Software Interface to the Caltech Ducted Fan

Richard M. Murray
Control and Dynamical Systems
California Institute of Technology

Class notes for CDS 111
8 April 1996

1 Introduction

The Caltech Ducted Fan is interfaced to an 80386 computer running an MSDOS-based real-time kernel called Sparrow [2] and using the Falcon Control Library [1]. Custom hardware is used to read in joint angles via encoders and generate PWM signals for the R/C servos and a speed controller at a software selectable rate. Currently, the joint angles are read in at 200Hz and the PWM signals are output at 50Hz, which is the standard update rate for R/C servos. The R/C servos control the flap angles and speed controller controls the motor speed, and consequently the fan force. The finite angular resolution of the encoders and unavoidable delay in observing states of the system make it desirable to run the controller as fast as possible.

A schematic of the ducted fan interface hardware and software is shown in Figure 1. This note describes the functionality of the software interface and gives details about how to load controllers, follow trajectories, and capture data.

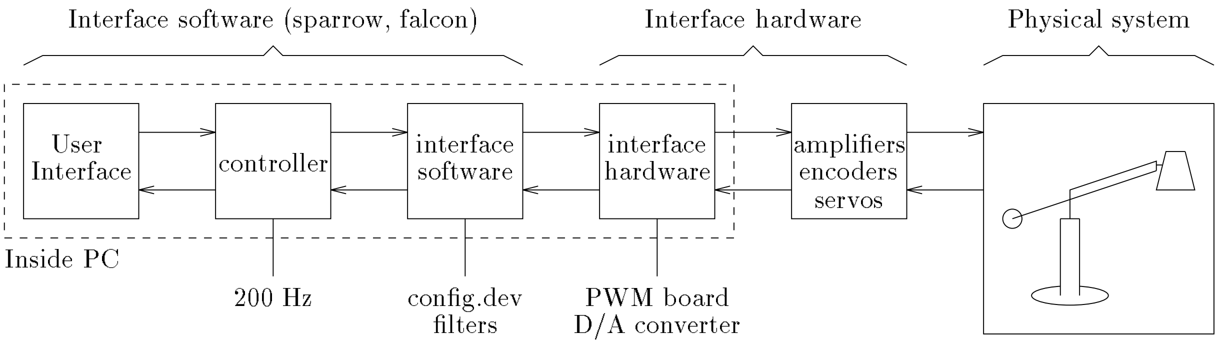

Figure 1: Ducted fan interface hardware and software

2 Overview of Sparrow

Sparrow is a collection of programs and a library of C functions intended to aid in the implementation of real-time controllers on PC-based data acquisition and control systems. It contains functions for executing control algorithms at a fixed rate, communicating with hardware interface cards, and displaying data in real-time. The core Sparrow library consists of three main modules:

Servo scheduling
: The servo module allows the user to have a C function called at a specified rate. For control applications, this function is typically responsible for reading sensor data from hardware, performing control law computations, and writing actuator data to hardware.

Channel interface
: The channel interface provides a device-independent interface to hardware. Device drivers have been written for a few commercial I/O boards and a wide variety of custom devices built in the lab. Once a device driver is written, reading and writing of data to hardware, data capture, and real-time data plotting is handled automatically.

Dynamic display
: The dynamic display module allows users to quickly and easily build up simple textual displays for controlling an experiment. Variables can be displayed and updated in real-time (while a controller is running) and commands can be executed by binding functions to keys or "buttons".

Each of these modules is described in detail in the Sparrow reference manual [2]. We briefly review some of the details of the modules that affect operation and performance of the ducted fan system.

2.1 Servo scheduling

Low-level servo routines are available in Sparrow for executing a C function at a specified rate. These functions allow a servo loop to be run at a fixed rate while a foreground program looks for asynchronous events which may affect the servo loop (such as user input). A servo loop is implemented by taking over the PC clock interrupt and forcing it to call a user function at the specified rate (in Hertz). The actual rate of the servo loop may be slightly different than the desired rate (due to granularity of the internal timers), but is guaranteed to be at least as fast as the desired rate. The servo rate must be at least 20 Hz and has a practical maximum of about 5000 Hz (with minimal code begin executed in the loop).

When the servo routine requires more time to execute than the amount of time allocated, a *servo overflow* occurs. The servo routine will call the servo loop function a second time while the first loop is still uncompleted. When the second call to the servo routine finishes executing, control is returned to the original servo function. When that function finishes executing, control is returned to the display program. Obviously, a repeated servo overflow can lead to a problem since no function will finish executing. However, in many instances a single overflow will occur due to disk and keyboard interrupts which can delay processing slightly. Sparrow allows a maximum of 8 overlapping interrupts (by default).

The status of the servo routines is indicated by a number of display flags in the upper right corner of the screen. Table 1 summarizes the various possibilities. Clearing the display screen will erase the servo flags and they will not reappear until a change of status occurs.

`red S`	servo handler is installed, but not enabled
`green S`	servo loop is being executed at specified rate
`green, flashing O`	a servo loop overflow has occurred, but the servo loop is still executing
`red, flashing O`	a servo loop overflow has occurred and the servo loop has been stopped
`X, any color`	an unrecoverable servo error has occurred. You will need to exit the program and restart

Table 1: Servo status flags.

2.2 Data capture

Data which is processed by the channel library can be automatically captured and stored on disk for further analysis. The capture routines save data to a temporary buffered output stream. A manager function monitors the stream's buffer; when it detects that the buffer has become more than halfway filled it initiates a flush of the buffer to the stream's associated file on disk, and resets the buffer for further data capture. Conceivably, then, the amount of data that can be stored is limited only by the maximum size for a disk file. The data is saved to the temporary output stream in raw binary format because this is fast; when the experiment is finished and speed is no longer a consideration, a function may be called that will convert the data in the temporary binary file to more easily handled ASCII text.

The output data file consists of a matrix of rows and columns. Each column corresponds to a given channel, in the order specified in the device configuration file. A separate row is written for each time sample. The data can be accessed in Matlab by using the function `load` to read in the data and access it in row, column fashion.

2.3 Dynamic display

The main interface to Sparrow programs is usually through a dynamic display screen. The Sparrow dynamic display package provides a simple method for constructing a user interface for a real-time control program. It provides a set of functions that can be used to build a screen-oriented interface which allows real-time display and modification of user variables as well as the execution of commands within the control package.

The overall usage of the dynamic display interface is similar to that of widget sets in graphical user-interfaces, but much more rudimentary. A display is constructed by placing a number of objects on the screen which can be selected and modified. Each object represents a particular instance of a type of data. For example, a data object is used to display the current value of a variable and also provide a means of changing the value of that variable. A button object can be defined to allow certain functions to be executed by the user when the button is pressed. The function attached to the button may alter the flow of the real-time control program or perform operations which otherwise modify the operating environment.

Commands can be performed on an object by selecting the object and pressing a key which has an appropriate action bound to it. The currently selected object is displayed in reverse video. Valid actions on a selected object include modifying its value and executing a function associated with the object. The former action is typical for data objects, which represent variables in the program, while the later is typical for button objects which execute user functions.

The selected object can be changed using cursor motion commands. By default, these are bound to the movement keys used by the `vi` editor. The arrow keys can also be used to move the cursor.

The default key bindings are:

```
q     exit-loop         exit from the main display manager loop
=     input-item        input a new value for an object
h     left-item         move the cursor to the left
j     down-item         move the cursor to the left
k     up-item           move the cursor to the left
l     right-item        move the cursor to the left
<CR>  execute           execute the object callback function
```

In addition, the arrow keys are bound to the appropriate cursor movements. These bindings can be changed or extended by user programs.

3 Overview of Falcon

Falcon is a collection of programs and a library of C functions intended to aid in the design and implementation of real-time control systems on a variety of platforms. In general, the same routines used during simulation are also used for real-time control. This common interface speeds the transition from design to implementation. `Falcon` is intended for use design, development, and implementation of controllers in a research environment. While the code is efficient, it is not optimized for any particular platform.

The `Falcon` library is divided into three sections:

Matrix based functions
: Many common applications require matrix manipulations. `Falcon` provides a matrix computation module which is used by several other modules, including the state space computation module and the linear parameter varying control module.

Table based functions
: Both trajectories and lookup tables are frequently encountered in control problems. The trajectory tracking module allows the storage of complex trajectories in a compact form. The lookup table module support tables of arbitrary dimension.

Matlab/Simulink interface
: Since design and analysis is often done in Matlab, an interface between `Falcon` and Matlab is provided. An s-function interface to each module is provided. This allows both analysis and the real-time implementation to share common data files. As soon as a controller is designed, it is ready to be tested on the experiment.

Each of these modules is described in detail in the Falcon reference manual [1]. We briefly review some of the details of the modules that affect operation and performance of the ducted fan system.

3.1 State space computations

The state space computation module provides a quick method for implementing digital controllers and filters within Sparrow. State space systems are loaded from either Matlab data files or from a formatted ASCII files. The file must contain four matrices, named `A`, `B`, `C`, `D`. The number of inputs, outputs and states is determined from the dimensions of the appropriate matrices and must be consistent with what the user program expects.

3.2 Trajectory Files

The trajectory reading module facilitates the implementation of trajectory tracking. Piecewise linear trajectories of an arbitrary number of variables are recreated from a formatted ASCII file. Essentially, this package implements a function which maps one variable (time, for example) to a vector (the desired trajectory, for example). The trajectory reading package is optimized to implement a piecewise linear function of one dimension that is accessed in an increasing manner. While the package can be used to implement a one-dimensional lookup table, the lookup table module is more efficient for random access functions.

A trajectory file consists of an ASCII file whose first line specifies the size of the trajectory, and subsequent lines contain the data. All values are read as doubles, and extra information is ignored. The numbers can be separated by either white space or tabs. The following example shows the required file format.

```
nrows   ncols
time1   data1a  data1b  data1c  data1d ...
time2   data2a  data2b  data2c  data2d ...
...
```

The first line gives the number of time elements to be read, `nrows`, and then number of elements in the trajectory vector, `ncols`. The time column is not counted as an element of the trajectory vector. If `ncols` is less than number of columns in the file, excess columns at the end of the line are ignored. If this number is larger than the number of columns on a line, the program assumes that the next line of the file is also data. After reading the appropriate number of data elements, the program advances to the next newline before reading a time value.

3.3 Lookup Tables

The lookup table module facilitates the approximation of nonlinear functions of arbitrary dimension. Each dimension is gridded uniformly, and the function's value is specified at each grid point. Linear interpolation is used to approximate values between grid points. Essentially, this module implements a mapping from a vector to another vector, with potentially different dimensions. A one dimensional lookup table differs from a trajectory in that the lookup table requires uniform spacing of points and is optimized for random access, while the trajectory is optimized for sequential access.

Lookup tables must be generated in advance and are saved as formatted ASCII files. The initial lines of the file specify the size and domain of the lookup table, and subsequent lines contain the data. All values are read as doubles, and extra information is ignored. The numbers can be separated by either white space or tabs. The following example shows the required file format:

```
Dimension  Outputs  Interpolated
dim1_low   dim1_step   dim1_high  dim1_count
dim2_low   dim2_step   dim2_high  dim2_count
...
dimN_low   dimN_step   dimN_high  dimN_count
dim1a dim2a ... dimNa      data1a  data1b  data1c  data1d ...
dim1a dim2a ... dimNb      data2a  data2b  data2c  data2d ...
...
```

`Dimension` is the dimension of the domain of the lookup table. `Outputs` is the number of outputs, the range of the lookup table. `Interpolated` is the number of outputs which are interpolated. It is often useful (status variables, for example) to not interpolate an output. Only the first `Interpolated` outputs are interpolated, the remaining ones are not. `dimI_low` and `dimI_high` specify the domain of the table in Ith dimension. `dimI_step` specifies the step size, and `dimI_count` is the number of grid points for the Ith dimension. This contains redundant information, since low + (count - 1) * step = high.

Finally, the coordinates and data are listed. Each line of the data specifies the function's value at one grid point. For an N-dimensional table with M outputs, the first N numbers specify the coordinate and the remaining numbers specify the output. The order is such that first dimension is held constant, while all other dimensions are incremented. Since the position in the datafile uniquely specifies the coordinate, the coordinate data is redundant. However, storing the data in a file greatly simplifies human editing of lookup tables. Currently, no verification of the coordinates is done, though this might change in the future.

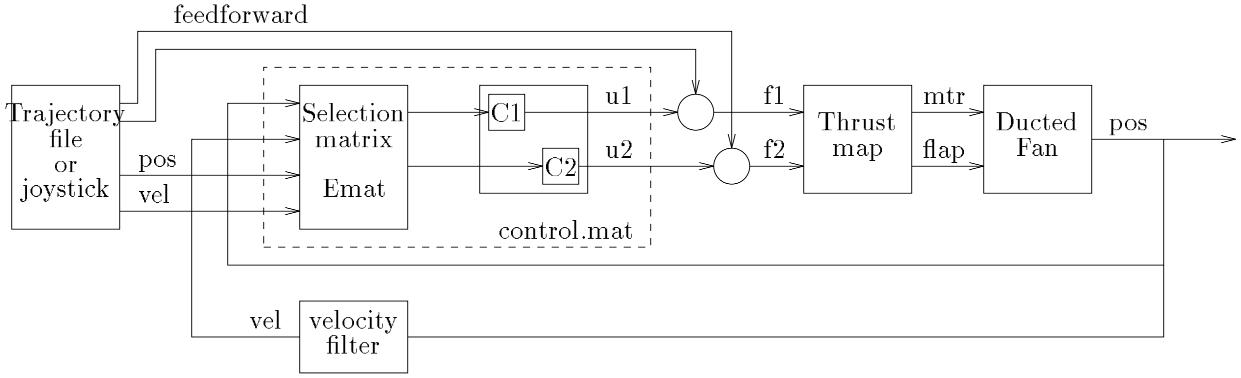

Figure 2: Ducted fan control architecture

4 User interface to the ducted fan

The ducted fan interface program, `dfan`, can be used to load and execute controllers on the ducted fan system, load and execute pre-computed trajectories, and capture and dump real-time data. The `dfan` program is built on top of the Sparrow library described above and makes use of the standard Sparrow interface routines. In this section we describe some of the details of the interface program and indicate how to use it for testing controllers on the ducted fan.

The basic control architecture for the ducted fan is shown in Figure 2 The controller (in the dotted box) accepts 12 inputs (the actual and desired position and velocity) and has two outputs. The outputs from the controller are added to any feedforward forces and then passed through a thrust map which converts desired forces to motor currents and flap angles. The output from the ducted fan are the position variables, which are then filter to determine the velocity.

The main display for the ducted fan is shown in Table 2. The top portion of the display shows the status and outputs of the main blocks of the controller. The middle of the display contains the current and desired states as well as paths to the configuration files which define the controllers. The bottom of the display lists the bindings of several function keys which have been bound to common actions.

Real-time data for the ducted fan can be captured and dumped to disk using the `F5` and `F6` function keys. The `F5` key toggles capture mode. When data is being captured a green 'C' will appear in the upper right corner of the screen. Data is dumped to disk using the `F6` key; the data format is given in Table 3. Data capture should be turned off before data is dumped to disk.

```
                              Caltech Ducted Fan
                                            Motor:  2.6467  Flaps:  -0.219  -0.219

        Gain    Control   FF      TBL     | Joystick      197   216       3
Axial    1.00     0.000   2.651   0.000   |   motor/flap  f1/f2
Side     1.00     0.000   0.000   0.000   |   pitch/alt   XY posn

State    Value   Goal    Error      | Configuration files:
Xpos:   -24.875   0.000  24.875     |   Chn config: config.dev
Ypos:    -0.041   0.000   0.041     |   Controller: d:/cds111/lib/dec_cg2.mat
Theta    -0.019   0.000   0.019     |   Vel filter: d:/cds111/lib/vf.mat
Xdot:     0.000   0.000   0.000     |   Thrust map: d:/cds111/lib/invmap.tbl
Ydot:     0.000   0.000   0.000     |   Traj file:
Thdot:    0.000   0.000   0.000     |   Dump file:  c:/tmp/dfan.dat

<F1> hover                  <F5> toggle capture     <F9>  reset defaults
<F2> pitch/alt ctrl         <F6> dump data          <F10> zero counters
<F3> XY control             <F7> start trajectory
<F4> raw ctrl                                       <ESC> control off

Emergency stop: hit spacebar or both joystick buttons
```

Table 2: Ducted fan user interface

Column	Signal	Description	Units	Formula
1	a3	pitch encoder	rad	$\theta = -\mathtt{a3}$
2	a2	altitude encoder	rad	$y = r_f * \mathtt{a2}$
3	a1	position encoder	rad	$x = r_f * \mathtt{a1}$
4	Xdot	X velocity	m/sec	$\dot{x}$
5	Ydot	Y velocity	m/sec	$\dot{y}$
6	Thdot	Pitch rate	rad/sec	$\dot{\theta}$
7	u1	Sideways force	N	u_1
8	u2	Axial force	N	u_2
9	Xdes	Desired position	m	x_{des}
10	Ydes	Desired altitude	m	y_{des}
11	Thdes	Desired pitch	m	θ_{des}

Table 3: Capture file format for ducted fan interface program.

5 Loading controllers

Controllers for the ducted fan are specified using a discrete time control law with 12 inputs and 2 outputs. The 12 inputs consist of the ducted fan position (x, y, θ), the estimated velocities, the desired position and the desired velocity. By choosing the proper linear transformation of the input variables, it is possible to implement stabilization and/or tracking using any of the positions and velocities.

To illustrate how a controller is specified, consider the following continuous time transfer functions for the altitude and pitch loops:

$$C_{\text{pitch}} = \frac{1}{4}\frac{s+10}{s/100+1} \qquad C_{\text{alt}} = \frac{5(s+1)}{s/10+1}$$

In Matlab, these controllers would be specified as

```
C1num = 0.25 * [1 10];
C1den = [1/100 1];
[A1c, B1c, C1c, D1c] = tf2ss(C1num, C1den);
```

and similarly for the altitude controller. These controllers can be converted to discrete time form using the `c2dm` function in Matlab:

```
[A1d, B1d, C1d, D1d] = c2dm(A1c, B1c, C1c, D1c, 1/200, 'matched');
```

A similar set of operations can be used to compute `[A2d, B2d, C2d, D2d]` for the altitude controller.

The next step is to combine the controllers and put them into the input/output format required by the `dfan` program. We start by creating a 2 input, 2 output (diagonal) controller from the SISO controllers:

```
[Ad, Bd, Cd, Dd] = append(A1d, B1d, C1d, D1d, A2d, B2d, C2d, D2d);
```

In order to create the 12 input, 2 output controller required by the `dfan` program, we multiple the inputs by a selection matrix which combines the actual and desired positions and velocities appropriately. For example, to send the pitch error to the first controller input and the altitude error to the second input, we use the following commands in Matlab:

```
# Select pitch and altitude error
Emat = [0 0 1 0 0 0 0 0 -1 0 0 0; 0 1 0 0 0 0 0 -1 0 0 0 0];

# Build up the final control law
A = Ad;
B = Bd * Emat;
C = Cd;
D = Dd * Emat;
```

Finally, we save the controller A, B, C, D matrices in a file:

```
save sample A B C D
```

This creates the file `sample.mat`, which can be used as the controller on the ducted fan control screen.

Column	Signal	Description	Units
1	t	time	sec
2	u2	Axial force	N
3	u1	Sideways force	N
4	Xpos	Desired position	m
5	Ypos	Desired altitude	m
6	Theta	Desired pitch	m
7	Xdot	Desired X velocity	m/sec
8	Ydot	Desired Y velocity	m/sec
9	Thdot	Desired Pitch rate	rad/sec

Table 4: Trajectory file format for ducted fan interface program.

6 Generating trajectories

Trajectories are implemented using the trajectory generation routines in the Falcon library. Each trajectory must specify the desired position and velocity as a function of time as well as the nominal feedforward forces along the trajectory. Only the data that is required for a given control algorithm actually needs to be valid data, but all entries must be present. The detailed format of the trajectory file is given in the Falcon documentation. The contents of each row are listed in Table 4 Note that the entries for `u2` and `u1` are inverted in the trajectory table.

The following predefined trajectries are available in the ducted fan library:

```
steps#.trj  combined step response
  steps1.trj       (Mike)         friction + aero
  steps1a.trj      (RMM)          same but constant FF terms
  steps2.trj       (Mike)         friction, no aero
  steps4.trj       (Michiel)      planar model, mx != my

roller.trj  (Mike)         rolling trajectory (2 to 1, XY only)
broll.trj   (Bobby)        second rolling traj (1.5:1, faster, XY only)
```

7 Additional customization

In addition to changing the controller and commanded trajectory, the `dfan` program can be programmed to use alternative velocity filters, thrust maps, and calibration constants. This section briefly documents the form of relevant data files.

7.1 Velocity filters

The ducted fan hardware measures the position of the axes of the stand using a set of quadrature encoders connected to a set of decoder chips. In order to obtain the velocity of the fan, a filter is used to approximately differentiate the position signal. The filter is a standard discrete time, state space system which is applied to each position channel separately. It has a single input (the position) and a single output (the velocity).

7.2 Thrust map

In order to apply the forces commanded by the controller, a thrust map is used which converts the desired axial and perpendicular forces to motor speed and flap angle commands. This thrust map is in the form of a Falcon lookup table with the following format (see [1] for details):

```
2 3 2
0.200000 0.200000 6.800000 34
-2.600000 0.200000 3.400000 31
f1 f2 motor flap flags
```

7.3 R/C servo calibration

In order to properly command the flap angles, the R/C servos must be calibrated and the proper calibration constants entered in Sparrow. The default constants are specified in `dfan.ini`:

```
pwm_offset[0]   1.250000
pwm_gain[0]     -0.010000
pwm_offset[1]   1.400000
pwm_gain[1]     -0.010000
```

References

[1] M. Kantner. *Falcon 1.0 Reference Manual.* California Institute of Technology, 1995. Available electronically from `http://avalon.caltech.edu/~sparrow`.

[2] R. M. Murray, E. L. Wemhoff, and M. Kantner. *Sparrow 2.1 Reference Manual.* California Institute of Technology, 1995. Available electronically from `http://avalon.caltech.edu/~sparrow`.