

The JAviator: A High-Payload Quadrotor UAV with High-Level Programming Capabilities*

Silviu S. Craciunas[†] Christoph M. Kirsch[‡]
Harald Röck[§] Rainer Trummer[¶]

Department of Computer Sciences
University of Salzburg, Austria

We present the JAviator (*Java Aviator*), a high-performance quadrotor model helicopter that is built around a high-integrity frame, which is horizontally and vertically symmetric, and supports high payloads through light-weight materials and advanced brushless motors. The JAviator is 1.3m in diameter, weighs 2.2kg in total, and generates a maximum lift of 5.4kg, which translates into a theoretical maximum payload of 3.2kg. Without payload the maximum flight time is around 40min. The JAviator may be programmed in Java using a real-time extension called Exotasks. As an alternative to Java threads, Exotasks enable time-portable programming of high-performance, hard real-time applications in Java. With Exotasks we are able to fly the JAviator using different hardware platforms and software workloads without changing any of the application-level real-time software. The JAviator also serves as a test platform for other software projects such as our own real-time operating system called Tiptoe, which is in a prototypical stage and meant to provide even stronger forms of time portability than Exotasks. Tiptoe supports the execution of real-time processes whose temporal behavior, even including system-level aspects such as their I/O communication and memory management, can be predicted per process, independently of the other processes. The paper describes the JAviator's hard- and software design in detail and reports on propulsion, flight, and software tests.

I. Introduction

Quadrotor UAVs are typically built light-weight with limitations in payload and programming capabilities. In this paper, we present the JAviator (*Java Aviator*),¹ a high-performance quadrotor model helicopter, which stands out from others with its fully symmetrical frame design (horizontally and vertically), support of high payloads, and high-level programming capabilities. The JAviator has a diameter of 1.3m and weighs 2.2kg, including the battery and all onboard electronics. Its propulsion system generates a maximum lift of 5.4kg, which translates into a theoretical maximum payload of 3.2kg. Without payload the maximum flight time is around 40min.

We have designed and manufactured the JAviator's frame, presented in Figure 1, completely from scratch using only high-performance materials such as carbon fiber, aircraft aluminum, and medical titanium. The frame, when seen from the side, is roughly comparable to the shape of a bicycle wheel with four "top" and four "bottom" spokes. The frame consists of two symmetric cross-like structures (top and bottom), each with a center ring to which four arms (the spokes) are attached. The center rings are vertically connected via eight pipes forming the JAviator's body. At the far ends, each pair of opposing arms is vertically connected by a rod serving as rotor axle. This design provides high mechanical integrity even with light-weight materials and helped us minimizing the number of different frame components. Our prototype was built entirely from

*This research is supported by a 2007 IBM Faculty Award, the EU ArtistDesign Network of Excellence on Embedded Systems Design, and a University of Salzburg Startup Grant.

[†]Ph.D. Candidate, silviu.craciunas@cs.uni-salzburg.at. Supported by the Austrian Science Fund No. P18913-N15.

[‡]Professor, christoph.kirsch@cs.uni-salzburg.at. AIAA Member.

[§]Ph.D. Candidate, harald.roeck@cs.uni-salzburg.at. AIAA Student Member.

[¶]Ph.D. Candidate, rainer.trummer@cs.uni-salzburg.at. AIAA Student Member.



Figure 1. The JAviator quadrotor helicopter.

hand-made parts, whereas the final, more advanced version presented here has already been produced in a small series consisting of five helicopters (used by us and our research collaborators).

The JAviator's propulsion system comprises of custom-built, hand-made brushless motors that were originally designed for model planes but modified for us according to our specifications to improve cooling and power. In particular, we had the motor form re-shaped (cooling) and the coil's impedance modified (power). The JAviator's safety system rests upon a printed circuit board for distributing power to the motors, which can be armed and disarmed by a remote control that operates on an entirely hardware-based, analog radio channel by-passing any of the helicopter's software-based flight control system. We have designed the power board and remote control from scratch with a particular focus on reliability and safety.

Unlike most other UAVs and, for that matter, most hard real-time systems, the JAviator may be programmed in Java using a real-time extension called Exotasks, which we developed in collaboration with IBM Research.^{2,3} Exotasks enable time-portable programming of high-performance, hard real-time applications in Java. The resulting application code is not only efficient and easier to develop than in lower-level languages such as C but also robust with respect to real time. Time-portable programs do not change their relevant real-time behavior across different hardware platforms and software workloads, similar to Java's write-once-run-anywhere paradigm for functional behavior but extended to the temporal domain. With Exotasks we are able to fly the JAviator using different hardware and workloads without changing any of the application-level real-time software. Except for device-level sensing and actuating code written in C, all Exotask-based software was written in Java.

Apart from the Exotask system, the JAviator also serves as a test platform for other software projects such as our own real-time operating system called Tiptoe,⁴ which is in a prototypical stage and meant to provide even stronger forms of time portability than Exotasks. The goal of the Tiptoe project is to develop systems software that supports the execution of real-time processes whose temporal behavior, even including system-level aspects such as their I/O communication⁵ and memory management,⁶ can be predicted per process, independently of the other processes. Tiptoe processes can be added and removed without changing the relevant real-time behavior of other processes, as long as there are sufficient computational resources. Tiptoe processes are implemented in the so-called workload-oriented programming model,⁷ which is a design methodology for specifying throughput and latency of real-time software processes even on the level of individual process actions such as system and procedure call invocations. The key programming abstraction is that the workload involved in executing a process action fully determines the action's response time, independently of any previous or concurrent actions.

The focus of this paper is on the JAviator's hard- and software design but not on its actual controller design. So far, the JAviator's flight control system supports manual flight and uses a simple combination of PD (roll/pitch) and PID (yaw) controllers for attitude control. We have also experimented with automatic altitude controllers but have not used them in the flight tests for this paper. In order to demonstrate the JAviator's programming capabilities, we have implemented its flight control system in Java using Exotasks (EControl) and developed a pure Java version (JControl) as baseline for the EControl version. Moreover, we have reimplemented the Java version of the flight control system in C for Tiptoe (TControl), which does not support the execution of Java programs in its current implementation, as well as for Linux with real-time patches applied (CControl) as baseline for the TControl version. We are able to fly the JAviator using all four controller implementations.

RELATED WORK We relate our work to hardware and software design aspects of other UAVs (mostly quadrotors). The most prevalent quadrotor design comprises four side arms that are attached to the body in a horizontally straight manner. This approach can be found in the design of the OS4,⁸ STARMAC II,⁹ Draganflyer I through V,¹⁰ and the EADS Quattrocopter,¹¹ to mention just a few. Another, similar approach is to attach the side arms at the deepest point of the body in an upward-pointing position, as realized in the Draganflyer X-Pro,¹⁰ which has the advantage of lowering the barycenter.

Many research projects involve commercially available remote-controlled quadrotor models, which have been modified to some extent.¹²⁻¹⁴ The most-commonly used representative for this purpose is the Draganflyer III and its successors.¹⁰ The latest version, the Draganflyer V-Ti, has a total diameter of 76cm and a curb weight of 482g. Unfortunately, without upgrading the complete propulsion system and applying some integrity-increasing modifications to the frame, the V-Ti is restricted to payloads not exceeding approximately 150g.

The quadrotor that comes closest to the JAviator, at least in terms of physical dimensions, is the Draganflyer X-Pro,¹⁰ which has a total diameter of 1.2m, a curb weight of 2.3kg, and is capable of carrying payloads of up to 500g. The X-Pro is originally shipped with 2-blade rotors but can be upgraded to 3-blade rotors if desired. Similar to our approach, the X-Pro's propulsion system comprises belt-drive gearing. However, in contrast to the JAviator's custom-built 35g 3-phase AC motors, the X-Pro is equipped with relatively heavy DC motors that account for a total of roughly 600g. Compared to the eight skinny side-arm pipes of the JAviator, the X-Pro contains four rather strong side-arm pipes, which are foldable to save space when transporting the helicopter.

Many research projects involving quadrotors are primarily concerned with control-engineering problems and typically use conventional frame designs. However, there are also approaches that differ largely such as the custom-built X-4 Flyer Mark II,¹⁵ which stands out with its unusual frame design and aero-elastic rotor blades. The frame is composed of four large rotor-arm stubs, where each stub consists of two similar carbon-fiber plates with some distance between them for attaching the motors. The rotors are inverted and placed downwards to obtain favorable integrity properties. The blades, which are mounted with a sprung-teetering rotor hub, have been modeled in accordance to blade element and momentum theory to achieve maximal thrust performance. Compared to other quadrotor models, the X-4 Flyer Mark II is much heavier with a curb weight of 4.0kg and intended to carry an additional payload of 1.0kg. Measurements that prove the efficiency of the individual rotor design have been presented, however, no flight tests have been conducted.¹⁶

Besides support of high payloads, high-level programming capabilities are another key feature of the JAviator. There are several related projects dedicated to improving real-time programming methodologies. One of the main challenges when programming in languages like Java is unpredictable execution delay in user applications induced by the garbage collector (GC). The real-time specification for Java (RTSJ)¹⁷ has been developed to overcome GC-induced execution delays by letting application developers handle at least some of the memory management manually for better real-time performance but at a loss of memory safety guarantees. Besides the JAviator, the only other example of a UAV programmed in Java we are aware of is the ScanEagle,¹⁸ which makes use of RTSJ instead of Exotasks. In contrast to RTSJ, however, Exotask code can be written in standard, memory-safe Java as long as certain data sharing is avoided, which is checked and enforced by compile-time tools and the runtime system. Exotasks use private heaps that are individually garbage-collected whenever the tasks do not execute. The additional memory structure enables low-latency execution of Java programs.^{2,3}

The Tiptoe project is related to numerous other projects aimed at building real-time operating systems, in particular, with small footprint such as TinyOS.¹⁹ Tiptoe is also related to projects focusing on real-time programming methodologies such as our own coordination languages for distributed control systems called Giotto²⁰ and its successor HTL,²¹ which inspired some aspects of the programming abstractions in Tiptoe. The real-time memory management system of Tiptoe⁶ supports memory allocation and deallocation in real time while keeping memory fragmentation small and predictable. Its design is influenced by the memory layout of IBM's Metronome real-time garbage collector.²²

PAPER OUTLINE The rest of the paper is structured as follows. In Section II, we describe the JAviator's frame design, propulsion system, power electronics, sensors, and computer system. Section III covers the JAviator's software platforms. We discuss our experiments in Section IV. Future work is outlined in Section V, followed by conclusions given in Section VI.

II. The JAviator

The JAviator, shown in Figure 1, is a custom-built quadrotor model helicopter, which we have designed and manufactured completely from scratch using only high-performance materials such as carbon fiber (CF), aircraft aluminum (AL), and medical titanium (TI). After creating a prototype entirely from hand-made parts, we have designed and built a final and more advanced version presented here. The final version is slightly larger but employs the same propulsion system than the prototype. The main portion of the frame is built from CF, whereas AL and TI were chosen for the connecting parts and propulsion groups. The JAviator has a total diameter of 1.3m and a curb weight of 2.2kg, including the battery and all onboard electronics. It is equipped with custom-built 3-phase AC motors, which are significantly stronger than conventional motors with identical weight. Depending on the gearing, the propulsion system is capable of generating a maximum lift of 5.4kg. Accordingly, besides lifting the helicopter’s curb weight, the JAviator can lift a theoretical maximum payload of 3.2kg. Without any additional payload, the current configuration of propulsion system and battery offers a flight time of approximately 40min.

II.A. Frame

The JAviator’s frame can be compared to a bicycle wheel that, for a better match, contains four “top” and four “bottom” spokes. More precisely, the frame is a light-weight construction consisting of a symmetrical (both horizontally and vertically) top and bottom frame. These two partial frames, as well as the eight vertical connecting pipes, are built entirely from CF and AL. The rotor axles, which connect the top and bottom frame at their far ends, are manufactured from TI with the aim to ensure high cohesiveness at reduced weight. The frame design of the JAviator not only incorporates light-weight materials, it also provides high mechanical integrity.



Figure 2. Central ring (left), vertical connecting pipes (middle), and complete body frame (right).

Figure 2 (left) shows the bottom center ring with four AL flanges installed that connect to the side-arm pipes. These flanges are designed to also hold eight vertical CF pipes, as shown in Figure 2 (middle). The top ring, which is identical to the bottom ring and similarly equipped with four AL flanges, completes the JAviator’s body frame, depicted in Figure 2 (right). In this illustration, the bottom ring is already closed with a thin CF plate to carry the onboard battery. An equally shaped top plate, not assembled in the present figure, is used to mount the inertial measurement unit (IMU). The body of the JAviator is formed like a cage, which is simple in its design, features fast and easy (re)assembly, and also ensures high integrity at relatively low weight. Compared to most other quadrotor bodies, this cage design acts like a protection frame for the onboard electronics and has proven in crashes to withstand collisions without serious damage. The four side arms consist of two CF pipes each with the same diameter as the vertical body pipes. Each side-arm pipe has AL connectors on both ends that fit to the AL flanges, as shown in Figure 3 (left), and also contain holes for mounting the rotor axles, shown in Figure 3 (right).

In order to achieve a high degree of precision and reproducibility, the following fabrication methods were used. The CF rings and plates were manufactured by flow-jet cutting, whereas laser cutting was applied for the AL flanges, TI motor mounts, and TI rotor triangles. The AL frame connectors as well as all remaining AL and TI rotor components are CNC-fabricated. Due to the symmetrical frame design, the amount of

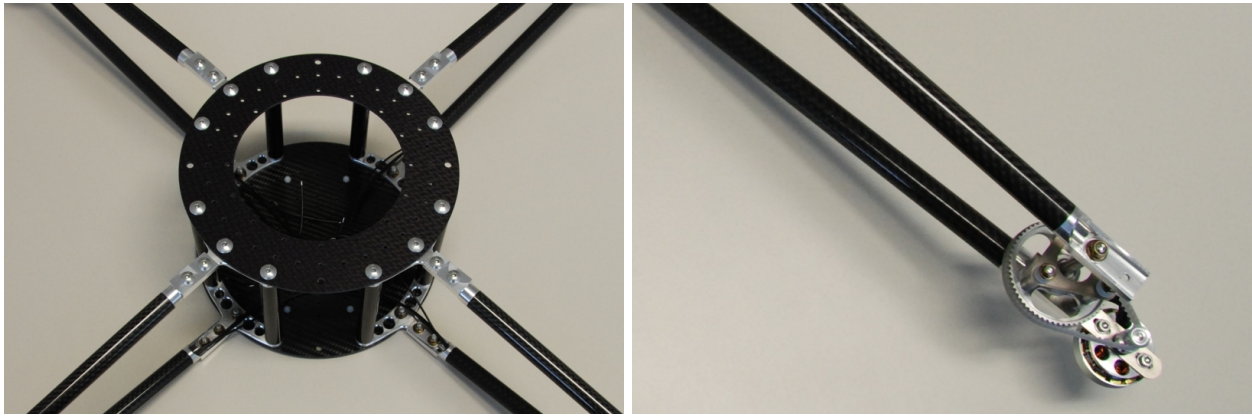


Figure 3. Side arms assembled (left) and side-arm rotor end (right).

comprising parts could be reduced to a lower number of different parts with higher quantity of occurrence in the following way. The JAviator frame contains eight body pipes as well as eight side-arm pipes. Figure 4 (left) shows the AL connector that serves as end tap for the body pipes. Due to the identical top and bottom frame, the same connector can be used on both ends of a body pipe for connecting the eight vertical pipes with the top and bottom ring. The design of the end taps for the side-arm pipes, however, turned out to be a challenge. A side arm consists of an upper and a lower pipe. Thus, there are four pipe ends to be equipped with appropriate connectors that need to satisfy the following five requirements: (1), the upper and lower connector at the body side need to connect to the top and bottom ring, respectively. (2), these connectors should also allow to tilt up the side arm continuously in the range of 0 to 6 degrees (to have the rotors point slightly outwards and lower the barycenter). (3), the upper and lower connector at the rotor side need to mount the rotor axle. (4), the lower one of these two connectors must provide a means for mounting a motor, whereas (5), the upper one should allow for mounting a signal light. For the prototype version of the JAviator, we designed hand-made connectors built from conventional AL pipes and profiles. Because the prototype neither had tilted rotors nor signal lights installed, the upper-pipe connectors could be made identical to the lower-pipe connectors. However, this solution was based on two different connectors. Moreover, the body-side connectors did not facilitate tilting the side arms.



Figure 4. Body-pipe connector (left), side-arm-pipe connector (middle), and side-arm flange (right).

In our effort to create a single connector that fulfills all of the five requirements, we designed the connector presented in Figure 4 (middle). This connector serves as end tap for both sides of both the upper and lower side-arm pipe. The problem with enabling a side-arm tilt was solved by designing a corresponding AL flange, depicted in Figure 4 (right), which contains a connecting strap that fits in the milled-out portion of the connector. The angle between the turned and milled portion of the connector was chosen to enable side-arm tilts of up to 6 degrees without stressing the comprised components. Any desired side-arm tilt in the admissible range can be obtained by adapting the connecting straps to the desired angle and appropriately

shortening the upper side-arm pipes. In case of the flange shown in the figure, the connecting strap is given a tilt of 3 degrees, which demands the upper side-arm pipe to be exactly 7mm shorter than the lower one. Basically, there is no need for tilting the rotors of a quadrotor. Nevertheless, we observed that the slightly conical air cushion generated by the upward-tilted rotors as well as the lowered barycenter positively affect the JAviator’s aerodynamical behavior. We did not verify this observation formally, but noticed a more stable behavior during lift-off and landing. Referring to the rest of the above connector specification, which addresses means for mounting a motor and a signal light, these problems could be solved easily. In particular, the milled-out portion of the connector was chosen such that it allows to assemble a thin strap of TI sheet for mounting a motor as well as a small printed circuit board for attaching a signal light. Accordingly, the number of different CF parts and AL connectors, listed in Figure 5 (left), could be reduced to the minimum required for the JAviator frame. A completely assembled JAviator frame, as shown in Figure 5 (right), weighs exactly 887g, including four rubber dumpers installed beneath the bottom ring, all required screws and nuts, as well as the four propulsion groups without motor and blades. Because the rotor axles are part of the frame construction, the rotors, belts, and motor mounts need to be installed already during assembly.

Component	Weight [g]	Quantity
Body Ring	53.3	2
Body Plate	25.6	2
Body Pipe	4.0	8
Body-pipe Connector	4.5	16
Side-arm Flange	12.7	8
Side-arm Pipe	8.0	8
Side-arm-pipe Connector	9.2	16



Figure 5. JAviator frame components (left) and picture of JAviator series production (right).

II.B. Propulsion

The propulsion system of the JAviator consists of four identical propulsion groups. Each rotor has three CF rotor blades attached, where two opposing rotors are equipped with clockwise and the other two with counter-clockwise spinning blades. Rather than using toothed gears, the transmission between motors and rotors is performed via belt drives, which tend to be less noisy and are more robust for this purpose. The belt-drive gears and belts are industrial DIN-norm T2.5 components, which can be found in tools of any description, for instance, small planing machines. The rotor is composed of an AL belt-drive gear with press-fitted stainless-steel bearings at both ends, shown in Figure 6 (left). This main gear directly carries the three AL shafts for mounting the blades. The shafts are conjuncted by a TI triangle at the upper end, depicted in Figure 6 (middle), for compensating centrifugal forces that might cause shaft deformations at high rotation speeds. The rotor axles are also manufactured from TI to achieve high integrity at low weight. Note that the rotor axles not only carry the rotors, they also serve as primary frame components that connect the upper and lower CF pipes of the side arms. Compared to conventional quadrotor designs where only one end of a rotor axle is connected with a side arm, the present double-end-mounted rotor-axle design has two significant advantages: (1), it enables precise adjusting of the four rotor axles regarding plane parallelism, and (2), it prevents the rotors from undesired twisting that might occur with one-end-mounted rotors.

In order to generate maximum thrust at minimum weight, we decided to use custom-built 3-phase AC motors also known as brushless motors or outrunners, shown in Figure 6 (right). They consist of CNC-fabricated AL parts and hand-coiled armatures, are equipped with a TI axle, and weigh only 35g. They were originally designed for model planes, but re-shaped for us according to our specifications. The modifications applied are as follows: (1), the armature side was re-shaped to fit the JAviator’s motor mount and all redundant material removed to enhance cooling. (2), the axle was placed upside-down, so that the axle’s stub is located on the mounting side rather than the outrunner side. (3), the armature was changed to contain only ten windings of 0.6mm wire per pole. We tested two gearing options for the belt drive, 1:6 and 1:5, and chose the stronger 1:5 transmission for all of our five JAviators, because the flight time without payload is similar but it provides more thrust.



Figure 6. Rotor bottom view (left), rotor top view (middle), and brushless motor (right).

Figure 7 (left) shows the top view of a propulsion group and reveals that the three rotor blades are in hinged position when not spinning. Strictly speaking, the blades are pre-stressed via springs to achieve a self-centering behavior during rotation. There are two reasons for using this technique: (1), pre-stressing the blades favors the use of thinner blades that would otherwise tend to flutter at higher rotation speeds. (2), it eliminates the need for positioning the blades correctly before each flight. When rotating, the blades fully expand at approximately half of the maximum rotation speed. The blades shown in the figure are custom-built according to our specifications and roughly similar in shape and size compared to the ones used for the Draganflyer X-Pro,¹⁰ but only half as thick, and therefore, less in weight by more than a third. Figure 7 (right) gives a closer view on a propulsion group, revealing how the springs are located between the blade shovels and the rotor's main gear. We have experimented with even thinner blades, but switched to the X-Pro blades for the series production since they allow easier and faster (dis)assembling. The latter figure also gives a closer view on the top of a motor, showing the coarse, low-impedance coil of the armature.

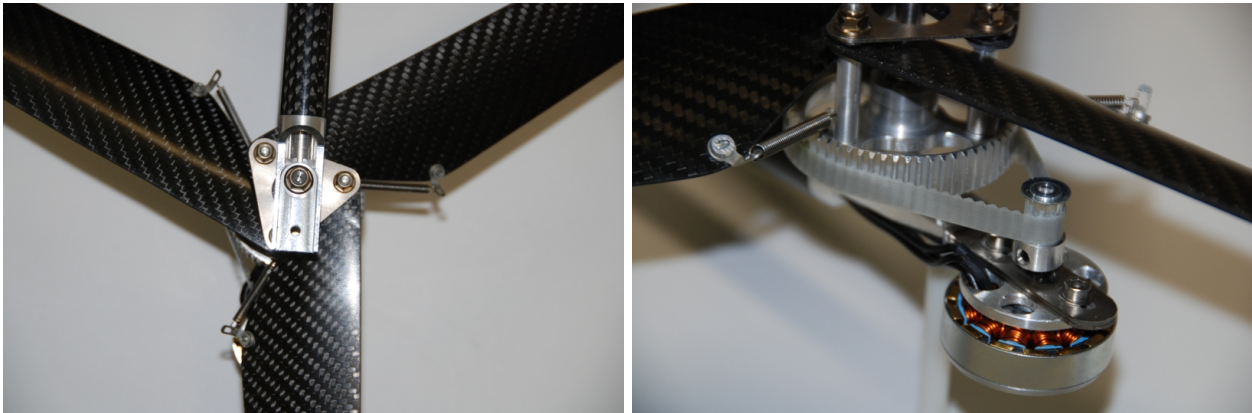


Figure 7. Self-centering rotor blades (left) and belt-driven propulsion group (right).

II.C. Power

The power consumed by the four motors and the avionics equipment is supplied by a single Thunder Power 4S3P 14.8V 6300mAh lithium-polymer battery. This battery, which weighs exactly 458g, is actually the heaviest component of the JAviator. Nevertheless, since the JAviator provides sufficient payload capabilities and its body offers enough remaining space, a second battery can be added to extend the flight time if desired. In the prototype version, the motor controllers were connected directly to the battery and the motors. In other words, in case of any hardware, software, and/or remote-connectivity failure, there was no way to shut down the motors independently of the onboard computer system. Performing merely a software-triggered shutdown can be considered risky and unsafe, especially if the remote connectivity breaks down. On the

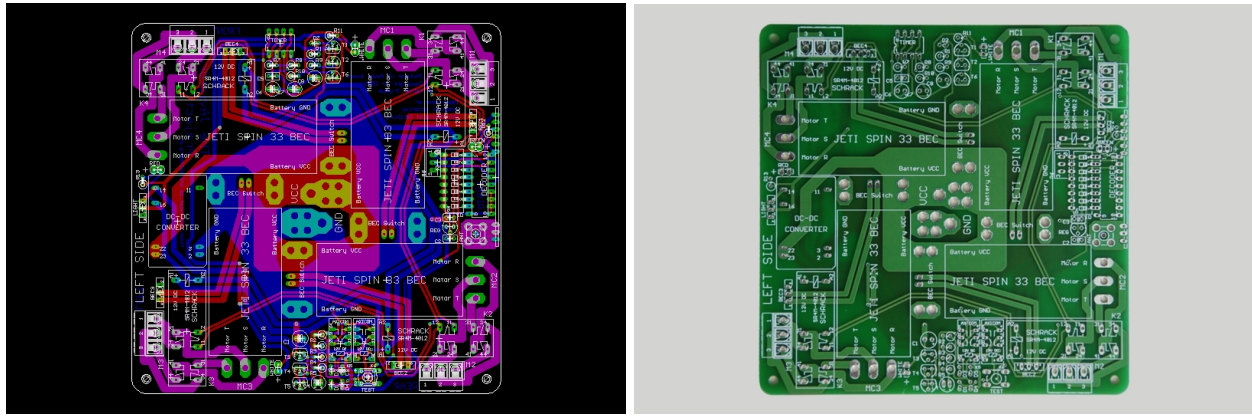


Figure 8. Power board two-layer layout (left) and manufactured printed circuit board (right).

other hand, cutting the pulse width modulation (PWM) signals to the motor controllers via some remotely controlled mechanism would not be very effective, because the motor controllers wait for 3s upon signal loss before they shut down. Primarily due to this lack of safety, we decided to develop a printed circuit board, which we call the *power board*, aimed to satisfy the following five requirements: (1), the board needs to provide a means for cutting the power to the motors independently of the onboard computer system. (2), this shutdown mechanism should also allow to arm the helicopter. More precisely, rather than just providing a means for shutting down the motors, arbitrary (dis)arming should be possible. (3), for safety reasons, the remote connection used for this purpose must be independent of the remote connection between the helicopter and the ground station. (4), a mechanism for driving signal lights at the upper side-arm ends should be provided for indicating the helicopter’s arming status. (5), to reduce the amount of wires between the battery and the motor controllers, and to facilitate a quick replacement of the entire high-power electronics, the four motor controllers should be integrated on the power board rather than being placed and connected separately.

Requirement (1) of cutting the power to the motors was solved by interrupting the 3-phase connection between each pair of motor controller and motor with a 3-pole relay. This decision was based on the following consideration: when demanding full thrust, each motor consumes about 12A, which sums up to a total of 48A at 14.8V battery voltage. Cutting this high DC is not easy and would at least require some huge relay. Even if employing four separate, say, 15A relays, the dimension and weight of such a relay would still exceed our admissible limit for this purpose. One final remark: cutting the power at the motor controllers’ input side is not very advantageous in case of the JAViator, since it would also cut the power to the onboard computer system, which is supplied through the motor controllers’ battery eliminator circuits (BECs). Consequently, we chose 3-pole relays for cutting the power directly between the motor controllers and the motors. We currently use the Schrack SR4M4012 security relay, which contains a separate chamber for each contact and is capable of switching 8A at 250V AC per contact. This relay suffices in our case, because the motor controller DC input of roughly 200W transforms to a 3-phase AC output of less than 120W per phase (120-degree phase shift $\Rightarrow P_{phase} = P_{total}/\sqrt{3} = 200W/\sqrt{3} = 115.47W$).

Requirement (2), which states that arbitrary (dis)arming should be possible, was solved by a mechanism that reacts to a transmitted signal in a time-dependent manner for both arming and disarming. More precisely, whenever a signal is transmitted from the sender, the mechanism disarms the helicopter, but at the same time, starts arming it by loading a specific capacitor of the relay control circuit. In order to fully load this capacitor, and hence to arm the helicopter, the signal must be sent for at least 3s without interruption, because any interruption would immediately reset the relay control circuit. As a consequence, the proper operation of the emergency shutdown mechanism is checked with each arming procedure.

Requirement (3), addressing the independent remote connection, was solved by integrating a Radiometrix RX3A 868MHz FM receiver, which matches a Radiometrix TX3A 868MHz FM sender that transmits the (dis)arming signal. The 868MHz band was chosen due to a European restriction that prohibits the usage of frequencies in this band for longer than 30s without interruption. In this sense, the entire 868MHz band is reserved for transmitting short signals and may not be used for long-term data transmission, which accounts for reduced interferences.

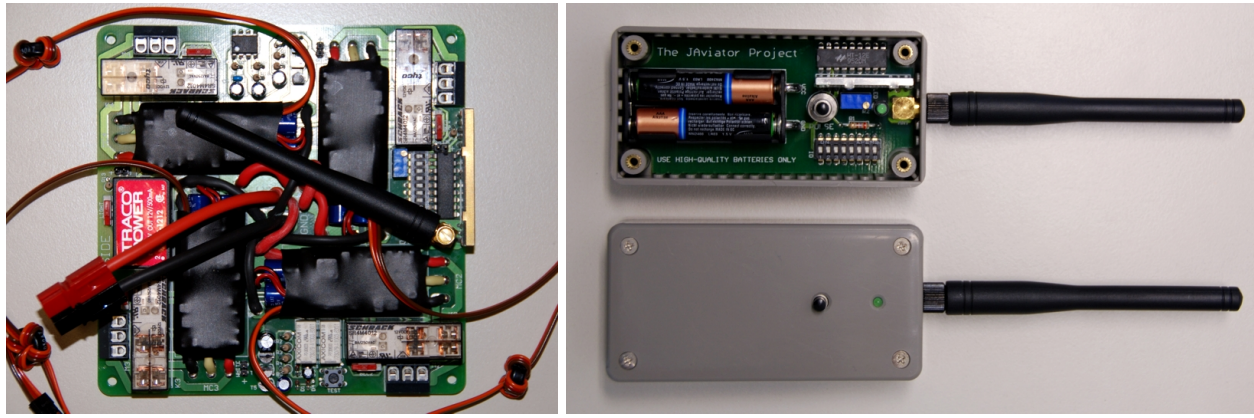


Figure 9. Fully populated power board (left) and emergency shutdown sender (right).

In fulfilling requirement (4), demanding a mechanism for driving signal lights, we equipped the JAviator with navigation lights according to the international rules for aircraft: red light on the left wing, green light on the right wing, and flashing beacons at the front and rear end. Primarily incorporated to give our JAviator some fancy note, these lights are coupled individually with the relay control circuit for visualizing the helicopter’s arming status.

Concerning requirement (5), the integration of the four motor controllers on the power board, we were faced with the problem of placing the motor controllers in a way that leads to shortest possible conductor paths between motor controllers, relays, and motor connectors. Figure 8 (left) illustrates the two-layer layout developed for the printed circuit board depicted in Figure 8 (right). There are four identical units comprising of motor controller, relay, and motor connector, located symmetrically in the four corners of the board. According to the standard coloring for PCB routing, color red represents the top layer and color blue the bottom one. It points out that the conductor paths between motor controllers, relays, and motor connectors appear in purple, which is due to the overlay of similar top and bottom routing. In case of the motor controllers’ AC output, this “double routing” prevents the conductor paths from overheating. However, in case of the motor controllers’ DC input, supporting wire bridges needed to be incorporated, indicated by the large holes around the board’s center, for distributing the high current of up to 50A.

Figure 9 (left) shows the power board fully populated with all components, including the four Jeti Spin 33 BEC brushless-motor controllers (black shrink-wrapped boxes). The motor controllers also supply the required power for the onboard computers and all sensors through their BECs. Due to the opportunity of consuming up to 2A per BEC, which is normally intended to drive the receiver and the servos, there is no need to incorporate additional low-voltage regulators for the JAviator’s avionics equipment. In the present figure, the 868MHz antenna is installed directly on the board for demonstration issues, but located externally and connected via a short coax cable when the board is installed on the helicopter. The sender developed for (dis)arming the JAviator is presented in Figure 9 (right). There is also an 8-pole DIP switch located on both the power board (right-middle) and the sender board (right-bottom) for pairing.

II.D. Sensors

The current sensor equipment consists of a MicroStrain 3DM-GX1 IMU and a Devantech SRF10 ultrasonic ranger. The IMU contains angular-rate gyros, accelerometers, and magnetometers for all three axes and provides triaxial orientation data in several formats like quaternions or Euler angles, either gyro-stabilized or instantaneous. The 3DM-GX1 provides an RS232 interface that is not directly supported by the onboard computer system. Therefore, an RS232-to-TTL converter is employed for adapting the signals. Figure 10 (left) shows the IMU (top) and converter (bottom) mounted on the top plate, which is installed on the top ring with the IMU pointing downwards to be located protected inside the JAviator’s body. The sonar sensor uses fully timed echoing for providing distance data in the range of centimeters. Even though this resolution can be considered as rather coarse for performing precise altitude control, this sensor is fairly sensitive regarding obstacles due to its conical echo. Clearly, regarding inertial movements, the best location for the sonar sensor would be in the center beneath the bottom plate. However, since the minimum distance

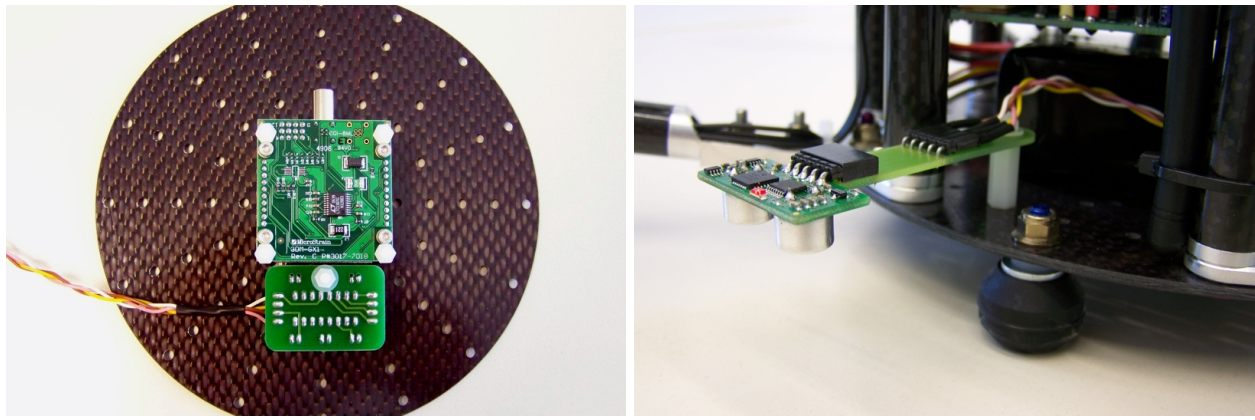


Figure 10. Gyro sensor mounted on top plate (left) and sonar sensor mounted on bottom ring (right).

that can be measured by this sensor is 3cm and there is less than 2cm between the bottom plate and the ground, it is mounted on the bottom ring as illustrated in Figure 10 (right). In order to obtain more precise altitude measurements and to enable automatic position control, the JAviator will be equipped with an additional distance laser sensor and a GPS receiver in the future, described in more detail in Section V.

II.E. Computers

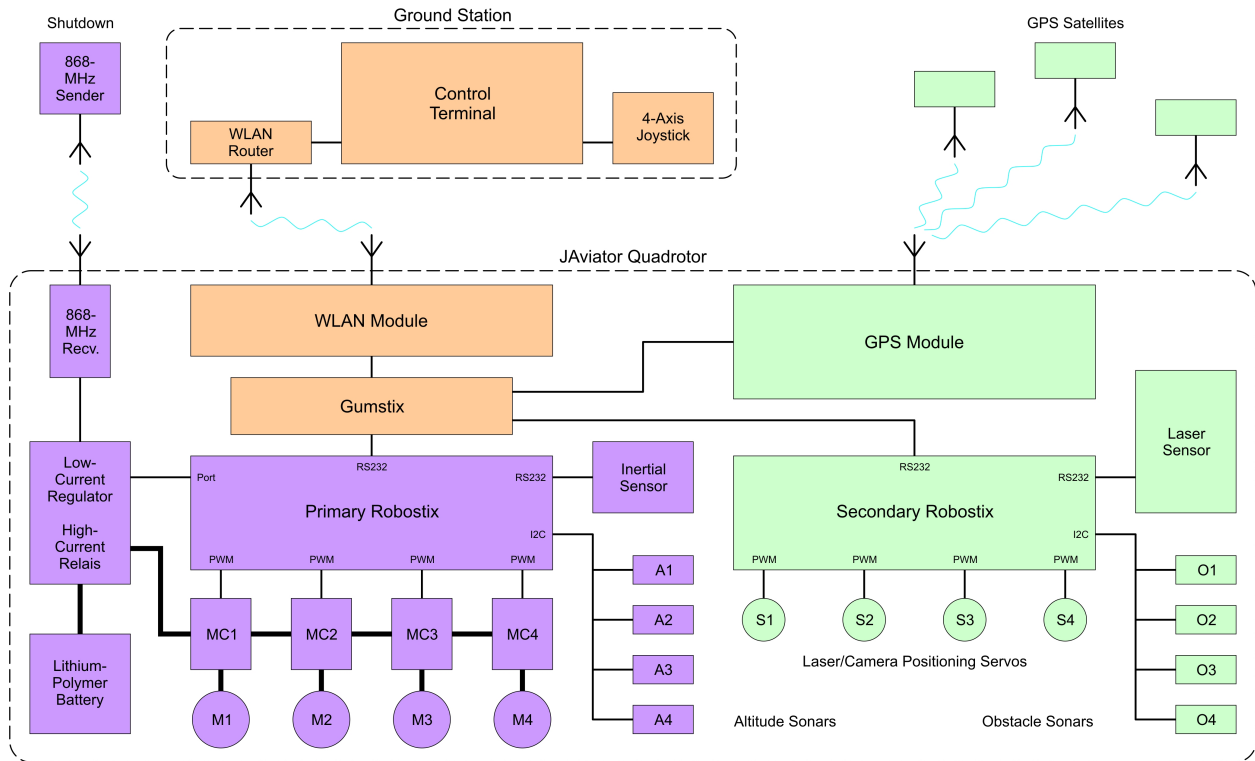


Figure 11. The JAviator's computer system.

The computer system for flying the JAviator consist of an onboard computer system, mounted inside the JAviator's body, and a ground station. Figure 11 depicts the schematic of the computer system including all sensors and actuators as well as the 868MHz emergency shutdown circuitry. Note that the GPS module and the secondary Robostix depicted in this figure is not yet employed, i.e., GPS-based position control and the installation of different obstacle recognition sensors is part of future work.

II.E.1. Onboard Computers

The JAviator is equipped with a so-called Robostix-Gumstix stack, which serves as the onboard computer system. The Robostix, presented in Figure 12 (left), contains an Atmel ATmega128 processor and provides the necessary communication interfaces to connect to the IMU and sonar sensor. Furthermore, it provides the required PWM units for driving the four motors. The Gumstix, shown in Figure 12 (middle), features an Intel XScale PXA255 CPU clocked at 400MHz, 64MB RAM, and 16MB flash memory. The operating system on the Gumstix is a Linux system running a kernel version with real-time extensions and support of high-resolution timers, which we modified to work with the Gumstix. Robostix and Gumstix communicate via an RS232 connection. Figure 12 (right) depicts a Robostix-Gumstix-NetCF stack, where the NetCF module provides an Ethernet connection that we use for tethered flights.



Figure 12. Robostix (left), Gumstix (middle), and Robostix-Gumstix-NetCF stack (right).

II.E.2. Ground Computers

The ground station comprises an IBM ThinkPad T60p laptop computer, a 4-axis joystick for piloting the JAviator, and a separate logging workstation, which collects trace data (time-stamped values of program variables) received from the onboard computer system. All ground computers run Linux and are connected either via Ethernet or WLAN to the onboard computers, depending on whether tethered or untethered flights are conducted.

III. Software

The JAviator’s software runs on the previously mentioned Robostix, Gumstix, and ground system hardware. The software is divided into three layers called Plant, Flight Control System (FCS), and Ground Control System (GCS), as illustrated in the left-hand portion of Figure 13. Each of these layers has been developed independently with well-defined interfaces to the other layers. Moreover, there are multiple, alternative implementations of all layers, which can be switched at compile time without changing any of the other layers.

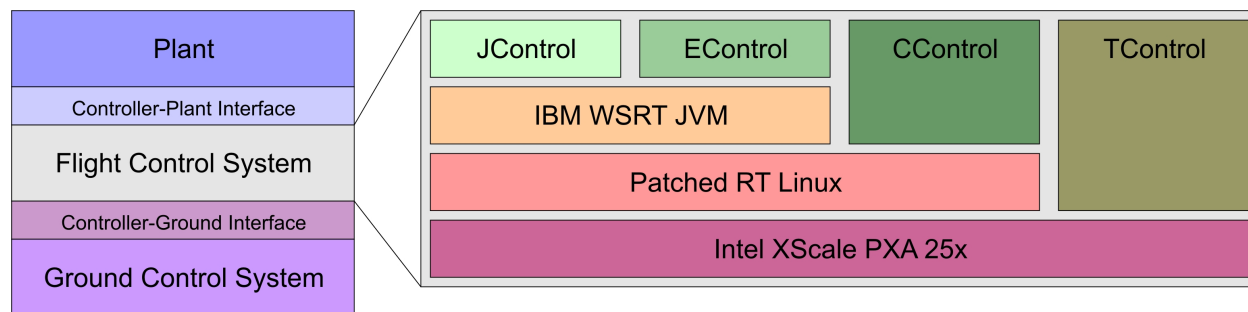


Figure 13. The JAviator’s software architecture.

The lowest layer, in terms of software hierarchy, is the Plant. It either communicates sensor and motor data with the real JAviator, or else implements a simulation of the JAviator’s dynamics. The FCS layer is

the most complex part of the JAviator’s software, and is shown in more detail in the right-hand portion of Figure 13. It implements the flight control algorithms and communicates with both, the Plant and the GCS. Finally, the GCS implements a control terminal for piloting the helicopter and displaying its status.

All three layers communicate via a simple, custom-designed message-passing protocol with the following format: a message is a packet with a header, a variable-size payload, and a checksum to ensure data integrity. Figure 14 depicts the packet format, which starts with the packet delimiter (0xFF) followed by two header bytes. The first of these two bytes contains the type of the packet such as, for instance, sensor data. The second byte encodes the payload size, followed by the payload, checksum, and another, final packet delimiter.



Figure 14. Packet format for communication between Plant, FCS, and GCS.

Due to the protocol-based layer separation, we are able to switch between alternative implementations of all three layers. This allows us to experiment with different control algorithms and implementations, run-time environments, and even programming languages. In the following sections, we provide more details on the different requirements and implementations of the layers described.

III.A. Flight Control System

All FCS versions implement the same control algorithms for stabilizing the JAviator. Attitude control is currently implemented by a simple combination of PD (roll/pitch) and PID (yaw) controllers. We have also experimented with automatic altitude control, which is, however, still prototypical. Within a period of 20ms (50Hz), the FCS receives sensor values from the Plant and computes new motor signals that are sent back to the Plant. When connected to the GCS, the FCS sends a report to the GCS at the end of each period containing the current sensor values and motor signals.

As depicted in the right-hand portion of Figure 13, there are four alternative FCS implementations: (1) a Java-thread-based controller written in Java (JControl), (2) an Exotask-based controller written in Java (EControl), a Linux-process-based controller written in C (CControl), and a Tiptoe-based controller written in C (TControl). Tiptoe is our own real-time operating system.⁴ JControl and EControl run on top of a Java virtual machine (JVM). CControl runs as a single-threaded Linux process. TControl is compiled into the Tiptoe kernel for lack of support of user processes in the current Tiptoe prototype. Except for TControl, we run all controllers on a Linux system with special real-time support for XScale CPUs. TControl, in contrast, runs bare-metal on Gumstix boards.

CControl and TControl are scheduled to be released later this year under an open source license. We are also working with IBM Research on releasing JControl and EControl as well as the Exotasks tools and runtime extensions for the IBM WebSphere Real Time (WSRT) JVM^{23,24} in the near future under a suitable license.

III.A.1. JControl

JControl implements the FCS in Java threads, which run on any JVM that supports Java version 1.4. The control algorithms, the timing, and the communication logic are all implemented in standard Java. Only five procedure calls to the RS232 serial-line driver require implementations as native methods in C. Additionally, JControl is designed to enable code reuse by other components. For instance, JControl’s communication subsystem is reused in the GCS and EControl implementations. Moreover, EControl reuses JControl’s code that implements the functional aspects of the control algorithms.

At runtime, JControl uses three separate threads of execution. In addition to the controller thread, which executes the control algorithm, there are two receiver threads in the communication subsystem. Both receiver threads are identical, except that one polls the Plant interface for incoming packets and the other one polls the GCS interface. If an incoming packet contains information needed by the control algorithm, a new Java object is generated and a reference to this object is stored in the communication subsystem. The controller thread retrieves the latest object at the start of each period. Packets that are not intended for the FCS are forwarded to the opposite communication interface.

Although JControl runs on any JVM with support of Java version 1.4, better real-time results in terms of latency and pause times are achieved with IBM’s WSRT JVM. It invokes the Metronome GC, which shortens

pause times and thus enables predictable execution of user applications written in Java.²² The WSRT JVM has been selected, for instance, by Raytheon as computing platform for the DD(X), the U.S. Navy’s future surface combatant ship program. All our test flights with JControl (and EControl) were conducted with the WSRT JVM modified for support of Exotasks.

III.A.2. EControl

EControl implements the FCS with Exotasks,^{2,3} which facilitate real-time programming in Java. Exotasks are isolated communicating tasks that execute in real time. Determinism is achieved by enforcing spatial isolation of each task and the use of the logical execution time (LET) model²⁰ to communicate between the tasks. Spatial isolation prohibits different tasks to share any state and therefore allows garbage-collecting tasks individually. Communication between tasks is done by sending messages through dedicated channels, which are managed by the Exotask runtime system. Messages are delivered at predefined logical instants of time (LET model) controlled by a scheduler that is part of the runtime system. Currently, the runtime system supports a Giotto-like scheduler²⁰ and an HTL scheduler.²¹ Giotto is a coordination language for distributed control systems based on the LET model. HTL is the most recent successor of Giotto. With the Giotto-like or the HTL scheduler, the Exotask system provides deterministic timing across changes of the hardware and software platform even in the presence of other Java threads. Note that the Exotask system also provides a scheduler framework that allows for other scheduler implementations that may not use the LET model. A detailed performance analysis of the Exotask system is available in our recent work.^{2,3}

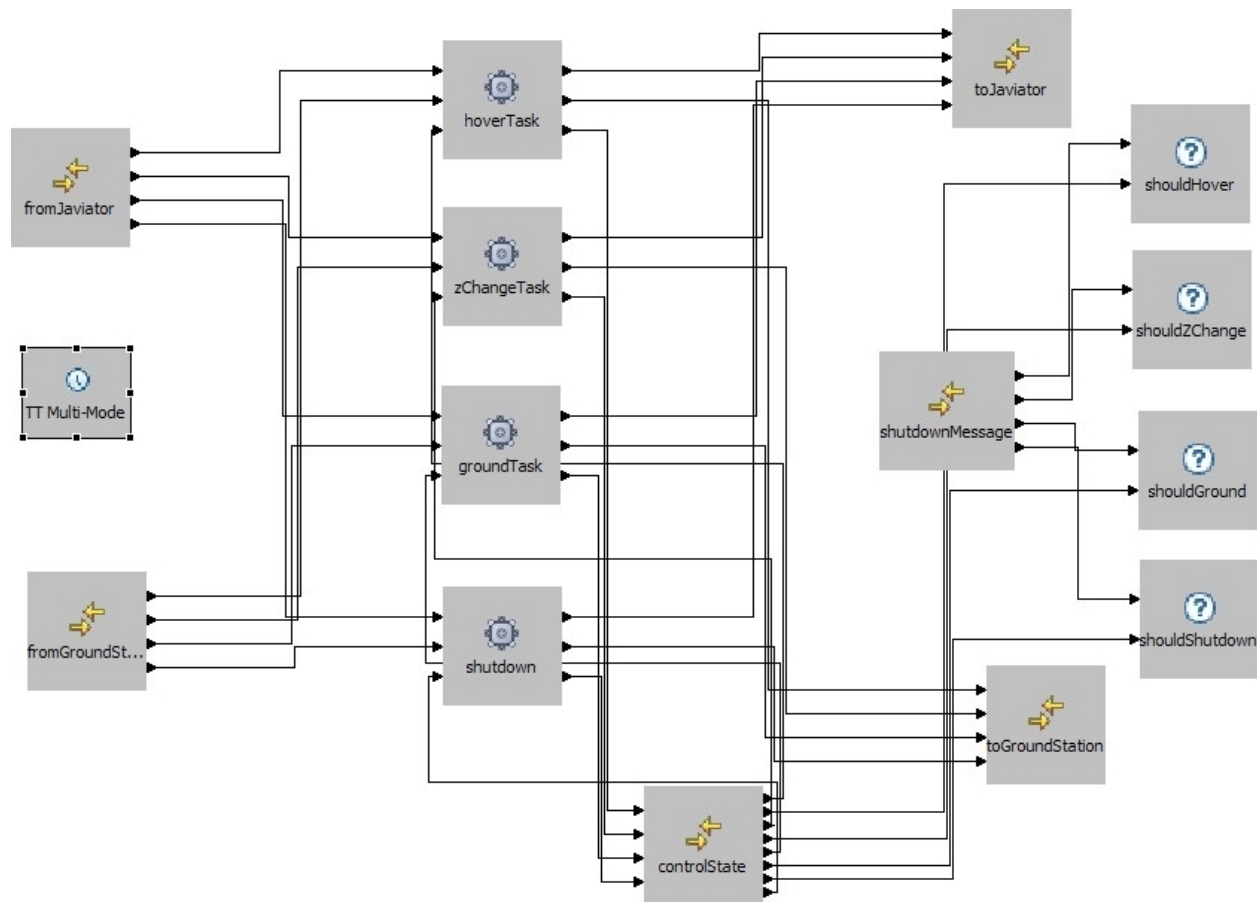


Figure 15. Exotask Graph of EControl in the graphical editor. This picture is taken from our recent publication.²

The tasks of a program in combination with their communication links result in a graph, where the nodes represent the tasks and the edges represent the communication connections. The graph of an Exotask program is called Exotask Graph, which is implemented either in Java source classes that follow a certain design pattern or via a graphical editor. Figure 15 is a screenshot of the JAviator’s Exotask Graph displayed in the graphical editor. This editor is shipped with the Exotask development system as an extension to

Eclipse, a popular integrated development environment for Java. While creating the Exotask Graph, the graphical editor performs additional error checks and, after successfully validating the graph, generates Java source code.

III.A.3. CControl

CControl implements the control algorithm in a single Linux process. It starts the control loop after the communication channels have been setup and initiated successfully. The control loop reads the latest sensor values from the communication subsystem and computes the new motor signals by invoking the PD/PID controllers described above. The new motor signals are then sent to the Plant by the communication subsystem.

Similar to JControl, CControl uses a communication subsystem that performs all I/O-related tasks. In contrast to JControl, however, CControl does not employ any additional threads of execution. More precisely, CControl's communication subsystem provides a procedure with a deadline as argument. After the control algorithm computation, the control loop calls this procedure with the start of the next period as argument. The communication subsystem is then active until the end of that period. It starts by sending all outgoing messages through the correct connections and then polls the incoming channels for new packets. All packets are parsed and data intended for the control algorithm are stored in a buffer. Data received from the GCS and intended for the Plant are forwarded without modifications. Similarly, packets that are received via the Plant connection intended for the GCS are also forwarded without any modification.

CControl implements a special framework to abstract the system-dependent parts of the controller. This design supports the execution of CControl on different systems by changing only the low-level implementation of the system-dependent modules, for instance, the I/O operations of the communication subsystem and the timing operations.

III.A.4. TControl

TControl is a port of CControl to our own real-time operating system called Tiptoe.⁴ The current version of TControl is hard-wired into the Tiptoe kernel for lack of support of user processes. Nevertheless, we were able to implement the FCS with Tiptoe, run it natively on the Gumstix, and fly the JAviator.

We are currently working on the first Tiptoe release, which will offer better support of user processes. The Tiptoe release will also include an adapted version of TControl as an example application. TControl will then be implemented in the so-called workload-oriented programming model⁷ of Tiptoe, which abstracts concurrency-dependent process execution times into concurrency-independent process response times, even on the level of individual process actions such as system and procedure call invocations. Workload-oriented programming is a design methodology for specifying throughput and latency of real-time software processes on the level of individual process actions. The key programming abstraction is that the workload involved in executing a process action, such as a system or procedure call, fully determines the action's response time, independently of any previous or concurrent actions. The model thus enables sequential and concurrent real-time process composition while maintaining each action's workload-determined real-time behavior. We plan to implement TControl in the workload-oriented programming model by specifying timing constraints on individual actions of the controller. Actions such as reading the sensors and updating the actuator signals have latency-oriented timing requirements while log and trace data usually need throughput guarantees. In the workload-oriented programming model, we are able to specify these different constraints within the control loop itself and guarantee the correct timely execution of each controller action on top of Tiptoe.

Similar to CControl, TControl connects to the Plant via a dedicated RS232 serial line. However, the Tiptoe kernel does not support TCP/IP, which is used to communicate with the GCS. Instead, Tiptoe multiplexes all TCP/IP traffic in real time in custom-designed Ethernet frames onto a dedicated Ethernet connection to another Gumstix host running our real-time enhanced version of Linux. The host machine demultiplexes all incoming traffic and relays it to regular wired or wireless Ethernet connected to the GCS.⁵ The other direction works similarly.

III.B. Plant

The Plant provides an abstraction of sensors and actuators to the FCS. It either communicates sensor and motor data with the real JAviator (Physical Plant), or else implements a simulation of the JAviator's

dynamics (Simulated Plant). The Physical Plant is a program written in C that runs on the Robostix and implements device drivers for sensors and actuators. The Simulated Plant is a program written in Java that runs on any JVM.

III.B.1. Controller-Plant Interface

The Controller-Plant interface defines the communication protocol between the FCS and the Plant. Only the FCS initiates communication. The FCS requests new sensor values by sending a packet containing new motor signals to the Plant. In response to this packet, the Plant generates a packet containing the current sensor values and sends it back to the FCS. The Plant keeps track of the time when packets are received and slowly shuts down the motors for safety if packets have not been received for a certain amount of time (200ms in our current setting).

III.B.2. Physical Plant

The Physical Plant on the Robostix periodically polls all sensors for new values and buffers them locally for remote requests by the FCS. The rotational speed of the four motors is controlled by generating individual PWM signals according to the motor signals received from the FCS. The physical connection between the Physical Plant and the FCS is via an RS232 serial line.

III.B.3. Simulated Plant

The Simulated Plant, also called the MockJAviator, is a simulation of the real JAviator. It receives the motor signals from the FCS and, in response, computes new sensor values according to an estimation of the real JAviator's behavior. To ensure compatibility with the FCS, the device-independent aspects of the Controller-Plant Interface implemented in the MockJAviator are the same as in the Physical Plant. However, instead of an RS232 serial line, the MockJAviator uses a TCP/IP connection to communicate with the FCS. Therefore, the FCS implements the device-dependent aspects of the Controller-Plant Interface not only on RS232 but also on Ethernet. An advantage of this design is that the MockJAviator can run either locally on the same machine as the FCS or remotely on some other machine.

III.C. Ground Control System

The GCS consists of a control terminal for piloting the helicopter and displaying its status, and a logging system, which collects trace data (time-stamped values of program variables) received from the onboard computer system.

III.C.1. Controller-Ground Interface

The Controller-Ground Interface defines the communication protocol between the FCS and the GCS. Again, only the FCS initiates communication. The FCS requests new navigation data by sending a packet containing current sensor data (received from the Plant) to the GCS. In response to this packet, the GCS generates a packet containing new navigation data and sends it back to the FCS. The sensor data packet sent by the FCS contains roll, pitch, yaw, and altitude values and their first derivatives as well as x- and y-position data and their second derivatives. Upon receiving a sensor data packet, the GCS responds with a navigation data packet containing new values for roll, pitch, yaw, and altitude. Both sensor and navigation data use the same packet format, as described previously and depicted in Figure 14.

III.C.2. Control Terminal

The Control Terminal, presented in Figure 16 (middle), displays the sensor data received from the FCS graphically and provides a means for piloting the connected Plant, either the real JAviator or the MockJAviator. The main functionality lies in monitoring and modifying the Plant's attitude and altitude. This is accomplished with four meters that indicate the "current" and "desired" roll, pitch, yaw, and altitude values through red and green needles, respectively. In case of the altitude meter, it is also possible to display and set the thrust directly without involving an altitude controller. Except for the yaw meter, each meter contains a safety option that allows to set a limit for the green needle. Accordingly, the user controls the



Figure 16. 3-D simulation of grounded JAviator (left), the JAviator Control Terminal (middle), and 3-D simulation of airborne JAviator (right).

connected Plant by changing the green needles, either by using the keyboard or, preferably, via a 4-axis joystick. Because the red needles refer to the Plant, they cannot be influenced by the user. The Control Terminal also displays the following controller information.

- *Mode* indicates the current mode of the controller (Ground, Ascend, Hover, Descend, or Halt).
- *Left, Right, Front,* and *Rear* are indicators that change their color depending on the current controller state (adjusting for roll, pitch, and/or yaw deviation that is out of tolerance).
- *Offsets* are composed of displaying both the relative motor offsets (differences between the previous and current motor signals) and the partial motor offsets (contribution of the roll, pitch, yaw, and altitude controller).
- *Signals* represent the four computed values that are assigned currently to the motors.

Further functionalities provided by the Control Terminal include a test mode for dynamically activating a specific controller module. This helps to test controller code that is under development. Another feature of the Control Terminal is a means for dynamically adjusting the controllers. More precisely, each parameter of each controller can be changed arbitrarily during flight without the need for landing and re-programming. There is also an emergency shutdown button available for stopping the motors. In our effort to provide a more advanced visualization and interaction with the connected Plant, the Control Terminal has been extended with a 3-D environment. This feature enables visual comparisons of the airborne JAviator against the computed flight behavior. In order to allow for studying flight behavior, the Control Terminal supports recording and replaying of arbitrary flights. Any recorded flight can thus be visualized while replaying. Moreover, the 3-D environment provides the opportunity to view a flight from different perspectives in the 3-D scene, freely adjustable via the rotation, scaling, and translation settings. There are also activatable helper planes that describe the desired attitude and altitude of the JAviator. These planes are activated in both Figure 16 (left) and Figure 16 (right) depicting the grounded and airborne JAviator, respectively. In this sense, the 3-D representation extends the navigation environment by offering full 3-D motion tracking and close-to-reality JAviator visualization effects.

III.D. Ground Logging System

In order to study and improve flight stability and system performance, we have extended the GCS with a logging system that enables real-time tracing during flights. This logging system makes use of IBM's TuningFork,²⁵ an Eclipse-based performance analysis and visualization tool for real-time applications with support for Java, JVM, C++, and Linux. TuningFork features a powerful data processing mechanism and user interface. It was designed originally for diagnosing IBM's real-time JVM and real-time Linux. Nonetheless, because of its scripting capabilities, it can also be used for tracing user-specified system data.

Connectivity within the GCS environment is established via sockets. All logged data and events are saved in a trace file together with the current controller parameters. Due to the additional parameter saving, it



Figure 17. IBM’s TuningFork application displaying traced JAviator data.

is possible to restore the exact controller setup for any logged flight. We are interested in both the system-specific values, such as memory mutator utilization and garbage collection activity, as well as the sensor and actuator data transmitted from and to the JAviator. TuningFork’s logging mechanism thus allows us to analyze and interpret various data streams in both real-time and hindsight. Figure 17 shows the TuningFork application displaying sensed roll, pitch, and altitude data in its visualization window.

IV. Experiments

IV.A. Propulsion Tests

In the endeavor to verify our estimations regarding thrust generation and flight times, we conducted several experiments with a fully assembled JAviator. The motors and the entire avionics equipment were powered during all tests via a single, onboard Thunder Power 4-cell lithium-polymer battery with a capacity of 6300mAh. The JAviator weighs a little less than 2.2kg in its ready-to-flight configuration, as indicated by the spring scale’s display in Figure 18 (left). We tested two different gearings (1:6 and 1:5) to reveal the propulsion system’s maximum capability. To do so, we placed the JAviator on a round launch pad with a small hole in the middle, which we normally use for tethered flights. We measured thrust with one end of a nylon rope attached to the center of the helicopter’s bottom plate, put through the hole in the launch pad, guided over a pulley, and the other end attached to a high-precision spring scale. The rope had a play of approximately 10cm, hence enabling the JAviator to lift off and hover freely, illustrated in Figure 18 (middle). Due to this setup, we were able to measure the thrust generated in addition to lifting the helicopter’s curb weight. The maximum thrust for both the 1:6 and 1:5 gearing was determined by letting the helicopter pull the rope with maximum power, but, for safety reasons, for no longer than 10s. During this time span the spring scale was oscillating around some mean value that can be found in Table 1. After determining the maximum thrust, some heavy weight was placed on top of the JAviator to prevent it from lifting off during the flight tests. In order to determine the maximum battery service times for both gearings and for both the helicopter’s curb weight with and without maximum payload, for each test the battery was fully drained until one of the motors dropped off, i.e., was shut down by its motor controller. Figure 18 (right) shows a rotor spinning at full speed, which, in case of the 1:5 gearing used, achieves almost 2500rpm, resulting in a generated thrust of almost 1.4kg. Note that the blades stay in horizontal line and do not bend up, which indicates that their mechanical limit is not reached.

Table 1 summarizes our measurements and gives an overview of the achieved lift capabilities and battery service times. It points out that, even though the battery service times drop significantly when demanding full thrust, the JAviator can provide both either high payload capabilities of up to 3.2kg or relatively long flight times. Regarding the measurements without payload it is remarkable that, compared to the 37min battery service time achieved with the weaker 1:6 gearing, the same measurement performed with the stronger 1:5 gearing resulted in 39min flight time. We assume that this result is due to certain tolerances associated with the charger, equalizer, and/or battery.

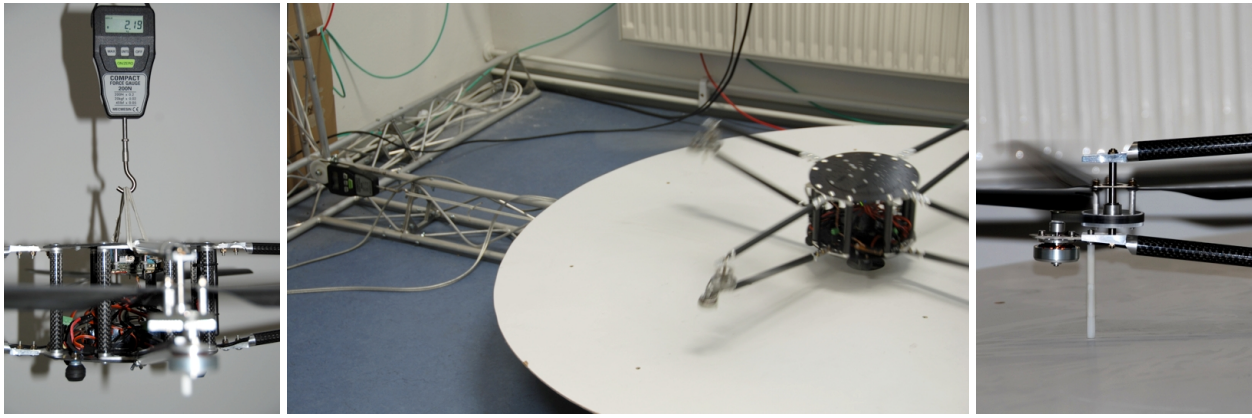


Figure 18. Measuring JAviator’s curb weight (left), determining maximum additional lift (middle), and rotor spinning at maximum speed (right).

Experiment	Gearing	Thrust [%]	Lift [kg]	Time [min]
JAviator without payload	1:6	43	2.2	37
JAviator + lifting 2.4kg	1:6	100	4.6	11
JAviator without payload	1:5	38	2.2	39
JAviator + lifting 3.2kg	1:5	100	5.4	8

Table 1. Measured lift capabilities and battery service times with different gearings and thrusts.

IV.B. Flight Tests

At this point, we can fly the JAviator with automatic attitude control but manual positioning via a standard 4-axis joystick. We currently use PD control for roll and pitch, and PID control for yaw. We have also experimented with automatic altitude control but decided to use manual thrust control in our experiments for safety reasons. In order to change the controllers’ behavior during flight without the need for landing and re-programming, we have implemented a means for dynamic adjustment of all controller parameters. This helped in finding a basic parameter setting that served as a starting point for fine-tuning each individual controller. The JAviator was powered autonomously by its onboard battery during all experiments. However, we had to connect it via Ethernet cable with the ground station due to unreliable connectivity via the onboard WLAN module. We discovered some incompatibility of the WLAN driver with the real-time patches applied to the Linux operating system, but have not solved this problem yet. Nevertheless, we performed Ethernet-tethered indoor flights to adjust the controllers and also to reveal the JAviator’s true payload capability. Within a limited space of roughly 5 times 5 meters, we succeeded in flying the JAviator several times for several minutes without interruption, as demonstrated in Figure 19 (left). To study the tradeoff regarding increased payload and decreased flight performance, we started our payload experiments by mounting two additional 6300mAh batteries beneath the bottom plate. Each battery weighs exactly 458g, which, together with the carbon-fiber plates and mounting screws, resulted in a payload of nearly 1kg. Except for an increased lift-off thrust requirement from 40% to 55% and a somewhat slower response time, no significant reduction in flight performance could be observed, shown in Figure 19 (middle). Therefore, and due to our propulsion test results, the question arose, what if lifting the JAviator’s curb weight, which would account for a payload of 2.2kg? To this end, we mounted another completely assembled JAviator beneath the first one, but rotated it by 45 degrees to have the top one’s rotors not in vertical line with the bottom one’s. Although the lift-off thrust increased to roughly 80%, there was still enough thrust reserve for controlling this “double-deck” JAviator. The flight performance was reduced significantly and turned out to demand extraordinary reaction times from the user. However, we have demonstrated that our JAviator is capable of carrying its entire curb weight, and at the same time, provides sufficient freedom for controlled flying, as illustrated in Figure 19 (right). Note that the same controller settings were used throughout all payload experiments. We believe that there is still potential for improving the JAviator’s high-payload flight performance by appropriately adjusting the controller parameters for this purpose.

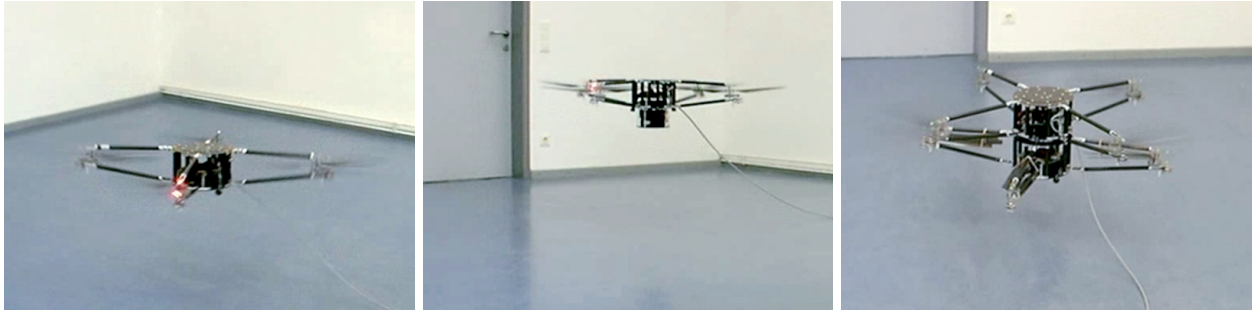


Figure 19. Flying without payload (left), carrying two batteries (middle), and carrying a second JAviator (right).

IV.C. Software Tests

The JAviator can operate autonomously between 8min and 39min, depending on the payload and gearing. However, for testing the entire software system and verifying its real-time capabilities over a longer period of time, a tethered-JAviator setup was used. In this setup, the JAviator was powered externally by a DC power supply, adjustable in the range of 0A to 50A at 0V to 80V, and connected to the ground station via an Ethernet cable. In order to prevent the JAviator from drifting around during tethered flights, a launching tower was used. This tower basically consists of four vertical 100cm metal pipes arranged such that the JAviator’s body fits between them. The JAviator was therefore restricted to horizontal movements within a radius of approximately 5cm, but could move vertically in the range of 0cm to 80cm.

All four pluggable controllers (JControl, EControl, CControl, and TControl) were tested for proper operation by comparing the real JAviator against the simulation. In case of the Java-based controllers JControl and EControl, TuningFork traces were generated to enable detailed analyses of the entire software system. With JControl, the first pure Java controller developed for the JAviator, the control behavior basically suffices for flying the helicopter. However, with the more advanced EControl based on Exotasks, we were able to demonstrate time portability and hence platform independence. In particular, we tested EControl in the presence of different software workloads (and on different hardware) and verified close-to-identical real-time behavior.^{2,3} In this context, Figure 20 (left) depicts the interarrival times for the `fromJaviator` Exotask, which is part of the JAviator control program and the first one to run in each 20ms controller period. In the present case, `fromJaviator` is executed on an XScale PXA255 CPU, whereas Figure 20 (right) illustrates the execution of this Exotask on the same CPU when a concurrently running thread is allocating 256KB of memory per second to exercise the runtime system’s garbage collector. Both figures use a linear scale ranging from 14ms to 25ms on the X-axis and a logarithmic scale on the Y-axis. The two figures will also appear in an upcoming publication.³

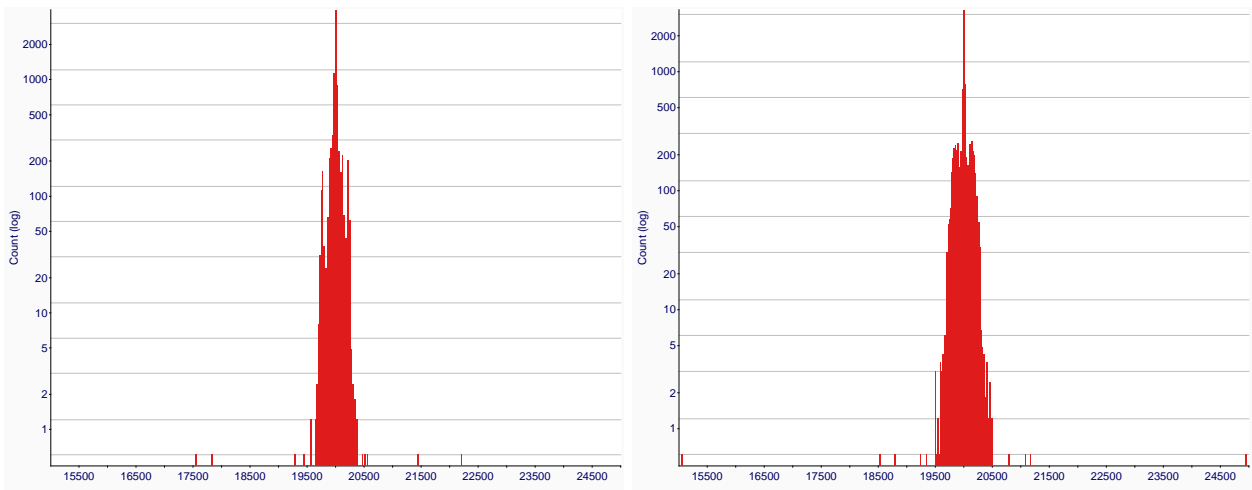


Figure 20. Interval between `fromJaviator` task executions on an XScale PXA255 platform when not performing any concurrent allocation (left) and when concurrently allocating 256KB of memory per second (right).

V. Future Work

In order to stabilize the JAviator’s position in the plane and to enable autonomous control, we intend to add a Septentrio AsterRx1 GPS receiver to the onboard computer system. This differential-GPS receiver, depicted in Figure 21 (left), provides position data with an accuracy of $\pm 10\text{cm}$ at an update rate of 10Hz. We expect that this accuracy will suffice for achieving nearly drift-free position stability while the JAviator is operating autonomously.

We experienced that performing altitude control is much harder than attitude control and it is of major importance to have sufficiently precise altitude data. The resolution of the sonar sensor involved currently lies in the range of centimeters, which makes it difficult to implement a satisfactory altitude controller. Therefore, we plan to equip the JAviator with an additional high-precision distance laser sensor. Figure 21 (right) shows the Dimetix LSM2-15 laser sensor module we use for this purpose. This distance sensor offers an accuracy of $\pm 1.5\text{mm}$ and a measuring range of up to 100m on natural surfaces without a target plate.

Since the complete LSM2-15 module weighs only 75g, we also plan to apply a second one mounted pivotably on top of the JAviator for measuring distances in the plane. Compared to the relatively short range of most small sonar sensors that rarely exceeds 5m, the LSM2-15 laser sensor allows to perform obstacle recognition within a radius of 100m.

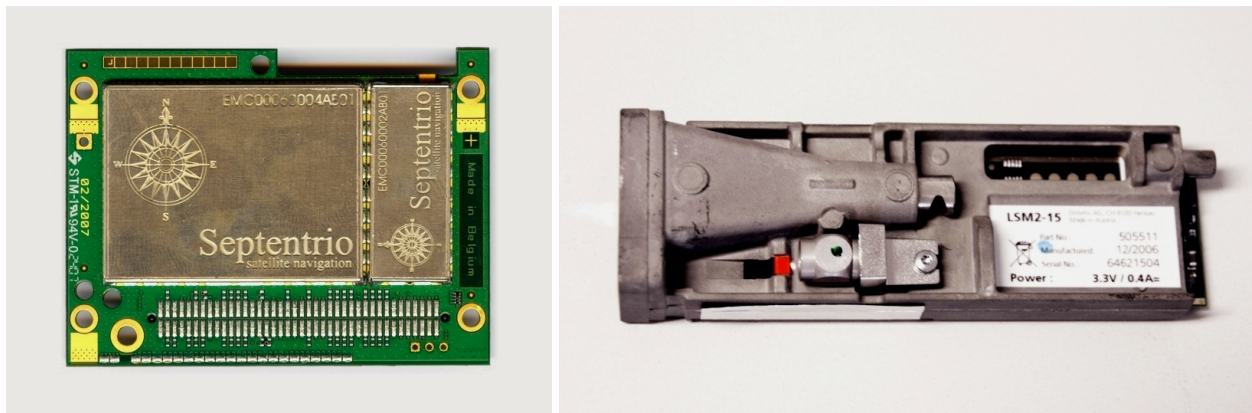


Figure 21. AsterRx1 GPS receiver (left) and LSM2-15 distance laser sensor (right).

VI. Conclusions

We have presented the JAviator, a high-performance quadrotor model helicopter with a fully symmetrical frame design, support of high payloads, and high-level programming capabilities. We have designed and manufactured the JAviator completely from scratch using only high-performance materials such as carbon fiber, aircraft aluminum, and medical titanium. The version presented here has already been produced in a small series consisting of five helicopters.

The JAviator may be programmed in Java using a real-time extension called Exotasks,^{2,3} which we developed in collaboration with IBM Research. Exotasks enable time-portable programming of high-performance, hard real-time applications in Java. Time-portable programs do not change their relevant real-time behavior across different hardware platforms and software workloads. With Exotasks we are able to fly the JAviator using different hardware and workloads without changing any of the application-level real-time software. Except for device-level sensing and actuating code written in C, all Exotask-based software was written in Java.

Apart from the Exotask system, the JAviator also serves as a test platform for other software projects such as our own real-time operating system called Tiptoe,⁴ which is in a prototypical stage and meant to provide even stronger forms of time portability than Exotasks. The goal of the Tiptoe project is to develop systems software that supports the execution of real-time processes whose temporal behavior, even including system-level aspects such as their I/O communication and memory management, can be predicted per process, independently of the other processes.

In order to demonstrate the JAviator’s programming capabilities, we have implemented its flight control system in Java using Exotasks and developed a pure Java version as baseline for the Exotask version. Moreover, we have reimplemented the Java version of the flight control system in C for Tiptoe, which does not support the execution of Java programs in its current implementation, as well as for Linux with real-time patches applied as baseline for the Tiptoe version. We are able to fly the JAviator using all four controller implementations.

References

- ¹Auerbach, J., Bacon, D., Craciunas, S., Iercan, D., Kirsch, C., Rajan, V., Röck, H., and Trummer, R., “The JAviator Project,” javiator.cs.uni-salzburg.at.
- ²Auerbach, J., Bacon, D., Iercan, D., Kirsch, C., Rajan, V., Röck, H., and Trummer, R., “Java Takes Flight: Time-portable Real-Time Programming with Exotasks,” *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, 2007.
- ³Auerbach, J., Bacon, D., Iercan, D., Kirsch, C., Rajan, V., Röck, H., and Trummer, R., “Low-Latency Time-portable Real-Time Programming with Exotasks,” To appear in *ACM Transactions on Embedded Computing Systems (TECS)*.
- ⁴Craciunas, S., Kirsch, C., Payer, H., Röck, H., Sokolova, A., Stadler, H., and Staudinger, R., “The Tiptoe Project,” tiptoe.cs.uni-salzburg.at.
- ⁵Stadler, H., Master’s thesis, University of Salzburg, Salzburg, Austria, 2008.
- ⁶Craciunas, S., Kirsch, C., Payer, H., Sokolova, A., Stadler, H., and Staudinger, R., “A Compacting Real-Time Memory Management System,” *Proceedings of the USENIX’08 Annual Technical Conference*, 2008.
- ⁷Craciunas, S., Kirsch, C., Röck, H., and Sokolova, A., “Real-Time Scheduling for Workload-oriented Programming,” Tech. Rep. 2008-02, University of Salzburg, 2008.
- ⁸Bouabdallah, S., Becker, M., de Perrot, V., and Siegwart, R., “Toward Obstacle Avoidance on Quadrotors,” *Proceedings of the International Symposium on Dynamic Problems of Mechanics (DINAME)*, 2007.
- ⁹Hoffmann, G., Huang, H., Waslander, S., and Tomlin, C., “Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment,” *Proceedings of the AIAA Guidance, Navigation, and Control Conference (GNC)*, 2007.
- ¹⁰Draganfly Innovations Inc., *RC Planes, Helicopters, and Blimps*, www.rctoys.com.
- ¹¹Arning, R. and Sassen, S., “Flight Control of Micro Aerial Vehicles,” *Proceedings of the AIAA Guidance, Navigation, and Control Conference (GNC)*, 2004.
- ¹²Chen, M. and Huzmezan, M., “A Combined MBPC/2 DOF H_∞ Controller for a Quad-Rotor UAV,” *Proceedings of the AIAA Guidance, Navigation, and Control Conference (GNC)*, 2003.
- ¹³Hoffmann, G., Rajnarayan, D., Waslander, S., Dostal, D., Jang, J., and Tomlin, C., “The Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control (STARMAC),” *Proceedings of the Digital Avionics Systems Conference (DASC)*, 2004.
- ¹⁴Tayebi, A. and McGilvray, S., “Attitude Stabilization of a VTOL Quadrotor Aircraft,” *IEEE Transactions on Control Systems Technology (TCST)*, Vol. 14, No. 3, 2006, pp. 562–571.
- ¹⁵Ponds, P., Mahony, R., and Corke, P., “Modelling and Control of a Quad-Rotor Robot,” *Proceedings of the Australasian Conference on Robotics and Automation (ARAA)*, 2006.
- ¹⁶Ponds, P., Mahony, R., Gresham, J., Corke, P., and Roberts, J., “Towards Dynamically-Favourable Quad-Rotor Aerial Robots,” *Proceedings of the Australasian Conference on Robotics and Automation (ARAA)*, 2004.
- ¹⁷Bollella, G., Gosling, J., Brosgol, B., Dibble, P., Furr, S., Hardin, D., and Turnbull, M., *The Real-Time Specification for Java*, The Java Series, Addison-Wesley, 2000.
- ¹⁸Armbruster, A., Baker, J., Cuneo, A., Flack, C., Holmes, D., Pizlo, F., Pla, E., Prochazka, M., and Vitek, J., “A Real-Time Java Virtual Machine with Applications in Avionics,” *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 7, No. 1, 2007, pp. 1–49.
- ¹⁹Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., and Culler, D., *Ambient Intelligence*, chap. TinyOS: An Operating System for Sensor Networks, Springer, 2005, pp. 115–148.
- ²⁰Henzinger, T., Horowitz, B., and Kirsch, C., “Giotto: A Time-triggered Language for Embedded Programming,” *Proceedings of the IEEE*, Vol. 91, No. 1, January 2003, pp. 84–99.
- ²¹Ghosal, A., Henzinger, T., Iercan, D., Kirsch, C., and Sangiovanni-Vincentelli, A., “A Hierarchical Coordination Language for Interacting Real-Time Tasks,” *Proceedings of the ACM International Conference on Embedded Software (EMSOFT)*, 2006.
- ²²Bacon, D., Cheng, P., and Rajan, V., “A Real-Time Garbage Collector with Low Overhead and Consistent Utilization,” *Proceedings of the ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, 2003.
- ²³IBM Corporation, *WebSphere Real Time User’s Guide*, 1st ed., 2006.
- ²⁴Auerbach, J., Bacon, D., Blainey, B., Cheng, P., Dawson, M., Fulton, M., Grove, D., Hart, D., and Stoodley, M., “Design and Implementation of a Comprehensive Real-Time Java Virtual Machine,” *Proceedings of the ACM International Conference on Embedded Software (EMSOFT)*, 2007.
- ²⁵IBM Corporation, *TuningFork Visualization Tool for Real-Time Systems*, www.alphaworks.ibm.com/tech/tuningfork.