The Learning of Robust and Flexible Skills with Respect to System Failures in a Flight Management System

Stefan Wierda

August 2009

Master Thesis
Human-Machine Communication
Dept of Artificial Intelligence
University of Groningen, The Netherlands

Supervisor:
Prof. Niels Taatgen (University of Groningen/Carnegie Mellon University, USA)

Abstract

In this thesis, I describe an experiment and introduce an ACT-R cognitive model used to investigate the learning of robust and flexible skills. In the study, 60 students participated in an experiment solving problems in a simulation of a Flight Management System. In order to investigate robustness and flexibility, state and display malfunctions would occur during the experiment. An instruction based on a list of steps, and an instruction based on pre- and postconditions of the environment were used to manipulate the use of internal and external control states. Using mixed effect models, the experimental data and model's data are compared. The model captures the overall performance of the participants in terms of reaction times and correctness scores. The results reveal that visual input is important in skill performance, and that a focus on pre- and postconditions aid participants to cope with system malfunctions.

Acknowledgements

In total, it took me seven years to reach this point in life where I finish my Master thesis and close a chapter in my life. Before I start with the findings of my thesis, I would like to thank a few people whose help I had reaching this point.

First, I want to thank my supervisor Niels Taatgen. Without his help, funding and feedback, I would not be able to go to the United States and do this research project. Secondly, I would like to thank Carnegie Mellon University and the people I worked with for their support and feedback on my project. Special thanks go to Ion Juvina, whose advice during our weekly meetings was of great support for me.

I would also like to thanks the University of Groningen, and especially the department of Artificial Intelligence. Thanks for the support of the staff and students, without them, I would not be able to finish my Bachelor and Master degree. Special thanks to Fokie Cnossen for supervising me in writing my Bachelor thesis. I would also like to thank her for being the second reviewer of this thesis.

Next to my education, I gained a lot of experience by taking a seat in the Faculty Council, board of the Groninger Student Union, board of CoVer (the Artificial Intelligence and Computer Science student association), and so fort. I would like to thank the people who supported me in those positions and helped me getting there.

Next, I would like to thank my American friends in Pittsburgh, who supported me during my stay in the United States and who contributed to a great experience I will remember for life.

Of course, I want to thank my friends in Groningen for their friendship and support during my time as student. In special, I want to thank Ingrid, who inspired, supported and encouraged me the past three years.

Table of Contents

Introduction

June the first, 2009. An Airbus of KLM Air France disappears from the radar a few hours after its departure from Rio de Janeiro to Paris. Supposedly, the airplane is struck by lightning while flying through a storm above the Atlantic Ocean. A week later debris of the plane is found floating in the Atlantic Ocean. What went wrong that particular flight? There are many speculations about the origin of the crash. The lightning struck could have caused a short circuit in the planes systems, leading to a devastating ending. Just before the crash, the plane transmitted emergency messages about system failures and the disconnecting of the automatic pilot. One of the sensors, which were probably malfunctioning, was the speed sensor. Why did all those events eventually lead to a crash of the airplane? That is a question for the crash investigators to answer. I do know that in most cases there are procedures to handle malfunctioning sensors and systems. Whether human error or a system malfunction was the cause of the crash of Air France Flight 447, pilots need to cope with these unexpected problems.

The way pilots learn the instructions for the abovementioned procedures are in a stepwise manner. Pilots learn a number of steps they have to execute to perform a certain procedure (Taatgen, Huss, Dickison, & Anderson, 2008). Whether this method is a good way of learning procedures is disputable and we will address this claim later in this chapter.

A particular field of research in human machine communication I find interesting is that of learning and mastering procedures. One of the reasons is that errors made in procedures can lead to fatal events, such as the crash mentioned above. Therefore, the learning of procedures as a skill and the effect on system malfunctions will be the theme of this thesis.

The acquisition of cognitive skills

Over the past decades, extensive research has been done on the acquisition of cognitive skills. Traditionally, research in cognitive skills focuses on problem solving and decision making in mathematical problems and games such as the Tower of Hanoi and chess. In his review, VanLehn makes a comparison between motor skill learning and cognitive skill learning, identifying three stages of learning (VanLehn, 1996). The early, intermediate and late phases, borrowed from research on the acquisition of motor skills (Fitts, 1964), ideally describe the process of learning a skill. In the first phase general knowledge is learned, whereas more specialized knowledge is missing. In the intermediate phase more specialized knowledge is acquired, but there are still flaws in performance due to missing and incorrect information. Finally, in the late phase problems can be solved without conceptual errors. Errors that occur in this phase are due to slips and mistakes.

Classic models of skill acquisition

Traditional computational models that try to capture skill acquisition usually start out in the first phase. There already is general information about the task and the domain present. The models then specialize that knowledge and eventually reach the last phase. Usually such models store information of partially completed problems to enable faster problem solving by simply retrieving the solution of a specific part of a problem. This stored information

can be in the form of rules or facts. Examples of these kinds of mechanisms are Logan's instance based learning (Logan, 1988), Newell and Rosenbloom's chunking mechanism (1981) implemented in Soar (Laird, Rosenbloom, & Newell, 1986), and ACT-R's production compilation system (Taatgen & Lee, 2003).
Laird, Rosenbloom and Newell (1986) present a model of learning to play the eight puzzle. The eight puzzle game is played as follows. The playing board consists out of a 3 x 3 grid containing in total nine squares. There are eight blocks, each block containing one of the numbers one to eight. One of the squares is empty; the other squares are randomly occupied by one of the eight blocks. Possible actions are moving one of the blocks towards the empty square (left, right, up or down). The game is finished whenever the blocks are in placed in an ascending, clockwise order; with block one being at the left corner of the grid (Figure 1). The probably more familiar version of this game is played with a scrambled picture. The objective is then to unscramble the picture by moving the blocks.

Initial state

4	2	3
8	7	1
	6	5

Goal state

1	2	3
8		4
7	6	5

Figure 1: An example of the Eight-Puzzle game

To determine what action to take, the Soar model perceives the state of the outside world, and consults its production system what to do. On its turn, the production system tries to match rules that apply to the current situation, and selects the possible moves. A simplified example of a production rule is "***if*** *my goal is to move number one to the upper left corner* ***and*** *the square left to number one is free* ***then*** *move number one to the left"*. The sorts of rules in the production system of Soar are divided in *problem spaces*. In the case of the game described here, one of the problem spaces contains the rules to move each block to its desired state. However, an *impasse* occurs in this problem state whenever there are multiple rules that can fire. To solve the impasse, the system will start searching in another problem state, which could provide an answer for the impasse. A sub-goal to solve this impasse is then created. If an answer is found, the solution is added to the first problem state. Whenever the impasse is encountered in the future, the production system will now find an answer in the current problem state without the need to create a subgoal. This process is called chunking in Soar and is one of the ways in which classic models specialize and acquire expert skill level. If in the second problem state an impasse would be encountered, another problem state would be consulted to find the answer, thus creating a subgoal hierarchy. Although Soar can

handle unfamiliar problems through solving an impasse, this mechanism only works whenever all the knowledge needed to solve the problem is available in the problem spaces. Whenever general knowledge needed to solve the problem is missing, Soar gets stuck at the problem. In general, Soar's chunking mechanism works not unlike production compilation in ACT-R or as instance learning of Logan. Thus, the problem that Soar faces is also faced by other cognitive architectures as ACT-R.
A drawback of the abovementioned models and their underlying theories is that it is hard, or in some cases impossible, to generalize the learned knowledge and apply it in new and unseen situations, especially when general information is missing. Thus, although familiar problems can be solved much faster than unfamiliar problems, the learned skills are not robust and flexible enough to employ them in unseen situations.

Embodied cognition
On the other side of the spectrum is the research field of *embodied cognition*[1]. "Embodied" means that cognition is rooted in its environment. Researchers such as Brooks focus on using the environment directly to guide future actions, instead of reasoning about it internally. In his 1991 conference paper "Intelligence Without Reason", Brooks (1991) attacks the traditional Artificial Intelligence approach. According to Brooks, the traditional approach focuses too much on top down reasoning leaving a gap between perception and action. He argues that it is necessary for a robot to be situated and embodied in the environment for intelligence to emerge. Although Brooks paper focuses on robots, it is in some sense applicable to cognitive architectures like those mentioned above. According to Brooks, modeling the world internally and reasoning about it can be computationally challenging. Using the perceptional input to reason about the world instead of an objective world model would be much more efficient. This might as well be true for cognitive models (and thus for humans). To quote Brooks, "The world is its own best model".
A working example of Brooks approach is the *subsumption architecture* (Brooks, 1986). The key aspect of the subsumption architecture is that it is a bottom up architecture, driven by the environment. The lower levels of the architecture imposes constraints through feedback from it sensors to the higher, goal-orientated levels. For example, if the lower level layer detects objects and is programmed to avoid those, it overrides the high level layer in charge of path planning and thus avoids collision. Perception and action are directly coupled without an overall control structure. This enabled Brooks to put the robot in an unknown environment able to navigate and avoid unexpected obstacles. More closely related to the topic of skill learning is the connectionist model of Botvinick and Plaut (2004). Using a *recurrent connectionist network* approach, they build a model that is capable of learning task sequences. The architecture of their model is shown in Figure 2.

[1] The term embedded cognition is also used by some researchers to refer to embodied cognition

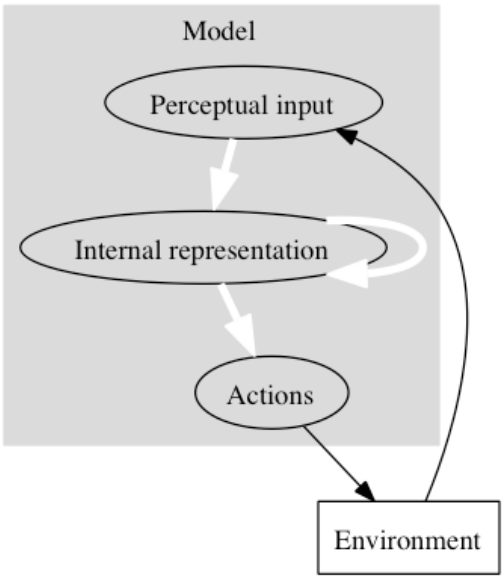


Figure 2. The architecture of the model presented by M. Botvinick and D. C. Plaut(2004). The circles in the grey box represent layers of units in the recurrent network. The perceptual input layer contains units for fixated objects as well as for held objects.

The sequence task in the study of Botvinick and Plaut was that of making coffee. For the input layer, they identified features representing the state of the environment, such as cup, lid, brown-liquid, and so on. Every identified feature has its own unit in the input layer. In the output layer the units corresponds to actions, such as pick-up, put-down, pour and so on. The internal representation part consisted of 50 hidden units, representing the internal context state of the model.
In the training phase of the model the correct sequence of making coffee was represented to the model, regardless of the model's output. The model then adjusted its weights to learn the patterns and sequence of the task. When testing the model no feedback about the correct sequence was given. In their simulations, Botvinick and Plaut showed that their model exhibits several features of normal and impaired routine sequential behavior. Through their model, they show that organized action can occur without explicit goals. However, they acknowledge that in some circumstances human action does involve explicit goals, something not captured by their model. In addition, the actions and perceptual input are predefined. The model cannot learn new actions or perceptual input.
Taatgen et al (2008) combined the strength of both the classical and embodied cognition approach to skill learning. They were able to build a model that could learn a skill robust and flexible enough to cope with unexpected and partial completed problems. I will discuss this approach in the next section.

Robust and flexible skill learning

Although it is current practice to teach pilots procedures as lists of steps, Taatgen et al (2008) found evidence that there is a better way to learn procedures and cognitive skills than the learning of a list of steps. In their research, students had to perform certain procedures in a computer simulation of a Boeing 777 Flight Management System (henceforth called FMS). Taatgen et al show that *context learning*, where the focus is the pre-

and postconditions of actions and the environment, is far more effective than just plain *list learning*, where the pilot just has to study a list of actions for each procedure. The principle of context learning is based on the *minimum control principle* (Taatgen, 2007), which argues that humans tend to minimize the amount of mental *control states* by deriving the state from perceptual input. In this context, control states can be seen as a measure of top-down control. In other words, control states keep track of the progress of a task, without the necessity of a feedback loop from the environment. In context learning, the pre- and postconditions of actions are specified and the actions are embedded in the environment. This way the current control state can be derived from the environment. According to Taatgen et al (2008) this explains why learning the context procedure is more effective than learning a list of steps, it is just easier and more effective to infer control states from the environment rather than keeping track of control states internally. In the experiment, students had to solve problems concerning the direct-to and remove-discontinuity procedures in the FMS simulation. In some of the cases, just one or both procedures could be applied literally, in other cases they had to solve problems caused by their copilot using adaptations of the learned procedures, thus requiring flexibility and robustness. The participants in the context learning group had higher performance ratings on and were faster in solving the problems than the participants in the list learning group. In the next chapter, I will further discuss the abovementioned procedures and discuss the task participants had to do in more detail.

Research on procedures and human error

In the field of human factors, extensive research has been done to human error and the impact of it on society (McIlvaine, 2006)(Dhillon & Lui, 2006). Most research concerns teamwork, stress, environments, and interfaces. Though research has been done on learning of procedures, most of it is focused on educational programs and not on the underlying cognitive mechanism for learning a particular procedure. Gaba (2001) did research in skill learning and human error in the field of medicine. He argues that a cookbook approach on learning medical procedures is not effective and results in too rigid skills. With the term cookbook, he refers to what Taatgen et al dubbed list learning.

Recently, Fennell et al (2006) published a paper on the impact of recall steps on errors made in a FMS task. They did research on learning FMS procedures, identifying two types of steps in the sequence of instructions: recognition and recall steps. A *recognition step* in the sequence is a step where the action is cued by salient labels or prompts. *Recall steps* on the other hand are not cued by the display and therefore a potential source of error. Pilots unfamiliar with the FMS learned several procedures and the *access errors* (hitting a wrong button or accessing a wrong routine) they made were analyzed. It turned out that there is a probability of .74 of making an access error, whenever the procedure has two recall steps. A probability of .13 was found for procedures with only one recall step. Whenever no recall steps were present, the probability for an access error was only .06. Their results show that perhaps environmental cues are important for recalling a step of the instructions. Possibly, it is easier for people to rely on environmental cues rather than internal cues. Note that the instructions in the study of Fennell et al were learned by a special method suited for their research. Pilots had to

reformulate the steps to promote remembering the step (e.g. as reported in their paper “Enter the winds” was reformulated to something like “Because winds are associated with waypoints, and waypoints are on the LEGS page and are part of the data of the route, enter the winds using the LEGS page, RTE Data prompt”).
Regarding the steps in a procedure, Cox and Young (2000) speculated that *device-specific* and *task-specific* steps rely on different kind of knowledge. The main difference between the two steps is that device-specific steps do not directly contribute to the goal of the task, whereas the task-specific steps do. Aments, Blandford and Cox (2009) hypothesized that different cognitive processes underlie these two types of steps. They argue that device-specific steps rely more on external cues, whereas internal cues are more important for task-specific steps. This hypothesis is derived from the Activation-based Goal Memory model (Altmann & Trafton, 2002). This model states that goals are associated with activation levels and that they can be triggered by means of retrieval cues. Such cues are for example associative internal links (as a sequence of successive steps) or an external cue (such as a display item). The device-specific tasks are required by the device and do not form a natural part of the task, thus its associative links are weaker than those of the task-specific tasks. Aments et al therefore reason that external cues are more important for the device, than for the task-specific steps. In the research of Aments et al participants had to play the spy game. The objective of the game was to fly a plane to a certain destination to deliver a secret message. To accomplish this objective the participants had to follow a procedure specifying what to do. In the procedure, there were 11 device-specific steps and 17 task-specific steps. In a first experiment, they evaluated the difference between the steps by looking at eye-tracker data. It turned out that when an error is not made, there is a significant difference between the device-specific and task-specific steps. In a second experiment, they removed semi-randomly the visual cues such as a button, an input field or another item. The assumption here was that removing an item would lead to a reduced external cuing for the step involved with that item. However, the location of the item was still functional so that the task could still be completed. In the study there were significantly more errors made on device-specific steps than on task-specific step. There were also more errors made on the steps where the item was removed from the interface. This result supports the assumption of reduced cueing. However, no significant interaction was found between the two device types and the visibility of an item. Ament et al argue that it was due to the power of the statistical test used (too few participants) or due to a floor effect in error ratings, rather than that, the hypothesis is false. Two other reasons could be that the external cuing is more complex and that therefore the experimental manipulation did not work as expected or there could be a confounding factor; the hidden step could be a predictor for the next correct step. I think that something else could be going on, and that both steps in their experiment relied strongly on the external cues. Prior to the task, people had to learn the task. Feedback and instructions were given in the training phase and whenever each participant completed the task two times without any errors, the training would end. What could have happened is that there was enough time to learn the instruction and associate the steps in the procedure with the cues in the environment. As Taatgen (2007) argued, whenever people can divert control to the environment, they do it. Since I had no access to the

instructions used in the experiment, it is hard to say something about the learning method. Nevertheless, I speculate that perhaps the participants learned the procedure by context and not necessary by internal step (in a list learning way). This way one would expect the task steps to be retrieved by external cues, rather then by internal cues.

ACT-R

To show that the minimal control principle was indeed a possible explanation for the improved performance Taatgen et al (2008) built a computational model in the cognitive architecture ACT-R. The ACT-R architecture (Anderson et al., 2004) is a cognitive architecture designed to model cognitive tasks. The purpose of ACT-R is to provide a simple theory for complex cognition (Anderson, 1996). At first most of the tasks modeled in ACT-R were related to learning and memory processes. Later on, researchers started to model complex tasks in ACT-R and eventually a perceptual- motor theory was adapted by ACT-R (Byrne & Anderson, 2001) and implemented in the newer versions of ACT-R.

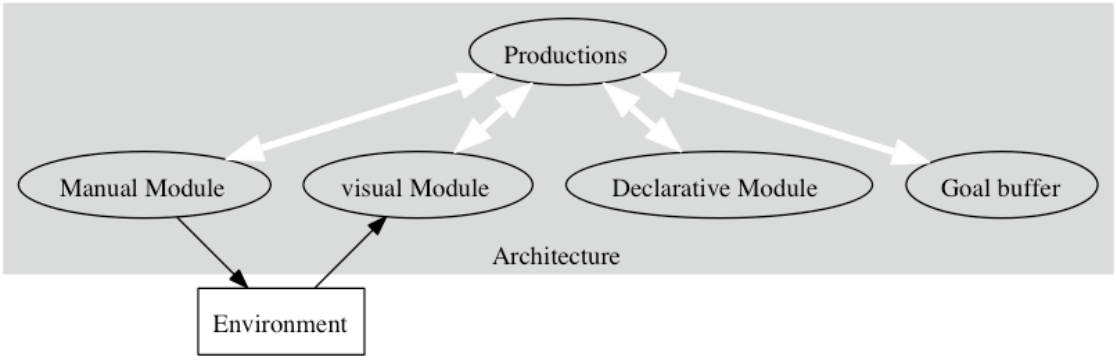


Figure 3. A simplified diagram of the ACT-R architecture

The ACT-R architecture (Figure 3) consists of several modules of which the production system is the core. This production system communicates with other modules through buffers implementing cognitive functions in ACT-R. The main modules currently implemented in ACT-R are the Intentional Module, Declarative Module, Visual Model, Motor (Manual) Module, Aural Module and Vocal Module. The production system has access to the chunks (pieces of information) in the modules' buffers. The chunks are matched, selected and executed by the production system, letting all modules communicate together through the production system. These modules work parallel to each other, whereas a module itself is a serial process.
ACT-R has been used to model several real world tasks including driving, aircraft maneuvering and recently the FMS task. I will discuss the model built by Taatgen et al later in this thesis.

Present study

From their research, Taatgen et al (2008) found evidence that both internal and external control states keep track of progress, and that context instructions are probably closely related to our internal representation of the task knowledge. As mentioned above, they came to these conclusions by building a model that incorporates both the classical and embedded cognition approach to skill learning. To investigate the distinction of internal and

external control further I propose an experiment based on the same task of Taatgen et al, but with a different experimental design. The same simulation will also be used in this research. In this research I will introduce system malfunctions, to see how robust both learning methods are coping, not only with partially or incorrectly completed problems, but also with system malfunctions. I hypothesize that because system malfunctions are unexpected, that this will test the robustness and flexibility of both methods more, revealing more of their underlying mechanisms.

State malfunctions and display malfunctions

For purpose of this study, I make a distinction between two types of malfunction. Whenever a *state malfunction* occurs, the system performs another action than the user intended. For example, the user wants to go to the LEGS page and presses LEGS. Now if a state malfunction occurs, the system goes to the RTE page instead of going to the LEGS page. Now, the user expects the system to be at the LEGS page, but the system really is at the RTE page. Hence, the users expected state of the system differs from the real state of the system. Thus, a state malfunction occurred.
A *display malfunction* is the second malfunction type. In this case the interface of the FMS gets partly scrambled.
I expect that a state malfunction is detected and recovered from faster by the context learning group than by the list learning group. Rather than being stuck in a list of steps, the context group will infer in what state or on what page the system is, and thus correct the malfunction more effectively.
For display malfunctions, it is not clear what can happen. On the one hand, the list group could perform better than the context group because they rely less on the environment and keep track through their internal learned steps. On the other hand, the context group could infer the current state of the system more easily by using the information they do get.
For the conditions where no system malfunctions occur, we expect to find the results as presented in the research of Taatgen et al. (2008) (i.e. the context group will perform better than the list group).

Research question

The research question in this thesis can be formulated as follows: Is the context learning method more flexible and robust to system failures than the list learning method?
The following hypotheses can be formulated with respect to this research question:

- I expect that the context group will perform better in resolving problems and executing procedures than the list group in case of no malfunctions and state malfunctions.
- In case of state malfunctions, there will be a greater difference in performance between the context and the list group in favor of the context group.
- In case of display malfunctions, it is not clear which group will perform relatively or absolutely better. The list group could perform even better than the context group, because they do not rely on the display as much as the context group does. On the other hand, the context group could perform better than the list group, because they probably have a better understanding of the system.

The Flight Management System experiment

To investigate the abovementioned research question, I conducted an experiment involving the FMS task used in the research of Taatgen et al (2008). As briefly mentioned above, the FMS is the heart of automation of modern airplanes (Figure 4). It is used in both military and commercial airplanes. This device controls the behavior of the plane and can take care of the whole flight, except for the taking off part. The pilot programs the FMS and supplies it with information about the route of the plane, load, passenger numbers, and so on. In turn, the FMS takes care of the flight part. Therefore, it is important that the pilot knows how to use this system; it is a crucial part of his job. In the research of Taatgen et al, the subject of research was two procedures part of *lateral navigation*: planning and modifying routes. These procedures are also used in the experiment described later in this thesis.

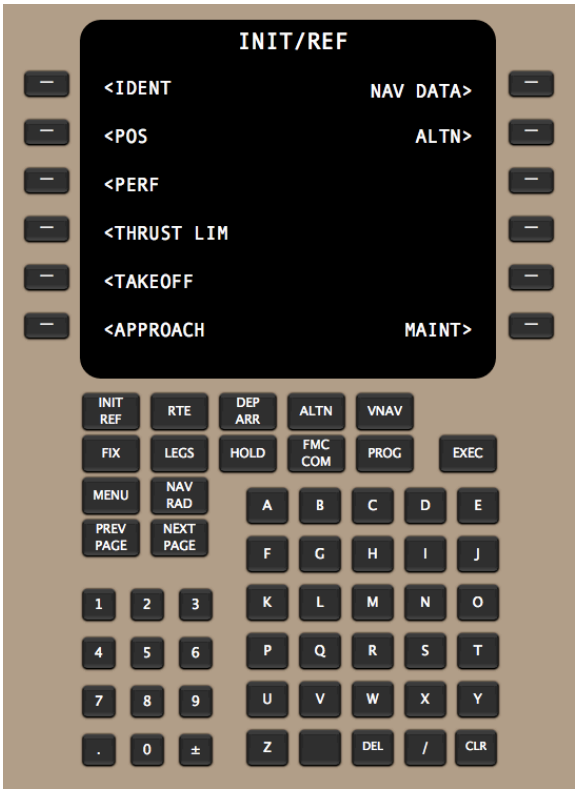


Figure 4. The layout of the Flight Management System of a Boeing 777. The buttons on the top left (with the '-' sign) are the L-keys. At the right, the R-keys can be found. If on a page the scratchpad is available, it will appear on the screen.

A route in aviation consists of a list of waypoints the plane has to follow from source to destination. These waypoints are often points on the map, but can also be radio beacons. To redirect the airplane the pilot has to access the route through the FMS and has to change waypoints in that route. Two specific procedures to change the route are the *direct-to* and *remove-discontinuity* procedures. The direct-to procedure is used whenever Air Traffic Control (ATC) gives the pilots the instructions to directly continue to a specific waypoint. Whenever the waypoint is not part of the current route, a discontinuity in the route will appear. That is, there is a missing link in the

route. To solve the discontinuity, the pilot has to execute the remove-discontinuity procedure. Both procedures are specified in Table 1.

Table 1. The direct-to and remove-discontinuity procedures

Procedure	Instructions
direct-to	1. Press the LEGS key 2. Enter the desired waypoint in the scratchpad 3. Push the 1L key 4. If the word discontinuity appears on the screen, follow the procedure to remove discontinuities 5. Verify the route on the Navigational Display 6. Press EXEC
remove-discontinuity	1. Press the LEGS key 2. Press the line select key after the discontinuity 3. Press the line with the THEN prompt

The FMS task

In the FMS task, participants have to follow instructions from Air Traffic Control and change the route of the airplane in a computer simulation of the FMS. In order to complete the task, they have to use two procedures used in lateral navigation, the *direct-to* and *remove-discontinuity* procedures. Figure 5 shows the interface of the simulator. ATC instructions for rerouting the airplane pop up at the beginning of each trial at the right corner of the screen. An example of such an instruction is "Flight 123 … proceed direct to BEX, continuing on to LOAMY" with the current route being e.g. LOAMY-KEOKK-BDF-BENKY. In order to solve this problem, participants have to first follow the direct-to procedure (pressing "LEGS", typing "BEX", pressing L1). While executing this procedure, a discontinuity pops up on the screen. Participants then have to follow the remove-discontinuity procedure (pressing "LEGS", pressing L3, pressing L2) and then finishing the direct-to procedure (looking at NAV, pressing EXEC). The new route of the plane now is BEX-LOAMY-KEOKK-BDF-BENKY. The discontinuity in this solution appears because BEX was not on the route before. If the instruction were something like "Flight 123 … proceed direct to BDF", then the direct-to procedure would be sufficient to

complete the task.

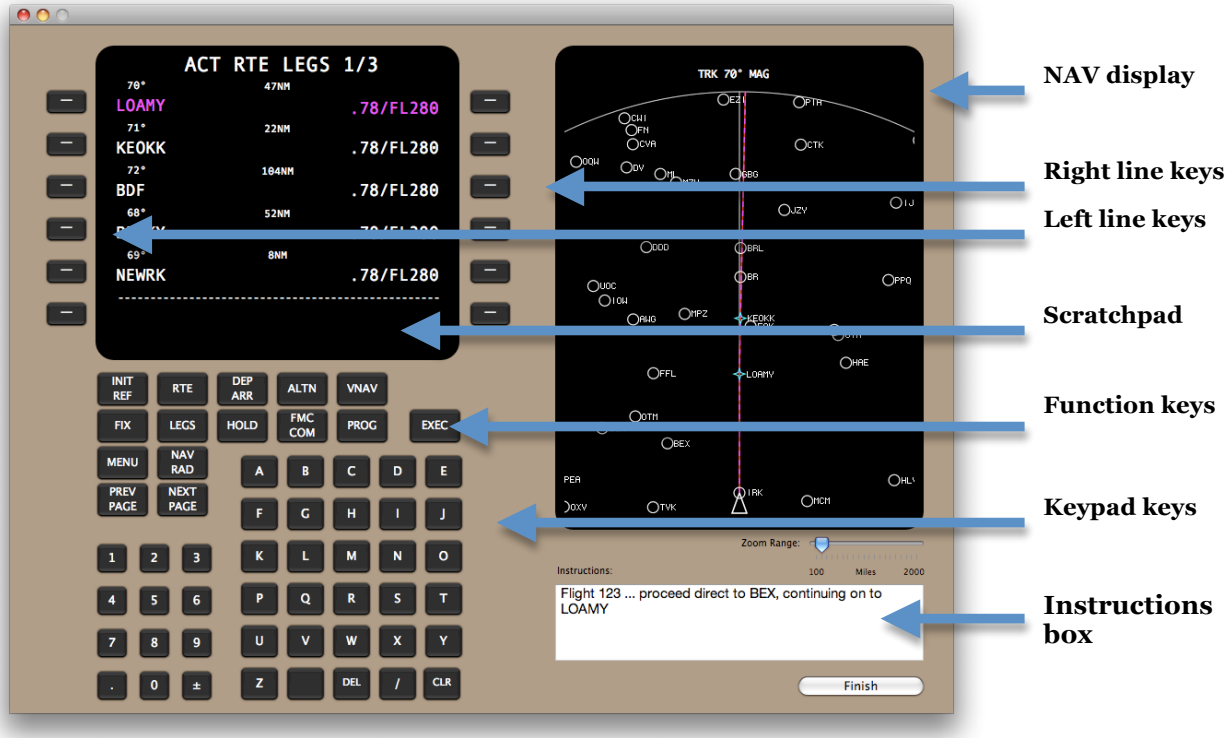


Figure 5. The Flight Management System computer simulation

Problems

In the FMS task, the participants can encounter three types of problems, categorized by their difficulty. *Easy problems* are problems where the participants only have to apply the direct-to procedure. In *medium problems* a discontinuity occurs whenever a participant applies the direct-to procedure. Thus in these problems the *remove-discontinuity* also has to be applied. Finally, there are the *hard problems*. Problems can be hard for several reasons. First, one of the waypoints referred to in the problem could not be found at the first LEGS page. Participants have to use the function key 'NEXT PAGE' and 'PREV PAGE', which were not mentioned in the procedure, in order to solve the problem. Secondly, the waypoint to be modified could be one later in the flight plan. This was also not covered by the procedures, requiring some generalization. These are the easier problems in its type. Slightly harder problems are the problems where a co-pilot already executed a part of the procedure. It is up to the participant to detect it and to finish the procedure. In these problems, the co-pilot sometimes had made a mistake, which makes the problems harder. Taatgen et al have a distinct category for these problems, but since the focus of this study is not on problem difficulty, I do not make this distinction.

Malfunctions

As mentioned above, there are two types of malfunctions that can occur in the FMS task. In one condition, display malfunctions are encountered. The display malfunctions consist out of scrambled lines on the screen of the FMS simulation. In the other condition state malfunctions can occur. If a state malfunction occurs, the FMS simulation will do another action than what the participants would expect. For example pressing the LEGS key will have the

effect of going to the RTE page or pressing a line key will copy the wrong content in the scratchpad. In some cases these malfunctions occur only the first time when one of the keys is pressed, otherwise if would be impossible in some cases to solve the problems.

Method

Participants

A total of 60 students from Carnegie Mellon University and the University of Pittsburgh participated in the experiment. They all received a payment for their participation. The context learning condition had 31 participants, whereas the list learning condition had 29 participants.

Procedure

First, the participants had to study the general background information (appendix I) of the FMS task. After they were finished, they had to return the background information and the interface training started. The purpose of this part was to get the participants familiar with the interface of the FMS. This training consisted of getting instruction such as “Press the L1 line select key” and “Copy BOND into the scratch pad and then press Finish”. The background information and interface training was the same for both groups. Whenever people were finished with the training, they had to study the instructions for the direct-to and remove-discontinuity procedures. These instructions were adapted for each of the two conditions (appendix II and appendix III). After they were finished studying the instructions, the participants had to hand them back to the experimenter.
The experiment consisted of four blocks (an overview of the blocks can be found in Table 2). The first block was the training block and consisted of six problems. The first three problems of this block were easy, the remaining three were medium problems. This block was also used to exclude participants from the analysis based on their performance. The criterion to be excluded from the analysis was a score of zero or one correct. The second block consisted of nine easy and nine medium problems. For each problem difficulty, three problems were given in the display malfunction condition, three problems were given in de state malfunction condition, and three problems served as control condition in which no malfunctions occurred. The third block consisted of eighteen problems. This time all the problems were hard problems. The problems in this block were also randomly ordered. In this block, nine problems were in the display malfunction condition, nine in the state malfunction condition and nine problems served as control. The fourth block also consisted of eighteen problems. This time they were all hard problems. No malfunction conditions were presented in this block. Where the other blocks did not contain co-pilot problems, this block contained twelve co-pilot problems. In half of the co-pilot problems, the co-pilot would have made a mistake. The ordering in this block was also random.
To prevent biasing the experiment, no help was given to the participants throughout the experiment. The experiment, including studying the background information and instructions, lasted about one hour.
For each participant the *reaction times* and *correctness score* were recorded.

Table 2. The design of the experiment

	Block 1 Easy and medium problems	Block 2 Easy and medium problems	Block 3 Hard problems (no co-pilot)	Block 4 Hard problems (with co-pilot)
Context Instructions	3 problems	6 control problems 6 display problems 6 state problems	9 control problems 9 display problems 9 state problems	18 problems
List Instructions	3 problems	6 control problems 6 display problems 6 state problems	9 control problems 9 display problems 9 state problems	18 problems

Results

Based on the abovementioned criterion, data of 16 participants were eliminated from the experiment. Nine of the eliminated participants were part of the context group; seven of them were part of the list group. Data of 44 participants remained with 22 participants in each condition. One problem with an average score of 0.00 was also excluded from the data. The results are displayed in Figure 6 and Figure 7.

Examining the correctness score displayed in Figure 6, it seems that the context group performs better overall. Looking for interactions in the graph, it seems that the list group performs worse relative to the control problems than the context group.

Figure 7 of the reaction times shows some interesting effects. Overall, the reaction times of the context group lies under those of the list group. It looks like the reaction time of the context group decreases in case of display malfunctions in the third block. Furthermore, the reaction times of the list group in case of state malfunctions looks higher than those of the context group relative to the control problems.

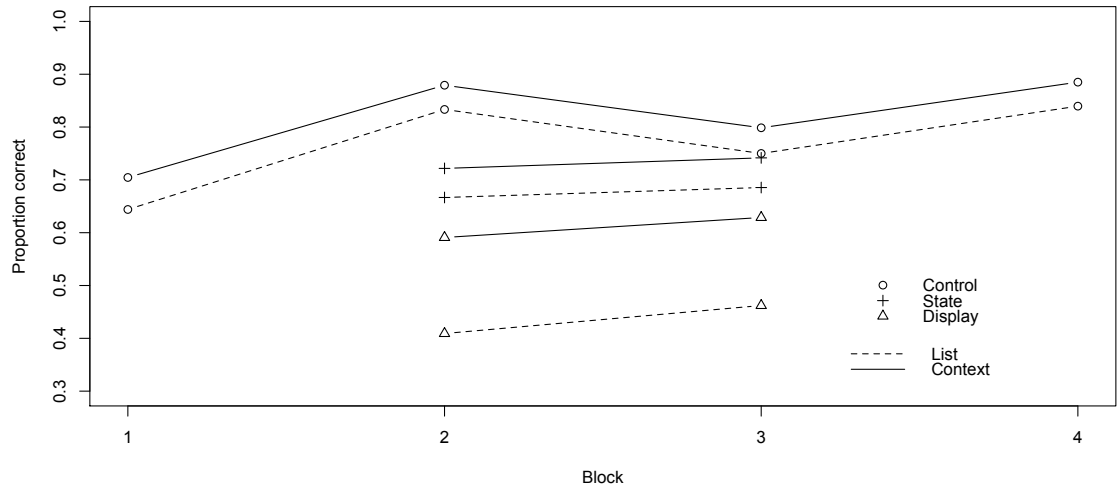


Figure 6. The results of the correctness score plotted per block.

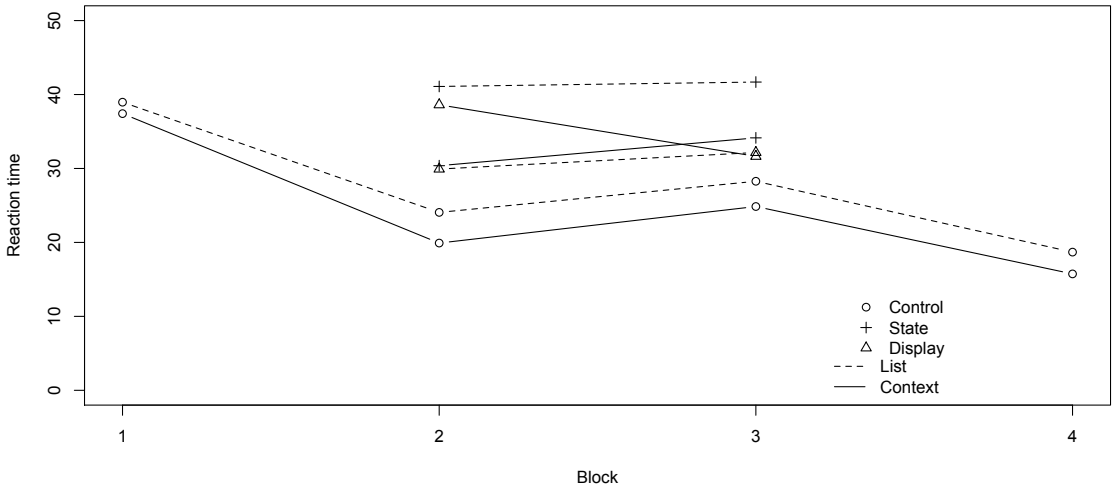


Figure 7. The results of the reaction time plotted per block. Reaction time is given in seconds.

LME-analysis for the correctness score

In order to investigate the effects in the data I fitted a mixed-effects model. The dependant variable was the *correctness* score. The fixed factors were *condition*, *malfunction type* and the square root of *sequence* (learning effect) with an interaction between malfunction type and condition. The *trial* and *subject* were taken as random factors in the LME analysis. With trial is meant the specific problem. Each problem causes variance and the problems are drawn from a bigger population of problems. Hence, the factor trial is treated as a random factor. The trials were randomly ordered within each block, thus the learning sequence is not caught in its entirely by the variable trials. Because the correctness score is a binary variable, a binomial logit (log odds) model was used. Because of the use of a logit model, the estimates of the models parameters are presented in log odds. To convert log odds back to correctness scores, the following formula is used: $P(s) = 1/(1+e^{-s})$. This means

that a total log odd of 0 is equal to 0.5 in terms of correctness scores. A negative log odd estimate lowers the changes below 0.5.
The results of the analysis can be found in Table 3. In order to see if the interaction between the malfunctions and instructions explain a significant amount of variance, I compared the model with a LME model with no interaction effects (i.e. *condition*, *malfunction type*, and *sequence* as fixed factors. The model with the interaction effect between malfunction type and condition turned out to explain significantly more variance ($\chi^2(1) = 6.77$, $p = 0.034$).
Because the focus lies on the malfunction type and condition, we left the problem type out of the analysis.

Table 3. Results of the LME analysis

	Estimate	Std. error	t-Value	p-Value
Intercept	0.328	0.409	0.804	0.422
Display malfunction	-1.422	0.196	-7.263	0.000
State malfunction	-0.735	0.212	-3.475	0.001
List instructions	-0.369	0.300	-1.230	0.219
Learning sequence	0.353	0.063	5.597	0.000
Display malfunction x List instructions	-0.584	0.249	-2.346	0.019
State malfunction x List instructions	0.099	0.265	0.374	0.709

Display malfunction
Whenever a display malfunction occurs there is an average decrease in the log odds of -1.422.
The effect of display malfunctions is significant ($p < 0.001$).

State malfunction
A state malfunction also significantly decreases the log odds with -0.735 ($p < 0.001$).

List instructions
While this effect is not significant, in earlier studies this main effect is significant (Taatgen, Huss, Dickison, & Anderson, 2008). The estimate is -0.369, indicating that the list instructions score is on average lower than the context instructions.

Learning sequence
The learning effect is highly significant ($p < 0.001$) and increases with a 0.353 in terms of log odds. Note that this estimation is based on the square root of the sequence to capture the learning effect.

Display malfunction x List instructions
When a display malfunction occurs in the list instructions, the log odd drops with an average of 0.584 ($p = 0.019$).

State malfunction x List instructions
No significant difference is found between the list and context instructions condition for the state malfunction.

LME-analysis for the reaction time

The reaction time results were also fitted with a mixed-effects model. The same factors were used as in the analysis for the correctness score as mentioned above. The model with fixed factors *malfunction type*, *condition*, and *sequence* with an interaction effect between *malfunction type* and *condition* was compared with the same model without the interaction effects using ANOVA. In this case, the model with the interaction effect between *malfunction type* and *condition* turned out not to explain significant more variance ($\chi^2(1) = 2.6$, $p = 0.39$). This time the log of the reaction time in seconds was the depended variable. The log function was used to transform the results towards a normal distribution (examining the plot of the reaction time gave a skewed curve, the log function showed the curve of a normal distribution). The results are displayed in Table 4. Note that only reaction times of correct trials are used.

Table 4. Results of the LME analysis

	Estimate	Std. error	t-Value	p-Value
Intercept	10.739	0.095	113.42	0.000
Display malfunction	0.290	0.033	8.78	0.000
State malfunction	0.424	0.031	13.78	0.000
List instructions	0.169	0.059	2.87	0.016
Learning sequence	-0.172	0.014	-12.04	0.000

Display malfunction

The effect of the display malfunctions is significant ($p<0.001$). Whenever a display malfunction occurs the reaction time is longer on average.

State malfunction

For state malfunctions the effect on the reaction time is also significant ($p<0.001$). Whenever a state malfunction occurs, the reaction is longer than the reaction time on display malfunctions.

List instructions

The main effect of instructions is significant ($p = 0.016$). If the condition is list instruction, reaction time increases.

Learning sequence

There is a significant effect of the learning sequence ($p<0.001$). Participants get faster as they do more trials.

Discussion

The research question of this thesis is "Is the context learning method more flexible and robust to system malfunctions than the list learning method?" (Ament, Blandford, & Cox, 2009).

One of the hypotheses we formulated with respect to this research question is that when no malfunctions occur, the context group will perform better. Taatgen et al (2008) found evidence in favor of this hypothesis in their research, as discussed in the introduction of this thesis. The results of the current study support this hypothesis. Although there is no main effect of the instruction condition for the correctness scores, the reaction times of the

context instruction group are significantly lower than those of the list instruction group. The fact that the instruction condition is not significant can be due to a few reasons. First, due to the complexity of the task and the individual differences of the participants, the effect is likely to be masked. Secondly, the interaction effect of the malfunctions and instruction within the mixed effect model explains part of the difference between the list instruction group and context instruction group, absorbing the main effect of instructions. Therefore, the main effect is likely to be moderated by the interaction between instructions and malfunctions. As stated above, the main effect of instructions is significant in the reaction times. The participants in the context instruction group perform the task faster than the participants in the list instruction group. The power of this effect in the mixed effect model is higher, for there are no interactions in this model. This is evidence that shows that the performance of the context group is indeed higher than that of the list instruction group.

The main effects of the state and display malfunctions in correctness scores and reaction times show that the manipulation of the malfunctions did work. Examining the plots (Figure 6 and Figure 7) and results shows evidence that the performance of the context instructions group is slightly better than the performance of the list instruction group. The context instruction group performed better in both the control and malfunction conditions than the list instructions group. The decrease of performance on problems with display malfunctions makes sense because it is probably harder to infer the current state of the external world.

The second hypothesis is that in case of state malfunctions, the context instruction group will perform better than the list instruction group. In contrast to display malfunctions, the drop in performance in state errors is less obvious, especially in the case of the context group. If this group solely relies on the visual input, the participants in the context learning condition should have scored as good as on the control problems. A cause for the drop of performance could be that the context group too sometimes relies on some internal control state, instead of on the visual input.

The interaction between state malfunction and instructions is not found significant in correctness scores, maybe due to the lack of power due to the occurrence in only two of the four blocks. However, it could be that there really is no interaction between the state and the instructions. What could have happened is that the instruction group learned to exert control to the environment while practicing enabling them to handle the state errors. Another explanation could be that the state errors were just to simple and that the necessity to perform more actions decreased correctness scores and increased reaction times.

No hypothesis was formulated on display malfunctions and instructions. As stated in the introduction, we speculated that either the context learning group or the list instruction group would perform better on the display malfunctions. The effect of this interaction is found significant in favor of the context instruction group. This means that there is evidence that the context instructions group can handle display malfunction better on average than the list instructions group. A reason for this could be that due to the allocation of control to the environment, the context group can derive the state of the system with more success when parts of the information that is available. The information that is visible guides the actions of the participants in the context

group, whereas the participants in the list learning group rely on their learning list of steps. It could also be that due to the formulation of the instructions of the context group, this group has a better understanding of the system. Therefore, they expect in what state the system is and how to react when a display malfunction occurs. In other words the learned pre- and postconditions have aided the context learning group in gaining a better understanding of the system, predicting what kind of information should be in the scrambled part of the display.
Moreover, this interaction between display malfunctions and instruction reveals evidence that visual cues are important in handling display malfunction, and that even a bit of visual information is enough to cope with the malfunction. The learned pre- and postconditions could also have aided the context learning group in gaining a better understanding of the system, perhaps predicting what kind of information should be in the scrambled part of the display.

A model of robust and flexible skill learning

The model used in this study is based on the model as described by Taatgen et al (2008). Illustrated in Figure 8 is the model of Taatgen et al. Figure 9 is an illustration of the model used in this study. Taatgen et al state that most of the ACT-R models are based on production rules, but that this is not a plausible account for acquiring skills. In their paper, they argue that a more plausible explanation is the storing of instructions in the declarative memory. General production rules are then defined which retrieve, interpret and carry out these instructions. To decide in which order to fire the rules, the easiest solution would be a list of steps. Each step then identifies the next step to be executed, remembering some internal control state. Learning instructions in a list manner could be compared to this approach. The second approach is defining environmental preconditions and postconditions for each step to be taken. This would mean that the environment triggers the next step, and this approach could be compared to the context learning method.
As mentioned above the model is based on the model of Taatgen et al and thus is very similar to their model. In the next paragraphs, the differences in the models will be pointed out as I continue describing the model.

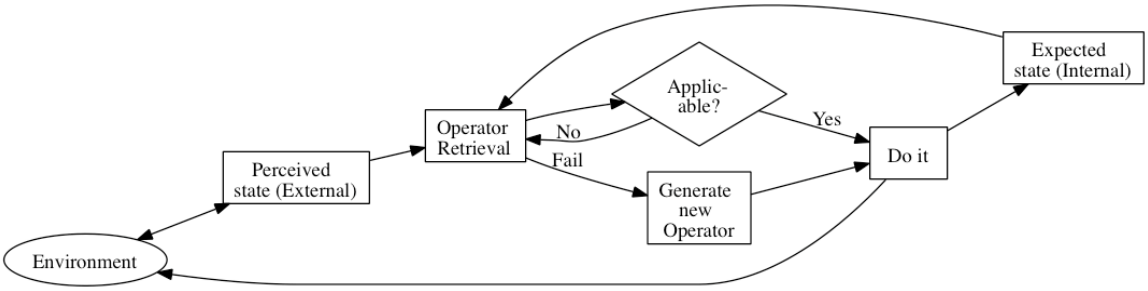


Figure 8. Outline of the model as published in the paper of Taatgen et al (2008)

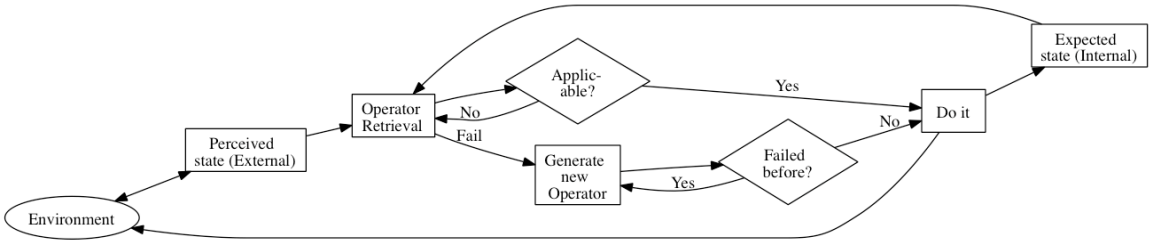


Figure 9. Outline of the model used in this study.

Representation of the external world and acting in it

To keep the model fast and the interpretation simple, I used an abstraction of the external world. Two words are presented to the visual system of the model, representing the state of the world. Although this abstraction seems to be too abstract, Taatgen et al showed in their model that it is good enough to make predictions and reason about the learning of procedures. The model acts

in the world by typing keys on a keyboard. This also is rather different than in the experiment, but this is accounted for since in both instruction conditions this difference occurs. Moreover, the participants in the experiment had a practice phase in which they learned to find and use the buttons of the interface. For both methods, this practice phase was the same, assuming that in the experimental phase the participants know where to find what button. If this assumption is correct, using a keyboard or using the mouse to control the FMS should not differentiate between the two instruction conditions.
Within the model, the state of the world is represented in the goal module. This module keeps track of the current goal, task and states. In the goal module, two kinds of states are represented. An internal state based on expectations, and an external state based on the visual input.
An example of this representation is “PAGE INIT”. These words indicate that the system currently is at the initial page. Whenever the “LEGS” button is pressed, the system would either show “PAGE DEST” or “PAGE LEGS”. Both states indicate that the system is currently at the LEGS page. However, the “PAGE DEST” indicates that this is the LEGS page where the destination waypoint is on. The model could now either type the destination in the scratch path or press the line key next to the destination in order to copy the destination to the scratch path. Whenever either one of those action is performed, the system would present the worlds “SCRATCH DESTINATION” to the model, which indicate the next state of the system. However, the system is still on the LEGS page.

Representation of task knowledge in the model

Task knowledge and instructions have an internal operator representation in the model. This is the same as with the model of Taatgen et al. To accommodate both learning methods in the model, the pre- and postcondition of an operator is either an internal or an external condition. For the list learning method, the conditions are internal, enabling every operator to trigger the following operator. The conditions in this case link the operators, and thus steps, to each other. This means that whenever a step in the instructions is executed, the following step will be triggered.
In contrast, for the context learning method the pre- and postconditions are external, relying on the environment. The pre- and postcondition in this context links the operators to the world. Whenever the state of the world matches the precondition of an operator, it would be executed. Postcondition in this case is the predicted state of the world after executing of that operator. For example, the state “PAGE LEGS” can be a postcondition for an operator to press the page-up key in order to find the destination page. The postcondition for this operator can be “PAGE DEST”. This indicates that whenever the system is on the LEGS page, the page-up key has to be pressed in order to find the destination page. The operator described here has external pre- and postconditions and is one of the operators of the context learning condition.
Another example of operators is one with internal pre- and postconditions. An operator with “STEP ONE” as precondition and “STEP TWO” as postcondition is typical for the list learning method. The action belonging to this operator is to press the line key of the destination waypoint, or to scratch the destination into the scratch pad. The conditions in this case present the point the model is at in the execution of a list of steps. Looking this step up at Table 1 tells us that this is the second step in the procedure. Thus, the precondition “STEP ONE”

indicates that the first step has to be executed in order to execute this step. The postcondition “STEP TWO” indicates that the second step has been executed.

Representation of general knowledge in the model

The general knowledge in the model consists of the production rules that control the decision making process. At first, the model perceives the state of the environment and then searches its declarative memory for an applicable operator. If so, the operator is placed in its operator module. Next, the operator is executed. After execution, the environment is inspected, and the outcome is saved in the declarative memory. How the production rules work together is outlined in Figure 9. Although most of the production rules are quite similar to the ones used by Taatgen et al, some of them are slightly different to fit the purpose of this model.
An example of a production rule is a rule to retrieve an operator from the declarative memory. The rule “retrieve-operator” states that whenever there is no operator in the buffer and the buffer of the declarative memory is free to use and empty, it should retrieve an operator based on the current task of the model. Whenever this rule fires, other rules checking this operator on applicability will fire. The process of deciding whether or not to execute an operator is described in the next section.

Finding an applicable operator

As mentioned above, operators are retrieved from the memory in order to see if they are applicable. If an operator’s condition matches the state of the external world, the operator is executed. However, the model also keeps an internal state of the world in its goal buffer. If an operator is found applicable, the model updates its internal state by the expected state (i.e. postcondition) of the operator. For the context learning method, this means that in most cases the external state will match the internal state. For the list learning method however, the internal state does not have to match the external state of the world. It could be that the internal state represents the next step to be executed, solely based on the last executed step. Whenever the internal state matches the precondition of the operator, it can also be found applicable. This way the model facilitates to learn both the learning methods and is able to act on them. If no applicable operators can be found, the model starts to explore for new operators. Note that the model in the context learning condition is forced to use the external state. The model in the list learning condition can either use the internal or external state, depending on the retrieved operator. The process of deciding if an operator is applicable is done by production rules. There are two types of production rules involved in this process, the rules that can accept an operator and the rules that can deny operators. For example, if an operator with precondition “STEP TWO” with the action to press the destination line key and postcondition “STEP THREE” is retrieved from the memory, but the systems internal state is a different step, e.g. “STEP ELEVEN” or an internal state based on an external representation e.g. “PAGE DEST”, the operator is found not applicable. However, if the internal or external state matches the state of the retrieved operator, it does fire. Note that the internal state is only matched whenever the state is a “STEP” state, except in case of display malfunctions.

Exploration for new operators

To quote Taatgen et al, one of the properties of learning from instructions is that instructions are forgotten easily. However, people can handle some of the situations in where they have forgotten steps in the instructions. The learned instructions are sometimes also not sufficient to solve the problems people face. This is the case in the FMS task. Forgetting of instructions is simulated in the model by eliminating operator chunks from the declarative memory at the start of the models run. There is a 30% chance for an operator to be forgotten by the model. This differs with the model of Taatgen et al who have a 25% chance of forgetting an operator. The reason for this is that the probability of the forgetting is a free parameter within both models and that the model described here gets too good due to changes in the exploration process as described below.

Whenever an operator is not in the declarative memory, or whenever its activation is too low to be retrieved, the exploration for new operators start. The way the model does this is by randomly guessing a possible action. As Taatgen et al report in the paper, people use this strategy too.

A major difference between the model described by Taatgen et al and the model used in this research is in the exploring strategy. After the action is tried out an operator will be created in the memory, and the result is stored in the declarative memory. If the action was successful, a new operator will be created that will be evaluated it for applicability the next time it is retrieved. However, if the result of the action does not bring the model closer to the solution the operator would also be stored. This time the operator will be flagged; it will not be used next time the model has to guess, which is a major difference between both models. In order to evaluate the outcome of an operator a hill-climbing algorithm is used. People somehow have the ability to judge whether they made progress or not, and this algorithm emulates this judging process. This may not be the most plausible way to emulate this ability to judge progress, but I have found no literature on operator exploration that accounts for a plausible approach of estimating the progress in a task. In this sense, the model is omnibus in evaluating the system state, which is unrealistic and oversimplified. Whereas the model of Taatgen et al only learned by the exploration of new operators, the current model can learn operators with external pre- and postcondition of executed internal operators. Whenever an operator is executed, the model stores the operator plus its external pre and postconditions into its declarative memory. As a result the model in the list condition learns the pre- and postcondition of operators with internal conditions.

The learning of specialized knowledge

The model learns new general knowledge by means of production compilation. Productions firing directly after each other are combined to a single rule, speeding up the executing of the procedure. Moreover, information of the declarative memory is used to compile into a new production rule, creating rules that are more specific. This has the advantage that the declarative memory is used less often, what saves time on retrieving operators.

For example if the retrieval rule mentioned above retrieves an operator with precondition “PAGE INIT” and postcondition “PAGE LEGS”, and the current system state is “PAGE INIT”, the operator will be found applicable. The

retrieve-operator rule is then compiled with the rule that found the operator applicable, combining the information of the retrieved operator and the two production rules in one production rule. Now whenever the system state is "PAGE INIT" this production can be selected, incorporating the firing of the two rules and the accessing of the declarative memory in one rule.

Handling display malfunctions

The model of Taatgen et al was not able to handle display malfunctions, thus this is a new part of the model. The model handles malfunctions in a fairly simple way. Whenever the display is scrambled (i.e. the visual input is unreadable), the system will assume that the state of the external world is the same as the expected state. This forces the model into using its internal state of the world. In case of a display malfunction, participants are sometimes still able to derive a state from the interface. Because of the abstraction level of the visual perception used in the model, the model itself cannot generate such a behavior. Thus, I estimated a free parameter representing the probability that the state is not successfully derived from the interface. This parameter is set to a probability of .80. Now, if a display error occurs and the previous system state was "PAGE INIT" and the operator to go to the legs page is executed, the expected state would be "PAGE LEGS". In the model, a rule is fired whenever the state cannot be derived, changing its external state presentation into the expected state. Therefore, the model expects to be in the "PAGE LEGS" and reasons from there. Whenever for example discontinuities occur, the model is likely to make mistakes because it acts on its expected state.

Handling state malfunctions

The way the model handles state malfunctions is simply be retrieving an operator and checking its applicability. This could be done either through the declarative memory or by matching a compiled production rule. As a consequence, there would be no difference between the control problems. However, looking at the data of the experiment this is not the case. There is even a difference in the correctness score of problems with state malfunctions and the control problems. This could be evidence that sometimes the internal state is being used, even in the context conditions. To simulate this, the model randomly encounters display malfunctions in all problems with an estimated probability of .10. As the free parameter described above, this parameter again represents the chance that the state of the system is not successfully abstracted from the display during the perception process. The use of the models internal state is forced on the model this way.
For example if the state changes of the system, i.e. a wrong line key action is performed while copying the destination to the scratchpad by the system, an scratchpad error would occur showing. Now if the state is scrambled by the random display malfunction, it forces the expected state on the system. While the model pressed the right key, the model expects the destination to be in the scratchpad and reasons from there. The way an error in solving the problem can occur is due to the mismatch between the internal and external state. I acknowledge that it would be better if the model would make errors on state malfunctions without the use of a free parameter, but due to the abstraction level of the visual input this is not possible.

Results

The model ran 150 times for both the context and list instructions. The same analysis as done on the experimental data was done on the model's data. This was done to compare the model and the data gathered during the experiment. In Figure 10 and Figure 11, the results of the model are displayed as a graph.

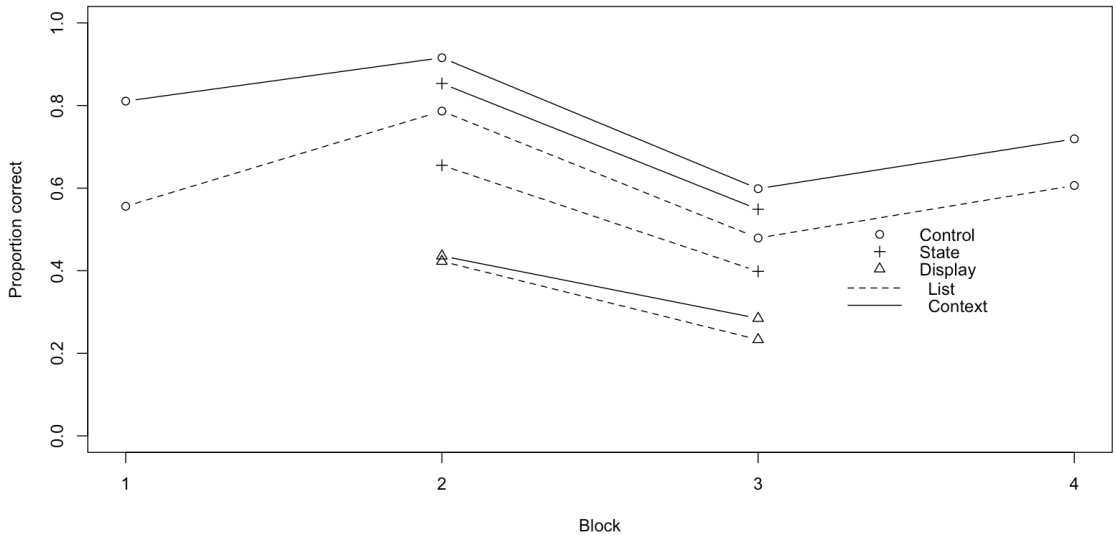


Figure 10. The results of the correctness score plotted per block.

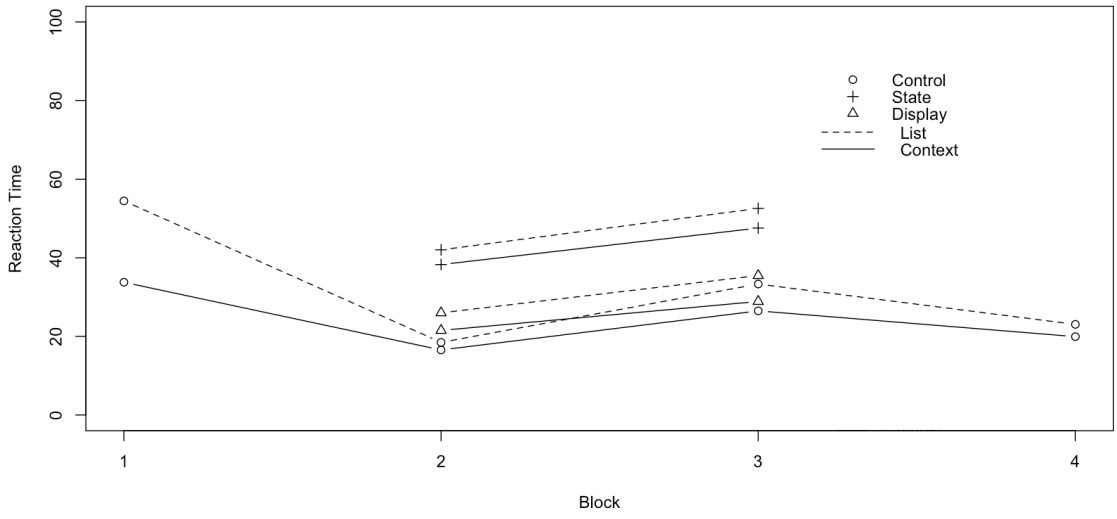


Figure 11. The results of the reaction time plotted per block. Reaction time is given in seconds.

LME-Analysis for the correctness score

The correctness scores of the model were fitted with a mixed effect model. The same model was used as with the correctness scores of the experimental data. As in the section on the results of the experiments, the effects found in the model's data will be discussed below. The results are summarized in Table 5. To facilitate comparison of the model and data, estimates of the data are also given. As with the mixed effect model of the data, estimates are given in log odds. Although it is hard to compare the estimates of the data and the model

individually per variable, the estimates show that in both the model and the experimental data display malfunctions score worse than the state malfunctions. Moreover, the models performance decreases over time as the estimate of the learning sequence show, in contrast to the experimental data, where the performance increases over time. I will further discuss this in the discussion.

Table 5. Results of the LME-analysis

	Estimate (data)	Estimate (model)	Std. error	t-Value	p-Value
Intercept	0.328	1.907	0.100	19.124	0.000
Display malfunction	-1.422	-2.023	0.066	-30.609	0.000
State malfunction	-0.735	-0.366	0.066	-5.35	0.000
List instructions	-0.369	-0.748	0.118	-6.323	0.000
Learning sequence	0.353	-0.121	0.011	-11.874	0.000
Display malfunction x List instructions	-0.584	0.575	0.095	6.047	0.000
State malfunction x List instructions	0.099	-0.103	0.092	-1.111	0.268

Display malfunction
The model's average log odd decreases significantly with -2.023 whenever a display malfunction problem is encountered ($p < 0.001$).

State malfunction
Whenever a state malfunction is encountered the log odds of the model decreased by 0.366 on average. This effect is also significant ($p<0.001$).

List instructions
The model in the list instructions condition score on average 0.748 lower than the model in the context instructions condition. In the models data this effect is significant ($p<0.001$).

Learning sequence
The learning effect in the model's data is negative. Log odds decrease with 0.121 on average. This effect is also significant ($p<0.001$).

Display malfunction x List instructions
The model in the list instructions condition scores significantly higher on average with a log odd of 0.575 than the model in the context condition whenever a display malfunction occurs.

State malfunction x List instructions
In the state malfunction condition, there is no significant difference between the context and list instruction conditions.

LME-analysis for the reaction time
The reaction times were also analyzed with a mixed effect model. The results can be found in Table 6 and will be further discussed below. The same mixed effect model is used as with the data of the participants to make it possible to compare the data from the model and the experimental data.

Table 6. Results of the LME analysis

	Estimate (data)	Estimate (model)	Std. error	t-Value	p-Value
Intercept	10.739	3.196	0.021	154.06	0.000
Display malfunction	0.290	0.045	0.021	2.17	0.030
State malfunction	0.424	0.735	0.016	44.93	0.000
List instructions	0.169	0.125	0.017	7.59	0.000
Learning sequence	-0.172	-0.068	0.003	-20.77	0.000

Display malfunction
The effect of the display malfunctions is significant ($p = 0.030$). For display errors, the reaction time goes up.

State malfunction
If the model encounters a state malfunction then the reaction time goes up according significantly ($p<0.001$).

List instructions
The main effect of instructions is also significant ($p<0.001$). Reaction time increases for the list instruction condition.

Learning sequence
There is a significant effect of the learning sequence ($p<0.001$). Participants get faster as they do more trials.

Discussion
In this session, I will discuss the performance of the model and compare the results of the model with those of the experiment.

Context instructions vs. list instructions
The main effect of instructions is significant in both the fixed effect model of the correctness scores and the fixed effect model of the reaction times. Although the latter is only significant in the fixed effect models capturing the effects of the experimental data, the model successfully captures the overall performance of the data, and is in line with the results found by Taatgen et al (2008). However, the effect is much larger in the model's data than in the experimental data. This difference is probably due to noise in the data and could probably be influenced by the parameters used by the model.

Malfunctions
Examining the main effects of the malfunctions of the correctness scores, I find that the manipulation of the malfunctions also worked in the model. In case of display malfunctions the model performance is much lower than in case of the state malfunctions. One could argue that this effect is only caused by the fact that display malfunctions occur during the state malfunctions to force use of the internal state. However, the model is also forced to use the internal state in the control problems. We could conclude that the malfunction in the model worked as it did in the experiment. Note that it is possible that the state errors were just too simple or that forcing the model to take more actions simply increases the chance on errors.

The effects of the malfunctions in the reaction times of the model differs somewhat of those effects found in the experimental data. While the mixed effects model show that state malfunctions take longer than display malfunctions, which is the case for both the model and experimental data, examining Figure 7 and Figure 11 reveals a difference. In block 2 it seems that the context instruction group performs much slower than they do in block three. This is probably because the encounter of a display malfunction confused the participants. It seems that the list instruction group is less confused, but this does not have to be entirely true, for they perform worse on correctness scores in case of display malfunctions in the experimental data. The experimental data does not catch the phenomena; the model does not get confused in terms of longer reaction times. Reason for this is probably due to the abstraction of the external world; the model does not hesitate before making a decision.

Difference in the effect of display malfunctions

The interaction between the display malfunction and instructions reveals a difference between the model's data and the experimental data. When looking at the interaction of the display malfunctions and instructions, the model performances on display malfunction problems compared to the control problems is worse for the context condition than for the list condition. This effect is opposite in the experimental data. The participants in the context group score higher on display malfunctions relative to the control problems than the participants in the list group. This strengthens the belief that visual cues are important in solving the problems with display malfunctions. Whereas the model successfully uses the step like operators to solve the display malfunction problems in the list condition, the experimental data shows empirical evidence that in fact the context condition benefits in case of display malfunctions. This difference can be due to the level of abstraction of the visual perception in combination with the ideal exploration algorithm implemented in the model.

Perhaps a model working with a more complex representation of the external world and a more plausible exploration mechanism can catch this difference. As mentioned earlier in this thesis, I did not find any literature regarding a plausible mechanism for the exploration of operators from a complex visual interface.

State malfunctions

Consistent with the experimental data, the mixed effect model of the correctness scores of the model shows no interaction effect between the state malfunction and instructions. This means that probably due to the learned condition operators, the list instruction group performs equally well on problems with malfunctions as the context instruction group relative to the control problems. This indicates that although the participants in the list instruction group, who first start out with step like operators, probably learn to rely on their visual input relatively fast. However, this does not happen as well as with the context instruction group, whose instructions are focused on visual input.

Difference in learning sequence

The most curious difference between the model's data and experimental data is the effect of the learning sequence. The learning sequence in the model's mixed effect model is negative for correctness scores. Examining both Figure 6 and Figure 10, we find that in the last two blocks the correctness score of the model lies lower than the correctness score of the participants. The model as presented in the paper of Taatgen et al (2008) had a better fit on the problems presented used in the last two blocks. The difference is probably caused by the fact that the internal state is forced up on the model presented in this paper during the task, causing the model to make more mistakes whenever a harder problem is presented. Examining the effect of learning in the reaction times, we find that the model does get faster over time. This is consistent with the effect of learning found in the reaction times of the experimental data. In the model, the compilation of new production rules, the increasing of activation of chunks, and the newly learned operators are cause for the decrease of reaction times. In case of the compilation of new production rules, the model gets faster because it skips the retrieval process of an operator. Secondly, as the activation of operator chunks raises by every new encounter of the operator, the retrieval time decreases, decreasing the reaction times. Finally, the exploration process of new operators increases reaction time. This happens because the model checks whether an operator has already been tried by retrieving operators from its declarative memory. Whenever the model successfully learns new operators, its exploring behavior decreases, decreasing reaction times.

General discussion and conclusions

In the discussion sections above, I already discussed the results of the experiment and those of the model. Below, I will discuss the implications of this study.

I already tried to answer the research question "Is the context learning method more flexible and robust to system malfunctions than the list learning method?" In short the answer to this research question is "yes". Overall, the context group performed better than the list group. In case of display malfunctions, the participants benefitted from the context learning method. In comparison to the control problems, the context group outperformed the list group. For the state errors, the context group did not seem to have an advantage. There is no relative difference found between control problems and problems with a state malfunction. Below, I will go into more detail on what makes the context learning method more flexible and robust.

Robustness

Taatgen et al (2008) associate robustness with "the ability to protect skilled performance from various disturbances, including unexpected events, interruptions, or changing demands." Handling forgotten operators or the recovering of errors is seen as robustness of a skill. Display malfunctions and state malfunctions as introduced in this thesis test the robustness of a skill in this definition of robustness. I presented empirical evidence that in cases of display malfunctions and state malfunctions the context learning method performed better than the list learning method. Although the model did not show all the interaction effects visible in the data, overall it showed that the context method worked better as the list learning method, using the theoretical basis of ACT-R and the minimum control principle. The fact that even the participants trained with the list learning method performed worse in the display malfunction condition, and that they performed equally well on state malfunction relatively to the control problems, shows that participants in the list instruction condition also try to exert control to the environment. This shows that even after a few trials the list instruction group learns the pre- and postconditions of the different operators.

Flexibility

Flexibility is defined as "the ability to apply a skill to new problems that are different from the problems that served as the basis for training" (Taatgen et al, 2008). Again, the evidence presented in this thesis shows that the context method is more flexible than the list method. In new problems, with and without malfunctions (block three and block four), the context learning method performs better than the list learning method.

Implications

The most important implication of this research is that the environment and its visual input are important in the process of robust and flexible skill acquisition. To handle new problems and system malfunctions, a learning method focusing on the environmental cued pre- and postconditions provides the most benefit. The model also implicate that people tend to exert as much control to the environment as possible, and that people tend to learn the post-

and preconditions of different system states fairly fast. However, from time to time, people use internal control as well. The reason for this remains unclear in this thesis. What could be going on is that people do not always succeed in deriving the current state from the display. E.g., if someone does not scan the complete display and misses important information, a wrong state could be derived or no state at al, forcing the expectation to play a bigger role.
Another implication of this thesis is that an abstraction of the external world does not always work. Even though it makes reasoning about system states and the learning of skills easier, it turned out not to be sufficient to capture the effects found in the experimental data. Although the model presented in this paper is enhanced with a more sophisticated operator exploration process and the ability to learn operators while executing them, improvement in the perceptual part of the model is needed to catch the whole learning process and effects found in the data.

Future research

While working on this thesis, I came across a few problems and challenges that in my opinion are interested to subject to future research. As mentioned above one of those challenges is to make a more accurate model of the world. This leads to another interesting subject, which is the exploration of operators. The model now *knows* whenever the current state is closer of further away from the goal state. It would be interesting to investigate how people get this feeling of progress and how the visual input of the environment is exactly formed to some kind of state.
Another interesting case would be to do another study on context and list learning, but by changing the paradigm used in this thesis. I would propose to train the list instruction group in a manner that they know which button to press where in the interface. This for example, by letting them push the sequence of buttons several times in a row, and then after that subjecting them to the experimental problems.

References

Altmann, E. M., & Trafton, J. G. (2002). Memory for goals: An activation-based model. *Cognitive Science , 26* (1), 39-83.

Ament, M. G., Blandford, A., & Cox, A. L. (2009). Different Cognitive Mechanisms Account for Different Types of Procedural Steps. In N. A. Taatgen, & H. van Rijn (Ed.), *Proceedings of the 31 Annual Conference of the Cognitive Science Society* (pp. 2170-2175). Cognitive Science Society.

Anderson, J. R. (1996). ACT: A simple theory of complex cognition. *American Psychologist , 51*, 355-365.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, G., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review , 111* (4), 1036-1060.

Botvinick, M., & Plaur, D. C. (2004). Doing without schema hierarchies: A recurrent connectionist approach to normal and impaired routine sequential action. *Psychological Review , 111* (2), 395-429.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation , 2* (1), 14-23.

Brooks, R. (1991). Intelligence Without Reason. *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (pp. 569-595). San Mateo, CA: Morgan Kaufmann.

Byrne, M. D., & Anderson, J. R. (2001). Serial modules in parallel: The psychological refractory period and perfect time-sharing. *Psychological Review , 108*, 847-869.

Cox, A. J., & Young, R. M. (2000). Device-oriented and task-oriented exploratory learning of interactive devices. In N. Taatgen, & J. Aasman (Ed.), *Proceedings of the third international conference on cognitive modelling* (pp. 70-77). Veenendaal,The Netherlands: Universal Press.

Dhillon, B. S., & Lui, Y. (2006). Human error in maintenance: a review. *Journal of Quality in Maintenance Engineering , 21* (1), 21-36.

Fennell, K., Sherry, L., Roberts, R. J., & Feary, M. (2006). Difficult Access: The Impact of Recall Steps on Flight Management System Errors. *The International Journal of Aviation Psychology , 16* (2), 175-196.

Fitts, P. M. (1964). Perceptual-motor skill learning. In A. W. Melton (Ed.), *Categories of Human Learning* (pp. 243-285). New York: Academic.

Gaba, D. M. (2001). Simulation-based training in anesthesia crisis resource management (ACRM): A decade of experience. *Simulation & Gaming , 32* (2), 175-193.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in Soar: The anatomy fo a general learning mechanism. *Machine Learning , 1* (1), 11-46.

Logan, G. D. (1988). Towards an Instance Theory of Automatization. *Psychological Review , 22*, 1-35.

McIlvaine, W. B. (2006). Human error and its impact on anesthesiology. *Seminars in Anesthesia, Perioperative Medicine and Pain , 25* (3), 172-179.

Newell, A., & Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 1-56). Hillsdale, NJ: Erlbaum.

Taatgen, N. A. (2007). The Minimal Control Principle. In W. D. Gray, *Intergrated Models of Cognitive Systems* (pp. 368-379). New York: Oxford University Press.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The Acquisition of Robust and Flexible Cognitive Skills. *Journal of Experimental Psychology: General , 137* (3), 548-565.
Taatgen, N., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors , 45* (1), 61-76.
VanLehn, K. (1996). COGNITIVE SKILL ACQUISITION. *Annual Review of Psychology , 47*, 513-539.

Appendix I: General instructions

Overview of the Experiment

The first step of this experiment requires that you thoroughly study the following training material. Then you will go through a short session of interface training. Once you have studied the material and successfully finish the interface training, you will take a short quiz to measure your pre-task knowledge. Then, you will perform an aviation task and your performance on this task will be measured. After that, you will fill out a post-task questionnaire. At last, you will fill out the demographics and payment forms.

Overview of the Flight Management System

In this experiment you will take the role of a pilot on a Boeing 777 airplane. Modern commercial aircraft, such as the 777, have computer autopilot systems known as *Flight Management Systems*. You will be trained on how to use a Flight Management System.

Your task will be to make changes to the flight plan of the plane as indicated by Air Traffic Control. The flight plan is simply the *route* that the plane's autopilot (Flight Management System) will follow. A route consists of a series of *waypoints* (GPS coordinates) that are linked together. The following is a printout of a route from Pittsburgh to Philadelphia (notice the waypoints in the second column):

	ID	Name	Freq.	Alt.	HDG To Next	Distance To Next
APT	KPIT	Pittsburgh Intl	------	1204	From : 284º / To : 104º	86.23 Nm (86.23 Nm)
INT	MIROY	----------	------	10000	From : 280º / To : 100º	17.87 Nm (104.09 Nm)
INT	COFAX	----------	------	18000	From : 281º / To : 101º	22.13 Nm (126.22 Nm)
INT	LOMON	----------	------	10000	From : 290º / To : 110º	45.85 Nm (172.07 Nm)
INT	BOUHN	----------	------	10000	From : 291º / To : 111º	26.06 Nm (198.13 Nm)
INT	TRAGG	----------	------	10000	From : 286º / To : 106º	8.78 Nm (206.91 Nm)
INT	BUNTS	----------	------	3000	From : 311º / To : 131º	26.11 Nm (233.02 Nm)
APT	KPHL	Philadelphia Intl	------	36		Total : 233.02 Nm

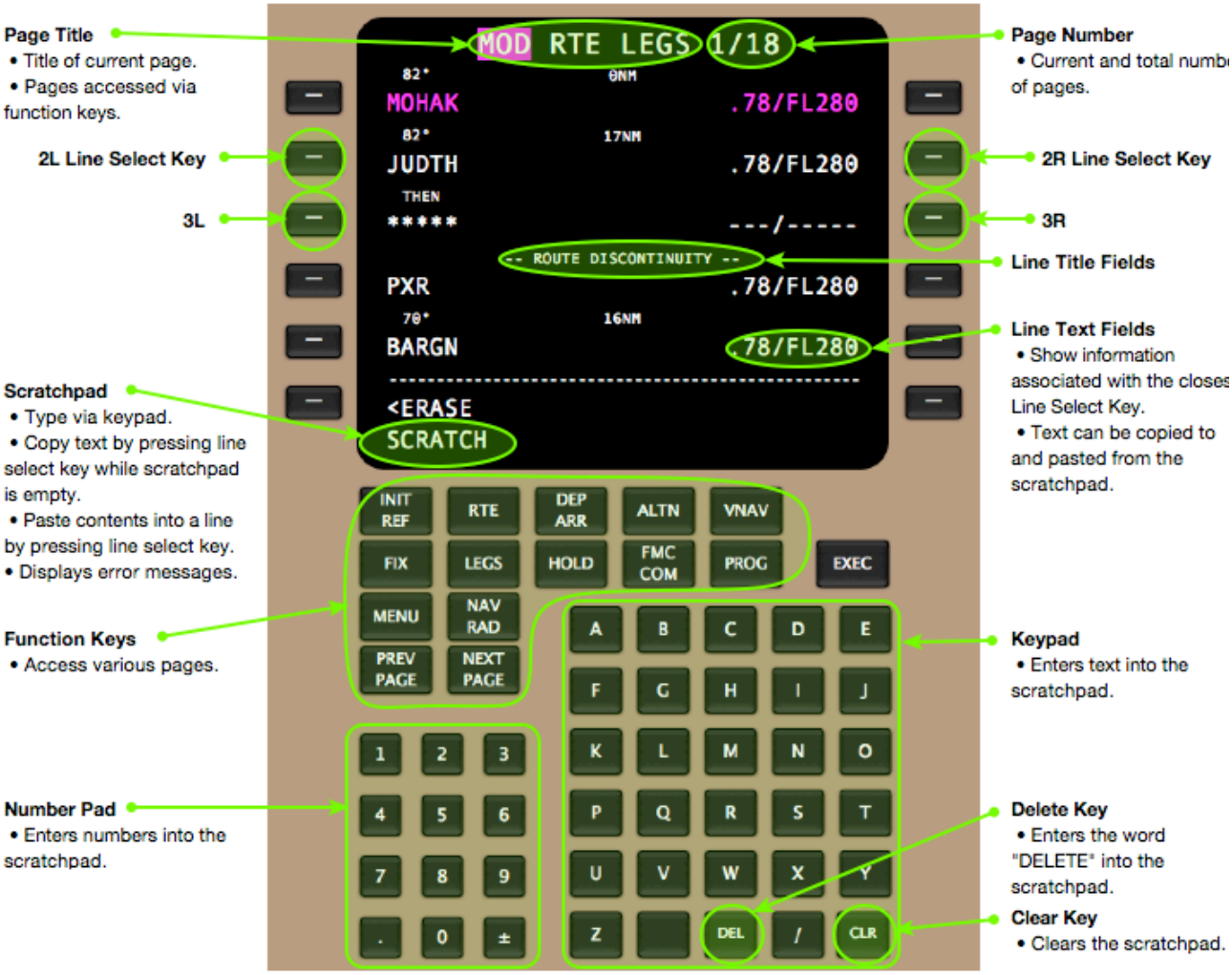
Routes, such as these, will be pre-programmed into the Flight Management System. Your task will be to make in-flight manipulations to a route by interacting with the airplane's Flight Management System in a computer simulator.

There are two relevant parts to the Flight Management System:

- The *Control Display Unit* displays text information and allows for the inputting of data and commands.
- The *Navigational Display* is a digital map on which you can see your current route.

The Control Display Unit (CDU)

The Control Display Unit consists of a keyboard and a display. The following is a labeled diagram:



The display contains a *Title Field*, six *Line Fields*, a *Scratch Pad* and a variety of buttons.

The title displays the title of the current page. Different pages are accessed with the function keys. A page may itself have one or more sub-pages. The current sub-page and maximum number of sub-pages are shown in the top, right corner. To cycle through the sub-pages one uses the “PREV PAGE” and “NEXT PAGE” keys.

There are 6 Line Select Keys on each side of the display (1L...6L, 1R...6R). Each Line Key is associated with a Text Field on the display showing information about the flight plan. Line keys are used for making or canceling changes in the flight plan.

The Scratchpad is used to input, copy, and paste text as well as display error messages. Text can be input into the scratchpad via the keypad. If there is text in the Scratchpad and the current Page allows the operation, pushing a line key will attempt to paste whatever is in the scratchpad into the text field associated (next to) with it. If the Scratchpad is empty, then pushing a Line Key may copy whatever is in the associated text field into the Scratchpad. If improper values are entered when using a

Line Key errors may occur. Error messages will appear in the scratchpad. These must be deleted via the use of the “CLR” key.

Primary Operations with the CDU

To change a flight plan:

- The current flight plan is shown on the LEGS and RTE pages
- Changes to the flight plan can be made by adding / removing / replacing waypoints and routes

To copy something to the scratchpad:

- The scratchpad must be empty (use the clear key)
- Select the line key next to the item that will be copied

To type text into the scratchpad:

- Select the correct sequence of keys on the keyboard

To Paste something from the scratchpad:

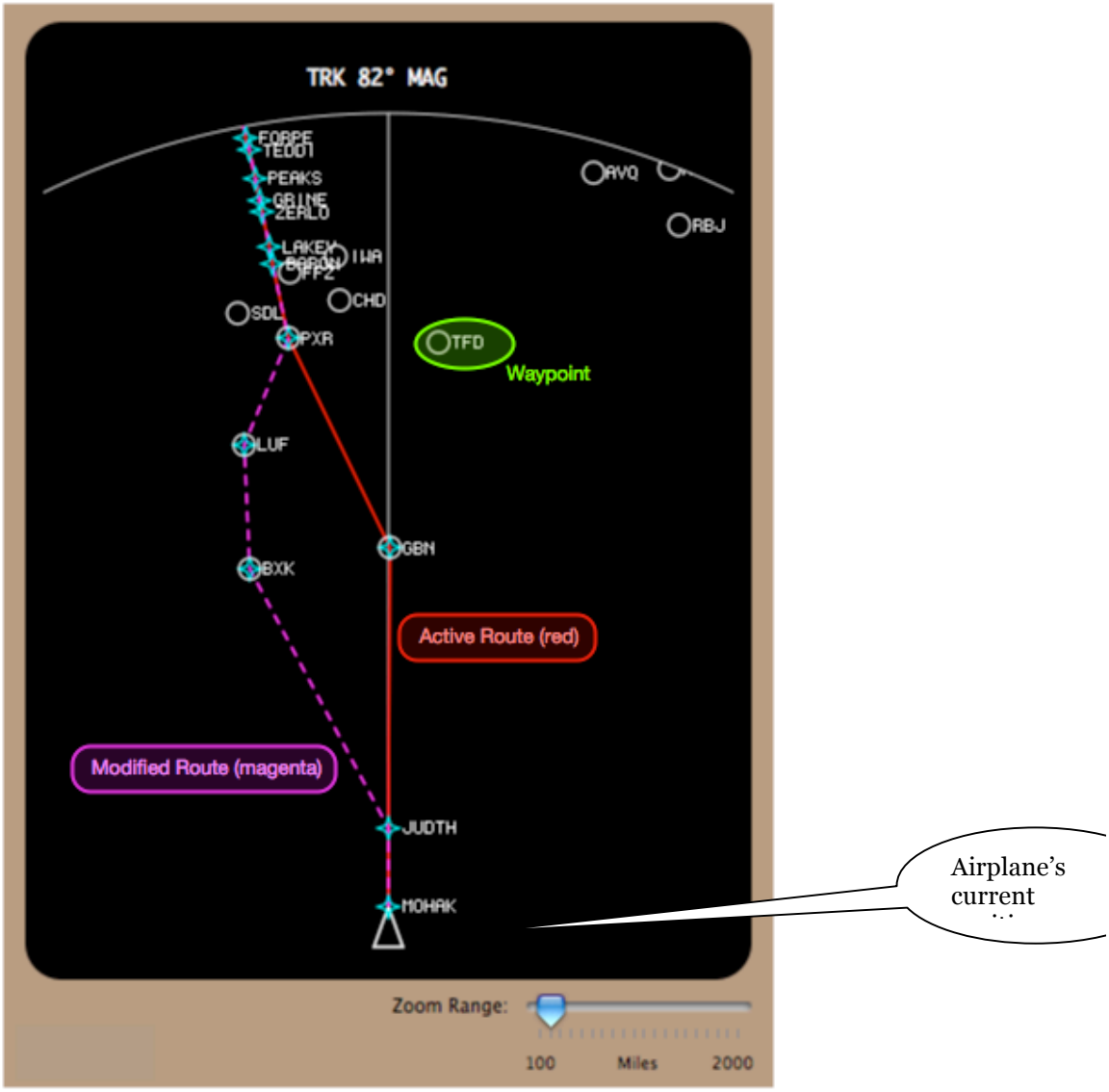
- Hit the line select key next to where you want to paste the item. It will then appear in that text field.

To make a change permanent:

- Press the EXEC key
- Finishing a task without pressing EXEC will produce an incorrect flight plan

The Navigation Display

The Navigation Display shows the current and modified routes, the planes position along those routes, and waypoints not on the route. The following is a labeled picture of the Navigation Display:



The triangle on the bottom of the screen represents the airplane's current position. The red line is the plane's current, active route. Again, a route is a series of waypoints. Once you start modifying the route, a second, dashed, magenta line will appear on the display, showing you what your current modification of the route looks like. The zoom range slider shows the current scale of the map. You can move the slider to change the zoom of the map. If the map-scale is above approximately 500 miles then only waypoints on your current route will be displayed. This is done to reduce clutter.

Task

Your task is to make modifications to the flight plan as requested by Air Traffic Control (ATC). ATC's instructions are given in the small rectangle at the lower-right corner of the FMS window, below the navigational display. Here you also see if your modifications are correct or incorrect.

The ATC will identify you as "Flight 123". You will start by reading the ATC's instruction. Then you have to make the requested modifications to the flight plan by using the function keys and the keypad. You can use the navigational display to check if the modified flight plan is correct. Press the EXEC key when you want to make a modification final. Press the "Finish" button to display the feedback (correct or incorrect) and advance to the next problem. At this point it is not possible to make any further changes to the flight plan, even if it is incorrect.

Try your best to give fast and correct solutions.

NOTE: This task will be performed with a mouse (no keyboard). VERY RARELY WILL YOU ACTUALLY WANT TO DOUBLE-CLICK. DOUBLE CLICKING CAN HINDER YOUR PROGRESS.

Appendix II: List instructions

Additional Instructions

When Air Traffic Control gives you a directive to proceed directly to some waypoint, go through the following steps:

1. Press the LEGS key
2. Enter the desired waypoint in the scratchpad
3. Push the 1L key
4. If the word “discontinuity” appears on the screen, follow the procedure to remove discontinuities.
5. Verify the route on the Navigational Display
6. Press EXEC

There is a separate procedure for removing discontinuities:

1. Press the LEGS key
2. Press the line select key after the discontinuity
3. Press the line key with the THEN prompt

ALSO: EACH KEYPRESS DENOTES AN ACTION VERY RARELY WILL YOU ACTUALLY WANT TO DOUBLE-CLICK. DOUBLECLICKING CAN EASILY HINDER YOUR PROGRESS

Appendix III: Context instructions

Additional Instructions

When Air Traffic Control gives you a directive to proceed directly to some waypoint, use the following list of steps to accomplish this:

- When you are not yet on the LEGS page, **Press the LEGS key** to get to the LEGS page.
- When you are on the LEGS page, **Enter the desired waypoint in the scratchpad** so you can put it into the flight plan
- When you have your new destination in the scratchpad, **Push the 1L key** to move the waypoint from the scratchpad into the first line of the flightplan.
- If "route discontinuity" appears on the screen, **follow the procedure to remove discontinuities** to remove the discontinuity
- When you have modified your route and have removed all route discontinuities, **Verify the route on the Navigational Display** to make sure that the FMS has made the changes as you intended them
- When you have verified that the route modification looks ok on the Navigational Display, **Press EXEC** to finalize the route modification.

There is a separate procedure for removing discontinuities:

- When you are not yet on the LEGS page, **Press the LEGS key** to get to the LEGS page.
- If you are on the LEGS page and there is a discontinuity in one of the lines, **Press the line select key after the discontinuity** to copy the waypoint after the discontinuity in the route into the scratchpad.
- If you have the waypoint after the discontinuity in your scratchpad, **Press the line key with the THEN prompt** to copy the waypoint after the discontinuity into the line with the discontinuity, and reconnect the route.

ALSO: EACH KEYPRESS DENOTES AN ACTION <u>VERY</u> RARELY WILL YOU ACTUALLY WANT TO DOUBLE-CLICK. DOUBLECLICKING CAN EASILY HINDER YOUR PROGRESS