LEARNING TO FLY: DEVELOPING AN AUTONOMOUS AERIAL VEHICLE USING HUMAN SKILL MODELING

By

STEPHEN B. STANCLIFF

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING

UNIVERSITY OF FLORIDA

2000

ACKNOWLEDGMENTS

The author would like to thank Dr. Michael Nechyba for his support of this research. Using his previous work on human skill modeling allowed the author to concentrate on the development of the hardware and onboard software rather than the neural network software.

Thanks also go to Jenny Laine, who provided some of the software used in the early stages of the project.

Special thanks go to Mr. Ray Helpling for sharing his knowledge and time as the pilot and primary source of expertise on remote-controlled airplanes.

The following companies have supported this research through donations or discounts: Analog Devices, Aveox, EMJ, Energizer, Maxim, Motorola.

TABLE OF CONTENTS

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Engineering

LEARNING TO FLY: DEVELOPING AN AUTONOMOUS AERIAL VEHICLE USING HUMAN SKILL MODELING

By

Stephen B. Stancliff

May 2000

Chairman: Dr. Michael C. Nechyba
Major Department: Electrical and Computer Engineering

In recent years much work has been done in the area of abstracting computational models of human control strategies. This type of modeling has been used successfully to create autonomous ground vehicles from observation of human drivers, both in simulation and on real roads. Little work has been done, however, in attempting such skill transfer from human pilots to autonomous aerial vehicles.

In this project a robotic airplane has been developed as a platform for studying human-to-machine skill transfer in aerial vehicles. This platform is capable of recording the control actions of a human pilot along with data from onboard sensors. These data have then been used to develop models of the human pilot's control strategies which will enable the airplane to fly autonomously.

The general scheme followed in developing a model of pilot control schemes is as follows: (1) Data representing the human control inputs are collected, along with sensor

data representing the state of the system at that point in time; (2) After preprocessing, this data is used to train a cascade neural network; (3) The trained network is then used as an autonomous controller.

At the time of this writing, the basic platform has been completed and an electronics suite sufficient for initial experiments has been integrated into the platform. Initial experiments have shown that, using the method described in this paper, a model can be created which accurately predicts the next command of the human pilot given past command and current and past sensor data. Additionally, the software and hardware necessary to proceed with the next set of experiments--flying autonomously--has been completed.

CHAPTER 1
INTRODUCTION

1.1 Background

Humans are much better at performing complex dynamic skills than at describing those skills in an algorithmic, machine-codeable way. This has limited our ability to develop intelligence in robots and other machines. This inability has limited not only the capabilities of individual machines, but also the extent to which humans and robots can work cooperatively. As a result there is a strong need to find ways to encapsulate human skills into models which can be used as control systems for robots and other machines.

One method of human skill modeling which has proven successful is the use of neural networks to develop a mapping between sensor inputs and human control outputs. Autonomous control and navigation of ground vehicles is one area of robotics research which has benefited from this type of modeling. Pomerleau, for example, has implemented in the ALVINN system real-time road following using data collected from a human driver [1,2]. The ALVINN system has learned to map from coarse video images of the road to a desired steering angle and has been demonstrated successfully on public roads at speeds up to 70 mph. Nechyba and Xu have used observation and modeling of human drivers within a driving simulator to successfully create autonomous control systems for both steering and acceleration [3,4,5].

Little previous work has been done, however, in using observation and modeling of human pilots to create intelligent autonomous aerial vehicles. This is surprising, since an intelligent autonomous aerial vehicle would have application in many areas. For instance, many of the activities that currently involve remotely piloted vehicles (RPVs) would benefit in some way from automation.

In some simpler RPV applications, such as surveying, reconnaissance, and target acquisition, adding intelligent control systems may enable the automation of the entire mission. For other applications, where more sophisticated control is required, it may be that a human pilot is still needed, but that the less difficult parts of the mission can be automated. Adding intelligence to RPVs could reduce personnel costs by reducing the amount of skill required of the human pilots and also by allowing a single pilot to control multiple vehicles.

Another application for human control modeling in aerial vehicles is in the area of pilot training. The training of novice pilots is an expensive, time-consuming, and in some cases dangerous process. It should be possible to use human skill models derived from observation of expert pilots to develop computer instructors which can then be used both to accelerate the learning process and also to help avoid costly student mistakes.

Automating these tasks is a job which can be readily accomplished using traditional control system methods. The method proposed here, however, has some significant advantages over traditional methods. First, the proposed method can be expected to save many hours of engineering labor, and thus much expense, in comparison with traditional methods. Traditional control system design methods require the development of mathematical models of the systems involved. In the case of

aerodynamic systems, these models are generally developed experimentally, requiring much engineering time and expertise. In contrast, the generation of a control system through human skill modeling requires only a rudimentary understanding of the system being controlled--enough to ensure that adequate sensor and human control information is being collected.

Finally, it is possible that human skill modeling, in combination with other methods, will produce control systems which are more robust than those produced by traditional control system methods and that it will allow the control of complex flight regimes, such as combat, which are not controllable using traditional methods. The reason for this is that the systems being controlled are fairly to highly nonlinear in nature. Traditional control system design methods generally operate under the assumption of local linearity, use a linear model of the system to be controlled, and generate a linear control system. The proposed method makes use of neural networks, which are nonlinear function approximators, resulting in a nonlinear control system when the control behavior exhibited by the human pilot is nonlinear.

1.2 Research Goals

Although the control challenges for an aerial vehicle are different from those for a ground vehicle, the basic paradigm of learning from humans is equally applicable. This research is therefore an attempt to extend to aerial vehicles some of the methods previously used for learning in ground vehicles.

This research is intended to be a preliminary study to demonstrate the feasibility of this modeling technique. As such the important results will not be the actual

autonomous flying behaviors demonstrated, but rather the demonstration that human skill modeling is a viable alternative to more traditional control system techniques in the area of aerial vehicles.

The goal of this research project was to develop an aerial robotic platform for use in human skill modeling experiments. The specific design goals for the platform were (1) to carry a payload of at least 3.5 lb, (2) to provide a time at altitude of at least five minutes, (3) to record attitude data from onboard sensors, and (4) to interface with the radio control system in order to record pilot controls and to output computer control commands.

1.3 Organization of Thesis

This thesis is organized as follows. **Chapter 2** discusses the development of the electrical and mechanical hardware comprising the robotic platform. **Chapter 3** describes the stages in the experimental process and the software developed for each of these stages. Finally, **Chapter 4** gives an overview of the experimental results achieved so far and a discussion of future goals for the project.

CHAPTER 2
HARDWARE

2.1 Airframe

Selection and Description

One of the first decisions to be made in this project was the selection of a suitable airframe as a basis for the experimental platform. Consideration was given to both fixed-wing and rotary-wing platforms. Each type of platform has advantages and disadvantages for the missions which are planned as part of this research. In general, a fixed-wing aircraft will have a greater payload-range, be more stable, and have fewer and simpler control mechanisms than a rotary-wing aircraft. Consequently, a fixed-wing aircraft requires a much less expensive radio system and less skill from the human operator. A rotary-wing aircraft, on the other hand, has the ability to hover, which can be useful for photography or surveillance, and it can operate from smaller landing areas. For the purposes of this research, the factors of cost, simplicity, stability, and payload capacity were deemed the most important, and they led to the selection of a fixed-wing platform.

The Avigator platform (**Figure 2.1**) is based on a radio-controlled (R/C) airplane kit, the Sig *Kadet Senior*. This particular airframe was chosen because of its slow and stable flying characteristics, its lightweight construction, and its large size (78" wingspan, 1150 in^2 wing area), the latter giving a large payload capacity, both in volume and in weight. The payload compartment is roughly 8" x 4.5" x 14.5", although some of that

space is taken up by the radio control equipment. The drawbacks to this airframe are that it is relatively difficult to assemble and it is relatively fragile. Both of these problems are consequences of its lightweight stick-frame construction.

Figure 2.1 - The Avigator platform

The airframe has a wing area of 1150 in^2 and an empty weight of about 6 lbs, giving a wing loading of about 12 oz/ft^2. A radio-controlled airplane is still considered lightly loaded at 20 oz/ft^2, and a highly stable airplane such as the *Kadet Senior* can be expected to exhibit good handling qualities with a wing loading as high as 30 oz/ft^2. In the case of the Avigator platform, a wing loading of 30 oz/ft^2 corresponds to a payload of about 9 lb. In payload-capacity tests, the addition of 7 lbs of payload caused no detrimental effects on the airplane's flying characteristics. The usable internal volume is currently a more limiting factor than the weight-carrying capacity, although an external payload compartment could be added in order to increase the payload volume.

Modifications

During construction, the airframe was modified in many small ways to accommodate the needs of this project. The structure of the fuselage beneath the wing was modified in order to maximize the volume and accessibility of the payload compartment (**Figure 2.2**). An extra servo was used to control the nosewheel, rather than the usual coupling of nosewheel and rudder, in order to eliminate the need for control rods running through the payload compartment. The nosewheel and throttle servos were moved forward into the nose, and the rudder and elevator servos were moved to the very rear of the payload compartment, again to maximize payload volume and accessibility. Crossmembers have been added to the payload compartment as needed for mounting of electronic components.

The platform was originally powered electrically (section **2.2**), requiring many modifications from the kit, which was designed for power by an internal combustion engine. In order to reduce the weight of the airframe, thinner wood was substituted in

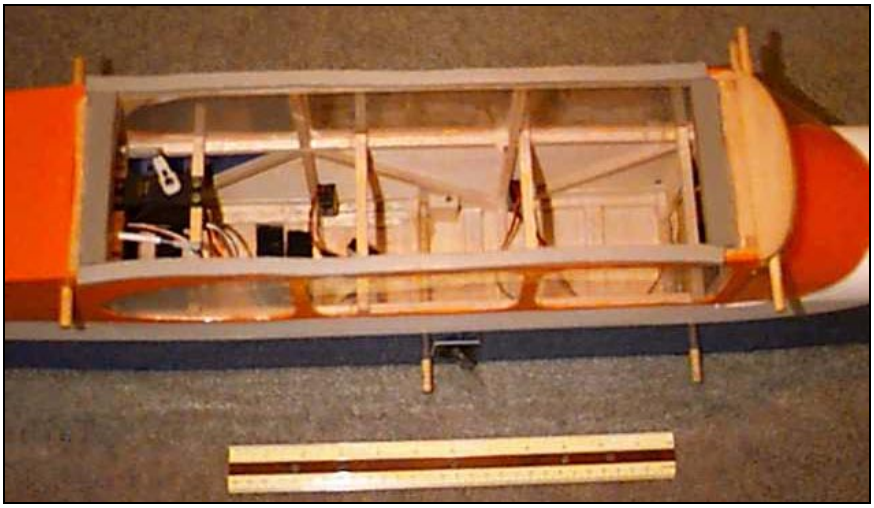

Figure 2.2 - Payload compartment

many places, most notably the sheeting covering the nose and fuselage sides, and the firewall and its supports. Lightweight components were substituted where possible, such as foam wheels instead of rubber and nylon pushrods instead of balsa.

The area behind the firewall was modified to accommodate the motor controller, and holes were cut in the firewall for the motor wiring to pass and to allow cooling air to flow over the motor and controller. A large hole in the rear of the fuselage behind the payload compartment served as an exit for this airflow. Dowels were added to the bottom of the fuselage to allow for mounting the battery pack externally using rubber bands. This method of mounting the large, heavy battery pack was chosen both to preserve space in the payload compartment and also so that, in the event of a crash, the battery pack could exit the airplane with a minimum of damage to the structure.

2.2 Propulsion

Initial Configuration

The usual powerplant for small R/C airplanes is a two-cycle internal combustion engine. These engines are highly reliable and have a high power-to-weight ratio. However, they also have characteristics which are detrimental to the reliable operation of electronic equipment, namely significant vibration and the presence of caustic fuel and exhaust. In recent years, with the advent of advanced battery technologies, electric motors have begun to make a significant inroads into R/C airplanes. In addition to the absence of vibration, fuel, and exhaust, an electric motor also offers low noise and a lower operating cost. The disadvantages of an electric motor include lower power-to-weight ratio, shorter flight times for similar performance, and a higher initial cost.

The desire to achieve reliability without the added weight and complexity involved in vibration-mounting and sealing against fuel and exhaust, as well as a desire to test the state of the art, led to the selection of an electric propulsion system.

There are both brushless and brushed motors designed for this application. The brushless motors generally have a higher peak efficiency than the brushed motors, and they have very high efficiency over a wide range of operating speeds, rather than the sharply peaked efficiency curves of the brushed motors. In addition, the absence of brushes reduces the amount of electromagnetic interference produced and reduces the need for periodic rebuilding of the motor. Brushed motors, on the other hand, are both cheaper themselves and also allow the use of simpler motor-speed controllers, resulting in a significantly lower startup cost. Again, the desires for reliability without adding extra shielding from electromagnetic interference as well as a desire to test the state of the art led to the selection of a brushless motor.

A final decision to be made in the initial powerplant configuration was the choice of battery chemistry. The technologies considered were nickel-cadmium (NiCd), nickel-metal-hydride (NiMH), and lithium-ion (Li+). NiCd is the most common battery chemistry in use today, including in the area of R/C airplanes. NiMH batteries have a power to weight ratio which is typically 1.5 times that of NiCd and are common in high-drain devices, such as laptop computers and power tools. Li+ batteries are a relatively new technology, have a power to weight ratio which is typically 3 times that of NiCd, and are mainly found in high-power, low-current applications such as laptop computers.

While the Li+ batteries are desirable due to their high power density, they are not capable of sourcing the high currents (8-10 C[1]) required by this application. As a result, NiMH batteries of a type designed for use in power tools were chosen. The cells used provide 2.2 Ah of capacity in a standard sub-C cell size and weigh 2.0 oz, for a power density of 21 Wh/lb. This number is on the low end of the NiMH range, which is a consequence of their ability to handle high discharge rates.

Electric Propulsion System Design

Once the motor type and battery specifications had been established, it was necessary to design the propulsion system to minimize energy losses in order to provide the longest possible flying time for the 3.5 lb payload. Several more-or-less scientific methods exist in the electric R/C community for this design process. Preliminary calculations indicated that the various methods produce similar results. The method used is a combination of the method outlined by Orme [**6**] and the manual for the software MotoCalc [**7**], which is a design tool that simplifies the repetitive calculations involved.

The first step in the design process was to estimate the number of sub-C cells required. Orme has a rule of thumb that a large, slow trainer-type aircraft needs one cell per 50 in^2 of wing area [**6**]. This rule generally results in a wing loading of about 20 oz/ft^2. Applying this rule to the Avigator airframe gave a requirement of about 24 cells (28.8 V). Using this estimated cell count, several motors were selected for evaluation, based on their voltage ratings. Preference was given to motors which had been flown by others in the *Kadet Senior*.

[1] C is the one-hour current capacity of the cell.

Design from this point was performed using the MotoCalc program. MotoCalc includes a database of characteristics of common motors, motor controllers, batteries, and airframes. It uses this information to calculate power usage, efficiencies, and flight envelope. All of the motors and controllers which were considered for this project were found in the MotoCalc database. The specifications for the NiMH cells were entered into the program's database. The *Kadet Senior* airframe was also not found in the database, but the unknown aerodynamic data for it were easily estimated from similar airframes in the database. These estimates were later refined by direct measurement of related aircraft dimensions. The weight of the payload was incorporated into the MotoCalc design by including it in the empty weight of the airframe.

The first step in designing with MotoCalc is to determine the most efficient current for a given motor and number of cells. The program does this by calculating the losses in the motor and wiring at different operating currents. For the motor that was eventually selected for the Avigator platform (Aveox 1406/4YSE), the peak motor efficiency with 24 cells occurs at about 18 A (**Figure 2.3**) and is about 89%. The

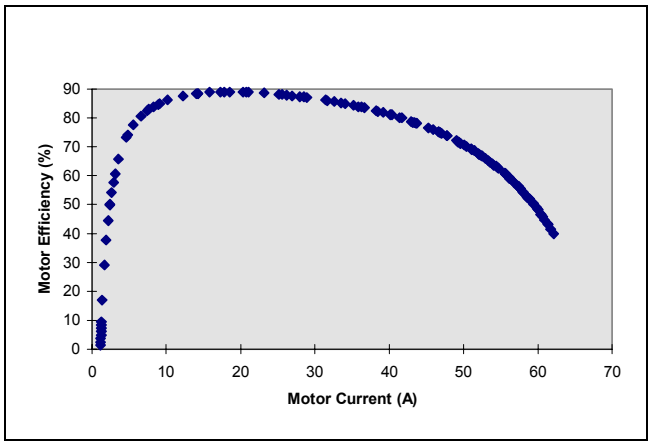

Figure 2.3 - Motor efficiency curve

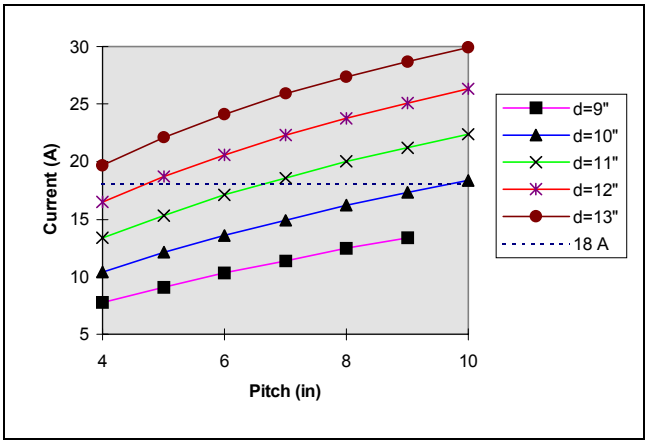

Figure 2.4 - Propeller selection

efficiency is greater than 88% for currents from 14 A to 25 A (a characteristic of brushless motors), allowing some freedom in the selection of propeller, cells, and gearbox.

At the most efficient current of 18 A, this motor will turn at about 33,000 rpm. The propeller should normally spin at 8,000-12,000 rpm at full throttle, so a gearbox was necessary. A gearbox with a ratio of 3.69:1 was chosen. The calculations were repeated with this gearbox and for a range of propeller sizes (**Figure 2.4**). The two propeller sizes which came closest to pulling the desired current of 18 A, while being able to maintain flight, were 11x7[2] and 12x6.

Performance predictions given by MotoCalc for the two selected propeller sizes are given in **Table 2.1**. By examining the last two rows of this table, initial rate of climb and flight duration, one can see that the 11x7 propeller will provide greater power for a shorter period of time.

[2] diameter (in) x pitch (in)

The flight durations given in **Table 2.1** are upper limits and cannot be reached in reality. They are based on level flight at a partial-throttle setting (in this case about 89%) and therefore do not account for the energy required for takeoff and climb to altitude. The worst-case duration, based on constant 100% throttle, was also calculated and is about four minutes for each propeller. These upper and lower bounds led to the conclusion that a 5-10 minute flight duration would be achievable.

Table 2.1 - Propeller performance

Prop size	11 x 7	12 x 6
Motor current (A)	18.6	20.6
Motor efficiency (%)	89.0	88.9
Prop speed (rpm)	8,853	8,489
Stall speed (mph)	21	21
Max. level speed (mph)	43	41
Rate of climb (fpm)	359	342
Duration (min.)	14:26	16:41

In actual use, the electric propulsion system consistently provided flights of 8-10 minutes with a 3-4 lb payload, thus meeting the design goals for payload and duration. As predicted, the 12x6 propeller provided slightly better flight duration than the 11x7, while the 11x7 provided slightly higher top speed.

Unfortunately, the components of the electric drivetrain proved highly unreliable, resulting in a lack of flying time and a lack of repeatability. After a few months of experimenting with the electrical propulsion system, it was decided to rebuild the platform with an internal combustion engine.

Internal Combustion Engine

As mentioned previously, most R/C aircraft are propelled by small two-cycle internal combustion engines. In recent years, four-cycle engines have become popular because they have lower noise, less vibration, greater reliability and more consistent performance. The downside of four-cycle engines is that they produce less power per pound and much less power per dollar in comparison to two-cycle engines. Low vibration was a priority for this platform, so a four-cycle engine was chosen. The engine chosen (O.S. FS52) has a displacement of 0.52 in^3 and is rated at 0.9 hp.

Installation of the new engine required significant modification to the airframe. Everything from the firewall forward had to be replaced, and some of the lightening that had been done previously had to be undone in order to strengthen the structure to withstand the vibration of the engine. With the elimination of the heavy battery pack which had been previously located forward of the center of gravity, it was expected that the center of gravity would shift too far rearward for stable flight, and additional changes were made to offset this, including moving the throttle and nosewheel servos farther forward in the nose. Finally, the various air holes and access hatches were eliminated and the space between the wing and fuselage sealed with foam tape, in order to keep fuel and exhaust from reaching the payload compartment. An additional step in this regard was the fabrication of an exhaust extension out of copper tubing, which directs the engine's exhaust past the payload compartment and downward. This latter addition has proven quite successful in use, eliminating most of the oily residue which usually covers R/C airplanes after flight.

With the new engine in place, the takeoff performance, payload capacity, and flight duration of the platform were significantly improved. Although the takeoff weight of the platform dropped by less than a pound compared with the electric propulsion system, the takeoff run required on a calm day with a 2 lb payload dropped from about 50 ft to about 30 ft. This performance was not noticeably changed when tested with an additional 5 lb of payload. In addition, the new engine provides flying times of 20+ minutes on a full tank of fuel with 3-4 lb of payload.

The new engine brought many difficulties in the form of vibration. With the electric motor, the sensors and computers had been hard-mounted to the airframe. With the new engine running, these components exhibited erratic behavior, in spite of the isolation of the engine from the airframe by rubber mounts. Several methods of soft-mounting the electronic components were tried. For most of the components, a sandwich of soft foam between plywood mounts (**Figure 2.5**) provided adequate damping. For the computer's hard drive and the accelerometer-based tilt sensor, however, no adequate

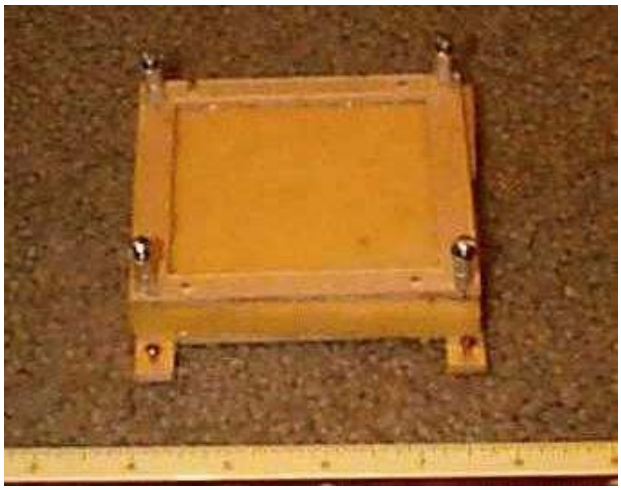

Figure 2.5 - Vibration reducing mount

vibration-reduction method has yet been found. As a result, these components have been left off, and the platform currently relies on a small solid-state disk drive and on the tilt sensor built into the compass.

2.3 Electronics

Radio-Control System

The transmitter used is a standard six-channel R/C airplane radio. The standard radio receiver has been modified to provide connections to the HC11 daughterboard (**p. 22**) for decoding of the pilot's control signals. Four standard R/C servos are used to control the throttle, nosewheel, rudder, and elevator. The fifth radio channel is a toggle switch and is used as input to the control multiplexing system (**p. 21**) to select between human and computer control of the aircraft. The sixth channel is a rotary knob and is currently unused.

Sensors

The platform currently implements detection of angular position about the three axes using a Precision Navigation TCM2 compass module (**Figure 2.6**). This module provides tilt-compensated heading output at tilt angles up to 45° through three magnetometers and a two-axis liquid-level tilt sensor. The output from the tilt sensor is used for measurement of pitch and roll angles. This module provides output as an ASCII message through an RS-232 interface, and it provides heading resolution of ±1.5° and tilt resolution of ±0.4°. The TCM2 module measures about 2.5 in x 2 in by 1.25 in and weighs about 2 oz.

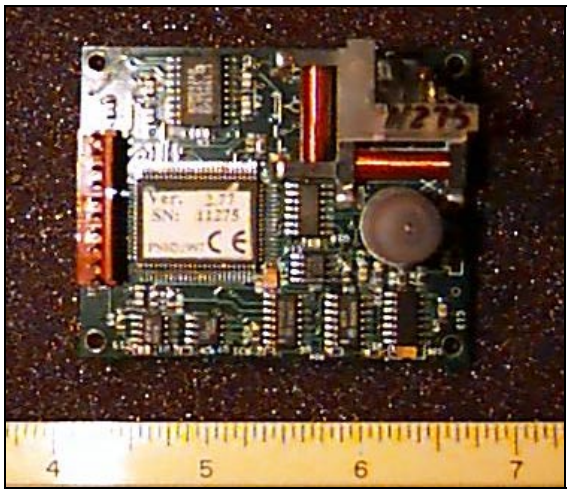

Figure 2.6 - TCM2 compass module

The liquid-level tilt sensor in the TCM2 module is ill-suited for a highly dynamic environment such as an airplane, but it has proven adequate for the early experiments in straight-and-level flight and constant-bank-angle turning. The limit of 45° bank angles as well as the poor dynamic response of this sensor have required a modification in the flying habits of the human pilots, as radio-control pilots generally execute turns in a much more extreme manner than is possible in manned aircraft.

A tilt sensor was developed based on the Analog Devices ADXL202 two-axis accelerometer. This device provides a duty-cycle-modulated output proportional to acceleration and reads ±2 g. A custom circuit board was fabricated for this circuit. The completed board (**Figure 2.7**) is about 1" square, weighs less than 0.2 oz, and provides a resolution (when decoded by the HC11) of less than one degree over a ± 90° tilt range.

This sensor proved quite reliable on the electric-powered airplane, and its small size and weight are perfectly suited for a small aerial platform. Unfortunately, this sensor is highly sensitive to vibration and has been unusable with the current platform because

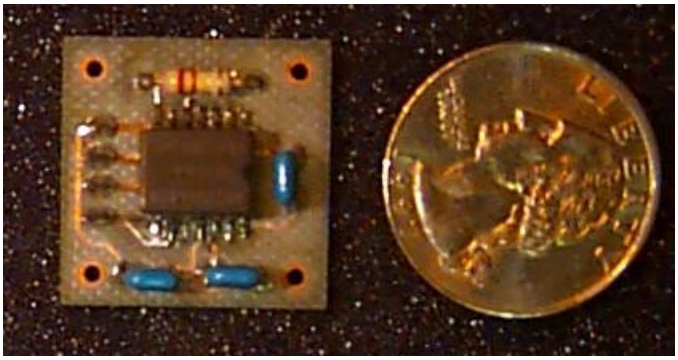

Figure 2.7 - Tilt sensor

of the large amount of vibration present. Construction of this tilt sensor is described in the datasheet for the ADXL202 [**8**].

Computers

The Avigator platform has two onboard computers, a Motorola MC68HC11 microcontroller for input/output (I/O) functions and a 386-class microcomputer for high-level processing and data storage. The HC11 is on a Mekatronix MRC11 board (**Figure 2.8**), which provides the HC11 in expanded mode allowing use of external memory. In the Avigator platform, this board is configured to provide 32k of SRAM and 32k of EPROM. The MRC11 board also provides a header for easy access to the HC11's I/O ports. This board is responsible for the capture of all sensor and control data except the compass data. The HC11 transmits this information to the 386 computer through an RS-232 serial connection.

The microcomputer used (**Figure 2.9**) is a 386SX-compatible processor on a JUMPtec MOPS LCD3 PC/104-format board. This board provides all of the features usually found in a desktop computer, including video and ethernet, on a 3.6" x 3.8" board

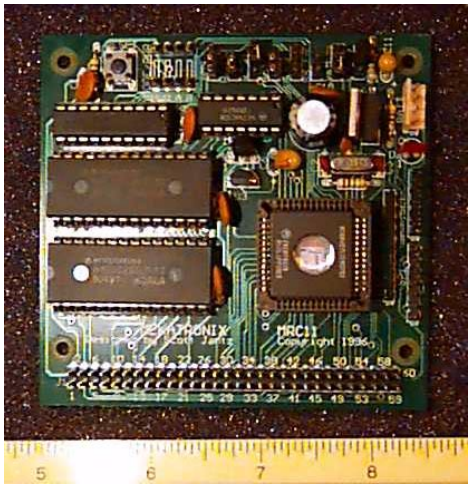

Figure 2.8 - MRC11

weighing less than 4 oz. The standard PC/104 bus will simplify future expansion by allowing the use of off-the-shelf daughterboards. A 386-class processor was chosen over a 486 or 586 primarily due to its much lower power consumption, which results in a much smaller battery pack required, leaving more volume and weight capacity available for electronics.

Figure 2.9 - MOPS LCD3

The 386 is responsible for interfacing with both the HC11 and the compass through its two RS-232 interfaces. Storage options include a floppy drive interface, a small (0.7 MB) built-in flash disk, and an IDE interface which can be used with a conventional hard drive or a flash disk. The user interface can be provided through the keyboard and VGA ports, through an ethernet connection to another computer, or through a parallel-port connection to another computer.

For most in-lab development and testing, a keyboard and monitor have been connected to the PC/104 board in order to work directly with it. Most of the field work has been done through a parallel-port connection from a laptop computer. Recently a small, low-power VGA monitor has been acquired which can be powered from a car battery, allowing the operator to interface directly with the 386 computer at the field, without an intermediary computer. This has improved ease of use in the field considerably, especially when debugging.

During the period when the platform was propelled electrically, a standard laptop hard drive was used for program and data storage. When the platform was converted to internal combustion power, the hard drive proved to be highly sensitive to the vibrations produced by the engine. No suitable method of damping has been found, so this component has been left off of the current platform. In order to carry out future experiments with a camera, a suitable method for achieving a large storage capacity will be needed. It is possible that a damping system can be found to use with the current hard drive, or that a hard drive more resistant to vibration can be employed. One possibility is that a faster-spinning hard drive may be more suitable, since informal experiments have

shown a relationship between the speed of the hard drive and the engine speeds at which failures occur, indicating that resonance may be a factor.

For many of the experiments since the conversion to engine power, data storage has consisted solely of the integrated 0.7 MB solid-state drive. This drive provides enough room for about 10 minutes of data, requiring that the wing be removed and the data downloaded from the onboard computer after each flight. Recently a 64 MB solid-state drive has been added to the platform, allowing the plane to be flown for a much longer time period without having to remove the wing and download the data.

Control Multiplexing System

Allowing a computer to have control of a powered aerial vehicle entails some considerable risk to life and property. It is therefore necessary that a human safety pilot be able to take control of the airplane from the computer at any time. A circuit (**Figure 2.10**) has been designed and implemented which enables this control by means of the fifth channel toggle switch on the radio transmitter. This circuit consists of three parts. The first part is a simple RC filter which converts the PWM signal from the receiver's fifth channel into a nearly constant voltage. The filter was designed such that the two positions of the toggle switch are converted into voltages of approximately 1.5 V and 2.5V.

The second part of the circuit is a comparator, which outputs a 1 if the input voltage is greater than approximately 2 volts and a 0 if it is less. The output from this second stage is used as the input to an 8:4 multiplexer, which has as one set of inputs the safety pilot's commands from the radio receiver and as the other set the computer's

commands from the HC11. This circuit draws its power from the radio receiver so that, in the event of a complete computer system or computer battery failure, the safety pilot can still take control of the airplane.

Figure 2.10 additionally shows the schematic for an RS-232 transceiver constructed around the Maxim MAX233 IC, which converts between the TTL-level serial signals of the HC11 and the RS-232 level serial signals of the 386. The MAX233 chip is convenient because it provides a single-chip transceiver solution.

A custom daughterboard (**Figure 2.11**) was built to mate with the MRC11, providing a convenient method for connecting the devices which must interface with the

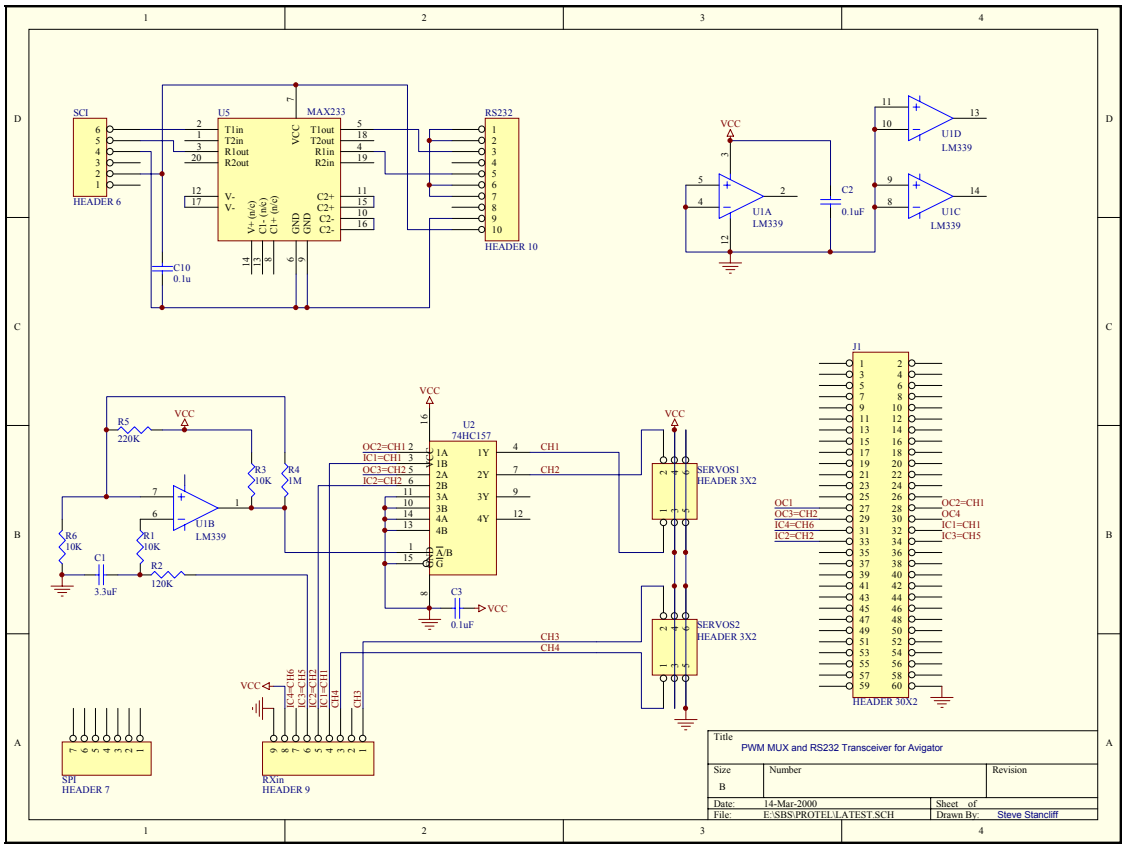

Figure 2.10 - RS-232 transceiver (top) and signal multiplexer (bottom)

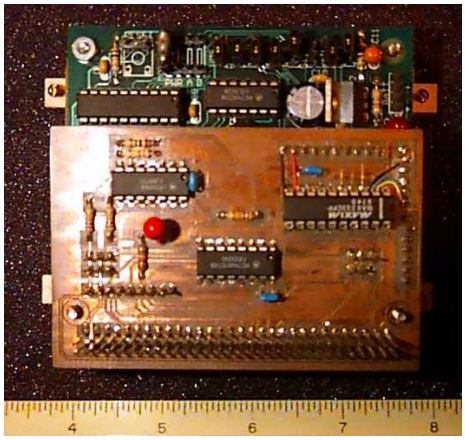

Figure 2.11 - MRC11 daughterboard

HC11. In addition to the control multiplexing circuit and the RS-232 transceiver circuit, this board provides a 10-pin RS-232 header for connection to the 386 computer, a 9 pin header for signal and power input from the radio receiver, four 3-pin servo headers for output signals to the servos, and a passthrough connector for the MRC11 header.

Figure 2.12 provides an overview of the electronic components and their interconnections.

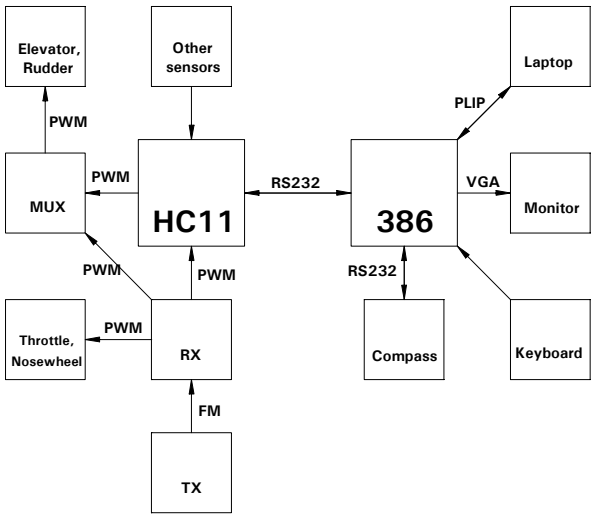

Figure 2.12 – System interconnections.

CHAPTER 3
SOFTWARE AND TRAINING

3.1 Overview

The general scheme followed in this project to develop a human skill model is as follows: (1) Data representing the human control inputs are collected, along with sensor data representing the state of the system at that point in time; (2) After preprocessing, this data is used to train a cascade neural network; (3) The trained neural network is then used to control the airplane.

3.2 Data Collection

During the data collection phase, the HC11 is responsible for capturing sensor data and also for decoding the pulse-width-modulated (PWM) signals from the radio receiver. Currently no sensors are connected to the HC11, although it was previously used to decode the accelerometer-based tilt sensor. The HC11 provides analog and digital input capability for future addition of sensors.

Decoding of the PWM signals is accomplished using the 'input capture' function of the HC11. In order to capture the PWM signals, an interrupt is set to occur when a rising edge occurs. During the interrupt routine for that rising edge, the interrupt is changed to occur on a falling edge. Subtracting the times at which these two interrupt events occur gives the pulse width. All input data is transmitted to the 386 through the

serial interface as ASCII data in a packet with a header and a checksum. All of the code for the HC11 (**Appendix B.1**) was written in C using the Imagecraft ICC11 compiler.

During data collection, the 386 sends requests for data to the compass and HC11 through it's two serial ports, receives the data, timestamps the data, and saves it to disk. Initially, efforts were made to synchronize the two data streams, but this resulted in a significant reduction in the overall sampling rate, since errors in one data stream would prevent saving of data from the other stream and also because the output rate of the compass is low compared to that of the HC11. Synchronization of the two streams was deemed unnecessary once it was realized that the data must be resampled during preprocessing to accommodate sampling period irregularities (**p. 26**). The data are saved to disk as a series of comma-separated numbers representing heading, pitch, roll, rudder control position and elevator control position.

All of the code for the 386 computer (**Appendix B.2**) was written in C using Borland C under MS-DOS. Linux was also tested as an operating system. Linux provides two significant advantages over MS-DOS: (1) a higher sampling rate and (2) the ability to log in remotely from another computer and observe running processes. The drawbacks of Linux are that it takes a long time to boot on this computer and that it requires significantly more disk space than MS-DOS (50 MB vs. 0.1 MB). Since most of the experiments to date have been run using only the 0.7 MB solid state disk for storage, the only operating system option has been MS-DOS. With the new 64 MB solid-state disk, Linux may again become a viable option.

3.3 Data Preprocessing

Data preprocessing and training are conducted offboard using a pentium-class computer. The first step in data processing is to manually select sections of the data which are representative of the behavior to be learned (e.g., 'flying straight-and-level'). This is a subjective task, since for dynamic real-world data it is often hard to say exactly where 'straight' ends and 'turn' begins. This step is performed using a spreadsheet program.

After the data segments representing the desired behavior have been selected, the data are preprocessed in the following steps: (1) The data are resampled to constant timesteps of 100 ms. The raw data, even without dropouts, have varying timesteps, due in part to the lack of synchronization of the two data streams and in part to the use of the solid state disk, which causes periodic system delays while writing data. Any time period of less than 300 ms is considered to be a valid interval, and the data is resampled. A gap in the data of greater than 300 ms is considered a break in the data stream, and the data following the gap is considered a new data segment. (2) Change in heading (Δh) values are calculated. The situation of the airplane moving across the 360°=0° boundary causes these values to not lie within a single 360° range, so they are adjusted by increments of 360° to lie within the range -180° to 180°. (3) Any data point with $|\Delta h|$ greater than 45°, or |pitch| or |roll| greater than 54°, or pilot command values outside a reasonable range is thrown out, resulting in the start of a new data segment. (4) All data are scaled to lie within the range ± 1. This is done to prevent differences in magnitude from biasing the training process. (5) The data are reformatted as a time-history so that a data vector now

consists of 20 values: 18 inputs (current and past sensor values plus past control values) and 2 outputs (current control values). (6) The data points are shuffled randomly. Steps 1-5 are performed by a C program (**Appendix B.3**) written for this project, while step 6 is performed using a program which was provided by Dr. Nechyba along with his neural network training program.

3.4 Training

After preprocessing, the data is used to train a cascade neural network (CNN). This part of the project makes use of Nechyba's research in human skill modeling [**4**,**10**,**11**], including his neural network training program (**Appendix B.4**). Nechyba has shown that

> continuous human control can be modeled well through CNNs, which are powerful nonlinear function approximators offering several advantages over more traditional neural network architectures: (1) The network architecture is not fixed prior to learning, but rather adapts as a function of learning. (2) Hidden units in the neural network can assume variable activation functions. And (3) the weights in the neural network are trained through the fast-converging node-decoupled extended Kalman filter. The flexibility of these cascade networks is ideal for [human skill] modeling, since few *a priori* assumptions are made about the underlying structure of the human controller.[**9**, p. 4][1]

3.5 Execution

During the execution stage, the HC11 and the 386 will receive sensory and control data as in the data collection stage, but now the 386 will apply the preprocessing steps to the data and then will use it as the input to the trained neural network. The resulting

[1] Here the author refers to **[10]**, **[11]**, and **[12]**.

control commands will then be sent to the HC11, which will use its ‘output compare’ function to generate PWM signals to control the servos. The output compare function sets an interrupt to go off after a certain amount of time has passed, and the interrupt routine cycles the output pin between the low and high states.

CHAPTER 4
RESULTS AND CONCLUSIONS

4.1 Training Results

To date, training has been performed using data representing straight and level flight. For this flight regime, it was expected that the resulting model would be fairly simple, given the highly linear correlation seen between the control inputs and sensor outputs. **Figure 4.1** illustrates the correlation between the rudder control input and the roll angle of the airplane.

The best results for this flight regime were indeed achieved by a linear control system. This control system was tested by using it to calculate predicted values for pilot commands given the time history of sensor and pilot commands. It was found to predict

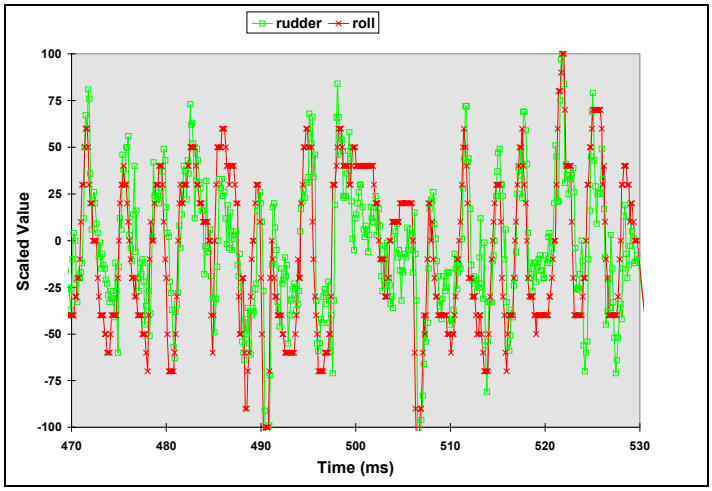

Figure 4.1 - Rudder command and roll angle

the pilot's control actions very well, with root-mean-square error of less than 0.1%. **Figure 4.2** illustrates the close correlation between the predicted and actual control values. It should be noted that this open-loop predictive ability does not necessarily indicate that the control system will perform well as a closed-loop controller.

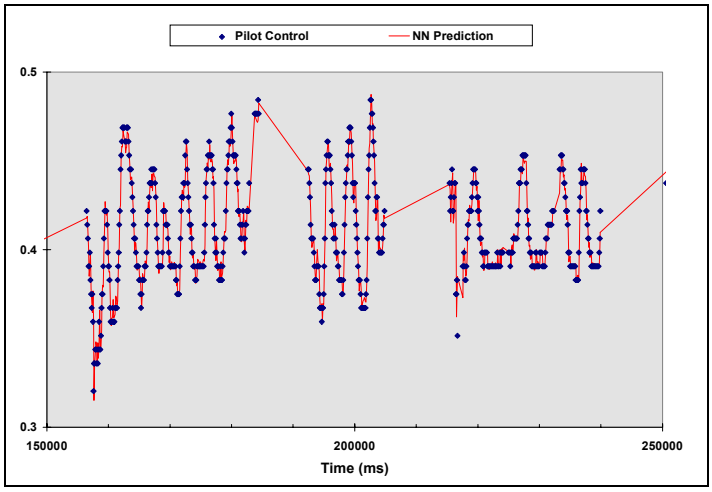

Figure 4.2 - Predicted and actual control values

4.2 Conclusions

This paper describes the design and construction of an aerial platform for use in human skill modeling experiments. At the time of this writing, the basic platform is completed and an electronics suite sufficient for initial experiments has been integrated into the platform. Initial experiments have shown that, using the method described in this paper, a model can be created which accurately predicts the next command of the human pilot given past command and current and past sensor data. Additionally, the software and hardware necessary to proceed with the next set of experiments--flying autonomously--has been completed.

4.3 Future Goals

There are many areas for future progress and improvements to this project. The immediate needs are listed below.

1. Additional data needs to be collected with the current hardware, new control models generated for straight-and-level flight, and then attempts made at autonomous flight.

2. Data needs to be collected, models trained, and attempts made at autonomous turning.

3. Additional sensors need to be integrated into the airframe to enable learning of more advanced behaviors, such as landing. Work has begun on development of a pressure-based altimeter for gross altitude measurement and a sonar-based height-above-ground detector for more precise altitude measurement near the ground. Simple landing behavior should be possible with only the addition of these altimeters.

4. Further investigation is required into vibration-reducing mounts that will allow the use of sensitive components such as the hard drive and the accelerometer-based tilt sensor.

APPENDIX A
PARTS LIST

Airplane Components	Approx Retail Value
Sig Kadet Senior airplane kit	$65
O.S. FP Surpass 52 engine	$200
Hitec Focus 6 radio and servos	$165
JUMPtec MOPS LCD3 computer with 6MB RAM	$450
M-Systems 64MB IDE flash drive	$300
Precision Navigation TCM2 compass module	$770
MRC11 microcontroller	$50
Batteries	$25
Additional materials to complete airplane kit	$125
Additional electronic parts	$75

Support Equipment	Approx Retail Value
9” VGA monitor	$70
DC-AC power inverter	$50
Additional equipment	$50

APPENDIX B
COMPUTER CODE

B.1 HC11 Code

Main file

```
//
// This is the HC11 code for the Avigator platform
// Written by Steve Stancliff
//

#include <stdio.h>
#include "hc11.h"
#include "mil.h"

typedef unsigned char BYTE;

BYTE calc_checksum(char *s);

BYTE   in1=40,in2=40,in5=40,in6=40;
BYTE   out1=120,out2=120,out3=120,out4=120;
//
// in1 - in6 contain captured pulse width
// out1 - out4 contain output pulse width
// outputs are set to neutral control position by default
//

#pragma interrupt_handler PulseCaptureIC4
/* channel 6 */
//
// this is the interrupt handler for Input Capture 4 port
// used to capture channel 6 from the receiver
// stores the captured value in in6
//
// the interrupt routine works by toggling the IC interrupt
// between triggering on a rising edge and triggering on a
// falling edge.
//
// the time between rising edge and falling edge gives the
// pulse width.  (I'm not sure what the units are, but it
// (doesn't matter).
//
void PulseCaptureIC4()
{
    static int  flag=0;
    static unsigned int risetime=0;

    if(flag==0){
        risetime=TOC5;
        SET_BIT(TCTL2,128);       // trigger on falling edge
        flag=1;
    }else{
        in6=(TOC5-risetime-1900)/9;   //in6 contains pulse width
                      //(value is scaled to fit in a byte)
        CLEAR_BIT(TCTL2,128);     // trigger on rising edge
        flag=0;
    }
```

```
    CLEAR_BIT(TFLG1,~8);
    return;
}

#pragma interrupt_handler PulseCaptureIC3
/* channel 5 */
void PulseCaptureIC3()
{
    static int  flag=0;
    static unsigned int risetime=0;

    if(flag==0){
        risetime=TIC3;
        SET_BIT(TCTL2,2);
        flag=1;
    }else{
        in5=(TIC3-risetime-1900)/9;
        CLEAR_BIT(TCTL2,2);
        flag=0;
    }

    CLEAR_BIT(TFLG1,~1);
    return;
}

#pragma interrupt_handler PulseCaptureIC2
/* channel 2 */
void PulseCaptureIC2()
{
    static int  flag=0;
    static unsigned int risetime=0;

    if(flag==0){
        risetime=TIC2;
        SET_BIT(TCTL2,8);
        flag=1;
    }else{
        in2=(TIC2-risetime-1900)/9;
        CLEAR_BIT(TCTL2,8);
        flag=0;
    }

    CLEAR_BIT(TFLG1,~2);
    return;
}

#pragma interrupt_handler PulseCaptureIC1
/* channel 1 */
void PulseCaptureIC1()
{
    static int  flag=0;
    static unsigned int risetime=0;

    if(flag==0){
        risetime=TIC1;
        SET_BIT(TCTL2,32);
        flag=1;
    }else{
        in1=(TIC1-risetime-1900)/9;
        CLEAR_BIT(TCTL2,32);
        flag=0;
    }

    CLEAR_BIT(TFLG1,~4);
    return;
}

/*
#define PULSE_WIDTH 3500
#pragma interrupt_handler PWMDriverOC5
//
// unused
//
void PWMDriverOC5()
{
```

```
    static int state=0;

    TFLG1=8;
    if(state==0){
        TOC5=PULSE_WIDTH;
        TCTL1=(TCTL1 & ~3) | 1;
    }else{
        TOC5=0;
        TCTL1 |=3;
    }
    state ^=1;
}

#pragma interrupt_handler PWMDriverOC4
//
// unused
//
void PWMDriverOC4()
{
    static int state=0;

    TFLG1=16;
    if(state==0){
          TOC4=PULSE_WIDTH;
        TCTL1=(TCTL1 & ~12) | 4;
    }else{
        TOC4=0;
        TCTL1 |=12;
    }
    state ^=1;
}
*/

#pragma interrupt_handler PWMDriverOC3
/* channel 2 out */
//
// interrupt routine for Output Capture 3 pin
// outputs a PWM signal using the pulse width stored
// in out3
//
// works by setting an interrupt to go off after a
// certain amount of time.  switches the output pin
// between high and low value at that interrupt
//
void PWMDriverOC3()
{
    static int state=0;

    TFLG1=32;
    if(state==0){
        TOC3=(int)((out2*9)+1900);//reconstitute pulse width from byte
        TCTL1=(TCTL1 & ~48) | 16;
    }else{
        TOC3=0;
        TCTL1 |=48;
    }
    state ^=1;
}

#pragma interrupt_handler PWMDriverOC2
/* channel 1 out */
void PWMDriverOC2()
{
    static int state=0;

    TFLG1=64;
    if(state==0){
        TOC2=(int)((out1*9)+1900);
        TCTL1=(TCTL1 & ~192) | 64;
    }else{
        TOC2=0;
        TCTL1 |=64;
    }
    state ^=1;
}
```

```
void setup_isrs()
//
// initialization
//
{
/*OC2*/
    TCTL1 |=192;
    TMSK1 |=64;

/*OC3*/
    TCTL1 |=48;
    TMSK1 |=32;

/*OC4
    TCTL1 |=12;
    TMSK1 |=16;

OC5
    TCTL1 |=3;
    TMSK1 |=8;
*/

/*IC4*/
    SET_BIT(PACTL,4);
    CLEAR_BIT(TCTL2,128);
    SET_BIT(TCTL2,64);
    SET_BIT(TMSK1,8);

/*IC3*/
    CLEAR_BIT(TCTL2,2);
    SET_BIT(TCTL2,1);
    SET_BIT(TMSK1,1);

/*IC2*/
    CLEAR_BIT(TCTL2,8);
    SET_BIT(TCTL2,4);
    SET_BIT(TMSK1,2);

/*IC1*/
    CLEAR_BIT(TCTL2,32);
    SET_BIT(TCTL2,16);
    SET_BIT(TMSK1,4);

    asm("cli");
    return;
}

int GetMessage()
//
// this routine receives the serial messages from the 386
//
{
    char  temp[256];
    int      i=1;

    temp[0]='#';
    temp[i]=getchar();
    while(temp[i]!='\n'){  // get chars until EOL
        i++;
        temp[i]=getchar();
        if(temp[i]=='#'){  // saw another BOL
            GetMessage();  // toss out current chars, recurse
            return(2); // 2 = got a message after some garbage
        }
    }
    i++;
    temp[i]='\0';
/*   printf(temp);*/
    ProcMessage(temp); // interpret message
    return(1); // 1 = got a message normally
}

char HextoDec(char in)
//
// sort of like atoi.
// converts an ASCII char representing a hex value
```

```
// into a char value with the proper numeric value.
// (ex: in='A' --> out=10
//
{
    if((in>47)&&(in<58)){
        return(in-48);
    }else if((in>64)&&(in<71)){
        return(in-55);
    }else if((in>96)&&(in<103)){
        return(in-87);
    }else{
        return(-1);
    }
}

int ProcMessage(char *temp)
//
// interprets messages
//
{
    char    out[256];

    if(!strcmp(temp,"#SEND*1C\r\n")){ // message requesting data transmission
        sprintf(out,"$DATA%2x,%2x,%2x,%2x*"
            ,(int)in1,(int)in2,(int)in5,(int)in6);
        printf("%s%2x\n",out,(int)calc_checksum(out));
    }else if(!strncmp(temp,"#SETSERVO",9)){  // message requesting command output
        if(strlen(temp) !=19){
            return(0);
        }else{
            unsigned char   a,b,c,d,e,f,cs,cs2;

            a=HextoDec(*(temp+15));
            b=HextoDec(*(temp+16));
            if((a==-1)||(b==-1)){
                return(0);
            }

            c=HextoDec(*(temp+9));
            d=HextoDec(*(temp+10));
            if((c==-1)||(d==-1)){
                return(0);
            }

            e=HextoDec(*(temp+12));
            f=HextoDec(*(temp+13));
            if((e==-1)||(f==-1)){
                return(0);
            }

            cs2=16*a+b;
            cs=calc_checksum(temp);
            if(cs!=cs2){
                return(0);
            }

            out1=c*16+d;
            if((out1>251)||(out1<4)){
                return(0);
            }

            out2=e*16+f;
            if((out2>251)||(out2<4)){
                return(0);
            }
        }
    }else{
        return(0); // 0 = no valid message found
    }
    return(1); // 1 = valid message found
}

BYTE calc_checksum(char *s)
//
// simple checksum routine.
// checksum covers chars between # and * non inclusive//
```

```
{
    BYTE n=0;
    char *cp;

    cp=s+1;
    while(*cp != '*'){
        n = n^(*cp);
        cp++;
    }
    return(n);
}

void main()
{
    setup_isrs();
    setbaud(BAUD9600);

    while(1){
        if(getchar()=='#'){
            GetMessage();
        }
    }
}

#include "vectors.c"
```

mil.h

```
/*****************************************************************
 * MEKATRONIX Copyright 1998                                     *
 * Title        mil.h
 * Programmer   Keith L. Doty
 * Date         March 10, 1998
 * Version      1
 *
 * Description                                                   *
 *  A collection of functions that operate on MC68HC11 registers.
 *   Object file is in libtj.a
*****************************************************************/

#ifndef _MIL_H
#define _MIL_H

#define CLEAR_BIT(x,y) x &= ~y;
/* Clear all bits in x corresponding to the bits set in y.
 * The other bits in x are not affected.
*/

#define SET_BIT(x,y) x |= y;
/* Set all bits in x corresponding to the bits set in y.
 * The other bits in x are not affected.
*/

#define CLEAR_FLAG(x,y) x &= y;
/* USE this routine ONLY for clearing flags. Clears flags in x corresponding
 * to the bits set in y. Other flags in x are not affected.
 */

#endif
```

hc11.h

```
#ifndef __HC11_H
#define __HC11_H   1
```

```
/* base address of register block, change this if you relocate the register
 * block. This is from an A8. May need to be changed for other HC11 members
 * or if you relocate the IO base address.
 */
#define _IO_BASE   0x1000
#define PORTA  *(unsigned char volatile *)(_IO_BASE + 0x00)
#define PIOC   *(unsigned char volatile *)(_IO_BASE + 0x02)
#define PORTC  *(unsigned char volatile *)(_IO_BASE + 0x03)
#define PORTB  *(unsigned char volatile *)(_IO_BASE + 0x04)
#define PORTCL *(unsigned char volatile *)(_IO_BASE + 0x05)
#define DDRC   *(unsigned char volatile *)(_IO_BASE + 0x07)
#define PORTD  *(unsigned char volatile *)(_IO_BASE + 0x08)
#define DDRD   *(unsigned char volatile *)(_IO_BASE + 0x09)
#define PORTE  *(unsigned char volatile *)(_IO_BASE + 0x0A)
#define CFORC  *(unsigned char volatile *)(_IO_BASE + 0x0B)
#define OC1M   *(unsigned char volatile *)(_IO_BASE + 0x0C)
#define OC1D   *(unsigned char volatile *)(_IO_BASE + 0x0D)
#define TCNT   *(unsigned short volatile *)(_IO_BASE + 0x0E)
#define TIC1   *(unsigned short volatile *)(_IO_BASE + 0x10)
#define TIC2   *(unsigned short volatile *)(_IO_BASE + 0x12)
#define TIC3   *(unsigned short volatile *)(_IO_BASE + 0x14)
#define TOC1   *(unsigned short volatile *)(_IO_BASE + 0x16)
#define TOC2   *(unsigned short volatile *)(_IO_BASE + 0x18)
#define TOC3   *(unsigned short volatile *)(_IO_BASE + 0x1A)
#define TOC4   *(unsigned short volatile *)(_IO_BASE + 0x1C)
#define TOC5   *(unsigned short volatile *)(_IO_BASE + 0x1E)
#define TCTL1  *(unsigned char volatile *)(_IO_BASE + 0x20)
#define TCTL2  *(unsigned char volatile *)(_IO_BASE + 0x21)
#define TMSK1  *(unsigned char volatile *)(_IO_BASE + 0x22)
#define TFLG1  *(unsigned char volatile *)(_IO_BASE + 0x23)
#define TMSK2  *(unsigned char volatile *)(_IO_BASE + 0x24)
#define TFLG2  *(unsigned char volatile *)(_IO_BASE + 0x25)
#define PACTL  *(unsigned char volatile *)(_IO_BASE + 0x26)
#define PACNT  *(unsigned char volatile *)(_IO_BASE + 0x27)
#define SPCR   *(unsigned char volatile *)(_IO_BASE + 0x28)
#define SPSR   *(unsigned char volatile *)(_IO_BASE + 0x29)
#define SPDR   *(unsigned char volatile *)(_IO_BASE + 0x2A)
#define BAUD   *(unsigned char volatile *)(_IO_BASE + 0x2B)
#define SCCR1  *(unsigned char volatile *)(_IO_BASE + 0x2C)
#define SCCR2  *(unsigned char volatile *)(_IO_BASE + 0x2D)
#define SCSR   *(unsigned char volatile *)(_IO_BASE + 0x2E)
#define SCDR   *(unsigned char volatile *)(_IO_BASE + 0x2F)
#define ADCTL  *(unsigned char volatile *)(_IO_BASE + 0x30)
#define ADR1   *(unsigned char volatile *)(_IO_BASE + 0x31)
#define ADR2   *(unsigned char volatile *)(_IO_BASE + 0x32)
#define ADR3   *(unsigned char volatile *)(_IO_BASE + 0x33)
#define ADR4   *(unsigned char volatile *)(_IO_BASE + 0x34)
#define OPTION *(unsigned char volatile *)(_IO_BASE + 0x39)
#define COPRST *(unsigned char volatile *)(_IO_BASE + 0x3A)
#define PPROG  *(unsigned char volatile *)(_IO_BASE + 0x3B)
#define HPRIO  *(unsigned char volatile *)(_IO_BASE + 0x3C)
#define INIT   *(unsigned char volatile *)(_IO_BASE + 0x3D)
#define TEST1  *(unsigned char volatile *)(_IO_BASE + 0x3E)
#define CONFIG *(unsigned char volatile *)(_IO_BASE + 0x3F)

unsigned int read_sci(void);
unsigned char read_spi(void);
void write_eeprom(unsigned char *addr, unsigned char c);
void write_sci(unsigned int);
void write_spi(unsigned char);

/* These values are for a 8Mhz clock
 * 0x3? set the SCP1|SCP0 to 0x3
 */
typedef enum {
    BAUD9600 = 0x30, BAUD4800 = 0x31, BAUD2400 = 0x32,
    BAUD1200 = 0x33, BAUD600 = 0x34, BAUD300 = 0x35
    } BaudRate;
void setbaud(BaudRate);

#ifndef INTR_ON
#define INTR_ON()  asm("   cli")
#define INTR_OFF() asm("   sei")
#endif

#ifndef bit
```

```
#define bit(x) (1 << (x))
#endif

#ifdef _SCI
/* SCI bits */
#define RDRF   bit(5)
#define TDRE   bit(7)
#define T8     bit(6)
#define R8     bit(7)
#endif

#ifdef _SPI
/* SPI bits */
#define MSTR   bit(4)
#define SPE    bit(6)
#define SPIF   bit(7)
#endif

#ifdef _EEPROM
/* EEPROM */
#define EEPGM  bit(0)
#define EELAT  bit(1)
#endif

#endif
```

B.2 386 Main Code

Main File

```
//
// This is the program for the 386 computer on the
// avigator platform.
//
// Written by Steve Stancliff
//
#define DEBUG
#define DEBUG6

#include <io.h>
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys\timeb.h>
#include <string.h>
#include "sio.h"

typedef unsigned char  BYTE;
typedef unsigned int      UINT;

typedef struct{
    long    time;   // time elapsed since startup (s)
    short   mtime;  // (ms)
    float   heading;
    float   pitch;
    float   roll;
    float   magX;   // magnetometer reading (unused)
    float   magY;   // will need these if external tilt sensor
    float   magZ;   // is used, to calculate heading
    float   temp;   // compass temperature (unused)
    UINT    servo[6];  // output values
    UINT    tilt[2];   // separate tilt sensor (unused)
}DATA; //Structure containing info for one point in time

void flush(FILE *stream);
FILE *fopen_unique(void);
void Output(char *s1,char *s2,struct timeb tm,FILE *fp);
void Output1(char *s1,struct timeb tm,FILE *fp);
void Output2(char *s2,struct timeb tm,FILE *fp);
BOOL ValidMessage(char *s);
BYTE calc_checksum(char *s);

DATA data;
struct timeb   starttm;
int filecount=0;

void main(int argc, char *argv[])
{
    FILE    *fp;
    char    flag1,flag2;
    char    s1[256]="",s2[256]="",*cp1,*cp2;
    struct timeb   tm,tm1,tm2;
    register char  c;

    printf("initializing\n");

    com1_init();
    com2_init();
    system("cls");
    ftime(&starttm);   // record current time
    ftime(&tm);
    flag1=0;   // flag for com1. 0 = initialized. 1 = request sent
    flag2=0;   // flag for com2. 2 = ??. 3 = line received
    cp1=s1;
    cp2=s2;
```

```
    fp=fopen_unique(); // open new output file
//  fp=stdout;
    if(fp==0){
        fp=stdout;
    }

    printf("running\n");
    com1_write("s?\r\n");  // data request for compass
    ftime(&tm1);       // time of data request
    com2_write("#SEND*1C\r\n");   // data request for HC11
    ftime(&tm2);       // time of request

    while(1){
        if(kbhit()){
            c=getch();
            if(c=='q'||c=='Q'){    // exit when type 'q'
                break;
            }
        }

        c=com_read(1);
        while(c!=-1 && c!='$'){    // read com1 until BOL found or buffer empty
            c=com_read(1);
        }
        if(c=='$'){// BOL found
            *cp1=c;
            cp1++;
            while(c!='\n'){    // read until EOL found
                c=com_read(1);
                if(c!=-1){
                    *cp1=c;
                    cp1++;
                }
            }
            *cp1='\0';
//          printf("%s",s1);
            cp1=s1;
            flag1=3;
        }

        c=com_read(2);
        while(c!=-1 && c!='$'){ // read com2 until BOL found or buffer empty
            c=com_read(2);
        }
        if(c=='$'){// BOL found
            *cp2=c;
            cp2++;
            while(c!='\n'){    // read until EOL found
                c=com_read(2);
                if(c!=-1){
                    *cp2=c;
                    cp2++;
                }
            }
            *cp2='\0';
//          printf("%s",s2);
            cp2=s2;
            flag2=3;
        }

//      printf("here %d %d\n",(int)flag1,(int)flag2);
        if((flag1==3)&&(flag2==3)){   // both have complete strings
            ftime(&tm);
            Output(s1,s2,tm,fp);
            if(argc>1){  // any command-line argument --> print to stdout
                Output(s1,s2,tm,stdout);
            }
            com1_write("s?\r\n");  // data request
            ftime(&tm1);
            com2_write("#SEND*1C\r\n"); // data request
            ftime(&tm2);
            flag1=1;
            flag2=1;
        }else if(flag2==3){    // just hc11 is ready to print
            ftime(&tm);
            Output2(s2,tm,fp);
```

```
            if(argc>1){
                Output2(s2,tm,stdout);
            }
            flag2=1;
            com2_write("#SEND*1C\r\n");
            ftime(&tm2);
        }else if(flag1==3){    // just compass is ready to print
            ftime(&tm);
            Output1(s1,tm,fp);
            if(argc>1){
                Output1(s1,tm,stdout);
            }
            flag1=1;
            com1_write("s?\r\n");
            ftime(&tm1);
        }
        ftime(&tm);
        if(tm.time>tm1.time){
            // 1 s has elapsed with no valid response, send new query
            com1_write("s?\r\n");
            ftime(&tm1);
        }else if((tm.time==tm1.time)&&(tm.millitm > tm1.millitm+100)){
            // 100 ms has elapsed with no valid response, send new query
            com1_write("s?\r\n");
            ftime(&tm1);
        }
        if(tm.time>tm2.time){
            com2_write("#SEND*1C\r\n");
            ftime(&tm2);
        }else if((tm.time==tm2.time)&&(tm.millitm > tm2.millitm+100)){
            com2_write("#SEND*1C\r\n");
            ftime(&tm2);
        }
    }
    com1_end();
    com2_end();
    return;

}

void Output(char *s1,char *s2,struct timeb tm,FILE *fp)
//
// this routine checks for valid messages, extracts the data from a message,
// and prints to disk
//
{
//  static int count=0;

    if(!ValidMessage(s1)){
        fprintf(fp,"bad1\n");
        fprintf(fp,s1);
        return;
    }
    if(!ValidMessage(s2)){
        fprintf(fp,"bad2\n");
        fprintf(fp,s2);
        return;
    }

//  printf("%s %s\n",s1,s2);

//HC11
    if(sscanf(s2,"$DATA%x,%x,%x,%x",&(data.servo[0]),&(data.servo[1])
            ,&(data.servo[2]),&(data.servo[3]))==EOF){
        fprintf(fp,"bad3\n");
        fprintf(fp,s2);
        return; //failed
    }

//Compass
    if(sscanf(s1,"$C%fP%fR%fX%fY%fZ%fT%f",&(data.heading),&(data.pitch)
                    ,&(data.roll))==EOF){
        fprintf(fp,"bad4\n");
        fprintf(fp,s1);
        return; //failed
    }
```

```
    data.time=tm.time-starttm.time;
    data.mtime=tm.millitm;

//  if(data.servo[3]<200){

        fprintf(fp,"%5ld%03d,%3d,%3d,%3d,%3d,%3d,%3d,%3d\n"
                ,data.time,data.mtime,(int)data.heading
                ,(int)data.pitch,(int)data.roll
                ,data.servo[0],data.servo[1]
                ,data.servo[2],data.servo[3]);

/*      fprintf(stdout,"%5ld%03d,%d,%d,%d,%d,%d,%d,%d\n"
                ,data.time,data.mtime,(int)data.heading
                ,(int)data.pitch,(int)data.roll
                ,data.servo[0],data.servo[1],data.servo[2],data.servo[3]);
    }
*/

    if(fp!=stdout){ // delayed writing to help reduce sampling delays caused by flash disk
        filecount++;
        if((filecount==51)){ // writes once 50 lines have been printed
            flush(fp);
            filecount=0;
        }
    }
    return;
}

void Output1(char *s1,struct timeb tm,FILE *fp)
{
//  static int count=0;

    if(!ValidMessage(s1)){
        fprintf(fp,"bad1\n");
        fprintf(fp,s1);
        return;
    }

//  printf("%s %s\n",s1,s2);

//Compass
    if(sscanf(s1,"$C%fP%fR%fX%fY%fZ%fT%f"
                  ,&(data.heading),&(data.pitch),&(data.roll))==EOF){
        fprintf(fp,"bad4\n");
        fprintf(fp,s1);
        return; //failed
    }

    data.time=tm.time-starttm.time;
    data.mtime=tm.millitm;

    fprintf(fp,"%5ld%03d,%3d,%3d,%3d,%3d,%3d,%3d,%3d\n"
                ,data.time,data.mtime,(int)data.heading
                ,(int)data.pitch,(int)data.roll,-99,-99,-99,-99);

    if(fp!=stdout){
        filecount++;
        if((filecount==51)){
            flush(fp);
            filecount=0;
        }
    }

    return;
}

void Output2(char *s2,struct timeb tm,FILE *fp)
{
//  static int count=0;

    if(!ValidMessage(s2)){
        fprintf(fp,"bad2\n");
        fprintf(fp,s2);
        return;
    }
```

```
//  printf("%s %s\n",s1,s2);

//HC11
    if(sscanf(s2,"$DATA%x,%x,%x,%x",&(data.servo[0]),&(data.servo[1])
            ,&(data.servo[2]),&(data.servo[3]))==EOF){
        fprintf(fp,"bad3\n");
        fprintf(fp,s2);
        return; //failed
    }

    data.time=tm.time-starttm.time;
    data.mtime=tm.millitm;

    fprintf(fp,"%5ld%03d,%3d,%3d,%3d,%3d,%3d,%3d,%3d\n"
                ,data.time,data.mtime
                ,-99,-99,-99
                ,data.servo[0],data.servo[1]
                ,data.servo[2],data.servo[3]);

    if(fp!=stdout){
        filecount++;
        if((filecount==51)){
            flush(fp);
            filecount=0;
        }
    }

    return;
}

BOOL ValidMessage(char *s)
//
// some simple checks for data corruption
//
{
    BYTE    cs=0xFF;
    int csin=0xFF;
    int n;

    if((*s != '$')&&(*s != '#')){
        return(FALSE);
    }
    n=strlen(s);
    if(*(s+n-1) != '\n'){
        return(FALSE);
    }
    if(*(s+n-2) != '\r'){
        return(FALSE);
    }
    if(*(s+n-5) != '*'){
        return(FALSE);
    }
    cs=calc_checksum(s);
    sscanf(s,"%*[^*]*%x",&csin);
    if((int)cs != csin){
        return(FALSE);
    }
    return(TRUE);
}

BYTE calc_checksum(char *s)
//
// simple data checksum
// checksum calculated for data between $ and * non-inclusive
//
{
    BYTE n=0;
    char *cp;

    cp=s+1;
    while(*cp != '*'){
        n = n^(*cp);
        cp++;
    }
    return(n);
```

```
}

FILE *fopen_unique()
//
// open a file with a unique name to avoid overwriting old data
// (e.g. out1, out2, ...)
//
{
    FILE    *fp;
    char    s[16],s2[3];
    register int i;

    for(i=0;i<30;i++){
        itoa(i,s2,10);
        sprintf(s,"out%s.txt",s2);
        fp=fopen(s,"r");
//      printf("%s\n",s);
        if(fp){
            fclose(fp);
        }else{
            fp=fopen(s,"w");
            return(fp);
        }
    }
    return(NULL);
}

void flush(FILE *stream)
//
// flush buffer to disk
//
{
      int duphandle;

      /* flush the stream's internal buffer */
      fflush(stream);

      /* make a duplicate file handle */
      duphandle = dup(fileno(stream));

      /* close the duplicate handle to flush\
          the DOS buffer */
      close(duphandle);
}
```

Serial I/O Routines (sio.c)

```
//
// These are the serial interface routines for the 386 computer
// on the avigator platform
// Written by Steve Stancliff
// Derived from code in C_Programmer's_Guide_to_Serial_Communications
// by Joe Campbell
//

#include <dos.h>
#include <stdio.h>
#include <sys\timeb.h>
#include "sio.h"

extern struct timeb    starttm;

void interrupt (far *oldint0c)();
void interrupt (far *oldint0b)();
RING ring[2];

void interrupt far int_0c_handler()
//COM1 handler
{
    unsigned char  cISR,cData;
```

```
    enable();  //allow interrupts

    cISR=inportb((COM1_BASE+ISR));

    if((cISR&ISR_RXRDY)==cISR){   //if exception is RXrdy
        cData=inportb((COM1_BASE));
        ring[0].buff[ring[0].in]=cData;
        if(cData=='$'){
            ring[0].out=ring[0].in;
        }
        ring[0].in++;
        if(ring[0].in==RING_BUFFER_SIZE){
            ring[0].in=0;
        }
    }
    outportb(PIC_EOI,EOI_VALUE);  //clear interrupt
    return;
}

void interrupt far int_0b_handler()
//COM2 handler
{
    unsigned char  cISR,cData;

//  enable();  //allow interrupts

    cISR=inportb((COM2_BASE+ISR));

    if((cISR&ISR_RXRDY)==cISR){   //if exception is RXrdy
        cData=inportb((COM2_BASE));
        ring[1].buff[ring[1].in]=cData;
        ring[1].in++;
        if(ring[1].in==RING_BUFFER_SIZE){
            ring[1].in=0;
        }
    }
    outportb(PIC_EOI,EOI_VALUE);  //clear interrupt
    return;
}

void com1_end()
{
    setvect(IRQ4_INT,oldint0c);   //restore original interrupt vector
    return;
}

void com2_end()
{
    setvect(IRQ3_INT,oldint0b);   //restore original interrupt vector
    return;
}

void com2_init()
{
    unsigned char cTempFlags;

    oldint0b=getvect(IRQ3_INT);   //save original interrupt vector
    setvect(IRQ3_INT,int_0b_handler); //set new interrupt vector

    outportb((COM2_BASE+LCR),SETBAUD);
    outport((COM2_BASE+DLAB),BR9600); //set baudrate
    outportb((COM2_BASE+LCR),(LCR_8_DB|LCR_NO_PARITY|LCR_1_SB));
    //set comm params

    cTempFlags=inportb((COM2_BASE+IER));
    cTempFlags|=IER_RXRDY;
    outportb((COM2_BASE+IER),cTempFlags);    //enable RXRDY interrupt in UART

    cTempFlags=inportb(PIC_ADDR);
    cTempFlags = cTempFlags & PIC_IRQ3;
    outportb(PIC_ADDR,cTempFlags);    //enable interrupts in PIC

    {
    register int i;
    for(i=0;i<6;i++){
```

```
        inport((COM2_BASE+i)); //clear interrupts
    }
    }

    outportb((COM2_BASE+MCR),(MCR_DTR|MCR_RTS|MCR_GPO2));
    //set DTR,RTS,enable interrupts in UART
    return;
}

void com1_init()
{
    unsigned char cTempFlags;

    oldint0c=getvect(IRQ4_INT);   //save original interrupt vector
    setvect(IRQ4_INT,int_0c_handler); //set new interrupt vector

    outportb((COM1_BASE+LCR),SETBAUD);
    outport((COM1_BASE+DLAB),BR38400);       //set baudrate
    outportb((COM1_BASE+LCR),(LCR_8_DB|LCR_NO_PARITY|LCR_1_SB));
    //set comm params

    cTempFlags=inportb((COM1_BASE+IER));
    cTempFlags|=IER_RXRDY;
    outportb((COM1_BASE+IER),cTempFlags); //enable RXRDY interrupt in UART

    cTempFlags=inportb(PIC_ADDR);
    cTempFlags = cTempFlags & PIC_IRQ4;
    outportb(PIC_ADDR,cTempFlags); //enable interrupts in PIC

    {
    register int i;
    for(i=0;i<6;i++){
        inport((COM1_BASE+i)); //clear interrupts
    }
    }

    outportb((COM1_BASE+MCR),(MCR_DTR|MCR_RTS|MCR_GPO2));
    //set DTR,RTS,enable interrupts in UART
    return;
}

char com_read(int port)
//read in character
{
    register int p=port-1;
    char c;

    if(ring[p].out==ring[p].in){
        c=-1;
        return(c);
    }
    c=ring[p].buff[ring[p].out];
    ring[p].out++;
    if(ring[p].out==RING_BUFFER_SIZE){
        ring[p].out=0;
    }
    return(c);
}

void com1_write(char *lpS)
//write a string
{
    char *c=lpS;
    BOOL fDone=FALSE;

    while(!fDone){
        if(!(inportb((COM1_BASE+LSR))&LSR_TBE)){
            continue;
        }
        if(*c){
            outportb((COM1_BASE),*c);
        }else{
            fDone=TRUE;
        }
        c++;
    }
```

```
    while(!inportb((COM1_BASE+LSR)&LSR_TXE));
    return;
}

void com2_write(char *lpS)
//write a string
{
    char *c=lpS;
    BOOL fDone=FALSE;

    while(!fDone){
        if(!(inportb((COM2_BASE+LSR))&LSR_TBE)){
            continue;
        }
        if(*c){
            outportb((COM2_BASE),*c);
        }else{
            fDone=TRUE;
        }
        c++;
    }
    while(!inportb((COM2_BASE+LSR)&LSR_TXE));
    return;
}
```

sio.h

```
// sio.h - header file for serial I/O routines

typedef unsigned char BOOL;

#define  TRUE  1
#define  FALSE 0

#define  SETBAUD  0x80
#define  BR2400   48
#define  BR9600   12
#define BR19200 6
#define BR38400 3

#define COM2_BASE       0x02F8
#define COM1_BASE       0x03F8
#define DLAB            0x00
#define LCR             0x03
#define LCR_8_DB        0x03
#define LCR_NO_PARITY   0x00
#define LCR_1_SB        0x00

#define  IRQ3_INT    0x0b
#define  IRQ4_INT    0x0C

#define  IER         0x01
#define  IER_RXRDY   0x01
#define  PIC_ADDR    0x21
#define  PIC_IRQ3    0xF7
#define  PIC_IRQ4    0xEF

#define  MCR      0x04
#define  MCR_DTR  0x01
#define  MCR_RTS  0x02
#define  MCR_GPO2 0x08

#define RING_BUFFER_SIZE   2048
#define ISR       0x02
#define ISR_RXRDY 0x04
#define PIC_EOI   0x20
#define EOI_VALUE 0x20

#define  LSR   0x05
#define  LSR_TBE  0x20
```

```
#define  LSR_TXE  0x40

typedef struct{
    unsigned char  buff[RING_BUFFER_SIZE];
    int            in;
    int            out;
}RING;  //ring buffer structure

//extern void interrupt far int_0c_handler();
//extern void interrupt far int_0b_handler();
extern void com1_end(void);
extern void com1_init(void);
extern void com2_end(void);
extern void com2_init(void);
extern void com1_write(char *lpS);
extern void com2_write(char *lpS);
extern char com_read(int port);
```

B.3 Data Preprocessing Code

```
//
// This is the code that preprocesses the training data for the
// avigator platform
// Written by Steve Stancliff
//

#include <stdio.h>
#include <math.h>
#include <conio.h>

typedef struct{
    long    t;  // time
    int h;  // heading
    int p;  // pitch
    int r;  // roll
    int A;  // first command channel
    int B;  // second command channel
}INPUT;

typedef struct{
    long    t;
    float dh;  // change in heading
    float dh1; // (at t-1)
    float dh2; // (at t-2)
    float dh3; // ...
    float p;
    float p1;
    float p2;
    float p3;
    float r;
    float r1;
    float r2;
    float r3;
    float A;
    float A1;
    float A2;
    float A3;
    float B;
    float B1;
    float B2;
    float B3;
}OUTPUT;

void output(FILE *outfp,OUTPUT o2);
int calc_output(INPUT i2,OUTPUT o2,INPUT i4,OUTPUT *o4);
float do_heading(int new,int old);
float do_pitchroll(int in);
float do_control(int in);
int readline(FILE *fp,INPUT *i);

void main()
{
    INPUT      i1={0,0,0,0,180,180},i2,i3,i4;
    OUTPUT  o2={-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},o4;
//  OUTPUT  o2={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},o4;
    FILE       *infp,*outfp;
    int     i=0;

    infp=fopen("in","r");
    outfp=fopen("out","w");

    readline(infp,&i1);
    i2=i1;
    while(readline(infp,&i3)==6){

printf("%d,%ld\t",++i,i3.t);

        if(i3.t-i1.t >300){    // gap too large, discard data
            OUTPUT  o={-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
```

```
            i1=i3;
            i2=i1;
            o2=o;
//          getch();
        }else if(i3.t - i2.t <100){
            i1=i3;
        }else if(i3.t - i2.t ==100){
            i4=i3;
            if(calc_output(i2,o2,i4,&o4)<0){
                OUTPUT o={-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
                i1=i3;
                i2=i1;
                o2=o;
//              printf("yes\n");
//              getch();
            }else{
                output(outfp,o4);
                i2=i4;
                o2=o4;
                i1=i3;
            }
        }else{
            while(i3.h-i2.h>180){  // bring headings into the range -180..180
                i3.h-=360;
            }
            while(i3.h-i2.h<-180){
                i3.h+=360;
            }

            while(i3.t - i2.t >100){ // resample (interpolate)
                i4.t=i2.t+100;
                i4.h=(int)(i4.t-i1.t)*(i3.h-i1.h)/(int)(i3.t-i1.t)+i1.h;
                i4.p=(int)(i4.t-i1.t)*(i3.p-i1.p)/(int)(i3.t-i1.t)+i1.p;
                i4.r=(int)(i4.t-i1.t)*(i3.r-i1.r)/(int)(i3.t-i1.t)+i1.r;
                i4.A=(int)(i4.t-i1.t)*(i3.A-i1.A)/(int)(i3.t-i1.t)+i1.A;
                i4.B=(int)(i4.t-i1.t)*(i3.B-i1.B)/(int)(i3.t-i1.t)+i1.B;
                if(calc_output(i2,o2,i4,&o4)<0){
                    OUTPUT o={-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
                    i1=i3;
                    i2=i1;
                    o2=o;
                    break;
//                  printf("yes\n");
//                  getch();
                }
                output(outfp,o4);
                i2=i4;
                o2=o4;
                if(i3.t-i2.t<100){
                    i1=i3;
                }
            }
            if(i3.t - i2.t ==100){
                i4=i3;
                if(calc_output(i2,o2,i4,&o4)<0){
                    OUTPUT o={-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
                    i1=i3;
                    i2=i1;
                    o2=o;
//                  printf("yes\n");
//                  getch();
                }else{
                    output(outfp,o4);
                    i2=i4;
                    o2=o4;
                    i1=i3;
                }
            }
        }
    }

    fclose(infp);
    fclose(outfp);
    return;
}
```

```
void output(FILE *outfp,OUTPUT o2)
{
    fprintf(outfp,
//      "%ld,"
        "%f %f %f %f"
        " %f %f %f %f"
//      " %f %f %f %f"
//      " %f %f %f %f"
        " %f %f"
        " %f %f %f %f\n"
//      ,o2.t
        ,o2.dh,o2.dh1,o2.dh2,o2.dh3
        ,o2.p,o2.p1,o2.p2,o2.p3
        ,o2.r,o2.r1,o2.r2,o2.r3
//      ,o2.A1,o2.A2,o2.A3
//      ,o2.B1,o2.B2,o2.B3
        ,o2.A,o2.B);
    return;
}

int calc_output(INPUT i1,OUTPUT o1,INPUT i2,OUTPUT *o2)
{
    o2->t=i2.t;
    o2->dh=do_heading(i2.h,i1.h);
    if(o2->dh==-1){
        return(-1);
    }
    o2->dh1=o1.dh;
    o2->dh2=o1.dh1;
    o2->dh3=o1.dh2;
    o2->p=do_pitchroll(i2.p);
    if(o2->p==-1){
        return(-2);
    }
    o2->p1=o1.p;
    o2->p2=o1.p1;
    o2->p3=o1.p2;
    o2->r=do_pitchroll(i2.r);
    if(o2->r==-1){
        return(-3);
    }
    o2->r1=o1.r;
    o2->r2=o1.r1;
    o2->r3=o1.r2;
    o2->A=do_control(i2.A);
    if(o2->A==-1){
        return(-4);
    }
    o2->A1=o1.A;
    o2->A2=o1.A1;
    o2->A3=o1.A2;
    o2->B=do_control(i2.B);
    if(o2->B==-1){
        return(-5);
    }
    o2->B1=o1.B;
    o2->B2=o1.B1;
    o2->B3=o1.B2;

    return(0);
}

float do_heading(int new,int old)
//
// calculate change in heading
//
{
    int dh=new-old;

    if(dh>180){
        dh-=360;
    }else if(dh<-180){
        dh+=360;
    }
    if(abs(dh)>45){
```

```
//      printf("dh>45\n");
//      printf("%d,%d,%d\n",new,old,dh);
//      getch();
        return(-1); // |dh| >45, discard data point
    }
    return(((float)dh+45.)/90.);
}

float do_pitchroll(int in)
//
// check for valid pitch, roll data
//
{
//return(in);
    if(abs(in)>54){
//      printf("pr>54\n");
//      getch();
        return(-1);// bad data, discard
    }
    return(((float)in+54.)/108.);
}

float do_control(int in)
//
// check for valid control data
//
{
//return(in);
    if((abs(in)>250) || (abs(in)<122)){
//      printf("control ob\n");
//      getch();
        return(-1); // bad data, discard
    }
    return(((float)in-122.)/128.);
}

int readline(FILE *fp,INPUT *i)
{
    return(fscanf(fp,"%ld,%d,%d,%d,%d,%d\n",&(i->t),&(i->h),&(i->p),&(i->r),&(i->A)
        ,&(i->B)));
}
```

B.4 Neural Network Training Code[1]

kalman_main.h

```
/*************************************************************************/
/*                                                                       */
/* Author: Michael C. Nechyba                                            */
/* Last modified: 6/23/97                                                */
/*                                                                       */
/*************************************************************************/

/*

There is one additional command line option not available through
the parameter file.  That is -patternerror. Together with ExistingFile and
TestFile (or -existingfile and -testfile), it will read in the network,
report the pattern-wise error for the test file, and exit. The reason it
is not accessible via the parameter file is because the ARGC macro resets
whatever gets read in.


Parameter explanations:

Type: I --> integer
      R --> floating-point number (real)
      S --> string
      (o) indicates optional parameter

All flags are decided by whether they are negative or not.

-------------------------------------------------------------------------
Parameter       Type    Explanation (default)

NTrainPatterns  I(o)    # of training patterns (length of file)
NCrossPatterns  I(o)    # of cross-validation patterns (length of file)
NTestPatterns   I(o)    # of test patterns (length of file)
TrainFile       S  training data file
CrossFile       S(o)    cross-validation data file
TestFile        S(o)    independent test data file (defaults to CrossFile)

ExistingFile    S(o)    read in existing network file
CascadeFile     S(o)    network file to write weights to ("network")
DumpWeights     I(o)    flag: save weights after each hidden unit (0, >0)

OutputType      *(o)    output unit type (defaults to LINEAR)
UnitType        **      hidden unit type

NInputs         I       number of inputs (without bias unit)
NOutputs        I       number of outputs
MaxHidden       I       maximum number of hidden units
WeightRange     R(o)    initial randomized weight range (1.0)
EtaP            R(o)   initial inverse diag P elements (.01)
EtaQ            R(o)    artificial process noise (.0001)
EtaR            R(o)    diagonal 'R' elements (1.0)
TimeConstant    R(o)    decay R matrix (0.0)
MaxIterations   I(o)    maximum iterations/candidate (1)
RandomSeed      I(o)    random seed (1)
RandomOrder     I(o)    flag: randomize order of training set (-1)
NCandidates     I(o)    # of candidate units (1)
ChangeThreshold R(o)    consider this a significant change in RMS error (.01)
ChangePeriod    I(o)    messure RMS error change over this many epochs (1)
Report          ***(o)  flag: report various things (-1 --> DONT_REPORT)
BitError        I(o)    flag: report bit error stats (-1)
ErrorStop       I(o)    flag: stop training when RMS error goes up (1)
UseCache        I(o)    flag: use cache over training set (1)
```

[1] All code in this section was provided by M. Nechyba

```
UseLookup       I(o)    flag: use look-up table for functions (1)
Trials          I(o)    # of trials (1)
InputNoise      R(o)    Add noise to inputs (0.0)
OutputNoise     R(o)    Add noise to outputs (0.0)
SaveNoiseFiles  I(o)    flag: save data files with added noise (-1)
                    (tmp.trn, tmp.cvd, tmp.tst) (exits without training)
NoLinear        I(o)    flag: have no connections between inputs/outputs (-1)

ActiveTrain     I(o)    flag: train with active data selection (-1) (n.im.)

DisplaySSE      I(o)    flag: also display SSE
DisplayPerOutput I(o)   flag: display RMS and/or SSE per output
DisplayMaxErr   I(o)    flag: display maximum error as well
DisplayStdDev   I(o)    flag: display standard deviation of errors as well
DisplayHistogram I(o)   flag/num: if negative, don't display histogram of
HistogramMin    R(o)    minimum for histogram
HistogramMax    R(o)    maximum for histogram
DisplayBitError  I(o)   flag: same as BitError

* SIGMOID SINE SINE2 GAUSSIAN BESSEL0 BESSEL1
  COSINE COSINE2 GTSINE SIGMOID2 LINEAR

** SIGMOID SINE SINE2 GAUSSIAN BESSEL0 BESSEL1
   COSINE COSINE2 GTSINE SIGMOID2 MIXED

*** -5 --> REPORT_COVARIANCE
    -4 --> REPORT_EACH_PATTERN
    -3 --> REPORT_EACH_EPOCH
    -1 --> DONT_REPORT
------------------------------------------------------------------------------ */

GlobalOptions Global;

PARAMETERS ParameterTable[] = {
  {"CascadeFile", STR, (POINTER) &Global.cascade_file},
  {"ExistingFile", STR, (POINTER) &Global.existing_file},
  {"TrainFile",   STR, (POINTER) &Global.train_file},
  {"CrossFile",   STR, (POINTER) &Global.cross_file},
  {"TestFile",    STR, (POINTER) &Global.test_file},
  {"OutputType",  STR, (POINTER) &Global.output_type},
  {"UnitType",    STR, (POINTER) &Global.unit_type},
  {"NInputs",     INT, (POINTER) &Global.n_inputs},
  {"NOutputs",    INT, (POINTER) &Global.n_outputs},
  {"MaxHidden",   INT, (POINTER) &Global.max_hidden},
  {"WeightRange", DBL, (POINTER) &Global.weight_range},
  {"EtaP",        DBL, (POINTER) &Global.kl.eta_p},
  {"EtaQ",        DBL, (POINTER) &Global.kl.eta_q},
  {"EtaR",        DBL, (POINTER) &Global.kl.eta_r},
  {"TimeConstant",DBL, (POINTER) &Global.kl.time_constant},
  {"MaxIterations", INT, (POINTER) &Global.max_iterate},
  {"RandomSeed",    INT, (POINTER) &Global.random_seed},
  {"RandomOrder",   INT, (POINTER) &Global.random_order},
  {"NCandidates",   INT, (POINTER) &Global.n_candidates},
  {"ChangeThreshold", DBL, (POINTER) &Global.change_threshold},
  {"ChangePeriod",    INT, (POINTER) &Global.change_period},
  {"DumpWeights",     INT, (POINTER) &Global.dump_weights},
  {"Report",   INT, (POINTER) &Global.report},
  {"BitError", INT, (POINTER) &Global.display_bit_err},
  {"ErrorStop", INT, (POINTER) &Global.error_stop},
  {"UseCache", INT, (POINTER) &Global.use_cache},
  {"UseLookup", INT, (POINTER) &Global.use_lookup},
  {"Trials", INT, (POINTER) &Global.trials},
  {"InputNoise", DBL, (POINTER) &Global.input_noise},
  {"OutputNoise", DBL, (POINTER) &Global.output_noise},
  {"SaveNoiseFiles", INT, (POINTER) &Global.save_noise_files},
  {"NoLinear", INT, (POINTER) &Global.no_linear},
  {"ActiveTrain", INT, (POINTER) &Global.active_train},
  {"ActiveInit", INT, (POINTER) &Global.active_init},
  {"ActiveTolerance",  DBL, (POINTER) &Global.active_tolerance},
  {"ActiveGroup", INT, (POINTER) &Global.active_group},
  {"DisplayPerOutput", INT, (POINTER) &Global.display_per_output},
  {"DisplaySSE", INT, (POINTER) &Global.display_sse},
  {"DisplayMaxErr", INT, (POINTER) &Global.display_max_err},
  {"DisplayStdDev", INT, (POINTER) &Global.display_std_dev},
  {"DisplayHistogram", INT, (POINTER) &Global.display_histogram},
  {"HistogramMin", DBL, (POINTER) &Global.histogram_min},
```

```
  {"HistogramMax", DBL, (POINTER) &Global.histogram_max},
  {"DisplayBitError", INT, (POINTER) &Global.display_bit_err},
  {"NCrossPatterns",  INT, (POINTER) &Global.n_cross_patterns},
  {"NTestPatterns",  INT, (POINTER) &Global.n_test_patterns},
  {"NTrainPatterns", INT, (POINTER) &Global.n_train_patterns}
};
```

kalman.h

```
/*************************************************************************/
/*                                      */
/* Author: Michael C. Nechyba                                    */
/* Last modified: 6/21/97                                              */
/*                                      */
/*************************************************************************/

#ifdef MANY_OUTPUTS
#  include "amdm.h"
#endif

#define PCT(a, b) (100. * ((double) (a))/((double) (b)))

#define MAX_OUTPUTS 50
#define MAX_HIST   500

#define REPORT_COVARIANCE -5
#define REPORT_EACH_PATTERN -4
#define REPORT_EACH_EPOCH   -3
#define DONT_REPORT         -1

#define NUMBER_OF_UNITS 11
#define MAX_CANDIDATES 100
#define MAX_ITERATE    1000

#define FALSE_OFF 0
#define FALSE_ON  1
#define TOTAL     2
#define CLASSIFY  3
#define BIT_DIM   4

#define MAT(m, i) (*(*(m++) + (i)))

#ifdef DSPSA
typedef struct spsa_struct {
  CASCADE_NET net_pos, net_neg;
  double *a, *c, *g, *delta;
  double *invc;
  int p;
} SPSA;
#endif

#ifndef DSPSA
typedef struct bit_struct {
  int false_off;
  int false_on;
  int total;
  int classify;

  double pct_false_off;
  double pct_false_on;
  double pct_classify;

} BitError;

typedef struct err_struct {
  double **dat;

  double **vec;
  int n_vec;
```

```
  double rms[MAX_OUTPUTS];
  double sse[MAX_OUTPUTS];
  double max[MAX_OUTPUTS];
  double stddev[MAX_OUTPUTS];

  int histogram[MAX_OUTPUTS][MAX_HIST];

  BitError bit;
} Error;


typedef struct kalman_struct {
  int n_nodes;
  double ***p;
  double *r;
  double **a;
  double **ai;
  double *err;
  double **psi;
  double *psi_out;
  double ***psim;
  double **v;
  double **vm;
  double *u;
  double *net_act;
  double **net_deriv;
  double ***tm;
  double **tv;

  double time_constant;
  double eta_p;
  double eta_q;
  double eta_r;
} KalmanFilter;
#endif

typedef struct global_struct {

  /* detailed error display flags */

  int display_per_output;
  int display_sse;
  int display_max_err;
  int display_std_dev;
  int display_histogram;
  int display_bit_err;

  double histogram_min, histogram_max;


  Error err_trn, err_cvd, err_tst;

  /* active data selection options */

  int active_train;
  int active_init;
  int active_group;

  double active_tolerance;


  /* end -- active data selection options */



  PARAMETERS *parameters;
  int n_parameters;

  int argc;
  char **argv;

  char parameter_file[STR_REG];
  char test_file[STR_REG];
  char existing_file[STR_REG];
  int pattern_wise_error;
  char train_file[STR_REG];
```

```
  char cross_file[STR_REG];
  char cascade_file[STR_REG];
  int dump_weights;

  int n_cross_patterns;
  int n_test_patterns;
  int n_train_patterns;
  int n_epochs;
  int error_stop;
  int use_cache;
  int use_lookup;
  int trials;
  int trial_number;
  int report;

  char unit_type[STR_REG];
  char output_type[STR_REG];

  int n_inputs;
  int n_outputs;
  int max_hidden;
  int no_linear;

  int change_period;
  double change_threshold;
  double err[MAX_ITERATE];
  int max_iterate;

#ifdef DSPSA
  SPSA sp;
  double a_coeff, c_coeff;
#else
  KalmanFilter kl;
#endif
  CASCADE_NET net[MAX_CANDIDATES];
  CASCADE_NET best_weights;

  double weight_range;
  double input_noise, output_noise;
  int save_noise_files;

  int random_seed;
  int n_candidates;

#ifndef DSPSA
  int random_order, *index;
#endif
  double **tst;
  double **trn;
  double **cross;
  double **cache_activations;
  double **cache_net_act;

#ifdef MANY_OUTPUTS
  dym d;
#endif
} GlobalOptions;

#ifndef DSPSA
extern int kalman_train(GlobalOptions *gl);
extern int kalman_one_unit(GlobalOptions *gl, int which);

extern int initialize_kalman_filter(CASCADE_NET *net,
                  KalmanFilter *kl, int n_nodes);
extern int initialize_kalman_net(GlobalOptions *gl);
#endif

extern int display_error(FILE *fp, GlobalOptions *gl, CASCADE_NET *net,
           int type, int n);
extern double quick_rms_error(CASCADE_NET *net, double **dat, int n_dat);
extern int sse_error(CASCADE_NET *net, Error *e);
extern double error_statistics(GlobalOptions *gl, CASCADE_NET *net, Error *e);

extern double bit_error(CASCADE_NET *net, Error *e);
extern int zero_linear_connections(CASCADE_NET *net);
extern int kalman_unit_activations_with_cache(CASCADE_NET *net,
```

```
                        double *inputs,
                        double *activations,
                        double *net_act);
extern int kalman_unit_activations(CASCADE_NET *net, double *inputs,
                 double *activations,
                 double *net_act);
extern int add_hidden_unit(CASCADE_NET *net, int type, double *weights);
extern int copy_kalman_net(CASCADE_NET *new, CASCADE_NET *old);
extern int create_kalman_net(CASCADE_NET *net,
               int n_inputs, int max_hidden, int n_outputs,
               double weight_range, int output_type);
extern int read_kalman_net(CASCADE_NET *net, char *file_name, int max_hidden);
extern int zero_kalman_net(CASCADE_NET *net);
extern int clear_kalman_net(CASCADE_NET *net);
extern int matrix_add_noise(double **mat, int n, int cl, int cr, double noise);
extern int randomize_index(int *index, int n);
extern int process_command_line(GlobalOptions *gl, int argc, char **argv);

#ifdef MANY_OUTPUTS
extern int inverse_new(double **inv2, double **mat, int n, dym *d);
extern int allocate_dym(dym *d, int n);
#endif

#ifdef DSPSA
extern int spsa_train(GlobalOptions *gl);
extern int display_error(FILE *fp, GlobalOptions *gl, CASCADE_NET *net,
           int type, int n);
extern double spsa_error(CASCADE_NET *net, double **dat, int n_dat);
extern int spsa_one_unit(GlobalOptions *gl, int which);
extern int spsa_all_unit(GlobalOptions *gl, int which);

extern int initialize_spsa(SPSA *sp, GlobalOptions *gl);
extern double c(int k, GlobalOptions *gl);
extern double a(int k, GlobalOptions *gl);
extern int initialize_spsa_net(GlobalOptions *gl);
#endif
```

nn_util.h

```
/*****************************************************************************/
/*                                                                           */
/* Author: Michael C. Nechyba                                                */
/* Last modified: 11/3/96                                                    */
/*                                                                           */
/*****************************************************************************/

#define USE_LOOKUP 1
#define USE_EXACT_FNC 0

#define FORMAT_ERROR -1

#define SIGMOID  0
#define SINE     1
#define SINE2    2
#define GAUSSIAN 3
#define BESSEL0  4
#define BESSEL1  5
#define COSINE   6
#define COSINE2  7
#define GTSINE   8
#define SIGMOID2 9
#define CARUNIT  10
#define LINEAR   11
#define MIXED    12

#define BESSEL_ROUND (100.)

#define SIGMOID_ROUND (15.)

#define PI_OVER_TWO (1.57079632679489661923)
```

```
#define PI         (3.14159265358979323846)
#define INV_PI     (0.31830988618379067154)
#ifndef TWO_PI
#define TWO_PI     (6.2831853071795864769)
#endif
#define INV_TWO_PI (0.15915494309189533577)
#define LOOKUP 10000

#define FRAC(x) (x - (double) (int) (x))

typedef struct funct_struct {
  double (*act_fnc)();
  double (*dact_fnc)();
} Func;

typedef struct lookup {
  double ft[LOOKUP];
  double factor;
} Lookup;

typedef struct cascade_net {
  double **weights;
  int *n_connections, *type;
  int n_units, n_inputs, n_outputs, n_hidden;
  int max_hidden;
} CASCADE_NET;

typedef struct bp_net {
  double **weights;
  double *outputs;
  int first_hidden, first_output;
  int **connections;
  int *n_connections;
  int n_units, n_inputs, n_outputs, n_hidden;
} BP_NET;


/*************************** Lookup tables ********************************/

extern Func Function[];
extern Lookup Sin_lu;
extern Lookup Sigmoid_lu;
extern Lookup Dsigmoid_lu;
extern Lookup Sigmoid2_lu;
extern Lookup Dsigmoid2_lu;
extern Lookup Carunit_lu;
extern Lookup Dcarunit_lu;
extern Lookup Gaussian_lu;
extern Lookup Dgaussian_lu;
extern Lookup Gtsin_lu;
extern Lookup Dgtsin_lu;
extern Lookup Bessel0_lu;
extern Lookup Dbessel0_lu;
extern Lookup Bessel1_lu;
extern Lookup Dbessel1_lu;
extern double NOValues[];

/*************************** Prototypes ***********************************/

extern int zero_net(CASCADE_NET *net);
extern int clear_net(CASCADE_NET *net);
extern int write_net(CASCADE_NET *net, char *file_name);
extern int read_net(CASCADE_NET *net, char *file_name);
extern CASCADE_NET *chop_weights(CASCADE_NET *net, double offset);
extern CASCADE_NET *add_noise(CASCADE_NET *net, double max_noise);
extern double *net_output(CASCADE_NET *net, double *inputs, double *outputs);
extern int net_output_class(CASCADE_NET *net, double *inputs);
extern double *net_output_multi(CASCADE_NET *class, double *class_inputs,
               CASCADE_NET *net, double *inputs,
               double *outputs);
extern int get_type(char *type);
extern int get_type_string(int type, char *str);
extern double random_noise(double range);
extern double compute_unit_value(CASCADE_NET *net, double *values,
             int unit, int unit_type);
```

```
extern double dsigmoid(double x);
extern double sigmoid(double x);

extern double dsigmoid2(double x);
extern double sigmoid2(double x);

extern double dcarunit(double x);
extern double carunit(double x);

extern double dgaussian(double x);
extern double gaussian(double x);

extern double dsin(double x);
extern double dcos(double x);

extern double dsin2(double x);
extern double sin2(double x);

extern double dcos2(double x);
extern double cos2(double x);

extern double dgtsin(double x);
extern double gtsin(double x);

extern double dlinear(double x);
extern double linear(double x);

extern double dbessel0(double x);
extern double bessel0(double x);

extern double dbessel1(double x);
extern double bessel1(double x);

extern double *bp_output(BP_NET *net, double *inputs, double *outputs);
extern int read_bp_net(BP_NET *net, char *file_name);
extern void connect_layers(BP_NET *net,
             int start1, int end1, int start2, int end2);
extern void build_data_structures(BP_NET *net);
extern int clear_bp_net(BP_NET *net);

extern int initialize_functions(int use_lookup);
extern int initialize_lookup(void);
extern int generate_lookup(Lookup *lu, double (*f)(), double min, double max);

extern double sin_t(double x);
extern double dsin_t(double x);
extern double cos_t(double x);
extern double dcos_t(double x);
extern double sin2_t(double x);
extern double dsin2_t(double x);
extern double cos2_t(double x);
extern double dcos2_t(double x);
extern double sigmoid_t(double x);
extern double dsigmoid_t(double x);
extern double sigmoid2_t(double x);
extern double dsigmoid2_t(double x);
extern double carunit_t(double x);
extern double dcarunit_t(double x);
extern double gaussian_t(double x);
extern double dgaussian_t(double x);
extern double gtsin_t(double x);
extern double dgtsin_t(double x);
extern double bessel0_t(double x);
extern double dbessel0_t(double x);
extern double bessel1_t(double x);
extern double dbessel1_t(double x);
```

kalman_main.c

```
/*****************************************************************************/
```

```
/*                                                                            */
/* Description: Cascade 2 w/extended Kalman filtering                         */
/* File: "kalman_main.c"                                                      */
/*                                                                            */
/* Author: Michael C. Nechyba                                                 */
/* Last modified: 6/21/97                                                     */
/*                                                                            */
/******************************************************************************/

#include <stdio.h>
#include <math.h>
#include "matrix.h"
#include "file_io.h"
#include "gen_math.h"
#include "nn_util.h"
#include "kalman.h"
#include "nn_util_main.h"
#include "kalman_main.h"

/************************** Main Program ***********************************/

int main(int argc, char **argv)
{
  int i;
  GlobalOptions *gl;
  CASCADE_NET *net;
  char str[STR_REG];

  gl = &Global;
  net = &gl->net[0];

  gl->n_parameters = sizeof(ParameterTable)/sizeof(PARAMETERS);
  gl->parameters = ParameterTable;
  gl->parameter_file[0] = (char) NULL;

  if(argc < 2) {
    fprintf(stderr, "-- Error: Too few arguments. Exiting.\n");
    exit(-2);
  }

  if(argv[1][0] != '-') {
    strcpy(&gl->parameter_file[0], argv[1]);

    if(!file_exist(&gl->parameter_file[0])) {
      fprintf(stderr, "-- Error: Parameter file does not exist. Exiting.\n");
      exit(-2);
    }

    gl->argc = --argc;
    gl->argv = ++argv;
  } else {
    gl->argc = argc;
    gl->argv = argv;
  }

  initialize_kalman_net(gl);
  initialize_kalman_filter(net, &gl->kl, -1);

  srandom(gl->random_seed);

  for(i = 0; i < gl->trials; i++) {
    gl->trial_number = i+1;
    if(gl->active_train < 0)
      kalman_train(gl);
    else
      kalman_active_train(gl);
  }
}

/************************** End Main Program *******************************/
```

kalman_opt.c

```
/*****************************************************************************/
/*                                                                           */
/* Title: Extended Kalman filtering for Cascade neural networks              */
/* File: "kalman_opt.c"                                                      */
/*                                                                           */
/* Author: Michael C. Nechyba                                          */
/* Last modified: 6/23/97                                                */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "statistics.h"
#include "matrix.h"
#include "file_io.h"
#include "nn_util.h"
#include "gen_math.h"
#include "kalman.h"
#include "arg.h"

#ifndef DSPSA
/*****************************************************************************/
/*
   kalman_train().
 */
/*****************************************************************************/

int kalman_active_train(GlobalOptions *gl)
{
  int i;

  for(i = 0; i < gl->active_init; i++) {
    /*
       select initial vectors
       */
  }

  /*
    train on current data set
    */

  return(1);
}

int kalman_train(GlobalOptions *gl)
{
  double *p, sum_off, sum_diag;
  int i, j, k, n_max, type, best_candidate, best_type, begin = 1, flag = -1;
  double *w, err, min, prev = 10000000.;
  CASCADE_NET *net;
  char str[STR_REG];

  net = &gl->net[0];
  n_max = net->n_inputs + net->max_hidden;
  w = allocate_vector(w, n_max + net->n_outputs);


  if(net->n_hidden == 0) {
    /* Train initial output connections */

    if(gl->no_linear == 0) {
      if(gl->report == REPORT_EACH_EPOCH) {
    fprintf(stdout, "#");
    display_error(stdout, gl, net, get_type(gl->output_type), 0);
      }
      kalman_one_unit(gl, 0);
    }
```

```
  for(i = 0; i < gl->n_candidates; i++)
    copy_kalman_net(&gl->net[i], net);

  display_error(stdout, gl, net, get_type(gl->output_type), 0);

  if(gl->dump_weights > 0) {
    sprintf(str, "%s.%d.%d", &gl->cascade_file[0], gl->trial_number, 0);
    write_net(net, str);
  }

} else {
  if(gl->pattern_wise_error > 0) {
    error_statistics(gl, net, &gl->err_tst);

    for(i = 0; i < gl->n_test_patterns; i++)
  for(j = 0; j < net->n_outputs; j++) {
    if(j == net->n_outputs - 1)
      fprintf(stdout, "%.9f\n", gl->err_tst.vec[j][i]);
    else
      fprintf(stdout, "%.9f ", gl->err_tst.vec[j][i]);
  }

    exit(1);
  } else {
    begin = net->n_hidden + 1;
    display_error(stdout, gl, net, net->type[net->n_hidden - 1],
          net->n_hidden);
  }
}

for (k = begin; k <= net->max_hidden; k++) {

  /* Train additional hidden units */

  if(gl->use_cache == 1 && k > 1) {
    for(i = 0; i < gl->n_train_patterns; i++)
  kalman_unit_activations(net,
             &gl->trn[i][0],
             &gl->cache_activations[i][0],
             &gl->cache_net_act[i][0]);
  }

  min = 1000000000.;
  best_candidate = 0;

  for(i = 0; i < gl->n_candidates; i++) {
    random_vector(w, net->n_inputs + net->n_hidden + net->n_outputs,
          -fabs(gl->weight_range), fabs(gl->weight_range));

    type = get_type(gl->unit_type) == MIXED ? (i % NUMBER_OF_UNITS) :
  get_type(gl->unit_type);

    add_hidden_unit(&gl->net[i], type, w);
    kalman_one_unit(gl, i);

    err = quick_rms_error(&gl->net[i], gl->cross, gl->n_cross_patterns);

    if(err < min) {
  min = err;
  best_candidate = i;
  best_type = type;
    }
  }

  if(gl->report == REPORT_COVARIANCE)
    fprintf(stdout, "Best candidate = %d\n", best_candidate);

  for(i = 0; i < gl->n_candidates; i++)
    copy_kalman_net(&gl->net[i], &gl->net[best_candidate]);

  display_error(stdout, gl, net, best_type, k);

  if(gl->error_stop == 1 && min > prev)
    k = net->max_hidden + 1;

  prev = min;
```

```
    if(gl->dump_weights > 0) {
      sprintf(str, "%s.%d.%d", &gl->cascade_file[0], gl->trial_number, k);
      write_net(net, str);
    }
  }

  if(gl->existing_file[0] == (char) NULL ||
     (!file_exist(&gl->existing_file[0]))) {
    for(i = 0; i < gl->n_candidates; i++) {
      clear_kalman_net(&gl->net[i]);
      create_kalman_net(&gl->net[i],
            gl->n_inputs, gl->max_hidden, gl->n_outputs,
            gl->weight_range, get_type(gl->output_type));
    }

    clear_kalman_net(&gl->best_weights);
    create_kalman_net(&gl->best_weights,
              gl->n_inputs, gl->max_hidden, gl->n_outputs,
              gl->weight_range, get_type(gl->output_type));
  } else {
    for(i = 0; i < gl->n_candidates; i++) {
      clear_kalman_net(&gl->net[i]);
      flag =
    read_kalman_net(&gl->net[i], &gl->existing_file[0], gl->max_hidden);
      if(flag != 1) {
    fprintf(stderr, "Cannot read in network file. Exiting.\n");
    exit(-1);
      }
    }

    clear_kalman_net(&gl->best_weights);
    flag = read_kalman_net(&gl->best_weights, &gl->existing_file[0],
               gl->max_hidden);
    if(flag != 1) {
      fprintf(stderr, "Cannot read in network file. Exiting.\n");
      exit(-1);
    }
  }

  gl->n_epochs = 0;

  free(w);

  return(1);
}
#endif

/*****************************************************************************/
/*
   error_statistics().
 */
/*****************************************************************************/

double error_statistics(GlobalOptions *gl, CASCADE_NET *net, Error *e)
{
  int i;
  double min, max, rms = 0.;

  sse_error(net, e);

  for(i = 0; i < net->n_outputs; i++) {
    e->rms[i] = sqrt(e->sse[i]/((double) e->n_vec));
    rms += e->sse[i];
  }

  rms = sqrt(rms/((double) e->n_vec));

  if(gl->display_max_err > 0) {
    for(i = 0; i < net->n_outputs; i++) {
      min = fabs(min_vec(e->vec[i], e->n_vec));
      max = fabs(max_vec(e->vec[i], e->n_vec));
      e->max[i] = max_real(min, max);
    }
  }
```

```
  if(gl->display_std_dev > 0) {
    for(i = 0; i < net->n_outputs; i++)
      e->stddev[i] = stat_stddev(e->vec[i], e->n_vec);
  }

  if(gl->display_bit_err > 0) {
    bit_error(net, e);
  }

  if(gl->display_histogram > 1) {
    for(i = 0; i < net->n_outputs; i++)
      histogram(&e->histogram[i][0], e->vec[i], e->n_vec,
        gl->display_histogram, gl->histogram_min,
        gl->histogram_max);
  }

  return(rms);
}

/*****************************************************************************/
/*
   display_error().
 */
/*****************************************************************************/

int display_error(FILE *fp, GlobalOptions *gl, CASCADE_NET *net, int type,
          int n)
{
  int i, j, ne;
  char str[STR_REG], str2[STR_REG];
  double rms[3], sse[3], tmp[3];
  Error *e[3];

  get_type_string(type, &str[0]);
  ne = gl->n_epochs;

  e[0] = &gl->err_trn;
  e[1] = &gl->err_cvd;
  e[2] = &gl->err_tst;

  for(i = 0; i < 3; i++) {
    rms[i] = error_statistics(gl, net, e[i]);
    sse[i] = rms[i]*rms[i]*e[i]->n_vec;
  }

  dsp(fp, str, ne, n, "RMS", 9, &rms[0]);
  if(gl->display_sse > 0)
    dsp(fp, str, ne, n, "SSE", 6, &sse[0]);

  if(gl->display_per_output > 0 && net->n_outputs > 1) {
    for(i = 0; i < net->n_outputs; i++) {
      for(j = 0; j < 3; j++)
    tmp[j] = e[j]->rms[i];

      sprintf(str2, "PerRMS %d", i);
      dsp(fp, str, ne, n, str2, 9, &tmp[0]);
    }

    if(gl->display_sse > 0) {
      for(i = 0; i < net->n_outputs; i++) {
    for(j = 0; j < 3; j++)
      tmp[j] = e[j]->sse[i];

    sprintf(str2, "PerSSE %d", i);
    dsp(fp, str, ne, n, str2, 6, &tmp[0]);
      }
    }
  }

  if(gl->display_max_err > 0) {
    for(i = 0; i < net->n_outputs; i++) {
      for(j = 0; j < 3; j++)
    tmp[j] = e[j]->max[i];

      sprintf(str2, "PerMAX %d", i);
      dsp(fp, str, ne, n, str2, 9, &tmp[0]);
```

```
    }
  }

  if(gl->display_std_dev > 0) {
    for(i = 0; i < net->n_outputs; i++) {
    for(j = 0; j < 3; j++)
    tmp[j] = e[j]->stddev[i];

    sprintf(str2, "PerSTDDEV %d", i);
    dsp(fp, str, ne, n, str2, 9, &tmp[0]);
    }
  }

  if(gl->display_bit_err > 0) {
    for(i = 0; i < 3; i++)
      tmp[i] = e[i]->bit.pct_classify;

    dsp(fp, str, ne, n, "ClassErrPct", 2, &tmp[0]);

    for(i = 0; i < 3; i++)
      tmp[i] = e[i]->bit.pct_false_off;

    dsp(fp, str, ne, n, "FalseOffPct", 2, &tmp[0]);

    for(i = 0; i < 3; i++)
      tmp[i] = e[i]->bit.pct_false_on;

    dsp(fp, str, ne, n, "FalseOnPct", 2, &tmp[0]);
  }

  if(gl->display_histogram > 1) {
    for(i = 0; i < net->n_outputs; i++) {
      fprintf(fp, "## %d ( %s - Epoch %d - Histogram Train %d): ",
          n, str, gl->n_epochs, i);
      display_int_vector(fp, &e[0]->histogram[i][0], gl->display_histogram, 0);
    }

    for(i = 0; i < net->n_outputs; i++) {
      fprintf(fp, "## %d ( %s - Epoch %d - Histogram Cross %d): ",
          n, str, gl->n_epochs, i);
      display_int_vector(fp, &e[1]->histogram[i][0], gl->display_histogram, 0);
    }

    for(i = 0; i < net->n_outputs; i++) {
      fprintf(fp, "## %d ( %s - Epoch %d - Histogram Test  %d): ",
          n, str, gl->n_epochs, i);
      display_int_vector(fp, &e[2]->histogram[i][0], gl->display_histogram, 0);
    }
  }

  return(1);
}

/*****************************************************************************/
/*
   dsp();
 */
/*****************************************************************************/

int dsp(FILE *fp, char *type, int n_epochs, int n, char *which, int precision,
    double *dat)
{
  int i;
  char format[STR_REG], format2[STR_REG];

 sprintf(format, "%%.%df ", precision);
 sprintf(format2, "%%.%df\n", precision);

 fprintf(fp, "# %d ( %s - Epoch %d - %s): ", n, type, n_epochs, which);

  for(i = 0; i < 2; i++)
    fprintf(fp, format, dat[i]);

  fprintf(fp, format2, dat[2]);

  return(1);
```

```
}

/*****************************************************************************/
/*
   bit_error().
 */
/*****************************************************************************/

double bit_error(CASCADE_NET *net, Error *e)
{
  int i, j, n_inm1;
  double *out, *des, *in;
  BitError *b;

  b = &e->bit;

  b->false_off = 0;
  b->false_on  = 0;
  b->total     = 0;
  b->classify  = 0;

  out = allocate_vector(out, net->n_outputs);
  n_inm1 = net->n_inputs - 1;

  for (i = 0; i < e->n_vec; i++) {

    in = &e->dat[i][0]; des = &e->dat[i][n_inm1];
    net_output(net, in, out);

    if(net->n_outputs > 1) {
      if(ind_max_vec(out, net->n_outputs) !=
     ind_max_vec(des, net->n_outputs))
    b->classify++;
    } else {
      if(sign(out[0]) != sign(des[0]))
    b->classify++;
    }

    for(j = 0; j < net->n_outputs; j++) {
      b->total++;

      if(sign(out[j]) != sign(des[j])) {
    if(sign(out[j]) > sign(des[j]))
      b->false_on++;
    else
      b->false_off++;
      }
    }
  }

  free(out);

  b->pct_false_off = PCT(b->false_off, b->total);
  b->pct_false_on  = PCT(b->false_on,  b->total);
  b->pct_classify  = PCT(b->classify,  e->n_vec);

  return(b->pct_classify);
}

/*****************************************************************************/
/*
   sse_error().
 */
/*****************************************************************************/

int sse_error(CASCADE_NET *net, Error *e)
{
  int i, j, n_inm1;
  double err = 0.0, *out, *des, *in, tmp;

  out = allocate_vector(out, net->n_outputs);
  n_inm1 = net->n_inputs - 1;

  for(j = 0; j < net->n_outputs; j++)
    e->sse[j] = 0.;
```

```
  for (i = 0; i < e->n_vec; i++) {

    in = &e->dat[i][0]; des = &e->dat[i][n_inm1];
    net_output(net, in, out);

    for(j = 0; j < net->n_outputs; j++) {
      tmp = e->vec[j][i] = (des[j] - out[j]);
      e->sse[j] += tmp*tmp;
    }
  }

  free(out);
}

/*****************************************************************************/
/*
   quick_rms_error().
 */
/*****************************************************************************/

double quick_rms_error(CASCADE_NET *net, double **dat, int n_dat)
{
  int i, j, n_inm1;
  double err = 0.0, *out, *des, *in, tmp;

  out = allocate_vector(out, net->n_outputs);
  n_inm1 = net->n_inputs - 1;

  for (i = 0; i < n_dat; i++) {

    in = &dat[i][0]; des = &dat[i][n_inm1];
    net_output(net, in, out);

    for(j = 0; j < net->n_outputs; j++) {
      tmp = (des[j] - out[j]);
      err += tmp*tmp;
    }

  }

  free(out);

  /* currently not multiplied by net->n_outputs */

  return(sqrt(err/((double) (n_dat))));
}

/*****************************************************************************/
/*
   zero_linear_connections().
 */
/*****************************************************************************/

int zero_linear_connections(CASCADE_NET *net)
{
  int i, j, ni;
  double *w;

  ni = net->n_inputs - 1;

  for(i = 0; i < net->n_outputs; i++) {

    w = net->weights[i + net->n_hidden];

    for(j = 0; j < ni; j++)
      *(w++) = 0.;
  }

  return(1);
}

#ifndef DSPSA
/*****************************************************************************/
/*
   kalman_one_unit().
 */
```

```
/*****************************************************************************/

int kalman_one_unit(GlobalOptions *gl, int which)
{
  int i, j, k, count = 0, done = 0, no_hidden, *type;
  int n_inm1, n_out, n_hid, n_hidm1, n_unt, n_untm1;
  int nu, nm;
  double net_act, weight_gain, p_gain, derr, decay;
  double *in, *out, *err, **trn;
  double **cch_act, **cch_na;
  double *u, *v, *wv, **w, *psi, *po, **p, *na;
  double **ai, **a, *ae;
  CASCADE_NET *net;
  KalmanFilter *kl;
  Func *f;

  net = &gl->net[which];
  kl = &gl->kl;
  f = &Function[0];

  n_inm1 = net->n_inputs - 1;
  n_out = net->n_outputs;
  n_hid = net->n_hidden;
  n_hidm1 = n_hid - 1;
  n_unt = net->n_units;
  n_untm1 = n_unt - 1;

  no_hidden = n_hid == 0 ? 1 : 0;

  if(gl->no_linear > 0) {
    zero_linear_connections(net);
    nu = n_inm1;
    nm = n_hid + 1;
  } else {
    nu = 0;
    nm = n_unt;
  }

  for(i = 0; i < kl->n_nodes; i++) {
    kl->p[i] = identity_matrix(kl->p[i], n_unt);
    scalar_multiply(kl->p[i], kl->p[i], n_unt, n_unt, 1./kl->eta_p);
  }

  while(done == 0) {
    gl->n_epochs++;
    decay = exp((double) -count * kl->time_constant);

    cch_act = gl->cache_activations;
    cch_na = gl->cache_net_act;
    trn = gl->trn;

    for(i = 0; i < gl->n_train_patterns; i++) {
      /*
#ifdef MANY_OUTPUTS
*/
      if(gl->report == REPORT_EACH_PATTERN) {
    fprintf(stdout, "*%d ", i);
    display_error(stdout, gl, net, net->type[n_hidm1], which);
      }
      /*
#endif
*/
      in = (gl->random_order > 0) ? gl->trn[gl->index[i]] : *(trn++);
      out = in + n_inm1;

      if(gl->use_cache == 1 && n_hid > 1) {
    kalman_unit_activations_with_cache(net, in, *(cch_act), *(cch_na));
    kl->u = *(cch_act++);
    kl->net_act = *(cch_na++);
      } else {
    kalman_unit_activations(net, in, kl->u, kl->net_act);
      }

      net_act = *(kl->net_act + n_hidm1);

      err = kl->err;
```

```
      u = kl->u + n_unt;

      for(j = 0; j < n_out; j++)
    *(err++) = *(out++) - (*(u++));

      ai = kl->tm[0];
      ae = kl->tv[0];

      if(no_hidden == 0) {
    psi = kl->psi[0];
    po = kl->psi_out;
    type = net->type + n_hid;
    na = kl->net_act + n_hid;
    w = net->weights + n_hid;
    for(j = 0; j < n_out; j++)
      *(psi++) =
        (*(po++) = f[*(type++)].dact_fnc(*(na++)))*MAT(w, n_untm1)*
       f[*(net->type + n_hidm1)].dact_fnc(net_act);

    vector_to_matrix(kl->psim[0], kl->psi[0], n_out);

    matrix_vector_product(&kl->v[0][0], kl->p[0], kl->u, n_untm1, n_untm1);
    scalar_multiply(ai, kl->psim[0],
           n_out, n_out, dot(kl->u, &kl->v[0][0], n_untm1));

      } else {
    type = net->type + n_hid;
    na = kl->net_act + n_hid;
    po = kl->psi_out;

    for(j = 0; j < n_out; j++)
      *(po++) = f[*(type++)].dact_fnc(*(na++));

    fill_matrix(ai, n_out, n_out, 0.);
      }

      po = kl->psi_out;

      for(j = 1; j < kl->n_nodes; j++) {
    matrix_vector_product(&kl->v[j][0], kl->p[j], &kl->u[nu], nm, nm);

    MAT(ai, j - 1) += (dot(&kl->u[nu], kl->v[j], nm)
              *(*(po))*(*(po++))
              + kl->r[j-1]*decay);
      }

#ifdef MANY_OUTPUTS
      inverse_new(kl->a, kl->tm[0], n_out, &gl->d);
#else
      inverse_sym(kl->a, kl->tm[0], n_out);
#endif

      if(no_hidden == 0) {
    p = kl->p[0];

    matrix_vector_product(ae, kl->a, kl->err, n_out, n_out);
    weight_gain = dot(ae, kl->psi[0], n_out);
    matrix_vector_product(kl->tv[1], kl->a, kl->psi[0], n_out, n_out);
    p_gain = -dot(kl->tv[1], kl->psi[0], n_out);

    wv = net->weights[n_hidm1]; v = kl->v[0];
    for(j = 0; j < n_untm1; j++)
      *(wv++) += weight_gain*(*(v++));

    vector_to_matrix_diag(kl->vm, &kl->v[0][0], n_untm1);
    scalar_multiply_diag(kl->vm, kl->vm, n_untm1, p_gain);
    matrix_add_diag(p, p, kl->vm, n_untm1);
    upper_diag_to_sym(p, n_untm1);

    for(j = 0; j < n_untm1; j++)
      MAT(p, j) += kl->eta_q;

      } else {
    matrix_vector_product(ae, kl->a, kl->err, n_out, n_out);
      }
```

```
    a = kl->a;
    po = kl->psi_out;

    for(j = 1; j < kl->n_nodes; j++) {
  p = kl->p[j];
  weight_gain =  (*(ae++))*(*po);
  p_gain = -MAT(a, j-1)*(*po)*(*(po++));

  wv = &net->weights[n_hidm1 + j][nu]; v = kl->v[j];
  for(k = 0; k < nm; k++)
    *(wv++) += weight_gain*(*(v++));

  vector_to_matrix_diag(kl->vm, kl->v[j], nm);
  scalar_multiply_diag(kl->vm, kl->vm, nm, p_gain);
  matrix_add_diag(p, p, kl->vm, nm);
  upper_diag_to_sym(p, nm);

  for(k = 0; k < nm; k++)
    MAT(p, k) += kl->eta_q;
    }
  }

  if(gl->report == REPORT_EACH_EPOCH) {
    fprintf(stdout, "#");
    if(no_hidden == 0)
  display_error(stdout, gl, net, net->type[n_hidm1], which);
    else
  display_error(stdout, gl, net, get_type(gl->output_type), 0);
  }

  gl->err[count] = quick_rms_error(net, gl->cross, gl->n_cross_patterns);

  if(ind_min_vec(gl->err, count + 1) == count)
    copy_kalman_net(&gl->best_weights, net);

  if(count >= gl->change_period) {
    derr = (gl->err[count] - gl->err[count - gl->change_period])/
  gl->err[count];

    if(derr > 0. || fabs(derr) < gl->change_threshold ||
   count >= gl->max_iterate - 1) {
  done = 1;
  copy_kalman_net(net, &gl->best_weights);
    }
  }


  count++;
}

if(gl->report == REPORT_COVARIANCE) {
  double diag = 0., off_diag = 0.;

  if(no_hidden == 0) {
    for(i = 0; i < n_untm1; i++)
  for(j = 0; j < n_untm1; j++) {
    if(i == j)
      diag += fabs(kl->p[0][i][j]);
    else
      off_diag += fabs(kl->p[0][i][j]);
  }
  }

  for(k = 1; k < kl->n_nodes; k++)
    for(i = 0; i < n_unt; i++)
  for(j = 0; j < n_unt; j++) {
    if(i == j)
      diag += fabs(kl->p[k][i][j]);
    else
      off_diag += fabs(kl->p[k][i][j]);
  }

  fprintf(stdout, "*%d %lf %lf %lf\n", which, off_diag, diag, off_diag/diag);
}

return(1);
```

```
}
#endif

/*****************************************************************************/
/*
   kalman_unit_activations calculates the net input activation for each unit
   as well as the activation for each unit.
 */
/*****************************************************************************/

int kalman_unit_activations(CASCADE_NET *net, double *inputs,
               double *activations, double *net_act)
{
  int i, j, tmp;
  double sum, *w, *a;

  a = activations;
  for(i = 0; i < net->n_units; i++) {
    if(i < net->n_inputs-1)
      *(a++) = *(inputs++);
    else if (i == net->n_inputs - 1)
      *a = 1.0;
    else  {
      tmp = i - net->n_inputs;
      sum = 0.0;

      a = activations;
      w = net->weights[tmp];

      for(j = 0; j < i; j++)
    sum += ((*(a++))*(*(w++)));

      net_act[tmp] = sum;
      activations[i] = Function[net->type[tmp]].act_fnc(sum);
    }
  }

  for(i = 0; i < net->n_outputs; i++) {
    tmp = i + net->n_hidden;
    sum = 0.;

    a = activations;
    w = net->weights[tmp];

    for(j = 0; j < net->n_units; j++)
      sum += ((*(a++))*(*(w++)));

    net_act[tmp] = sum;
    activations[i + net->n_units] = Function[net->type[tmp]].act_fnc(sum);
  }

  return(1);
}

/*****************************************************************************/
/*
   kalman_unit_activations_with_cache does the same as
   kalman_unit_activations but assumes that all activations and net
   activations have already been calculated except for the last hidden
   unit and the output units.
 */
/*****************************************************************************/

int kalman_unit_activations_with_cache(CASCADE_NET *net, double *inputs,
                       double *activations, double *net_act)
{
  int i, j, unit, net_unit;
  double sum = 0.0, *w, *a;

  unit = net->n_units - 1;
  net_unit =  unit - net->n_inputs;

  if(net_unit >= 0) {
    a = activations;
    w = net->weights[net_unit];
```

```
    for(i = 0; i < unit; i++)
      sum += ((*(a++))*(*(w++)));

    net_act[net_unit] = sum;
    activations[unit] = Function[net->type[net_unit]].act_fnc(sum);
  }

  for(i = 0; i < net->n_outputs; i++) {
    net_unit = i + net->n_units - net->n_inputs;
    sum = 0.0;

    a = activations;
    w = net->weights[i + net->n_hidden];

    for(j = 0; j < net->n_units; j++)
      sum += ((*(a++))*(*(w++)));

    net_act[net_unit] = sum;
    activations[i + net->n_units] = Function[net->type[net_unit]].act_fnc(sum);
  }

  return(1);
}

#ifndef DSPSA
/*****************************************************************************/
/*
   initialize_kalman_filter() reserves and initialize all memory necessary
   for kalman_filtering.
 */
/*****************************************************************************/

#define TMP_KLM_MAX 2

int initialize_kalman_filter(CASCADE_NET *net, KalmanFilter *kl, int n_nodes)
{
  int i, j, n_max, n_out, net_act;

  n_out = net->n_outputs;
  n_max = net->n_inputs + net->max_hidden;
  net_act = net->max_hidden + net->n_outputs;

  kl->n_nodes = n_nodes < 0 ? n_out + 1 : n_nodes;
  kl->p = (double ***) malloc(kl->n_nodes * sizeof(double **));

  for(i = 0; i < kl->n_nodes; i++) {
    kl->p[i] = allocate_matrix(kl->p[i], n_max, n_max);
  }

  kl->r = allocate_vector(kl->r, n_out);
  for(i = 0; i < n_out; i++)
    kl->r[i] = kl->eta_r;

  kl->a = allocate_matrix(kl->a, n_out, n_out);
  kl->ai = allocate_matrix(kl->ai, n_out, n_out);

  kl->err = allocate_vector(kl->err, n_out);

  kl->psi = allocate_matrix(kl->psi, kl->n_nodes, n_out);
  kl->psi_out = allocate_vector(kl->psi_out, n_out);

  kl->psim = (double ***) malloc(kl->n_nodes * sizeof(double **));
  for(i = 0; i < kl->n_nodes; i++)
    kl->psim[i] = allocate_matrix(kl->psim[i], n_out, n_out);

  kl->v = allocate_matrix(kl->v, kl->n_nodes, n_max);
  kl->vm = allocate_matrix(kl->vm, n_max, n_max);
  kl->u = allocate_vector(kl->u, n_max + n_out);
  kl->net_act = allocate_vector(kl->net_act, net_act);
  kl->net_deriv = allocate_matrix(kl->net_deriv, net_act, net_act);

  kl->tm = (double ***) malloc(TMP_KLM_MAX * sizeof(double **));

  for(i = 0; i < TMP_KLM_MAX; i++)
    kl->tm[i] = allocate_matrix(kl->tm[i], n_max, n_max);
```

```
  kl->tv = allocate_matrix(kl->tv, TMP_KLM_MAX, n_max);

  return(1);
}

/*****************************************************************************/
/*
   initialize_kalman_net() reads in a parameter file and initializes all
   global variables.
 */
/*****************************************************************************/

int initialize_kalman_net(GlobalOptions *gl)
{
  int i, j, n_max, flag = -1;

  strcpy(&gl->unit_type[0], "SIGMOID");
  strcpy(&gl->output_type[0], "LINEAR");

  gl->report = DONT_REPORT;
  gl->save_noise_files = -1;

  gl->n_inputs = -1;
  gl->n_outputs = -1;
  gl->max_hidden = -1;
  gl->test_file[0] = (char) NULL;
  gl->cross_file[0] = (char) NULL;
  gl->train_file[0] = (char) NULL;
  gl->existing_file[0] = (char) NULL;
  gl->pattern_wise_error = -1;
  gl->error_stop = 1;
  gl->use_cache = 1;
  gl->use_lookup = 1;
  gl->trials = 1;

  /* detailed error display flags */

  gl->display_per_output = -1;
  gl->display_sse = -1;
  gl->display_max_err = -1;
  gl->display_std_dev = -1;
  gl->display_histogram = -1;
  gl->histogram_min = 1.;
  gl->histogram_min = -1.;

  /* active data selection options */

  gl->active_train = -1;
  gl->active_group = 1;
  gl->active_init = 1;
  gl->active_tolerance = 0.1;

  /* end -- active data selection options */

  gl->n_cross_patterns = -1;
  gl->n_test_patterns = -1;
  gl->n_train_patterns = -1;
  gl->n_epochs = 0;

  gl->dump_weights = 1;
  strcpy(&gl->cascade_file[0], "network");

  gl->max_iterate = 1;
  gl->change_threshold = .01;
  gl->change_period = 1;

  gl->weight_range = 1.;
  gl->input_noise = 0.;
  gl->output_noise = 0.;
  gl->random_seed = 1;
  gl->random_order = -1;
  gl->n_candidates = 1;
  gl->no_linear = 0;
  gl->display_bit_err = 1;
```

```
gl->kl.eta_p = 0.01;
gl->kl.eta_q = 0.0001;
gl->kl.eta_r = 1.;
gl->kl.time_constant = 0.;

if(file_exist(&gl->parameter_file[0]))
  get_parameters(gl->parameter_file, gl->parameters, gl->n_parameters);

process_command_line(gl, gl->argc, gl->argv);
/* display_parameters(gl->parameters, gl->n_parameters, stderr); */

if(gl->n_inputs == -1 ||
   gl->n_outputs == -1 ||
   gl->max_hidden == -1 ||
   gl->train_file[0] == (char) NULL) {
  fprintf(stderr,
      "Fatal error: key parameter is missing. Exiting.\n");
  exit(-1);
}

initialize_functions(gl->use_lookup);

if(gl->max_iterate > MAX_ITERATE)
  gl->max_iterate = MAX_ITERATE;

if(gl->n_train_patterns < 1)
  gl->n_train_patterns = count_lines(gl->train_file);

gl->trn = read_matrix_file(gl->train_file, gl->trn,
              gl->n_train_patterns,
              gl->n_inputs + gl->n_outputs);

gl->err_trn.n_vec = gl->n_train_patterns;
gl->err_trn.vec = allocate_matrix(gl->err_trn.vec, gl->n_outputs,
                 gl->err_trn.n_vec);
gl->err_trn.dat = gl->trn;

if(gl->random_order > 0) {
  gl->index = (int *) malloc(sizeof(int) * gl->n_train_patterns);
  for(i = 0; i < gl->n_train_patterns; i++)
    gl->index[i] = i;

  randomize_index(gl->index, gl->n_train_patterns);
}

if(gl->cross_file[0] == (char) NULL ||
   (!strcmp(&gl->cross_file[0], &gl->train_file[0]))) {
  fprintf(stderr, "Cross validation file will be same as training file.\n");
  gl->n_cross_patterns = gl->n_train_patterns;
  gl->cross = gl->trn;

  gl->err_cvd.n_vec = gl->n_cross_patterns;
  gl->err_cvd.vec = gl->err_trn.vec;
  gl->err_cvd.dat = gl->err_trn.dat;
} else {
  if(gl->n_cross_patterns < 1)
    gl->n_cross_patterns = count_lines(gl->cross_file);

  gl->cross = read_matrix_file(gl->cross_file, gl->cross,
              gl->n_cross_patterns,
              gl->n_inputs + gl->n_outputs);

  gl->err_cvd.n_vec = gl->n_cross_patterns;
  gl->err_cvd.vec = allocate_matrix(gl->err_cvd.vec, gl->n_outputs,
                  gl->err_cvd.n_vec);
  gl->err_cvd.dat = gl->cross;
}

if(gl->test_file[0] == (char) NULL ||
   (!strcmp(&gl->test_file[0], &gl->cross_file[0]))) {
  fprintf(stderr, "Test file will be the same as cross validation file.\n");
  gl->n_test_patterns = gl->n_cross_patterns;
  gl->tst = gl->cross;

  gl->err_tst.n_vec = gl->n_test_patterns;
  gl->err_tst.vec = gl->err_cvd.vec;
```

```
  gl->err_tst.dat = gl->err_cvd.dat;
} else {
  if(gl->n_test_patterns < 1)
    gl->n_test_patterns = count_lines(gl->test_file);

  gl->tst = read_matrix_file(gl->test_file, gl->tst,
               gl->n_test_patterns,
               gl->n_inputs + gl->n_outputs);

  gl->err_tst.n_vec = gl->n_test_patterns;
  gl->err_tst.vec = allocate_matrix(gl->err_tst.vec, gl->n_outputs,
                 gl->err_tst.n_vec);
  gl->err_tst.dat = gl->tst;
}

if(!is_zero(gl->input_noise)) {
  matrix_add_noise(gl->trn,
          gl->n_train_patterns, 0,
          gl->n_inputs, gl->input_noise);

  if(gl->cross != gl->trn)
    matrix_add_noise(gl->cross,
            gl->n_cross_patterns, 0,
            gl->n_inputs, gl->input_noise);

  if(gl->cross != gl->tst)
    matrix_add_noise(gl->tst,
            gl->n_test_patterns, 0,
            gl->n_inputs, gl->input_noise);
}

if(!is_zero(gl->output_noise)) {
  matrix_add_noise(gl->trn,
          gl->n_train_patterns, gl->n_inputs,
          gl->n_inputs + gl->n_outputs, gl->output_noise);

  if(gl->cross != gl->trn)
    matrix_add_noise(gl->cross,
            gl->n_cross_patterns, gl->n_inputs,
            gl->n_inputs + gl->n_outputs, gl->output_noise);

  if(gl->cross != gl->tst)
    matrix_add_noise(gl->tst,
            gl->n_test_patterns, gl->n_inputs,
            gl->n_inputs + gl->n_outputs, gl->output_noise);
}

if(!is_zero(gl->output_noise) || !is_zero(gl->input_noise)) {
  if(gl->save_noise_files > 0) {
    FILE *fp;

    fp = fopen("tmp.trn", "w");
    display_matrix(fp, gl->trn, gl->n_train_patterns,
          gl->n_inputs + gl->n_outputs, 6);
    fclose(fp);

    if(gl->cross != gl->trn) {
  fp = fopen("tmp.cvd", "w");
  display_matrix(fp, gl->cross, gl->n_cross_patterns,
            gl->n_inputs + gl->n_outputs, 6);
  fclose(fp);
    }

    if(gl->cross != gl->tst) {
  fp = fopen("tmp.tst", "w");
  display_matrix(fp, gl->tst, gl->n_test_patterns,
            gl->n_inputs + gl->n_outputs, 6);
  fclose(fp);
    }

    fprintf(stderr, "Saved noise files to tmp.trn tmp.cvd tmp.tst.\n");
    exit(1);
  }
}

n_max = gl->max_hidden + gl->n_inputs + 1;
```

```
  if(gl->use_cache == 1) {
    gl->cache_activations = allocate_matrix(gl->cache_activations,
                      gl->n_train_patterns,
                      n_max + gl->n_outputs);

    gl->cache_net_act = allocate_matrix(gl->cache_net_act,
                  gl->n_train_patterns,
                  gl->max_hidden + gl->n_outputs);
  }


  if(gl->existing_file[0] == (char) NULL ||
     (!file_exist(&gl->existing_file[0]))) {
    for(i = 0; i < gl->n_candidates; i++)
      create_kalman_net(&gl->net[i],
           gl->n_inputs, gl->max_hidden, gl->n_outputs,
           gl->weight_range, get_type(gl->output_type));

    create_kalman_net(&gl->best_weights,
             gl->n_inputs, gl->max_hidden, gl->n_outputs,
             gl->weight_range, get_type(gl->output_type));
  } else {
    for(i = 0; i < gl->n_candidates; i++) {
      flag =
    read_kalman_net(&gl->net[i], &gl->existing_file[0], gl->max_hidden);
      if(flag != 1) {
    fprintf(stderr, "Cannot read in network file. Exiting.\n");
    exit(-1);
      }
    }

    flag = read_kalman_net(&gl->best_weights, &gl->existing_file[0],
              gl->max_hidden);
    if(flag != 1) {
      fprintf(stderr, "Cannot read in network file. Exiting.\n");
      exit(-1);
    }

  }

#ifdef MANY_OUTPUTS
  allocate_dym(&gl->d, MAX_OUTPUTS);
#endif

  return(1);
}
#endif

/*****************************************************************************/
/*
   add_hidden_unit() adds a hidden unit with specified weights and type to
   a cascade network structure originally created with create_kalman_net().

   NOTE: THIS FUNCTION IS ONLY COMPATIBLE WITH NETS CREATED BY
   read_kalman_net()!
 */
/*****************************************************************************/

int add_hidden_unit(CASCADE_NET *net, int type, double *weights)
{
  int i, j, ni, no, nu, nh, nt;

  ni = net->n_inputs;
  nh = net->n_hidden;
  no = net->n_outputs;
  nu = net->n_units;

  nu++;

  for(i = 0; i < no; i++)
    net->n_connections[nu + i] = nu;

  nu--;

  for(i = no; i > 0; i--) {
```

```
    nt = nh + i;
    net->type[nt] = net->type[nt - 1];

    for(j = 0; j < net->n_connections[nu + i] - 1; j++)
      net->weights[nt][j] = net->weights[nt - 1][j];

    net->weights[nt][net->n_connections[nu + i] - 1] =
      weights[nu + i - 1];
  }

  for(j = 0; j < net->n_connections[nu]; j++)
    net->weights[nh][j] = weights[j];

  net->type[nh] = type;

  net->n_hidden++;
  net->n_units++;

  return(1);
}

/****************************************************************************/
/*
   copy_kalman_net() copies one net to another. All memory is assumed to be
   allocated.
 */
/****************************************************************************/

int copy_kalman_net(CASCADE_NET *new, CASCADE_NET *old)
{
  int i, n, m;

  new->n_units = old->n_units;
  new->n_inputs = old->n_inputs;
  new->n_outputs = old->n_outputs;
  new->n_hidden = old->n_hidden;
  new->max_hidden = old->max_hidden;

  n = old->n_inputs + old->n_outputs + old->max_hidden;

  for(i = 0; i < n; i++) {
    new->type[i] = old->type[i];
    new->n_connections[i] = old->n_connections[i];
  }

  n = old->n_outputs + old->max_hidden;
  m = old->n_inputs  + old->max_hidden;

  copy_matrix(old->weights, new->weights, n, m);

  return(1);
}

/****************************************************************************/
/*
   create_kalman_net() creates a cascade network with no hidden units and
   linear output units. It reserves all potentially necessary memory as
   hidden units are added to the cascade network.
 */
/****************************************************************************/

int create_kalman_net(CASCADE_NET *net,
              int n_inputs, int max_hidden, int n_outputs,
              double weight_range, int output_type)
{
  double **weights;
  int i, j, n_max, n_weights, n, m;

  net->n_inputs = ++n_inputs;
  net->n_hidden = 0;
  net->n_outputs = n_outputs;
  net->n_units = net->n_inputs + net->n_hidden;
  net->max_hidden = max_hidden;

  n_max = net->n_inputs + net->n_outputs + net->max_hidden;
  n_weights = net->n_units + net->n_outputs;
```

```
  net->n_connections = (int *) malloc(n_max * sizeof(int));
  is_null((int *) net->n_connections);

  net->type = (int *) malloc(n_max * sizeof(int));
  is_null((int *) net->type);

  for(i = 0; i < n_weights; i++)
    if(i < net->n_units && i < net->n_inputs)
      net->n_connections[i] = 0;
    else if (i < net->n_units && i >= net->n_inputs)
      net->n_connections[i] = i;
    else
      net->n_connections[i] = net->n_units;

  n_max -= net->n_inputs;
  n_weights -= net->n_inputs;

  n = net->n_outputs + net->max_hidden;
  m = net->n_inputs  + net->max_hidden;

  net->weights = allocate_matrix(net->weights, n, m);
  is_null((int *) net->weights);

  weights = allocate_matrix(weights, n, m);
  is_null((int *) weights);

  random_matrix(weights, n, m, -fabs(weight_range), fabs(weight_range));

  for(i = 0; i < n_weights; i++) {
    net->type[i] = output_type;

    for(j = 0; j < net->n_connections[i+net->n_inputs]; j++)
      net->weights[i][j] = weights[i][j];
  }

  free(weights);

  return(1);
}

/*****************************************************************************/
/*
   zero_kalman_net() zeros out a kalman net. Does not reclaim any memory.
 */
/*****************************************************************************/

int zero_kalman_net(CASCADE_NET *net)
{
  net->weights = NULL;
  net->n_connections = NULL;
  net->type = NULL;

  net->n_units = 0;
  net->n_inputs = 0;
  net->n_outputs = 0;
  net->n_hidden = 0;
  net->max_hidden = 0;

  return(1);
}

/*****************************************************************************/
/*
   clear_kalman_net() reclaims memory in a kalman net.
 */
/*****************************************************************************/

int clear_kalman_net(CASCADE_NET *net)
{
  free_matrix(net->weights, net->max_hidden);
  free(net->n_connections);
  free(net->type);

  return(1);
```

```
}

/*****************************************************************************/
/*
   read_kalman_net() reads in networks in a similar manner as read_net().
   However, an additional parameter is passed which reserves memory for
   a maximum number of hidden units. This makes this function compatible
   with add_hidden_unit(). It also makes use of the additional parameter
   in the CASCADE_NET structure max_hidden, which is ignored by all pre-
   kalman functions (such as read_net()).
 */
/*****************************************************************************/

int read_kalman_net(CASCADE_NET *net, char *file_name, int max_hidden)
{
  FILE *fp;
  char type[255];
  int  i, j, format, n_weights, n_max;

  fp = get_fp(file_name, stdin, fp);
  if(fp == NULL)
    return(-4);

  fscanf(fp, "%d %d %d", &net->n_inputs, &net->n_hidden, &net->n_outputs);

  if(max_hidden < 0 || net->n_hidden > max_hidden)
    net->max_hidden = net->n_hidden;
  else
    net->max_hidden = max_hidden;

  net->n_inputs++;
  net->n_units = net->n_inputs + net->n_hidden;

  n_max = net->n_inputs + net->n_outputs + net->max_hidden;
  n_weights = net->n_units + net->n_outputs;

  net->n_connections = (int *) malloc(n_max * sizeof(int));
  is_null((int *) net->n_connections);

  net->type = (int *) malloc(n_max * sizeof(int));
  is_null((int *) net->type);

  for(i = 0; i < n_weights; i++)
    if(i < net->n_units && i < net->n_inputs)
      net->n_connections[i] = 0;
    else if (i < net->n_units && i >= net->n_inputs)
      net->n_connections[i] = i;
    else
      net->n_connections[i] = net->n_units;

  n_max -= net->n_inputs;
  n_weights -= net->n_inputs;

  net->weights = allocate_matrix(net->weights,
                net->n_outputs + net->max_hidden,
                net->n_inputs  + net->max_hidden);
  is_null((int *) net->weights);

  for(i = 0; i < n_weights; i++) {
    if(i < net->n_hidden) {
      fscanf(fp, "%s %d", &type, &format);

      if(format != 0) {
    fclose(fp);
    clear_net(net);
    return(-2);
      }
    } else if (i == net->n_hidden) {
      fscanf(fp, "%s", &type);
    }

    net->type[i] = get_type(type);

    if(net->type[i] < 0) {
      fclose(fp);
      clear_net(net);
```

```
      return(-3);
    }

    for(j = 0; j < net->n_connections[i+net->n_inputs]; j++)
      fscanf(fp, "%lf", &net->weights[i][j]);
  }

  fclose(fp);
  return(1);
}

int matrix_add_noise(double **mat, int n, int cl, int cr, double noise)
{
  int i, j, m;
  double *v;

  m = cr - cl;

  for(i = 0; i < n; i++) {
    v = *(mat++) + cl;
    for(j = 0; j < m; j++)
      *(v++) += random_real2(-fabs(noise), fabs(noise));
  }

  return(1);
}

int randomize_index(int *index, int n)
{
  int i, ind1, ind2, tmp;

  fprintf(stderr, "Generating randomizing index.\n");

  for(i = 0; i < n; i++)
    index[i] = i;

  for(i = 1; i < n; i++) {
    ind1 = n - i;
    ind2 = random_integer(ind1);

    tmp = index[ind1];
    index[ind1] = index[ind2];
    index[ind2] = tmp;
  }

  return(1);
}

int process_command_line(GlobalOptions *gl, int argc, char **argv)
{
  arg_parse
    (argc, argv,
     "", "Usage: klm <net_file> [options]",
     "-nolinear [%d]", &gl->no_linear, "no direct I/O connections [%d]",
     gl->no_linear,
     "-ninputs [%d]", &gl->n_inputs, "number of inputs [%d]", gl->n_inputs,
     "-noutputs [%d]", &gl->n_outputs, "number of inputs [%d]", gl->n_outputs,
     "-ntrain [%d]", &gl->n_train_patterns,
     "set number of training patterns [%d]", gl->n_train_patterns,
     "-ncross [%d]", &gl->n_cross_patterns,
     "set number of training patterns [%d]", gl->n_cross_patterns,
     "-ntest [%d]", &gl->n_test_patterns,
     "set number of training patterns [%d]", gl->n_test_patterns,
     "-seed [%d]", &gl->random_seed,
     "set random seed [%d]", gl->random_seed,
     "-inputnoise [%F]", &gl->input_noise,
     "set input noise level [%lf]", gl->input_noise,
     "-outputnoise [%F]", &gl->output_noise,
     "set input noise level [%lf]", gl->output_noise,
     "-weightrange [%F]", &gl->weight_range,
     "set range for random weights [%lf, %lf]",
     -fabs(gl->weight_range),
     fabs(gl->weight_range),
     "-savenoisefiles [%d]", &gl->save_noise_files,
     "save files with added input/output noise [%d]",
     gl->save_noise_files,
```

```
      "-maxiterations [%d]", &gl->max_iterate,
      "maximum iterations/hidden unit [%d]", gl->max_iterate,
      "-maxhidden [%d]", &gl->max_hidden,
      "maximum # of hidden units [%d]", gl->max_hidden,

      "-trainfile [%s]", &gl->train_file[0],
      "set training data file name [%s]", gl->train_file,
      "-crossfile [%s]", &gl->cross_file[0],
      "set cross validation data file name [%s]", gl->cross_file,
      "-testfile [%s]", &gl->test_file[0],
      "set test file name [%s]", gl->test_file,

      "-cascadefile [%s]", &gl->cascade_file[0],
      "set output cascade file name [%s]", gl->cascade_file,
      "-existingfile [%s]", &gl->existing_file[0],
      "set existing network file to read in",
      "-dumpweights [%d]", &gl->dump_weights,
      "save weight files to cascade file arg [%d]", gl->dump_weights,
      "-trials [%d]", &gl->trials, "set # of trials [%d]", gl->trials,
      "-changeperiod [%d]", &gl->change_period,
      "set the change period to measure error over [%d]", gl->change_period,
      "-changethreshold [%F]", &gl->change_threshold,
      "set the change threshold value [%lf]", gl->change_threshold,
      "-report [%d]", &gl->report, "set reporting level [%d]", gl->report,
      "-outputtype [%s]", &gl->output_type[0],
      "set output type [%s]", gl->output_type,
      "-unittype [%s]", &gl->unit_type[0],
      "set unit type [%s]", gl->unit_type,
      "-histogram [%d %F %F]", &gl->display_histogram,
      &gl->histogram_min, &gl->histogram_max,
      "set histogram params [%d %lf %lf]",
      gl->display_histogram, gl->histogram_min, gl->histogram_max,
      "-sse [%d]", &gl->display_sse, "display sse flag [%d]",
      gl->display_sse,
      "-peroutput [%d]", &gl->display_per_output,
      "display per-output flag [%d]", gl->display_per_output,
      "-maxerr [%d]", &gl->display_max_err, "display max error flag [%d]",
      gl->display_max_err,
      "-stddev [%d]", &gl->display_std_dev, "display std. deviation flag [%d]",
      gl->display_std_dev,
      "-biterror [%d]", &gl->display_bit_err,
      "report bit errors as well as rms errors [%d]",
      gl->display_bit_err,
      "-patternerror", ARG_FLAG(&gl->pattern_wise_error),
      "display pattern-by-pattern error of test file (with existingfile)",
      NULL);

  return(1);
}

#ifdef MANY_OUTPUTS
int inverse_new(double **inv2, double **mat, int n, dym *d)
{
  int i, j;
  double *vd;
  float *vf;

  d->rows = n; d->cols = n;
  for(i = 0; i < n; i++) {
    vf = d->tdarr[i];
    vd = *(mat++);
    for(j = 0; j < n; j++)
      *(vf++) = (float) *(vd++);
  }

  invert_dym(d, d);

  for(i = 0; i < n; i++) {
    vf = d->tdarr[i];
    vd = *(inv2++);
    for(j = 0; j < n; j++)
      *(vd++) = (double) *(vf++);
  }

  return(1);
}
```

```
#define DYM_CODE 4508

int allocate_dym(dym *d, int n)
{
  d->dym_code = DYM_CODE;
  d->rows = n;
  d->cols = n;
  d->rows_allocated = n;
  d->tdarr = am_malloc_2d_floats(n, n);

  return(1);
}
#endif
```

nn_util.c

```
/*****************************************************************************/
/*                                                                           */
/* Title: Cascade network utilities                                          */
/* File: "nn_util.c"                                                         */
/*                                                                           */
/* Author: Michael C. Nechyba                                                */
/* Last modified: 11/3/96                                                    */
/*                                                                           */
/* (Now can handle both Cascade 2 nets and BP nets generated with qp and     */
/*  MATHEMATICA_DUMP set to 1.)                                              */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "gen_math.h"
#include "matrix.h"
#include "file_io.h"
#include "nn_util.h"

int zero_net(CASCADE_NET *net)
{
  net->weights = NULL;
  net->n_connections = NULL;
  net->type = NULL;

  net->n_units = 0;
  net->n_inputs = 0;
  net->n_outputs = 0;
  net->n_hidden = 0;

  return(1);
}

int clear_net(CASCADE_NET *net)
{
  int i, n_weights;

  n_weights = net->n_units + net->n_outputs - net->n_inputs;

  for(i = 0; i < n_weights; i++)
    free(net->weights[i]);

  free(net->weights);
  free(net->n_connections);
  free(net->type);

  return(1);
}

int write_net(CASCADE_NET *net, char *file_name)
```

```
{
  int i, j;
  FILE *fp;
  char type[STR_REG];

  fp = get_fp(file_name, stdout, fp);

  fprintf(fp, "%d %d %d\n", net->n_inputs - 1, net->n_hidden, net->n_outputs);

  for(i = 0; i < net->n_hidden; i++) {
    get_type_string(net->type[i], &type[0]);
    fprintf(fp, "%s 0\n", type);
    for(j = 0; j < net->n_connections[i + net->n_inputs] - 1; j++)
      fprintf(fp, "%17.15g ", net->weights[i][j]);
    fprintf(fp, "%17.15g\n",
        net->weights[i][net->n_connections[i + net->n_inputs] - 1]);
  }

  get_type_string(net->type[net->n_hidden], &type[0]);
  fprintf(fp, "%s\n", type);

  for(i = net->n_hidden; i < (net->n_hidden + net->n_outputs); i++) {
    for(j = 0; j < net->n_connections[i + net->n_inputs] - 1; j++)
      fprintf(fp, "%17.15g ", net->weights[i][j]);
    fprintf(fp, "%17.15g\n",
        net->weights[i][net->n_connections[i + net->n_inputs] - 1]);
  }

  fclose(fp);

  return(1);
}

int read_net(CASCADE_NET *net, char *file_name)
{
  FILE *fp;
  char type[255];
  int  i, j, format, n_weights;

  fp = get_fp(file_name, stdin, fp);
  if(fp == NULL)
    return(-4);

  fscanf(fp, "%d %d %d", &net->n_inputs, &net->n_hidden, &net->n_outputs);

  net->n_inputs++;
  net->n_units = net->n_inputs + net->n_hidden;

  n_weights = net->n_units + net->n_outputs;

  net->n_connections = (int *) malloc(n_weights * sizeof(int));
  is_null((int *) net->n_connections);

  net->type = (int *) malloc(n_weights *sizeof(int));
  is_null((int *) net->type);

  for(i = 0; i < n_weights; i++)
    if(i < net->n_units && i < net->n_inputs)
      net->n_connections[i] = 0;
    else if (i < net->n_units && i >= net->n_inputs)
      net->n_connections[i] = i;
    else
      net->n_connections[i] = net->n_units;

  n_weights -= net->n_inputs;

  net->weights = (double **) malloc(n_weights*sizeof(double *));
  is_null((int *) net->weights);

  for(i = 0; i < n_weights; i++) {
    net->weights[i] = (double *)
      malloc(net->n_connections[i+net->n_inputs]*sizeof(double));
    is_null((int *) net->weights[i]);

    if(i < net->n_hidden) {
      fscanf(fp, "%s %d", &type, &format);
```

```
      if(format != 0) {
    fclose(fp);
    clear_net(net);
    return(-2);
      }
    } else if (i == net->n_hidden) {
      fscanf(fp, "%s", &type);
    }

    net->type[i] = get_type(type);

    if(net->type[i] < 0) {
      fclose(fp);
      clear_net(net);
      return(-3);
    }

    for(j = 0; j < net->n_connections[i+net->n_inputs]; j++)
      fscanf(fp, "%lf", &net->weights[i][j]);
  }

  fclose(fp);
  return(1);
}

int get_type(char *type)
{
  if(!strcmp(type, "SIGMOID2"))
    return(SIGMOID2);
  else if(!strcmp(type, "SIGMOID"))
    return(SIGMOID);
  else if(!strcmp(type, "CARUNIT"))
    return(CARUNIT);
  else if(!strcmp(type, "GAUSSIAN"))
    return(GAUSSIAN);
  else if(!strcmp(type, "BESSEL0"))
    return(BESSEL0);
  else if(!strcmp(type, "BESSEL1"))
    return(BESSEL1);
  else if(!strcmp(type, "SIN"))
    return(SINE);
  else if(!strcmp(type, "COS"))
    return(COSINE);
  else if(!strcmp(type, "SIN2"))
    return(SINE2);
  else if(!strcmp(type, "COS2"))
    return(COSINE2);
  else if(!strcmp(type, "GTSIN"))
    return(GTSINE);
  else if(!strcmp(type, "LINEAR"))
    return(LINEAR);
  else if(!strcmp(type, "MIXED"))
    return(MIXED);
  else
    return(FORMAT_ERROR);
}

int get_type_string(int type, char *str)
{
  switch(type) {
  case SIGMOID2:  strcpy(str, "SIGMOID2"); break;
  case SIGMOID:   strcpy(str, "SIGMOID");  break;
  case CARUNIT:   strcpy(str, "CARUNIT");  break;
  case GAUSSIAN:  strcpy(str, "GAUSSIAN"); break;
  case BESSEL0:   strcpy(str, "BESSEL0");  break;
  case BESSEL1:   strcpy(str, "BESSEL1");  break;
  case SINE:      strcpy(str, "SIN");      break;
  case COSINE:    strcpy(str, "COS");      break;
  case SINE2:     strcpy(str, "SIN2");     break;
  case COSINE2:   strcpy(str, "COS2");     break;
  case GTSINE:    strcpy(str, "GTSIN");    break;
  default:        strcpy(str, "LINEAR");   break;
  }

  return(1);
```

```
}

CASCADE_NET *chop_weights(CASCADE_NET *net, double offset)
{
  int i,j, n_weights;

  n_weights = net->n_units + net->n_outputs - net->n_inputs;

  for(i = 0; i < n_weights; i++)
    for(j = 0; j < net->n_connections[i+net->n_inputs]; j++)
      if(fabs(net->weights[i][j]) < offset)
    net->weights[i][j] = 0.;

  return(net);
}

CASCADE_NET *add_noise(CASCADE_NET *net, double max_noise)
{
  int i,j, n_weights;

  n_weights = net->n_units + net->n_outputs - net->n_inputs;

  for(i = 0; i < n_weights; i++)
    for(j = 0; j < net->n_connections[i+net->n_inputs]; j++)
      net->weights[i][j] += random_noise(max_noise);

  return(net);
}

double random_noise(double range)
{
  return(random_real2(-range, range));
}

double *net_output_multi(CASCADE_NET *class, double *class_inputs,
            CASCADE_NET *net, double *inputs, double *outputs)
{
  int cl;

  cl = net_output_class(class, class_inputs == NULL ? inputs : class_inputs);

  net_output(&net[cl], inputs, outputs);

  return(outputs);
}

#define NN_CLASS_MAX 10

int net_output_class(CASCADE_NET *net, double *inputs)
{
  double outputs[NN_CLASS_MAX];

  net_output(net, inputs, &outputs[0]);

  return(ind_max_vec(&outputs[0], net->n_outputs));
}

double *net_output(CASCADE_NET *net, double *inputs, double *outputs)
{
  int i,j;
  double *output_weights, *v, *values, *out;

  values = &NOValues[0]; v = values;

  for(i = 0; i < net->n_units; i++)
    if(i < net->n_inputs-1)
      *(v++) = *(inputs++);
    else if (i == net->n_inputs - 1)
      *(v++) = 1.;
    else
      *(v++) = compute_unit_value(net, values, i,
                  net->type[i - net->n_inputs]);

  out = outputs;
  for(i = 0; i < net->n_outputs; i++) {
    *out = 0.;
```

```
    output_weights = net->weights[i + net->n_units - net->n_inputs];

    v = values;
    for(j = 0; j < net->n_units; j++)
      *out += (*(v++) * (*(output_weights++)));

    *(out++) = Function[net->type[i + net->n_hidden]].act_fnc(outputs[i]);
  }

  return(outputs);
}

double compute_unit_value(CASCADE_NET *net, double *values,
             int unit, int unit_type)
{
  int i;
  double *w, sum = 0.0;

  w = net->weights[unit - net->n_inputs];

  for(i = 0; i < unit; i++)
    sum += (*(values++) * (*(w++)));

  return(Function[unit_type].act_fnc(sum));
}

#define ACT_PRIME_OFFSET 0.1

double dsigmoid(double x)
{
  if (x < -(SIGMOID_ROUND) || x > SIGMOID_ROUND)
    return(0.);
  else {
    double v;

    v = exp(-x);

#ifndef ACT_PRIME
    return(v/((1. + v)*(1. + v)));
#else
    return(ACT_PRIME_OFFSET + v/((1. + v)*(1. + v)));
#endif
  }
}

double sigmoid(double x)
{
  if (x < -SIGMOID_ROUND)
    return(-0.5);
  else if (x > SIGMOID_ROUND)
    return(0.5);
  else
    return(1.0/(1.0 + exp(-x)) - 0.5);
}

double dsigmoid2(double x)
{
  if (x < -SIGMOID_ROUND || x > SIGMOID_ROUND)
    return(0.);
  else {
    double v;

    v = exp(-x);

#ifndef ACT_PRIME
    return(2.*v/((1. + v)*(1. + v)));
#else
    return(ACT_PRIME_OFFSET + 2.*v/((1. + v)*(1. + v)));
#endif
  }
}

double sigmoid2(double x)
{
  if (x < -SIGMOID_ROUND)
    return(-1.);
```

```
  else if (x > SIGMOID_ROUND)
    return(1.);
  else
    return(2.0/(1.0 + exp(-x)) - 1.);
}

double dcarunit(double x)
{
  if (x < -SIGMOID_ROUND || x > SIGMOID_ROUND)
    return(0.);
  else {
    double v;

    v = 0.5*exp(-x);

    return(1.5*v/((1. + v)*(1. + v)));
  }
}

double carunit(double x)
{
  if (x < -SIGMOID_ROUND)
    return(-1.);
  else if (x > SIGMOID_ROUND)
    return(0.5);
  else
    return(1.5/(1. + 0.5*exp(-x)) - 1.);
}

double dgaussian(double x)
{
  double tmp;

  tmp = -0.5*x*x;

  if(tmp < -75.0)
    return(0.0);
  else
    return(-x*exp(tmp));
}

double gaussian(double x)
{
  double tmp;

  tmp = -0.5*x*x;

  if(tmp < -75.0)
    return(0.0);
  else
    return(exp(tmp));
}

double dsin(double x)
{
  return(cos(x));
}

double dcos(double x)
{
  return(-sin(x));
}

double dsin2(double x)
{
  return(2.*cos(fmod(2.0 * x, 2.0*M_PI)));
}

double sin2(double x)
{
  return(sin(fmod(2.0 * x, 2.0*M_PI)));
}

double dcos2(double x)
{
  return(-2.*sin(fmod(2.0 * x, 2.0*M_PI)));
```

```
}

double cos2(double x)
{
  return(cos(fmod(2.0 * x, 2.0*M_PI)));
}

double dgtsin(double x)
{
  double v;

  v = gtsin(x);

  if (x > M_PI_2)
    return(-(x - M_PI_2) * v);
  else if (x < -M_PI_2)
    return(-(x + M_PI_2) * v);
  else
    return(cos(x));

}
double gtsin(double x)
{
  double tmp;

  if (x > M_PI_2)  {
    tmp = x - M_PI_2;
    tmp = -0.5 * tmp * tmp;
    if (tmp < -75.0)
      return(0.0);
    return(exp(tmp));
  } else if (x < -M_PI_2) {
    tmp = x + M_PI_2;
    tmp = -0.5 * tmp * tmp;
    if (tmp < -75.0)
      return(0.0);
    return(-exp(tmp));
  } else
    return(sin(x));
}

double dlinear(double x)
{
  return(1.);
}

double linear(double x)
{
  return(x);
}

double dbessel0(double x)
{
#ifdef MacOS
  fprintf(stderr, "Bessel functions are not implemented in MacOS yet. Exiting.\n");
  exit(-1);
#else
  return(-j1(x));
#endif
}

double bessel0(double x)
{
#ifdef MacOS
  fprintf(stderr, "Bessel functions are not implemented in MacOS yet. Exiting.\n");
  exit(-1);
#else
  return(j0(x));
#endif
}

double dbessel1(double x)
{
#ifdef MacOS
  fprintf(stderr, "Bessel functions are not implemented in MacOS yet. Exiting.\n");
  exit(-1);
```

```
#else
  if(!is_zero(x))
    return(j0(x) - j1(x)/x);
  else
    return(0.5);
#endif
}

double bessel1(double x)
{
#ifdef MacOS
  fprintf(stderr, "Bessel functions are not implemented in MacOS yet. Exiting.\n");
  exit(-1);
#else
  return(j1(x));
#endif
}

/***************************** BP/QP utils ***********************************/

void build_data_structures(BP_NET *net)
{
  int nu;

  net->n_units = 1 + net->n_inputs + net->n_hidden + net->n_outputs;
  nu = net->n_units;

  net->first_hidden = 1 + net->n_inputs;
  net->first_output = 1 + net->n_inputs + net->n_hidden;

  net->connections = int_allocate_matrix(net->connections, nu, nu);

  net->weights = allocate_matrix(net->weights, nu, nu);
  net->outputs = allocate_vector(net->outputs, nu);
  net->n_connections = int_allocate_vector(net->n_connections, nu);

  net->outputs = zero_vector(net->outputs, nu);
  net->outputs[0] = 1.0;

  net->n_connections = int_zero_vector(net->n_connections, nu);
}

void connect_layers(BP_NET *net, int start1, int end1, int start2, int end2)
{
  int i, j, k;

  for(i = start2; i <= end2; i++) {
    if(net->n_connections[i] == 0) {
      net->n_connections[i]  += 1;
      net->connections[i][0] = 0;
      k = 1;
    } else
      k = net->n_connections[i];

    for(j = start1; j <= end1; j++) {
      net->n_connections[i]  += 1;
      net->connections[i][k] = j;
      k++;
    }
  }
}

int read_bp_net(BP_NET *net, char *file_name)
{
  FILE *fp;
  int i, j, k, l, connect;
  double in_weight;
  int tmp[4];

  fp = get_fp(file_name, stdin, fp);
  if(fp == NULL)
    return(-4);

  fscanf(fp, "%d %d %d", &net->n_inputs, &net->n_hidden, &net->n_outputs);
  build_data_structures(net);
```

```
  fscanf(fp, "%d", &connect);
  for(i = 0; i < connect; i++) {
    fscanf(fp, "%d %d %d %d", &tmp[0], &tmp[1], &tmp[2], &tmp[3]);
    connect_layers(net, tmp[0], tmp[1], tmp[2], tmp[3]);
  }

  for(i = 0; i < net->n_units; i++)
    for(j = 0; j < net->n_units; j++)
      net->weights[i][j] = 0.0;

  for(j = net->first_hidden; j < net->n_units; j++)
    for(i = 0; i < net->n_connections[j]; i++) {
      fscanf(fp, "%d %d %lf", &k, &l, &in_weight);
      if((j == k) && (l == net->connections[j][i])) {
    net->weights[j][i] = in_weight;
      }
    }

  fclose (fp);
  return(1);
}

double *bp_output(BP_NET *net, double *inputs, double *outputs)
{
  int i, j, *connect;
  double sum, *weight;

  for(i = 0; i < net->n_inputs; i++)
    net->outputs[i+1] = inputs[i];

  for(j = net->first_hidden; j < net->n_units; j++) {
    connect = net->connections[j];
    weight = net->weights[j];
    sum = 0.0;
    i = net->n_connections[j];
    while (i > 0) {
      i--;
      sum += (net->outputs[connect[i]] * weight[i]);
    }
    net->outputs[j] = sigmoid(sum);
  }

  for(j = net->first_output; j < net->n_units; j++)
    outputs[j - net->first_output] = net->outputs[j];

  return(outputs);
}

int clear_bp_net(BP_NET *net)
{
  int nu;

  nu = net->n_units;

  net->connections = int_free_matrix(net->connections, nu);

  net->weights = free_matrix(net->weights, nu);
  net->outputs = free_vector(net->outputs);

  net->n_connections = int_free_vector(net->n_connections);

  return(1);
}

/*****************************************************************************/

int initialize_functions(int use_lookup)
{
  if(use_lookup > 0) {
    initialize_lookup();

    Function[SIGMOID].act_fnc = sigmoid_t;
    Function[SIGMOID].dact_fnc = dsigmoid_t;

    Function[SIGMOID2].act_fnc = sigmoid2_t;
    Function[SIGMOID2].dact_fnc = dsigmoid2_t;
```

```
    Function[CARUNIT].act_fnc = carunit_t;
    Function[CARUNIT].dact_fnc = dcarunit_t;

    Function[SINE].act_fnc = sin_t;
    Function[SINE].dact_fnc = dsin_t;

    Function[SINE2].act_fnc = sin2_t;
    Function[SINE2].dact_fnc = dsin2_t;

    Function[GAUSSIAN].act_fnc = gaussian_t;
    Function[GAUSSIAN].dact_fnc = dgaussian_t;

    Function[BESSEL0].act_fnc = bessel0_t;
    Function[BESSEL0].dact_fnc = dbessel0_t;

    Function[BESSEL1].act_fnc = bessel1_t;
    Function[BESSEL1].dact_fnc = dbessel1_t;

    Function[COSINE].act_fnc = cos_t;
    Function[COSINE].dact_fnc = dcos_t;

    Function[COSINE2].act_fnc = cos2_t;
    Function[COSINE2].dact_fnc = dcos2_t;

    Function[GTSINE].act_fnc = gtsin_t;
    Function[GTSINE].dact_fnc = dgtsin_t;

    Function[LINEAR].act_fnc = linear;
    Function[LINEAR].dact_fnc = dlinear;
  } else {
    Function[SIGMOID].act_fnc = sigmoid;
    Function[SIGMOID].dact_fnc = dsigmoid;

    Function[SIGMOID2].act_fnc = sigmoid2;
    Function[SIGMOID2].dact_fnc = dsigmoid2;

    Function[CARUNIT].act_fnc = carunit;
    Function[CARUNIT].dact_fnc = dcarunit;

    Function[SINE].act_fnc = sin;
    Function[SINE].dact_fnc = dsin;

    Function[SINE2].act_fnc = sin2;
    Function[SINE2].dact_fnc = dsin2;

    Function[GAUSSIAN].act_fnc = gaussian;
    Function[GAUSSIAN].dact_fnc = dgaussian;

    Function[BESSEL0].act_fnc = bessel0;
    Function[BESSEL0].dact_fnc = dbessel0;

    Function[BESSEL1].act_fnc = bessel1;
    Function[BESSEL1].dact_fnc = dbessel1;

    Function[COSINE].act_fnc = cos;
    Function[COSINE].dact_fnc = dcos;

    Function[COSINE2].act_fnc = cos2;
    Function[COSINE2].dact_fnc = dcos2;

    Function[GTSINE].act_fnc = gtsin;
    Function[GTSINE].dact_fnc = dgtsin;

    Function[LINEAR].act_fnc = linear;
    Function[LINEAR].dact_fnc = dlinear;
  }

  return(1);
}

int initialize_lookup(void)
{
  generate_lookup(&Sin_lu, sin, -TWO_PI, TWO_PI);

  generate_lookup(&Sigmoid_lu, sigmoid, -SIGMOID_ROUND, SIGMOID_ROUND);
```

```
  generate_lookup(&Dsigmoid_lu, dsigmoid, -SIGMOID_ROUND, SIGMOID_ROUND);

  generate_lookup(&Sigmoid2_lu, sigmoid2,   -SIGMOID_ROUND, SIGMOID_ROUND);
  generate_lookup(&Dsigmoid2_lu, dsigmoid2, -SIGMOID_ROUND, SIGMOID_ROUND);

  generate_lookup(&Carunit_lu, carunit,   -SIGMOID_ROUND, SIGMOID_ROUND);
  generate_lookup(&Dcarunit_lu, dcarunit, -SIGMOID_ROUND, SIGMOID_ROUND);

  generate_lookup(&Gaussian_lu, gaussian,   -SIGMOID_ROUND, SIGMOID_ROUND);
  generate_lookup(&Dgaussian_lu, dgaussian, -SIGMOID_ROUND, SIGMOID_ROUND);

  generate_lookup(&Gtsin_lu, gtsin,   -SIGMOID_ROUND, SIGMOID_ROUND);
  generate_lookup(&Dgtsin_lu, dgtsin, -SIGMOID_ROUND, SIGMOID_ROUND);

  generate_lookup(&Bessel0_lu, bessel0,   -BESSEL_ROUND, BESSEL_ROUND);
  generate_lookup(&Dbessel0_lu, dbessel0, -BESSEL_ROUND, BESSEL_ROUND);

  generate_lookup(&Bessel1_lu, bessel1,   -BESSEL_ROUND, BESSEL_ROUND);
  generate_lookup(&Dbessel1_lu, dbessel1, -BESSEL_ROUND, BESSEL_ROUND);

  return(1);
}

int generate_lookup(Lookup *lu, double (*f)(), double min, double max)
{
  int i;
  double x, fac;

  fac = (max - min)/((double) LOOKUP - 1.);

  for(i = 0; i < LOOKUP; i++) {
    x = min + ((double) i)*fac;

    lu->ft[i] = f(x);
  }

  lu->factor = 1./fac;

  return(1);
}

double sin_t(double x)
{
  int i;
  double y, ind, alpha;

  y = x*INV_TWO_PI;
  y = FRAC(y)*TWO_PI;

  ind = (y + TWO_PI)*Sin_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Sin_lu.ft[i] * (1. - alpha) + Sin_lu.ft[i+1] * alpha);
}

double dsin_t(double x)
{
  return(cos_t(x));
}

double cos_t(double x)
{
  return(sin_t(x + PI_OVER_TWO));
}

double dcos_t(double x)
{
  return(sin_t(-(x)));
}

double sin2_t(double x)
{
  return(sin_t(2.*(x)));
}
```

```
double dsin2_t(double x)
{
  return(2. * cos2_t(x));
}

double cos2_t(double x)
{
  return(cos_t(2.*(x)));
}

double dcos2_t(double x)
{
  return(-2. * sin2_t(x));
}

double sigmoid_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -SIGMOID_ROUND)
    return(-0.5);
  else if (x > SIGMOID_ROUND)
    return(0.5);

  ind = (x + SIGMOID_ROUND)*Sigmoid_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Sigmoid_lu.ft[i] * (1. - alpha) + Sigmoid_lu.ft[i+1] * alpha);
}

double dsigmoid_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -SIGMOID_ROUND || x > SIGMOID_ROUND)
    return(0.0);

  ind = (x + SIGMOID_ROUND)*Dsigmoid_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Dsigmoid_lu.ft[i] * (1. - alpha) + Dsigmoid_lu.ft[i+1] * alpha);
}

double sigmoid2_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -SIGMOID_ROUND)
    return(-1.);
  else if (x > SIGMOID_ROUND)
    return(1.);

  ind = (x + SIGMOID_ROUND)*Sigmoid2_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Sigmoid2_lu.ft[i] * (1. - alpha) + Sigmoid2_lu.ft[i+1] * alpha);
}

double dsigmoid2_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -SIGMOID_ROUND || x > SIGMOID_ROUND)
    return(0.0);
```

```
  ind = (x + SIGMOID_ROUND)*Dsigmoid2_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Dsigmoid2_lu.ft[i] * (1. - alpha) + Dsigmoid2_lu.ft[i+1] * alpha);
}

double carunit_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -SIGMOID_ROUND)
    return(-1.);
  else if (x > SIGMOID_ROUND)
    return(0.5);

  ind = (x + SIGMOID_ROUND)*Carunit_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Carunit_lu.ft[i] * (1. - alpha) + Carunit_lu.ft[i+1] * alpha);
}

double dcarunit_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -SIGMOID_ROUND || x > SIGMOID_ROUND)
    return(0.0);

  ind = (x + SIGMOID_ROUND)*Dcarunit_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Dcarunit_lu.ft[i] * (1. - alpha) + Dcarunit_lu.ft[i+1] * alpha);
}

double gaussian_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -SIGMOID_ROUND || x > SIGMOID_ROUND)
    return(0.);

  ind = (x + SIGMOID_ROUND)*Gaussian_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Gaussian_lu.ft[i] * (1. - alpha) + Gaussian_lu.ft[i+1] * alpha);
}

double dgaussian_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -SIGMOID_ROUND || x > SIGMOID_ROUND)
    return(0.0);

  ind = (x + SIGMOID_ROUND)*Dgaussian_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Dgaussian_lu.ft[i] * (1. - alpha) + Dgaussian_lu.ft[i+1] * alpha);
}

double gtsin_t(double x)
{
```

```
  int i;
  double ind, alpha;

  if(x < -SIGMOID_ROUND || x > SIGMOID_ROUND)
    return(0.);

  ind = (x + SIGMOID_ROUND)*Gtsin_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Gtsin_lu.ft[i] * (1. - alpha) + Gtsin_lu.ft[i+1] * alpha);
}

double dgtsin_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -SIGMOID_ROUND || x > SIGMOID_ROUND)
    return(0.0);

  ind = (x + SIGMOID_ROUND)*Dgtsin_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Dgtsin_lu.ft[i] * (1. - alpha) + Dgtsin_lu.ft[i+1] * alpha);
}

double bessel0_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -BESSEL_ROUND || x > BESSEL_ROUND)
    return(0.);

  ind = (x + BESSEL_ROUND)*Bessel0_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Bessel0_lu.ft[i] * (1. - alpha) + Bessel0_lu.ft[i+1] * alpha);
}

double dbessel0_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -BESSEL_ROUND || x > BESSEL_ROUND)
    return(0.0);

  ind = (x + BESSEL_ROUND)*Dbessel0_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Dbessel0_lu.ft[i] * (1. - alpha) + Dbessel0_lu.ft[i+1] * alpha);
}

double bessel1_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -BESSEL_ROUND || x > BESSEL_ROUND)
    return(0.0);

  ind = (x + BESSEL_ROUND)*Bessel1_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Bessel1_lu.ft[i] * (1. - alpha) + Bessel1_lu.ft[i+1] * alpha);
```

```
}

double dbessel1_t(double x)
{
  int i;
  double ind, alpha;

  if(x < -BESSEL_ROUND || x > BESSEL_ROUND)
    return(0.0);

  ind = (x + BESSEL_ROUND)*Dbessel1_lu.factor;
  i = (int) ind;

  alpha = ind - (double) i;

  return(Dbessel1_lu.ft[i] * (1. - alpha) + Dbessel1_lu.ft[i+1] * alpha);
}
```

REFERENCES

[1] D.A. Pomerleau, "Neural Network Perception for Mobile Robot Guidance," Ph.D. thesis, Carnegie Mellon University, 1992.

[2] D.A. Pomerleau, "Reliability Estimation for Neural Network Based Autonomous Driving," *Robotics and Autonomous Systems*, vol. 12, no. 3-4, pp. 113-9, 1994.

[3] M.C. Nechyba and Y. Xu, "Human Control Strategy: Abstraction, Verification and Replication," *IEEE Control Systems Magazine*, vol. 17, no. 5, pp. 48-61, 1997.

[4] M.C. Nechyba, *Learning and Validation of Human Control Strategies*, Ph.D. thesis, Carnegie Mellon University, 1998. (http://www.ri.cmu.edu/pubs/pub_478.html)

[5] M.C. Nechyba and Y. Xu, "On Discontinuous Human Control Strategies," *Proc. IEEE Int. Conference on Robotics and Automation*, vol. 3, pp. 2237-43, 1998. (http://www.ri.cmu.edu/pubs/pub_1077.html)

[6] M. Orme, "How to Choose the Right Motor for Your Airplane, Parts 1 and 2," Aveox Inc., Westlake Village, CA. Retrieved April, 2000. (http://www.aveox.com/selection.html)

[7] "MotoCalc Manual," Capable Computing, Inc., Moorefield, Ont., 1998. (http://www.motocalc.com/motocalc.htm)

[8] "Low Cost 2 g/10 g Dual Axis iMEMS Accelerometers with Digital Output," Analog Devices Corporation, Norwood, MA, 1999. (http://www.analog.com/pdf/ADXL202_10_b.pdf)

[9] S.B. Stancliff, J.L. Laine, and M.C. Nechyba, "Learning to Fly: Design and Construction of an Autonomous Airplane," *1999 Florida Conference on Recent Advances in Robotics*. (http://cimar.me.ufl.edu/FLA99/session4/flying.pdf)

[10] M.C. Nechyba and Y. Xu, "Cascade Neural Networks with Node-Decoupled Extended Kalman Filtering," *Proc. IEEE Int. Symp. on Computational Intelligence in Robotics and Automation*, vol. 1, pp. 214-9, 1997. (http://www.ri.cmu.edu/pubs/pub_946.html)

[11] M.C. Nechyba and Y. Xu, "Towards Human Control Strategy Learning: Neural Network Approach with Variable Activation Functions," Technical Report CMU-RI-TR-95-09, Carnegie Mellon University, 1995. (http://www.ri.cmu.edu/pubs/pub_368.html)

[12] S.E. Fahlman, L.D. Baker and J.A. Boyan, "The Cascade 2 Learning Architecture," Technical Report CMU-CS-TR-96-184, Carnegie Mellon University, 1996.

BIOGRAPHICAL SKETCH

Stephen B. Stancliff was born in Norfolk, V.A., and grew up in Charlotte, N.C.. He received a Bachelor of Science in Aeronautical and Astronautical Engineering and a Bachelor of Arts in December 1991 from Purdue University. His interest in robotics was ignited while teaching robotics to middle school students at The Charlotte Latin School. Upon completion of the Master of Engineering, he intends to pursue a doctoral degree in robotics and a master's degree in philosophy at Carnegie Mellon University.