(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 July 2006 (20.07.2006)

PCT

(10) International Publication Number
WO 2006/076647 A2

(51) International Patent Classification:
G01C 23/00 (2006.01)

(21) International Application Number:
PCT/US2006/001347

(22) International Filing Date: 13 January 2006 (13.01.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/644,105 14 January 2005 (14.01.2005) US

(71) Applicant *(for all designated States except US)*: **SYSTEMS TECHNOLOGY, INC.** [US/US]; 13766 S. Hawthorne Boulevard, Hawthorne, CA 90250 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **BACHELDER, Edward N.** [US/US]; 615 S. Catalina, Redondo Beach, CA 90277 (US). **LEE, Dong-chan** [US/US]; 14528 Avis Avenue, Lawndale, CA 90260 (US). **APONSO, Bimal** [US/US]; 32012 Hawksmoor Drive, Rancho Palos Verdes, CA 90274 (US).

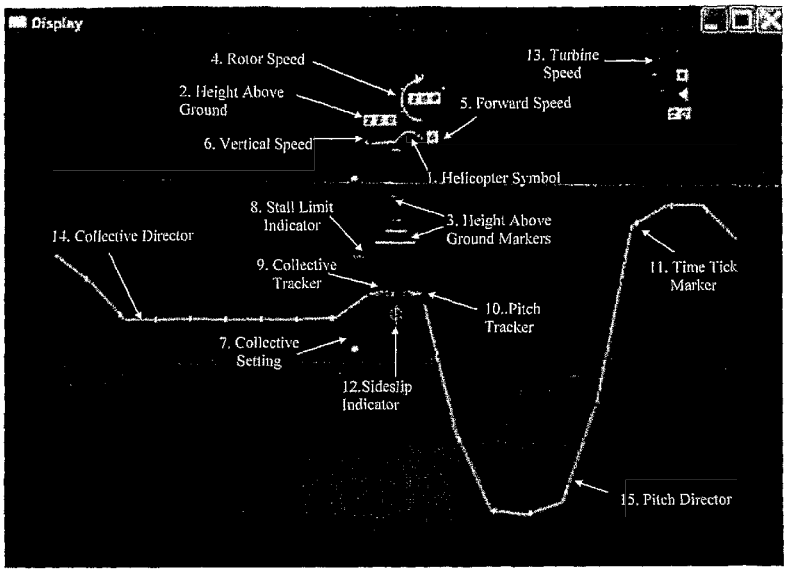
(74) Agent: **HOKANSON, Jon E.**; Lewis Brisbois Bisgaard & Smith LLP, 221 N. Figueroa Street, Suite 1200, Los Angeles, CA 90012 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT,

[Continued on next page]

(54) Title: AUTOROTATION FLIGHT CONTROL SYSTEM



(57) Abstract: The present invention provides computer implemented methodology that permits the safe landing and recovery of rotorcraft following engine failure. With this invention successful autorotations may be performed from well within the unsafe operating area of the height- velocity profile of a helicopter by employing the fast and robust real-time trajectory optimization algorithm that commands control motion through an intuitive pilot display, or directly in the case of autonomous rotorcraft. The algorithm generates optimal trajectories and control commands via the direct-collocation optimization method, solved using a nonlinear programming problem solver. The control inputs computed are collective pitch and aircraft pitch, which are easily tracked and manipulated by the pilot or converted to control actuator commands for automated operation during autorotation in the case of an autonomous rotorcraft. The formulation of the optimal control problem has been carefully tailored so the solutions resemble those of an expert pilot, accounting for the performance limitations of the rotorcraft and safety concerns.

WO 2006/076647 A2

RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *of inventorship (Rule 4.17(iv))*

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

AUTOROTATION FLIGHT CONTROL SYSTEM

The United States Government has a paid-up license in this invention and the right in limited circumstances to require the patent owner to license others on reasonable terms as provided for by the terms of NASA Contracts NAS2-02008 and NAS2-02096 awarded by the NASA, Ames Research Center, Moffett Field, California.

FIELD OF THE INVENTION

The present invention relates to a methodology using optimal control for application to the time critical maneuvering of dynamic systems including vehicles such as rotorcraft. The methodology is implemented in a computer-based system for calculating and displaying optimal control input commands to a human-operator for autorotation flight control of a rotorcraft and is adapted for training helicopter pilots in a flight simulator on safe maneuvering in time critical situations involving total engine power failure (autorotation) and partial power failure. The methodology can also be used for automated guidance of dynamic systems including vehicles such as rotorcraft in time critical maneuvering situations and in an automated system that will provide the highest likelihood of a safe landing if the pilot is incapacitated or if the vehicle is unmanned.

BACKGROUND ART

A series of analytical and experimental work has been done to understand and describe the nature of the dynamics and pilot's recovery techniques in rotorcraft's power failure. Johnson (Ref. 1) analytically described the dynamics of rotorcraft's autorotation. Lee (Refs. 2, 3), Zhao (Refs. 4-6), Carlson (Refs. 7-10), and Okuno (Refs 11, 12) investigated the application of constrained optimization to investigate the safe operational envelopes for autorotation and reduced-power situations for a variety of rotorcraft ranging from single-engine (OH-58A, Refs. 2-3) to multi-engine, for instance UH-60A and Bell M430, (Refs. 4-6, 8, 11, 12, 10) to tilt -rotor

(Refs. 7, 9, 10). Johnson (Ref. 1) investigated the autorotation of a helicopter from a hover, and Lee (Refs. 2, 3) refined the problem formulation by adding inequality constraints for thrust and vertical velocity. Lee postulated that the "avoid" regions in the height-velocity (H-V) restriction curve could be substantially reduced if optimal pilot inputs were used during autorotation. References 2 and 3 used a point-mass model of an OH-58A helicopter and the cost function was a weighted sum of the squared horizontal and vertical components of the helicopter velocity at touchdown. The point-mass model had two degrees-of-freedom (vertical and horizontal velocity) with an additional rotor speed degree-of-freedom. The inputs (horizontal and vertical thrust) required to minimize the cost function were computed using nonlinear optimal control theory. The correlation between flight data and the optimal results established the adequacy of the use of a point mass model in the optimal helicopter landing study (Ref. 2, 3). References 2 and 3 also validated the method by comparing the optimal profiles (helicopter states and controls) with available autorotation flight-test data for the OH-58A. A unique feature of the Refs. 2 and 3 formulation was the addition of path inequality constraints on components of both the control and the state vectors. The control variable inequality constraint is a reflection of the limited amount of thrust that is available to the pilot in the autorotation maneuver without stalling the rotor. The state variable inequality constraint is an upper bound on either the vertical sink rate of the helicopter or the rotor angular speed during descent. "Slack" variables were employed to convert these path inequality constraints into path equality constraints. The resultant two-point boundary-value problem with path equality constraints was successfully solved using the Sequential Gradient Restoration Algorithm (SGRA). With bounds on the control and state vectors, the optimal solutions obtained will realistically reflect the limitations of the helicopter and its pilot. The model in Ref. 2 and 3 used assumed zero-wind, vertical plane motion, and zero-slip flight. Zhao (Ref 4-6) extended the work by Lee (Ref. 2, 3) to investigate the takeoff and landing trajectories of a dual-engine helicopter in the event of a single engine failure. Zhao also used the SGRA for computing the optimal trajectories and used different constructions for the objective (cost) function to investigate optimal profiles for continued and rejected landings

and takeoffs in the event of a single engine failure. In addition to touchdown velocity, horizontal distance was also included in the objective function to examine the implications of an engine failure on the safe return and landing or continued flight of the helicopter. A point-mass model of a UH-60A helicopter was used in this work with improvements to the model to include engine torque and a ground-effect model. Carlson (Ref. 7-10) launched from the previous body of work and used optimal control theory to investigate the unsafe (avoid) regions of the H-V envelope in the event of single-engine failure as well as complete engine failure situations in a civil tiltrotor aircraft and a dual engine helicopter. A relatively sophisticated three degree-of-freedom (vertical and horizontal velocity and pitch attitude) rotorcraft model was used with an added rotor speed degree-of-freedom and a non-linear aerodynamic model of the XV-15 tilt-rotor aircraft and the Bell M430 helicopter. An important contribution of the Refs. 7-10 work was the improvement in the optimization method. The Ref. 7-10 work demonstrated that the SGRA optimization method was not robust in the face of more complex problem formulations. The Refs. 7-10 work successfully implemented a direct method of optimization (Ref. 13) where the continuous two-point boundary value problem is discretized into a parameter optimization problem. The optimization used a well-established and mature nonlinear programming algorithm that is commercially available (Refs. 14, 15). The present invention applies a similar strategy to compute the optimal control inputs and resulting flight path for rotorcraft autorotation.

List of References

(1) Wayne Johnson, "Helicopter Optimal Descent and Landing after Power Loss," NASA Technical Memorandum, NASA TM 73244, May 1977.

(2) Allan Y. Lee, "Optimal Landing of a Helicopter in Autorotation," Ph.D. Dissertation, Stanford University, July 1985.

(3) Allan Y. Lee, Arthur E. Bryson, Jr., and William S. Hindson, "Optimal Landing of a Helicopter in Autorotation," Journal of Guidance, Vol. 11, No. 1, pp 7-12, Jan.-Feb. 1988.

(4) Y. Zhao and R. T. N. Chen, "Critical Consideration for Helicopters During Runway Takeoffs," Journal of Aircraft, Vol. 32, No. 4, pp 773-781, Jul.-Aug. 1995.

(5) Y. Zhao, Ali A. Jhemi, and R. T. N. Chen, "Optimal Vertical Takeoff and Landing Helicopter Operation in One Engine Failure," Journal of Aircraft, Vol. 33, No. 2, pp 337-346, Mar.-Apr. 1996.

(6) R. T. N. Chen and Y. Zhao, "Optimal Trajectories for the Helicopter in One-Engine-Inoperative Terminal-Area Operations," Presented at the FVP Symposium on "Advances in Rotorcraft Technology", Ottawa, Canada, May 1996.

(7) Eric B. Carlson, "Optimal Tiltrotor Aircraft Operations During Power Failure," Ph.D. Dissertation, University of Minnesota, July 1999.

(8) Eric B. Carlson, "An Analytical Methodology for Category A Performance Prediction and Extrapolation," Presented at the American Helicopter Society 57th Annual Forum, Washington, DC, May 9-11, 2001.

(9) Eric B. Carson and Y. Zhao, "Prediction of Tiltrotor Height-Velocity Diagrams Using Optimal Control Theory," Journal of Aircraft, Vol. 40, No. 5, pp 896-905, Sep.-Oct. 2003.

(10) Ali A. Jhemi, Eric B. Carlson, Y. Zhao, and R. T. N. Chen, "Optimization of Rotorcraft Flight Following Engine Failure," Journal of American Helicopter Society, Vol. 49, No. 2, pp 117-126, Apr. 2004.

(11) Y. Okuno and Keiji Kawachi, "Optimal Takeoff of a Helicopter for Category A V/STOL Operation," Journal of Aircraft, Vol. 30, No. 2, pp 235-240, Mar.-Apr. 1993.

(12) Y. Okuno, Keiji Kawachi, Akira Azuma, and Shigeru Saito, "Analytical Prediction of Height-Velocity Diagram of a Helicopter Using Optimal Control Theory," Journal of Guidance, Vol. 14, No. 2, pp 453-459, Mar.-Apr. 1991.

(13) C. R. Hargraves and S. W. Paris, "Direct Trajectory Optimization Using Nonlinear Programming and Collocation," Journal of Guidance, Vol. 10, No. 4, pp 338-342, Jul.-Aug. 1987.

(14) Philip E. GILL, Walter MURRAY, Michael A. SAUNDERS, and Margaret H. Wright, "USER'S GUIDE FOR NPSOL 5.0," Technical Report SOL 86-1, Stanford University, Revised July 30, 1998.

(15) Philip E. GILL, Walter MURRAY, and Michael A. SAUNDERS, "USER'S GUIDE FOR SNOP Version 6.0," Stanford University, December 2002.

(16) Watts, Joseph C., Gregory W. Condon, and John V. Pincavage, "Height-Velocity Test, OH-58A Helicopter," USAASTA Project No. 69-16, June 1971.

(17) Dooley, L. W. and Yeary, R. D., "Flight Test Evaluation of the High Inertia Rotor System," USARTL-TR-79-9, June 1979.

(18) E. N. Bachelder and Bimal L. Aponso "Using Optimal Control for Rotorcraft Autorotation Training," Proceedings of the American Helicopter Society 59th Annual Forum, Phoenix, Arizona, May 6 – 8, 2003.

SUMMARY DISCLOSURE OF THE INVENTION

The autorotation capability of helicopters following engine power failure is a unique feature that can provide a means for executing a safe landing. However, the autorotation

maneuver can require considerable skill and proficiency that is not normally acquired through nominal flight training.

In most autorotation training, pilots receive in-flight instruction on autorotation technique using initial conditions that are well outside of the hover-velocity (H-V) restriction curve of the helicopter flown - and the engine remains powered. Additionally, the entry conditions (altitude, relative wind direction, and especially airspeed) are usually consistent from one practice autorotation to another (within model and instructor). Autorotation training in a simulator is an infrequent event for most pilots, and even the best simulators poorly reproduce the cues required during an actual autorotation. The primary utility of simulators as an autorotation training aid, therefore, is to develop a proficient instrument scan procedure. The likelihood of a successful autorotation performed under actual instrument conditions, however, is extremely remote. Clearly rotary pilots have few resources to help them train toward and maintain autorotation proficiency, so that the autorotation is usually regarded as a 'take what comes and pray' maneuver.

In one aspect the present invention comprises the application of a real-time trajectory optimization method for guiding a manned rotorcraft, an autonomous unmanned rotorcraft, or a remote operator of an unmanned rotorcraft, through an autorotation in the event of partial or total loss of power. The invention provides for safe landing of such a rotorcraft. Further, successful autorotations may be performed from well within the manufacturer's designated unsafe operating area of the height-velocity profile of a rotorcraft or helicopter by employing the fast and robust optimal algorithm of the present invention. The invention applies nonlinear constrained optimal control theory to solve for a vehicle's trajectory and the required control inputs to accomplish a successful autorotation. The guidance algorithm of the present invention generates optimal trajectories and control commands via the direct-collocation optimization method, solved using a commercially available nonlinear programming problem solver. The control inputs computed by optimal control formulation are collective pitch and aircraft pitch, which are easily manipulated by an onboard or remote pilot or converted to collective and longitudinal cyclic commands in the

case of an autonomous unmanned rotorcraft. The formulation of the optimal control problem has been carefully tailored to enable the solutions to resemble those of an expert pilot, accounting for the performance limitations of the rotorcraft as well as safety concerns. A preview of the commanded flight control input suite, which is dynamically updated as the vehicle state changes in time, is provided to the pilot of a manned or remotely operated unmanned rotorcraft through an intuitive visual display. In the case of an autonomous unmanned rotorcraft the present invention provides commands for control motion directly through a link to a conventional commercially available autopilot.

In another aspect the present invention comprises a novel training methodology and a system that takes advantage of automation's potential as a high-speed decision aid and the strengths of human pattern recognition and conditioning. In this embodiment the invention is coupled with a flight simulator to train pilots across a range of rotorcraft platforms. Using the invention's command preview display and other display functions incorporated with a flight simulator a pilot trainee should be able to execute numerous maneuvers previously considered outside the operational envelope, in addition to performing 'standard' emergencies with a high degree of control consistency and accuracy.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a depiction of a single rotor helicopter.

Figures 2a and 2b depict a Frasca International Bell 206 Flight Training Device (FTD).

Figure 3 is a block diagram depicting the interface between the optimal guidance and the FTD.

Figure 4 is a Height-Velocity diagram for the Bell 206L-4 Helicopter Results.

Figure 5 is a diagram depicting the Automated autorotation flight conditions evaluated.

Figure 6 is a diagram depicting the touchdown ground-speed and sink-rate (light weight condition).

Figure 7 is a diagram depicting the touchdown ground-speed and sink-rate (medium and heavy weight conditions).

Figure 8 is a diagram depicting a time history for selected flight and control parameters for simulated automatic autorotation from 200ft/0kts; light weight condition (2900 lbs).

Figure 9 is a diagram depicting a time history for selected flight and control parameters for simulated automatic autorotation from 400ft/0kts; light weight condition (3100 lbs).

Figure 10 is a diagram depicting a time history for selected flight and control parameters for simulated automatic autorotation from 20ft/70kts; light weight condition (3085 lbs).

Figure 11 is a diagram depicting a time history for selected flight and control parameters for simulated automatic autorotation from 300ft/60kts; light weight condition (3085 lbs).

Figure 12 is a diagram depicting a time history for selected flight and control parameters for simulated automatic autorotation from 400ft/0kts; heavy weight condition (4440 lbs).

Figure 13 is a diagram depicting a schematic illustration of a first embodiment of the current invention adapted for training rotorcraft pilots on a flight simulator.

Figure 14 is a diagram depicting a system schematic of the current invention.

Figure 15 is a diagram depicting a description of guidance visual display components as a part of the current invention.

Figure 16 is a diagram depicting a schematic illustration of a second embodiment of the current invention adapted for automatically guiding a manned or unmanned rotorcraft.

Figure 17 is a diagram depicting a schematic illustration of a third embodiment of the current invention adapted as a computer-based training device for autorotation/reduced-power emergency flight.

INDUSTRIAL APPLICABILITY OF THE INVENTION AND MODES FOR CARRYING OUT THE INVENTION

The present invention is directed to systems for autorotation flight control, and in particular to computer implemented systems that provides directions for controlling the flight of

helicopters or of other rotorcraft upon loss of power to maximize the likelihood of a safe landing. The present invention may take the form of various embodiments, such as for example in a system adapted for a flight simulator for single engine, single rotor helicopters, a flight simulator for multiple engine, single or multiple rotor helicopters or a flight simulator for other rotorcraft. Embodiments of the present invention may also take the form of control systems for use in real working helicopters or other rotorcraft (as opposed to a simulator). When adapted for use in piloted working aircraft, the system is be adapted to provide display information for controlling the flight of the aircraft to maximize the likelihood of safe landing and/or is be adapted to provide automatic control inputs to the aircraft for such landings. When adapted for use in drones or other aircraft without pilots the system is be adapted for providing remote display for remote control of the aircraft and/or for automatic control inputs to the aircraft.

In the following description, numerous specific details are set forth to provide a more thorough description of embodiments of the invention. In light of the present disclosure, other embodiments will become obvious to those of ordinary skill in the art and such embodiments are within the scope of the present invention. It will be apparent, however, to one skilled in the art, that the invention may be practiced without these specific details. In other instances, well known features have not been described in detail so as not to obscure the invention. Except as noted herein, common components and connections, identified by common reference designators function in like manner.

In the description and included mathematical expressions the symbols used have the definitions or meanings stated in the following key to nomenclature:

a	rotor blade two-dimensional lift curve slope (rad^{-1})
C_P	power coefficient
C_T	thrust coefficient
(C_x, C_z)	(horizontal, vertical) component of thrust coefficient

c_{d_0} mean profile drag coefficient of rotor blades

f_e equivalent flat plate area for fuselage (ft^2)

f_G ground effect factor

f_I induced velocity factor

g gravitational acceleration (ft/s^2)

(h,d) (vertical, horizontal) position (ft)

H_{hub} rotor hub height when helicopter is on the ground (ft)

I_R polar moment of inertia of the main rotor blade (slug-ft^2)

J cost function

K_{ind} induced power factor

m mass of helicopter (slugs)

P_S available shaft power (lbf.ft/s)

P_{res} residual shaft power (lbf.ft/s)

R main rotor radius (ft)

t_f estimated flight time (s)

(u,w) (horizontal, vertical) velocity components (ft/s)

α tip path plane angle (rad)

γ weighting factor in cost function

δ_c collective pitch angle position (rad)

δ_{col} normalized collective pitch angle position

δ_{cyc} normalized longitudinal cyclic position

η helicopter power efficiency factor

λ rotor inflow ratio

μ rotor advance ratio

ρ	air density (slugs/ft^3)
σ	rotor solidity ratio
τ_p	turboshaft engine time constant (s)
θ	aircraft pitch angle (rad)
Ω	main rotor angular speed (rpm)
υ	rotor induced velocity (ft/s)
υ_h	induced velocity at hover (ft/s)
$()_0$	initial values at engine failure
$()_{max}$	maximum value allowed
$()_{min}$	minimum value allowed
$()_{ref}$	reference value
$()_{nom}$	reference value

The Rotorcraft Model

The rotorcraft equations of motions are detailed below.

$$m\dot{w} = mg - \rho(\pi R^2)(\Omega R)^2 C_z - \frac{1}{2}\rho f_e w\sqrt{u2 + w2} \quad (1)$$

$$m\dot{u} = \rho(\pi R^2)(\Omega R)^2 C_x - \frac{1}{2}\rho f_e u\sqrt{u2 + w2} \quad (2)$$

$$I_R\Omega\dot{\Omega} = P_S - \frac{1}{\eta}\rho(\pi R^2)(\Omega R)^2 C_P \quad (3)$$

$$\dot{h} = -w \quad (4)$$

$$\dot{d} = u \quad (5)$$

$$\dot{P}_S = \frac{1}{\tau_p}(P_{res} - P_S) \quad (6)$$

where, P_{res} is the steady-state power remaining following a throttle cut during a simulated engine failure.

In Eq. (6) a first order response is assumed for turboshaft engines (Ref. 4). The coefficients are defined as:

$$C_P = \frac{1}{8}\sigma c_d + C_T\lambda \tag{7}$$

$$C_x = C_T \sin\alpha \tag{8}$$

$$C_z = C_T \cos\alpha \tag{9}$$

λ is the inflow ratio defined as (Ref. 4):

$$\lambda = \frac{u\sin\alpha - w\cos\alpha + \nu}{\Omega R} \tag{10}$$

and the induced velocity ν is approximated as:

$$\nu = K_{ind} v_h f_I f_G \tag{11}$$

v_h is the reference induced velocity at hover defined as:

$$v_h = (\Omega R)\sqrt{(\frac{C_T}{2})} \tag{12}$$

The induced velocity parameter f_I is defined as the ratio of the actual induced velocity to the reference velocity v_h. The following expression is used to determine f_I:

$$f_I = \begin{cases} 1/\sqrt{(b^2 + (a + f_I)^2)} & If\ (2a+3)^2 + b^2 \geq 1.0; \\ a(.373a^2 + .598b^2 - 1.991) & otherwise \end{cases} \tag{13}$$

where, a and b are defined as:

$$a = \frac{u\sin\alpha - w\cos\alpha}{v_h} \tag{14}$$

$$b = \frac{u\cos\alpha + w\sin\alpha}{v_h} \tag{15}$$

The term f_G accounts for the decrease in induced velocity due to ground effect. The source model (Ref. 4) appears as:

$$f_G = 1 - \frac{R^2 \cos^2 \theta_w}{16(h + H_R)^2} \tag{16}$$

where,

$$\cos^2 \theta_w = \frac{(-wC_T + vC_z)^2}{(-wC_T + vC_z)^2 + (uC_T + vC_x)^2} \tag{17}$$

The tip path plane angle α and the aircraft pitch angle θ are effectively equivalent for the purposes of aircraft control. The collective pitch, computed using blade element theory (Ref. 2), appears as

$$\delta_c = \frac{(1 + \frac{3}{2}\mu^2)(\frac{6C_T}{a\sigma}) + \frac{3}{2}\lambda(1 - \frac{1}{2}\mu^2)}{(1 - \mu^2 + \frac{9}{4}\mu^4)} \tag{18}$$

where σ and a are the rotor solidity ratio and rotor blade two dimensional lift curve slope respectively. The advance ratio μ is defined as

$$\mu = \frac{u\cos\alpha + w\sin\alpha}{\Omega R} \tag{19}$$

The Optimal Autorotation Problem Formulation

A direct method of optimization was used following the work done by Carlson in Ref. 7. In the direct method the two-point boundary value problem is transformed into a parameter optimization problem. In such a formulation the states and controls are the parameters to be solved satisfying the dynamics and other physical limitations at discrete points in time (nodes), which can be solved using standard non-linear programming methods and software. The direct collocation method is used where both the rotorcraft states and controls are discretized

throughout time and the rotorcraft equations-of-motion are imposed as a set of non-linear equality constraints at each point in time (or node). Based on the experience documented in Ref. 7, this method has a better convergence radius with a wider range of initial guesses (more robust to initial guess values) than other parameterization methods. The disadvantage of this method is that the dimension of the problem becomes large due to the discretization of the states and control at each node or point in time. As in Ref. 8, the parameter optimization problem was solved using the Sequential Quadratic Programming (SQP) algorithm as implemented in the SNOPT software package (Ref. 15).

The Constraints On Solution of the Problem

(a) *Equality Constraints*

1. Initial Value Constraints

States: $(0,\ u_0,\ \Omega_0,\ h_0,\ 0,\ P_0)$

Controls: (C_{T_0}, α_0)

2. Final Value Constraints

States: $(\infty,\ \infty,\ \infty,\ 0,\ \infty,\ \infty)$

3. Equations of Motion at each node

(b) *Inequality Constraints*

1. State Constraints

$$-w_{\max} \le w \le w_{\max}$$

$$0 \le u \le \infty$$

$$\Omega_{\min} \le \Omega \le \Omega_{\max}$$

$$0 \le h \le \infty$$

$$0 \le d \le \infty$$

$$0 \le P_S \le \infty$$

2. Controls Constraints:

$$C_{T_{\min}} \le C_T \le C_{T_{\max}}$$
$$\alpha_{\min} \le \alpha \le \alpha_{\max}$$

The above constraints on states and controls are defined by

$$w_{\max} = 60 \text{ fps}$$
$$\Omega_{\min} = 0.75\Omega_0$$
$$\Omega_{\max} = 1.05\Omega_0$$
$$\alpha_{\min} = -20 \text{ deg}$$
$$\alpha_{\max} = 34 \text{ deg}$$

where, $\alpha_{\min}$ and $\alpha_{\max}$ are chosen as the minimum and maximum pitch values observed in flight test data (Refs. 16 and 17). $C_{T_{\min}}$ and $C_{T_{\max}}$ are aircraft-specific, with $C_{T_{\min}}$ associated with the minimum collective pitch, and $C_{T_{\max}}$ associated with blade stall. Also, to impose realistic collective range, the collective bounds are implemented such that:

$$\delta_{col_{\min}} \le \delta_{col} \le \delta_{col_{\max}}$$

The conversion between δ_{col} and C_T has been performed via Eq. (18) and an iterative method based on trim estimation. The constraint on the pitch angle near the ground has been imposed to prevent the tail from hitting the ground. The constraint is the function of aircraft geometry, such as the tailboom length, and altitude, and, as a result, the optimal solution guarantees that the aircraft's tail doesn't hit the ground at the final touchdown.

The Objective Function

The objective function is the sum of weighted penalties consisting of forward speed and sink rate at the final touchdown as well as the control rates for thrust coefficients and tip path angles at each node. The minimization of control rates provides smoother and consistent behavior of optimal solutions.

$$J = Q_1 \sum_{i=1}^{N-1} \left[\frac{C_T(i+1) - C_T(i)}{\Delta t} \right]^2 + Q_2 \sum_{i=1}^{N-1} \left[\frac{\alpha(i+1) - \alpha(i)}{\Delta t} \right]^2 + Q_3 (u_{t_f})^2 + Q_3 (w_{t_f})^2 \quad (20)$$

where, i is the node number (where $i = 1$ is the first node at t = 0) and Q_i represents proper weighting factors that is selective for best performance.

Validation of the algorithm using flight data was presented previously (Ref. 18) and showed that the optimal trajectories computed with this formulation were reasonable when compared with those accomplished by an expert pilot in flight tests.

The Flare Law

In real-time application for automated autorotation, performance differences between the rotorcraft dynamics and the point-mass model used in the optimization as well as simulation timing issues cause a mismatch in the altitude predicted by the optimization algorithm and the actual altitude of the rotorcraft (simulation, in this case). During initial development it was noticed that this mismatch caused the rotorcraft to flare too early or too late. To compensate for these deficiencies, a flare law was devised that would take over from the optimal guidance at a pre-determined altitude near the ground and flare the rotorcraft based on a more conventional compensatory control law. In practical terms, this flare law attempted to recreate the final flare and landing performed by a pilot based on outside visual cues. The purpose of the optimal trajectory was to bring the rotorcraft to a pre-flare altitude at an energy condition that was conducive to a safe flare and landing.

The flare law is preferably activated at a height of approximately 30 ft above ground and uses a non-linear algorithm to modulate airspeed through rotorcraft pitch attitude and to modulate rotor-speed and sink-rate through collective control. The activation altitude required

some adjustment during development and evaluation to compensate for the variations in aircraft weight.

Brief Description of the Simulator

Development and evaluation of the automatic autorotation and autorotation flight director display of the present invention took place on a commercial helicopter Flight Training Device (FTD) manufactured by Frasca International, Urbana, IL. Although not officially certified, the FTD used for the evaluation incorporated a level of fidelity necessary for achieving FAA Certification as a Level 4 FTD. The FTD was a fixed-base simulation of a Bell-206L-4 single-turbine, single rotor helicopter (Figure 1) with a realistic reproduction of the cockpit with a frame and dual controls and a dome visual system with 180-deg horizontal and 60-deg vertical visual field-of-view (Figure 2). An additional graphics channel provided visual imagery immediately below the cockpit door and through the chin window on the pilot's side. The cockpit controllers were replicas of the actual cyclic, collective and pedal controls and had realistic feel.

Complete engine failures could be triggered from the simulator operator's station at any time. Engine failures resulted in immediate loss of all engine power and the activation of appropriate warning lights and audio alarms. A low-rotor RPM warning light was also provided. The rotorcraft simulation model was a rotor disk model with aerodynamic models for the fuselage and empennage surfaces. The rotorcraft model had previously been evaluated by line pilots as part of the FTD acceptance testing and found to be representative of the actual aircraft in the regular and autorotation flight regimes. The primary development pilot for this project, Ed Bachelder, an experienced helicopter pilot (SH-60B pilot) also found the rotorcraft simulation to be realistic.

Implementation of the Optimal Guidance Algorithm

With reference to Figure 3, a block diagram indicates how a laptop personal computer (PC) running the real-time optimization algorithm was linked with the Frasca simulation computer.

The PC used for the development and evaluation of the optimal guidance was a conventional commercial laptop PC with a 2 GHz Intel Pentium® processor and a Windows 2000® operating system. The PC accepted rotorcraft state and control information at a nominal 30Hz data rate and output collective, cyclic, and pedal control positions to the simulation computer, also at a 30Hz data rate. Communication was facilitated through an Ethernet link using standard Microsoft Windows compatible communication protocol. During powered flight, the optimal algorithm continuously updated the optimal solution based on the rotorcraft states (primarily speed and altitude) being received from the simulation computer. In effect, the optimizer continuously computed an updated optimal trajectory for autorotation with the assumption that an engine failure had just occurred. Typically, a new update was available every 3 sec-or sooner. Initially, when an engine failure occurred, the automatic autorotation guidance was based on the last optimal trajectory update that was available. As presently implemented, the optimal trajectory is updated throughout the autorotation maneuver. The optimal guidance algorithm considers only the optimal trajectory in the longitudinal axis (collective and longitudinal cyclic commands only). During the development and evaluation process a simple compensatory feedback control was implemented to maintain roll attitude and heading via lateral cyclic and pedal commands.

During the development and evaluation process a guidance display was generated on the laptop computer to provide an indication of how well the helicopter was following the optimal guidance during automatic autorotations. For piloted operations of actual working aircraft, such as with a remote operator, the display is used as a flight director to guide the operator on the optimal control timing and magnitude inputs required to accomplish a safe landing. The

guidance display includes a novel display concept that guides a human operator in following and performing the optimal control inputs by providing a preview of the complete trajectory.

The primary intent of the development and validation of the optimal guidance algorithm in this real-time simulation environment was to evaluate the robustness of the guidance algorithm across the flight envelop of the simulated helicopter. Invariably, however, emphasis was placed on the "worst case" flight conditions; i.e., entry into autorotation from flight conditions that are well within the "avoid" region of the height-velocity diagram for this helicopter (shown in Figure 4) as these clearly illustrate the benefit of the optimal guidance provided by the present invention. Development and refinement of the optimal guidance algorithm and its real-time mechanization at flight conditions within the "avoid" region of the H-V diagram also maximizes the probability that the guidance provided by the present invention will enable safe autorotations from flight conditions outside the avoid region. The majority of the development and evaluation of the optimal guidance and the flight director display was performed at a vehicle light-weight condition with limited evaluations at the vehicle heavy (maximum gross weight) and medium weight conditions.

The optimal control algorithm uses a simple point-mass type model for the rotorcraft. For the algorithm to provide appropriate autorotation guidance, therefore, it was necessary to fine-tune the point-mass model parameters such that the dynamics and performance of the point-mass model approximated the rotorcraft model as implemented in the simulator as closely as possible. For automated autorotations, it was particularly important to scale and bias the optimal control inputs computed by the optimal algorithm so that it would be able to backdrive the simulation correctly. An automated procedure was setup using Matlab® to facilitate this parameter optimization process using rotorcraft state and control time history data obtained from the simulator.

The Simulator Results

Following three-week period of development on the Frasca FTD in Urbana, IL, the automatic autorotation capability was refined to an extent that allowed evaluation of the algorithm over a range of autorotation entry conditions. The entry conditions that were attempted at light (2900 lbs), medium (3500 lbs), and heavy (4450 lbs) vehicle weight configurations using the automatic autorotation guidance are presented on a height-velocity diagram in Figure 5. The manufacturer's height-velocity "avoid" regions are indicated in Figure 5 by dashed lines labeled for the rotorcraft's weight. Successful landings are shown as open or clear symbols and crash landings are shown as solid or filled1 symbols. Crash landings represent those where the touchdown sink-rate or forward speed exceeded the manufacturer's specified limitations for the rotorcraft. Tail-strikes were also counted as crash landings. The determination of a safe or crash landing was made by the Frasca simulation software.

As may be observed with reference to Figure 5, it is clearly established that using the optimal guidance of the present invention, safe autorotations are possible from well inside the "avoid" regions of the H-V curve including the high-speed region. Fewer evaluations were conducted at the medium and heavy vehicle weight conditions. At the heavy and medium vehicle weight conditions, it is expected that refining the constraints (rotor-speed and vertical speed limits, for example) as well as the flare law parameters would have allowed greater success than was demonstrated during the course of development and evaluation of the algorithm. Nevertheless, safe landings were accomplished at these weight conditions from well within the "avoid" regions of the H-V curve for these weights, although not with the consistency that was achieved at the light-weight condition.

The touchdown sink-rates and forward speeds for all the automated autorotation entry conditions shown in Figure 5 are presented in Figure 6 (light-weight condition) and Figure 7 (medium and heavy weight conditions). Figures 6 and 7 indicate that, in most situations, touchdown conditions were well within the limitations of the rotorcraft. Almost all the landings

were accomplished with some forward velocity. This is especially true in the heavy and medium weight conditions. This is primarily due to the use of the flare law for the landing. Examination of the optimal solutions for these evaluations indicated that if the helicopter had been landed using the optimal algorithm (assuming the aforementioned technical difficulties were resolved), the forward velocities at touchdown would have been reduced.

Selected representative time histories for the automated autorotations are presented in Figures 8, 9, 10, and 11 for the light weight condition and Figure 12 for the heavy-weight condition. In each of these examples, the engine is failed at time t = 0 and the displayed time history is ended when touchdown is registered by the simulation computer. Figures 8 and 9 demonstrate the extreme nature of the maneuver that is required when autorotating from a hover at 200 ft and 400 ft altitude (above ground level). Figures 8 and 9 demonstrate that it is possible to autorotate safely from well within the avoid region of the H-V curve, if the control inputs are well-timed and of appropriate magnitude. At the lower entry altitude (Figure 8), immediate nose down pitch attitude of approximately 30 degrees is commanded whereas collective is lowered to zero over a period of roughly 3 sec following engine failure. A pitch pull-up is commenced at an altitude of approximately 100 ft continuing into a landing flare using pitch attitude and collective input at approximately 50 ft altitude. The sharp discontinuity in the longitudinal cyclic at approximately 50 ft altitude marks the transition from the optimal algorithm to the flare law. Rotor speed is maintained above 80% throughout most of the maneuver with rotor speed reducing to 60% at touchdown as rotor speed is sacrificed to reduce the touchdown sink rate. No attempt was made to refine the algorithm to smoothly transition between these modes, hence the sharp discontinuity. Modification of the algorithm and/or the flare law to smooth the transition between these modes is within the skill of one of ordinary skill in the art and is within the scope of the present invention.

With reference to Figure 8, the longer maneuver time allowed by the higher entry altitude is evident. The collective is lowered immediately but there is no command to push the nose over or pitch down and gain airspeed until the rotor-speed approaches its lower constraint of 75%. To

maintain rotor speed above the constraint of 75%, the optimal guidance algorithm trades altitude for airspeed and for maintaining rotor speed. With reference to Figure 9, a maximum nose-down pitch attitude of 40 degrees is observed. A run-on landing is achieved at a forward speed of approximately 20 kts and a touchdown sink rate of almost zero.

Figure 10 demonstrates the effectiveness of the optimal guidance algorithm for an entry condition in the high-speed "avoid" region of the H-V curve. Due to the low altitude, the flare law almost immediately overrides the optimal algorithm. The helicopter is commanded to pitch up and trades airspeed for rotor-speed and altitude, placing it in a suitable energy state for a safe flare and touchdown at a forward speed of less than 10 kts. Figure 11 demonstrates an autorotation from an entry condition that is outside the manufacturer's recommended avoid region of the H-V curve for the light-weight condition. In response to the optimal guidance commands, the helicopter initially pitches nose-up to reduce airspeed followed by nose-down pitch to gain airspeed and maintain rotor speed above the constraint limit of 75%. Touchdown is achieved at a sink rate of 3 ft/sec and a forward speed of 40 kts.

The capability of the automatic guidance algorithm of the present invention to safely autorotate for the heavy-weight condition is demonstrated in Figure 12. The engine is failed when the helicopter is at a hover at an altitude of 400 ft above ground. When contrasted with an autorotation from a similar entry condition for the light-weight condition (Figure 9), the helicopter sinks more rapidly resulting in a shorter flight time. The optimal guidance commands an almost immediate push-over to gain airspeed (contrast with almost no pitch input for several seconds in Figure 9) and maintain rotor-speed with a very rapid pull-up to about 35 degrees to arrest sink rate at low altitude. The pull-up results in the rapid increase in rotor-speed to approximately 100% which is traded-off for sink-rate reduction using collective. Touchdown is achieved at a sink rate close to zero and a forward speed of 27 kts. As would be expected the heavier weight conditions proved to leave very little room for computational or timing errors.

The appropriately formulated optimization algorithm of the present invention may be used to provide autorotation guidance in real-time to a rotorcraft. This "automated autorotation"

capability is beneficial on unmanned rotorcraft where redundancy for failure management is not necessarily a primary design requirement. The present optimal guidance method has demonstrated a repeatable capability to safely autorotate a helicopter from a variety of entry conditions and a range of weights, even when these entry conditions are well within the avoid region of the height-velocity diagram.

Display Implementation

The present invention relates to a human-operator cueing and training methodology using optimal control for application to the time critical maneuvering of dynamic systems including vehicles. The methodology can also be used for automated guidance of dynamic systems through time critical maneuvers. The description of the invention uses a particular application example of rotorcraft pilot training and automatic guidance. Figure 13 illustrates the invention when applied for training rotorcraft pilots on autorotation and reduced-power flight using a flight simulator. In this application (Figure 13), a standard PC with the invented system installed is linked with a flight simulator and accepts rotorcraft state and control information from the connected flight simulator. Communication uses an Ethernet link using standard Microsoft Windows compatible communication protocol. During powered flight, the optimal algorithm continuously updates the optimal solution based on current rotorcraft states being received from the simulation computer. Thus the optimizer continuously computes an updated optimal trajectory for autorotation with the assumption that an engine failure had just occurred. Typically, a new update is available within a couple of seconds. When an engine failure occurs, the automatic autorotation guidance is based on the last optimal trajectory update that was computed. The optimal algorithm considers only the optimal trajectory in the longitudinal axis (collective and longitudinal cyclic commands only).

Figure 14 describes the software implementation of the optimal algorithm as a flowchart. The software starts with initializing all necessary rotorcraft parameters and setting all necessary

constraints and costs to compute the optimal controls. The parameters are vehicle specific so that they can be adjusted for different vehicles and dynamic systems -- a rotorcraft in this application. Next, the current flight conditions as well as the current environmental information such as wind, weight changes, and atmospheric temperature changes to computes air density are read into the software. The rotorcraft collective control input position from the flight simulator is converted to thrust that is used in the rotorcraft dynamic model to compute optimal controls. The software also estimates the best guess values of optimal controls to facilitate the computation of the optimal guidance solution. After the software finishes the computation, it converts the optimal thrust solution to collective and cyclic control inputs that can be displayed on the guidance display.

A guidance display is also generated on the PC that provides a preview of the optimal control solution with time and facilitates tracking of the optimal solution by the pilot through the maneuver. To learn the optimal control inputs necessary for safe recovery from the power-loss or reduced-power situation, the pilot simply has to track the guidance lines as discussed below. Repeated flights on a flight simulator using this guidance will provide the pilot with a clear understanding of the control inputs and rotorcraft trajectory to be flown for safe recovery. Figure 15 illustrates the guidance display.

With reference to Figure 15, the rotorcraft or helicopter symbol (1) is denoted by a stylized graphic intended to be readily recognized as a side view of a helicopter and the key aircraft states are anchored to this symbol to facilitate rapid mental processing as the symbol moves on the display. The helicopter symbol (1) also pitches with the helicopter pitch. The dimensions of the helicopter symbol (1) are drawn to scale with the altitude axis so that the pilot can see when tail contact is imminent and the relation between tail height and pitch.

With further reference to Figure 15, the helicopter tail acts as a pointer to the radar altimeter readout (2). The radar altimeter readout (2) is preferably positioned behind or aft of the helicopter symbol (1) on the display. A series of short horizontal lines arrayed vertically or stacked below the helicopter symbol (1) is an altitude pipper or height above ground markers (3)

which indicate the height-above-ground by short horizontal lines or markers corresponding preferably to heights of 150, 80, 40, 20, 10, and 0 feet. If the helicopter is above 150 feet (as in Figure 15), the helicopter symbol (1) will remain fixed at the 150 feet marker until the altitude goes below 150 feet, at which point the helicopter symbol (1) begins descending. A rotor speed indicator (4) includes a rotary pointer and digital readout box that is positioned above the helicopter symbol (1). The rotor speed indicator (4) changes from steady to blinking if the rotor speed falls below 90 % or rises above 110 %. A forward speed indicator (5) emanates and extends as a (body-axis referenced) vector from the nose of the helicopter symbol (1). The length of the vector (5) is in direct proportion to the forward speed of the helicopter. The forward speed readout in knots is tagged to the head of the forward speed indicator vector (5). A vertical speed indicator (6) vector (ground referenced) emanates and extends vertically downward from the tail of the helicopter symbol (1). The vertical speed vector (6) originates from the tail since this is the natural point of interest for that state. The forward and vertical speed vectors are shown in Figure 15. The scales on the vertical and forward speeds are identical and dimensioned with respect to the radar altimeter (i.e., 10 fps corresponds to a 10 foot increment on the altimeter (2)). The ticks on the vertical speed vector correspond to 5 fps, while on the forward speed bar ticks denote 10 knots increments. It is to be noted that the lengths of these vectors are scaled so that the pilot can weigh them equally. When the vertical speed vector touches the ground reference marker (attitude pipper), there is one second remaining prior to tail contact (based on the current vertical speed), at which time the pipper blinks in intensity to alert the pilot of the impending contact. This unique feature results from the scaling chosen, allowing the pilot to refine control timing.

With continued reference to Figure 15, a collective range setting indicator (7) scale is positioned on the display to the left of the altitude pipper or height above ground markers (3). The white ticks on the collective indicator (7) denote the 0% and 100% collective positions. The rotor blade stall limit indicator (8) (red bar) shows the collective setting corresponding to the blade stall limit at that particular point in time, and it varies considerably throughout the

autorotation. The collective range setting indicator (8) moves correspondingly with the collective inputs from the pilot. A left-pointing triangle (9) positioned below the altitude pipper (3) and to the right of the collective indicator (7) points to and tracks the current collective position. If this collective tracker pointer (9) nears or exceeds the rotor blade stall limit, it will change from a steady preferably white color to blinking alternating colors to alert the pilot that lift will be lost. One aspect of the maneuver that is almost never considered in autorotation training is the stall limit (presumably because one can't see it or predict it with the standard instrument layout), but it easily exceeded, to the detriment of the maneuver. This limit is predicted based on the point-mass helicopter model. The pointers are fixed in the display to allow the pilot better tracking. The collective range indicator moves with the collective input from a pilot so that the pilot can have a clear idea of his current collective input and the overall possible range of collective movements. A right-pointing triangle (10) positioned below the altitude pipper (3) and to the right of the collective tracker pointer (9) points to and tracks the current pitch position. The white right-pointing triangle below the altitude pipper points to the current pitch position. For example, the pilot should follow the pitch commands displayed in Figure 15 with the pitch tracker pointer (10). Time marks (11) are displayed on the optimal collective and pitch commands as tick marks for every second to give a pilot a better preview of overall profiles and the anticipated time remaining to complete maneuver.

Referring further to Figure 15, the sideslip indicator (12) is shown below the attitude pipper as a ball referenced to a fixed vertical centerline. The sideslip indicator ball will move to the right or left with respect to the nominal centerline in response to corresponding sideslip. The engine turbine speed indicator (13) is shown on the upper right of the display as a rotary pointer and digital readout box for displaying percent of turbine maximum speed. Guidance commands to the collective (left white line) and pitch (right white line) are displayed as time profiles for the collective director (14) command suite and pitch director (15) command suite, with a time tick for every second. The contact points with the collective and pitch pointers represent the present time or time equal to zero. The command profile lines indicate the anticipated time to complete

autorotation in seconds. These profile lines move in time so that the collective command profile scrolls right and the pitch command profile scrolls towards the left. The pilot must move the controls to minimize the vertical separation between the current control setting (left collective tracker pointer (9), right pitch tracker pointer (10)) and the coincident command. A crucial advantage that the present display has over the more traditional flight director is that the pilot is given a highly usable view of future control motion and time. Using this preview the pilot can anticipate control motion as well as anticipated time to complete maneuver, which is critical to precise and timely control tracking. The optimal commands will change from steady color to blinking with a different color when the "auto flare law" would be activated if the autopilot mode were in use. In this way, a pilot will be alerted to prepare for the landing flare. The color of the optimal commands change to denote the quality of optimal solutions. Due to the rapid changes of entry conditions and numerical complexity associated with the optimization algorithm the optimal solution might not have converged. In this case, the optimal commands change color to indicate that the commands are not based on a converged solution, in which case the displayed commands are from the last solution that converged.

Figure 16 illustrates the application of the invention to automatic control of a vehicle or dynamic system (a rotorcraft in the example application). The implementation is similar to that indicated in Figure 13 except that the optimal control solutions are fed back to the flight simulator or actual vehicle and used to replace the normal control inputs. The optimal solution will then guide the simulator or actual vehicle to a safe recovery from the power-loss or reduced-power situation. When acting as an automatic guidance and control system, a compensatory feedback control law is also implemented to maintain roll attitude and heading via lateral cyclic and pedal commands and the system sends the optimal control commands to the simulator to drive the simulator for safe landing in autorotation. A separate flare algorithm takes over near the ground to compensate for possible differences in the rotorcraft dynamics between the system and the simulator.

Figure 17 illustrates the application of the invention to a computer-based training device for rotorcraft autorotation and reduced-power emergency flight situations. The basic operation of the algorithm follows that depicted in Figure 14 except that there is no connection with a simulation or flight vehicle. The optimal solution is displayed to the trainee pilot and the simulated rotorcraft together with a computer-generated scene of the pilot's view out of the rotorcraft. When activated by the trainee pilot, the simulated rotorcraft follows the computed optimal trajectory, providing the trainee pilot with an understanding of the rotorcraft attitudes, path and control inputs necessary for safe recovery. The software will allow the trainee pilot to adjust the rotorcraft initial and final conditions and examine the effect of these conditions on the optimal solution.

In order to give the pilot proficiency at entering the autorotation profile, simulated engine failure is initiated at various altitudes, airspeeds, and horizontal locations relative to a geographically fixed landing site. This will exercise the full envelope of entry conditions without the pilot having to indicate to the computer the intended point of touchdown. The display also may be used as an on-board pilot preview of the optimal autorotation maneuver strategy. As the helicopter readies for departure from a hover, the autorotation computer will begin computing the optimal inputs and display them. The pilot would include the display in his instrument scan so that if the engine were to fail at any given time an image of the control profile would be mentally available. The entry into the autorotation would therefore be executed precognitively, followed by scanning of the autorotation display and cockpit instruments during the steady-state phase (if there is one) and just prior to the flare. In instances where out-of-balance flight is required, (to prevent site overshoot, rotor overspeed, or to compensate for other conditions) the pedal control profile will command appropriately so that the pilot may develop skill in slipping the helicopter according to the situation.

The training display concept of the present invention where the operator is provided with visual cues on where to place the controls at the current instant as well as provide a preview of where the controls should be in the future (based on the optimal algorithm) has application to any

vehicle or device that requires time-critical inputs for safe operation. Employing trajectories and control inputs using constrained optimization can be applied to any vehicle or device that requires time-critical inputs for safe operation.

The concepts, algorithms and routines for implementing the real-time dynamic visual display methodology of the present invention are further disclosed and described in the following Table 1 which provides representative examples, in a common programming language, of computer code capable of implementing the primary portions, but not the entirety, of the visual display of the present invention in a suitable computer processing environment. Table 1 is a listing of the computer code for the DrawDisplay.CPP display guidance-commands and flying information routine of the Guidance-Commands Display and Communication Module of the computer implementation of the present invention.

The scope of the appended claims will be clear from the entirety of the present disclosure. It will be obvious to those of ordinary skill in the art that the concepts, algorithms and displays of the present invention may be implemented in alternative code formulations and/or in other programming languages and such alternative formulation or formulations are within the scope of the present invention.

Thus, a real-time trajectory optimization method for guiding a rotorcraft in the event of loss of engine power is described in conjunction with one or more specific embodiments. The invention is defined by the following claims and their full scope of equivalents.

```
/* ============================================================================== */
#include   <basetsd.h>
#include   <windows.h>                  // Header File For Windows
#include   <stdio.h>                    // Header File For Standard Input / Output
#include   <string.h>
#include   <stdarg.h>
#include   <gl\gl.h>                    // Header File For The OpenGL32 Library
#include   <gl\glu.h>                   // Header File For The GLu32 Library
#include   <gl\glaux.h>                 // Header File For The Glaux Library
#include   <gl\glut.h>                  // Graphics library
#include   <time.h>                     // Timing routines
#include   <math.h>                     // Math library
/* ============================================================================== */
/* ============================================================================== */
#include   "UnitConversion.H"
#include   "LoadParams.H"
#include   "Graphics.H"
#include   "States.H"
#include   "FlareLawParams.H"
/* ============================================================================== */
/* ============================================================================== */
#include   "Optimize.H"
/* ============================================================================== */
/* ============================================================================== */
#include   "DrawDisplay.H"
/* ============================================================================== */

DrawDisplay::DrawDisplay()
{
      agl     = 0;
//    alt_beg  = 100;
      land_flag = false;
//    time_bias = 0.0;
}

DrawDisplay::~DrawDisplay()
{
}

void DrawDisplay::GetValues(double _state[], double _ALP_NOW, double _COLL_NOW, double _col[],
                            double _t_sim, double _t_fail, double _t_fail_st,
                            bool _cmd_flag, bool _sim_init, bool _ENG_FAIL, bool _ot_flag,
                            double _chs_psi, int _inform)
{
      for (int i=0;i<NUMSTATES;i++) state[i] = _state[i];

      ALP_NOW  = _ALP_NOW;
      COLL_NOW = _COLL_NOW;

      for (i=0;i<(sizeof(_col)/sizeof(double));i++) col[i] = _col[i];

      t_sim    = _t_sim;
      t_fail   = _t_fail;
      t_fail_st = _t_fail_st;

      cmd_flag = _cmd_flag;
      sim_init = _sim_init;
      ENG_FAIL = _ENG_FAIL;
      ot_flag  = _ot_flag;
```

TABLE 1

```
    chs_psi = _chs_psi;
    inform = _inform;
}

void  DrawDisplay::ReturnValues()
{
}

Int DrawDisplay::DrawGLScene(OptResult *optR)
{
    // Here's Where We Do All The Drawing

    bool  blink  = false;
    bool  tst_flag = false;
    bool  anti   = true;                    // Antialiasing
    bool  SHAD     = false;
    bool  FIMP  = false;
    bool  TILE  = true;
    bool  AMRK     = true;

    double     xtr, ytr, ztr, xtrt, ytrt, xsc, ysc, fact, gnd_spd;
    double trail;
    double rgbf[4] = {1,1,1,.7};
    char  text1a[80],text1b[80],text1c[80],text2[80],text3a[80],text3b[80],text3c[80];
    char   text4[80],text5[80],text6[80],text7[80],text8[80];
    char text9[80], text10[80],text11[80],text12[80],text13[80],text14[80],text15[80];
    double     thta_dif, thta_tile, lin_dif,sbias, abias,tbias,wbias;

    if(!tst_flag)
    {
        trail = 100;
        if (view == OUT_THE_WINDOW)
        {
            nn = 0;
            en = 0;
            dn = 0;
        }
    else if(view != OUT_THE_WINDOW)
    {
        nn = -cos(state[PSI] + chs_psi)*trail;
        en = -sin(state[PSI] + chs_psi)*trail;
        dn = -5;
    }
    glPushMatrix();
    if (view == OUT_THE_WINDOW)        /* out-the-window view */
    {
    /* Transformations to set up coords to draw ground */
        glRotatef(90.0, 0.0, 1.0, 0.0);
        glRotatef(90.0, 1.0, 0.0, 0.0);

    /* xyz are now oriented with body axis */

    /* undo Euler angles */
        glRotatef(-state[PHI] * 180.0 / PI, 1.0, 0.0, 0.0);
        glRotatef(-state[THETA] * 180.0 / PI, 0.0, 1.0, 0.0);
        glRotatef(-state[PSI] * 180.0 /PI, 0.0, 0.0, 1.0);

    /* xyz are now oriented with NED axes, but still centered on aircraft */

    /* undo vehicle's position */
```

```
        glTranslatef(-state[NORTH], -state[EAST], -state[DOWN]);
    }

    else if (view == CHASEPLANE)
    {   /* chase plane view */
        glRotatef(90.0, 0.0, 1.0, 0.0);
        glRotatef(90.0, 1.0, 0.0, 0.0);

    /* xyz are now oriented with body axis */


    /* undo Euler angles */
        glRotatef((-state[PSI] - chs_psi) * 180.0 /PI, 0.0, 0.0, 1.0);


    /* xyz are now oriented with NED axes, but still centered on aircraft */

    /* undo vehicle's position */

        glTranslatef(-(state[NORTH]+nn), -(state[EAST]+en), -(state[DOWN]+dn));

        /* enter viewing transforms here to create proper chaseplane view */
    }

    else
    {   /* (view == TOWER) --> tower view */

        /* enter viewing transforms here to create proper tower view */
    }

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    if(view == CHASEPLANE)
    {
        glPushMatrix();
        draw_plane();
        draw_helshad();
        glPopMatrix();
    }
    draw_ground_plane();
    if(TILE)draw_tiles();

    if(SHAD)draw_shad();
    if(FIMP)draw_imp();
    draw_tchdwn();
    glPopMatrix();

    /* draw own aircraft if view from chaseplane or tower */

    if (view == OUT_THE_WINDOW)
    {
    glPushMatrix();
    glDisable(GL_DEPTH_TEST);
//  draw_pip();
//  draw_ladder();
//  draw_roll();
        if(t_fail < 4)draw_torque();
    draw_rpm();
    draw_coll();
    draw_alt();
```

```
        draw_bug();
        draw_ball();
        if(cmd_flag)
        {
//          if(inform == 0 || inform == 4 || inform == 9)
            if(inform == 0 || inform == 4 && !ENG_FAIL)
            {
//              {
                    update_cmds(optR);
//                  draw_cmds(optR);
//              }
            }
            draw_cmds(optR);
        }

        glEnable(GL_DEPTH_TEST);
        glPopMatrix();

        }
        /* draw own aircraft if view from chaseplane or tower */


        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texture[0]);
        fact = .01;
        xtr = -520.0*fact; ytr = -380.0*fact; ztr = -1000.*fact;
        xsc = 1.f*fact; ysc = 1.f*fact;


        strcpy(text1a, "CENTER MOUSE, PRESS 'C' KEY, LEFT CLICK MOUSE");
        strcpy(text1b, "PRESS 'C' KEY, PULL TRIGGER SWITCH");
        strcpy(text1c, "PRESS 'C' KEY");
        strcpy(text2, "Autopilot Mode");
        strcpy(text3a, "Mouse Control");
        strcpy(text3b, "Joystick Control");
        strcpy(text3c, "Keyboard Control");
        strcpy(text4, "GHOST");
        strcpy(text5, "T");
        strcpy(text6, "KTS");
        strcpy(text7, "AGL");
        strcpy(text8, "FPS");

        strcpy(text9, "DEG");
        strcpy(text10, "AIRCRAFT STATES AT TOUCHDOWN:");
        strcpy(text11, "SINK RATE (FPS)");
        strcpy(text12, "PITCH (DEG)");
        strcpy(text13, "GND SPD (KTS)");
        strcpy(text14, "NR (Perc)");
        strcpy(text15, "STALL MARGIN (Perc)");

        xtrt = xtr;
        ytrt = ytr;

        gnd_spd = sqrt(state[U]*state[U] + state[V]*state[V]);
        if(sim_init){

//      thta_tile = atan(ALT_INIT/(GNDSPD*TTGO))*57.3;
        thta_tile = atan(400.0/(GNDSPD*TTGO))*57.3;
        thta_dif = FOV/2 - thta_tile;
        lin_dif = (520/(FOV/2))*thta_dif;
```

```
    if(view == CHASEPLANE){

        if(agl <10.)abias = 16.;
        else if(agl >= 10. && agl < 100.)abias = 8.;
        else abias = 0.;

        if(state[U]/1.69 < 10.)sbias = 16.;
        else if(state[U]/1.69 >=10. && state[U]/1.69 < 100.)sbias = 8.;
        else sbias = 0.;

        if(state[W] < 10.)wbias = 16.;
        else if(state[W] >=10. && state[W] < 100.)wbias = 8.;
        else wbias = 0.;

        if(state[THETA]*rad2deg < 10.&& state[THETA]*rad2deg >= 0.)tbias = 16.;
        else if(state[THETA]*rad2deg >=10. && state[THETA]*rad2deg < 100.)tbias = 8.;
        else tbias = 0.;

        glPrint(blink,xtr+(840+abias)*fact,ytr+400*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],rgbf[3],"%1.0f",agl);
        glPrint(blink,xtr+(840)*fact,ytr+380*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],rgbf[3],text7);
        glPrint(blink,xtr+(180+sbias)*fact,ytr+400*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],rgbf[3],"%1.0f",state[U]/1.69);
        glPrint(blink,xtr+(180)*fact,ytr+380*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],rgbf[3],text6);


        glPrint(blink,xtr+(840+wbias)*fact,ytr+300*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],rgbf[3],"%1.0f",state[W]);
        glPrint(blink,xtr+(840)*fact,ytr+280*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],rgbf[3],text8);

    }

    }
//  if(state[ALT]/hfct < 5)land_flag = TRUE;
    if(land_flag){
        glDisable(GL_DEPTH_TEST);
        glPrint(blink,xtr+(600)*fact,ytr+220*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],1.,text10);
        glPrint(blink,xtr+(600)*fact,ytr+180*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],1.,text11);
        glPrint(blink,xtr+(600+300)*fact,ytr+180*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],1.,"%1.0f",state[W]/wfct);
        glPrint(blink,xtr+(600)*fact,ytr+160*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],1.,text12);

glPrint(blink,xtr+(600+300)*fact,ytr+160*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],1.,"%1.0f",state[THETA]*rad2deg);
        glPrint(blink,xtr+(600)*fact,ytr+140*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],1.,text13);
        glPrint(blink,xtr+(600+300)*fact,ytr+140*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],1.,"%1.0f",state[U]/1.69/ufct);
        glPrint(blink,xtr+(600)*fact,ytr+120*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],1.,text14);
        glPrint(blink,xtr+(600+300)*fact,ytr+120*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],1.,"%d",nr);
        glPrint(blink,xtr+(600)*fact,ytr+100*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],1.,text15);
        glPrint(blink,xtr+(600+300)*fact,ytr+100*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],1.,"%d",st_mrg);
        glEnable(GL_DEPTH_TEST);
    }

    if (anti)                                                   // Is Anti true?
    {
        glEnable(GL_LINE_SMOOTH);                               // If So, Enable Antialiasing
    }
    glDisable(GL_TEXTURE_2D);
    } // end of tst_flag
    return true;                                    // Everything Went OK
}

/* ================================================================================ */
```

```
/* draw alt guides                                                                    */
/* ================================================================================== */

/* ================================================================================== */
void DrawDisplay::draw_ladder(void)
{

        bool blink = false;
        int i, ist,ifn, p_flag;
        double th_pos;
        float tik_maj[3][3] = {{-.3, 0, 0. },
                                                {0, 0, 0. },
                                                { .3, 0, 0. }};

        float tik_min[2][3] = {{-.04, 0, 0. },
                                                { .04, 0, 0. }};
        float  xtr, ytr, ztr, xsc, ysc, fact;
        float  rgbf[4] = {1,1,1,.7};
        char  text1[80];

        glBindTexture(GL_TEXTURE_2D, texture[0]);
        fact = .01;
        xtr = -520.0*fact; ytr = -380.0*fact; ztr = -1000.*fact;
        xsc = 1.f*fact; ysc = 1.f*fact;

        strcpy(text1, "10");
//              glPrint(blink,xtr+480*fact,ytr+600*fact,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],.8,text1);

        glPushMatrix();
//      glColor4f(1,1,0, .5);
        glColor4f(1,1,1, .7);


    glRotatef(state[PHI]*57.3,0.,0.,1.);
              glTranslatef(1.9, 0, -10);


        p_flag = 0;
        if(state[THETA]*57.3 > 3){
                ist = 1;
                ifn = 2+ceil(state[THETA]*57.3/10);
                p_flag = 1;
        }
        else if(state[THETA]*57.3 < -3){
                ist = -1;
                ifn = -2+floor(state[THETA]*57.3/10);
                p_flag = 2;
        }
        if(p_flag == 1){

                for(i=ist;i<ifn;i++){
                glPushMatrix();
//              if(state[PHI] <=0){
                glPushMatrix();
                th_pos = -10*(double)sin(state[THETA] - i*10/57.3);
                glTranslatef(.15,th_pos, 0. );
                glLineWidth(2);
//              glRotatef(i*10,0.,0.,1.);
                glBegin(GL_LINES);
                glVertex3fv(tik_maj[0]);
```

```
            glVertex3fv(tik_maj[1]);
            glEnd();

            glRotatef(i*10,0.,0.,1.);
            glBegin(GL_LINES);
            glVertex3fv(tik_maj[1]);
            glVertex3fv(tik_maj[2]);
            glEnd();
            glPopMatrix();

            glPushMatrix();
            th_pos = -10*(double)sin(state[THETA] - ((i-1)*10+5)/57.3);
            glTranslatef(.15,th_pos, 0. );
            glLineWidth(1);
            glBegin(GL_LINES);
            glVertex3fv(tik_min[0]);
            glVertex3fv(tik_min[1]);
            glEnd();
            glPopMatrix();
//          }

//          if(state[PHI] > 0){

            glPushMatrix();
            th_pos = -10*(double)sin(state[THETA] - i*10/57.3);
            glTranslatef(-.15-3.8,th_pos, 0. );
            glLineWidth(2);
            glRotatef(-i*10,0.,0.,1.);
            glBegin(GL_LINES);
            glVertex3fv(tik_maj[0]);
            glVertex3fv(tik_maj[1]);
            glEnd();

            glRotatef(i*10,0.,0.,1.);
            glBegin(GL_LINES);
            glVertex3fv(tik_maj[1]);
            glVertex3fv(tik_maj[2]);
            glEnd();
            glPopMatrix();

            glPushMatrix();
            th_pos = -10*(double)sin(state[THETA] - ((i-1)*10+5)/57.3);
            glTranslatef(-.15-3.8,th_pos, 0. );
            glLineWidth(1);
            glBegin(GL_LINES);
            glVertex3fv(tik_min[0]);
            glVertex3fv(tik_min[1]);
            glEnd();
            glPopMatrix();
            glPopMatrix();


            }
        }
        if(p_flag == 2){
            for(i=ist;i>ifn;i--){
                glPushMatrix();
                glLineWidth(2);

                glPushMatrix();
```

```
        th_pos = -10*(double)sin(state[THETA] - i*10/57.3);
        glTranslatef(.15,th_pos, 0. );
//      glRotatef(i*10,0.,0.,1.);
        glBegin(GL_LINES);
        glVertex3fv(tik_maj[0]);
        glVertex3fv(tik_maj[1]);
        glEnd();

        glRotatef(i*10,0.,0.,1.);
        glBegin(GL_LINES);
        glVertex3fv(tik_maj[1]);
        glVertex3fv(tik_maj[2]);
        glEnd();
        glPopMatrix();

        glPushMatrix();
//      glLineWidth(3);
        th_pos = -10*(double)sin(state[THETA] - (i*10+5)/57.3);
        glTranslatef(.1,th_pos, 0. );
        glLineWidth(1);
        glBegin(GL_LINES);
        glVertex3fv(tik_min[0]);
        glVertex3fv(tik_min[1]);
        glEnd();
        glPopMatrix();


        glPushMatrix();
        th_pos = -10*(double)sin(state[THETA] - i*10/57.3);
        glTranslatef(-.15-3.8,th_pos, 0. );
        glLineWidth(2);
        glRotatef(-i*10,0.,0.,1.);
        glBegin(GL_LINES);
        glVertex3fv(tik_maj[0]);
        glVertex3fv(tik_maj[1]);
        glEnd();

        glRotatef(i*10,0.,0.,1.);
        glBegin(GL_LINES);
        glVertex3fv(tik_maj[1]);
        glVertex3fv(tik_maj[2]);
        glEnd();
        glPopMatrix();

        glPushMatrix();
//      glLineWidth(3);
        th_pos = -10*(double)sin(state[THETA] - (i*10+5)/57.3);
        glTranslatef(-.1-3.8,th_pos, 0. );
        glLineWidth(1);
        glBegin(GL_LINES);
        glVertex3fv(tik_min[0]);
        glVertex3fv(tik_min[1]);
        glEnd();
        glPopMatrix();

        glPopMatrix();
        }
}

glPopMatrix();
```

```
}

void DrawDisplay::draw_roll(void) {
        int i, ifn_min,ifn_maj, p_flag,num_i;
        float tik_out[3][3] = {{-.2, -.07, 0. },
                                                {-.2, .07, 0. },
                                                { 0, 0, 0. }};

        float tik_out_rt[3][3] = {{.2, -.05, 0. },
                                                {.2, .05, 0. },
                                                { -.2, 0, 0. }};


        float tik_in[3][3] = {{ .2, .07, 0. },
                                                {.2, -.07, 0. },
                                                {0, 0, 0. }};
        float tik[2][3] = {{-.03, 0, 0. },
                                                { .03, 0, 0. }};
        float tik_big[2][3] = {{-.2, 0, 0. },
                                                { .2, 0, 0. }};
        glPushMatrix();
        glColor4f(1,1,1, .5);
        glRotatef(state[PHI]*57.3,0,0,1);

        p_flag = 0;
        if(state[PHI]*57.3 > 3){
                ifn_min = 1+ceil(state[PHI]*57.3/10);
                ifn_maj = 1+ceil(state[PHI]*57.3/30);
                p_flag = 1;
        }
        else if(state[PHI]*57.3 < -3){
                ifn_min = -1+floor(state[PHI]*57.3/10);
                ifn_maj = -1+floor(state[PHI]*57.3/30);
                p_flag = 2;
        }

//      if(cmd_flag)num_i = 1;
//      if(!cmd_flag)num_i = 2;
        num_i = 2;

                for(i=num_i;i>0;i--){
                glColor4f(1,1,0, .5);
                glPushMatrix();
                glLineWidth(2);
                glRotatef(180*i,0,0,1);
                glTranslatef(-1.5,0,-10);
                glPushMatrix();
                glTranslatef(-.2,0,0);
                glBegin(GL_TRIANGLES);;
                glVertex3fv(tik_out_rt[0]);
                glVertex3fv(tik_out_rt[1]);
                glVertex3fv(tik_out_rt[2]);
                glEnd();
                glPopMatrix();
                glPopMatrix();
                }
        glLineWidth(2);
    if(p_flag == 1){

                for(i=1;i<(ifn_maj-1)*3;i++){
```

```
            glColor4f(1,1,0, .5);
            glPushMatrix();
            glRotatef(-10*i,0,0,1);
            glTranslatef(1.6,0,-10);
            glBegin(GL_LINES);
            glVertex3fv(tik[0]);
            glVertex3fv(tik[1]);
            glEnd();
            glPopMatrix();
            }

            for(i=1;i<ifn_maj;i++){
            glPushMatrix();
            glRotatef(-30*i,0,0,1);
            glTranslatef(1.6,0,-10);
            glBegin(GL_LINE_LOOP);;
            glVertex3fv(tik_in[0]);
            glVertex3fv(tik_in[1]);
            glVertex3fv(tik_in[2]);
            glEnd();
            glPopMatrix();
            }
      }


      if(p_flag == 2){

            for(i=-1;i>(ifn_maj+1)*3;i--){
                  glColor4f(1,1,0, .5);
            glPushMatrix();
            glRotatef(-10*i,0,0,1);
            glTranslatef(-1.6,0,-10);
            glBegin(GL_LINES);
            glVertex3fv(tik[0]);
            glVertex3fv(tik[1]);
            glEnd();
            glPopMatrix();
            }

            for(i=-1;i>ifn_maj;i--){
            glPushMatrix();
            glRotatef(-30*i,0,0,1);
            glTranslatef(-1.6,0,-10);
            glBegin(GL_LINE_LOOP);;
            glVertex3fv(tik_out[0]);
            glVertex3fv(tik_out[1]);
            glVertex3fv(tik_out[2]);
            glEnd();
            glPopMatrix();
            }
      }
  glPopMatrix();
}

void DrawDisplay::draw_torque(void) {

      bool blink;
      float XCTR, YCTR, move_x, move_y, tq_x, rad_1;
      float tik[3][3] = {{-.1, .12, 0. },
```

```
                                {0, 0, 0. },
                                {.1, .12, 0. }};
float ytik[2][3] = {{0, .025, 0. },
                                {0, -.025, 0. }};
float rad = .3;
int tq, i, r_cnt, r_1;
float  xtr, ytr, ztr, xsc, ysc, fact;
float  rgbf[4] = {1,1,1,.7};
char  text1[80];

glBindTexture(GL_TEXTURE_2D, texture[0]);
fact = .01;
xtr = -520.0*fact; ytr = -380.0*fact; ztr = -10;
xsc = 1.f*fact; ysc = 1.f*fact;
strcpy(text1, "TQ");

tq = (int) 100*state[PW]/pwmax;
r_1 = .5*(tq - 100);

move_x = 0;
move_y = 0;
XCTR = 2+move_x;
YCTR = 1.6+move_y;
tq_x = 0;
if(tq < 100 && tq >= 10)tq_x = .1;
if(tq < 10)tq_x = .2;
r_cnt = (50 + r_1);
glColor3f(.80, .80, .80);
blink = false;
if(tq < 5 && tq > 0)blink = true;
glPrint(blink,3.85+2*move_x,2.7+2*move_y,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],rgbf[3],text1);
glPrint(blink,3.78+2*move_x+tq_x,3.26+2*move_y,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],rgbf[3],"%d",(int)tq);
glPushMatrix();
glTranslatef(XCTR,YCTR,-10);
glBegin(GL_LINE_STRIP);
for (i = r_cnt; i > -1; i--)glVertex2f(XCTR + rad*cos(i*.02*PI+PI/2), YCTR - rad*sin(i*.02*PI+PI/2));
glEnd();
glEnd();
glPopMatrix();

glPushMatrix();
glTranslatef(2*XCTR + rad*cos(r_cnt*.02*PI+PI/2), 2*YCTR - rad*sin(r_cnt*.02*PI+PI/2),-10);
glRotatef(-(r_cnt*.02*PI+PI/2)*57.3, 0.0, 0.0, 1.0);
glBegin(GL_TRIANGLES);
        glVertex3fv(tik[0]);
        glVertex3fv(tik[1]);
        glVertex3fv(tik[2]);
glEnd();
glPopMatrix();

rad_1 = 1.15*rad;
for(i=1;i<5;i++){
glPushMatrix();
glTranslatef(2*XCTR + rad_1*cos(12.25*i*.02*PI+PI/2), 2*YCTR - rad_1*sin(12.25*i*.02*PI+PI/2),-10);
glRotatef(-(12.25*i*.02*PI)*57.3, 0.0, 0.0, 1.0);
glBegin(GL_LINE_STRIP);
        glVertex3fv(ytik[0]);
        glVertex3fv(ytik[1]);
glEnd();
glPopMatrix();
```

```
    }

}

void DrawDisplay::draw_coll(void)
{
        bool blink;
        float XCTR, YCTR, move_x, move_y, tq_x, k1,k2;
        float tik[3][2] = {{-.05, 0. },
                                        {.05, 0.},
                                        {.0 , 0.}};
        float tik_maj[2][2] = {{-.05, 0. },
                                        {.05 , 0.}};

        float tik_out_rt[3][2] = {{-.1, -.03},
                                                {-.1, .03},
                                                { .1, 0.}};

        float tik_out_lt[3][2] = {{.1, -.03},
                                                {.1, .03},
                                                { -.1, 0}};
        float rad = .3;
        int i, r_cnt, r_1=0;
        float   ztr, xsc, ysc, fact;
        float   rgbf[4] = {1,1,1,.7};
        char  text1[80];
        float kx,ky,kz;
        float   coll_lim;
        float   MAX_PULL;

        MAX_PULL = max_pull();

//      coll_per = 100.0*(COLL_NOW - COLL_MIN)/0.35;  DCL and ENB on 7/1/04
//      coll_lim = 100.0*(MAX_PULL - COLL_MIN)/0.35;

        coll_per = 100.0*(COLL_NOW)/0.35;
        coll_lim = 100.0*(MAX_PULL)/0.35;

        st_mrg = (int)100.0*(coll_lim - coll_per)/coll_lim;
        nr = (int) state[O]/(OMG_NOM*ofct);

        glMatrixMode(GL_PROJECTION);
        glPushMatrix();

        glLoadIdentity();
        gluOrtho2D(0.0, XMAXSCREEN, 0.0, YMAXSCREEN);
        glMatrixMode(GL_MODELVIEW);

        glBindTexture(GL_TEXTURE_2D, texture[0]);

        fact = .01;
        ztr = -10;
        xsc = 1.f*fact; ysc = 1.f*fact;
        strcpy(text1, "COL");

        kx = .3;
        ky = .3;
        kz = .3;
        move_x = 0;
        move_y = 0;
```

```
if(view == CHASEPLANE)XCTR = -1.3;
else if(view == OUT_THE_WINDOW)XCTR = -.65;
YCTR = 0.;
tq_x = 0;
if(coll_per < 100 && coll_per >= 10)tq_x = .1;
if(coll_per < 10)tq_x = .2;
tq_x = 0;
r_cnt = (50 + r_1);
glColor3f(.80, .80, .80);
blink = false;

k1 = .9;
k2 = .02;
glPushMatrix();
glColor4f(1., .3, .3,1.);
glScalef(kx,ky,kz);
glTranslatef(XCTR+.3,YCTR-k2*coll_per,0);

// collective scale

for(i=0;i<101;i=i+100){
        glColor3f(.85, 1., 1.);
        tik[0][0] = -.02;
        tik[1][0] = .02;
        tik[0][1] = (k2*i);
        tik[1][1] = (k2*i);
        glPushMatrix();
        glLineWidth(5);
        glBegin(GL_LINES);
        glVertex3fv(tik[0]);
        glVertex3fv(tik[1]);
        glEnd();
        glPopMatrix();
        }

 glColor4f(.85, 1., 1.,.1);
 glTranslatef(-.01,0,0);
        tik[0][0] = 0.;
        tik[1][0] = 0.;
        tik[0][1] = (k2*0);
        tik[1][1] = (k2*100);
        glPushMatrix();
        glLineWidth(5);
        glBegin(GL_LINES);
        glVertex3fv(tik[0]);
        glVertex3fv(tik[1]);
        glEnd();
        glPopMatrix();
 glPopMatrix();

// draw collective pointer

 glPushMatrix();
 glScalef(kx,ky,kz);
 glTranslatef(XCTR+.52,YCTR+0,0);
 glLineWidth(1);

 if(st_mrg < 5){
                if(blink_one)glColor4f(1.,.2,.2,1.);
                if(blink_two)glColor4f(0., 1., 1.,1.);
```

```
                glBegin(GL_TRIANGLES);
                glVertex3fv(tik_out_lt[0]);
                glVertex3fv(tik_out_lt[1]);
                glVertex3fv(tik_out_lt[2]);
                glEnd();
    }
    else if(st_mrg >= 5){
            glColor4f(.8,.8,1.,.9);
            if(!cmd_flag)glBegin(GL_TRIANGLES);
            if(cmd_flag)glBegin(GL_LINE_LOOP);
            glVertex3fv(tik_out_lt[0]);
            glVertex3fv(tik_out_lt[1]);
            glVertex3fv(tik_out_lt[2]);
            glEnd();
    }

    glPopMatrix();


    // draw stall limit

    glPushMatrix();
    glScalef(kx,ky,kz);
    glLineWidth(3);
    glColor4f(1., .3, .3,1.);
    glTranslatef(XCTR+.33,YCTR+k2*(coll_lim - coll_per),0);
    glBegin(GL_LINES);;
    glVertex3fv(tik_maj[0]);
    glVertex3fv(tik_maj[1]);
    glEnd();
    glPopMatrix();

// draw pitch pointer

    glLineWidth(1);
    glColor4f(.8,.8,1.,.9);
    glPushMatrix();
    glScalef(kx,ky,kz);
    glTranslatef(XCTR+.77,YCTR+0,0);

    glColor4f(.8,.8,1.,.9);
    glBegin(GL_LINE_LOOP);
    glVertex3fv(tik_out_rt[0]);
    glVertex3fv(tik_out_rt[1]);
    glVertex3fv(tik_out_rt[2]);
    glEnd();

    glPopMatrix();

    /*
    glScalef(kx,ky,kz);
    glTranslatef(XCTR+1.6,YCTR-1*sin(ALP_CMD),0);

    // pitch scale

    for(i=-40;i<41;i=i+40){
            glColor3f(.85, 1., 1.);
            tik[0][1] = (k2*i);
            tik[1][1] = (k2*i);
            glPushMatrix();
```

```
            glLineWidth(1);
            glBegin(GL_LINES);
            glVertex3fv(tik[0]);
            glVertex3fv(tik[1]);
            glEnd();
            glPopMatrix();
            }
        glPopMatrix();
*/
        glMatrixMode(GL_PROJECTION);
        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glLineWidth(2);


}

float DrawDisplay::max_pull(void) {

        double u10, u20, x1, x2, tstx;
        double fi0, fg1,f, toler, dfdx, trm1, trm2,.trm3,trm4,trm5;
        double nu_norm1, nu_norm, c2w, lam0, mu0, fg2, fi2;
        double ct_max_non;
    int its,icon,iters;
        float MAX_PULL;

        ct_max_non = CT_MAX*cfct;


        u10 = cos(ALP_NOW) * ct_max_non;
        u20 = sin(ALP_NOW) * ct_max_non;


        trm1 = u10*u10 + u20*u20;
        trm2 = pow(trm1,.75);

        x1 = p0*(state[U]*u20 - state[W]*u10)/((state[O]/100.0)*trm2);
        x2 = p0*(state[U]*u10 + state[W]*u20)/((state[O]/100.0)*trm2);

        tstx = (2*x1 + 3)*(2*x1 + 3) + x2*x2;

    if(tstx >= 1)
        {
       fi0  = 1;
       fi2  = fi0;
       toler = 0.001;
       its  = 100;
       iters = 0;
       icon  = 1;
       while(icon != 0 && iters<its)
              {
          f = pow(fi2,4) + 2*x1*pow(fi2,3) + (x1*x1 + x2*x2)*fi2*fi2-1;
          dfdx = 4*pow(fi2,3) + 6*x1*fi2*fi2 + 2*(x1*x1 + x2*x2)*fi2;
          fi2 = fi2 - f/dfdx;
          icon = 1;
          if (fabs(fi2-fi0)/fabs(fi2) < toler)icon = 0;
          fi0 = fi2;
                    iters = iters + 1;
       }
```

```
}

    if(tstx < 1)   fi2 = x1*(0.373*x1*x1 + 0.598*x2*x2 - 1.991);

    nu_norm1 = k0*fi2*sqrt(ct_max_non)*(state[O]/100.0)/0.01;
    trm1 = -state[W]*ct_max_non + nu_norm1*u10;
    trm2 =  state[U]*ct_max_non + nu_norm1*u20;
    trm3 = 4*((state[ALT]) / hfct + hub);
    c2w = trm1*trm1/(trm1*trm1 + trm2*trm2);
    fg1 = 1 - r*r*c2w/(trm3*trm3);

    nu_norm = k0*fi2*fg1*sqrt(ct_max_non)*(state[O]/100.0)/0.01;
    trm1 = -state[W]*ct_max_non + nu_norm1*u10;
    trm2 =  state[U]*ct_max_non + nu_norm1*u20;
    trm3 = 4*((state[ALT]) / hfct + hub);
    c2w = trm1*trm1/(trm1*trm1 + trm2*trm2);
    fg2 = 1 - r*r*c2w/(trm3*trm3);


    lam0 = 0.01*(state[U]*u20 - state[W]*u10)/((state[O]/100.0)*ct_max_non) + k0*fi2*fg2*sqrt(ct_max_non);
    mu0  = 0.01*(state[U]*u10 + state[W]*u20)/((state[O]/100.0)*ct_max_non);

    trm1   = mu0;
    trm2   = trm1*trm1;
    trm3   = (trm2*1.5+1)*(ct_max_non*6/(cla*sig*cfct));
    trm4   = lam0*1.5*(1-trm2*0.5);
    trm5   = (1-trm2+trm2*trm2*2.25);
    MAX_PULL = (trm3 + trm4)/trm5;

    return MAX_PULL;
}

void DrawDisplay::draw_alt(void)
{
    bool blink;
    float XCTR, YCTR, move_x, move_y, tq_x, alt, k1,k2,ialt,i_cnt;
    float tik[3][2] = {{-.005, 0.},
                       {.005, 0.},
                       {.0 , 0.}};
    float tik_maj[3][2] = {{-.15, 0. },
                       {.15, 0.},
                       {.0 , 0.}};
    float tmp[3][2]= {{-.05, 0.},
                       {.05, 0.},
                       {.0 , 0.}};

    float tik_out_rt[3][2] = {{-.2, -.05},
                              {-.2, .05},
                              { .2, 0.}};

    float tik_out_lt[3][2] = {{.2, -.07},
                              {.2, .0},
                              { -.2, 0}};
    float rad = .3;
    int i, r_cnt, r_1=0,num;
    float  ztr, xsc, ysc, fact;
    float  rgbf[4] = {1,1,1,.7};
    char  text1[80];

    glMatrixMode(GL_PROJECTION);
```

```
glPushMatrix();

glLoadIdentity();
gluOrtho2D(0.0, XMAXSCREEN, 0.0, YMAXSCREEN);
glMatrixMode(GL_MODELVIEW);

glBindTexture(GL_TEXTURE_2D, texture[0]);
fact = .01;
ztr = -10;
xsc = 1.f*fact; ysc = 1.f*fact;
strcpy(text1, "SPD");

alt = state[ALT]/hfct;

move_x = 0;
move_y = .08;
XCTR = 0.+move_x;
YCTR = .1+move_y;
tq_x = 0;
if(alt < 100 && alt >= 10)tq_x = .1;
if(alt < 10)tq_x = .2;
tq_x = 0;
r_cnt = (50 + r_1);
glColor3f(.80, .80, .80);
blink = false;
glPushMatrix();
glTranslatef(XCTR,YCTR,0.);
ialt = alt_beg;
num = 4;
if(alt < alt_beg){
        ialt = 50;
        num = 3;
}
if(alt < 50){
        ialt = 25;
        num = 2;
}
if(alt < 25){
        ialt = 12.5;
        num = 1;
}
if(alt < 12.5){
        ialt = 12.5;
        num = 0;
}

k1 = .9;
k2 = .0065/2;
glLineWidth(1);
i_cnt = num+1;
for(i=0;i<num;i++){
        glColor3f(.85, 1., 1.);
        tik[0][0] = -.05/i_cnt;
        tik[1][0] = .05/i_cnt;
        tik[0][1] = (k2*ialt);
        tik[1][1] = (k2*ialt);
        ialt = ialt/2;
        glPushMatrix();
        glBegin(GL_LINES);
        glVertex2fv(tik[0]);
```

```
            glVertex2fv(tik[1]);
            glEnd();
            glPopMatrix();
            i_cnt --;
            }
            tik[0][0] = -.05/i_cnt;
            tik[1][0] = .05/i_cnt;
            tik[0][1] = 0;
            tik[1][1] = 0;
            glLineWidth(2);
            glColor3f(1., 1., .2);
            glBegin(GL_LINES);
            glVertex2fv(tik[0]);
            glVertex2fv(tik[1]);
            glEnd();

      glPopMatrix();

glMatrixMode(GL_PROJECTION);
      glPopMatrix();
      glMatrixMode(GL_MODELVIEW);
}

void DrawDisplay::draw_bug(void)
{
      bool blink;
      float XCTR, YCTR, move_x, move_y, tq_x, alt, k1,k2,k4,talt,vspd,hspd;
      float kw2;
      float bug[6][3] = {{.4, 0., 0. },
                                    {.26, .13, 0.},
                                    {.13, .13, 0.},
                                    {.0, 0., 0.},
                                    {-.6, 0., 0.},
                                    {-.62, 0.05, 0.}};
      float vtik[2][3] = {{0., .025, 0. },
                                    {0., -.025, 0.}};
      float htik[2][3] = {{-.035, .0, 0. },
                                    {.035, .0, 0.}};
      float utik[2][3] = {{0., 0., 0. },
                                    {0., 0., 0.}};
      float dtik[2][3] = {{.0, 0., 0. },
                                    {.0, 0., 0.}};
      float tmp[3][3]= {{-.05, 0., 0. },
                                    {.05, 0., 0.},
                                    {.0 , 0., 0. }};

      float tik_out_rt[3][3] = {{-.2, -.05, 0. },
                                          {-.2, .05, 0. },
                                          { .2, 0, 0. }};

      float tik_out_lt[3][3] = {{.2, -.07, 0. },
                                          {.2, .0, 0. },
                                          { -.2, 0, 0. }};
      float rad = .3;
      double p_trm;
      int i, r_cnt, r_1=0, ispd;
      float ztr, xsc, ysc, fact,scl,tq_s,tq_v;
      float xbias, tail,v_inc;
      float  rgbf[4] = {1,1,1,.7};
```

```
char  text1[80];

tail = .4;

glBindTexture(GL_TEXTURE_2D, texture[0]);
fact = .01;
ztr = -10;
xsc = 1.f*fact; ysc = 1.f*fact;
strcpy(text1, "SPD");

hspd = state[U]/ufct/1.69;
alt  = state[ALT]/hfct;
vspd = state[W]/wfct;

move_x = 0;
move_y = .08;
XCTR = -.45;
YCTR = .8;
tq_x = 0;
if(alt < 100 && alt >= 10)tq_x = .1;
if(alt < 10)tq_x = .2;
tq_s = 0;
tq_v = 0;
if(vspd < 100 && vspd >= 10)tq_v = .1;
if(vspd < 10)tq_v = .1;
r_cnt = (50 + r_1);
blink = false;
k1=.1;
k2 = .014;
kw2 = 1.*k2;
k4 = .02;
xbias = .45;
scl = 1.0;
if(alt > alt_beg)talt = alt_beg + 1;
if(alt <= alt_beg)talt = alt;

glLineWidth(2);
glPushMatrix();
glTranslatef(XCTR+xbias,YCTR+k2*talt,-10.);
glRotatef(state[THETA]*57.3, 0.0, 0.0, 1.0);
glScalef(scl, scl, scl);
glColor3f(1., 1., 1.);
//  if(inform == 0 || inform == 4 || inform == 9)glColor3f(1., .6, 0.);
if(inform == 0 || inform == 4) glColor3f(1., .6, 0.);
glBegin(GL_LINES);
glVertex3fv(bug[0]);
glVertex3fv(bug[1]);
glEnd();
glBegin(GL_LINES);
glVertex3fv(bug[1]);
glVertex3fv(bug[2]);
glEnd();
glBegin(GL_LINES);
glVertex3fv(bug[2]);
glVertex3fv(bug[3]);
glEnd();
glBegin(GL_LINES);
glVertex3fv(bug[3]);
glVertex3fv(bug[4]);
glEnd();
```

```
        glBegin(GL_LINES);
        glVertex3fv(bug[4]);
        glVertex3fv(bug[5]);
        glEnd();
        glPopMatrix();

//      draw fwd velocity vector

        utik[0][0] += bug[0][0];
        utik[1][0] += k2*hspd*1.69 + bug[0][0];
        glLineWidth(1.5);
        glColor4f(.8,.8,1.,1);
        glPushMatrix();
        glTranslatef(XCTR+xbias,YCTR+k2*talt,-10.);
        glRotatef(state[THETA]*57.3, 0.0, 0.0, 1.0);
        glBegin(GL_LINES);
        glVertex3fv(utik[0]);
        glVertex3fv(utik[1]);
        glEnd();
        glPopMatrix();

//      ticks



        glPushMatrix();
        glLineWidth(2);
        glTranslatef(XCTR+xbias,YCTR+k2*talt,-10.);
        glRotatef(state[THETA]*57.3, 0.0, 0.0, 1.0);
        ispd = floor(hspd/10)+1;
        for(i=1;i<ispd;i++){
                glColor3f(1.,1.,1.);
                vtik[0][0] = k2*i*16.9 + bug[0][0];
                vtik[1][0] = k2*i*16.9 + bug[0][0];
        glBegin(GL_LINES);
        glVertex3fv(vtik[0]);
        glVertex3fv(vtik[1]);
        glEnd();
        }
        glPopMatrix();
        glLineWidth(2);
        p_trm = 1.3*k2*hspd*1.69*sin(state[THETA]);
        glPrint(blink,XCTR+.9+tq_s +
 k2*hspd*1.69,YCTR+.15+k2*talt+p_trm,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],rgbf[3],"%d",(int)hspd);

        //      print out alitude box

        glPushMatrix();
        dtik[0][0] = cos(state[THETA])*bug[4][0];
        dtik[0][1] = bug[4][1]-     (tail)*sin(state[THETA]);
        dtik[1][0] = cos(state[THETA])*bug[4][0];
        dtik[1][1] = bug[4][1]-     (tail)*sin(state[THETA]);
        if(alt > alt_beg)talt = alt_beg + 1;
        if(alt <= alt_beg)talt = alt;

        glPrint(blink,XCTR+.0+tq_x,1.2+k2*talt,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],rgbf[3],"%d",(int)alt);
        glTranslatef(XCTR+xbias,YCTR+k2*talt,-10.);
        glScalef(scl, scl, scl);
        glColor4f(.8,.8,1.,.9);
        glLineWidth(1.5);
```

```
        dtik[1][1] -= kw2*vspd;

        //      draw sink rate vector

        glBegin(GL_LINES);
        glVertex3fv(dtik[0]);
        glVertex3fv(dtik[1]);
        glEnd();
        glPopMatrix();

//      ticks

        htik[0][0] += cos(state[THETA])*bug[4][0];
        htik[1][0] += cos(state[THETA])*bug[4][0];
        glPushMatrix();
        glLineWidth(2);
        glTranslatef(XCTR+xbias,YCTR+k2*talt,-10.);
        v_inc = 5.;
        ispd = floor(vspd/v_inc)+1;
        for(i=1;i<ispd;i++){
                glColor4f(1.,1.,1.,1.);
                htik[0][1] = -kw2*i*v_inc+cos(state[THETA])*bug[4][1]-tail*sin(state[THETA]);
                htik[1][1] = -kw2*i*v_inc+cos(state[THETA])*bug[4][1]-tail*sin(state[THETA]);
        glBegin(GL_LINES);
        glVertex3fv(htik[0]);
        glVertex3fv(htik[1]);
        glEnd();
        }
        glPopMatrix();
        glLineWidth(1);
}

void DrawDisplay::draw_rpm(void) {
        _
        bool blink;
        float XCTR, YCTR, move_x, move_y, rpm_x, rad_1,talt,k2,alt;
        float tik[3][3] = {{-.07, .1, 0. },
                                        {0, 0, 0. },
                                        {.07, .1, 0. }};
        float ytik[2][3] = {{0, .02, 0. },
                                        {0, -.02, 0. }};
        float rad = .3;
        int i, r_cnt, r_1,r_lim;
        float  xtr, ytr, ztr, xsc, ysc, fact;
        float  rgbf[4] = {1.,1.,1,.7};
        char  text1[80];

        fact = .01;
        xtr = -520.0*fact; ytr = -380.0*fact; ztr = -10;
        xsc = 1.f*fact; ysc = 1.f*fact;
        strcpy(text1, "RPM");

        r_1 = 1.67*(nr - 100);
        k2 = .013;
        alt = state[ALT]/hfct;
        move_x = 0;
        move_y = .07;
        if(alt > alt_beg)talt = alt_beg + 1;
        if(alt <= alt_beg)talt = alt;
        XCTR = .4 - .2*sin(state[THETA]);
```

```
        YCTR = k2*talt+1.5 + .2*sin(state[THETA]);
        rpm_x = 0;
        if(nr < 100 && nr >= 10)rpm_x = .1;
        if(nr < 10)rpm_x = .2;

        glColor3f(1.,1.,1.);
        blink = false;
        r_cnt = (50 + r_1);
        if(nr < 90 || nr > 110 )blink = true;
        if(alt < 30.)blink = false;
        glPrint(blink,XCTR+rpm_x-.2,YCTR+.1,ztr,xsc,ysc,1,rgbf[0],rgbf[1],rgbf[2],rgbf[3],"%d",nr);
        glPushMatrix();
        glTranslatef(0,0,-10.);
        glLineWidth(2);
        glBegin(GL_LINE_STRIP);
        for (i = r_cnt; i > -1; i--)glVertex2f(XCTR + rad*cos(i*.02*PI+PI/2), YCTR - rad*sin(i*.02*PI+PI/2));
        glEnd();
        glPopMatrix();

        rad_1 = 1.15*rad;
        r_lim = 4;
        if(nr > 99)r_lim = 5;
        if(nr > 109)r_lim = 6;
        for(i=1;i<r_lim;i++){
        glPushMatrix();
        glTranslatef(XCTR + rad_1*cos(16.67*i*.02*PI+PI/2), YCTR - rad_1*sin(16.67*i*.02*PI+PI/2),-10);
        glRotatef(-(16.67*i*.02*PI)*57.3, 0.0, 0.0, 1.0);
        glBegin(GL_LINES);
                glVertex3fv(ytik[0]);
                glVertex3fv(ytik[1]);
        glEnd();
        glPopMatrix();
        }

        glPushMatrix();
        glTranslatef(XCTR + rad*cos(r_cnt*.02*PI+PI/2), YCTR - rad*sin(r_cnt*.02*PI+PI/2),-10);
        glRotatef(-(r_cnt*.02*PI+PI/2)*57.3, 0.0, 0.0, 1.0);
        glColor3f(1.,1.,1.);
        glBegin(GL_TRIANGLES);
                glVertex3fv(tik[0]);
                glVertex3fv(tik[1]);
                glVertex3fv(tik[2]);
        glEnd();
        glPopMatrix();
}

void DrawDisplay::draw_ball(void) {
        float XCTR, YCTR, move_x, move_y, rpm_x;
        float vertik[2][3] = {{0, .15, 0. },
                                        {0, -.15, 0. }};
        float rad = .3;
        int i;
        float  ztr, xsc, ysc, fact;
        float  rgbf[4] = {1,1,1,.7};
        char  text1[80];

        glBindTexture(GL_TEXTURE_2D, texture[0]);
        fact = .01;
        ztr = -10;
        xsc = 1.f*fact; ysc = 1.f*fact;
```

```
        strcpy(text1, "RPM");

        move_x = 0;
        move_y = 0;
        XCTR = 0.+move_x;
        YCTR = -.3+move_y;
        rpm_x = 0;

        glColor3f(.80, .80, .80);

        glPushMatrix();
        glTranslatef(XCTR,YCTR,-10);
        glBegin(GL_LINE_LOOP);
        for (i = 1; i <34; i++)glVertex2f(.08*cos(i*.06*PI),.08*sin(i*.06*PI));
        glEnd();
        glPopMatrix();

        glPushMatrix();
        glTranslatef(XCTR,YCTR,-10);
        glBegin(GL_LINES);
        glVertex3fv(vertik[0]);
        glVertex3fv(vertik[1]);
        glEnd();
        glPopMatrix();
}

/*-------------------------*/
/* draw FP shadow */
/*-------------------------*/

void DrawDisplay::draw_shad(void) {
        int i, j, cnt;
        float vv[4][3];
        double x_00, x_10, y_00, y_01, z_00, z_01, z_10, z_11;
        double h_r, h_l;
        double x_proj, y_proj, z_proj;
        double t_inc,psi_up;
        double trm1,trm2,trm3;
        double x_psi, y_psi, psi_inc,ux_rel,uy_rel,vx_rel,vy_rel;
        double sep, cpsi, spsi, cphi,sphi,ctheta,stheta;
        int i_00_00,j_00_00,i_00_10,j_00_10,i_00_11,j_00_11,i_00_01,j_00_01;
        int i_10_00,j_10_00,i_10_10,j_10_10,i_10_11,j_10_11,i_10_01,j_10_01;
        int i_11_00,j_11_00,i_11_10,j_11_10,i_11_11,j_11_11,i_11_01,j_11_01;
        int i_01_00,j_01_00,i_01_10,j_01_10,i_01_11,j_01_11,i_01_01,j_01_01;
        double bx_00_00,bx_00_10,by_00_00,by_00_01,bx_10_00,bx_10_10,by_10_00,by_10_01;
        double bx_11_00,bx_11_10,by_11_00,by_11_01,bx_01_00,bx_01_10,by_01_00,by_01_01;

        sep = .5;

        spsi = sin(state[PSI]);
        cpsi = cos(state[PSI]);
        sphi = sin(state[PHI]);
        cphi = cos(state[PHI]);
        stheta = sin(state[THETA]);
        ctheta = cos(state[THETA]);

        x_proj =  state[NORTH]+(state[U]*cpsi+state[V]*spsi)*TTGO;
        y_proj =  state[EAST] +(state[U]*spsi+state[V]*cpsi)*TTGO;
        x_proj = state[NORTH];
        y_proj = state[EAST];
```

```
z_proj = state[DOWN];
psi_up = state[PSI];

t_inc = .01;

trm1 = state[U]*t_inc;
trm2 = state[V]*t_inc;
trm3 = state[W]*t_inc;

psi_inc = 0;

ux_rel = trm1*cos(psi_inc/2);
uy_rel = trm1*sin(psi_inc/2);

vx_rel = -trm2*sin(psi_inc/2);
vy_rel = trm2*cos(psi_inc/2);

for( cnt =0;cnt< floor(TTGO/t_inc);cnt ++){


        x_psi = (ux_rel+vx_rel)*cos(psi_up) + (uy_rel+vy_rel)*sin(psi_up);
        y_psi = (ux_rel+vx_rel)*sin(psi_up) + (uy_rel+vy_rel)*cos(psi_up);
        x_proj += x_psi;
        y_proj += y_psi;
        z_proj += trm3;

        i = floor(x_proj/FPB);
        j = floor(y_proj/FPB);
        psi_up += psi_inc;
        if(psi_up >= 2*PI)psi_up = 0;
        if(psi_up <= 0)psi_up = 2*PI;

}

x_00 = x_proj - TW/2.;
x_10 = x_proj + TW/2.;
y_00 = y_proj - TW/2.;
y_01 = y_proj + TW/2.;

i_00_00 = floor(x_00/FPB);
j_00_00 = floor(y_00/FPB);
i_00_10 = i_00_00 + 1;
j_00_10 = j_00_00;
i_00_11 = i_00_10;
j_00_11 = j_00_00 + 1;
i_00_01 = i_00_00;
j_00_01 = j_00_11;

i_10_00 = floor(x_10/FPB);
j_10_00 = j_00_00;
i_10_10 = i_10_00 + 1;
j_10_10 = j_10_00;
i_10_11 = i_10_10;
j_10_11 = j_10_00 + 1;
i_10_01 = i_10_00;
j_10_01 = j_10_11;

i_11_00 = i_10_00;
j_11_00 = floor(y_01/FPB);
i_11_10 = i_11_00 + 1;
```

```
j_11_10 = j_11_00;
i_11_11 = i_11_10;
j_11_11 = j_11_00 + 1;
i_11_01 = i_11_00;
j_11_01 = j_11_11;

i_01_00 = i_00_00;
j_01_00 = j_11_00;
i_01_10 = i_01_00 + 1;
j_01_10 = j_01_00;
i_01_11 = i_01_10;
j_01_11 = j_01_00 + 1;
i_01_01 = i_01_00;
j_01_01 = j_01_11;

bx_00_00 = i_00_00*FPB;
bx_00_10 = i_00_10*FPB;
by_00_00 = j_00_00*FPB;
by_00_01 = j_00_01*FPB;

bx_10_00 = i_10_00*FPB;
bx_10_10 = i_10_10*FPB;
by_10_00 = j_10_00*FPB;
by_10_01 = j_10_01*FPB;

bx_11_00 = i_11_00*FPB;
bx_11_10 = i_11_10*FPB;
by_11_00 = j_11_00*FPB;
by_11_01 = j_11_01*FPB;

bx_01_00 = i_01_00*FPB;
bx_01_10 = i_01_10*FPB;
by_01_00 = j_01_00*FPB;
by_01_01 = j_01_01*FPB;

        h_l = interp(bx_00_00, bx_00_10, x_00, gnd[i_00_00][j_00_00], gnd[i_00_10][j_00_00]);
        h_r = interp(bx_00_00, bx_00_10, x_00, gnd[i_00_00][j_00_01], gnd[i_00_10][j_00_01]);
        z_00 = interp(by_00_00, by_00_01, y_00, h_l, h_r);

        h_l = interp(bx_10_00, bx_10_10, x_10, gnd[i_10_00][j_10_00], gnd[i_10_10][j_10_00]);
        h_r = interp(bx_10_00, bx_10_10, x_10, gnd[i_10_00][j_10_01], gnd[i_10_10][j_10_01]);
        z_10 = interp(by_10_00, by_10_01, y_00, h_l, h_r);

        h_l = interp(bx_11_00, bx_11_10, x_10, gnd[i_11_00][j_11_00], gnd[i_11_10][j_11_00]);
        h_r = interp(bx_11_00, bx_11_10, x_10, gnd[i_11_00][j_11_01], gnd[i_11_10][j_11_01]);
        z_11 = interp(by_11_00, by_11_01, y_01, h_l, h_r);

        h_l = interp(bx_01_00, bx_01_10, x_00, gnd[i_01_00][j_01_00], gnd[i_01_10][j_01_00]);
        h_r = interp(bx_01_00, bx_01_10, x_00, gnd[i_01_00][j_01_01], gnd[i_01_10][j_01_01]);
        z_01 = interp(by_01_00, by_01_01, y_01, h_l, h_r);

vv[0][0] = x_00;
vv[0][1] = y_00;
vv[0][2] = z_00 - lyft;

vv[1][0] = x_10;
vv[1][1] = y_00;
vv[1][2] = z_10 - lyft;

vv[2][0] = x_10;
```

```
        vv[2][1] = y_01;
        vv[2][2] = z_11 - lyft;

        vv[3][0] = x_00;
        vv[3][1] = y_01;
        vv[3][2] = z_01 - lyft;


//  glColor3f(95., .3, 0);
  glColor3f(.85, 1, 1);

        glBegin(GL_LINES);
        {
                glVertex3fv(vv[0]);
                glVertex3fv(vv[1]);
        }
        glEnd();

        glBegin(GL_LINES);
        {
                glVertex3fv(vv[1]);
                glVertex3fv(vv[2]);
        }
        glEnd();

        glBegin(GL_LINES);
        {
                glVertex3fv(vv[2]);
                glVertex3fv(vv[3]);
        }
        glEnd();

        glBegin(GL_LINES);
        {
                glVertex3fv(vv[3]);
                glVertex3fv(vv[0]);
        }
        glEnd();

}

/*-------------------------*/
/* draw FPVector impact point */
/*-------------------------*/

void DrawDisplay::draw_imp(void) {
        int i, j, cnt;
        double x_proj, y_proj, z_proj;
        double t_inc;
        double psi_up;
        double trm1,trm2,trm3;
        double x_psi, y_psi, h_diff,psi_inc,ux_rel,uy_rel,vx_rel,vy_rel;
        int brake;

        x_proj = state[NORTH];
        y_proj = state[EAST];
        z_proj = state[DOWN];
        psi_up = state[PSI];

        t_inc = .01;
```

```
    brake = 0;

    trm1 = state[U]*t_inc;
    trm2 = state[V]*t_inc;
    trm3 = state[W]*t_inc;

    psi_inc = 0;

    ux_rel =  trm1*cos(psi_inc/2);
    uy_rel =  trm1*sin(psi_inc/2);

    vx_rel = -trm2*sin(psi_inc/2);
    vy_rel =  trm2*cos(psi_inc/2);

    h_diff = 1000;

    for( cnt =0;cnt< floor(20/t_inc);cnt ++){


          x_psi =  (ux_rel+vx_rel)*cos(psi_up) + (uy_rel+vy_rel)*sin(psi_up);
          y_psi =  (ux_rel+vx_rel)*sin(psi_up) + (uy_rel+vy_rel)*cos(psi_up);
          x_proj += x_psi;
          y_proj += y_psi;
          z_proj += trm3;

          i = floor(x_proj/FPB);
          j = floor(y_proj/FPB);
          h_diff = -(z_proj - check_imp_ht(x_proj,y_proj,i,j));
          if(h_diff < 0){
                brake = 1;
                goto a;
          }
          psi_up += psi_inc;
          if(psi_up >= 2*PI)psi_up = 0;
          if(psi_up <= 0)psi_up = 2*PI;

    }

    a:    if(brake)draw_foot(x_proj,y_proj,psi_up);
}

/*-------------------------------------------------*/
/* draw a large rectangle to make the horizon      */
/* (better would be to do this in an ortho projection) */
/*-------------------------------------------------*/

void DrawDisplay::draw_ground_plane(void) {

    /* turn off the zbuffer while we draw the ground plane */
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture[3]);

    glPushMatrix();
    glColor3f(.7,1.,.7);

    glBegin(GL_POLYGON);
    {

          glTexCoord2f(0.0f, 0.0f);glVertex3f(-50000., -50000., 20.0);
          glTexCoord2f(0.f, 1.0f);glVertex3f(50000., -50000., 20.0);
```

```
            glTexCoord2f(1.f, 1.f);glVertex3f(50000., 50000., 20.0);
            glTexCoord2f(1.0f, 0.f);glVertex3f(-50000., 50000., 20.0);

        }
        glEnd();
        glPopMatrix();

//      glEnable(GL_DEPTH_TEST);
        glDisable(GL_TEXTURE_2D);
}

void DrawDisplay::draw_plane(void)
{
        float       kf = 3.28;
        GLfloat             LightPosition[4];       // Light Position

        glPushMatrix();

        glTranslatef(state[NORTH], state[EAST], state[DOWN]);
        LightPosition[0]=state[NORTH];
        LightPosition[1]=state[EAST];
        LightPosition[2]=state[DOWN];
        LightPosition[3]=1;

        glRotatef(state[PSI]*57.3, 0.0, 0.0, 1.0);
        glRotatef(state[THETA]*57.3, 0.0, 1.0, 0.0);
        glRotatef(state[PHI]*57.3, 1.0, 0.0, 0.0);
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texture[1]);
        glPushMatrix();
        glTranslatef(-.5,0.,0.);
        draw_rotor();
        glPopMatrix();


//      transmission casing
glScalef(kf, kf, kf);

        glBegin(GL_TRIANGLE_STRIP);
        {
                glTexCoord2f(0.0f, 0.0f);glVertex3f(4.419, -0, -.789);   //1
                glTexCoord2f(0.f, 1.0f);glVertex3f(3.782, -.4, -.704);   //6
                glTexCoord2f(1.f, 0.0f);glVertex3f(3.642, -.1, -1.347);  //2
                glTexCoord2f(0.f, 0.0f);glVertex3f(3.160, -.456, -.777); //5
                glTexCoord2f(1.f, -1.f);glVertex3f(3.096, -.2, -1.408);  //3
                glTexCoord2f(0.0f, -1.f);glVertex3f(2.79,-.492,-1.006);          //4
                glTexCoord2f(0.0f, 0.0f);glVertex3f(2.355, -.2, -1.978); //42
                glTexCoord2f(0.f, 1.0f);glVertex3f(2.016, -.590, -1.128);        //43
                }
        glEnd();

        glBegin(GL_QUAD_STRIP);
        {
                glTexCoord2f(-1.f, 0.0f);glVertex3f(1.622,-.2,-2.063);           //41
                glTexCoord2f(1.0f, 0.0f);glVertex3f(2.355, -.2, -1.978); //42
                glTexCoord2f(0.f, 0.0f);glVertex3f(1.853,-.436,-1.359);          //44
                glTexCoord2f(0.f, 1.0f);glVertex3f(2.016, -.590, -1.128);        //43
                glTexCoord2f(1.f, -1.0f);glVertex3f(1.72,-.578,-1.183);          //45
                glTexCoord2f(0.0f, 1.f);glVertex3f(1.761,-.425,-.849);           //5b
                glTexCoord2f(0.0f, -1.f);glVertex3f(.725,-.501,-1.345);          //46
```

```
        glTexCoord2f(1.f, -1.0f);glVertex3f(.421,-.823,-.897);          //53b
        glTexCoord2f(1.f, 0.0f);glVertex3f(.375,-.436,-1.454);          //49
        glTexCoord2f(1.0f, -1.f);glVertex3f(-1.047,-.713,-.81);         //32b
        glTexCoord2f(0.0f, -1.f);glVertex3f(-1.993,-.501,-1.195);       //34b
        glTexCoord2f(1.f, -1.f);glVertex3f(-2.549,-.333,-.72);          //31
        glTexCoord2f(0.0f, -1.f);glVertex3f(-1.275,-.2,-1.808);  //36
        glTexCoord2f(1.f, 1.f);glVertex3f(-2.623,-.2,-1.189);    //33
        }
glEnd();

glBegin(GL_QUAD_STRIP);
{
        glTexCoord2f(0.f, 1.0f);glVertex3f(1.853,-.436,-1.359);         //44
        glTexCoord2f(1.f, 1.0f);glVertex3f(1.72,-.578,-1.183);          //45
        glTexCoord2f(1.f, 0.0f);glVertex3f(1.622,-.2,-2.063);           //41
        glTexCoord2f(0.0f, -1.f);glVertex3f(.725,-.501,-1.345);         //46
        glTexCoord2f(1.f, -1.f);glVertex3f(.814,-.2,-2.063);            //40
        glTexCoord2f(1.f, 0.0f);glVertex3f(.668,-.467,-1.633);          //47
        glTexCoord2f(1.f, 1.0f);glVertex3f(.389,-.2,-2.366);            //39
        glTexCoord2f(1.f, 0.0f);glVertex3f(-.293,-.272,-1.931);         //50b
        }
glEnd();

glBegin(GL_TRIANGLE_FAN);
{
        glTexCoord2f(1.f, 0.0f);glVertex3f(-.293,-.272,-1.931);         //50b
        glTexCoord2f(1.f, 1.0f);glVertex3f(.389,-.2,-2.366);            //39
        glTexCoord2f(0.0f, -1.f);glVertex3f(0.,-.2,-2.338);      //39b
        glTexCoord2f(1.f, 0.0f);glVertex3f(-.6192,-.2,-2.293);   //38
        glTexCoord2f(1.f, -1.f);glVertex3f(-1.093,-.2,-2.16);           //37
        glTexCoord2f(0.0f, -1.f);glVertex3f(-1.275,-.2,-1.808);  //36
}
glEnd();

glBegin(GL_TRIANGLE_FAN);
{
        glTexCoord2f(1.f, 0.0f);glVertex3f(-.47,-.46,-1.685);           //50
        glTexCoord2f(0.0f, -1.f);glVertex3f(-1.275,-.2,-1.808);  //36
        glTexCoord2f(0.0f, 1.f);glVertex3f(-1.993,-.501,-1.195);//34b
        glTexCoord2f(1.f, 0.0f);glVertex3f(.375,-.436,-1.454);          //49
        glTexCoord2f(1.f, 1.0f);glVertex3f(.668,-.467,-1.633);          //47
        glTexCoord2f(1.f, 0.0f);glVertex3f(-.293,-.272,-1.931);         //50b
        glTexCoord2f(0.0f, -1.f);glVertex3f(-1.275,-.2,-1.808);  //36
}
glEnd();


glBegin(GL_TRIANGLES);
{
        glTexCoord2f(1.f, 0.0f);glVertex3f(.375,-.436,-1.454);          //49
        glTexCoord2f(1.f, 0.0f);glVertex3f(.668,-.467,-1.633);          //47
        glTexCoord2f(0.0f, -1.f);glVertex3f(.725,-.501,-1.345);         //46
}
glEnd();

glBegin(GL_TRIANGLE_FAN);
{
        glTexCoord2f(0.f, 1.0f);glVertex3f(3.654,-.0,-.4);       //7b
        glTexCoord2f(0.0f, 0.0f);glVertex3f(4.419, -.0, -.789);  //1
        glTexCoord2f(0.f, 1.0f);glVertex3f(3.782, -.4, -.704);   //6
```

```
        glTexCoord2f(0.f, 0.0f);glVertex3f(3.160, -.456, -.777); //5
        glTexCoord2f(0.0f, -1.f);glVertex3f(2.79,-.492,-1.006);        //4
        glTexCoord2f(0.f, 1.0f);glVertex3f(2.016, -.590, -1.128);      //43
        glTexCoord2f(0.0f, -1.f);glVertex3f(1.761,-.425,-.849);        //5b
        glTexCoord2f(1.f, 0.0f);glVertex3f(1.53,-.0,-.28);             //9
        glTexCoord2f(0.f, -1.0f);glVertex3f(1.785,-.0,-.134);          //8
        glTexCoord2f(0.f, 1.0f);glVertex3f(3.557,-.0,-.109);           //7
}
glEnd();

glBegin(GL_QUAD_STRIP);
{
        glTexCoord2f(1.f, 0.0f);glVertex3f(1.53,-.0,-.28);             //9
        glTexCoord2f(0.0f, -1.f);glVertex3f(1.761,-.425,-.849);        //5b
        glTexCoord2f(1.f, -1.f);glVertex3f(.218,-.0,-.243);      //12
        glTexCoord2f(1.f, 0.0f);glVertex3f(.421,-.823,-.897);          //53b
        glTexCoord2f(0.0f, 1.f);glVertex3f(-1.41,-0,-.182);            //14
        glTexCoord2f(0.0f, -1.f);glVertex3f(-1.047,-.713,-.81);        //32b
        glTexCoord2f(1.f, 1.f);glVertex3f(-3.861,-.0,-.206);           //15
        glTexCoord2f(1.f, -1.f);glVertex3f(-2.549,-.333,-.72);         //31
        }
glEnd();

glBegin(GL_TRIANGLE_FAN);
{
        glTexCoord2f(0.0f, -1.f);glVertex3f(-4.99,-.15,-.631);         //28
        glTexCoord2f(1.f, -1.f);glVertex3f(-3.861,-.0,-.206);          //15
        glTexCoord2f(1.f, 1.f);glVertex3f(-2.549,-.333,-.72);          //31
        glTexCoord2f(1.f, -1.f);glVertex3f(-2.623,-.0,-1.189);   //33
        glTexCoord2f(1.f, 1.0f);glVertex3f(-4.48,-.0,-1.007);          //26
        glTexCoord2f(1.f, 0.0f);glVertex3f(-5.184,-.0,-1.529);   //25
        glTexCoord2f(1.f, -1.f);glVertex3f(-6.083,-.0,-2.451);         //24
        glTexCoord2f(0.0f, -1.f);glVertex3f(-6.71,-.0,-2.342);         //23
        glTexCoord2f(1.f, 0.0f);glVertex3f(-6.289,-.0,-1.396);         //21
        glTexCoord2f(0.f, 1.0f);glVertex3f(-6.508,-.0,-1.03);          //20
        glTexCoord2f(0.0f, 0.0f);glVertex3f(-6.70,-.0,-.558);          //19
        glTexCoord2f(0.0f, -1.f);glVertex3f(-6.483,-.0,-.328);         //18
        glTexCoord2f(-1.f, 1.0f);glVertex3f(-5.804,-.0,-.012);         //17
        glTexCoord2f(1.f, 0.0f);glVertex3f(-5.,-.0,-.036);             //16
        glTexCoord2f(1.f, -1.f);glVertex3f(-3.861,-0,-.206);           //15
}
glEnd();

// Right Side

glBegin(GL_TRIANGLE_STRIP);
{
        glTexCoord2f(0.0f, 0.0f);glVertex3f(4.419, .0, -.789);   //1
        glTexCoord2f(0.f, 1.0f);glVertex3f(3.782, .4, -.704);    //6
        glTexCoord2f(1.f, 0.0f);glVertex3f(3.642, .1, -1.347);         //2
        glTexCoord2f(0.f, 0.0f);glVertex3f(3.160, .456, -.777);  //5
        glTexCoord2f(1.f, -1.f);glVertex3f(3.096, .15, -1.408);  //3
        glTexCoord2f(0.0f, -1.f);glVertex3f(2.79,.492,-1.006);         //4
        glTexCoord2f(0.0f, 0.0f);glVertex3f(2.355, .235, -1.978);      //42
        glTexCoord2f(0.f, 1.0f);glVertex3f(2.016, .590, -1.128);//43
        }
glEnd();

glBegin(GL_QUAD_STRIP);
{
```

```
        glTexCoord2f(1.f, 0.0f);glVertex3f(1.622,.2,-2.063);          //41
        glTexCoord2f(0.0f, 0.0f);glVertex3f(2.355, .2, -1.978);  //42
        glTexCoord2f(0.f, 1.0f);glVertex3f(1.853,.436,-1.359);        //44
        glTexCoord2f(0.f, -1.0f);glVertex3f(2.016, .590, -1.128);     //43
        glTexCoord2f(1.f, 1.0f);glVertex3f(1.72,.578,-1.183);         //45
        glTexCoord2f(0.0f, 1.f);glVertex3f(1.761,.425,-.849);         //5b
        glTexCoord2f(0.0f, -1.f);glVertex3f(.725,.501,-1.345);        //46
        glTexCoord2f(1.f, 0.0f);glVertex3f(.421,.823,-.897);          //53b
        glTexCoord2f(-1.f, 0.0f);glVertex3f(.375,.436,-1.454);        //49
        glTexCoord2f(0.0f, -1.f);glVertex3f(-1.047,.713,-.81);        //32b
        glTexCoord2f(0.0f, 1.f);glVertex3f(-1.993,.501,-1.195); //34b
        glTexCoord2f(1.f, -1.f);glVertex3f(-2.549,.333,-.72);         //31
        glTexCoord2f(0.0f, 0.f);glVertex3f(-1.275,.2,-1.808);    //36
        glTexCoord2f(1.f, -1.f);glVertex3f(-2.623,.2,-1.189);    //33
        }
glEnd();

glBegin(GL_QUAD_STRIP);
{
        glTexCoord2f(0.f, 1.0f);glVertex3f(1.853,.436,-1.359);        //44
        glTexCoord2f(1.f, 1.0f);glVertex3f(1.72,.578,-1.183);         //45
        glTexCoord2f(1.f, 0.0f);glVertex3f(1.622,.2,-2.063);          //41
        glTexCoord2f(0.0f, -1.f);glVertex3f(.725,.501,-1.345);        //46
        glTexCoord2f(1.f, -1.f);glVertex3f(.814,.2,-2.063);           //40
        glTexCoord2f(1.f, 0.0f);glVertex3f(.668,.467,-1.633);         //47
        glTexCoord2f(1.f, 1.0f);glVertex3f(.389,.2,-2.366);           //39
        glTexCoord2f(1.f, 0.0f);glVertex3f(-.293,.272,-1.931);        //50b
        }
glEnd();

glBegin(GL_TRIANGLE_FAN);
{
        glTexCoord2f(1.f, 0.0f);glVertex3f(-.293,.272,-1.931);        //50b
        glTexCoord2f(1.f, 1.0f);glVertex3f(.389,.2,-2.366);           //39
        glTexCoord2f(0.0f, -1.f);glVertex3f(0.,.2,-2.338);       //39b
        glTexCoord2f(1.f, 0.0f);glVertex3f(-.6192,.2,-2.293);    //38
        glTexCoord2f(1.f, -1.f);glVertex3f(-1.093,.2,-2.16);          //37
        glTexCoord2f(0.0f, -1.f);glVertex3f(-1.275,.2,-1.808);   //36
}
glEnd();

glBegin(GL_TRIANGLE_FAN);
{
        glTexCoord2f(1.f, 0.0f);glVertex3f(-.47,.46,-1.685);          //50
        glTexCoord2f(0.0f, 1.f);glVertex3f(-1.275,.2,-1.808);    //36
        glTexCoord2f(0.0f, -1.f);glVertex3f(-1.993,.501,-1.195);//34b
        glTexCoord2f(1.f, -1.0f);glVertex3f(.375,.436,-1.454);        //49
        glTexCoord2f(1.f, 0.0f);glVertex3f(.668,.467,-1.633);         //47
        glTexCoord2f(1.f, 1.0f);glVertex3f(-.293,.272,-1.931);        //50b
        glTexCoord2f(0.0f, -1.f);glVertex3f(-1.275,.2,-1.808);   //36
}
glEnd();

glBegin(GL_TRIANGLES);
{
        glTexCoord2f(1.f, 0.0f);glVertex3f(.375,.436,-1.454);         //49
        glTexCoord2f(1.f, -1.0f);glVertex3f(.668,.467,-1.633);        //47
        glTexCoord2f(0.0f, -1.f);glVertex3f(.725,.501,-1.345);        //46
}
glEnd();
```

```
glBegin(GL_TRIANGLE_FAN);
{
        glTexCoord2f(0.f, 1.0f);glVertex3f(3.654,0,-.4);          //7b
        glTexCoord2f(0.0f, 0.0f);glVertex3f(4.419, 0., -.789);   //1
        glTexCoord2f(0.f, 1.0f);glVertex3f(3.782, .4, -.704);    //6
        glTexCoord2f(0.f, 0.0f);glVertex3f(3.160, .456, -.777); //5
        glTexCoord2f(0.0f, -1.f);glVertex3f(2.79,.492,-1.006);          //4
        glTexCoord2f(0.f, 1.0f);glVertex3f(2.016, .590, -1.128);//43
        glTexCoord2f(0.0f, -1.f);glVertex3f(1.761,.425,-.849);          //5b
        glTexCoord2f(1.f, 0.0f);glVertex3f(1.53,.0,-.28);               //9
        glTexCoord2f(0.f, 1.0f);glVertex3f(1.785,0,-.134);              //8
        glTexCoord2f(0.f, 0.0f);glVertex3f(3.557,0,-.109);              //7
}
glEnd();

glBegin(GL_QUAD_STRIP);
{
        glTexCoord2f(1.f, 0.0f);glVertex3f(1.53,0,-.28);                //9
        glTexCoord2f(0.0f, -1.f);glVertex3f(1.761,.425,-.849);          //5b
        glTexCoord2f(1.f, -1.f);glVertex3f(.218,0,-.243);         //12
        glTexCoord2f(1.f, 0.0f);glVertex3f(.421,.823,-.897);            //53b
        glTexCoord2f(0.0f, -1.f);glVertex3f(-1.41,0,-.182);                  //14
        glTexCoord2f(0.0f, 1.f);glVertex3f(-1.047,.713,-.81);           //32b
        glTexCoord2f(1.f, -1.f);glVertex3f(-3.861,0,-.206);             //15
        glTexCoord2f(1.f, 0.f);glVertex3f(-2.549,.333,-.72);            //31
        }
glEnd();

glBegin(GL_TRIANGLE_FAN);
{
        glTexCoord2f(0.0f, -1.f);glVertex3f(-4.99,.15,-.631);           //28
        glTexCoord2f(1.f, -1.f);glVertex3f(-3.861,.0,-.206);            //15
        glTexCoord2f(1.f, 1.f);glVertex3f(-2.549,.333,-.72);            //31
        glTexCoord2f(1.f, -1.f);glVertex3f(-2.623,0,-1.189);            //33
        glTexCoord2f(1.f, 1.0f);glVertex3f(-4.48,.0,-1.007);            //26
        glTexCoord2f(1.f, 0.0f);glVertex3f(-5.184,.0,-1.529);           //25
        glTexCoord2f(1.f, -1.f);glVertex3f(-6.083,.0,-2.451);           //24
        glTexCoord2f(0.0f, -1.f);glVertex3f(-6.71,.0,-2.342);           //23
        glTexCoord2f(1.f, 0.0f);glVertex3f(-6.289,.0,-1.396);           //21
        glTexCoord2f(0.f, 1.0f);glVertex3f(-6.508,.0,-1.03);            //20
        glTexCoord2f(0.0f, 0.0f);glVertex3f(-6.70,.0,-.558);            //19
        glTexCoord2f(0.0f, -1.f);glVertex3f(-6.483,.0,-.328);           //18
        glTexCoord2f(1.f, 1.0f);glVertex3f(-5.804,.0,-.012);            //17
        glTexCoord2f(1.f, 0.0f);glVertex3f(-5.,.0,.036);                //16
        glTexCoord2f(1.f, -1.f);glVertex3f(-3.861,.0,-.206);            //15
}
glEnd();

// Top

glBegin(GL_TRIANGLE_STRIP);
{
        glTexCoord2f(0.0f, 0.0f);glVertex3f(4.419, .0, -.789);   //1
        glTexCoord2f(1.f, 0.0f);glVertex3f(3.642, .1, -1.347);   //2
        glTexCoord2f(1.f, 1.0f);glVertex3f(3.642, -.1, -1.347);  //2
        glTexCoord2f(1.f, -1.f);glVertex3f(3.096, .15, -1.408);  //3
        glTexCoord2f(1.f, 1.f);glVertex3f(3.096, -.15, -1.408);//3
        glTexCoord2f(0.0f, 0.0f);glVertex3f(2.355, .235, -1.978);       //42
        glTexCoord2f(0.0f, 1.0f);glVertex3f(2.355, -.235, -1.978);      //42
```

```
            glTexCoord2f(1.f, 0.0f);glVertex3f(1.622,.2,-2.063);          //41
            glTexCoord2f(1.f, -1.0f);glVertex3f(1.622,-.2,-2.063);        //41
            glTexCoord2f(1.f, 1.f);glVertex3f(.814,.2,-2.063);      //40
            glTexCoord2f(1.f, -1.f);glVertex3f(.814,-.2,-2.063);          //40
            glTexCoord2f(1.f, 1.0f);glVertex3f(.389,.2,-2.366);           //39
            glTexCoord2f(1.f, 0.0f);glVertex3f(.389,-.2,-2.366);          //39
            glTexCoord2f(0.0f, -1.f);glVertex3f(0.,.2,-2.338);      //39b
            glTexCoord2f(0.0f, 1.f);glVertex3f(0.,-.2,-2.338);      //39b
            glTexCoord2f(1.f, 1.f);glVertex3f(-.6192,.2,-2.293);    //38
            glTexCoord2f(0.f, 0.0f);glVertex3f(-.6192,-.2,-2.293);  //38
            glTexCoord2f(1.f, -1.f);glVertex3f(-1.093,.2,-2.16);          //37
            glTexCoord2f(0.f, -1.f);glVertex3f(-1.093,-.2,-2.16);         //37
            glTexCoord2f(0.0f, -1.f);glVertex3f(-1.275,.2,-1.808);  //36
            glTexCoord2f(0.0f, 1.f);glVertex3f(-1.275,-.2,-1.808);  //36
            glTexCoord2f(1.f, -1.f);glVertex3f(-2.623,0,-1.189);          //33
            }
        glEnd();


        // TAIL
        glBegin(GL_POLYGON);
        {
//              glNormal3f( 0.0f, -1.f, 1.0f);
            glTexCoord2f(0.0f, 0.0f);glVertex3f(-6.00, .80, -2.257);
            glTexCoord2f(1.f, 0.0f);glVertex3f(-6.714, .80, -2.257);
            glTexCoord2f(1.f, -1.f);glVertex3f(-6.714, -.80, -2.257);
            glTexCoord2f(0.0f, -1.f);glVertex3f(-6.00, -.80, -2.257);
        }
        glEnd();

//      cockpit glass

        glDisable(GL_TEXTURE_2D);

        glPopMatrix();
}

void DrawDisplay::draw_helshad(void) {

        float vec[10][3] = {{4.194,0.,0.},
                                        {2.,.7,0.},
                                        {0.,.9,0.},
                                        {-2.,.3,0.},
                                        {-6.2,.05,0.},
                                        {-6.2,0.,0.},
                                        {-6.2,-.05,0.},
                                        {-2.,-.3,0.},
                                        {0.,-.9,0.},
                                        {2.,-.7,0.}};

        float tail[4][3] = {{-5.8, .80, 0.},
                                        {-6.2, .80,0.},
                                        {-6.2, -.80,0.},
                                        {-5.8, -.80,0.}};


        double kf,kfx,alt;

        alt = state[ALT]/hfct;
```

```
        if(alt > 20)kf = 3.28*1.3;
        if(alt <= 30)kf = 3.28*(1. + .3*alt/20.);
        kf = 3.28;
        kfx = kf*cos(state[THETA]);


        glPushMatrix();
        glColor4f(.3,.75,.4,.4);
        glTranslatef(state[NORTH]+alt*.1, state[EAST], -1.);
        glScalef(kfx,kf,kf);
        glBegin(GL_POLYGON);
        glVertex3fv(vec[0]);
        glVertex3fv(vec[1]);
        glVertex3fv(vec[2]);
        glVertex3fv(vec[3]);
        glVertex3fv(vec[4]);
        glVertex3fv(vec[5]);
        glVertex3fv(vec[6]);
        glVertex3fv(vec[7]);
        glVertex3fv(vec[8]);
        glVertex3fv(vec[9]);
        glEnd();
        glBegin(GL_POLYGON);
        glVertex3fv(tail[0]);
        glVertex3fv(tail[1]);
        glVertex3fv(tail[2]);
        glVertex3fv(tail[3]);
        glEnd();
        glPopMatrix();
}

/*---------------------------*/
/* draw landing spot */
/*---------------------------*/

void DrawDisplay::draw_tchdwn(void) {
        static double dist,x_tch,y_tch,z_tch;

        float long_l[2][3] = {{-.5,.5,0.},
                                        {-.5,5.,0.}};
        float shrt[2][3] = {{-.5,5.,0.},
                                        {.5,5.,0.}};
        float long_r[2][3] = {{.5,5.,0.},
                                        {.5,.5,0.}};
        float trunk[2][3] = {{0.,0.,0.},
                                        {0.,0.,-8.}};
        float brnch[2][3] = {{0.,0.,-8.},
                                        {0.,1.,-6.}};
        dist = 770.6;

        if(!ot_flag){
                x_tch = state[NORTH] + cos(state[PSI])*dist;
                y_tch = state[EAST] + sin(state[PSI])*dist;
//              z_tch = state[DOWN] + ALT_INIT-1.;
                z_tch = state[DOWN] + 400.0-1.0;
        }
        else if(ot_flag){
                glColor3f(1.,.2,.2);
                glPushMatrix();
                glTranslatef(x_tch,y_tch,z_tch);
```

```
            glLineWidth(1);
            glScalef(2.,2.,2.);
            glBegin(GL_LINE_STRIP);
            glVertex3fv(long_l[0]);
            glVertex3fv(long_l[1]);
            glVertex3fv(shrt[0]);
            glVertex3fv(shrt[1]);
            glVertex3fv(long_r[0]);
            glVertex3fv(long_r[1]);
            glEnd();
            glRotatef(90.,0.,0.,1.);
            glBegin(GL_LINE_STRIP);
            glVertex3fv(long_l[0]);
            glVertex3fv(long_l[1]);
            glVertex3fv(shrt[0]);
            glVertex3fv(shrt[1]);
            glVertex3fv(long_r[0]);
            glVertex3fv(long_r[1]);
            glEnd();
            glRotatef(90.,0.,0.,1.);
            glBegin(GL_LINE_STRIP);
            glVertex3fv(long_l[0]);
            glVertex3fv(long_l[1]);
            glVertex3fv(shrt[0]);
            glVertex3fv(shrt[1]);
            glVertex3fv(long_r[0]);
            glVertex3fv(long_r[1]);
            glEnd();
            glRotatef(90.,0.,0.,1.);
            glBegin(GL_LINE_STRIP);
            glVertex3fv(long_l[0]);
            glVertex3fv(long_l[1]);
            glVertex3fv(shrt[0]);
            glVertex3fv(shrt[1]);
            glVertex3fv(long_r[0]);
            glVertex3fv(long_r[1]);
            glEnd();
            glPopMatrix();
      }
}

void DrawDisplay::update_cmds(OptResult *optR)
{
      double *t_t   = optR->t_t;
      double *alp_t = optR->alp_t;
      double *col_t = optR->col_t;
      double *alp   = new double [30];

      for( int i=0; i < optR->ToPilot; i++ )
      {
            alp[i]        = (alp_t[i]*deg2rad);
            ptch[i]       = -alp[i] + mast_inc;
//          col[i]        = col_t[i]*deg2rad - COLL_MIN;  // DCL & ENB on 7/1/04
            col[i]        = col_t[i]*deg2rad;
            inp_time[i] = t_t[i];
      }
}

void DrawDisplay::draw_cmds(OptResult *optR)
{
```

```
bool    swtch_flag;
int             t_opt_cnt = 0;
int             i_brk, i, num_tiks,j,j_cnt;
double  delt, ctsc, cytr, atsc,atr, ctr,csc,asc;
double  k2;
double  t_otto;
float   aseg[50][3] = {0};
float   cseg[50][3] = {0};
float   vertik[2][3] = {0};
float   v_aseg[500] = {0};
float   v_cseg[500] = {0};
static bool h_flag = TRUE;
float auto_alt;

//      auto_alt = (60 - state[U]/ufct/4);
//      auto_alt = (60 - state[U]/ufct/4 + state[W]/wfct/2);//sss
//      auto_alt = (75 - state[U]/ufct/4 + state[W]/wfct/2);
//      auto_alt = 50;
        auto_alt = 32*min(1,(1 - state[U]/ufct/100));

        COLL_CMD = 0;
        THETA_CMD = 0;

        if(state[ALT]/hfct > auto_alt && h_flag)
        {
                swtch_flag = FALSE;
        }
        else
        {
                swtch_flag = FALSE;
//              swtch_flag = TRUE;
                h_flag = FALSE;
        }

        if(!swtch_flag)
        {

        atsc = 0.5;
        atr  = 0.4;
        asc  = 5.;
        k2   = 0.02;

        csc  = k2*100/(.12*.35/.15);
        ctsc = -0.5;
        ctr  = -.4;
        cytr = 0.;

        delt = 1;

        glColor4f(.6,.6,.8,1);
        if(!h_flag)glColor4f(0,0,.8,1);

        t_otto = 0;
        glLineWidth(3);
```

```
        if(ENG_FAIL){
                t_fail = t_sim - t_fail_st;
                t_otto = t_fail;
                t_opt_cnt += 1;
        }

        for(i=0;i<optR->ToPilot-1;i++)
        {
                if(t_otto < inp_time[i+1]){
                        i_brk = i;
                        break;
                }
        }
        if(t_otto >= inp_time[optR->ToPilot-1])i_brk = optR->ToPilot-1;
        if(t_otto < inp_time[optR->ToPilot-1]){

                aseg[0][0] = 0;
                aseg[0][1] = sin(interp((inp_time[i_brk]-t_otto),(inp_time[i_brk+1]-
t_otto),time_bias,(ptch[i_brk]),(ptch[i_brk+1])));

                THETA_CMD = asin(aseg[0][1]);

                for(i=1;i<optR->ToPilot-i_brk;i++){
                        aseg[i][0] = inp_time[i_brk+i] - t_otto - time_bias;
                        aseg[i][1] = sin(ptch[i_brk+i]);
                }

                glPushMatrix();
//              glTranslatef(atr,-10*sin(state[THETA]),-10. );
//              glTranslatef(atr,-5*sin(THETA_CMD),-10. );
//              glTranslatef(atr,-5*sin(asin(aseg[0][1])),-10. );
                glTranslatef(atr,-5*sin(state[THETA]),-10. );
                glScalef(atsc,asc,1.);
                glBegin(GL_LINE_STRIP);
                for(i=0;i<optR->ToPilot-i_brk;i++)glVertex3fv(aseg[i]);
                glEnd();
                glPopMatrix();
        }
        else if(t_otto >= inp_time[optR->ToPilot-1]){
                THETA_CMD = (ptch[optR->ToPilot-1]);
        }

        num_tiks = floor(inp_time[optR->ToPilot-1] - t_otto);
//      num_tiks = 10;

        j_cnt = 1;
        for(j=0;j<num_tiks;j++){
                while((inp_time[j_cnt]-t_otto) <= j+1){
                        j_cnt++;
                }
                if((inp_time[j_cnt]-t_otto) > j+1)
                {
                        v_aseg[j] = sin(interp((inp_time[j_cnt-1]-t_otto),(inp_time[j_cnt]-
t_otto),j+1+time_bias,(ptch[j_cnt-1]),(ptch[j_cnt])));
                        v_cseg[j] = sin(interp((inp_time[j_cnt-1]-t_otto),(inp_time[j_cnt]-
t_otto),j+1+time_bias,(col[j_cnt-1]),(col[j_cnt])));
```

```
                }
        }
        glColor4f(1,1,1,1);
        glPointSize (7);
        glLineWidth(2);
        for(i=0;i<num_tiks;i++){
        vertik[0][0] = i+1;
        vertik[1][0] = i+1;
        vertik[0][1] = v_aseg[i] + .0075;
        vertik[1][1] = v_aseg[i] - .0075;
        glPushMatrix();
//      glTranslatef(atr,-10*sin(state[THETA]),-10. );
//      glTranslatef(atr,-5*sin(THETA_CMD),-10. );
//      glTranslatef(atr,-5*sin(asin(aseg[0][1])),-10. );
        glTranslatef(atr,-5*sin(state[THETA]),-10. );
        glScalef(atsc,asc,1.);
        glBegin(GL_LINES);
        glVertex3fv(vertik[0]);
        glVertex3fv(vertik[1]);
        glEnd();
        glPopMatrix();
        }


        // collective commands


        glColor4f(.6,.6,.8,1);
        if(!h_flag)glColor4f(0,0,.8,1);
                for(i=0;i<optR->ToPilot-1;i++)
        {
                if(t_otto < inp_time[i+1]){
                        i_brk = i;
                        break;
                }
        }
        if(t_otto >= inp_time[optR->ToPilot-1])i_brk = optR->ToPilot-1;
        if(t_otto < inp_time[optR->ToPilot-1]){

                cseg[0][0] = 0.0;
                cseg[0][1] = (interp((inp_time[i_brk]-t_otto),(inp_time[i_brk+1]-
t_otto),time_bias,(col[i_brk]),(col[i_brk+1])));
                COLL_CMD = (cseg[0][1]);
                for(i=1;i<optR->ToPilot-i_brk;i++){
                        cseg[i][0] = inp_time[i_brk+i] - t_otto - time_bias;
                        cseg[i][1] = (col[i_brk+i]);
                }
                glLineWidth(3);
                glPushMatrix();
                glTranslatef(ctr,cytr-.025*coll_per,-10. );
                glScalef(ctsc,csc,1.);
                glBegin(GL_LINE_STRIP);
                for(i=0;i<optR->ToPilot-i_brk;i++)glVertex3fv(cseg[i]);
                glEnd();
                glPopMatrix();
        }
```

```
        else if(t_otto >= inp_time[optR->ToPilot-1]){
                COLL_CMD = (col[optR->ToPilot-1]);
        }


        // time ticks

        glColor4f(1,1,1,1);
        glLineWidth(2);
        for(i=0;i<num_tiks;i++){
        vertik[0][0] = i+1;
        vertik[1][0] = i+1;
        vertik[0][1] = v_cseg[i] + .0055;
        vertik[1][1] = v_cseg[i] - .0055;
        glPushMatrix();
        glTranslatef(ctr,cytr-.025*coll_per,-10. );
        glScalef(ctsc,csc,1.);
        glBegin(GL_LINES);
        glVertex3fv(vertik[0]);
        glVertex3fv(vertik[1]);
        glEnd();
        glPopMatrix();
        }
}
else
{
        /*
        float alt_st = auto_alt;//max(5,auto_alt - 2);
        float u_st = 40*1.69;
        float alt_gain = .1/alt_st;
        float sink_st;
        static float theta_max;
        float st_per;
        float sink_gain;
        float sink_comp;
        static bool alp_flag = TRUE;
        float skid_ht = 3.;
        float let_out = 1;
        float omg_gain;
        float dummy1;
        omg_gain = .01*(state[O] - 1.0);

        st_per = 0.5 + omg_gain;
        sink_gain = 3/20*(1.+ alt_gain*(alt_st - (state[ALT]/hfct - skid_ht)));
        sink_st = max(1,(state[ALT]/hfct));
        if(alp_flag)
        {
                theta_max = 25*min(1,(1 - max(((u_st - state[U]/ufct)/u_st),0)));
                alp_flag = FALSE;
        }
        if(state[W]/wfct < 3)let_out = min(0,(1+state[W]/wfct/10));
        THETA_CMD = (theta_max/57.3*(1 - max(((alt_st - (state[ALT]/hfct - skid_ht))/alt_st),0))+5/57.3);
        sink_comp = max(0,state[W]/wfct);
        dummy1 = abs((sink_comp) - sink_st);
        COLL_CMD = let_out*max_pull()*(st_per + min((1-st_per),(1-st_per)*sink_gain*dummy1));
        */
```

```
//
        float u_st = 40*1.69;
        float alt_gain = .1/auto_alt;
        float sink_st;
        static float theta_max;
        float st_per = .6;
        float sink_gain;
        float sink_comp;
        float col_cnst;
        static bool alp_flag = TRUE;
        sink_gain = 3/20.*(1.+ alt_gain*(auto_alt - state[ALT]/hfct));
        sink_st = max(1,(state[ALT]/hfct));
        sink_comp = max(0,state[W]/wfct);
        col_cnst = sink_gain*(sink_comp - sink_st);


        if(alp_flag)
        {
                theta_max = 25*min(1,(1 - (u_st - state[U]/ufct)/u_st));
                alp_flag = FALSE;
        }
        THETA_CMD = theta_max/57.3*(1 - 1/auto_alt*(auto_alt - state[ALT]/hfct));
        COLL_CMD = max_pull()*(st_per + min((1-st_per),(1-st_per)*col_cnst));
/*
FILE *pnFile = NULL;
if( !pnFile )
pnFile = fopen( "errorout.txt", "a" );
char buf[128];
sprintf( buf, " alt %g col_cnst %g\n", state[ALT]/hfct, max_pull());
fprintf( pnFile, buf );
fflush( pnFile );
fclose(pnFile);
*/
}
}
void DrawDisplay::draw_tiles(void)
{
        int i, j;
        double cpsi, spsi, tan_th, y0, x0, x_max, y_max, x, y;
        double trm1, trm2, that, x_lo, y_lo, x_hi, y_hi;
        double hdg;

double TMAXX = 100;
double TMAXY = 100;

if(view == CHASEPLANE)hdg = chs_psi;
if(view == OUT_THE_WINDOW)hdg = state[PSI];

cpsi = cos(hdg);
```

```
spsi = sin(hdg);

    thta = 80/57.3/2;
    tan_th = tan(thta);
    x0 = state[NORTH]+nn;
    y0 = state[EAST]+en;
    x_max = __min(FPT * TMAXX, MAX_TD);
    y_max = __min(FPT * TMAXY, MAX_TD);
    x_lo = __max((x0-x_max),0);
    x_hi = __min((x0+x_max),FPB*BMAXX);
    y_lo = __max((y0-y_max),0);
    y_hi = __min((y0+y_max),FPB*BMAXY);

    glPushMatrix();

    if((hdg) <= 45/57.3 || (hdg) > 315/57.3){

        for(x = x0;x<=x_hi;x += FPT){
            i = floor(x/FPT)-4*cpsi;
            trm1 = y0 + (x - x0)*(-cpsi*tan_th + spsi)/( cpsi+spsi*tan_th);
            trm2 = y0 + (x - x0)*( cpsi*tan_th + spsi)/( cpsi-spsi*tan_th);
            if(trm1 < 0)trm1 = 0;
            if(trm2 > y_hi)trm2 = y_hi;

            for(y = trm1;y<= trm2; y+=FPT){
                j = floor(y/FPT)-4*spsi;;
                draw_lines(i,j);
            }
        }
    }

    else if((hdg) > 45/57.3 && (hdg) <= 135/57.3){

        for(y = y0;y<=y_hi;y += FPT){
            j = floor(y/FPT)-4*spsi;
            trm1 = x0 + (y - y0)*( cpsi+spsi*tan_th)/(-cpsi*tan_th + spsi);
            trm2 = x0 + (y - y0)*( cpsi-spsi*tan_th)/( cpsi*tan_th + spsi);
            if(trm2 < 0)trm2 = 0;
            if(trm1 > x_hi)trm1 = x_hi;

            for(x = trm2;x<= trm1; x+=FPT){
                i = floor(x/FPT)-4*cpsi;;
                draw_lines(i,j);
            }
        }
    }

    else if((hdg) > 135/57.3 && (hdg) <= 225/57.3){

        for(x = x0;x>=x_lo;x -= FPT){
            i = floor(x/FPT)-4*cpsi;
            trm1 = y0 + (x - x0)*(-cpsi*tan_th + spsi)/( cpsi+spsi*tan_th);
            trm2 = y0 + (x - x0)*( cpsi*tan_th + spsi)/( cpsi-spsi*tan_th);
            if(trm2 < 0)trm2 = 0;
            if(trm1 > y_hi)trm1 = y_hi;

            for(y = trm2;y<= trm1; y+=FPT){
                j = floor(y/FPT)-4*spsi;;
                draw_lines(i,j);
            }
```

```
            }
        }

        else if((hdg) > 225/57.3 && (hdg) <= 315/57.3){

                for(y = y0;y>=y_lo;y -= FPT){
                        j = floor(y/FPT)-4*spsi;
                        trm1 = x0 + (y - y0)*( cpsi+spsi*tan_th)/(-cpsi*tan_th + spsi);
                        trm2 = x0 + (y - y0)*( cpsi-spsi*tan_th)/( cpsi*tan_th + spsi);
                        if(trm1 < 0)trm1 = 0;
                        if(trm2 > x_hi)trm2 = x_hi;

                        for(x = trm1;x<= trm2; x+=FPT){
                                i = floor(x/FPT)-4*cpsi;;
                                draw_lines(i,j);
                        }
                }
        }

        glPopMatrix();

}


void DrawDisplay::draw_pip(void)
{
  int i,j;
  double crit;


  float bore_fp[6][3] =         {{-.65, 0, 0. },
                                {-.15, 0, 0.},
                                {-.25,-.1, 0.  },
                                {.65, 0, 0. },
                                {.15, 0, 0.},
                                {.25,-.1, 0.}};

  float hor[4][3] =     {{-3., 0, 0. },
                                {-2, 0, 0. },
                                { 2, 0, 0. },
                                { 3. , 0, 0.}};


   glLineWidth(1);
   glColor4f(1., 1., 0, 1);
   crit = state[ALT]/hfct-fabs(6*3.28*sin(state[THETA]));
   if(crit <= 1.5*state[W]/wfct && !land_flag){
          if(blink_one){
                glColor4f(1., 1., 0, 1.);
                glLineWidth(3);
          }
               if(blink_two)glColor4f(1., 1., 0, 1);
       }


   for(i=0;i<6;i++) {
          for(j=0;j<3;j++) {
                        bore_fp[i][j] = bore_fp[i][j] * 2.;
          }
   }
```

```
    glPushMatrix();
 glTranslatef(0, 0, -10);

 glBegin(GL_LINES);
        glVertex3fv(bore_fp[0]);
        glVertex3fv(bore_fp[1]);
 glEnd();
 glBegin(GL_LINES);
        glVertex3fv(bore_fp[3]);
        glVertex3fv(bore_fp[4]);
 glEnd();
glPopMatrix();


/*

 for(i=0;i<4;i++) {
      for(j=0;j<3;j++) {
            hor[i][j] = hor[i][j] * 1.;
      }
 }

glPushMatrix();
glColor3f(.0, .8, 1.);
glLineWidth(2);
 glTranslatef(0, 0, -10);
  glRotatef(state[PHI]*57.3,0.,0.,1.);
  glTranslatef(0.,-10*(double)sin(state[THETA]), 0. );
   glBegin(GL_LINES);
        glVertex3fv(hor[0]);
        glVertex3fv(hor[1]);
 glEnd();
 glBegin(GL_LINES);
        glVertex3fv(hor[2]);
        glVertex3fv(hor[3]);
 glEnd();
glPopMatrix();
*/

}

void DrawDisplay::draw_rotor(void)
{
      double      t_Ind;
      int i;
      float drg;
      glPushMatrix();
      if(!land_flag)t_Ind = t_sim;
      drg = exp(1*(t_Ind-t_sim));
      glRotatef(100*(t_Ind - 1*drg), 0.0, 0.0, 1.0);
      for(i=0;i<5;i++){
            glPushMatrix();
            glRotatef(72*i, 0.0, 0.0, 1.0);
            glBegin(GL_POLYGON);
            {
                  glTexCoord2f(0.0f, 0.0f);glVertex3f(0.0, .75, -8.5);
                  glTexCoord2f(1.f, 0.0f);glVertex3f(18.0, .75, -9.5);
                  glTexCoord2f(1.f, -1.f);glVertex3f(18.0, -.75, -9.5);
```

```
                glTexCoord2f(0.0f, -1.f);glVertex3f(0.0, -.75, -8.5);
            }
            glEnd();

            glBegin(GL_POLYGON);
            {
                glTexCoord2f(0.0f, 0.0f);glVertex3f(1.5, 1.2, -8);
                glTexCoord2f(1.f, -1.f);glVertex3f(1.5, -1.2, -8);
                glTexCoord2f(1.f, 0.0f);glVertex3f(0.0, 0, -9);

            }
            glEnd();
            glPopMatrix();
        }
        glPopMatrix();
}

void DrawDisplay::draw_lines(int i, int j) {

        float vv[4][3], red, grn, blu;
        double x_00, x_10, y_00, y_01, z_00, z_01, z_10, z_11;
        double h_l, h_r, px, py, pz, dsp_dist, trma;
        int i_blk, j_blk;
        double mag, x_tile, y_tile, z_tile;
        int i_00_00,j_00_00,i_00_10,j_00_10,i_00_11,j_00_11,i_00_01,j_00_01;
        int i_10_00,j_10_00,i_10_10,j_10_10,i_10_11,j_10_11,i_10_01,j_10_01;
        int i_11_00,j_11_00,i_11_10,j_11_10,i_11_11,j_11_11,i_11_01,j_11_01;
        int i_01_00,j_01_00,i_01_10,j_01_10,i_01_11,j_01_11,i_01_01,j_01_01;
        double bx_00_00,bx_00_10,by_00_00,by_00_01,bx_10_00,bx_10_10,by_10_00,by_10_01;
        double bx_11_00,bx_11_10,by_11_00,by_11_01,bx_01_00,bx_01_10,by_01_00,by_01_01;

        x_tile = i * FPT;
        y_tile = j * FPT;

        i_blk = floor(x_tile/FPB);
        j_blk = floor(y_tile/FPB);

        z_00 = gnd[i_blk][j_blk];
        z_10 = gnd[i_blk+1][j_blk];
        z_01 = gnd[i_blk][j_blk+1];
        z_11 = gnd[i_blk+1][j_blk+1];

        h_l = interp(i_blk*FPB, (i_blk+1)*FPB, x_tile, z_00, z_10);
        h_r = interp(i_blk*FPB, (i_blk+1)*FPB, x_tile, z_01, z_11);
        z_tile = interp(j_blk*FPB, (j_blk+1)*FPB, y_tile, h_l, h_r);

        px = (x_tile - state[NORTH]);
        py = (y_tile - state[EAST]);
        pz = (z_tile - state[DOWN]);

        mag   = sqrt(px*px + py*py);

        red = 1. - mag/MAX_TD;
        if(red < .6)red = .6;

        grn = .9 - mag/MAX_TD;
        if(grn < .7)grn = .7;

        blu = .85 - mag/MAX_TD;
        if(blu < .4)blu = .4;
```

```
        glColor4f(red, grn, blu,.5);
//      glColor3f(1,1,1);

/* assign ground blocks to respective tile corners */

        x_00 = x_tile - TW/2.;
        x_10 = x_tile + TW/2.;
        y_00 = y_tile - TW/2.;
        y_01 = y_tile + TW/2.;

        i_00_00 = floor(x_00/FPB);
        j_00_00 = floor(y_00/FPB);
        i_00_10 = i_00_00 + 1;
        j_00_10 = j_00_00;
        i_00_11 = i_00_10;
        j_00_11 = j_00_00 + 1;
        i_00_01 = i_00_00;
        j_00_01 = j_00_11;

        i_10_00 = floor(x_10/FPB);
        j_10_00 = j_00_00;
        i_10_10 = i_10_00 + 1;
        j_10_10 = j_10_00;
        i_10_11 = i_10_10;
        j_10_11 = j_10_00 + 1;
        i_10_01 = i_10_00;
        j_10_01 = j_10_11;

        i_11_00 = i_10_00;
        j_11_00 = floor(y_01/FPB);
        i_11_10 = i_11_00 + 1;
        j_11_10 = j_11_00;
        i_11_11 = i_11_10;
        j_11_11 = j_11_00 + 1;
        i_11_01 = i_11_00;
        j_11_01 = j_11_11;

        i_01_00 = i_00_00;
        j_01_00 = j_11_00;
        i_01_10 = i_01_00 + 1;
        j_01_10 = j_01_00;
        i_01_11 = i_01_10;
        j_01_11 = j_01_00 + 1;
        i_01_01 = i_01_00;
        j_01_01 = j_01_11;

        bx_00_00 = i_00_00*FPB;
        bx_00_10 = i_00_10*FPB;
        by_00_00 = j_00_00*FPB;
        by_00_01 = j_00_01*FPB;

        bx_10_00 = i_10_00*FPB;
        bx_10_10 = i_10_10*FPB;
        by_10_00 = j_10_00*FPB;
        by_10_01 = j_10_01*FPB;

        bx_11_00 = i_11_00*FPB;
        bx_11_10 = i_11_10*FPB;
```

```
by_11_00 = j_11_00*FPB;
by_11_01 = j_11_01*FPB;

bx_01_00 = i_01_00*FPB;
bx_01_10 = i_01_10*FPB;
by_01_00 = j_01_00*FPB;
by_01_01 = j_01_01*FPB;

if(agl > 100)dsp_dist = 150*TW;
if(agl <= 100){
        trma = agl;
        if(agl < 0 )trma = 2;
        dsp_dist = 150*TW + 500/trma;
}

glPushMatrix();
glEnable(GL_DEPTH_TEST);


if(mag < dsp_dist)
{

        h_l = interp(bx_00_00, bx_00_10, x_00, gnd[i_00_00][j_00_00], gnd[i_00_10][j_00_00]);
        h_r = interp(bx_00_00, bx_00_10, x_00, gnd[i_00_00][j_00_01], gnd[i_00_10][j_00_01]);
        z_00 = interp(by_00_00, by_00_01, y_00, h_l, h_r);

        h_l = interp(bx_10_00, bx_10_10, x_10, gnd[i_10_00][j_10_00], gnd[i_10_10][j_10_00]);
        h_r = interp(bx_10_00, bx_10_10, x_10, gnd[i_10_00][j_10_01], gnd[i_10_10][j_10_01]);
        z_10 = interp(by_10_00, by_10_01, y_00, h_l, h_r);

        h_l = interp(bx_11_00, bx_11_10, x_10, gnd[i_11_00][j_11_00], gnd[i_11_10][j_11_00]);
        h_r = interp(bx_11_00, bx_11_10, x_10, gnd[i_11_00][j_11_01], gnd[i_11_10][j_11_01]);
        z_11 = interp(by_11_00, by_11_01, y_01, h_l, h_r);

        h_l = interp(bx_01_00, bx_01_10, x_00, gnd[i_01_00][j_01_00], gnd[i_01_10][j_01_00]);
        h_r = interp(bx_01_00, bx_01_10, x_00, gnd[i_01_00][j_01_01], gnd[i_01_10][j_01_01]);
        z_01 = interp(by_01_00, by_01_01, y_01, h_l, h_r);

        vv[0][0] = x_00;
        vv[0][1] = y_00;
        vv[0][2] = z_00 - lyft;

        vv[1][0] = x_10;
        vv[1][1] = y_00;
        vv[1][2] = z_10 - lyft;

        vv[2][0] = x_10;
        vv[2][1] = y_01;
        vv[2][2] = z_11 - lyft;

        vv[3][0] = x_00;
        vv[3][1] = y_01;
        vv[3][2] = z_01 - lyft;


        glBegin(GL_LINES);
        {
                glVertex3fv(vv[0]);
                glVertex3fv(vv[1]);
        }
```

```
            glEnd();

            glBegin(GL_LINES);
            {
                  glVertex3fv(vv[1]);
                  glVertex3fv(vv[2]);
            }
            glEnd();

            glBegin(GL_LINES);
            {
                  glVertex3fv(vv[2]);
                  glVertex3fv(vv[3]);
            }
            glEnd();

            glBegin(GL_LINES);
            {
                  glVertex3fv(vv[3]);
                  glVertex3fv(vv[0]);
            }
            glEnd();
      }

      else if(mag > dsp_dist)
      {
            vv[0][0] = x_tile;
            vv[0][1] = y_tile;
            vv[0][2] = z_tile - lyft;

            glBegin(GL_LINES);
      {
            glVertex3fv(vv[0]);
            vv[0][0] += TW;
            glVertex3fv(vv[0]);
      }
      glEnd();

      glBegin(GL_LINES);
      {
            glVertex3fv(vv[0]);
            vv[0][1] += TW;
            glVertex3fv(vv[0]);
      }
      glEnd();

            }
      glPopMatrix();
}

GLvoid DrawDisplay::glPrint(bool blink,float xtr,float ytr, float ztr, float xsc,float ysc,
    /                                    int set,float rd,float gn,float bl,float tr, char *fmt, ...)
{
      char        text[256];                                    // Holds Our String
      va_list           ap;                                     // Pointer To List Of
Arguments

      if (fmt == NULL)                                          // If There's No Text
            return;                                             // Do Nothing
```

```
    va_start(ap, fmt);                                  // Parses The String For Variables
      vsprintf(text, fmt, ap);                          // And Converts Symbols To Actual Numbers
    va_end(ap);                                         // Results Are Stored In Text

    if (set>1)                                          // Did User Choose An Invalid
Character Set?
    {
        set=1;                                          // If So, Select Set 1 (Italic)
    }
    glEnable(GL_TEXTURE_2D);                            // Enable Texture Mapping
    glLoadIdentity();                                   // Reset The Modelview Matrix
    glPushMatrix();
    if(!blink){
            glColor4f(rd,gn,bl,tr);
        }
        else if(blink){
            if(blink_one)glColor4f(rd,gn,bl,tr);
            if(blink_two)glColor4f(1.f,1.f,0.f,1);
        }
    glTranslatef(xtr,ytr,ztr);                          // Position The Text (0,0 - Bottom Left)
    glRotatef(180,1,0,0);
    glListBase(base-32+(128*set));                      // Choose The Font Set (0 or 1)

    if (set==0)                                         // If Set 0 Is Being Used Enlarge
Font
    {
        glScalef(1.5f,2.0f,1.0f);                       // Enlarge Font Width And Height
    }
    glScalef(xsc,ysc,1.0f);
    glCallLists(strlen(text),GL_UNSIGNED_BYTE, text);   // Write The Text To The Screen
    glDisable(GL_TEXTURE_2D);                           // Disable Texture Mapping
    glPopMatrix();
}


/*--------------------------*/
/* draw FPVector footprint */
/*--------------------------*/

void DrawDisplay::draw_foot(double x_proj,double y_proj, double psi_up) {

    float vv[4][3];
    double x_00, x_10, y_00,z_00, z_10, z_11,x_11, y_11,y_10;
    double h_r, h_l;
    double cpsi, spsi;
    int i_00_00,j_00_00,l_00_10,j_00_10,l_00_11,j_00_11,l_00_01,j_00_01;
    int i_10_00,j_10_00,l_10_10,j_10_10,l_10_11,j_10_11,l_10_01,j_10_01;
    int i_11_00,j_11_00,l_11_10,j_11_10,l_11_11,j_11_11,l_11_01,j_11_01;
    double bx_00_00,bx_00_10,by_00_00,by_00_01,bx_10_00,bx_10_10,by_10_00,by_10_01;
    double bx_11_00,bx_11_10,by_11_00,by_11_01;

    spsi = sin(psi_up);
    cpsi = cos(psi_up);

  x_00 = x_proj + TW/2.*spsi;
    x_10 = x_proj + TW*cpsi;
    x_11 = x_proj - TW/2*spsi;
    y_00 = y_proj - TW/2.*cpsi;
    y_10 = y_proj + TW/2.*spsi;
    y_11 = y_proj + TW/2.*cpsi;
```

```
i_00_00 = floor(x_00/FPB);
j_00_00 = floor(y_00/FPB);
i_00_10 = i_00_00 + 1;
j_00_10 = j_00_00;
i_00_11 = i_00_10;
j_00_11 = j_00_00 + 1;
i_00_01 = i_00_00;
j_00_01 = j_00_11;

i_10_00 = floor(x_10/FPB);
j_10_00 = floor(y_10/FPB);
i_10_10 = i_10_00 + 1;
j_10_10 = j_10_00;
i_10_11 = i_10_10;
j_10_11 = j_10_00 + 1;
i_10_01 = i_10_00;
j_10_01 = j_10_11;

i_11_00 = floor(x_11/FPB);
j_11_00 = floor(y_11/FPB);
i_11_10 = i_11_00 + 1;
j_11_10 = j_11_00;
i_11_11 = i_11_10;
j_11_11 = j_11_00 + 1;
i_11_01 = i_11_00;
j_11_01 = j_11_11;

bx_00_00 = i_00_00*FPB;
bx_00_10 = i_00_10*FPB;
by_00_00 = j_00_00*FPB;
by_00_01 = j_00_01*FPB;

bx_10_00 = i_10_00*FPB;
bx_10_10 = i_10_10*FPB;
by_10_00 = j_10_00*FPB;
by_10_01 = j_10_01*FPB;

bx_11_00 = i_11_00*FPB;
bx_11_10 = i_11_10*FPB;
by_11_00 = j_11_00*FPB;
by_11_01 = j_11_01*FPB;

        h_l = interp(bx_00_00, bx_00_10, x_00, gnd[i_00_00][j_00_00], gnd[i_00_10][j_00_00]);
        h_r = interp(bx_00_00, bx_00_10, x_00, gnd[i_00_00][j_00_01], gnd[i_00_10][j_00_01]);
        z_00 = interp(by_00_00, by_00_01, y_00, h_l, h_r);

        h_l = interp(bx_10_00, bx_10_10, x_10, gnd[i_10_00][j_10_00], gnd[i_10_10][j_10_00]);
        h_r = interp(bx_10_00, bx_10_10, x_10, gnd[i_10_00][j_10_01], gnd[i_10_10][j_10_01]);
        z_10 = interp(by_10_00, by_10_01, y_10, h_l, h_r);

        h_l = interp(bx_11_00, bx_11_10, x_11, gnd[i_11_00][j_11_00], gnd[i_11_10][j_11_00]);
        h_r = interp(bx_11_00, bx_11_10, x_11, gnd[i_11_00][j_11_01], gnd[i_11_10][j_11_01]);
        z_11 = interp(by_11_00, by_11_01, y_11, h_l, h_r);

vv[0][0] = x_00;
vv[0][1] = y_00;
vv[0][2] = z_00 - lyft;

vv[1][0] = x_10;
```

```
        vv[1][1] = y_10;
        vv[1][2] = z_10 - lyft;

        vv[2][0] = x_11;
        vv[2][1] = y_11;
        vv[2][2] = z_11 - lyft;


  glColor3f(95., .3, 0);
        glBegin(GL_LINES);
        {
                glVertex3fv(vv[0]);
                glVertex3fv(vv[1]);
        }
        glEnd();

        glBegin(GL_LINES);
        {
                glVertex3fv(vv[1]);
                glVertex3fv(vv[2]);
        }
        glEnd();

        glBegin(GL_LINES);
        {
                glVertex3fv(vv[2]);
                glVertex3fv(vv[0]);
        }
        glEnd();
}

double DrawDisplay::check_imp_ht(double x_pos, double y_pos, int i, int j)
{
        double h_r, h_l, imp_ht; -

        h_l = interp(i*FPB, (i+1)*FPB, x_pos, gnd[i][j], gnd[i+1][j]);
        h_r = interp(i*FPB, (i+1)*FPB, x_pos, gnd[i][j+1], gnd[i+1][j+1]);
        imp_ht = interp(j*FPB, (j+1)*FPB, y_pos, h_l, h_r);

        return imp_ht;
}

double DrawDisplay::interp(double x_lo, double x_hi, double x_pt, double y_lo, double y_hi)
{
        double a, b, y_pt;

        a = x_pt - x_lo;
        b = x_hi - x_pt;
        y_pt = (a*y_hi + b*y_lo)/(a + b);
        return y_pt;
}
```

WHAT IS CLAIMED IS:

1. A computer implemented method for guiding a pilot during rotorcraft autorotation comprising the steps of:

(a) determining the current state of the rotorcraft;

(b) computing the current constrained optimal trajectory of the rotorcraft for autorotation to landing;

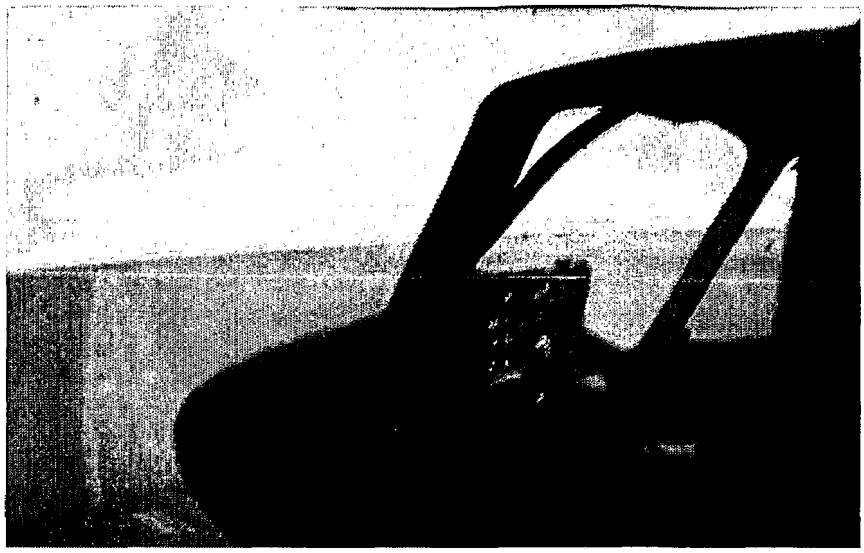
(c) computing the control inputs required to achieve the current optimal trajectory;

(d) providing the pilot visual cues where to currently position the controls to follow the current optimal trajectory;

(e) providing the pilot a visual preview of when and where to position the controls at future times to follow the current optimal trajectory;

(f) repeating steps (b) through (e) until landing occurs.

Figure 1. Bell 206L-4 single rotor helicopter



(a)

(b)

Figure 2. Frasca International Bell 206 Flight Training Device (FTD)

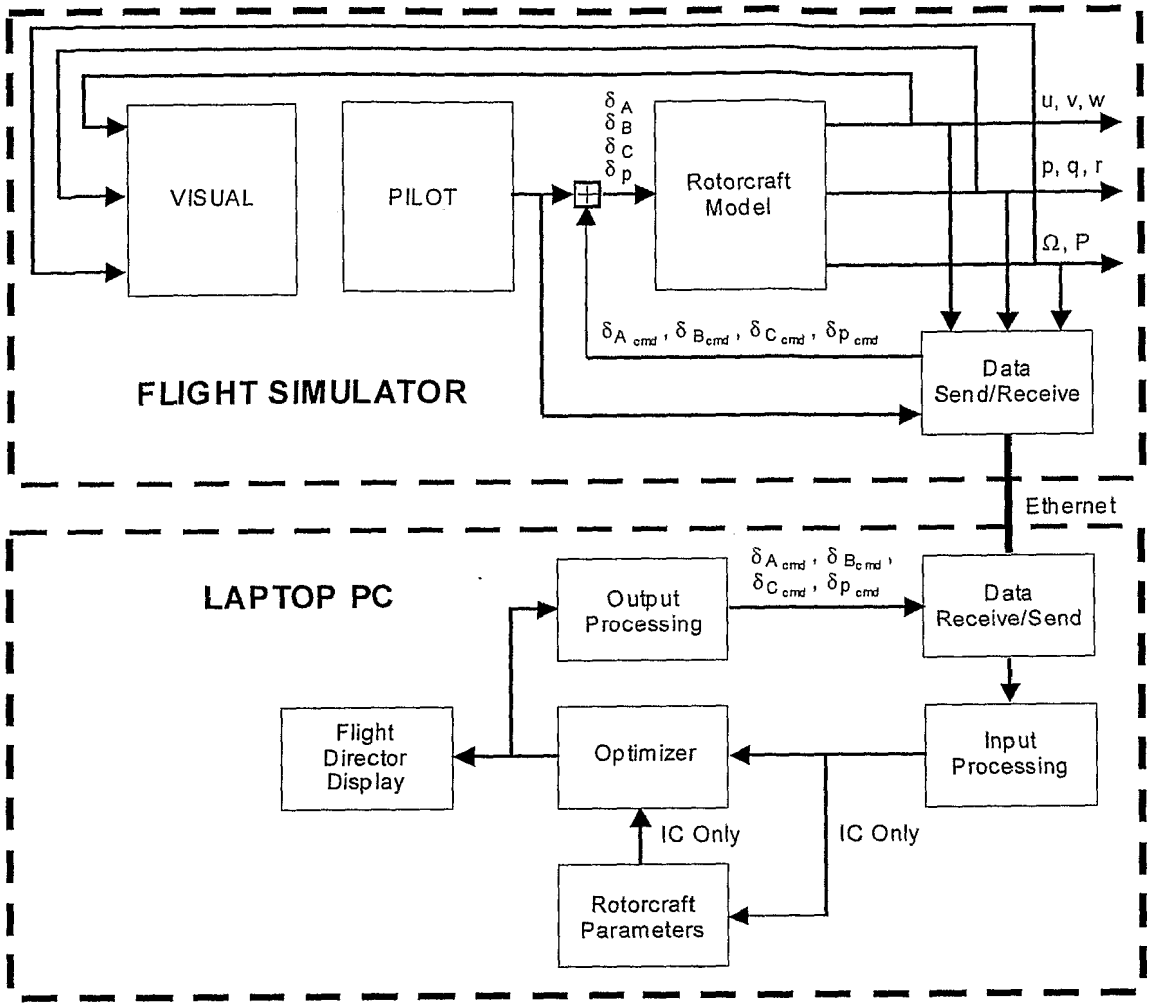


Figure 3. Interface between the optimal guidance and the FTD

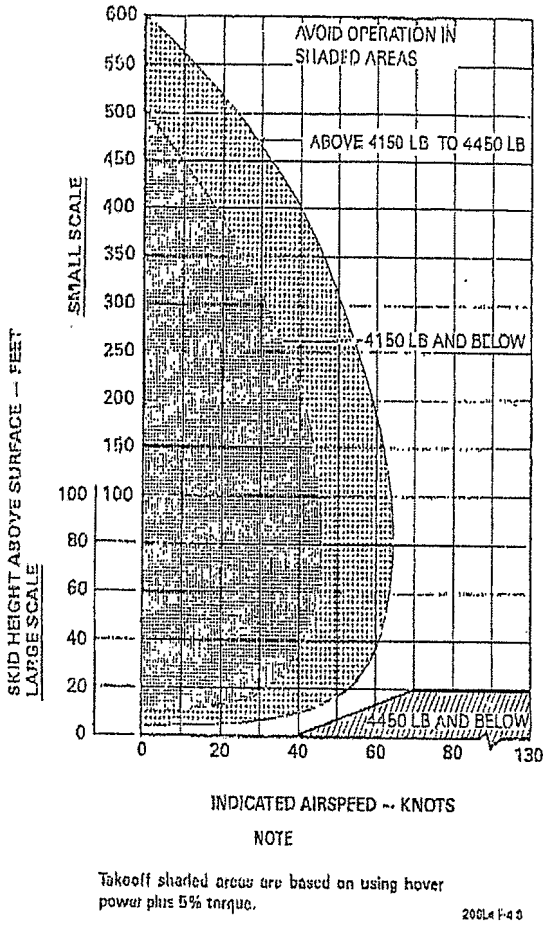


Figure 4. Height-Velocity diagram for the Bell 206L-4 Helicopter Results

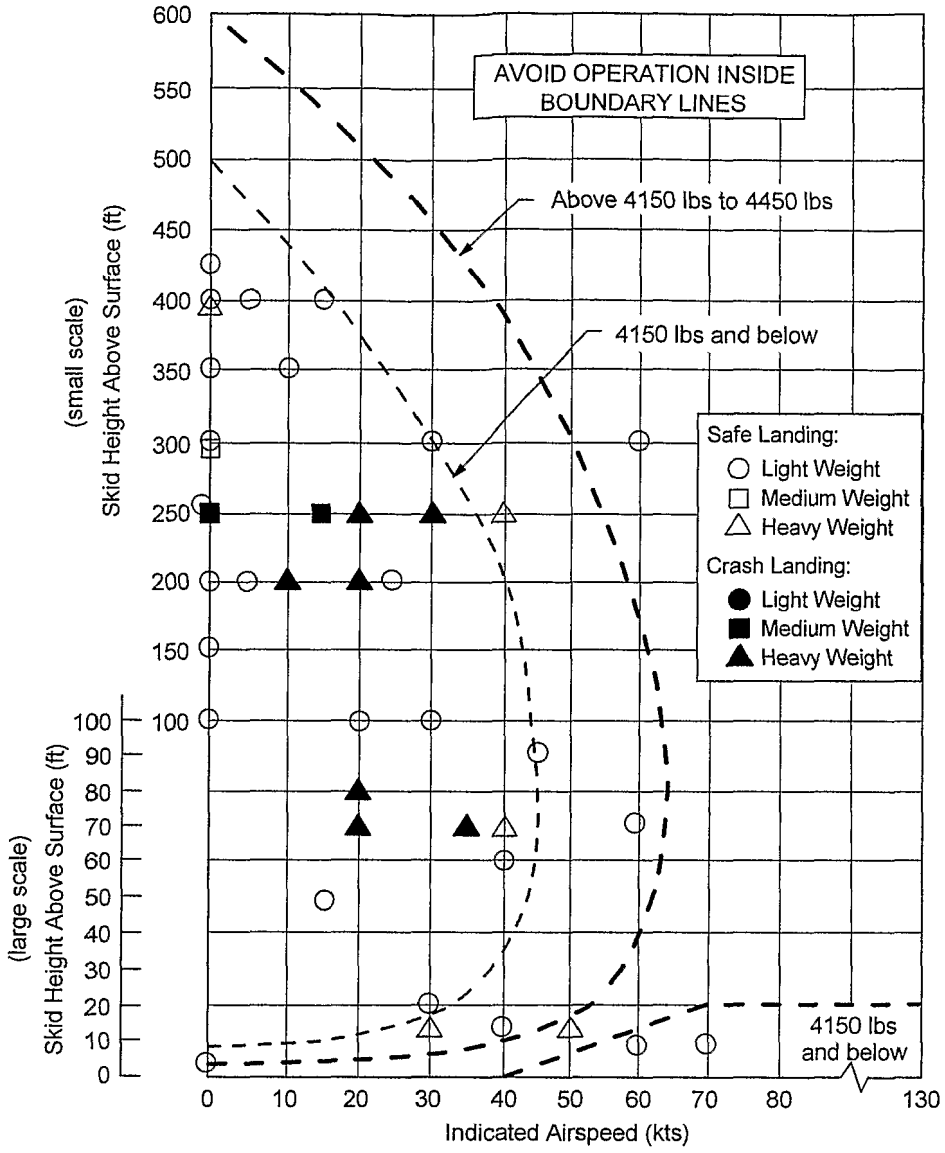


Figure 5. Automated autorotation flight conditions evaluated

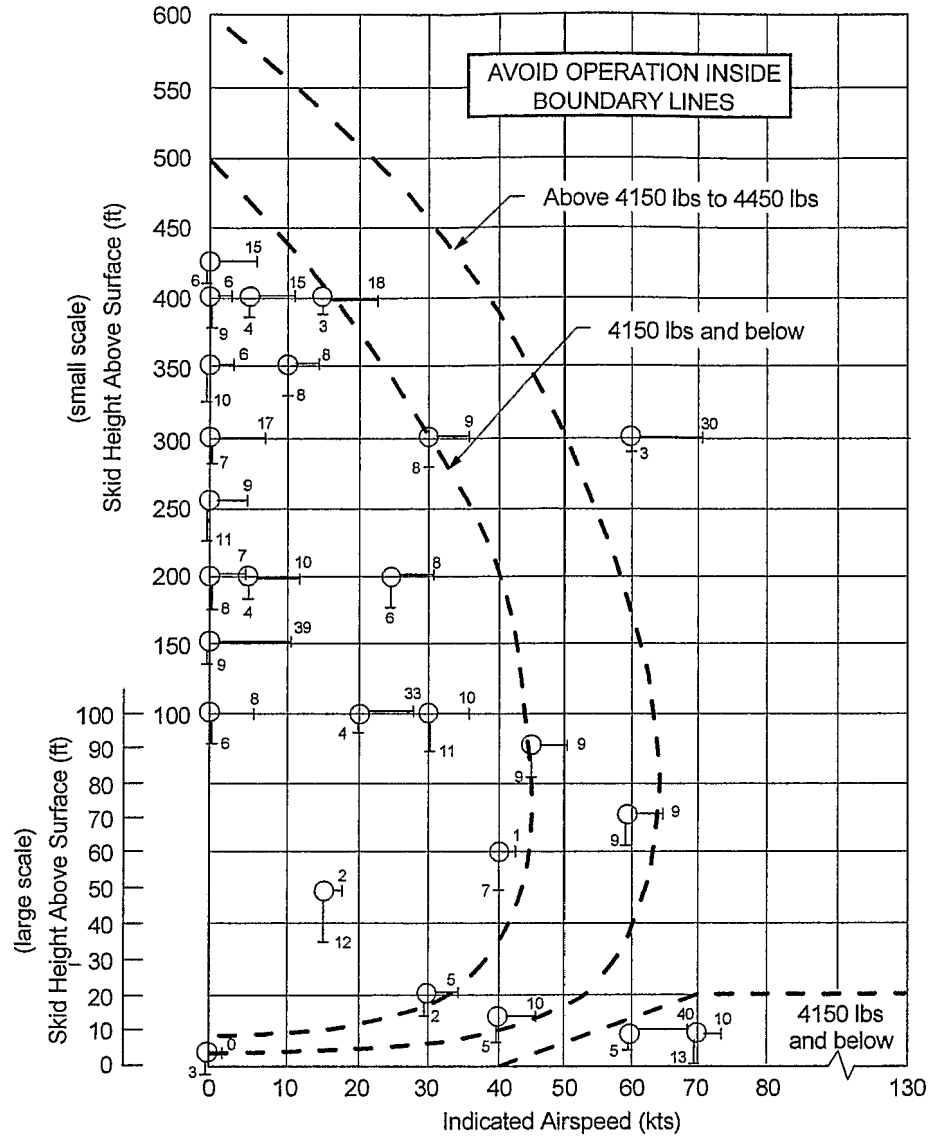


Figure 6 Touchdown ground-speed and sink-rate (light weight condition)

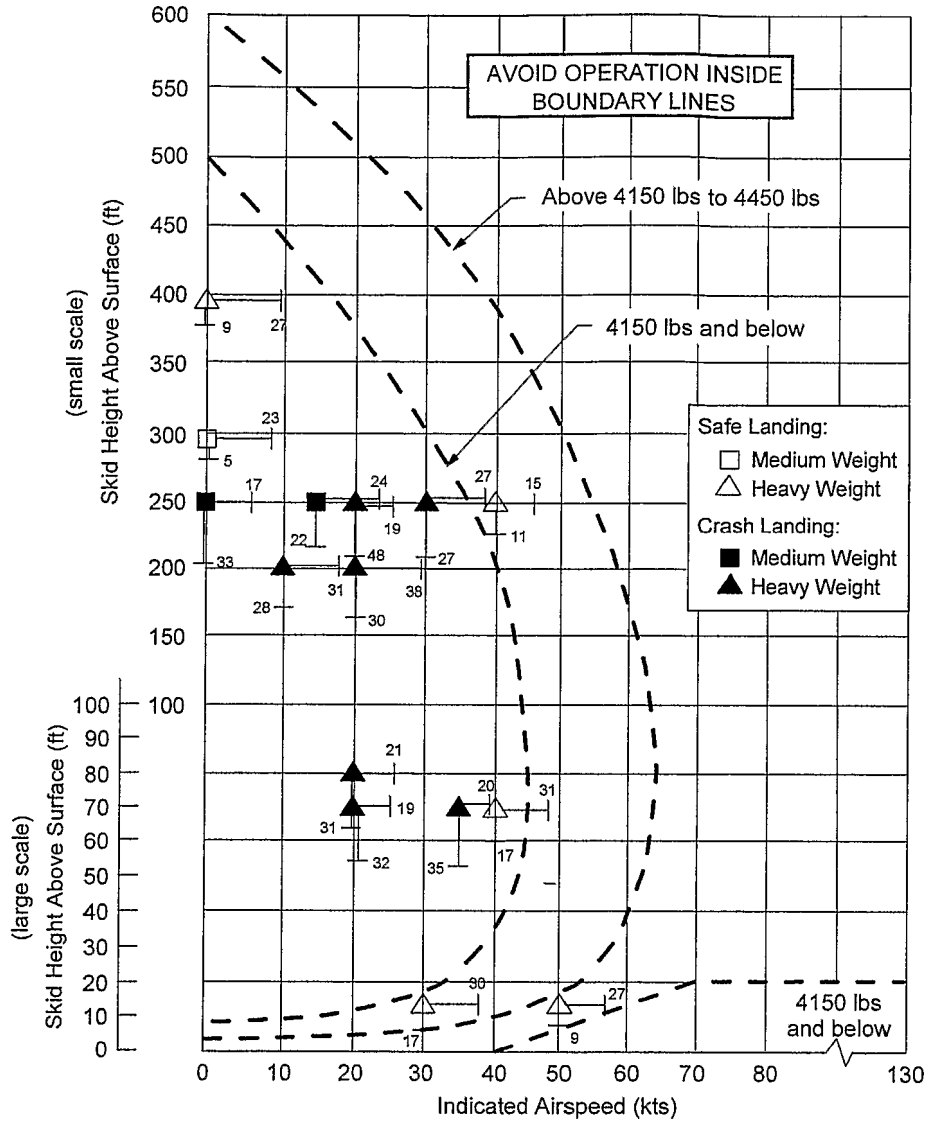


Figure 7. Touchdown ground-speed and sink-rate (medium and heavy weight conditions)

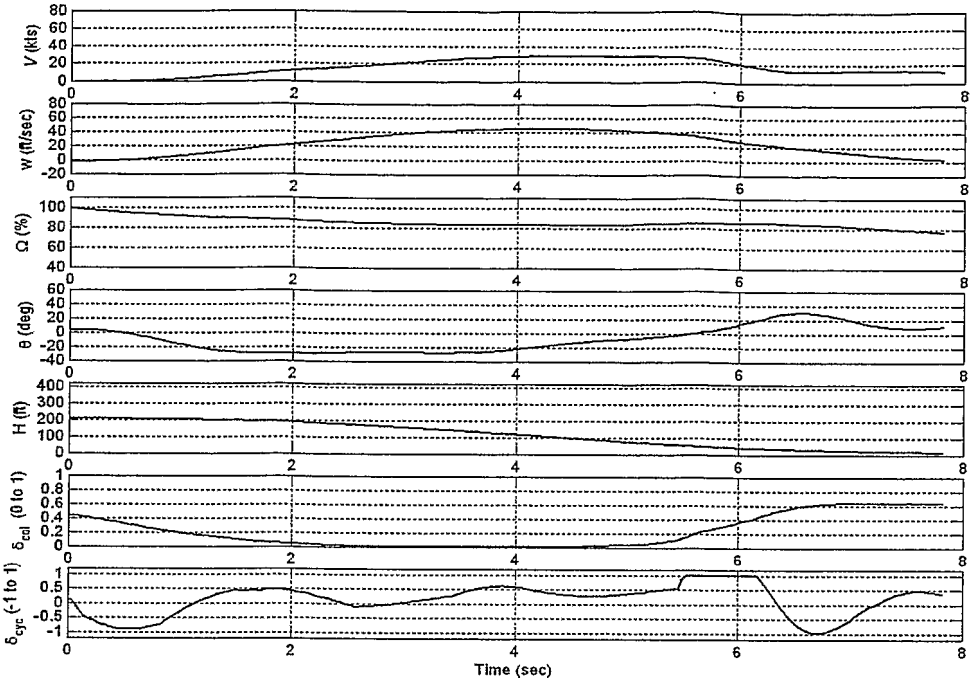


Figure 8. Automatic autorotation from 200ft/0kts; light weight condition (2900 lbs)

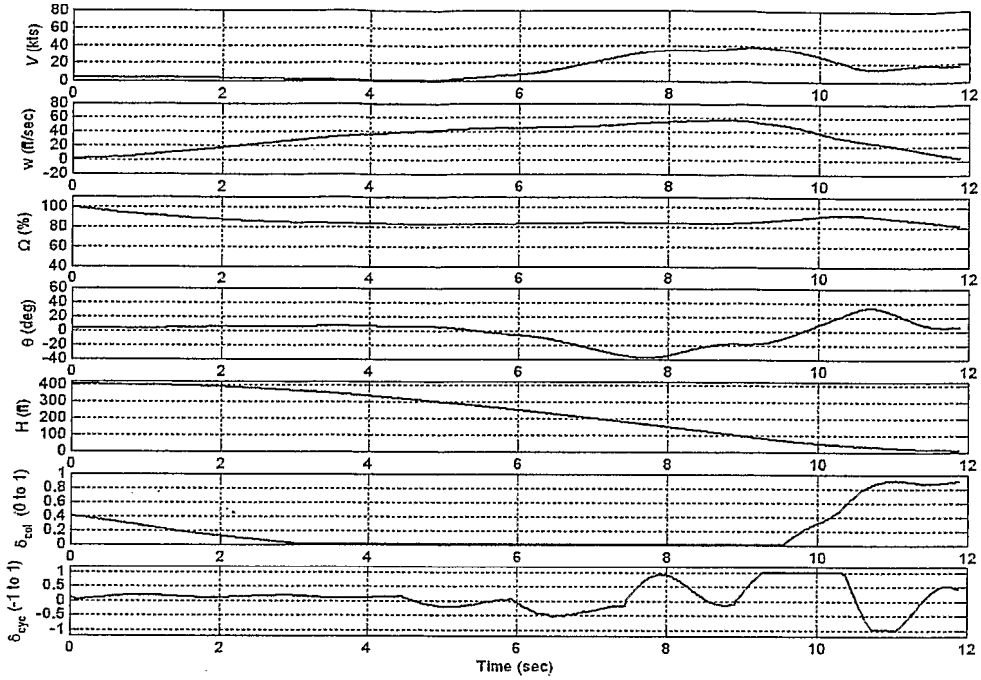


Figure 9. Automatic autorotation from 400ft/0kts; light weight condition (3100 lbs)

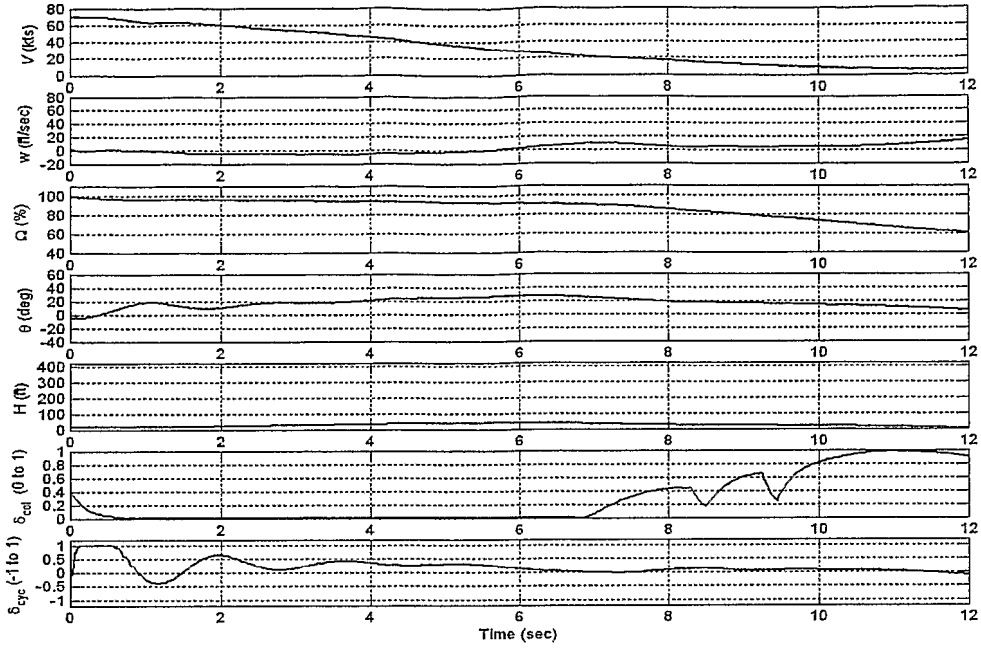


Figure 10. Automatic autorotation from 20ft/70kts; light weight condition (3085 lbs)

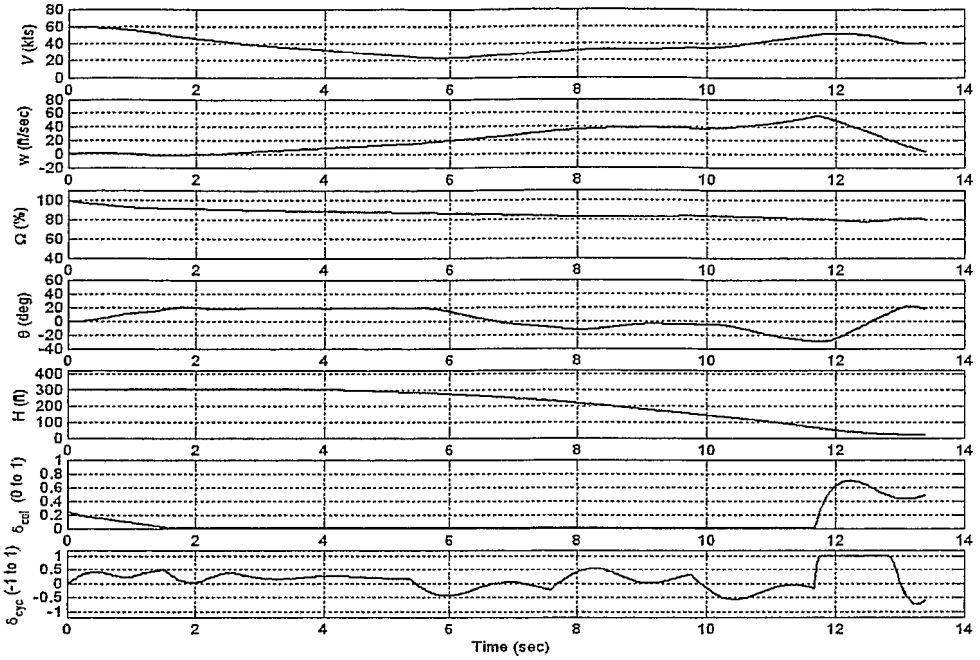


Figure 11. Automatic autorotation from 300ft/60kts; light weight condition (3085 lbs)

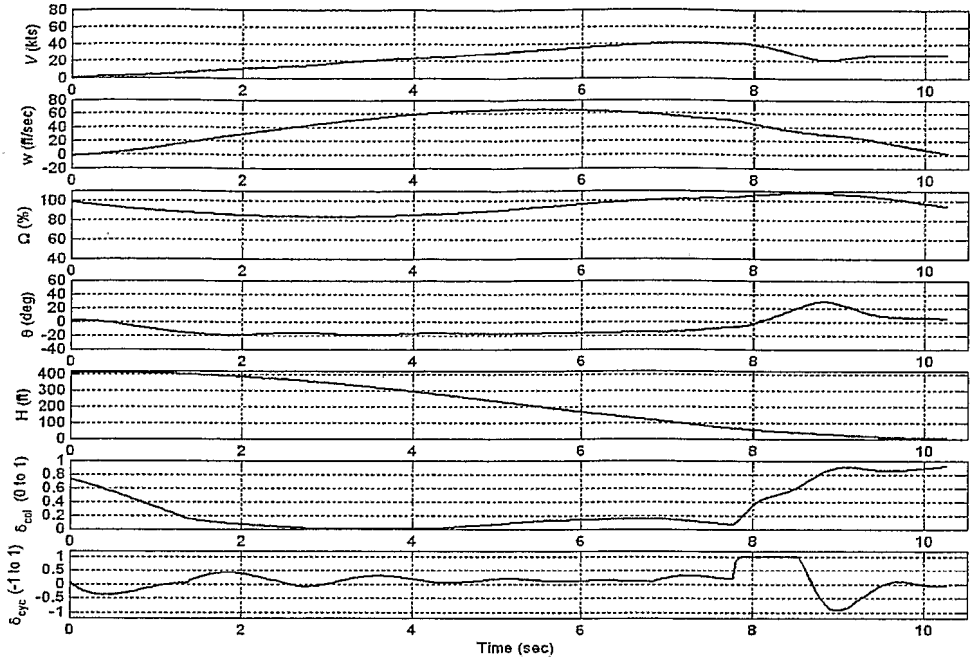


Figure 12. Automatic autorotation from 400ft/0kts; heavy weight condition (4440 lbs)

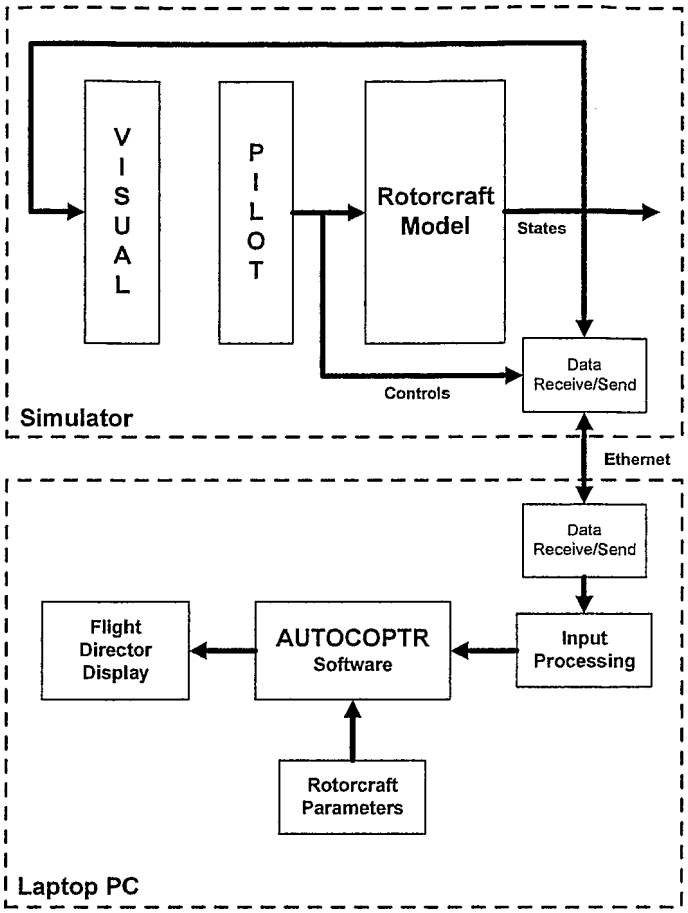

VISUAL
PILOT
Rotorcraft Model
States
Controls
Data Receive/Send
Simulator
Ethernet
Data Receive/Send
Flight Director Display
AUTOCOPTR Software
Input Processing
Rotorcraft Parameters
Laptop PC

Figure 13

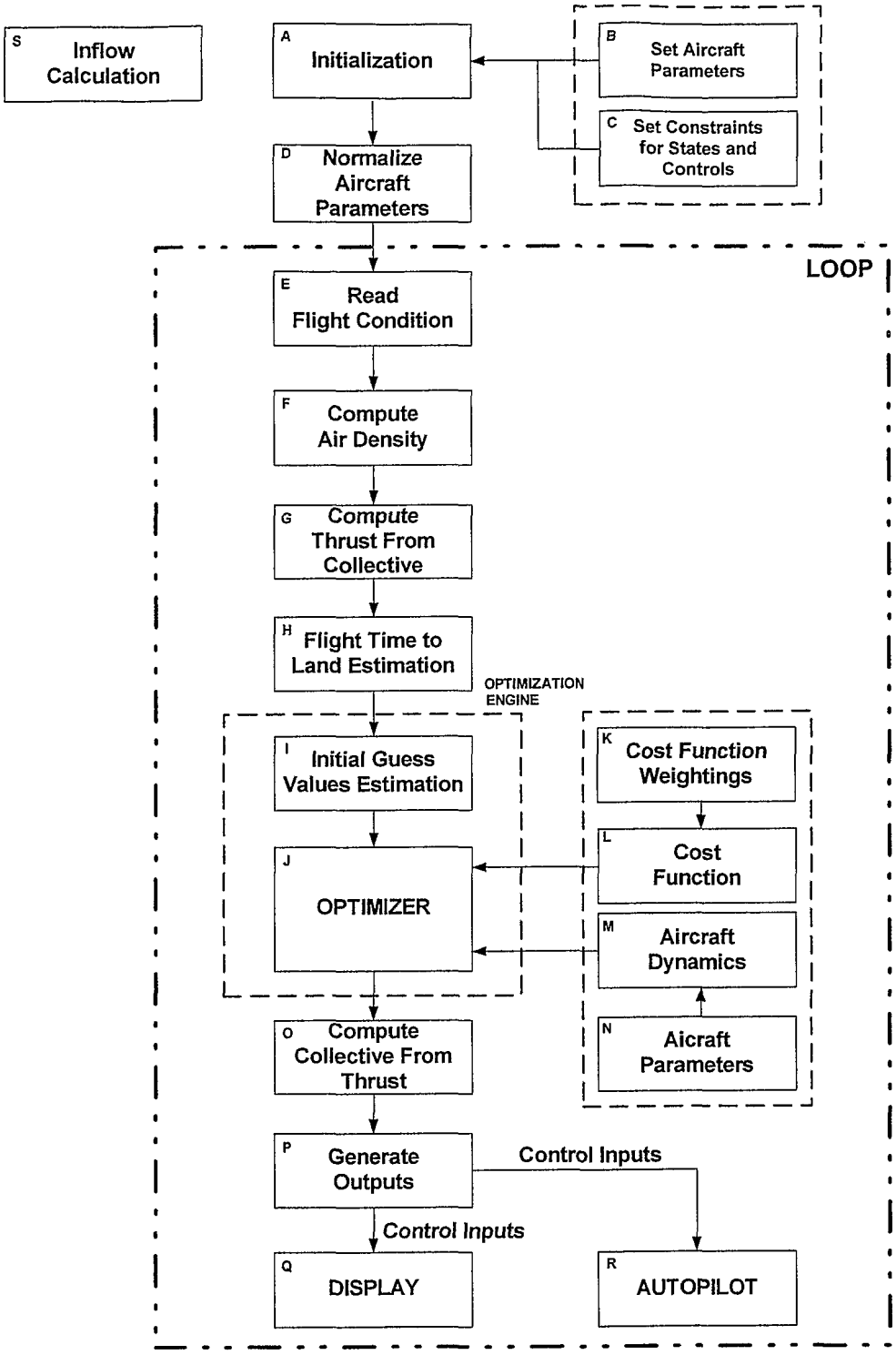

S
Inflow Calculation
A
Initialization
B
Set Aircraft Parameters
C
Set Constraints for States and Controls
D
Normalize Aircraft Parameters
LOOP
E
Read Flight Condition
F
Compute Air Density
G
Compute Thrust From Collective
H
Flight Time to Land Estimation
OPTIMIZATION ENGINE
I
Initial Guess Values Estimation
J
OPTIMIZER
K
Cost Function Weightings
L
Cost Function
M
Aircraft Dynamics
N
Aicraft Parameters
O
Compute Collective From Thrust
P
Generate Outputs
Control Inputs
Control Inputs
Q
DISPLAY
R
AUTOPILOT

Figure 14

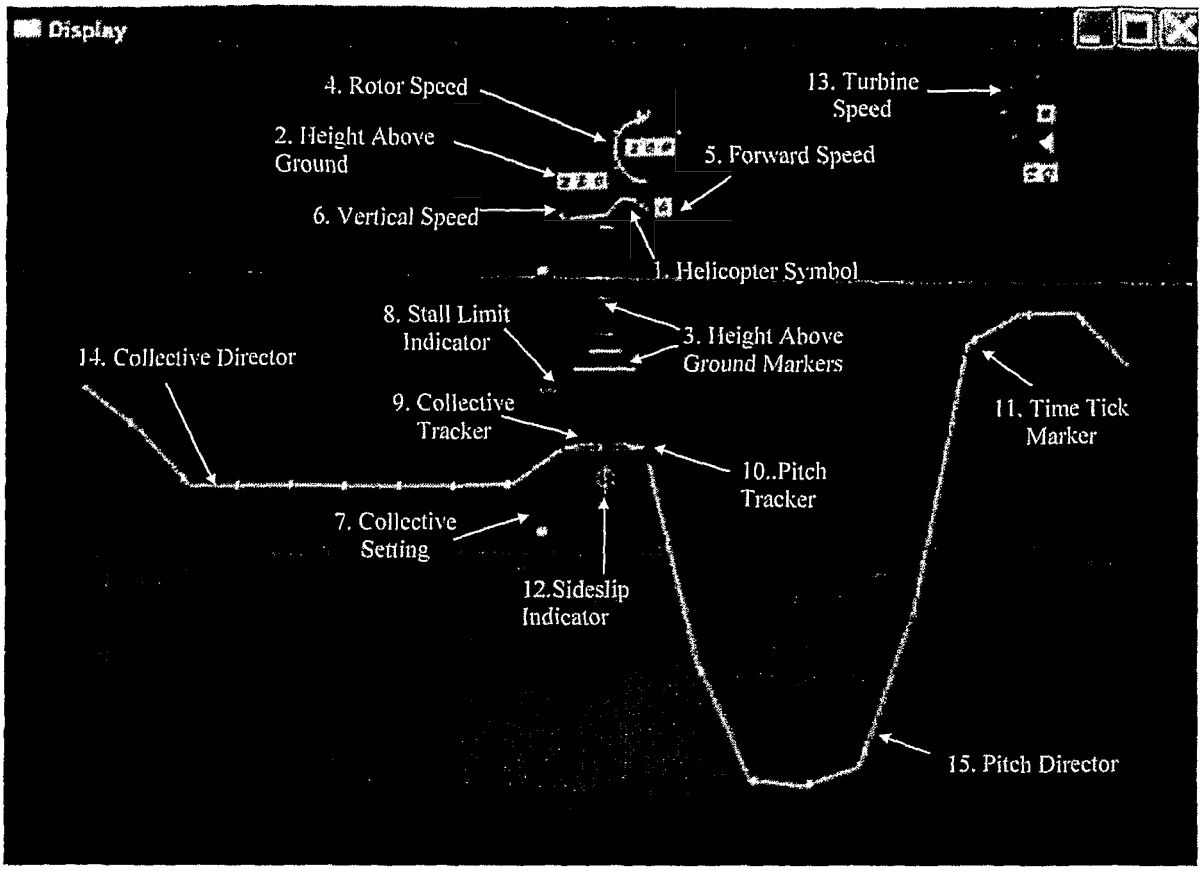

Display
4. Rotor Speed
13. Turbine Speed
2. Height Above Ground
5. Forward Speed
6. Vertical Speed
1. Helicopter Symbol
8. Stall Limit Indicator
3. Height Above Ground Markers
14. Collective Director
9. Collective Tracker
11. Time Tick Marker
10..Pitch Tracker
7. Collective Setting
12.Sideslip Indicator
15. Pitch Director

Figure 15

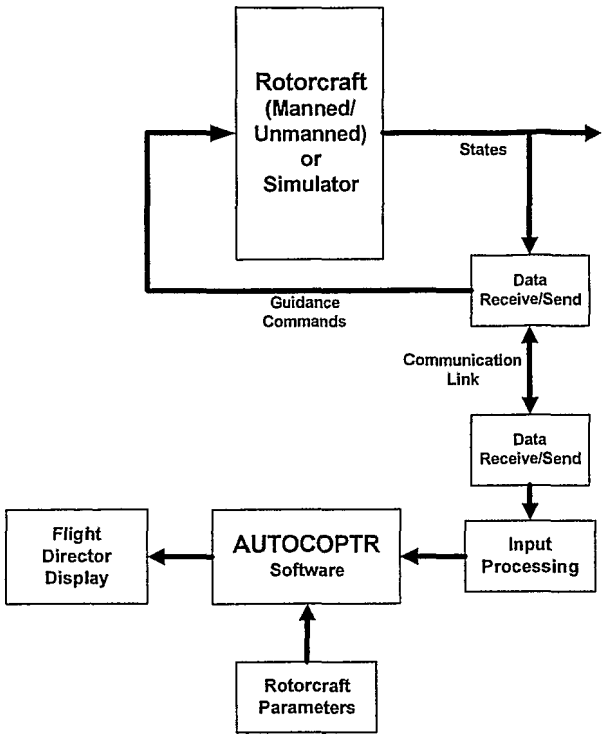

Rotorcraft (Manned/ Unmanned) or Simulator
States
Data Receive/Send
Guidance Commands
Communication Link
Data Receive/Send
Flight Director Display
AUTOCOPTR Software
Input Processing
Rotorcraft Parameters

Figure 16

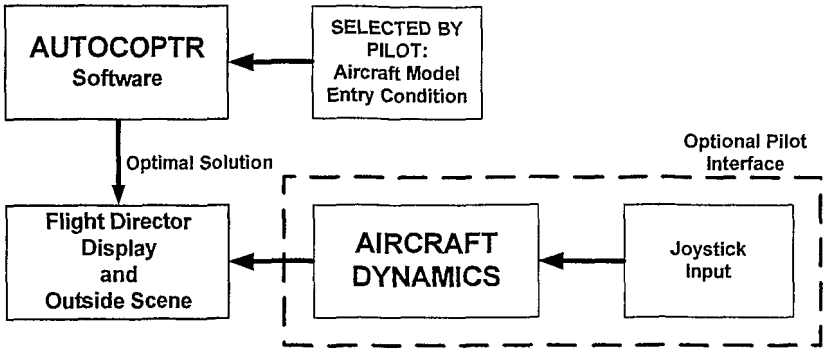

AUTOCOPTR
Software
SELECTED BY
PILOT:
Aircraft Model
Entry Condition
Optimal Solution
Optional Pilot
Interface
Flight Director
Display
and
Outside Scene
AIRCRAFT
DYNAMICS
Joystick
Input

Figure 17