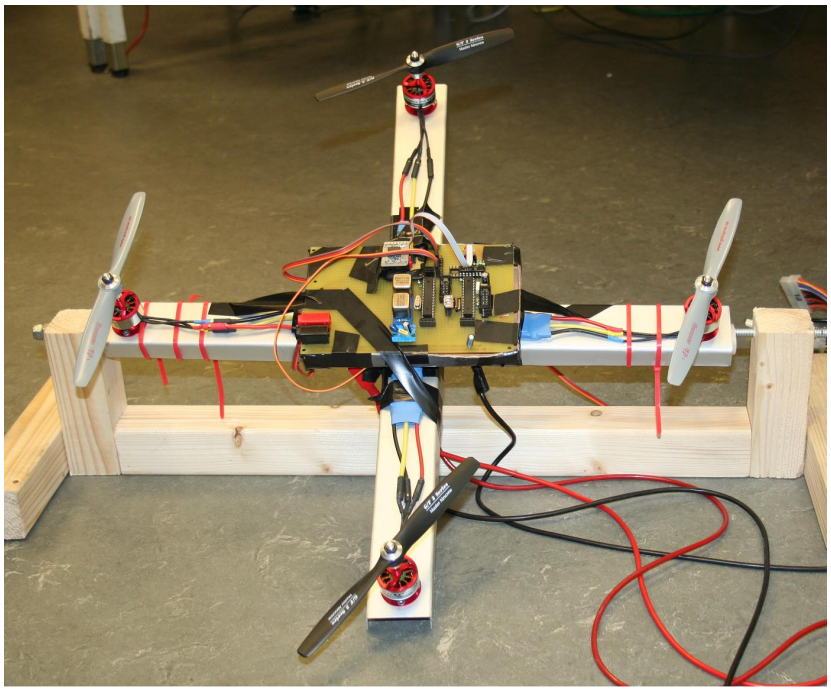
Projekt Quadcopter

Tore Paulsson & Björn Lineke

18 maj 2008



1 Abstract

This project consists of constructing a four propellered helicopter. The propellers are each positioned on the end of a cross made of aluminium. The helicopter should in finished state be able to hover on a beforehand chosen altitude. It should be able to maintain control even if wind or other types of load disturbances enter the system. The main computer force is two atmega88s, the slave collecting measurement and the master calculating motor speed variables by using the measurement data. The modules used in the project is a 2D-Accelerometer, two gyros and a IR-module. How these work and how they are used is included in this report under the title Implementering.

Innehåll

2 Inledning

Projektet går ut på att bygga en flygande helikopter med fyra motorer. Själva stommen är två aluminiumbalkar som bildar ett kors, på varje hörn sätts en motor. Styrlogiken och datainsamling sker av komponenter på mitten av konstruktionen. Huvudprocessorn som används är två stycken ATmega88or, en slave som samlar in majoriteten av all mätdata och en master som innehåller själva reglerloopen och som styr motorerna. Helikoptern skall när allt är klart kunna stå stilla på förutbestämd höjd och regleras för att klara av vindar och liknande belastningar.

Att reglera detta krävs några olika mätuppgifter, dessa tillhandahålls av en 2D-Accelerometer, två stycken gyromoduler och IR-modul. Koden för avläsning av de olika komponenterna finns i appendix B med förklarande kommentarer. Huvudidén för avläsning och hur de olika modulerna fungerar hittar man under avsnittet Implementering.

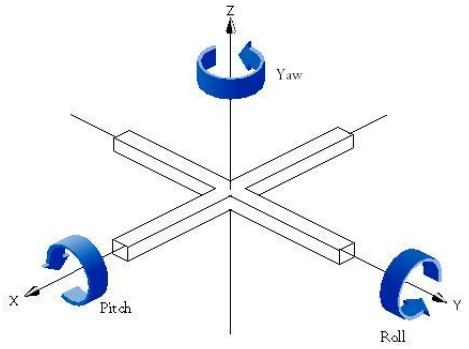
Problem uppkom under konstruktionen som ledde till att huvudidén fick ändras och det kom tillägg till konstruktionen. Vissa av dessa var enklare att lösa medan vissa andra var för svåra. Största problemet som uppkom i konstruktionsfasen var bygget av motorstyrning till de borstlösa motorerna. Då detta kräver mycket bra reglering och avläsning eftersom systemet innehåller mycket störningar.

3 Specifikation

Färdiga konstruktionen skall kunna sväva på förutbestämd höjd, runt 10-60 cm utan att driva i sidled eller rotera. Klara av störningar i form av laststörningar som t.ex. vind eller om man knuffar till den. Ha möjlighet att installera blåtandsmodul för att styra om tiden räcker till.

4 Lösningsförslag

QuadCoptern kommer att behöva regleras och styras för pitch, roll, yaw och höjd, se figur 1. Pitch och roll kommer att kunna lösas genom att använda en accelerometer och två gyron. Accelerometern ger vinkel från normalläget i 2-led (x och y) och gyrot ger vinkelhastigheten i ett led, därför behöver man två stycken gyron. Man skulle kunna använda gyrosignalen och integrera för att få en vinkel, men då det kändes mer exakt att använda en accelerometer utöver gyron. För att reglera jaw används gyrot som känner av rotationshastighet. En IR-modul används för att få avläsning i höjdled. Huvudprocessorn blir en ATmega88 som samlar in data och reglerar. Detta bevisade sig senare vara för mycket för en ATmega88 och dels därför kopplades en till ATmega88 in. Den andra faktor var att de båda gyrona som fanns tillgängliga hade samma address varför de inte kunde sitta på samma bus.



Figur 1: Pitch, roll, yaw

5 Implementering

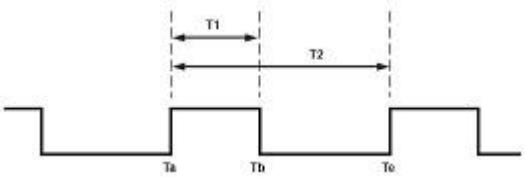
5.1 ATmega88

Två stycken som används för att samla mätdata och beräkna styrsignaler. ATmegorna är externt klockade med en kristall på 20MHz och ingen prescaler. Masterprocessorn får mätvärden från slaven och utifrån dessa beräknar bredden på servopulsen som blir insignal till motorstyrningen.

Processorerna kommunicerar genom en SPI-Bus. Kort förklarat har båda processorerna har ett SPI-register som laddas genom mjukvara och vid skrivning till detta register i mastermjukvaran initieras en överföring, då byter slav och master registerinformation. Är avbrott igångsatt får man avbrott när överföringen är klar och informationen i SPI-dataregistret är då den mottagna informationen. För ytterliggare förklaring hänvisas läsaren till ATmega88as Datablad och AVR151 (se referenser).

5.2 Accelerometer

Använder en ADXL202 för att bestämma lutningen på QuadCoptern, den har en digital utgång som man kan med en räknare beräkna duty cycle. Pulståget ser ut som i figur 2. Duty cycle beräknas som kvoten mellan T1 och T2. Mjukvaran nollställer räknaren vid T_a, sedan tar tider för T_b och T_e och dessa bildar sedan kvoten T1 genom T2.



Figur 2: Pulsbredd accelerometer

Utsignalerna är två stycken, x-axel och y-axel, dessa skiftas varje gång T_e har registrerats. Genom att använda processorns Input Capture Unit kan man ta tiderna på ett smidigt sätt. Då processorn bara har en IC ingång använder vi en Demuxkrets och låter processorn välja vilken signal som skall gå igenom och beräknas. Hur duty cycle används för att bestämma accelerationen kan läsas i ADXL202s datablad och i referenser (AN-604). Då accelerationen är bestämd kan man beräkna med vilken vinkel som accelerometern lutar.

Accelerometern kan känna av accelerationen tack vare en fri massa som är fastsatt med polysilikonfjädrar som ger motstånd mot accelerationskrafter. Då denna massa blir utsatt för en kraft finns samband enligt ekvation 1 som ger accelerationen från fjäderkonstanten, förskjutning i x/y-led och masstyngd.

$$a = \frac{kx}{m} \tag{1}$$

5.3 Gyro

Gyrot som har använts i projektet är ett två axlars vibrationsgyro. Ett vibrationsgyro har små element som vibrerar och av dessa bestämma rotationshastigheten.

För att kunna få ut rotationen i 3 dimensioner så behövs två stycken gyromoduler. För att kommunicera med dessa moduler användes TWI-bussen från ATmegorna, dessvärre har dessa moduler samma anropsadress. För att lösa detta problem så har processorerna vars ett gyro som de kan hämta värden ifrån. Gyrona har en upplösning på 32 counts per grad/sec, vilket kan ge en väldigt exakt rotationshastighet.

Via TWI-bussen hämtas de 16-bitars tal som representerar rotationshastigheten och beräknas sen i processorn. När QuadCoptern står still så läses gyrot av och värdet används sedan som referens när allt är stilla. För att få rotationen i grader/sec beräknas värdena enlig formel 2.

$$Rot_{deg/sec} = \frac{Rot - Rot_{ref}}{32} \tag{2}$$

För att initiera gyrot behöver man läsa från minnet på gyrots EEPROM och sedan skriva in dessa fabriksvärden till gyro delen. EEPROM'en har inte samma adress som gyro delen och kan anropas separat fast på samma buss. Nedan i figur 3 är höljet avtaget från gyrot och man kan se de element som vibrerar.

TWI-bussen som används är en buss där man endast använder två pinnar för att kommunicera och det finns fasta protokoll som beskriver hur kommmunikationen skall gå till. Funktionerna som används när gyrot initieras är att läsa 8-bitar och skriva 8-bitar och hastigheten på bussen under denna process är 100 kHz. När uppstarten av gyrot är gjort så ändras hastigheten till ca 350 kHz för att kunna få in värdena fortare från gyrot. Den ena av de två pinnar är klockan som bestämmer busshastigheten och den andra är data pinnen där bitarna skickas. Mer information finns i databladet för ATmega88 under "Two Wire Interface".

Figur 3: Gyro MG1101

5.4 IR-modul

IR-Modulen som fanns tillgänglig var en SHARP GP2D12, denna skickar ut en analog signal som beskriver avstånd till ett objekt. Genom att rikta denna ned mot marken kan man få en bra avläsning på vilken höjd QuadCoptern ligger. Har ett spann som sträcker sig mellan 10 till 80 cm, vilket täcker kravspecifikationerna. Tyvärr är den analoga amplitudsignalen inte linjär med avståndet, därför det behövs en tabell för att få reda på avstånd i cm. Då vi inte har stort intresse av att veta exakt avstånd till marken och tidbrist görs regleringen på ett inställt värde direkt ifrån AD-omvandlaren.

Figur 4: Sharp GP2D12

5.5 Motorer

Motorerna som har använts är 4 st borstlösa modellflygsmotorer, med en maximal lyftkraft på 0,85 kg vid 11 000 rpm. För att driva dessa vid full effekt krävs 12 V och 10 A per motor. Nedan i figur 5 syns motorn som vi har använt.

Figur 5: Emax outrunner motor

5.6 Motorstyrning

Motorstyrningen styr de fyra motorerna med en simulerad trefas signal. Motorstyrningarna styrs genom en servopuls som skickas från en av processorerna

Figur 6: 30 A motor styrning

6 Utförande

Utvecklingen av QuadCoptern delades in i olika delar som sedan i slutet sattes samman till en komplett maskin. De listas här i ungefärlig ordning, vissa delar jobbades på parallellt. För felsökning användes en UART modul så att man kan under tiden skriva ut variabelvärden till datorn.

6.1 Kravspecifikation

Började med att diskutera vad prototypen skulle klara av, resultatet står under rubriken Kravspecifikation i början av rapporten. Det flödesschema som vi skapade i början och hur resultatet blev finns i appendix C.

6.2 Motorstyrning

Motorstyrningen var det första som började utvecklas, efter en lång tid insågs det att det var för komplicerad återkoppling för att bygga en styrning som fungerade stabilt. Vi kom så långt så att vi fick motorerna att varva upp men återkopplingen som skulle göra att faserna skiftade i rätt tid för att inte motorn skulle komma ur fas fungerade inte bra. Med en switchfrekvens på 16 kHz och strömmar upp till 10 A blev det väldigt mycket störningar och faserna skiftade fel och vi kom fram till att det var ett projekt i sig att få motorstyrningen att fungera bra. Lösningen blev att köpa färdiga motorstyrningar så att arbetet kunde fortsätta. Motorerna drar väldigt mycket ström och av ekonomiska skäl drivs dessa av nätaggregat från datorer som kan leverara ca 10 A/st. I ett senare skede är det tänkt att batterier skall driva motorerna.

6.3 Accelerometer

Parallellt med Motorstyrning utvecklades accelerometermodulen. Denna var lätt att koppla in och sedan behövdes det kodas en del för att få igång allt. Problem uppstod med avbrottsrutiner och felberäkningar eftersom ingen i gruppen hade någon tidigare erfarenhet med C-språket eller att programmera ATmegor. Fixpunktsberäkningar för att beräkna accelerationen från accelerometern blev aldrig fysikaliskt tillförlitliga och därför genom att ta referensvärde (vid noll graders lutning) från duty cycle skulle en enkel regulator kunna utvecklas genom att jämföra med referensvärde.

6.4 IR-Modul

Utgången från IR-Modulen var en analog signal därför initierades en pinne för AD-omvandling. Denna går i Free Running Mode och värdet uppdaterades regelbundet. Detta var den lättaste modulen att implementera i systemet då den inte krävde mycket kunskap om programmering. Sedan då värdet på AD-registret direkt användes som insignal behövdes inget ytterligare göras.

6.5 Gyro

För att att få gyrot att fungera behövde vi få igång TWI-bussen på ATmegan. Innan vi fick det att fungera korrekt hade vi problem med att vi inte fick någon

kontakt alls med gyrot, detta berodde dock bara på att vi körde avläsningen från EEPROM'en med för hög hastighet. För att sedan testa gyrot skrev vi en funktion som pollade gyrot så vi lärde oss hur vi skulle komunicera med det. Svårigheten kom sedan när bussen skulle styras med interrupts. För att få detta att fungera behövdes någon variabel som höll kontroll på var i överföringen man var. Initieringen av gyrot styrs fortfarande via polling, men under drift så anropas funktionen att hämta värden och läses sedan av när värdet har hämtats. På grund av att adressen som gyrot anropas på inte går att ändra och vi behöver kunna läsa av rotationen i tre riktningar, så sitter det ett gyro på varje processor, men ett av gyrona läses bara av i en av riktningarna.

7 Diskussion & slutsats

Då vi inte kunde programmera C har vi under projektets gång lärt oss många nya funktioner. Att kunna leta upp information och lära sig själv är något som man blivit mycket bättre på. Att förstå datablad är mycket enklare än vad det var i början av projektet. Att strukturera upp och dela in ett stort problem i delproblem, vilket är ett måste då man jobbar i grupp. I början av projektet jobbade vi tillsammans mycket med samma problem, detta tog dock för lång tid. När vi sedan delade in uppgifterna och diskuterade lösningsförslag som man sedan jobbade vidare med gick arbetet mycket snabbare.

Bland de många problem som kom var det minst hälften som var små missar och att man varit slarvig när man testade ny kod så att lite av den gamla fanns kvar. Ett exempel på detta var när en i gruppen råkade sätta igång CompareA på en räknare utan att skriva en avbrottshanterare. Detta ledde till att när räknaren nått max varvar och blir 0x00, ges ett avbrott vilket leder till att processorn tappar bort sig. Som följd av detta startade processorn om. Detta fel tog ett bra tag att hitta i koden.

Som det ser ut nu är projektet inte klart innan rapportinlämning. Hårdvaran i QuadCoptern är fungerande men på grund av den avancerande reglertekniken går det inte att få en stabil process. Tankar och idéer kring hur regleringen skall göras finns och kommer publiceras i en projektrapport för kursen Projektkurs i Reglerteknik.

Då man kollar tillbaka på allt som vi i gruppen lärt oss är vi nöjda, men arbetet kommer fortsätta för att få QuadCoptern i helt fungerande skick.

8 Referenser

8.1 Datablad

Processor ATmega88
www.atmel.com/dyn/resources/prod_documents/doc2545.pdf
Gyro MG1101
moffa.nu/stiff/elektronik/DE01300−001_Data_Sheet_MG1101_RevB.pdf
Accelerometer ADXL202
www.media.mit.edu/resenv/classes/MAS836/Inertialnotes/ADXL202_10_b.pdf
Analog avståndsmätare Sharp GP2D12
www.parallax.com/dl/docs/prod/acc/SharpGP2D12Snrs.pdf

8.2 Application notes

Motorstyrning AVR444: Sensorless control of 3-phase brushless DC motors
www.atmel.com/dyn/resources/prod_documents/doc8012.pdf
SPI-bussen AVR151: Setup and use of the SPI
www.atmel.com/dyn/resources/prod_documents/doc2585.pdf
Accelerometer AN-604: Using the ADXL202 Duty Cycle Output Application Note (REV. 0)
www.analog.com/UploadedFiles/Application_Notes/320058905AN604.pdf

8.3 Artiklar

Stabilization of a mini rotorcraft with four rotors
Av: Pedro Castillo, Rogelio Lazano, Alejandro Dzul
IEEE Control System Magazine
Appendix A

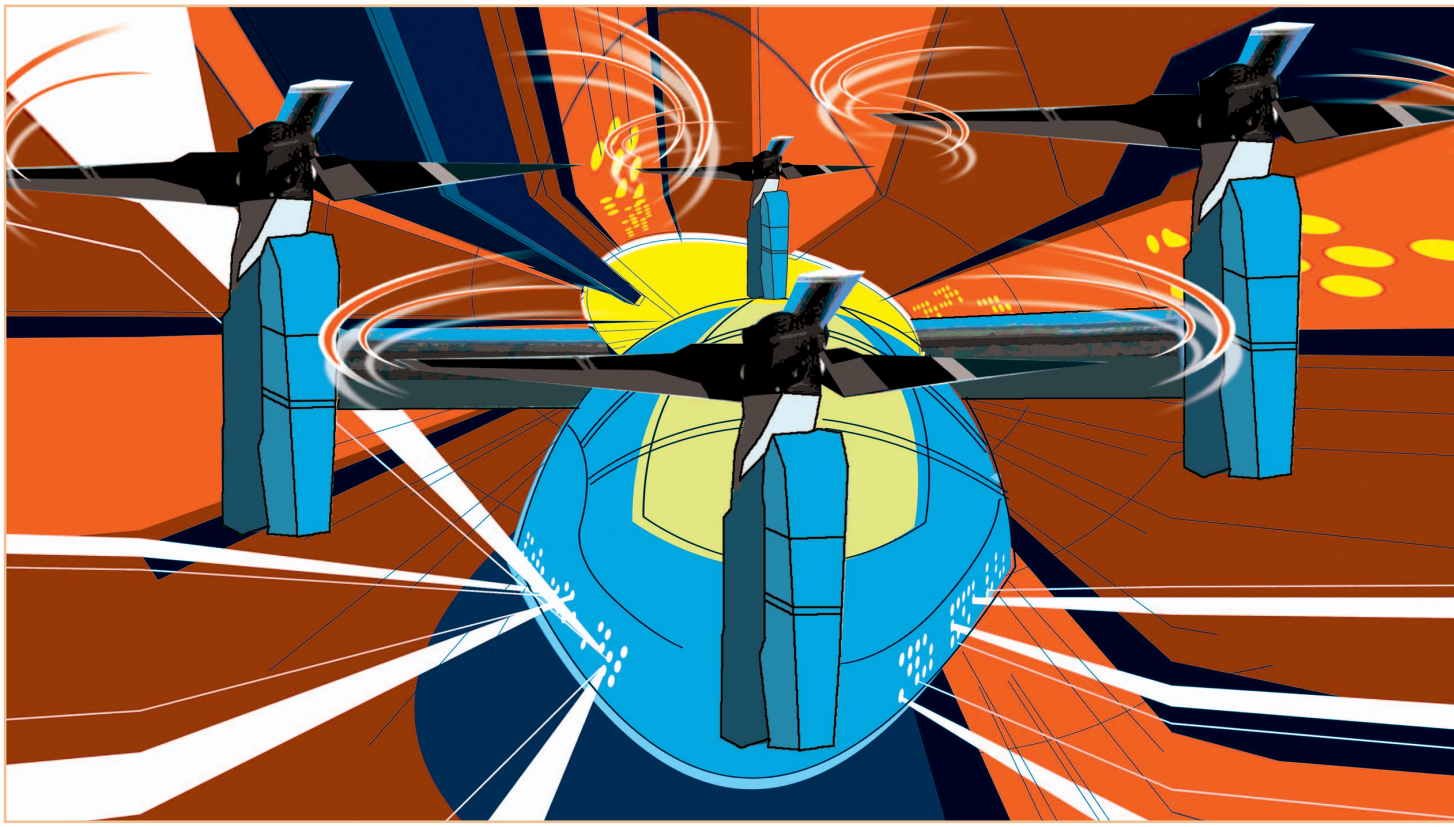
9 Appendix

A Stabilization of a mini rotorcraft with four rotors

Se nästa sida

FEATURE

Experimental implementation
of linear and nonlinear control laws



© DIGITAL VISION

Stabilization of a Mini Rotorcraft with Four Rotors

By Pedro Castillo,
Rogelio Lozano,
and Alejandro Dzul

Flight control of unmanned helicopters is an area that poses interesting problems for control researchers. The classical control strategy for helicopters assumes a linear model obtained for a particular operating point. Applying modern nonlinear control theory can improve the performance of the controller and enable the tracking of aggressive trajectories, as demonstrated in [1] for a 5-ft diameter main rotor helicopter.

Civil and military applications of autonomous flying vehicles have been steadily increasing over the last few years. Traffic surveillance, air pollution monitoring, area mapping, agricultural applications, and remote inspection require high maneuverability and robustness with respect to disturbances. Since rotary wing vehicles can take off and land in limited spaces and hover above targets, these vehicles have certain advantages over conventional fixed-wing aircraft for surveillance and inspection tasks.

1066-033X/05/$20.00©2005IEEE

The quad rotor is an excellent flying vehicle for investigating issues in automatic control, advanced sensor technology, and computer science. Autonomous flight poses research challenges such as intelligent control of aerial robots, three-dimensional (3-D) trajectory planning, multivehicle air traffic management, and collision avoidance. Since the quad rotor is dynamically unstable, control algorithms are required for stabilization.

Applying modern nonlinear control theory can improve the performance of the controller and enable tracking of aggressive trajectories.

The classical helicopter consists of a main rotor and a tail rotor. Other types of helicopters include the twin rotor, or tandem helicopter, and the coaxial rotors, helicopters having two counter-rotating rotors on the same shaft. We are particularly interested in controlling the quad rotor, a helicopter having four rotors, as shown in Figure 1. The electric quad-rotor helicopter is mechanically simpler and easier to repair than a classical electric helicopter. Since the mini rotorcraft is easy to operate, this device serves as a convenient laboratory testbed for studying modern nonlinear control techniques. Commercially available quad-rotor helicopters, such as those manufactured by Draganfly and Intellicopter, have four electric motors. Quad-rotor helicopters using four combustion engines do not present any advantages compared to classical helicopters. At the current time, none are commercially available.

The quad-rotor control problem is similar to that of controlling a planar vertical takeoff and landing (PVTOL) aircraft, which evolves in a vertical plane [2]–[4]. The aircraft has three degrees of freedom, (x, y, θ), corresponding to its position in the plane and pitch angle. The PVTOL has two independent thrusters that produce a force and an angular momentum; it thus has three degrees of freedom and only two inputs. Hence, the PVTOL is underactuated.

Characteristics of the Quad Rotor

Conventional helicopters modify the lift force by varying the collective pitch of the rotor blades. These helicopters use a mechanical device known as a swashplate to change the pitch angle of the rotor blades in a cyclic manner so as to obtain the pitch and roll control torques of the vehicle. In contrast, the quad rotor does not have a swashplate, and it has constant pitch blades. A quad rotor is controlled by varying the angular speed of each rotor. The force f_i produced by motor i is proportional to the square of the angular speed, that is $f_i = k\omega_i^2$. Since each motor turns in a fixed direction, the produced force f_i is always positive (see Figure 1). The front and rear motors rotate counterclockwise, while the other two motors rotate clockwise. With this arrangement, gyroscopic effects and aerodynamic torques tend to cancel in trimmed flight. The main thrust is the sum of the individual thrusts of each motor (see Figure 1). The pitch torque is a function of the difference $f_1 - f_3$, the roll torque is a function of $f_2 - f_4$, and the yaw torque is the sum $\tau_{M_1} + \tau_{M_2} + \tau_{M_3} + \tau_{M_4}$, where τ_{M_i} is the reaction torque of motor i due to shaft acceleration and the blade's drag. Using Newton's second law and neglecting shaft friction, we have

$$I_M \dot{\omega}_i = -b\,\omega_i^2 + \tau_{M_i},$$

where I_M is the angular momentum of the ith motor and $b > 0$ is a constant.

A quad rotor moves forward by pitching. This motion is obtained by increasing the speed of the rear motor M_3 while reducing the speed of the front motor M_1. Likewise, roll motion

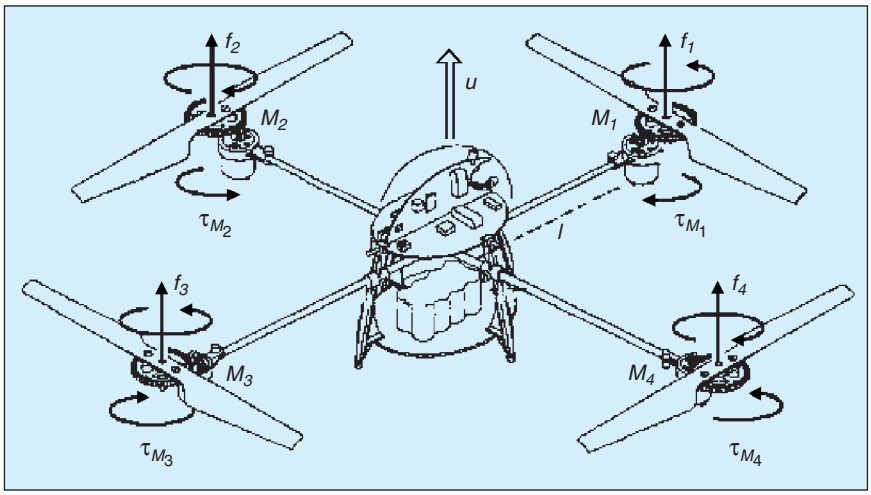


Figure 1. *The quad-rotor control inputs. The mini rotorcraft is controlled by varying the speeds of four electric motors. Each motor produces a thrust f_i, and these thrusts combine to generate the main thrust $u = \sum_{i=1}^{4} f_i$. The difference between the front rotor blade speed and the rear rotor blade speed produces a pitch torque. The roll torque is produced similarly. The yaw torque is the sum of the torques of each motor [see (1) and (2)].*

is obtained using the lateral motors. Yaw motion is obtained by increasing the torque of the front and rear motors, τ_{M_1} and τ_{M_3}, respectively, while decreasing the torque of the lateral motors, τ_{M_2} and τ_{M_4}. These motions can be accomplished while keeping the total thrust $u = f_1 + f_2 + f_3 + f_4$ constant. In steady state, that is, when $\dot{\omega}_i = 0$, the yaw torque is

$$\tau_\psi = b\left(\omega_1^2 + \omega_3^2 - \omega_2^2 - \omega_4^2\right), \tag{1}$$

and the main thrust is

$$u = k\sum_{i=1}^{4} \omega_i^2. \tag{2}$$

In view of its configuration, the quad rotor shares some similarities with the PVTOL aircraft. Indeed, if the roll or pitch and yaw angles are set to zero, the quad rotor reduces to a PVTOL and can be viewed as two PVTOLs connected with orthogonal axes.

In this article, we first model the dynamics of the quad rotor. Then we propose a control strategy by viewing the aircraft as the interconnection of two PVTOLs. The control algorithm is based on the nested saturation control strategy introduced in [5]. Using computer simulations, we compare the performance of the nonlinear control algorithm with an linear quadratic regulator (LQR) control law. The controller is implemented on a PC, and we present the results of laboratory experiments. Finally, we present some conclusions.

Dynamical Model

In this section, we derive a dynamical model of the quad rotor. This model is obtained by representing the aircraft as a rigid body evolving in a 3-D space due to the main thrust and three torques. The main thrust u is shown in Figure 1. The dynamics of the four electric motors are fast and, thus, are neglected.

The generalized coordinates of the rotorcraft are

$$q = (x, y, z, \psi, \theta, \phi) \in \mathbb{R}^6,$$

where $\xi = (x, y, z) \in \mathbb{R}^3$ denotes the position of the center of mass of the helicopter relative to a fixed inertial frame $\mathcal{I}$, and $\eta = (\psi, \theta, \phi) \in \mathbb{R}^3$ are the Euler angles, ψ is the yaw angle around the z-axis, θ is the pitch angle around the modified y-axis, and ϕ is the roll angle around the modified x-axis [6]–[8], which represent the orientation of the rotorcraft.

Define the Lagrangian

$$L(q, \dot{q}) = T_{\text{trans}} + T_{\text{rot}} - U,$$

where $T_{\text{trans}} = (m/2)\dot{\xi}^T\dot{\xi}$ is the translational kinetic energy, $T_{\text{rot}} = (1/2)\omega^T\mathbf{I}\,\omega$ is the rotational kinetic energy, $U = mgz$ is the potential energy, z is the rotorcraft altitude, m is the mass of the quad rotor, ω is the angular velocity, $\mathbf{I}$ is the inertia matrix, and g is the acceleration due to gravity. The angular velocity vector ω resolved in the body fixed frame is related to the generalized velocities $\dot{\eta}$ (in the region where the Euler angles are valid) by means of the kinematic relationship [9]

$$\dot{\eta} = W_v^{-1}\omega,$$

where

$$W_v = \begin{bmatrix} -\sin\theta & 0 & 1 \\ \cos\theta\sin\psi & \cos\psi & 0 \\ \cos\theta\cos\psi & -\sin\psi & 0 \end{bmatrix}.$$

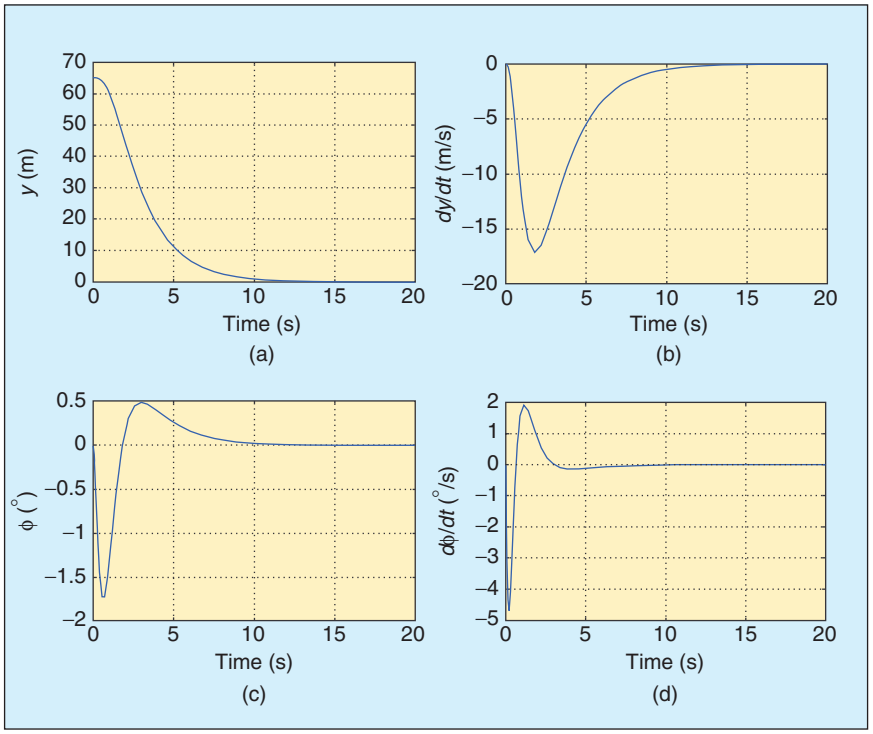


Figure 2. *Simulation of the LQR control law applied to the linear system (22). The initial conditions are altitude $y(0) = 70$ m and roll $\phi(0) = 0°$. In this case, y and ϕ converge to zero as expected.*

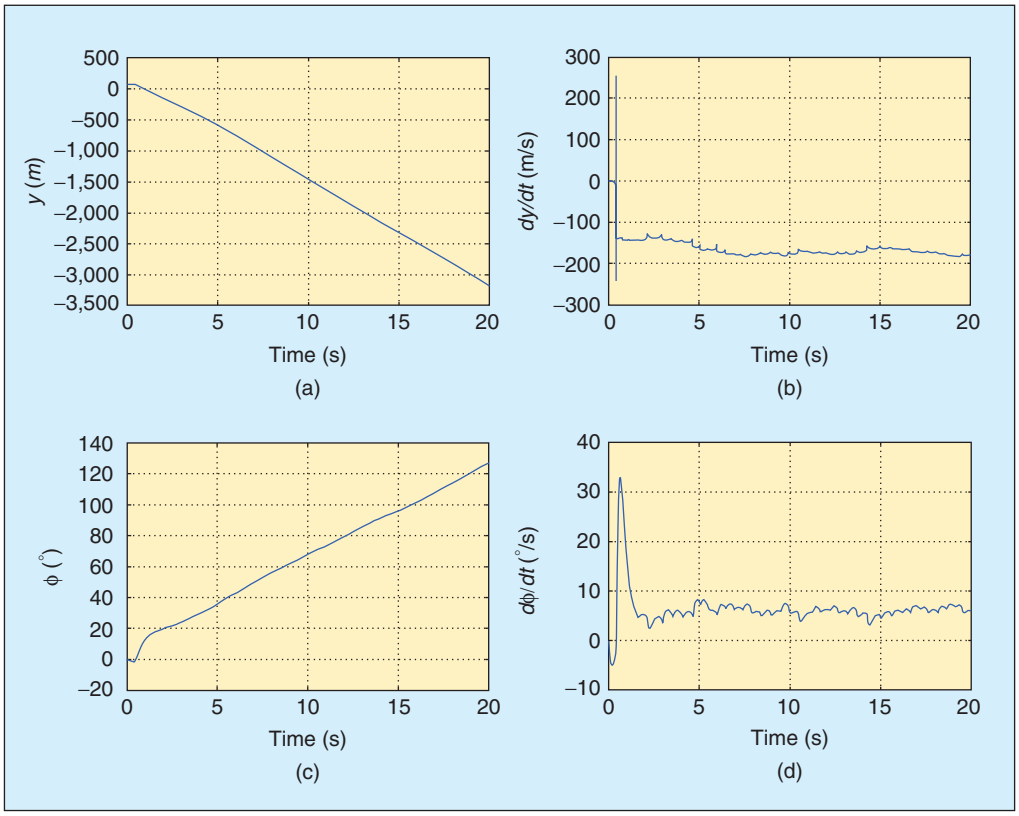


Figure 3. *Simulation of the LQR control law applied to the nonlinear subsystem (10) and (15). The initial conditions are $y(0) = 70$ m and $\phi(0) = 0^\circ$. Both y and ϕ diverge when the LQR controller is applied to the nonlinear subsystem (10) and (15). Therefore, the stability of the closed-loop system is not global, and the initial condition considered is outside of the domain of attraction.*

Define

$$\mathbb{J} = \mathbb{J}(\eta) = W_v^T \mathbf{I}\, W_v,$$

so that

$$T_{\text{rot}} = \frac{1}{2}\dot{\eta}^T \mathbb{J}\dot{\eta}.$$

Thus, the matrix $\mathbb{J} = \mathbb{J}(\eta)$ acts as the inertia matrix for the full rotational kinetic energy of the helicopter expressed in terms of the generalized coordinates η.

The model of the full rotorcraft dynamics is obtained from Euler-Lagrange equations with external generalized forces

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}} - \frac{\partial \mathcal{L}}{\partial q} = \begin{bmatrix} F_\xi \\ \tau \end{bmatrix},$$

where $F_\xi = R\widehat{F} \in \mathbb{R}^3$ is the translational force applied to the rotorcraft due to main thrust, $\tau \in \mathbb{R}^3$ represents the yaw, pitch, and roll moments, and R denotes the rotational matrix $R(\psi, \theta, \phi) \in SO(3)$ representing the orientation of the aircraft relative to a fixed inertial frame. From Figure 1, it follows that

$$\widehat{F} = \begin{bmatrix} 0 \\ 0 \\ u \end{bmatrix},$$

where u is the main thrust directed out the top of the aircraft expressed as

$$u = \sum_{i=1}^{4} f_i,$$

and, for $i = 1, \ldots, 4$, f_i is the force produced by motor M_i, as shown in Figure 1. Typically, $f_i = k\omega_i^2$, where k is a constant and ω_i is the angular speed of the ith motor. We assume that the center of gravity is located at the intersection of the line joining motors M_1 and M_3 and the line joining motors M_2 and M_4 (see Figure 1). The generalized torques are thus

Table 1. Parameter values used in the nonlinear control laws (16), (17), (20), and (21). These parameters are manually tuned to improve the performance of the closed-loop system.

Control parameter	Value
a_{z_1}	0.001
a_{z_2}	0.002
a_{ψ_1}	2.374
a_{ψ_2}	0.08
b_{ϕ_1}	2
b_{ϕ_2}	1
b_{ϕ_3}	0.2
b_{ϕ_4}	0.1
b_{θ_1}	2
b_{θ_2}	1
b_{θ_3}	0.2
b_{θ_4}	0.1
T	17 ms
ms	

$$\tau = \begin{bmatrix} \tau_\psi \\ \tau_\theta \\ \tau_\phi \end{bmatrix} \triangleq \begin{bmatrix} \sum_{i=1}^{4} \tau_{M_i} \\ (f_2 - f_4)\ell \\ (f_3 - f_1)\ell \end{bmatrix},$$

where ℓ is the distance between the motors and the center of gravity, and τ_{M_i} is the moment produced by motor M_i, $i = 1, \ldots, 4$, around the center of gravity of the aircraft.

Thus, we obtain

$$m\ddot{\xi} + \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} = F_\xi = R\widehat{F}, \tag{3}$$

$$\mathbb{J}\ddot{\eta} + C(\eta, \dot{\eta})\dot{\eta} = \tau, \tag{4}$$

where

$$C(\eta, \dot{\eta}) = \dot{\mathbb{J}} - \frac{1}{2}\frac{\partial}{\partial \eta}(\dot{\eta}^T \mathbb{J})$$

is the Coriolis term, which contains the gyroscopic and centrifugal terms associated with the η dependence of $\mathbb{J}$.

Finally, we obtain from (3) and (4)

$$m\ddot{x} = -u\sin\theta, \tag{5}$$

$$m\ddot{y} = u\cos\theta\sin\phi, \tag{6}$$

$$m\ddot{z} = u\cos\theta\cos\phi - mg, \tag{7}$$

$$\ddot{\psi} = \tilde{\tau}_\psi, \tag{8}$$

$$\ddot{\theta} = \tilde{\tau}_\theta, \tag{9}$$

$$\ddot{\phi} = \tilde{\tau}_\phi, \tag{10}$$

where x and y are coordinates in the horizontal plane, z is the vertical position, and $\tilde{\tau}_\psi$, $\tilde{\tau}_\theta$, and $\tilde{\tau}_\phi$ are the yawing moment, pitching moment, and rolling moment, respectively. These moments are related to the generalized torques τ_ψ, τ_θ, τ_ϕ by

$$\tilde{\tau} = \begin{bmatrix} \tilde{\tau}_\psi \\ \tilde{\tau}_\theta \\ \tilde{\tau}_\phi \end{bmatrix} = \mathrm{J}^{-1}(\tau - C(\eta, \dot{\eta})\dot{\eta}).$$

Control Strategy

In this section, we develop a control strategy for stabilizing the quad rotor at hover. The controller synthesis procedure regulates each state sequentially using a priority rule as follows. We first use the main thrust u to stabilize the altitude

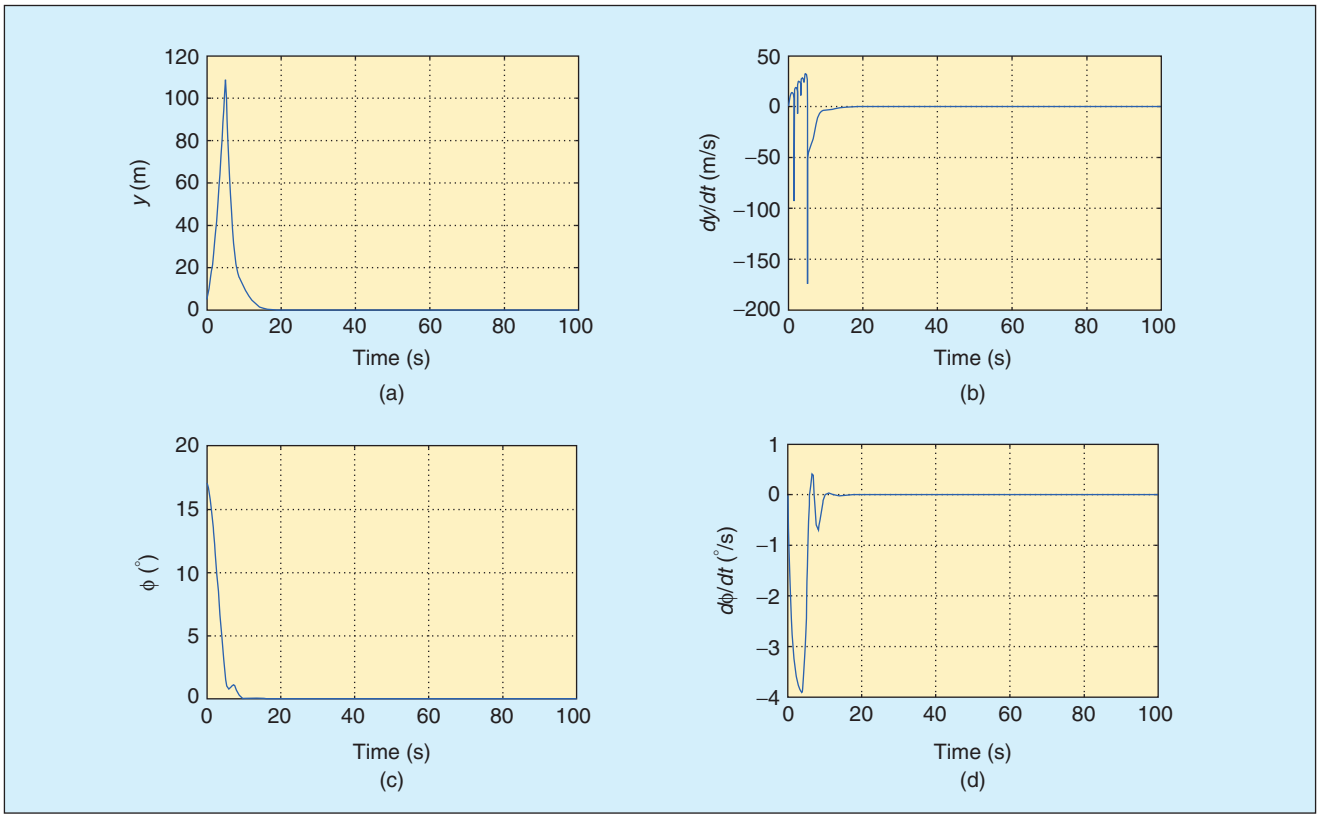


Figure 4. *Simulation of the nonlinear control law (20) with the nonlinear subsystem (10) and (15). The initial conditions are $y(0) = 5$ m, $\dot{y}(0) = 0$ m/s, $\dot{\phi}(0) = 0°/s$, and $\phi(0) = 17°$. The states y, $\dot{y}$, ϕ, $\dot{\phi}$ converge to zero despite the fact that the initial condition $\phi(0)$ is far from the origin.*

of the rotorcraft. Next, we stabilize the yaw angle. We then control the roll angle ϕ and the y displacement using a controller designed for the PVTOL model [2], [10]. Finally, the pitch angle θ and the x displacement are controlled.

The proposed control strategy is simple to implement and easy to tune. The experimental setup is such that the four control inputs can independently operate in either manual or automatic modes. For flight safety reasons, this feature is helpful for implementing the control strategy. The quad-rotor helicopter can be operated in semi-automatic mode, in which the remote pilot commands only the altitude and the desired position, leaving the orientation stabilization task to the control law.

Control of Altitude and Yaw

The vertical displacement z in (7) is controlled by forcing the altitude to satisfy the dynamics of a linear system. Thus, we set

$$u = (r_1 + mg)\frac{1}{\cos\theta\cos\phi}, \tag{11}$$

where r_1 is given by the proportional derivative (PD) controller

$$r_1 \triangleq -a_{z_1}\dot{z} - a_{z_2}(z - z_d), \tag{12}$$

where a_{z_1} and a_{z_2} are positive constants and z_d is a positive constant representing the desired altitude. To control yaw angle, we set

$$\tilde{\tau}_\psi = -a_{\psi_1}\dot{\psi} - a_{\psi_2}(\psi - \psi_d). \tag{13}$$

Introducing (11)–(13) into (5)–(8) and assuming $\cos\theta\cos\phi \neq 0$, that is, $\theta, \phi \in (-\pi/2, \pi/2)$, we obtain

$$m\ddot{x} = -(r_1 + mg)\frac{\tan\theta}{\cos\phi}, \tag{14}$$

$$m\ddot{y} = (r_1 + mg)\tan\phi, \tag{15}$$

$$\ddot{z} = \frac{1}{m}\left(-a_{z_1}\dot{z} - a_{z_2}(z - z_d)\right), \tag{16}$$

$$\ddot{\psi} = -a_{\psi_1}\dot{\psi} - a_{\psi_2}(\psi - \psi_d). \tag{17}$$

The control gains a_{ψ_1}, a_{ψ_2}, a_{z_1},and a_{z_2} are positive constants chosen to ensure stable, well-damped response of the quad rotor. From (16) and (17) it follows that, if ψ_d and z_d are constants, then ψ and z converge. Therefore, $\dot{\psi}$ and $\ddot{\psi} \to 0$, which, using (17), implies that $\psi \to \psi_d$. Similarly, $z \to z_d$.

Control of Lateral Position and Roll

We now determine the input $\tilde{\tau}_\phi$ such that y and ϕ, in (10) and (15), converge to zero. We assume $\psi_d \equiv 0$ in (13) and (17). Therefore, from (17) it follows that $\psi \to 0$. Note that (12) and (16) imply that $r_1 \to 0$.

Since the quad-rotor control inputs are subject to amplitude physical constraints,

$$\begin{aligned} 0 < u < 4\ V,\\ |\tilde{\tau}_\psi| \leq 2\ V,\\ |\tilde{\tau}_\theta| \leq 2\ V,\\ |\tilde{\tau}_\phi| \leq 2\ V, \end{aligned}$$

we use the control strategy developed in [5]. The nested saturation technique developed in [5] can exponentially stabilize a chain of integrators with bounded input. The amplitudes of the saturation functions can be chosen in such a way that, after a finite time T', the roll angle lies in the interval $-1\ \text{rad} \leq \phi \leq 1\ \text{rad}$. Therefore, for $t > T'$ $|\tan\phi - \phi| < 0.54$. Thus, after sufficient time, r_1 is small and the (y, ϕ) subsystem reduces to

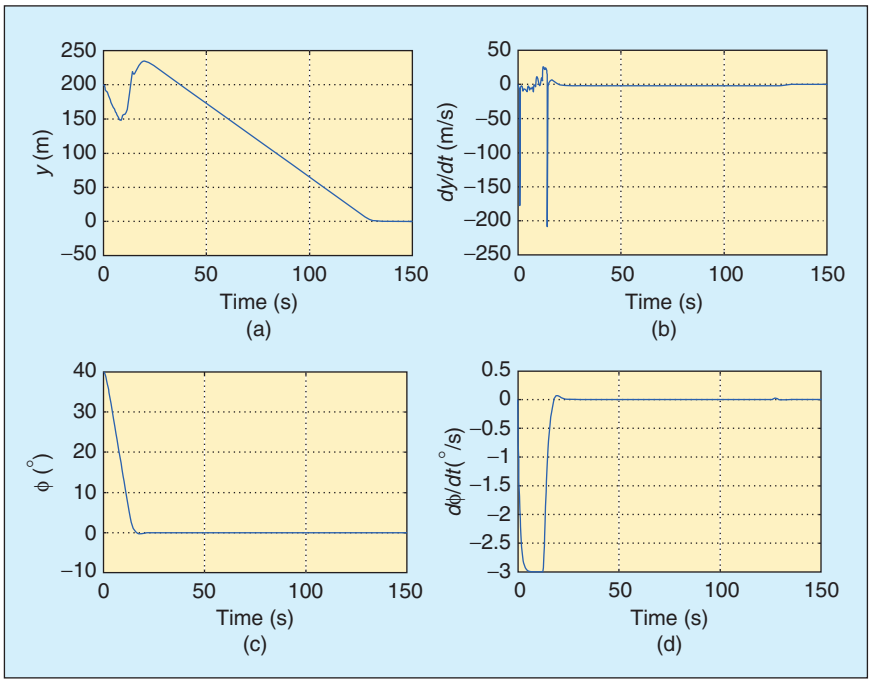


Figure 5. *Simulation of the nonlinear control law with the nonlinear subsystem (10) and (15). The initial conditions are $y(0) = 200$ m, $\dot{y}(0) = 0$ m/s, $\dot{\phi}(0) = 0°/s$, and $\phi(0) = 40°$. These values are chosen to show that the states $(y, \dot{y}, \phi, \dot{\phi})$ converge to zero when the initial roll angular displacement and y position are very far from the origin. Notice that the control strategy first brings the roll angle ϕ close to zero and then carries the y position to the origin.*

$$\ddot{y} = g\phi, \tag{18}$$
$$\ddot{\phi} = \tilde{\tau}_\phi, \tag{19}$$

which represents four integrators in cascade.

For (18)–(19), the nested saturation controller has the form

$$\tilde{\tau}_\phi = -\sigma_{\phi_1}\left(\dot{\phi} + \sigma_{\phi_2}\left(\phi + \dot{\phi} + \sigma_{\phi_3}\left(2\phi + \dot{\phi} + \frac{\dot{y}}{g} + \sigma_{\phi_4}\left(\dot{\phi} + 3\phi + 3\frac{\dot{y}}{g} + \frac{y}{g}\right)\right)\right)\right), \tag{20}$$

where σ_a is a saturation function of the form

$$\sigma_a(s) = \begin{cases} -a & s < -a, \\ s & -a \le s \le a, \\ a & s > a. \end{cases}$$

The closed loop is asymptotically stable (see [5]), and therefore ϕ, $\dot{\phi}$, y, and $\dot{y}$ converge to zero.

Control of Forward Position and Pitch

For small ϕ and r_1, (14) reduces to $\ddot{x} = -g\tan\theta$. The (x, θ) subsystem is

$$\ddot{x} = -g\tan\theta,$$
$$\ddot{\theta} = \tilde{\tau}_\theta.$$

Using a procedure similar to the one proposed for the roll control, we obtain

$$\tilde{\tau}_\theta = -\sigma_{\theta_1}\left(\dot{\theta} + \sigma_{\theta_2}\left(\theta + \dot{\theta} + \sigma_{\theta_3}\left(2\theta + \dot{\theta} - \frac{\dot{x}}{g} + \sigma_{\theta_4}\left(\dot{\theta} + 3\theta - 3\frac{\dot{x}}{g} - \frac{x}{g}\right)\right)\right)\right), \tag{21}$$

and thus θ, $\dot{\theta}$, x, and $\dot{x}$ also converge to zero.

Simulation Results

In this section, we compare the nonlinear control law (20) to a linear LQR controller. We focus our attention on the (y, ϕ) subsystem (18) and (19).

Define $\bar{x} = [y\ \dot{y}\ \phi\ \dot{\phi}]^T$. Then (18) and (19) can be rewritten as

$$\dot{\bar{x}} = A\bar{x} + B\bar{u}, \tag{22}$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & g & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \bar{u} = \tilde{\tau}_\phi.$$

A state feedback control input is given by

$$\bar{u} = -K\bar{x}, \tag{23}$$

where $K = R^{-1}B^TP$, and P is the unique, positive-semidefinite solution to the algebraic Riccati equation. Using the control input (23) into (22), we obtain

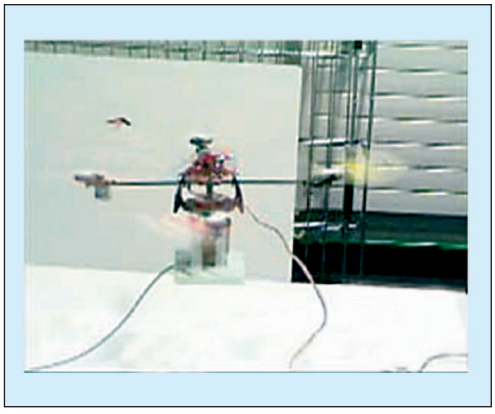


Figure 6. *Real-time quad-rotor control platform in autonomous hover. The control inputs are sent to the helicopter through a radio link. The wires attached to the rotorcraft provide connections to the power supply and the attitude sensor.*

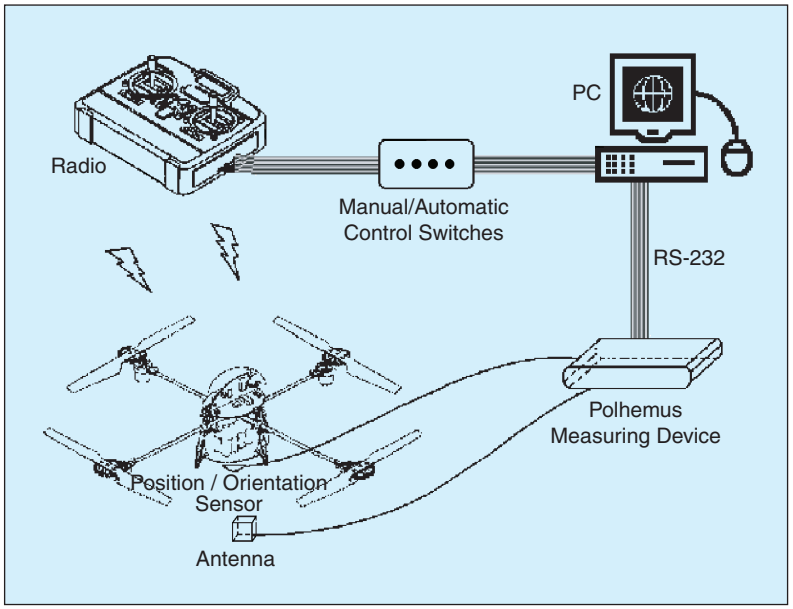


Figure 7. *Experimental setup. The position/orientation sensor on board the quad rotor is connected to a PC for feedback control. The PC generates control inputs, which are sent to the helicopter through the radio link.*

$$\dot{\bar{x}} = (A - BK)\bar{x}. \tag{24}$$

Choosing

$$Q = \begin{bmatrix} 1 & -2 & -4 & 6 \\ -2 & 4 & 8 & -12 \\ -4 & 8 & 16 & -24 \\ 6 & -12 & -24 & 36 \end{bmatrix}$$

and $R = 1$, the resulting gain that stabilizes system (24) is $K = [\,1 \quad 3.2848 \quad 29.3030 \quad 9.7266\,]$.

The closed-loop eigenvalues of $A - BK$ are -5.2393, -2.3946, -1.6056, and -0.4870, and thus the closed-loop system is asymptotically stable. Figure 2 shows the simulation results using the LQR control algorithm. The desired position is $\bar{x}_d = 0$. Note that for the initial conditions $y(0) = 70$ m, $\phi(0) = 0^\circ$, the state converges to zero.

Applying the LQR control law to the nonlinear system (10) and (15), and using the same conditions as before, the states y and ϕ diverge (see Figure 3). We observe in simulation that, for initial positions close to the desired position, the states converge to zero. However, if the rotorcraft is far from the desired position, the closed-loop system diverges (see Figure 3). This divergence is due to the fact that a large error in y produces a large angular displacement ϕ; therefore equation (18) is no longer an acceptable approximation for (15).

We now simulate the closed-loop system with the nonlinear control algorithm (20). We consider the initial conditions $y(0) = 5$ m and $\phi(0) = 17^\circ$. Figure 4 shows the simulation results for this subsystem (10) and (15) with the gains in Table 1. Note that $y \to 0$, $\dot{y} \to 0$, $\phi \to 0$, and $\dot{\phi} \to 0$. We observed that the speed of convergence increases as the amplitudes of the saturation functions increase. This trend is due to the fact that larger control inputs are allowed. Figure 5 shows similar results when the initial conditions are far from the desired position.

Simulation results show that, contrary to the LQR controller, the nonlinear controller in (20) stabilizes the equilibrium of subsystem (y, ϕ) around the origin for initial conditions far away from the desired position.

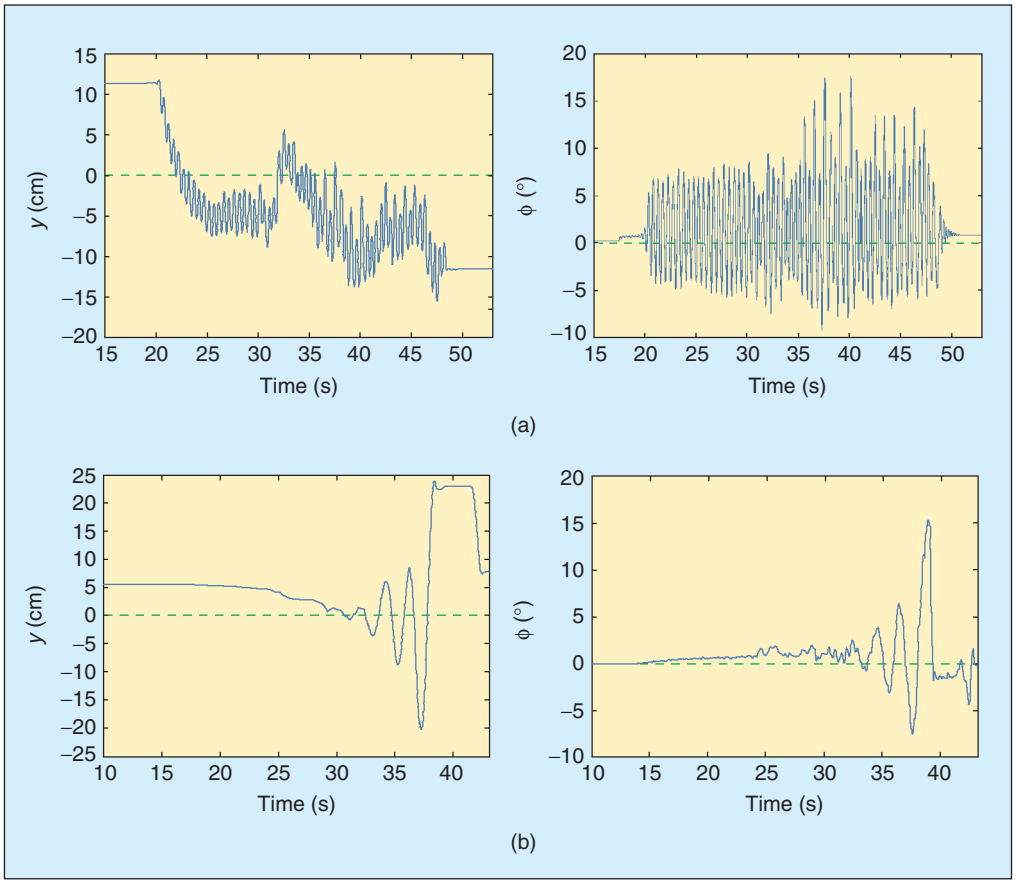


Figure 8. *Unstable response of the quad rotor for the LQR control law applied to the (ϕ, y) subsystem. (a) The dotted lines represent the desired trajectory for the initial conditions $y(0) = 12$ cm and $\phi(0) = 0^\circ$. The oscillations in the roll angle ϕ prevent the helicopter from taking off. In this experiment, we use the same controller parameters as in the simulation presented in Figure 2. (b) The LQR gains are manually adjusted to improve the performance of the mini rotorcraft, although the performance is inadequate for hovering.*

Experimental Results

Figure 6 shows the quad-rotor platform in autonomous hover. The experimental platform is composed of a Draganflyer helicopter, a Futaba 72-MHz radio, a Pentium II PC, and a 3-D tracker system (Polhemus) [11] for measuring the position (x, y, z) and orientation (ψ, θ, ϕ) of the quad rotor. The Polhemus is connected through an RS232 link to the PC (see Figure 7). The remote control system consists of a four-channel Futaba FM hobby radio.

An electronic circuit board in the helicopter contains three gyros, four pulsewidth modulation (PWM) speed controllers, a safety switch, and a microprocessor that mixes the pilot's commands to obtain the appropriate rotor control

inputs. The radio and the PC are connected using Advantech PCL-818HG and PCL-726 data-acquisition cards. To simplify the experiments, each control input is independently switched between automatic and manual control modes (see Figure 7).

The gyro stabilization introduces damping into the system and enables the quad rotor to be controlled manually. Without this gyro stabilization, it is almost impossible for a pilot to control the quad rotor manually [12], [13]. However, gyro stabilization, which represents only an angular speed feedback, is not sufficient for autonomous hover; for hover the quad rotor requires an attitude sensor, such as the Polhemus sensor, and a control law based on angular displacement feedback.

The control law requires the derivatives of the position (x, y, z) and the orientation (ψ, θ, ϕ). These derivatives are obtained numerically using the first-order approximation $\dot{q}(t) \approx (q(t) - q(t - T))/T$, where T is the sampling period. In all the experiments the position and orientation are provided by a Polhemus measuring device (see Figure 7).

LQR Control

Real-time experiments using the LQR control law are carried out with manual altitude control, that is, u is given by a pilot. To stabilize the system, we first implemented the LQR gains from the simulation results.

Figure 8(a) shows the lateral position and roll orientation of the quad rotor. As can be seen, the roll angle of the aircraft oscillates considerably, so that the helicopter cannot hover. To reduce the oscillations, we modify the gains to improve the performance. After numerous trials we significantly reduced the oscillations, as shown in Figure 8(b). Nevertheless, the obtained performance is not adequate to perform autonomous hovering.

Nonlinear Control Scheme

To apply the nonlinear control algorithm (20) to the rotorcraft, we place the aircraft in an arbitrary position, which is $(x, y, z) = (9, 12, 0)$ cm. The control objective is to make the rotorcraft hover at an altitude of 20 cm, that is, we

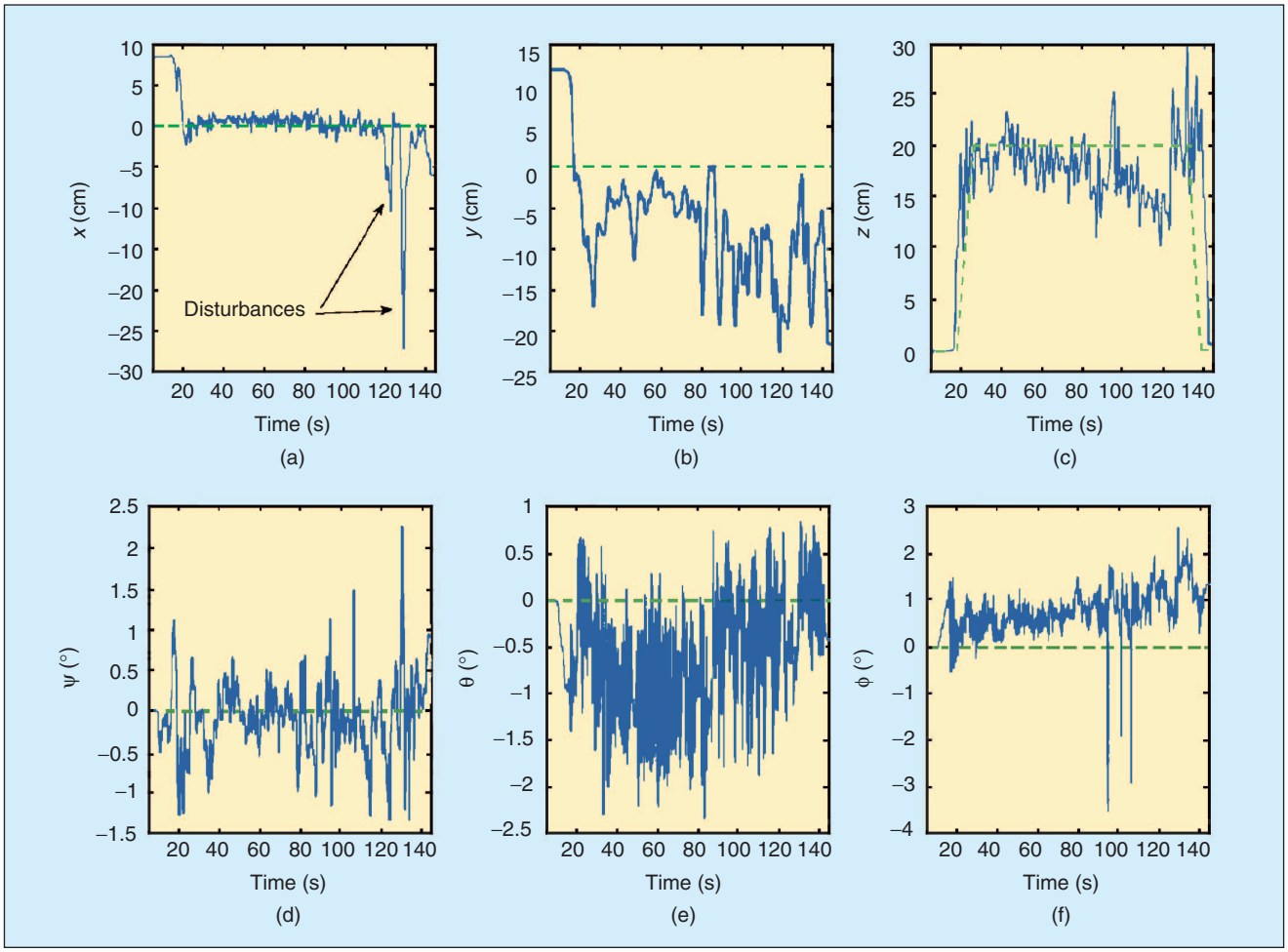


Figure 9. *Response of the quad rotor with position disturbances. The dotted lines represent the desired trajectory. The initial conditions are* $(x, y, z) = (8.7, 12, 0)$ *cm and* $(\psi, \theta, \phi) = (0^\circ, 0^\circ, 0^\circ)$. *The nonlinear controller recovers from position disturbances in x, y, and z introduced by pushing the quad rotor. The commanded hovering altitude is 20 cm.*

wish to reach the position $(x, y, z) = (0, 0, 20)$ cm while $(\psi, \theta, \phi) = (0^\circ, 0^\circ, 0^\circ)$, as shown in Figure 9. The controller parameters used in the experiment are given in Table 1. These parameters are tuned to obtain the best performance in practice.

The amplitudes of the saturation functions in the control law (20) are tuned as follows. We first tune the amplitude of σ_{ϕ_1} so that the roll angular velocity $\dot{\phi}$ remains close to zero even when a disturbance is introduced manually. We next select the amplitude of σ_{ϕ_2} in such a way that the quad-rotor roll angle is sufficiently small. In both cases, we avoid choosing high amplitude, which normally leads to oscillations. The amplitude of σ_{ϕ_3} is chosen so that the effect of a small disturbance in the horizontal speed $\dot{y}$ is soon compensated. Finally, the amplitude of σ_{ϕ_4} is chosen such that y is kept close to the desired position.

Figure 9 shows the performance obtained when we introduce a disturbance manually on the x-axis of -25 cm at time 125 s, five disturbances of -20 cm on the y–axis at times 25 s, 80 s, 90 s, 120 s, and 130 s, two disturbances of -10 cm on the z-axis at times 80 s and 115 s, and a disturbance of $+10$ cm on the z-axis at time 130 s.

We also study the system response to aggressive perturbations of the roll angle. In this experiment, we first apply a force manually to reach a roll angle of $+10^\circ$. At 95 s, we perturb the roll angle by -30°. As show in Figures 10 and 11, the response remains bounded.

Conclusions

We have presented a stabilization nonlinear control algorithm for a mini rotorcraft with four rotors. The dynamic model of the rotorcraft was obtained using a Lagrange approach. The proposed control algorithm is based on a

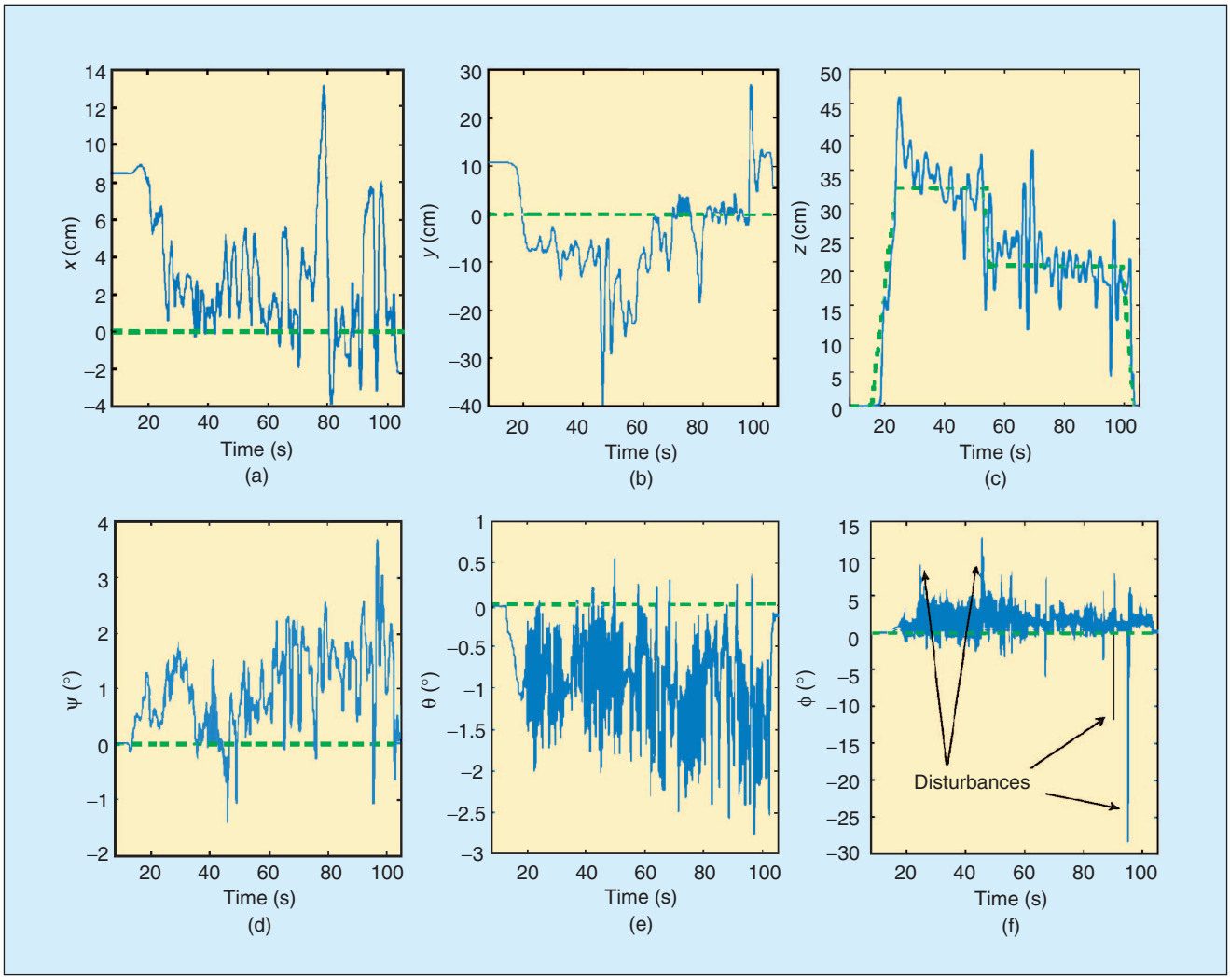


Figure 10. *Response of the quad rotor with roll disturbances. The initial conditions are* $(x, y, z) = (8, 12, 0)$ *cm and* $(\psi, \theta, \phi) = (0^\circ, 0^\circ, 0^\circ)$. *The dotted lines represent the desired setpoints. The roll angle* ϕ *is manually perturbed by* $+10^\circ$ *and* -30° *during the experiment. This experiment shows that the nonlinear control law can recover from large orientation perturbations.*

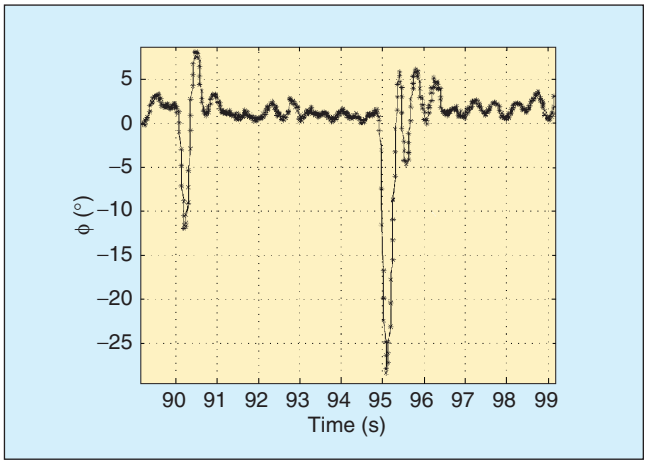


Figure 11. *System response to a roll angle perturbation. This closeup view corresponds to the roll angle in Figure 10. With the nonlinear control law, the subsystem (10) and (15) recovers from a roll perturbation of* -30°.

nested saturation control strategy, which is such that the amplitude constraints on the control input are satisfied.

The control strategy was applied to the mini rotorcraft, and the experimental results show that the controller performs satisfactorily even when significant disturbances are introduced into the system. Furthermore, experimental results show that the proposed nonlinear controller performs better than an LQR linear controller.

Acknowledgments

We are grateful for the financial support of the French Picardie Region Council, DGA, and LAFMAA (CNRS-CONACYT).

References

[1] V. Gavrilets, I. Martinos, B. Mettler, and E. Feron, "Control logic for automated aerobatic flight of miniature helicopter," in *Proc. AIAA,* Monterey, CA, Aug. 5–8, 2002, p. 4834.

[2] P. Castillo, R. Lozano, and A. Dzul, *Modelling and Control of Mini-Flying Machines* (Springer-Verlag Series in Advances in Industrial Control). New York: Springer-Verlag, 2005.

[3] I. Fantoni and R. Lozano, *Control of Nonlinear Mechanical Underactuated Systems (*Communications and Control Engineering Series). New York: Springer-Verlag, 2001.

[4] L. Marconi, A. Isidori, and A. Serrani, "Autonomous vertical landing on an oscillating platform: An internal-model based approach," *Automatica*, vol. 38, no. 1, pp. 21–32, 2002.

[5] A.R. Teel, "Global stabilization and restricted tracking for multiple integrators with bounded controls," *Syst. Contr. Lett.*, vol. 18, no. 3, pp. 165–171, 1992.

[6] T.S. Alderete, "Simulator aero model implementation" [Online], NASA Ames Research Center, Moffett Field, CA. Available: http://www.simlabs.arc.nasa.gov/library_docs/rt_sim_docs/Toms.pdf

[7] B. Etkin and L. Duff Reid, *Dynamics of Flight.* New York: Wiley, 1959.

[8] B.W. McCormick, *Aerodynamics Aeronautics and Flight Mechanics.* New York: Wiley, 1995.

[9] H. Goldstein, *Classical Mechanics,* 2nd ed. *(*Addison Wesley Series in Physics). Reading, MA: Addison-Wesley, 1980.

[10] J. Hauser, S. Sastry, and G. Meyer, "Nonlinear control design for slightly nonminimum phase systems: Application to V/STOL aircraft," *Automatica*, vol. 28, no. 4, pp. 665–679, 1992.

[11] *Fastrack 3Space Polhemus User's Manual*, Polhemus, Colchester, VT, 2001.

[12] P. Wayner, "Gyroscopes that don't spin make it easy to hover," *NY Times*, Aug. 8, 2002 [Online]. Available: http://www.nytimes.com

[13] N. Sacco, "How the Draganflyer flies," *Rotory Mag.*, 2002 [Online]. Available: http://www.rctoys.com/pdf/draganflyer3_rotorymagazine. pdf

Pedro Castillo (castillo@hds.utc.fr) obtained the B.S. degree in electromechanical engineering in 1997 from the Instituto Tecnológico de Zacatepec, Morelos, Mexico. He received an M.Sc. degree in electrical engineering from the Centro de Investigación y de Estudios Avanzados (CINVESTAV), Mexico, in 2000, and a Ph.D. in automatic control from the University of Technology of Compiègne, France, in 2004. His research interests include real-time control applications, nonlinear dynamics and control, aerospace vehicles, vision, and underactuated mechanical systems. He can be contacted at Heudiasyc-UTC, UMR CNRS 6599, B.P. 20529, 60205 Compiègne, France.

Rogelio Lozano received the B.S. degree in electronic engineering from the National Polytechnic Institute of Mexico in 1975, the M.S. degree in electrical engineering from the Centro de Investigación y de Estudios Avanzados (CINVESTAV), Mexico, in 1977, and the Ph.D. in automatic control from Laboratoire d'Automatique de Grenoble, France, in 1981. He joined the Department of Electrical Engineering at the CINVESTAV, Mexico, in 1981, where he worked until 1989. He was head of the Section of Automatic Control from June 1985 to August 1987. He has held visiting positions at the University of Newcastle, Australia, NASA Langley Research Center, and Laboratoire d'Automatique de Grenoble, France. Since 1990, he has been CNRS research director at the University of Technology of Compiègne, France, and, since 1995, head of the CNRS Laboratory Heudiasyc (Heuristique et Diagnostic des Systèmes Complexes). His research interests are in adaptive control, passivity, nonlinear systems, underactuated mechanical systems, and autonomous helicopters.

Alejandro Dzul received his B.S. degree in electronic engineering in 1993 and his M.S. degree in electrical engineering in 1997, both from Instituto Tecnológico de La Laguna, Mexico. He received the Ph.D. in automatic control from Université de Technologie de Compiègne, France, in 2002. Since 2003, he has been a research professor in the Electrical and Electronic Engineering Department at Instituto Tecnológico de La Laguna. His current research interests are in nonlinear dynamics and control and real-time control with applications to aerospace vehicles.

B Programkod

Se nästa sida

accel.h

```
volatile uint16_t TX1cal,TX2cal,TY1cal,TY2cal, TX1, TX2, TY1,TY2;
volatile uint16_t Previous_Duty_CycleX, DutycalX, TactualX;
volatile uint16_t Previous_Duty_CycleY, DutycalY, TactualY;
volatile uint8_t accstatus;
volatile uint16_t accel, Ta,Tb,Tc,Td,Te, K;

uint16_t getXaccel();
int getYaccel();
void calibrate();
void delay();
void init_accel();
```

accel.c

```
#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include "accel.h"

/*Funktion för beräkning av acceleration i X led*/
uint16_t getXaccel() {
      accel=1;
      TactualX = (uint16_t)(((uint32_t)(TX1cal*TX2))/TX2cal);
      Previous_Duty_CycleX=((uint32_t)65536*(TX1-TactualX))/TX2;
      return accel; //Blev aldrig rätt på beräkning av acceleration,
      //därför används Previous_Duty_Cycle istället.
}

int getYaccel() {
      accel=2;
      TactualY = (uint16_t)(((uint32_t)(TY1cal*TY2))/TY2cal);
      Previous_Duty_CycleY=((uint32_t)65536*(TY1-TactualY))/TY2;
      return accel;
}

ISR(TIMER1_CAPT_vect){
      cli();
      if(accstatus==0){ //Tiden Ta,
            TCNT1  =0; // Nollställ räknaren
            TCCR1B ^= _BV(ICES1); //Ändra avbrottstriggning
            TIFR1 |= _BV(ICF1); //Måste rensa efter man ändrat trigg
            accstatus =  1;
      }else if(accstatus==1){
            TX1 =  ICR1; //Ta Tb värdet, vilket pga att Ta är noll
blir T1
            TCCR1B ^= _BV(ICES1);
            TIFR1 |= _BV(ICF1);
            accstatus =  2;
      }else if(accstatus==2){
            TX2 = ICR1; //Ta Tc värdet
            TIFR1 |= _BV(ICF1);
            PORTD |=  _BV(PD7); //styr mux så man får in y-axel
            accstatus  = 3;
      }else if(accstatus==3){ //Upprepa samma procidur
            TCNT1  =  0;
            TCCR1B ^= _BV(ICES1);
            TIFR1 |= _BV(ICF1);
            accstatus =  4;
      }else if(accstatus==4){
```

```
            TY1 = ICR1;
            TCCR1B ^= _BV(ICES1);
            TIFR1 |= _BV(ICF1);
            accstatus  =  5;
      }else if(accstatus==5){
            TY2 =   ICR1;
            TIFR1  |=  _BV(ICF1);
            PORTD &= ~_BV(PD7);
            accstatus  = 0;
      }else{
            accstatus =0;
            //Fel
      }
      sei();

}

/*Tar kalibreringsvärde när QuadCoptern står på marken (innan
motorerna sätts igång)*/
void calibrate() {
      for(int n=0; n<10;n++){
            delay(0xFF);
      }
      getXaccel();
      getYaccel();
      TY1cal=TY1;
      TY2cal=TY2;
      TX1cal=TX1;
      TX2cal=TX2;
      DutycalX=Previous_Duty_CycleX;
      DutycalY=Previous_Duty_CycleY;
      K = (uint16_t)((uint32_t)4*(TX2cal*180)/TX2cal);
}

/*Godtyckligt lång delayfunktion*/
void delay(int n) {
      for(int i = 0; i < n; i++) {
            for(int q = 0; q < 10; q++)
                  asm volatile("nop");
      }
}
/*Initiera accelerometer*/
void init_accel(){
      DDRB &= ~_BV(PB0); //X-accel ingång
      DDRD |= _BV(PD7); //Styrutgång
      TCCR1B |= _BV(CS10)| _BV(ICNC1); //Initiera timer, 1x
prescaler, Enable Noise Canceler, Capture on Rising-edge
      TIMSK1 |= _BV(ICIE1);// | _BV(OCIE1A); //Input Capture
Interrupt Enable on ICP1
}
```

servo.h

```
volatile uint16_t servo1,servo2,servo3,servo4; //Min= 0x4E (78) Max=
0x9C (156)

void init_Servopuls();
```

servo.c

```
#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include "servo.h"


uint8_t next;
/*Initiera Servopuls*/
void init_Servopuls(){
      DDRC |= _BV(PC0)|_BV(PC1)|_BV(PC2)|_BV(PC3); //Servoutgångar,
PC0..3
      TCCR0B |= _BV(CS02);// | _BV(CS00); //Timer0, 256x prescaler
      TIMSK0 |= _BV(OCIE0A); //Enable timer0 compare interrupt
      TIFR0 |= _BV(OCF0A); //Counter output compare A match
      next=1;
      PORTC &= ~_BV(PC0);
      servo1 =10; //Min= 0x4E (78) Max= 0x9C (156)
      servo2 =10;
      servo3 =10;
      servo4 =10;
      sei(); //Enable global interrupt
}

/*Servopuls interruptrutin för att byta utgångar*/
ISR(TIMER0_COMPA_vect){
      cli();
      switch(next){
            case 1: //Sätt PC0 hög tills nästa compare på värde
servo1
                  OCR0A = servo1;
                  PORTC |= _BV(PC0);
                  TCNT0=0;
                  break;
            case 2: //Sätt PC0 låg, PC1 hög tills nästa compare på
värde servo2
                  OCR0A = servo2;
                  PORTC &= ~_BV(PC0);
                  PORTC |= _BV(PC1);
                  TCNT0=0;
                  break;
            case 3: //Upprepa för PC2
                  OCR0A = servo3;
                  PORTC &= ~_BV(PC1);
                  PORTC |= _BV(PC2);
                  TCNT0=0;
                  break;
            case 4: //Upprepa för PC3
                  OCR0A = servo4;
                  PORTC &= ~_BV(PC2);
                  PORTC |= _BV(PC3);
                  TCNT0=0;
                  break;
```

```
            case 5: //Alla utgångar låga
                  OCR0A = 0xFF;
                  PORTC &= ~_BV(PC3);
                  TCNT0=0;
                  break;
            default:
                  OCR0A  = 0xFF;
                  if(next >= 10){ //Alla utgångar låga i 5x255 counts
sen starta om
                        next = 0;
                  }
                  TCNT0=0;
                  break;
      }
      next+=1;
      sei();
}
```

IR.h

```
volatile uint8_t ir;
void init_IR();
void getIR();
```

IR.c

```
#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include "IR.h"
#include "accel.h"

/*Initierar AD-omvandlaren på PD0 i Free Running Mode*/
void init_IR(){
      DDRD &= ~_BV(PD0);
      ADMUX |= _BV(REFS0);
      ADCSRA |= _BV(ADEN) | _BV(ADSC) | _BV(ADATE) | 0x03;
//Prescaler x128
}

/*Läser av AD-registret (10 bitar)*/
void getIR(){
      uint16_t temp=ADCL;
      temp|=ADCH<<8;
      ir=temp;
}
```

SPI.h

```
volatile uint16_t nextSPI; //Slave variables
volatile  uint16_t degam,degbm,irm,x,y,sentSPI; //Master variables

void init_SPI_slave();
void init_SPI_master();
void start_trans();
```

SPI.c

```
#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include "SPI.h"
#include "IR.h"
#include "accel.h"
#include "twi.h"

//Initiate Slave
void init_SPI_slave(){
      uint8_t  fakeread;
      DDRB  |= _BV(PB4); //MISO output
      SPCR |= _BV(SPE) | _BV(SPIE);//Enable SPI, Enable interrupt
      SPCR &= ~_BV(MSTR); //Slave
      fakeread=SPSR;
      fakeread=SPDR;
      nextSPI=0;
      SPDR=0x1;
}
//Initiate Master
void init_SPI_master(){
      uint8_t fakeread;
      DDRB |=  _BV(PB3) | _BV(PB5) | _BV(PB2);  //MOSI,SCK,SS  output
      PORTB |= _BV(PB2);  //no transmission,  must be  done  first
      SPCR |= _BV(SPE) |  _BV(MSTR);//Enable SPI, set  master
      SPCR |=  _BV(SPR0) |  _BV(SPIE);  //16x prescaler, Enable
interrupt
      fakeread=SPSR;
      fakeread=SPDR;
      sentSPI=0;
}

/*Start transfer, startar överföring av all mätdata*/
void start_trans(){
      if(sentSPI==0){
            PORTB &= ~_BV(PB2);
            sentSPI=1;
            SPDR=1;
      }
}

/*Avbrottshanterare för att ladda SPI-registret med ny information*/
ISR(SPI_STC_vect){
      cli();
      if(0x10&SPCR){ //Master handler
            uint8_t temp = SPDR; //Ladda in mottaget värde
            if(sentSPI==1){
                  SPDR=2;
                  sentSPI=2;
```

```
            }else if(sentSPI==2){
                  SPDR=3;
                  degam=temp<<8;
                  sentSPI=3;
            }else if(sentSPI==3){
                  SPDR=4;
                  degam|=temp;
                  sentSPI=4;
            }else if(sentSPI==4){
                  SPDR=5;
                  degbm=temp<<8;
                  sentSPI=5;
            }else if(sentSPI==5){
                  SPDR=6;
                  degbm|=temp;
                  sentSPI=6;
            }else if(sentSPI==6){
                  SPDR=7;
                  x=temp<<8;
                  sentSPI=7;
            }else if(sentSPI==7){
                  SPDR=8;
                  x|=temp;
                  sentSPI=8;
            }else if(sentSPI==8){
                  SPDR=9;
                  y=temp<<8;
                  sentSPI=9;
            }else if(sentSPI==9){
                  y|=temp;
                  sentSPI=0;
                  PORTB |= _BV(PB2);
            }else
                  temp=SPDR; //just do a fake read;
      }else{ //Slave handler
            nextSPI=SPDR; //Ladda in val av nästa värde (för
synkning)
            if(nextSPI==1){
                  SPDR = dega>>8;
            }else if(nextSPI==2){
                  SPDR = 0xff&dega;
            }else if(nextSPI==3){
                  SPDR = degb>>8;
            }else if(nextSPI==4){
                  SPDR = 0xff&degb;
            }else if(nextSPI==5){
                  SPDR = Previous_Duty_CycleX>>8;
            }else if(nextSPI==6){
                  SPDR = 0xff&Previous_Duty_CycleX;
            }else if(nextSPI==7){
                  SPDR = Previous_Duty_CycleY>>8;
            }else if(nextSPI==8){
                  SPDR = 0xff&Previous_Duty_CycleY;
            }else if(nextSPI==9){
                  SPDR=0x1;
            }
      }
      sei();
}
```

TWI.h

```
volatile uint16_t aoffset;
volatile uint16_t boffset;
volatile uint16_t rota;
volatile uint16_t rotb;
volatile uint8_t status, nextTWCR, mem, test;
volatile uint16_t oldb, olda, dega;
volatile int degb;


void GetRot();

void GetRotB();

void FakeWrite(uint8_t adr, uint8_t mem);

void Write8(uint8_t adr, uint8_t mem1, uint8_t data);

uint8_t Read8(uint8_t adr, uint8_t mem1);

void InitTWI(void);

void PowerUpGyro(void);

void StartUpGyro(void);

ISR(TWI_vect);
```

TWI.c

```
#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#include <avr/interrupt.h>
#include "twi.h"

//Beräknar båda rotationsriktningarna
void CalcRot(){
      dega=(((rota>>1)+(olda>>1))-aoffset);
      degb=(((rotb>>1)+(oldb>>1))-boffset);
}
//Beräkanar endast en rotaionsriktning
void CalcRotB(){
      degb=(((rotb>>1)+(oldb>>1))-boffset);
}

//Startar avläsning från gyrot resten avbrottshanterat. Växlar mellan
att hämta från A eller B
void GetRot(){
      if(status  ==  0){
            if(mem ==  0x00){
                  mem =  0x02;
            }else{
                  mem  =  0x00;
            }
            status  = 1;
```

```
            TWCR = (1<<TWEN) |  (1<<TWSTA) | (1<<TWINT) | (1<<TWIE);
      }
}

//Startar avläsning från gyrot, men endast från riktning B
void GetRotB(){
      if(status  ==  0){
            status  =  1;
            mem =  0x02;
            TWCR = (1<<TWEN) |  (1<<TWSTA) | (1<<TWINT) | (1<<TWIE);
      }
}
//Gör en låtsas skrivning till minnet för att få minnenspekaren på
rätt ställe
void FakeWrite(uint8_t adr, uint8_t mem){
      TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
      while(!(TWCR & (1<<TWINT)));
      TWDR = adr & 0xfe;
      TWCR = (1<<TWINT) | (1<<TWEN);
      while(!(TWCR & (1<<TWINT)));
      TWDR = mem;
      TWCR = (1<<TWINT) | (1<<TWEN);
      while(!(TWCR & (1<<TWINT)));
}
//Skriver en byte
void Write8(uint8_t adr, uint8_t mem1, uint8_t data){
      FakeWrite(adr, mem1);
      TWDR = data;
      TWCR = (1<<TWINT) | (1<<TWEN);
      while(!(TWCR & (1<<TWINT)));
      TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TWEN);
}

//Läser en byte
uint8_t Read8(uint8_t adr, uint8_t mem1){
      uint8_t temp;
      FakeWrite(adr, mem1);
      TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
      while(!(TWCR & (1<<TWINT)));
      TWDR = adr | 0x01;
      TWCR = (1<<TWINT) | (1<<TWEN);
      while(!(TWCR & (1<<TWINT)));
      TWCR = (1<<TWINT) | (1<<TWEN);
      while(!(TWCR & (1<<TWINT)));
      temp = TWDR;
      TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TW,N);
      return temp;
}
//Startar TWI bussen
void InitTWI(void){
      //Sätter TWI till 100 kHz vid AVR 20 MHz
      TWBR = 0x17;
      status = 0;
      TWSR = (1<<TWPS0);
}
//Uppstartssekvens för gyrot när strömmen ansluts
void PowerUpGyro(void){

      for(uint8_t i = 0x00; i<0x10; i++){
            Write8(0xaf, i, (Read8(0xa1, i)));
      }
```

```
	Write8(0xaf, 0x0f, 0x40);
}
//Uppstart och kallibrering av gyrot efter PowerUp
void StartUpGyro(void){

	aoffset = 0xffff;
	boffset = 0xffff;
	Write8(0xaf, 0x0f, 0x41);
	//Låter gyrot lugna ner sig
	_delay_ms(200);
	_delay_ms(200);
	while(!((Read8(0xaf, 0x0f)) & 0x02));//Pollar gyrot ifall det
är redo
	//Kontrollerar ifall den förra while lopen gjorde rätt
	while(aoffset == 0xFFFF | boffset  == 0xFFFF){
		aoffset = (Read8(0xaf, 0x00))<<8;
		aoffset |= Read8(0xaf, 0x01);
		boffset = (Read8(0xaf, 0x02))<<8;
		boffset |= Read8(0xaf,0x03);
	}
	//Ökar hastigheten på bussen
	TWBR = 0x13;
	TWSR &= ~_BV(TWPS0);
}

//Avbrottshanteraren som hanterar avbrottet från TWIn
ISR(TWI_vect){
	cli();
	switch(status){
		case 1: {//Skickar adressen med skrivinstruktionen till
gyrot
			TWDR = 0xae;
			status = 2;
			nextTWCR = 0;
		}break;

		case 2: {//Skickar var vi vill skriva, detta för att få
pekaren i minnet rätt när vi ska läsa
			TWDR = mem;
			nextTWCR = 0;
			status = 3;
		}break;

		case 3: {//Startar igen
			nextTWCR = (1<<TWSTA);
			status = 4;
		}break;

		case 4: {//Skickar adressen med läsinstruktionen till
gyrot
			TWDR = 0xaf;
			nextTWCR = 0;
			status = 5;
		}break;

		case 5: {//Bekräftar acken och väntar på data
			status = 6;
			nextTWCR = (1<<TWEA);
		}break;

		case 6: {//Läser första byten
```

```
                        olda  = rota;
                        oldb = rotb;
                        if(mem == 0x00){
                              rota = TWDR<<8;
                        }else{
                              rotb = TWDR<<8;
                        }
                        nextTWCR = 0;
                        status = 7;
                  }break;

                  case 7: {//Läser andra byten

                        if(mem == 0x00){
                              rota |= TWDR;
                        }else{
                              rotb |=TWDR;
                        }
                        nextTWCR = (1<<TWSTO);
                        status = 0;
                  }break;

            }
            //Skickar nästa instruktion
            TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWIE) | nextTWCR;
            sei();
}
```

main.c (slave)

```
/****************/
/*Slave Code    */
/*   for        */
/*QuadCopter    */
/****************/
#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include "../lib/uart.h"
#include "../lib/accel.h"
#include "../lib/IR.h"
#include "../lib/servo.h"
#include "../lib/SPI.h"
#include "../lib/twi.h"

/* define CPU frequency in Mhz here if not defined in Makefile */
#ifndef F_CPU
#define F_CPU 20000000UL
#endif

int main (void) {
      //Initiering
      //Uppstart för allt som används av slaven
      init_accel();
      init_SPI_slave();
      init_IR();
      InitTWI();
      //Uppstart av gyro
      PowerUpGyro();
      delay(0xFFFF);
      delay(0xFFFF);
      StartUpGyro();
      sei();
      calibrate();
      cli();
      //Kalibrering av gyro
      for(int i = 0; i<100; i++){
            delay(0xffff);
            StartUpGyro();
      }
      sei();

      calibrate();
      uart_puts("Digiproj & Reglerproj");
      int i = 0;
      int h = 0;
      //Huvudprogrammet
      while(1){
            GetRot();
            CalcRot();
            i = getYaccel();
            h = getXaccel();
            getIR();
      }
      return 0;
}
```

main.c (master)

```
/****************/
/*Master Code   */
/*   for        */
/*QuadCopter    */
/****************/

#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include "../lib/uart.h"
#include "../lib/SPI.h"
#include "../lib/servo.h"
#include "../lib/twi.h"

volatile uint16_t xcal, xold;
volatile int calc;

/* define CPU frequency in Mhz here if not defined in Makefile */
#ifndef F_CPU
#define F_CPU 20000000UL
#endif

/* 57600 baud */
#define UART_BAUD_RATE      57600
/*
void uart_hex(unsigned char data);
void uart_hexint(int data);
void uart_newline();
*/
/*
SPI tar emot på:
Accelermeter variabler:
x,y

IR variabler:
irm

Gyrovariabler:
rotam
rotbm
rotb

*/

//volatile uint16_t  xcal;

/*Godtyckligt lång delayfunktion*/
void delay(int n) {
      for(int i = 0; i < n; i++) {
            for(int q = 0; q < 10; q++)
                  asm volatile("nop");
      }
}

int main(void){
      uart_init(UART_BAUD_SELECT(UART_BAUD_RATE,F_CPU));
      //Aktiverar funktioner
      init_SPI_master();
      init_Servopuls();
```

```
InitTWI();
PowerUpGyro();
//Delay för att gyrot ska lugna sig
delay(0xFFFF);
delay(0xFFFF);
StartUpGyro();
//Mer delay för gyrot
for(int i = 0; i<100; i++){
      delay(0xffff);
      StartUpGyro();
}
//variabler för att få motorstyrningen att starta
int first=1;
int second=1;
sei();
uart_putc(12);
uart_puts("SPI-TEST");
uart_newline();
//Kontrollerar så accelerometern ligger inom acceptabla värden
while(x == 0xffff | x <= 0x5000){
      start_trans();
      uart_hexint(x);
      uart_newline();
}
delay(0xffff);
xcal = x;
//Programlopen
while(1){
      //Beräknar servopulsen till motorerna
      int gyro = degb>>8;
      calc = (int)(xcal-x)>>6;
      int u = calc-gyro;
      //Sätter pulsen till 2 av motorerna
      servo3 = 90-u;
      servo1 = 90+u;
      GetRotB();
      CalcRotB();
      start_trans();
      //Skriver ut värdena till en PC
      uart_puts("calc: ");
      uart_hexint(calc);
      uart_puts("  u: ");
      uart_hexint(u);
      uart_puts("  Gyro: ");
      uart_hexint(gyro);
      uart_puts("  Servo1: ");
      uart_hexint(servo1);
      uart_puts("  Servo3: ");
      uart_hexint(servo3);
      uart_newline();

      //Startup sekvens för motorerna
      if(first){
            for(int i=30;  i<150;  i++){
                  servo1=i;
                  servo3=i;
                  for(int n=0;  n<9000; n++){
                        delay(0xffff);
                  }
            }
            first=0;
```

```
            }
            if(second){
                  for(int i=150;  i>30;  i--){
                        servo1=i;
                        servo3=i;
                        for(int n=0;  n<9000; n++){
                              delay(0xffff);
                        }
                  }
                  first=1;
                  second=0;
            }


      }
      return  0;
}



/*Uart funktioner*/
void uart_hex(unsigned char data) {
      unsigned char aHex[] =
{'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
      uart_putc(aHex[((data>>4)&0x0F)]);
      uart_putc(aHex[(data&0x0F)]);

}
void uart_hexint(int data) {
      unsigned char aHex[] =
{'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
      uart_putc(aHex[((data>>12)&0x0F)]);
      uart_putc(aHex[((data>>8)&0x0F)]);
      uart_putc(aHex[((data>>4)&0x0F)]);
      uart_putc(aHex[(data&0x0F)]);
}
void uart_newline(){
      uart_putc(10);
      uart_putc(13);
}
```

C Flödesschema

Se nästa sida