different tasks. Event flags are used when a task needs to synchronize with the occurrence of multiple events or relavant tasks. For intertask communication, a mailbox is employed to exchange a message to convey information between tasks.

Figure 6 illustrates the overall real-time software architecture for the autopilot. In the diagram two binary semaphores are utilized for two different objects corresponding to the wireless modem and a reference path curve, respectively. Any task that requires getting access on those objects needs to be blocked (by semaphore pending) until the corresponding semaphore is either non-zero or released (by semaphore posting). Consequently, only one task has an exclusive access on the objects at a time, which allows data compatibility among different tasks.

The 'flags' are posted by the triggering tasks and are consumed by the waiting tasks (arrow heads). All the significant data / flags are posted to the 'Global Data Storage (White boxes)' to be used by any task when required. (a) The 'checkered boxes' represent all the communication related tasks related to the HILS / Ground Station control. (b) The 'grey boxes' are responsible for processing the 'Math and Controls' involved in Navigation and Flight Stability. The thick 'white bus arrow' emanating from the 'Parsing Command' task constitutes a number of flags that are posted based on the commands issued from the Ground Station. Also shown are the 'Mailboxes' which are used to post messages by different tasks. The tasks pending on the mailboxes are triggered on when an appropriate message is posted. The Ground Station communicates with the UAV wirelessly as shown and the HILS is conducted using wired serial ports between a simulator (computer) and the autopilot.

4.2.2 Benefit of using the real-time kernel

Robustness: The Real Time Kernel provides many error handling capabilities during deadlock situations, we have been able to resolve the deadlocks using the timing features of the Semaphore-Pend or Flag-Pend operations, wherein which we can factor in the timing in-coherencies inherent in the resources sharing between tasks.

Flexibility and Ease of maintenance: The entire architecture for the autopilot's software has been designed keeping in mind the object oriented requirements of an applications engineer. The Real Time Kernel has played a big part in achieving this goal. The Architecture has been designed to keep the code flexible enough to add Higher level Tasks like the Multi-resolution Wave-let path planning algorithms etc without engrossing into the system level intricacies of handling and programming a microcontroller/microprocessor. The Architecture's flexibility also makes its extremely efficient to debug faults in low-level / mid-level / high-level tasks without interfering with the other tasks.

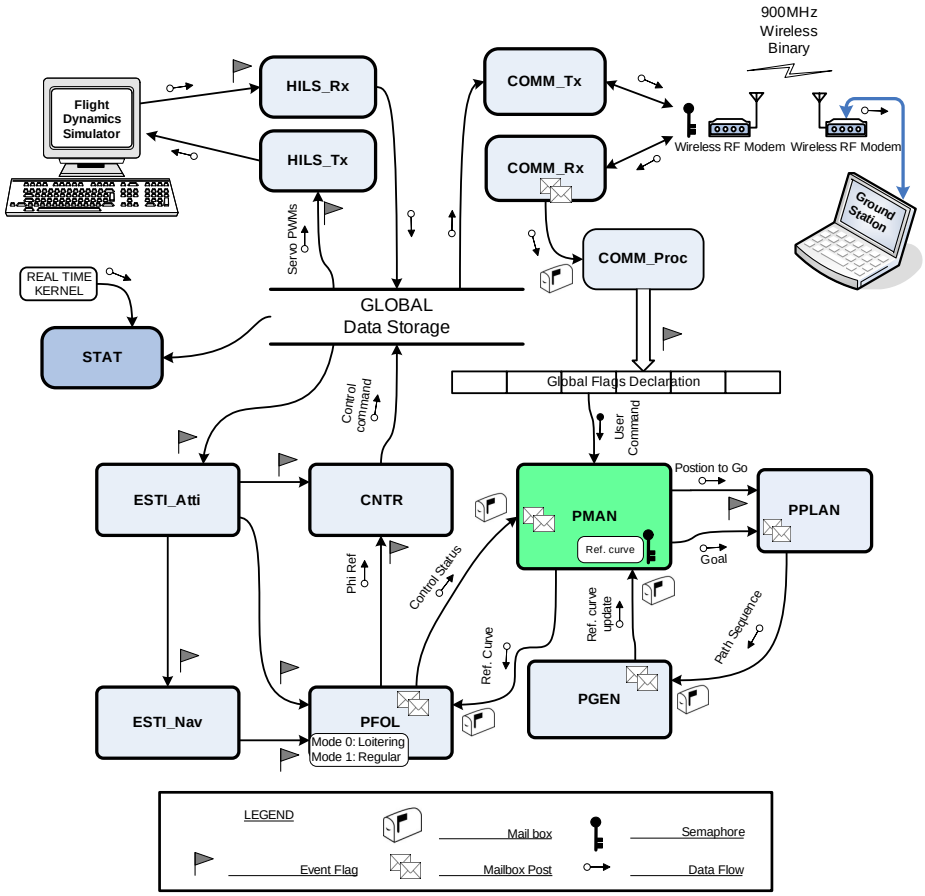


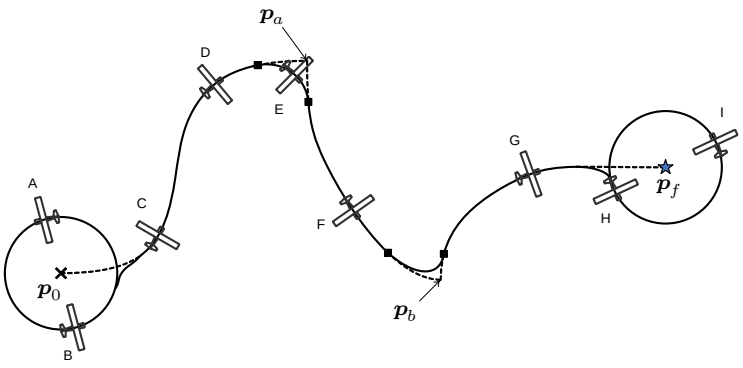
Fig. 6 A entire real-time software architecture for the path planning and control of the on-board UAV.

5 Hardware-in-the-loop Simulation Results

In this section we present real-time hardware-in-the-loop simulation results of the hierarchical path control algorithm using a small micro-controller. Details of the implementation are discussed in the sequel.

5.1 Simulation scenario

The environment $\mathcal{W}$ is the elevation map of a certain area in US state. The environment is assumed to be square of dimension 128×128 units, which corresponds to 9.6×9.6 km. Taking into account the available memory of the micro-controller, we choose the fine resolution level as $J_{\max} = 6$ and the coarser resolution level as $J_{\min} = 3$. Cells at the fine resolution have dimensions 150×150 meters, which is slightly larger than the minimum turning radius of the fixed-wing UAV. The minimum turning radius is



Step	Task description
A	Initially, the UAV is loitering around the initial position with the circle radius R_l
B	Calculate the the first path segment from $\boldsymbol{p}_0$ to $\boldsymbol{p}_a$
C	Break away from the loiter circle, start to follow the first path segment
D	Calculate the second path segment from $\boldsymbol{p}_a$ to $\boldsymbol{p}_b$, and a transient path connecting two paths
E	UAV is on the transient path
F	Calculate the third path segment, and a transient path
G	UAV is approaching the goal position, no path calculated
H	The goal is reached, end of the path control, get on the loitering circle
I	UAV is loitering around the goal position $\boldsymbol{p}_f$

Fig. 7 Illustration of the real-time implementation of the proposed hierarchical path control algorithm.

approximately calculated for the UAV flying at the constant speed of $V_T = 20$ [m/sec] with the bounded roll angle $|\phi| \leq 30^\circ$, resulting in $R_{\min} \approx 70$ [meter].

The objective of the UAV is to follow a path from the initial position to the final position while circumventing the obstacles over a certain elevation threshold. Figure 7 illustrates the detail on-line time implementation of the proposed path planning and control algorithm. Initially, the UAV is loitering around the initial position $\boldsymbol{p}_0$ until a local path segment from $\boldsymbol{p}_0$ to $\boldsymbol{p}_a$ is computed (Step A,B). Subsequently, the path following controller is engaged to follow the path (Step C,D). At step D, the UAV replans to compute a new path from the intermediate location $\boldsymbol{p}_a$ to the goal, resulting in the second local path segments from $\boldsymbol{p}_a$ to $\boldsymbol{p}_b$ extracted from the path templates that are derived from off-line optimization.[6] The first and second path segments are stitched by a transient B-spline path assuring the continuity condition at each intersection points (marked by black squares). This process iterates until the final position $\boldsymbol{p}_f$ is reached (Step H), when the UAV engages to loiter around the goal location.

5.2 Simulation results

Figure 8 shows the simulation results of the hierarchical path control implementation. Specifically, figures on the right side show the close-up view of the simulation. The channels are drawn by polygonal lines, and the UAV smoothly follows the reference path avoiding the possible obstacles outside the channels. Consequently, the UAV reaches the final destination in a collision free manner, as seen in Fig. 9(f).

5.3 Real-time kernel run-time statistics

In order to evaluate and measure the performance of the real-time software framework, we use different metrics available within the real-time kernel.

The amount of time for which the CPU is utilized by the kernel and tasks execution can be retrived from the CPU usage metric, which is % duty cycle of the CPU, over given statistics sampling interval. When high CPU demanding tasks such as PPLAN and PGEN come into place, higher percentage CPU usage results in much quicker completion of these tasks. Hence, higher percentage CPU usage imply higher performance and efficiency. Figure 10(a) shows this metric during the run-time.

6 Conclusions

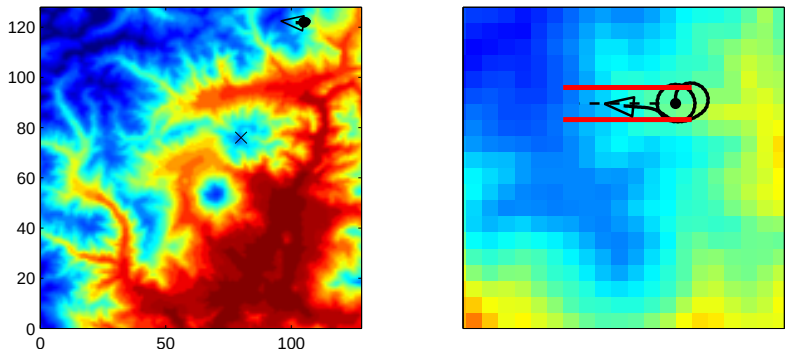
We implement a hierarchical path planning and control of a small UAV on the actual hardware platform. Based on the high fidelity hardware-in-the-loop (HIL) simulation environment, the proposed hierarchical path planning and control algorithm has been validated through the on-line, real-time implementation on a small autopilot. By a seamless integration of the control algorithms for path planning, path smoothing, and path following employing a combined cooperative/preemptive scheduling method, it has been demonstrated that the UAV equipped with a small autopilot having limited computational resources manages to accomplish the mission objective to reach the goal while avoiding obstacles without human intervention. In the final version of the paper, an implementation issue of employing a real-time OS, which incorporates fully preemptive scheduling, will be dealt with to discuss the merit of using real-time OS such as flexibility of task management, robustness of code execution, etc. A quantitative metric will be also provided to show the improvement of using real-time OS, as compared to the scheduling method described in Section 4.

Acknowledgements Partial support for this work has been provided by NSF award CMS-0510259

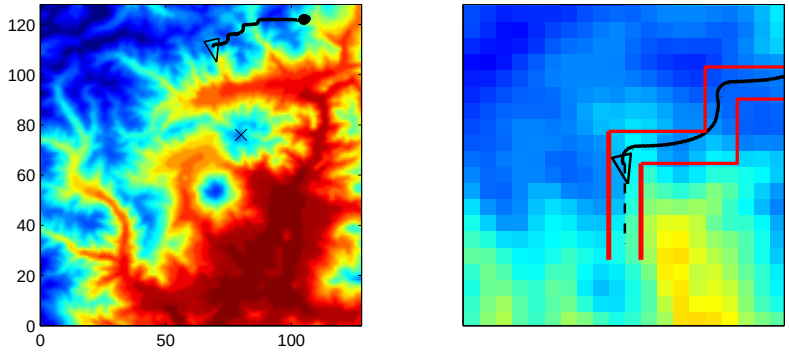
References

1. Jung, D., Levy, E.J., Zhou, D., Fink, R., Moshe, J., Earl, A., Tsiotras, P.: Design and Development of a Low-Cost Test-Bed for Undergraduate Education in UAVs. In: Proceedings of the 44^{th} IEEE Conference on Decision and Control, pp. 2739–2744. Seville, Spain (2005)
2. Jung, D., Tsiotras, P.: Inertial Attitude and Position Reference System Development for a Small UAV. In: AIAA Infotech at Aerospace. Rohnert Park, CA (2007). AIAA Paper 2007-2763
3. Jung, D., Tsiotras, P.: Modelling and Hardware-in-the-loop Simulation for a Small Unmanned Aerial Vehicle. In: AIAA Infotech at Aerospace. Rohnert Park, CA (2007). AIAA Paper 2007-2768
4. Jung, D., Tsiotras, P.: Bank-To-Turn Control for a Small UAV using Backstepping and Parameter Adaptation. In: International Federation of Automatic Control (IFAC) World Congress. Seoul, Korea (2008)
5. Jung, D., Tsiotras, P.: Multiresolution On-Line Path Planning for Small Unmanned Aerial Vehicles. In: American Control Conference. Seattle, WA (2008)
6. Jung, D., Tsiotras, P.: On-line Path Generation for Small Unmanned Aerial Vehicles Using B-Spline Path Templates. In: AIAA Guidance, Navigation and Control Conference. Honolulu, HI (2008). AIAA Paper 2008-7135

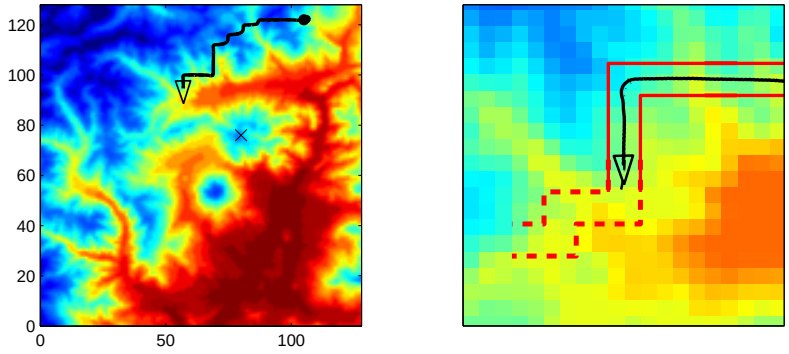
7. Labrosse, J.J.: MicroC/OS-II - The Real-Time Kernel, 2 edn. CMPBooks, San Francisco, CA (2002)



(a) $t = 64.5$ [sec]

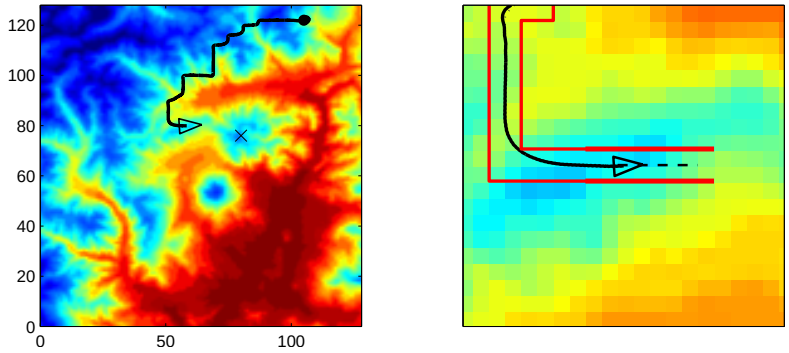


(b) $t = 222.0$ [sec]

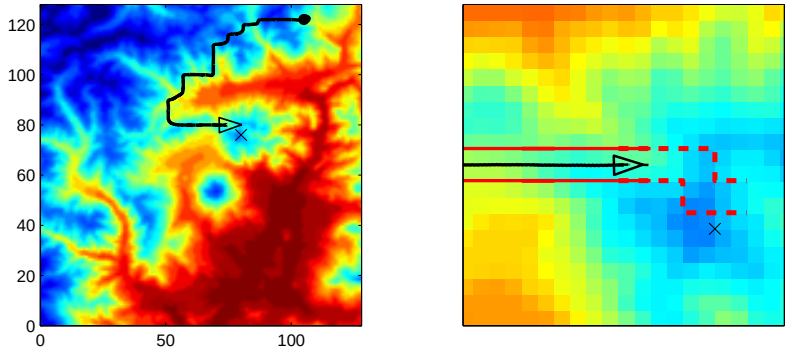


(c) $t = 333.0$ [sec]

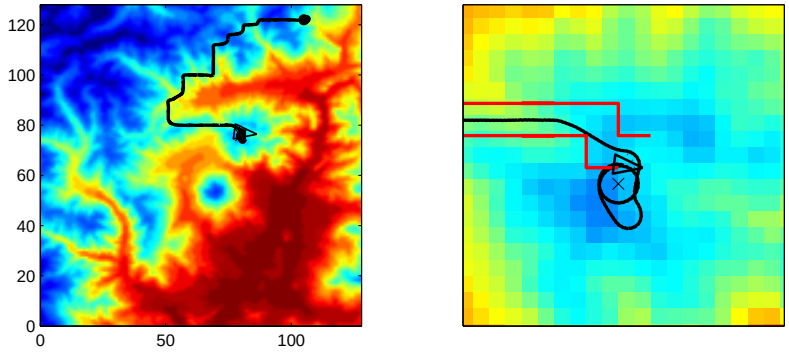
Fig. 8 Simulation results of the hierarchical path control implementation. Figures on the right show the close-up view of the simulation. At each instant the channel is drawn by polygonal lines, where the smooth path segment from the path templates stays. The actual path followed by the UAV is drawn on top of the reference path.



(d) $t = 429.0$ [sec]

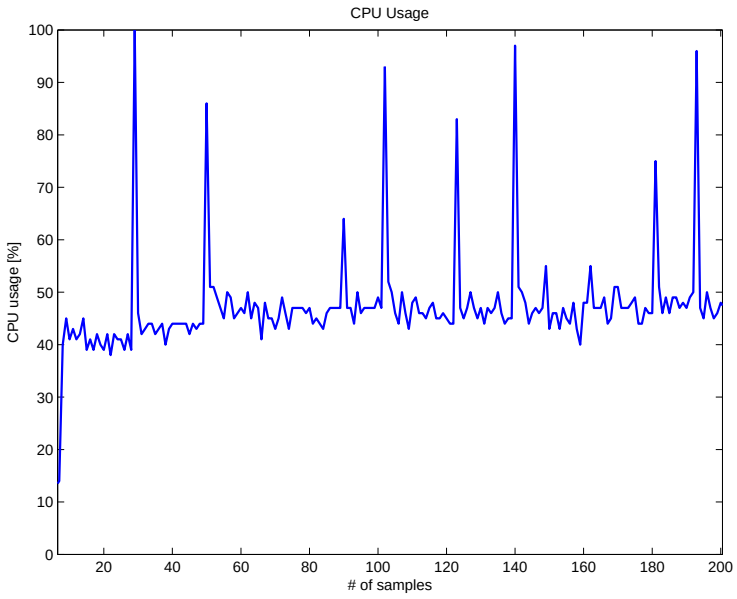


(e) $t = 492.5$ [sec]

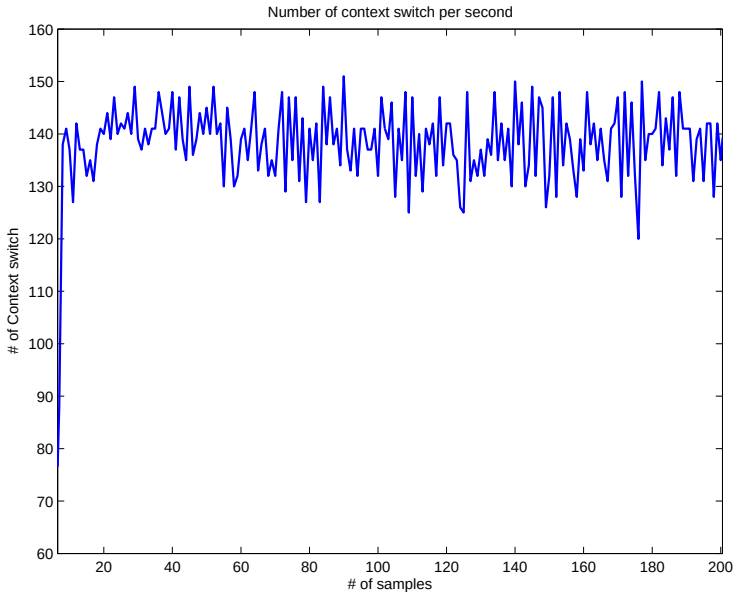


(f) $t = 591.5$ [sec]

Fig. 9 Simulation results of the hierarchical path control implementation. (cont'd)

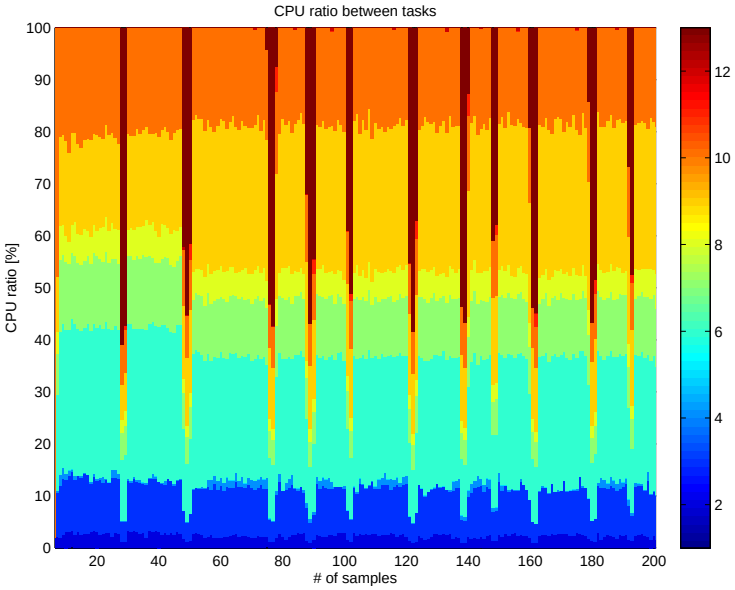


(a) CPU usage

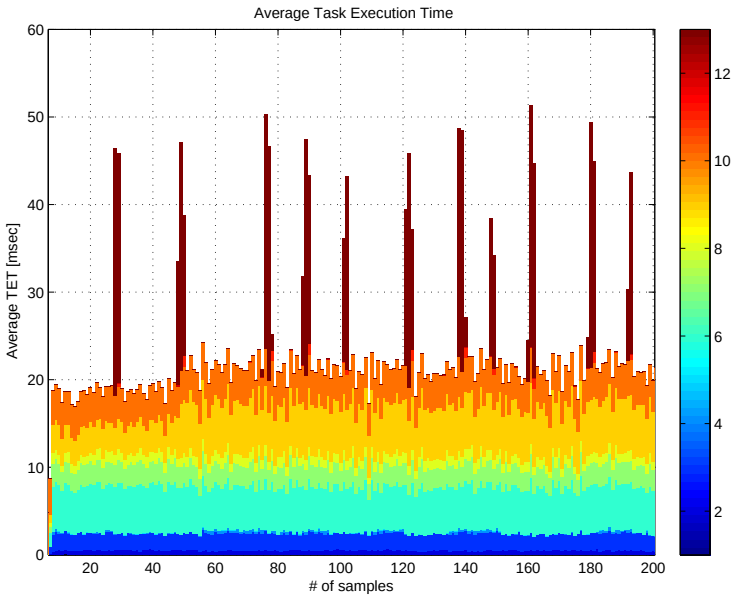


(b) Number of context switch per second

Fig. 10 Real-time kernel run-time statistics: CPU usage and number of context switch.



(a) CPU usage



(b) Average tasks execution time

Fig. 11 Real-time kernel run-time statistics: Tasks execution time v/s CPU ratio