Learning to Fly

Claude Sammut
Scott Hurst
Dana Kedzier

School of Computer Science and Engineering
University of New South Wales
Sydney, Australia

Donald Michie

The Turing Institute
36 North Hanover Street
Glasgow, G1 2AD
United Kingdom

Abstract

This paper describes experiments in applying inductive learning to the task of acquiring a complex motor skill by observing human subjects. A flight simulation program has been modified to log the actions of a human subject as he or she flies an aircraft. The log file is used to create the input to an induction program. The output from the induction program is tested by running the simulator in autopilot mode where the autopilot code is derived from the decision tree formed by induction. The autopilot must fly the plane according to a strictly defined flight plan.

1. THE PROBLEM

In this paper, we report on experiments that demonstrate machine learning of a reactive strategy to control a dynamic system by observing a controller that is already skilled in the task. We have modified a flight simulation program to log the actions taken by a human subject as he or she flies an aircraft. The log file is used to create the input to an induction program. The quality of the output from the induction program is tested by running the simulator in autopilot mode where the autopilot code is derived from the decision tree formed by induction.

A practical motivation for trying to solve this problem is that it is often difficult to construct controllers for complex systems using classical methods. Anderson and Miller (1991) describe a problem with present-day autolanders, namely that they are not designed to handle large gusts of wind when close to landing. Similar problems occur for helicopter pilots who must manoeuvre their aircraft in high winds while there is a load slung beneath the helicopter. Learning by trial-and-error could be used in simulation, but if we already have a skilled controller, namely, a human pilot, then it is more economical to learn by observing the pilot.

While control systems have been the subject of much research in machine learning in recent years, we know of few attempts to learn control rules by observing human behaviour. Michie, Bain and Hayes-Michie (1990) used an induction program to learn rules for balancing a pole (in simulation) and earlier work by Donaldson (1960), Widrow and Smith (1964) and Chambers and Michie (1969) demonstrated the feasibility of learning by imitation, also for pole-balancing. To our knowledge, the autopilot described here is the most complex control system constructed by machine learning methods. The task we set ourselves was to teach the autopilot how to take off; fly to a set altitude and distance; turn around and land. We describe our experiments with a particular aircraft simulation and discuss the problems encountered and how they were solved. We also discuss some of the remaining difficulties.

2. THE FLIGHT SIMULATOR

The source code to a flight simulator was made available to us by Silicon Graphics Incorporated. The central control mechanism of the simulator is a loop that interrogates the aircraft controls and updates the state of the simulation according to a set of equations of motion. Before repeating the loop, the instruments in the display are updated. The simulator gives the user a choice of aircraft to fly. We have restricted all of our experiments to the simulation of a Cessna, being easier for our subjects to learn to fly than the various fighters or larger aircraft available.

One feature of the flight simulator that has had a significant effect on our experiments is that it is non-deterministic. The simulator runs on a multi-tasking Unix system, not on a dedicated real-time system. Thus, it is not possible to give a guaranteed real-time response because the flight simulator can be interrupted by other processes or I/O traffic. If nothing is done to compensate for these interruptions, a person operating the simulator would notice that the program's response to control actions would change. If no other processes were stealing CPU time it would respond quickly but it could become very sluggish when other processes were competing for the CPU.

To minimise the effects of variations in execution speed, the simulator regularly interrogates a real-time clock. This is used to calculate the number of main control loops be-

ing executed each second. If the simulation has slowed down since the last interrogation, the time interval used in solving the equations of motion is altered to allow the simulation to 'catch up'. The time interval is also changed in response to an increase in execution speed. To a human operator, who has a sense of time, this approximates uniform response. However, these adjustments do not ensure a perfectly uniform response. Therefore, to an autopilot that has no external sense of time, the effects of its control actions will be somewhat different from one run to the next and even during one flight.

We have chosen to treat this problem as a challenge. If we are able to devise rules that can control a noisy system, we will have done well and in fact, the rules that have been generated can handle considerable variation. Thus we can be optimistic that the methods we are developing can be extended to more complex systems that have real disturbances such as wind and genuinely noisy controls.

Another 'feature' that we discovered about the Silicon Graphics flight simulator is that the rudder does not have a realistic effect on the aircraft. Fortunately this did not affect us since none of our pilots used the rudder. While a real pilot would frown upon this practice, it is possible to fly a real airplane without using the rudder (the rudder is used in turns to stop the plane from 'sliding' with the result that the g-forces are not directed towards the floor as they should be).

3. LOGGING FLIGHT INFORMATION

The display update has been modified so that when the pilot performs a control action by moving the control stick (the mouse) or changing the thrust or flaps settings, the state of the simulation is written to a log file. Initially, we obtained the services of 20 volunteers, believing that the more logs we had from a variety of subjects the more robust would be our rules. As we discuss later, we found that it was better to collect many logs from a small number of pilots. All the results presented below are derived from the logs of three subjects who each 'flew' 30 times.

At the start of a flight, the aircraft is pointing North, down the runway. The subject is required to fly a well-defined flight plan that consists of the following manoeuvres:

1. Take off and fly to an altitude of 2,000 feet.
2. Level out and fly to a distance of 32,000 feet from the starting point.
3. Turn right to a compass heading of approximately 330˚. The subjects were actually told to head toward a particular point in the scenery that corresponds to that heading.
4. At a North/South distance of 42,000 feet, turn left to head back towards the runway. The scenery contains grid marks on the ground. The starting point for the turn is when the last grid line was reached. This corresponds to about 42,000 feet. The turn is considered complete when the azimuth is between 140˚ and 180˚.
5. Line up on the runway. The aircraft was considered to be lined up when the aircraft's azimuth is less than 5˚ off the heading of the runway and the twist is less that ±10˚ from horizontal.
6. Descend to the runway, keeping in line. The subjects were given the hint that they should have an 'aiming point' near the beginning of the runway.
7. Land on the runway.

We will refer to the performance of a control action as an 'event'. During a flight, up to 1,000 events can be recorded. With three pilots and 30 flights each the complete data set consists of about 90,000 events. The data recorded in each event are:

on_ground	boolean: is the plane on the ground?
g_limit	boolean: have we exceeded the plane's g limit
wing_stall	boolean: has the plane stalled?
twist	integer: 0 to 360˚ (in tenths of a degree, see below)
elevation	integer: 0 to 360˚ (in tenths of a degree, see below)
azimuth	integer: 0 to 360˚ (in tenths of a degree, see below)
roll_speed	integer: 0 to 360˚ (in tenths of a degree per second)
elevation_speed	integer: 0 to 360˚ (in tenths of a degree per second)
azimuth_speed	integer: 0 to 360˚ (in tenths of a degree per second)
airspeed	integer: (in knots)
climbspeed	integer: (feet per second)
E/W distance	real: E/W distance from centre of runway (in feet)
altitude	real: (in feet)
N/S distance	real: N/S distance from northern end of runway (in feet)
fuel	integer: (in pounds)
rollers	real: ±4.3
elevator	real: ±3.0
rudder	real: not used
thrust	integer: 0 to 100%
flaps	integer: 0˚, 10˚ or 20˚

The elevation of the aircraft is the angle of the nose relative to the horizon. The azimuth is the aircraft's compass heading and the twist is the angle of the wings relative to the horizon. The elevator angle is changed by pushing the mouse forward (positive) or back (negative). The rollers are changed by pushing the mouse left (positive) or right (negative). Thrust and flaps are incremented and decremented in fixed steps by keystrokes. The angular effects of the elevator and rollers are cumulative. For example, in

straight and level flight, if the stick is pushed left, the aircraft will roll anti-clockwise. The aircraft will continue rolling until the stick is centred. The thrust and flaps settings are absolute.

A valid criticism of our data collection method is that we are not recording the same information that the subject is using and thus we make it difficult for the induction program to reproduce the pilot's behaviour. For example, it was mentioned previously that subjects use an aiming point on the runway to adjust their trajectory while approaching the runway. No information directly related to an aiming point is recorded in the data. Our assumption is that enough other data are recorded to allow the induction program to do its job.

RESPONSE TIMES

When an event is recorded, the state of the simulation at the instant that an action is performed could be output. However, there is always a delay in response to a stimulus, so ideally we should output the state of the simulation when the stimulus occurred along with the action that was performed some time later in response to the stimulus. But how do we know what the stimulus was? Unfortunately there is no way of knowing. Human responses to sudden stimuli take approximately one second but this can vary considerably. For example, while flying, the pilot usually anticipates where the aircraft will be in the near future and prepares the response before the stimulus occurs.

Our approach has been is as follows. Each time the simulator passes through its main control loop, the current state of the simulation is stored in a circular buffer. We estimate how many loops are executed each second. When a control action is performed, the action is output, along with the state of the simulation as it was some time before. How much earlier is determined by the size of the buffer. Of the three subjects used in these experiments, one operated the simulator with a delay of 40 loops (corresponding to a two or three second delay) and the other two subjects used a 20 loop delay (between one and one and a half seconds).

4. DATA ANALYSIS

Even with a well-specified flight plan such as the one we are using here, there is a large degree of variation in the way different subjects fly. Because of this variation, the number of flights we have is not sufficient to allow an induction program to distinguish useful actions from noise using the raw data. However, it would not be very practical if it were necessary to fly hundreds of flights before anything useful could be obtained. So before applying the induction program to the data, we perform some analysis to assist it.

We have used C4.5 (Quinlan, 1987) as the induction program in these experiments. Learning reactive strategies is a task for which C4.5 was never intended. However, we chose it for our initial investigation because we are familiar with it and it is reliable and well known. Having the source code also made it easier for us to generate the decision trees as if-statements in C. This was necessary so that the decision tree code be inserted into the simulator.

CUSTOMISED AUTOPILOTS

The learning task was simplified by restricting induction to one set of pilot data at a time. Thus, an autopilot has been constructed for each of the three subjects who generated training data. The reason for separating pilot data is that each pilot can fly the same flight plan in different ways. For example, straight and level flight can be maintained by adjusting the throttle. When an airplane's elevation is zero, it can still climb since higher speeds increase lift. Adjusting the throttle to maintain a steady altitude is the correct way of achieving straight and level flight. However, another way of maintaining constant altitude is to make regular adjustments to the elevators causing the airplane to pitch up or down. One of the subjects flew stage 2 by adjusting the throttle, the other two adjusted the elevators. We want the induction program to learn a consistent way of flying, so we are training it to emulate a particular pilot.

FLIGHT STAGES

The data from each flight were segmented into the seven stages described in section 3. In the flight plan described, the pilot must achieve several, successive goals, corresponding to the end of each stage. Each stage requires a different manoeuvre. Having already defined the sub-tasks and told the human subjects what they are, we gave the learning program the same advantage.

DECISION TREES AND CONTROL ACTIONS

In each stage we construct four separate decision trees, one for each of the elevator, rollers, thrust and flaps. A program filters the flight logs generating four input files for the induction program. The attributes of a training example are the flight parameters described earlier. The dependent variable or class value is the attribute describing a control action. Thus, when generating a decision tree for flaps, the flaps column is treated as the class value and the other columns in the data file, including the settings of the elevator, rollers and thrust, are treated as ordinary attributes.

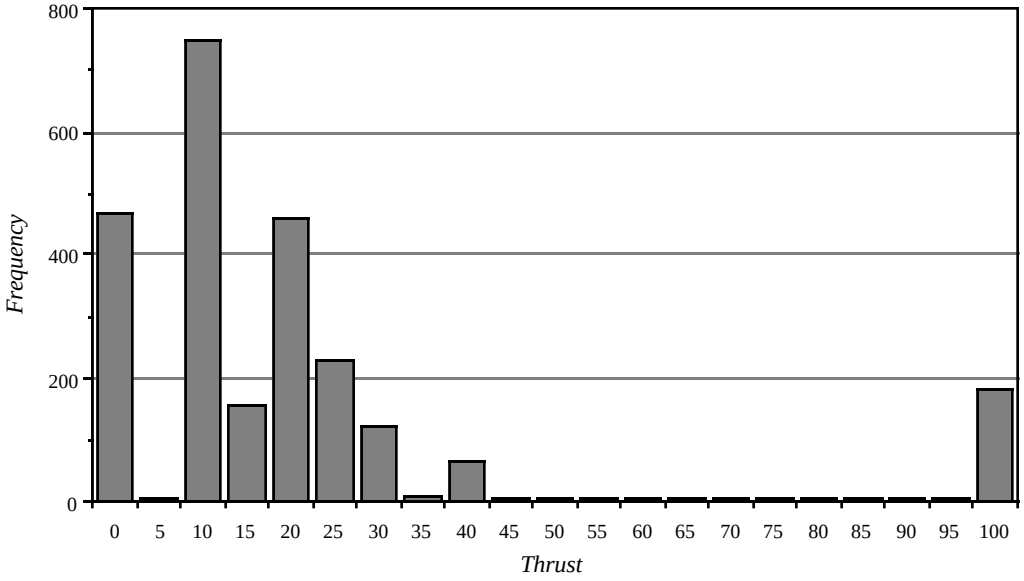

Figure 1. Frequency of thrust values in stage 6

DETERMINING CLASS VALUES

C4.5 expects class values to be discrete but the values for elevator, rollers, thrust and flaps are numeric. We will soon be experimenting with decision tree induction programs that have numeric output. However, for these experiments, a preprocessor breaks up the action settings into sub-ranges that can be given discrete labels. Sub-ranges are chosen by analysing the frequency of occurrence of action values. This analysis must be done for each pilot to correctly reflect differing flying styles. There are two disadvantages to this method. One is that if the sub-ranges are poorly chosen, the rules generated will use controls that are too fine or too coarse. Secondly, C4.5 has no concept of ordered class values, so classes cannot be combined during the construction of the decision tree.

Figure 1 shows the frequency of thrust values in stage 6 of the data for one pilot. Since thrust is controlled by a keystroke, it is increased and decreased by a fixed amount, 10%. The values with very low frequencies are those that were passed through on the way to a desired setting. The graph reflects the facts that this pilot held the thrust at 100% until the approach to the runway began. The thrust was then brought down to 40% immediately and gradually decreased to 10% where it remained for most of the approach. Close to the runway, the thrust was cut to 0 and the plane glided down the rest of the way.

In this case, class values corresponding to 0, 10, 15, 10, 25, 30, 35, 40 and 100 were used. Anything above 40% was considered full-throttle. Anything below 10% was considered idle. Another reasonable clustering of values could be to group values from 15 to 35 together.

ABSOLUTE AND INCREMENTAL CONTROLS

An event is recorded when there is a change in one of the control settings. A change is determined by keeping the previous state of the simulation in a buffer. If any of the control settings are different in the current state, a change is recognised. For example, if the thrust is being reduced from 100% to 40%, all of the values in between are recorded. For thrust, these values are easily eliminated as noise during induction.

It is not so easy to eliminate spurious values from the elevator and rollers data. Both thrust and flaps can be set to a particular value and left. However, the effects of the elevator and rollers are cumulative. If we want to bank the aircraft to the left, the stick will be pushed left for a short time and then centred since keeping it left will cause the airplane to roll. Thus, the stick will be centred after most elevator or roller actions. This means that many low elevator and roller values will be recorded as the stick is pushed out and returned to the centre position.

To ensure that records of low elevator and roller values do not swamp the other data, another filter program removes all but the steady points and extreme points in stick movement. Figure 2 shows a small sample of roller settings during a flight. Each point on the graph represents one event. Clearly many of the points are recorded as part of a single movement. The filter program looks for points

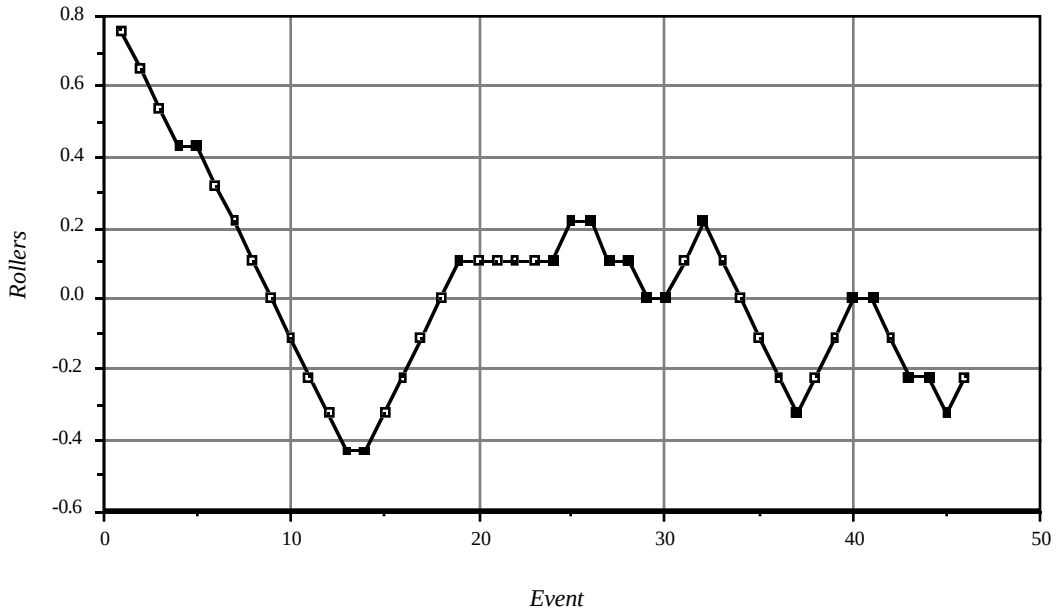

Figure 2. Change in rollers

of inflection in the graph and only passes those on to the induction program. In this graph, only the points marked in black will get through the filter.

5. GENERATING THE AUTOPILOT

After processing the data as described above, we can finally submit them to C4.5 to be summarised as rules that can be executed in a controller.

PRUNING THE DECISION TREE

C4.5 has two parameters that can be varied by the user to adjust tree pruning. We have experimented with them to try to obtain the simplest workable rules. One parameter controls C4.5's confidence level. That is, the program will prune the decision tree so that it maintains a minimum classification accuracy with respect to test data. The second parameter controls the minimum number of instances required for a split. For example, if this parameter is set to 10, then no branch in the tree will be created unless at least 10 examples descend down that branch.

We proceed by generating decision trees using the default parameter settings, testing the rules in the simulator and then gradually adjusting the parameters to obtain simpler rules. This continues until the rule 'breaks, ie. it is no longer able to control the plane correctly.

TIME AND CAUSALITY

The rules constructed by C4.5 are purely reactive. They make decisions on the basis of the values in a single state of the simulation. The induction program has no concept of time or causality. In connection with this, some strange rules can turn up. For example, the rule below for thrust in the descent stage was derived from data that was not filtered as described above. There were 2,513 examples in the training set, the minimum split size was set to 5 (since the data from five flights were combined) and the confidence parameter was set to 0.1%.

```
airspeed > 127 : thrust_100
airspeed <= 127 :
| X_feet > 121.33 : thrust_30
| X_feet <= 121.33 :
| | elevation <= -43 :
| | | Z_feet > -11514.8 : thrust_0
| | | Z_feet <= -11514.8 :
| | | | climbspeed <= -13 : thrust_0
| | | | climbspeed > -13 :
| | | | | Z_feet > -18475.8 : thrust_10
| | | | | Z_feet <= -18475.8 :
| | | | | | Y_feet <= 1535.21 : thrust_20
| | | | | | Y_feet > 1535.21 : thrust_10
| | elevation > -43 :
| | | Y_feet <= 638.76 : thrust_25
| | | Y_feet > 638.76 :
| | | | Z_feet <= -26230.1 : thrust_15
| | | | Z_feet > -26230.1 : thrust_20
```

Only the first two lines of this rule are of interest at present. The first line states that when the airspeed is greater than 127 knots then the thrust should be 100%. When the airspeed is less than or equal to 127 knots the thrust is lower. The exact value being determined by the remainder of the decision tree. Thus C4.5 has correctly detected a correlation between speed and thrust. Unfortunately it uses the speed to determine the thrust when it should be the other way around.

By introducing the response time delays described in section 3 and the filtering in section 4 causality problems can be overcome to some extent, but rules like this sometimes still occur. At present the only way around this is to hope that savage pruning will improve the rule. For the case above, C4.5 was re-run, this time with the minimum split size set to 500 resulting in the following rule:

```
Z_feet <= -30642 : thrust_100
Z_feet > -30642 :
| elevation > -43 : thrust_20
| elevation <= -43 :
| | Z_feet <= -16382 : thrust_10
| | Z_feet > -16382 : thrust_0
```

This is quite sensible. Z_feet is the distance from the runway. As the airplane nears the runway, it decreases thrust progressively. The elevation rule says that if the nose is pointing down by more than 4.3˚ then increase the thrust to 20%. This will cause the nose to rise and then the thrust will be reduced to 0 or 10% depending on the distance from the runway. While we wish the aircraft to descend during this stage of the flight, it should not descend too steeply. This rule, working with the elevator rule controls the angle of descent.

We believe that learning could be improved by including some knowledge of causality in the system so that it is able to correctly identify dependencies among variables.

6. LINKING THE AUTOPILOT WITH THE SIMULATOR

To test the induced rules, the original autopilot code in the simulator is replaced by the rules. A post-processor converts C4.5's decision trees into if-statements in C so that they can be incorporated into the flight simulator easily. Hand-crafted C code determines which stage the flight has reached and decides when to change stages. The appropriate rules for each stage are then selected in a switch statement. Each stage has four, independent if-statements, one for each action.

DELAYS

When the data from the human pilots were recorded, a delay to account for human response time was included. Since the rules were derived from this data, their effects should be delayed by the same amount as was used when the data were recorded. When a rule fires, instead of letting it effect a control setting directly, the rule's output value is stored in a circular buffer. There is one for each of the four controls. The value used for the control setting is one of the previous values in the buffer. A lag constant defines how far to go back into the buffer to get the control setting. The size of the buffer must be set to give a lag that approximates the lag when the data were recorded.

AVERAGING CONTROL SETTINGS

Earlier we had shown how we eliminate intermediate values in roller and elevator actions so that the induction program is not swamped with spurious data. The rules that result from this data can set values instantaneously as if the stick were moved with infinite speed from one position to another. Clearly this is unrealistic. When control values are taken from the delay buffer, they enter another circular buffer. The controls are set to the average of the values in the buffer. This ensures that controls change smoothly. The larger the buffer, the more gentle are the control changes. By experimentation, we have found that a buffer length of 5 approximates the speed with which the human pilots moved the controls.

7. FLYING ON AUTOPILOT

We have succeeded in synthesising control rules for a complete flight, including a safe landing. The rules fly the Cessna in a manner very similar to that of the pilot whose data were used to construct the rules. In some cases, the autopilot flies more smoothly than the human pilot. We demonstrate how these rules operate by describing the controllers built for the first four stages. The last three stages are too complex to include in this paper.

STAGE 1

The critical rule at take-off is the elevator rule:

```
elevation > 4 : level_pitch
elevation <= 4 :
| airspeed <= 0 : level_pitch
| airspeed > 0 : pitch_up_5
```

This states that as thrust is applied and the elevation is level, pull back on the stick until the elevation increases to 4˚. Because of the delay, the final elevation usually reaches 11˚ which is close to the values usually obtained by the pilot. `pitch_up_5` indicates a large elevator action, whereas, `pitch_up_1` would indicate a gentle elevator action. The other significant control at this stage is flaps:

```
elevation <= 6 : full_flaps
elevation > 6 : no_flaps
```

Once the aircraft has reached an elevation angle of 6˚, the flaps are raised.

STAGE 2

In stage 2, the autopilot is required to attain level flight. Again this is done through the elevator rule:

```
climbspeed <= 13 : level_pitch
climbspeed > 13 : pitch_down_1
```

When the climb rate exceeds 13 feet/second push the stick forward gently to bring climb rate down. While this rule makes sense, it does not completely stop the climb. The pilot timed the application of the control carefully so that by the time the stick was re-centred the climb rate was zero and remained so. This rule brings the climb rate down significantly but does not zero it. As a result, the aircraft climbs more than the pilot would have allowed it to.

STAGE 3

Stage 3 requires a gentle right turn. The rollers rule is:

```
twist <= -23 : left_roll_3
twist > -23 :
| azimuth <= -25 : no_roll
| azimuth > -25 : right_roll_2
```

C4.5 was designed to handle categorical and numeric values, but it was not designed to handle values like angles and compass headings that are circular. To help it a little, values such as twist and azimuth (compass heading) which range from 0 to 360˚ were converted to ranges from -180˚ to +180˚. Thus, the roll rule states that while the twist is less than 23˚ from horizontal and the aircraft is heading North, bank right. When the twist has reached 23˚ bank left to steady the aircraft. The azimuth rule ensure that the airplane will not bank right again if its heading is more than 25˚ off North.

STAGE 4

The stage 4 rules are more complex than those for the previous stages. To make them understandable, they have been greatly simplified by over-pruning. They are presented to illustrate an important point, that is that rules can work in tandem although there is no explicit link between them. The following rules are for the rollers and elevator in a sharp turn left.

```
azimuth > 114 : right_roll_1
azimuth <= 114 :
| twist <= 8 : left_roll_4
| twist > 8 : no_roll

twist <= 2 : level_pitch
twist > 2 :
| twist <= 10 : pitch_up_1
| twist > 10 : pitch_up_2
```

A sharp turn requires coordination between roller and elevator actions. As the aircraft banks to a steep angle, the elevator is pulled back. The rollers rule states that while the compass heading has not yet reached 114˚, bank left provided that the twist angle does not exceed 8˚. The elevator rule states that as long as the aircraft has no twist, leave the elevator at level pitch. If the twist exceeds 2˚ then pull back on the stick. The stick must be pulled back more sharply for a greater twist. Since the rollers cause twist, the elevator rule is invoked to produce a coordinated turn.

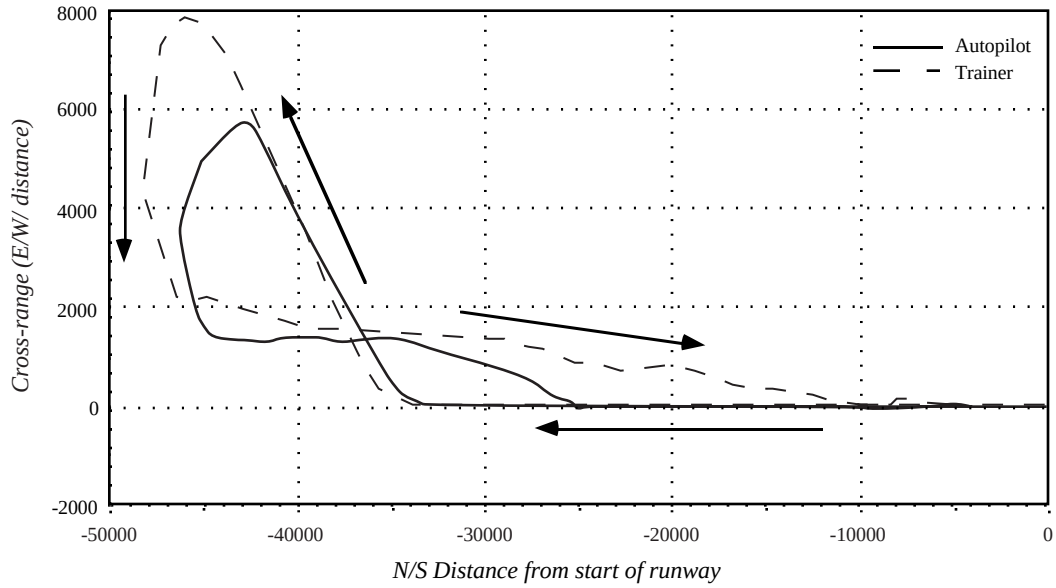

Figure 3: Cross-range Profile for Trainer and Autopilot

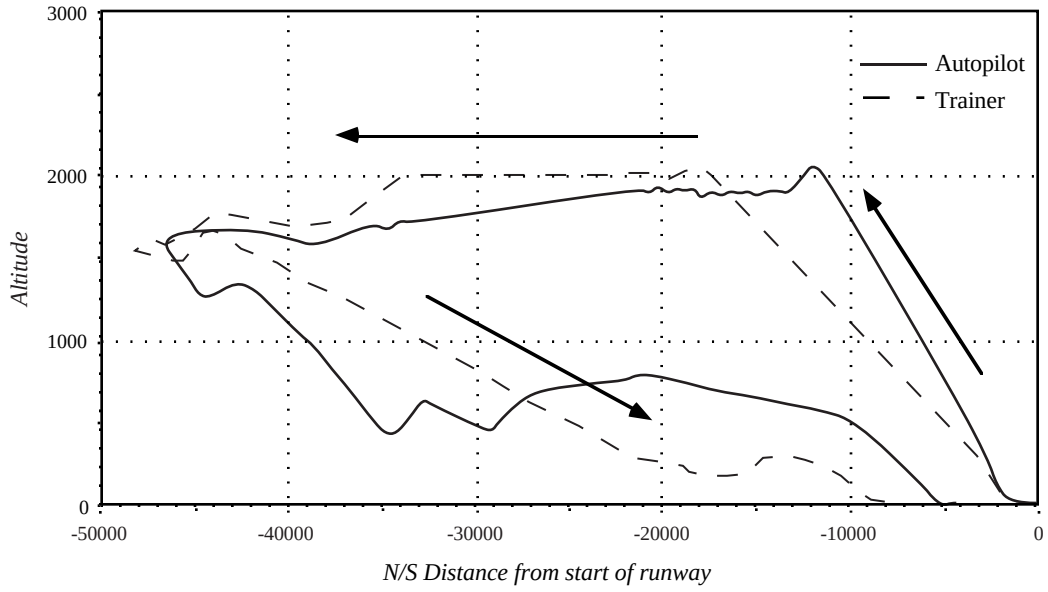

Figure 4: Altitude Profile for Trainer and Autopilot

THE COMPLETE FLIGHT

The best way of measuring the performance of the rules is to compare a flight by a human pilot with a flight by the autopilot. Figures 3 and 4 show profiles of a flight by one of the human pilots plotted with the profiles of a flight by the autopilot derived from that pilot's data. Figure 3 shows the ground track of the aircraft while Figure 4 shows a horizontal view of the flight path. Note that the vertical and horizontal axes are shown at different scales.

8. DISCUSSION

One of the interesting things we have learned in this study is that good pilots are bad! The autopilot whose profiles are shown in figures 4 and 5 was derived from a human pilot who had to make many course corrections during his flights. While such a flight is not a pretty sight, it provides useful data for the induction program. Pilots who are frugal in their use of the controls give few examples of what to do when things go wrong.

We have observed a "clean-up" effect noted in Michie, Bain and Hayes-Michie (1990). The flight log of any trainer will contain many spurious actions due to human inconsistency and corrections required as a result of inattention. It appears that effects of these examples are pruned away by C4.5, leaving a control rule which flies very smoothly. This effect was particularly noticeable in the approach stage of the flight when the trainer performed many roll manoeuvres to keep the aircraft lined-up on the runway. We have informally observed that the autopilot does a much better job of maintaining a steady glide path to the runaway. Future experiments will attempt to quantify this difference in performance.

9. CONCLUSION

Almost all applications of inductive learning, so far, have been in classification tasks such as medical diagnosis. For example, medical records of patients' symptoms and accompanying diagnoses made by physicians are entered into an induction program which constructs rules that will automatically diagnose new patients on the basis of the previous data. The output is a classification. Just as diagnostic rules can be learned by observing a physician at work, we should be able to learn how to control a system by watching a human operator at work. In this case, the data provided to the induction program are logs of the actions taken by the operator in response to changes in the system. We have used a simple procedural model for inductively building sets of control rules. An induced rule-set constitutes a "strategy" for the given sub-task – a kind of classifier that maps state records into action names, rather than mapping patient records into disease names. But in both the medical and control cases, there is a fundamental difference to be drawn between a purely symptomatic classification and one based on an understanding of the domain's causality. The latter requires declarative models able to support "what if" analysis, recognising that actions (just like diseases) occur in response to, and result in, changes in the system being controlled. Symptomatic classification only deals with static data and does not cope explicitly with temporal and causal relations.

In our preliminary study we were able to demonstrate the feasibility of learning a specific control task. The next challenge is to build a generalised method that can learn basic skills that can be used in a variety of tasks. These skills become building blocks that can be assembled into a complete new controller to meet the demands of a specified task.

One of the limitations we have encountered with existing learning algorithms is that they can only use the primitive attributes supplied in the data. This results in control rules that cannot be understood by a human expert. Constructive induction (or predicate invention) may be necessary to build higher-level attributes that simplify the rules. A methodology for doing this, using the human expert as the source of the required backward-chained control hierarchy, is known as "structured induction" (Shapiro, 1987). Our requirement now is for progress towards automating this kind of structuring (see Muggleton and Buntine, 1988).

Machine learning of control systems may lead to a better understanding of subcognitive skills which are inaccessible to introspection. For example, if you are asked by what method you ride a bicycle, you will not be able to provide an adequate answer because that skill has been learned and is executed at a subconscious level. By monitoring the performance of a subcognitive skill, we are able to construct a functional description of that skill in the form of symbolic rules. This not only reveals the nature of the skill but also may be used as an aid to training since the student can be explicitly shown what he or she is doing.

Learning control rules by induction provides a new way of building complex control systems quickly and easily. Where these involve safety critical tasks, the "clean-up" effect mentioned in the Discussion holds particular interest. While our experiments have been primarily concerned with flight automation, inductive methods can be applied to a wide range of related problems. For example, an anaesthetist can be seen as controlling a patient in an operating theatre in much the same way as a pilot controls an aircraft. The anaesthetist monitors the patient's condition just as a pilot monitors the aircraft's instruments. The anaesthetist changes dosages of drugs and gases to alter the state of a system (the patient) in the same way that a pilot alters thrust and attitude to control the state of a system (the aircraft). A flight plan can be divided into stages where different control strategies are required, eg. take-off, straight and level flight, landing, etc. So too, the administration of anaesthetics can be divided into stages: putting the patient to sleep, maintaining a steady state during the operation and revival after the procedure has been completed. Process control in safety-critical applications in industry should also be mentioned.

Our current research is aimed at producing a reliable and reproducible method for building controllers. Future work will be directed towards understanding the effects of causality and using structured and constructive induction to help make the control rules more compact and more readable.

ACKNOWLEDGMENTS

Jim Kehoe and Peter Horne conducted the human reaction time studies and collected much valuable data. Mark Pendrith performed many of the modifications to the flight simulator. Silicon Graphics Incorporated made the source code of the flight simulator available. This research has been supported by the Australian Research Council and the University of New South Wales.

REFERENCES

Anderson, C. W. and Miller, W. T. (1991). A set of challenging control problems. In Miller, Sutton and Werbos (Eds.), *Neural Networks for Control*, MIT Press.

Chambers, R. A. and Michie, D. (1969). Man-machine co-operation on a learning task. In R. Parslow, R. Prowse and R. Elliott-Green (Eds.), *Computer Graphics: Techniques and Applications* London: Plenum.

Donaldson, P. E. K. (1960). Error decorrelation: a technique for matching a class of functions. In *Proceedings of the Third International Conference on Medical Electronics*, (pp. 173-178).

Michie, D., Bain, M., and Hayes-Michie, J.E. (1990). Cognitive models from subcognitive skills. In M. Grimble, S. McGhee and P. Mowforth (Eds.) *Knowledge-base Systems in Industrial Control.* Peter Peregrinus.

Muggleton, S. and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Machine. Learning Conference* (pp. 339-352). Ann Arbor, Michigan: Morgan Kaufmann,.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-Machine Studies*, **27**, 221-234.

Sammut, C. and Michie, D. (1991). Controlling a 'black-box' simulation of a spacecraft. *AI Magazine*, **12**(1), 56-63.

Shapiro, A.D. (1987). *Structured Induction in Expert Systems*. Addison-Wesley.

Widrow, B. and Smith, F. W. (1964). Pattern recognising control systems. In J. T. Tou and R. H. Wilcox (Eds.), *Computer and Information Sciences* Clever Hume Press.