UNIVERSITY OF CALIFORNIA
RIVERSIDE

Statics Intelligent Tutoring System

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Mechanical Engineering

by

Chia-Keng Lee

June 2011

The Thesis of Chia-Keng Lee is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

First of all, I would like to thank the members of my committee for their time and valuable suggestions. I am deeply and especially indebted to my advisor, Professor Thomas F. Stahovich, for his constant advice and support, as well as the endless hours he has spent providing suggestions on improvement to the programs, thesis, and presentations. Without that, this thesis literally stops right here.

I also would like to thank my lab mates for their help on my research. I especially thank Eric Peterson for the endless help with technical issues, especially Statics; Weesan Lee for Newton's pen and network setups; Tyler Bischel for his seat in the lab; Josiah Jordan for chats; Matt for labeling; and Jim for the awesome coffee.

Thanks to my parents and sister for their support.

Thanks to Paul Dorman, my coworker who helped me edit my thesis. Thanks to my friends, for entertainment purposes.

To my parents, James Lee, Sherry Lee

and my sister, Juliann Lee,

and my grandmother, Heung-Sheh Lee

ABSTRACT OF THE THESIS

Statics Intelligent Tutoring System

by

Chia-Keng Lee

Master of Science, Graduate Program in Mechanical Engineering
University of California, Riverside, June 2011
Dr. Thomas F. Stahovich, Chairperson

We present an intelligent pen-based tutoring system for statics – the sub-discipline of engineering mechanics concerned with the analysis of mechanical systems in equilibrium under the action of forces. The system scaffolds students in the construction of free-body diagrams and equilibrium equations for planar devices comprised of one or more rigid bodies.

While there has been extensive research in intelligent tutoring systems, most existing systems rely on traditional WIMP (Windows, Icons, Menus, Pointer) interfaces. With these systems, students typically select the correct problem solution from among a set of predefined choices. With our pen-based interface, by contrast, students are guided in constructing solutions from scratch, mirroring the way they solve problems in ordinary practice, which recent research suggests is particularly important for effective instruction.

Our system embodies several innovations including a novel instructional technique that focuses students' attention on a system boundary as a tool for constructing free body diagrams, a tutorial feedback system based on "buggy rules" that attempts to diagnose and correct problem-solving errors typical of novice students, and a hierarchical feedback system which promotes independent problem-solving skills. In winter 2010, the tutoring system was used by 100 students in an undergraduate statics course at the University of California, Riverside. Results from pre- and post-tests reveal measurable learning gains after only a short exposure to the system. In an attitudinal survey, students reported that, while there is room for improvement, the interface was preferable to a WIMP interface and the methodology implemented in the system was valuable for learning statics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

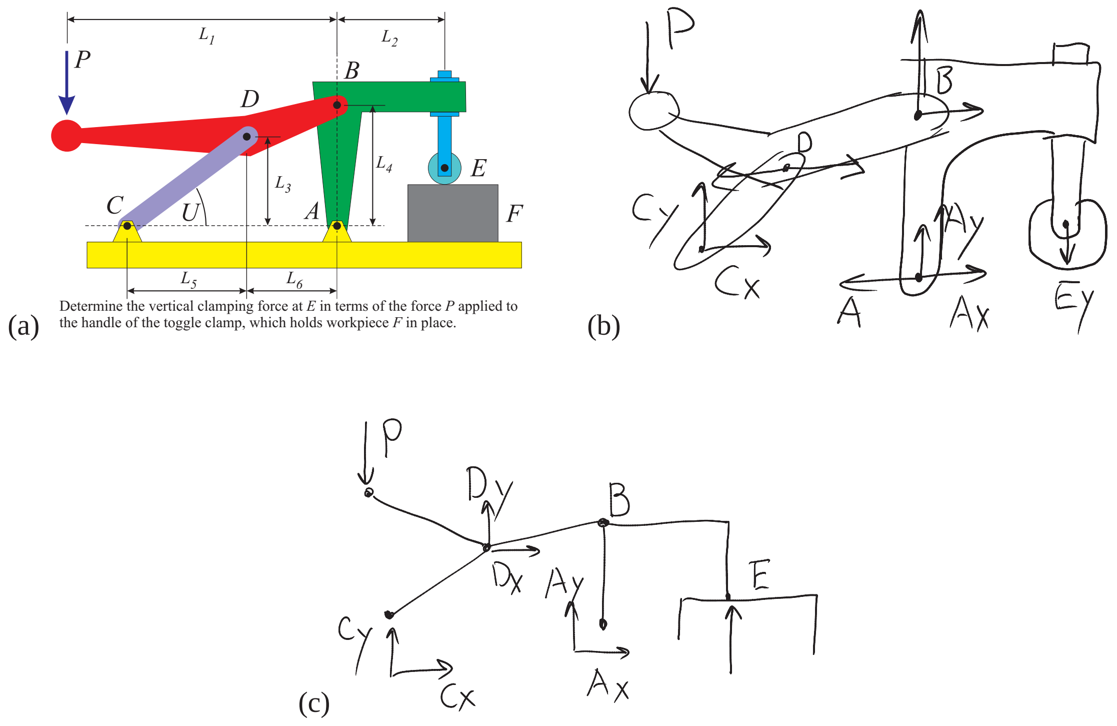
Intelligent tutoring systems have long been a focus of research, with applications spanning a wide range of subjects such as computer programming [7], law [28], medicine [29], physics [30], and statics [25]. Most current systems rely on traditional WIMP (Windows, Icons, Menus, Pointer) user interfaces. While such interfaces may be useful in some domains, they have clear limitations. For example, they are typically not well-suited for instruction in domains in which solutions require complex graphics. Engineering statics is one such domain. Solving statics problems requires the construction of free body diagrams and equilibrium equations, two tasks which are difficult to perform with a mouse and keyboard.

Here we present our efforts to create a pen-based tutoring system that scaffolds students in solving statics problems in much the same way they ordinarily solve them with pencil and paper. This work is motivated by recent research comparing student performance across different user interfaces showing that "as the interfaces departed more from familiar work practice..., students would experience greater cognitive load such that performance would deteriorate in speed, attentional focus, meta-cognitive control, correctness of problem solutions, and memory" [21]. While that work used systems that provided no tutorial assistance, the findings provide compelling evidence of the potential of natural user interfaces for instructional tools.

Our statics tutoring system, called Newton's Pen II (NP2), employs a novel instructional design. A key element of this design is our use of an explicit system boundary as a tool for constructing free body diagrams. Novice students often struggle to define the system of interest. For example, they may attempt to draw a free body diagram by simply redrawing the original problem figure, with all of its details. Alternatively, they may draw a variety of incorrect abstractions, such as stick figures or forces floating in space, with no clear connection to objects. Figure 1.1 shows typical errors. NP2 scaffolds the student in selecting the system of interest by guiding the student in the selection of an explicit system boundary. To construct a free body diagram, the student uses a stylus to trace the system boundary directly on the problem picture, and then drags the boundary to the free body diagram workspace. Having an explicit boundary helps the student distinguish between external forces, which must be represented on the free body diagram, and internal forces which should not appear.

To provide tutorial feedback, NP2 uses a "buggy rules" approach that attempts to diagnose and correct problem-solving errors typical of novice students. These rules were based in part on examination of student work from a statics course taught at the University of California, Riverside. Feedback is hierarchical, with general suggestions given first, and more detailed help given in response to repeated queries from the student. This design is intended to promote independent problem-solving skills by giving students the opportunity to identify mistakes on their own.

To evaluate the educational usefulness of the NP2, we conducted a user study in an undergraduate statics course. This sophomore-level course included just over 100 students. The study took place during the regularly scheduled discussion sections, and because of time-constraints, we



Determine the vertical clamping force at *E* in terms of the force *P* applied to the handle of the toggle clamp, which holds workpiece *F* in place.

Figure 1.1: Typical errors in free body diagrams drawn by novice statics students. (a) Problem description. (b) Free body diagram constructed by redrawing the original problem figure. Internal forces are erroneously included. (c) An abstract free body diagram drawn as a stick figure.

assessed only learning gains related to the construction of free body diagrams, and did not consider gains for equilibrium equations. Comparison of performance on pre- and post-tests revealed significant learning gains. Furthermore, in an attitudinal survey, students reported that, while there is room for improvement, the interface was preferable to a WIMP interface and the instructional design implemented in the system was valuable for learning statics.

NP2 is designed to run on a tablet PC. In future work, we would like to compare the effectiveness of a tutoring system running on a tablet PC to one running on a digital pen and paper system. Therefore, in developing NP2, we first created a toolkit that abstracts away the details of the specific hardware platform, thus facilitating porting NP2 to other pen-based systems.

The next section places this work in the context of related work. This is followed by an overview of the toolkit. Afterwards, we present an overview of the Newton's Pen II system and details of the system’s design. Finally, results of the user study are presented and discussed.

Chapter 2

Literature Review

2.1 Toolkit

There are two toolkits that are closely related to our work. Satin [11] is a powerful toolkit for pen-based interfaces development. It can process and render ink, and can recognize pen strokes. While Satin is intended to provide functionality for creating graphical user interfaces, our toolkit is intended to facilitate the development of pen-based applications that can run on multiple hardware platforms.

PENTOOLS is another toolkit closely related to ours [9]. It is developed in MATLAB and designed for rapid prototyping of on-line pen computing applications. The toolkit provides functionality for storing stroke data, computing stroke features, and editing strokes. Unlike our toolkit, PENTOOLS does not support multiple hardware platforms and does not provide shape recognizers.

InkKit [5] and Silk [15] can automatically generate a working user interface from a designer's sketch of it. These toolkits are not hardware independent and are intended for building traditional GUI interfaces rather than pen-based applications.

2.2 Intelligent Tutoring System

Intelligent tutoring systems have been developed for a wide variety of domains [7, 28, 25, 29, 30]. Nearly all of these systems employ WIMP interfaces and work from unambiguous input provided with a keyboard and mouse. Our work, on the other hand, focuses on building pedagogically-sound, pen-based interfaces for tutoring systems. As such, we need to deal with the challenges of working from ambiguous, hand-drawn input.

The development of natural pen-based user interfaces is an area of active research. A wide variety of experimental systems have been developed, including tools for: simulating simple, hand-drawn mechanical devices [1]; sketching user interfaces [16]; drawing Unified Modeling Language diagrams [10]; interpreting hand-drawn equations [19]; and understanding military tactics[8]. Likewise, there has been significant progress in sketching 3D shapes [32].

Researchers have recently begun to explore the application of pen-based interfaces to instructional systems. For example, Oviatt [21] compared student performance using a variety of platforms including paper and pencil, the Anoto digital pen [3], and the tablet PC. These platforms were used only as devices for recording problem-solving work. No tutoring capabilities were provided. This work by Oviatt suggests that as the interface departs from familiar work practice, student performance decreases on a variety of measures such as speed and correctness. In a related study evaluating interfaces for entering mathematical equations into a computer, Anthony [4] found that pen-input of equations is substantially more efficient than keyboard entry, and is greatly preferred by users.

Classroom Presenter [2] is perhaps one of the most widely used pen-based instructional tools. This lecturing system allows students and instructors to communicate during lectures using tablet computers. However, this system neither interprets what is written nor does it provide tutoring capabilities.

Newton's Pen [17] is a statics tutoring system implemented on LeapFrog Inc.'s FLY^{TM} pentop computer. The FLY^{TM}, which employs Anoto technology [3], is a ballpoint pen with an embedded computer processor and an integrated digitizer that works in conjunction with dot-patterned paper. While Newton's Pen is limited to the analysis of particle equilibrium problems, NP2 provides instruction for more complex frame and machine problems involving multiple rigid bodies. Similarly, Newton's Pen is only capable of providing simple tutorial feedback, while NP2 uses a "buggy rules" approach to diagnose and correct problem-solving errors typical of novice students.

Kirchhoff's Pen is a pen-based tutoring system for teaching Kirchhoff's Law [6]. Similar to NP2, this system was designed for a tablet PC. However, there are several important differences between the systems. For example, the two domains involve very different recognition tasks. Kirchhoff's pen must interpret annotations on a circuit, while NP2 must interpret complex free body diagrams. Similarly, the two systems have different tutorial content, with NP2 using a more sophisticated buggy rules approach.

The InTEL system [26] is a state-of-the-art web-based statics tutoring system. To construct the free body diagram, a student clicks the mouse to select objects from the problem description. Forces can be added at predefined locations in predefined orientations. By contrast, NP2 requires the student to construct a free body diagram from scratch by drawing a system boundary and drawing forces at any location the student deems appropriate. We believe that constructing a solution from scratch facilitates a more effective learning experience than allowing the student to select the correct answer from a list.

NP2 must recognize hand-drawn sketches in real time. This process is difficult because hand-writing is often ambiguous. NP2 employs a combination of the inverse-curvature recognizer [18] and the dollar recognizer [31]. We extended the latter to recognize arrows with greater accuracy.

Chapter 3

A Pen-Based Toolkit

3.1 Introduction

Different pen-based hardware platforms have different application programming interfaces (APIs). Therefore, applications designed for one platform will typically have to be rewritten to work on another. As an alternative, we created a toolkit that provides a standard API that abstracts away the details of the hardware-specific APIs. A software developer can create an application using the standard API and the toolkit can automatically translate the API calls to work with the device specific APIs.

We implemented the standard API as a reusable toolkit. In addition to providing hardware abstraction, the toolkit also provides a variety of functions that are typically used when building pen-based applications such as shape recognition, stroke resampling, etc.

Therefore, we have three goals:

- To create a toolkit that facilitates the development of portable pen-based applications that run on multiple platforms.

- To provide a library of functions, such as shape recognition and stroke resampling, to facilitate the creation of pen-based applications.

- To enable applications for pentop computers to be developed on a desktop computer with its greater processing power, visual display, and symbolic debugger.

3.2 Standard API

Currently, the toolkit can be used to create applications that run directly on a tablet PC or on a computer with a mouse. The toolkit can also be used to create applications for a FLY^{TM} pentop computer. In this case, however, the applications run on a PC, with the FLY^{TM} tethered to the PC with a debugging cable. The applications can not run directly on the FLY^{TM} because the toolkit is in Java while the FLY^{TM}'s API is in C++. However, we plan to extend our toolkit to work on the Livescribe pentop computers which can run Java natively.

The standard API provides display functionality including the ability to play audio and synthesized speech, and the ability to render pen strokes on video displays. The API also provides a set of event listeners which can be used to respond to user interface events such as pen down events, pen up events, pen movement events, and button press events. If an application needs to respond to a particular kind of event, the application must activate the appropriate listener and register an event handler for it. The completed set of listeners are described in table 3.1.

Because pen-based applications often have soft buttons, which are regions on the work area that trigger an action when tapped, the toolkit provides functionality for buttons. For hardware with graphical displays, the toolkit automatically renders soft buttons on the display. The toolkit provides event listeners that enable an application to respond to button click events. The complete set of button functions are described in the table 3.2.

Method Name	**Description**
void down(DataPointEvent)	Handles a pen down event.
void up(DataPointEvent)	Handles a pen up event.
void dragged(DataPointEvent)	Handles a drag event.
void stroke(StrokeEvent)	Handles a stroke event.
void tap(DataPointEvent)	Handles a tap event.
void doubleTap(DataPointEvent)	Handles a double tap event.
void strokeDeleting(DataPointEvent)	Occurs when a delete begins.
void strokeDeleted(DataIndexSetEvent)	Occurs when a stroke delete ends.
void rightButtonPressed(DataPointEvent)	Occurs when the main button on a pen device is pressed.
void rightButtonReleased(StrokeEvent)	Occurs when the main button on a pen device is released.
void rightButtonClicked(DataPointEvent)	Occurs when the main button on a pen device is clicked.

Table 3.1: A list of methods for event listeners.

3.3 Library

The toolkit contains a variety of recognition algorithms. It contains two single-stroke recognizers, the Rubine recognizer [27] and the dollar recognizer [31]. It also includes the multi-stroke image-based recognizer developed by Kara [14]. This multi-stroke recognizer uses four distance metrics to compare an unknown shape to the definition shapes. These four distance metrics are the Hausdorff Distance, Modified Hausdorff Distance, Tanimoto Similarity Coefficient, and Yule Coefficient. The unknown shape may not be in the same orientation as the definition shapes. Therefore, the image-based recognizer uses search to find the optimal alignment between the unknown and the templates. We have added options to turn off the rotational search and to use only one distance metric (the Hausdorff Distance). While this results in faster recognition, it also increases the possibility of recognition errors.

There are certain mathematical functions that are commonly used in processing ink, such as computing angles between vectors, and dot products between vectors. The toolkit includes a

Methods	**Description**
void setX(int)	Sets the x coordinate of the button.
void setY(int)	Sets the y coordinate of the button.
void setHeight(int)	Sets the height of the button.
void setWidth(int)	Sets the width of the button.
void setName(String)	Sets the name of the button.
UUID getUID()	Gets the unique id of the button.
void copy(Button)	Copies the properties of another Button.
boolean isPointInButton(DataPoint)	Checks if a point is within the button.
void actionEvent(ButtonEvent)	Forces event notification on all listeners to this button.
void addActionListener(ButtonActionListener)	Adds a listener to the button.

Table 3.2: A list of methods in the Button class.

variety of mathematical functions as listed in table 3.3.

Method Name	**Description**
double euclideanDistance(DataPoint, DataPoint)	Euclidean distance between two points.
double manhattanDistance(DataPoint, DataPoint)	Manhattan distance between two points.
double dot(DataPoint, DataPoint)	Dot product between two vectors.
double magnitude(DataPoint)	Magnitude of a vector.
double angleBetweenTwoVectors(DataPoint, DataPoint)	Angle between two vectors.
double[] leastSquareLine(DataPoint...)	Least squares line fit.
DataPoint closestPtBTWLineAndPt (DataPoint, DataPoint, DataPoint)	Closest point between a line and a point.
double minDistanceBTWLineAndPt (DataPoint, DataPoint, DataPoint)	Minimum distance between a line to a point.
ArrayList$< DataPoint >$ Resample(ArrayList$< DataPoint >$,int)	Resamples the ink to defined number of points.
Stroke smoothStroke(Stroke, int)	Smoothes the stroke.
ArrayList$< DataPoint >$ ScaleNonUniformly	Scales strokes non-uniformly.
void scaleStrokeByPercent(Stroke, double)	Scales a stroke to a percentage of its' size
ArrayList$< DataPoint >$ RotateBy (ArrayList$< DataPoint >$, double)	Rotate a stroke by a certain degree.
double inkLength(ArrayList$< DataPoint >$)	Compute ink length of a stroke.
DataPoint Centroid(ArrayList$< DataPoint >$)	Compute the centroid of a stroke.
Stroke dehooking(Stroke, int, int, double)	Dehooks the stroke.
Stroke reverseStroke(Stroke)	Reverses the stroke.
Stroke smoothStroke(Stroke, int)	Smoothes the stroke.
void translateStroke(double, double, Stroke)	Translate the stroke.
double getStraightnessRatio(Stroke)	Computes if the stroke is a line.

Table 3.3: A list of mathematical and stroke methods.

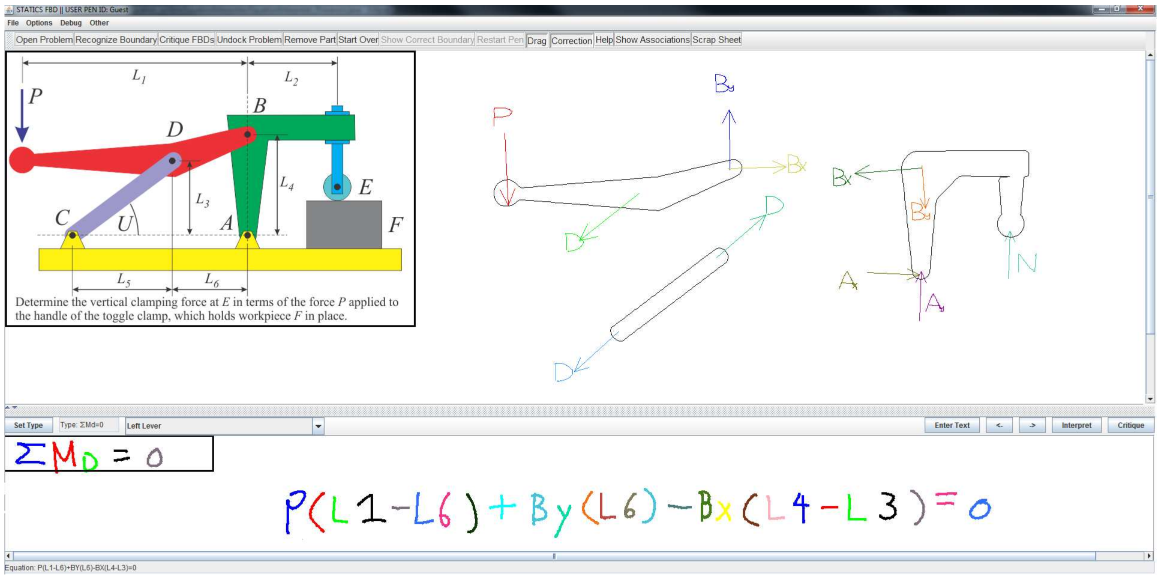
Chapter 4

Statics Tutoring System

4.1 System Overview

NP2 scaffolds students through the process of both constructing free body diagrams and constructing equilibrium equations. Figure 4.1 shows the program's user interface. The top portion of the program window contains the problem description and the free body diagram workspace. The bottom portion contains the equation workspace.

To construct a free body diagram, the student begins by identifying the boundary of the system of interest. The student does this by using the stylus to trace the boundary directly on the problem picture (Figure 4.2a). When the student then taps the "Recognize Boundary" button, NP2 identifies which objects have been included in the system boundary. If the student has selected an improper or ambiguous boundary, the system provides appropriate feedback. For example, if the boundary "cuts" through a rigid body, the student is informed that cutting through a rigid body reveals potentially complicated internal forces, and thus a different boundary should be selected. Likewise, if the student has not accurately traced a set of objects, the student is informed that the boundary is ambiguous. Once the student has defined a suitable boundary, he uses the stylus to drag



Determine the vertical clamping force at *E* in terms of the force *P* applied to the handle of the toggle clamp, which holds workpiece *F* in place.

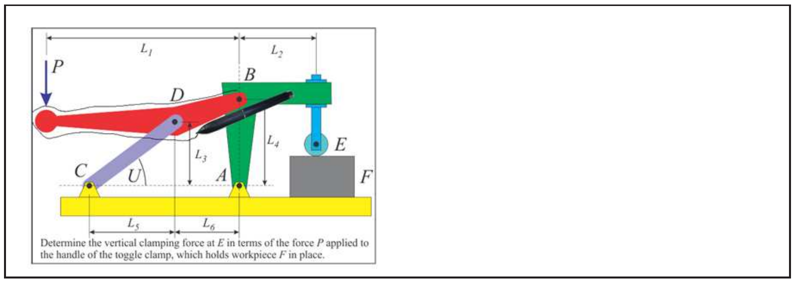
Equation: P(L1-L6)+BY(L6)-BX(L4-L3)=0

Figure 4.1: The Newton's Pen II System. The top of the application window contains the problem description and the free body diagram workspace. The bottom contains the equation workspace.

it to the free body diagram workspace (Figure 4.2b).

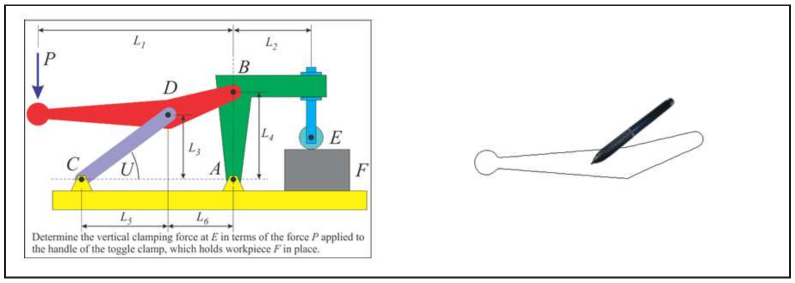
The notion of a system boundary is a critical part of the NP2's instructional design. Novice students often have difficulty defining a system, and may attempt to draw a free body diagram by simply redrawing the original problem figure, with all of its details (Figure 1.1b). Alternatively, they may draw a variety of incorrect abstractions, such as stick figures or forces floating in space, with no clear connection to objects (Figure 1.1c). NP2 is designed to overcome these misconceptions by guiding the student to explicitly identify the boundary of the system. Having such a boundary helps the student distinguish between external forces, which must be represented on the free body diagram, and internal forces which should not appear there.

To complete the free body diagram, the student draws the forces and moments, which are drawn in the usual way as straight and curved arrows, respectively. These arrows must be drawn with a single stroke. (Our current implementation allows multi-stroke arrows, but this was not used in the user study described in Section 5.1.) After drawing, the student must pause briefly to trigger



Determine the vertical clamping force at E in terms of the force P applied to the handle of the toggle clamp, which holds workpiece F in place.

(a)



Determine the vertical clamping force at E in terms of the force P applied to the handle of the toggle clamp, which holds workpiece F in place.

(b)

Figure 4.2: To construct a free body diagram, the student first (a) uses the stylus to trace the desired system boundary directly on the problem description, then (b) drags the boundary to the free body diagram workspace.

recognition. If a pen stroke is recognized as a force or moment arrow, the ink is replaced with a machine-drawn arrow to indicate the interpretation.

Forces and moments can be labeled by drawing directly on the workspace, or by using a text entry window (Figure 4.3). In the former case, the characters are recognized after a pause and the ink is replaced with its interpretation, which is rendered with a hand-drawn character font. We use this approach, rather than using a traditional computer font, to preserve the informality of a student's handwritten work. The student can correct misinterpretations by erasing and rewriting the text, or by pressing the button on the barrel of the stylus and circling the ink, triggering the display of alternative interpretations from which the student may select the intended characters.

To initiate use of the text entry window, the student simply draws an "L"-shaped pen stroke, causing the window to appear. The text entry window contains two character entry boxes. After each character is written, it is recognized and replaced with its interpretation. If the text is misinterpreted, the student can select the correct interpretation from the provided lists of alternatives.

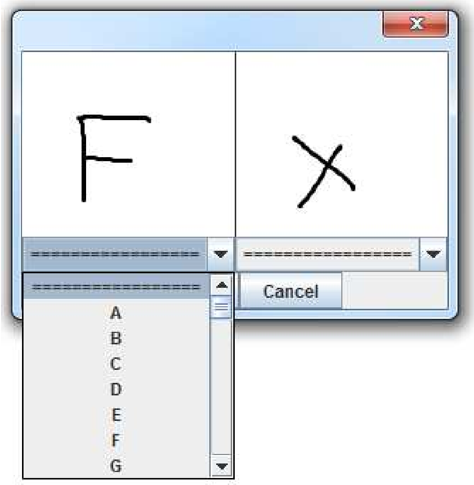


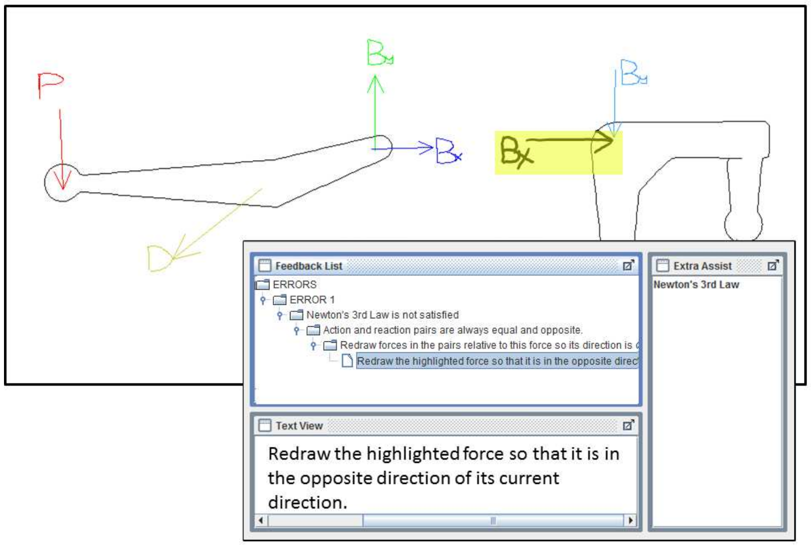
Figure 4.3: Force and moment labels can be entered using a text entry window. Alternative interpretations are provided below each handwritten character.

Once text entry is complete, the final interpretation is displayed in the workspace with a handwritten font.

NP2 matches each force and moment label with the nearest arrow. Color coding is used to indicate the matches. For forces that are not aligned with a coordinate direction, the student must specify the angle. This is done by drawing an extension line, an arc, and a single character to label the angle.

The analysis of frames and machines typically requires the construction of multiple free body diagrams. The student can construct additional free body diagrams by tracing additional boundaries, dragging them to the workspace, and labeling the forces.

Once the student has constructed all of the free body diagrams, he can obtain tutorial feedback by tapping the "Critique FBDs" button. If the student has correctly drawn all of the necessary free body diagrams, he is informed of the success and is instructed to move on to the construction of the equilibrium equations. If there are errors, the student is informed of them. The feedback system is based on a "buggy rules" approach that attempts to diagnose and correct problem-solving errors typical of novice students. As described in more detail in Section 4.4.1, the system can explain a wide range of conceptual errors such as violations of Newton's Third Law, improperly modeled reactions, and confusion about two-force members. As shown in Figure 4.4, the feedback is presented hierarchically, with general comments given first, and more specific help provided later. For example, if Newton's Third Law is violated, the student is first informed that "Newton's 3rd Law is not satisfied." The student can tap the pen on the message to receive a more specific message, such as "Action and reaction pairs are always equal and opposite." Continued requests for more details result in the program highlighting the offending force on the free body diagram and, in the example in Figure 4.4, informing the student that he should reverse the direction of the force.



ERRORS
ERROR 1
Newton's 3rd Law is not satisfied
Action and reaction pairs are always equal and opposite.
Redraw forces in the pairs relative to this force so its direction is
Redraw the highlighted force so that it is in the opposite direc

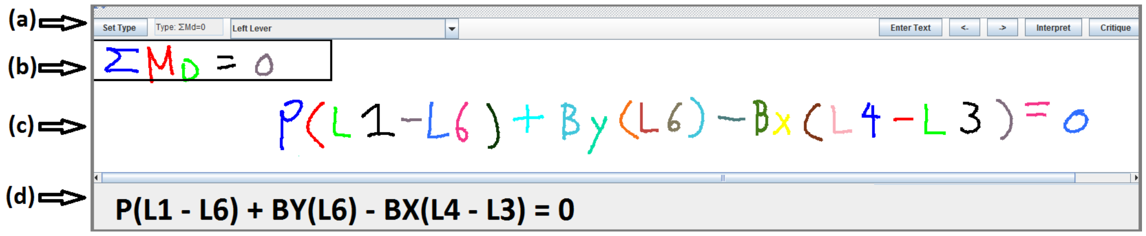
Newton's 3rd Law

Redraw the highlighted force so that it is in the opposite direction of its current direction.

Figure 4.4: Help is presented hierarchically. Top-level help messages provide general suggestions. Lower level messages are more specific. Displaying the lowest level message reveals the remedy to the error, and causes the relevant portion of the free body diagram to be highlighted (yellow rectangle).

Equilibrium equations are constructed in the equation workspace at the bottom of the program window. To begin, the student specifies which of the free body diagrams is being considered.

This is done using a pull down menu containing a list of the free body diagrams which are named according to the bodies they contain, such as “left lever.” The student then specifies the type of equilibrium equation he is constructing. For example, to indicate the x-equilibrium equation, the student writes $\sum F_x = 0$ in the “equation type” area shown in Figure 4.5b. Similarly, to indicate the moment equilibrium equations, the student writes $\sum M_p = 0$, where p is a labeled point in the problem. Tapping the “Set Type” button triggers recognition of the equation type.



P(L1 - L6) + BY(L6) - BX(L4 - L3) = 0

Figure 4.5: The equation workspace: (a) action buttons, (b) equation type area, (c) equation area, (d) equation interpretation area. The ink in the equation is color coded to display the grouping of the strokes into characters.

After specifying the equation type, the student writes the equation itself in the equation area (Figure 4.5c). The student triggers recognition by tapping the “Interpret” button. The program then displays the interpretation in the equation display area (Figure 4.5d). If there are interpretation errors, the student can correct them using the correction methods described earlier. When the student has confirmed that the program has correctly interpreted the equation, he can tap the “Critique” button to obtain feedback, which is again presented hierarchically.

To construct additional equations, the student taps the next page (“$\rightarrow$”) button, which presents a new equation area. The student can use the next page and previous page (“$\leftarrow$”) buttons to navigate through his work.

The next section describes the techniques that NP2 uses to interpret a student’s work, including the techniques for interpreting the system boundary, and the techniques for recognizing arrows and text. This is followed by a description of the tutorial feedback system.

4.2 Interpreting System Boundaries

To identify the bodies contained in the student's system boundary, NP2 first converts the strokes that comprise the boundary into a polygon. It then computes the intersection between this boundary polygon and the polygons of the bodies in the problem. If this calculation reveals that most of a given body is inside the boundary polygon, then that body is considered to be inside the boundary.

We have found that boundaries are often drawn with multiple pen strokes. If a boundary is drawn with a single pen stroke, then a polygon could be formed trivially by connecting the end points. When multiple pen strokes are used, however, it is necessary to merge the overlapping regions together. There are many contour identification techniques designed for bitmaps that could be used for this task. However, since our contour is comprised of strokes rather than a collection of pixels, we can create more efficient methods. Speed is particularly important for an online application such as ours.

We need to merge only the ends of the pen strokes as this is where the discontinuities occur. Two stroke ends are considered for merging, only if they are sufficiently near each other. We have found that users often draw all of the strokes in the boundary in the same sense. For example, when drawing a square with four pen strokes, all would be drawn clockwise or all would be drawn counterclockwise. Thus, our algorithm considers merging portions of two strokes only when their

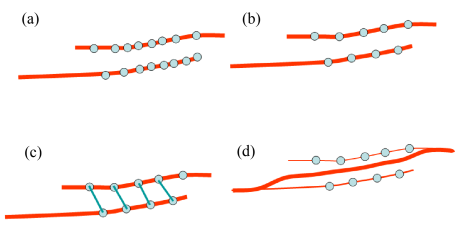


Figure 4.6: Stroke merging. (a) raw data points. (b) Resampled data points. (c) Aligned points on one stroke with those on the other. (d) Merged stroke constructed by merging matched points.

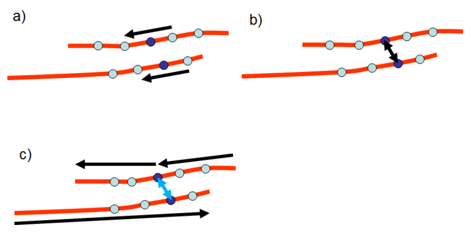


Figure 4.7: Stroke merging criteria. (a) Tangent angle must differ by no more than 10°. (b) Euclidean distance must be less than 35 pixels. (c) Path length must be greater than at least twice the Euclidean distance.

tangent directions are similar. Figure 4.6 shows the steps for merging.

To reduce computation time, the algorithm begins by resampling each stroke to a maximum of 128 equidistant points. To reduce noise, the resampled data points are smoothed by averaging the location of each point with the locations of the point on each side. Once the ink is resampled, the algorithm considers only the first and last 10% of the stroke for possible merging. For a point on one stroke to be merged with a point on another stroke (or a point on the other end of the same stroke) to be merged, the following criteria must be satisfied (Figure 4.7): First, the tangent directions of the two points must differ by no more than 10°. Second, the Euclidean distance between the two points must be less than 35 pixels. Finally, if the two points are on the same stroke, the Euclidean distance between them must be at least twice the arc length that separates them. The last criteria prevents a point on a stroke from being merged with its immediate neighbors, while allowing opposite ends of the stroke to be merged.

To determine if a particular set of bodies is included in the student's boundary polygon, NP2 first constructs a "nominal polygon" by merging the boundaries of the components. It also constructs a"minimum polygon" and a "maximum polygon" by computing polygons offset from the nominal polygon. The maximum polygon completely contains the nominal polygon, which in turn completely contains the minimum polygon. NPT generates this set of polygons for each combination of components.

A set of components is considered to be selected by the boundary polygon if: (a) the boundary polygon contains at least 80% of their nominal polygon, (b) their entire minimum polygon is contained within the boundary polygon, and (c) the boundary polygon does not contain their entire maximum polygon.

While a system boundary should not cut through a rigid body, it is acceptable for the boundary to cut through a flexible element, such as a cable. If at least 10% of a flexible element is contained within the boundary polygon, the portion inside the boundary is considered to be selected.

4.3 Recognition of Force and Moment Arrows

After the FBD boundary has been dragged into the FBD workspace, the student draws arrows to represent forces and moments, with straight arrows for the former and curved ones for the latter. NP2 is capable of recognizing arrows drawn tail to head with either one or two strokes. The system accomplishes this by combining two different arrow recognizers: an *inverse-curvature* recognizer and a *dollar arrow* recognizer.

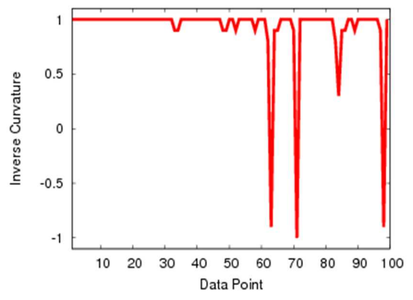


Figure 4.8: Inverse curvature of an arrow. The three corners of the arrowhead are the three smallest values of inverse curvature. Image from [18].

The inverse-curvature recognizer identifies arrows by the three corners typically found on arrowhead. With this technique, the stroke is first resampled to 128 points to reduce computation time and then smoothed to remove noise. Next, line segments are constructed between each pair of consecutive points. The “inverse curvature” at a point is defined as the cosine of the angle between the two adjacent segments. If the adjacent segments are nearly collinear, the inverse curvature value is close to one. If the segments make a sharp angle, the inverse curvature value is much smaller than one. In the case of a 180° bend, for instance, the inverse curvature is negative one. The three corners of an arrowhead can be detected by their small inverse curvature values (Figure 4.8).

The inverse-curvature recognizer uses a neural network to classify pen strokes. The 128 inverse curvature values for the resampled pen stroke serve as the features to the classifier. The neural network was trained with approximately 100 arrows and 600 non-arrows (letters and numbers) from 5 different users. (The complete details can be found in [17].)

The dollar arrow recognizer is a modified version of the dollar recognizer [31]. The dollar recognizer has difficulty recognizing arrows, because the shafts may be arbitrarily long. We have modified the algorithm to ignore most of the shaft. To do this, the dollar Arrow recognizer uses inverse curvature to identify the three corners of the arrowhead. The corners are used to compute the length of the line segments forming the arrowhead. If the shaft (i.e., the “non-arrowhead” portion of the arrow) is longer than twice this average length of the arrowhead segments, the shaft is clipped to this length, otherwise the entire shaft is used. After this preprocessing, recognition continues with the usual dollar recognizer approach.

Both of these recognizers accepts a stroke as input and returns a confidence value between $[0, 1]$ indicating the likelihood that the stroke is an arrow. The confidence values of the two recognizers are combined by taking their average. If this average is greater than 0.8, the stroke is classified as an arrow. Both the inverse-curvature and dollar arrow recognizer are designed for single strokes. To use them for multi-stroke stroke arrows, the end of the first stroke is joined to the start of the second stroke. Arrows with more than two strokes can be processed in a similar fashion, but in this case, it is useful to consider all permutations of stroke order when appending strokes.

To determine if an arrow is a force or a moment, NP2 fits a linear least squares line to the shaft of the arrow. If the error of fit is less than a threshold, the arrow is considered to be a force, otherwise it is considered to be a moment.

4.3.1 Force Labels

Forces and moments can be labeled by drawing directly on the workspace, or by using a text entry window. In the former case, the characters are recognized with the Microsoft Handwriting Recognizer (MHR) [23] after a user-adjustable timeout. The ink is then replaced with its interpretation, which is rendered with a hand-drawn character font. We use this approach, rather than using a traditional computer font, to preserve the informality of a student's handwritten work. The student can correct misinterpretations by erasing and rewriting the text, or by pressing the button on the barrel of the stylus and circling the ink, triggering the display of alternative interpretations from which the student may select the intended characters.

To initiate the use of the text entry window, the student simply draws an "L"-shaped pen stroke, causing the window to appear. The text entry window contains two character entry boxes, as force labels are limited to two characters. After each character is written, it is recognized using the MHR and replaced with its interpretation. If the text is misinterpreted, the student can select the correct interpretation from the provided lists of alternatives. Once text entry is complete, the final interpretation is displayed in the workspace with a handwritten font.

4.4 Tutorial Feedback

NP2 interprets the student's handwritten solutions to statics problems and provides tutorial feedback when there are problem-solving errors. The system employs two separate feedback systems, one for free body diagrams, and one for equilibrium equations. In the sections below, we describe the workings of these two systems.

4.4.1 Critiquing Free Body Diagrams

To interpret the system boundary that the student has traced, the program first joins the pen strokes to form a closed region. The program then computes the intersection between this region and each of the bodies in the problem figure. If at least 80% of the area of a body is contained within the region, it is considered to be included in the system boundary. If less than 10% is contained within the region, the body is considered to be outside the system boundary. Otherwise, it is assumed that the boundary cuts through the body, which would reveal internal forces. In this case, the student is immediately instructed to draw a new boundary to avoid this problem. If a significant amount of the region intersects no bodies (i.e., the boundary contains significant empty space), the boundary does not accurately define a system, and the student is immediately instructed to trace the boundary more carefully.

When the student has completed the free body diagrams for a problem, he can tap the "Critique FBDs" button to obtain feedback. The program generates the feedback by using a set of solution templates and a set of "buggy rules". NP2 contains multiple solution templates representing common correct and incorrect solutions for each problem. Each template includes the properties of the forces and their expected locations. Forces are represented by a variety of properties such as the source of the force, which is either reaction or applied, the direction, which may be either known or unknown, and if the force is applied, the name of the force. Reaction forces are further described in terms of the physical constraint that produces them, such as a pivot, a roller joint, or a flexible element. If a body is a two-force member, the two force locations are labeled as such. Often, students incorrectly assume that all bar-shaped components are two-force members. Sometimes, students even incorrectly model bar-like protrusions as two-force members. To help the program diagnose such errors, force locations that have a bar-like shape are labeled as such. For example, the three ends of a "T"-shaped body would be labeled as having a bar-like shape.

NP2's "buggy rules" encompass a wide range of conceptual errors of the sort that novice students typically make. Some are specialized to problems that require only a single free body diagram for the solution, while most apply to frame and machine problems requiring multiple free body diagrams. Each rule contains multiple levels of tutorial feedback that progresses from general suggestions to the actual remedy for the error.

NP2's rule base contains a total of 61 rules for free body diagram errors. Here we summarize the major conceptual errors these rules diagnose:

- Incomplete formulation, single free body diagram problem: For problems requiring only a single free body diagram, the free body diagram selected does not include all of the forces that must be solved for. The student is instructed to select another system.

- Statically indeterminate, single free body diagram problem: For problems requiring only a single free body diagram, the free body diagram contains too many unknowns and thus is unsolvable. The student is instructed to select another

  system.

- Statically indeterminate composite free body diagram: The student has drawn only the free body diagram for the entire system and it is statically indeterminate. The student is instructed to decompose the system into multiple free body diagrams.

- Incomplete decomposition: The student has decomposed the problem into multiple free body diagrams, but one or more of them is statically indeterminate. The student is instructed to decompose the problematic free body diagrams into smaller systems.

- Missing free body diagrams: The student has decomposed the problem into multiple free body diagrams, but additional free body diagrams are needed.

- Pivot support modeled improperly: One or more of the two force components is missing or the components are not orthogonal.

- Roller modeled incorrectly: The normal force is missing, the normal force is not actually normal to the surface, or a friction force is included.

- Incorrect applied force: An applied force (as opposed to a reaction force) is missing, in the wrong direction, or has the wrong label. If a force is defined explicitly in the problem, then the student should have drew the force at that location. Additionally, students are warned if a weight force is labeled “m”, as students often confuse mass and weight. Similarly, students are also warned if a weight force is labeled “Wg.” For loads, students will be informed to draw the load so that it is normal to the surface. Additionally, students will be informed if they included more than one force or attempted to represent load as force components.

- Flexible element modeled incorrectly: The force is missing, is not aligned with the element, or is not in tension. Also, students are warned if they treated the flexible element as a pivot by decomposing into force components.

- Rough surface contact modeled incorrectly: Either the friction force, normal force, or both are missing or in the wrong direction. Furthermore, the friction and normal force should be orthogonal to each other.

- Internal Forces: The free body diagram includes forces internal to the system.

- Two-force member: One or both of the forces on the ends of the two-force member are missing, they are not in opposite directions, or they have different labels. Alternatively, there is one or more extra force.

- Newton's Third Law error: A pair of reaction forces does not satisfy the third law of action and reaction. Rules include one reaction has more components than the other, the forces are not in opposite directions, or the forces have been given different labels.

- A multi-force body confused with a two-force member: If a multi-force body is modeled as having exactly two equal and opposite forces at locations that have a bar-like shape, the student may have considered the body to be a two-force member. The student is informed that not all bodies with a bar-like shape are two-force members and that they may have confused the body with a two-force member.

- Missing forces: Any missing forces not explained by other rules are simply reported as missing forces.

- Extra force: Any extra forces not explained by other rules are simply reported as extra forces.

4.4.2 Critiquing Equations

The first step in interpreting an equation is to group the pen strokes into individual characters. NP2 does this by using a statistical grouping algorithm based on the one reported in [22]. As shown in Figure 4.5, the program renders each character with a unique color, enabling the student to detect any grouping errors. Next, the handwriting recognizer built into the Microsoft Windows operating system [24] is used to recognize the individual characters.

It is difficult to obtain high accuracy when recognizing individual characters. Handwriting recognizers (i.e., "word recognizers") overcome recognition errors by using a lexicon. Such an approach would not work for a tutoring application, because it would bias the system to correct answers, potentially causing the system to overlook actual problem-solving errors. Instead, we have developed a novel error correction technique that uses a hidden Markov model derived from a grammar for legal equations and statistics about how the underlying character recognizer misclassifies characters. This approach, which is described in more detail in [12] has proven to be very effective at correcting interpretation errors.

Once an equation has been interpreted, a heuristic matching algorithm is used to match the terms in the student's equations with the most likely intended terms in the correct solution. The heuristics consider which parts of an equation term are most indicative of intent. For example, the force label is a better indicator than the trigonometric component. Consider a situation in which the student has written "$Fcos(U)+Psin(U) = 0$" and the correct equation is "$Fsin(U)+Pcos(U) = 0$". In this case the student's first term is likely intended to represent the first term in the correct equation, despite the fact that the trigonometric component matches the second term in the correct equation.

After the terms have been matched, a set of "buggy rules" is used to provide feedback for any differences. Again, help is provided hierarchically, progressing from general guidance to specific remedies. Also, the guidance is provided in terms of statics concepts, rather than a simple enumeration of the differences between the student's equation and the correct one. For example, if in the x-equilibrium equation the students has written "$Fcos(U)$" but "$Fsin(U)$" is expected, the system informs the students that "the wrong component has been selected for force F". More specific feedback instructs the student to "select the component parallel to the x-direction."

The "buggy rules" consider a variety of errors including: missing forces, force components, and moments; extra forces, force components, and moments; incorrect force components (wrong trigonometric function); missing or incorrect moment arms; and sign errors. (See [12] for complete details of the equation critiquing process.)

Chapter 5

User Study

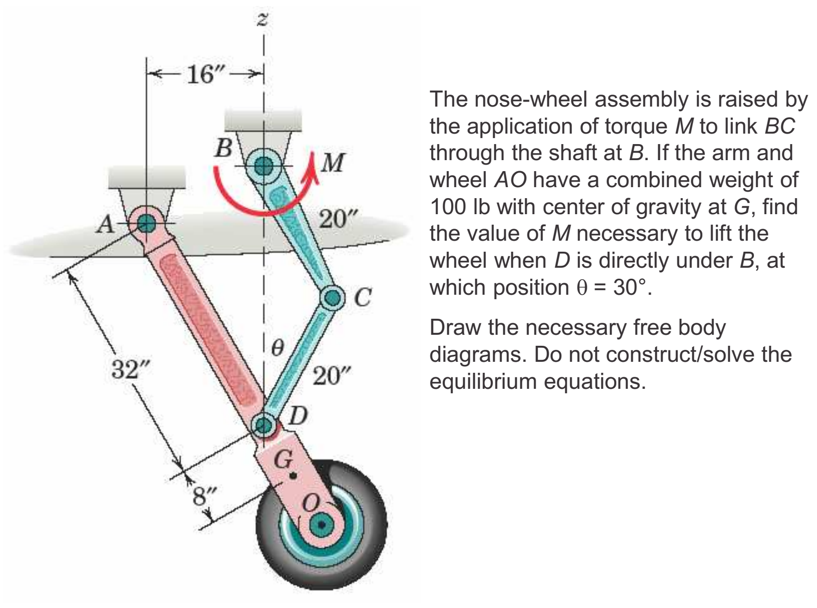
5.1 System Evaluation

NP2 was integrated into the sophomore-level engineering statics course at University of California, Riverside in the winter quarter of 2010. The class had an enrollment of just over 100 students. In a survey in which 95 students responded, 91 of the students reported being Mechanical Engineering majors. Furthermore, 84 students reported being male, and 11 reported being female.

Students used NP2 in their discussion sections four times during the quarter. In the first session, students used the system to learn to construct free body diagrams for problems involving the analysis of a single 2D body. In the second session, students used the system to construct both free body diagrams and equilibrium equations for single 2D bodies. In the third session, students used the system to construct free body diagrams for multi-body frame and machine problems. Finally, in the fourth session, students used the system to construct both free body diagrams and equilibrium equations for frame and machine problems.

To evaluate NP2's educational usefulness, we used pre- and post-tests during the third session to measure learning gains for the construction of free body diagrams for frame and machine problems. We did not measure gains for equilibrium equations, as the 50-minute discussion sections were not long enough for students to both complete pre- and post-tests and use the system to complete a tutorial problem.

Our study employed two test problems. Problem A (Figure 5.1) concerns the analysis of an airplane landing gear. Problem B (Figure 5.2) concerns an automobile hoist. Half of the students used Problem A for the pre-test, and B for the post-test. The other half solved the problems in the reverse sequence. All students used NP2 to complete a tutorial problem concerning a toggle clamp (Figure 4.1). Because of the limited number of tablet PCs available, students worked in pairs while completing the tutorial.

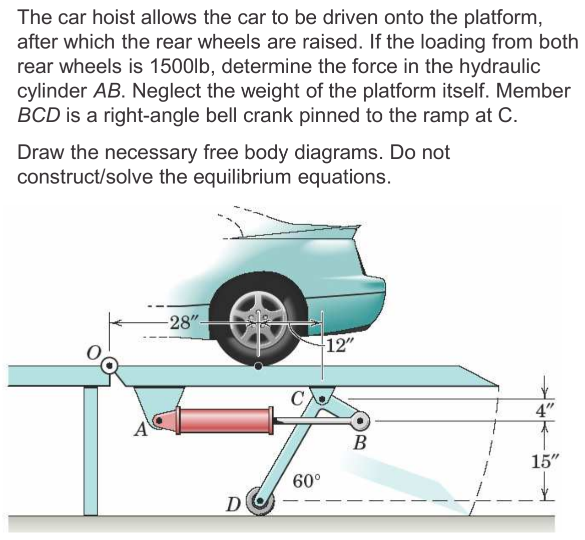


The nose-wheel assembly is raised by the application of torque *M* to link *BC* through the shaft at *B*. If the arm and wheel *AO* have a combined weight of 100 lb with center of gravity at *G*, find the value of *M* necessary to lift the wheel when *D* is directly under *B*, at which position $\theta = 30°$.

Draw the necessary free body diagrams. Do not construct/solve the equilibrium equations.

Figure 5.1: Problem A used for pre- and post-test evaluation. From [20].

Students completed the tests using an Anoto-based Livescribe digital pen and paper. This enabled us to accurately measure the amount of time it took each student to complete the tests. Some students did not have their digital pen available to complete the tests. In such cases, their test data was excluded from the analysis. Overall, the analysis included 36 students who had A as a



The car hoist allows the car to be driven onto the platform, after which the rear wheels are raised. If the loading from both rear wheels is 1500lb, determine the force in the hydraulic cylinder *AB*. Neglect the weight of the platform itself. Member *BCD* is a right-angle bell crank pinned to the ramp at C.

Draw the necessary free body diagrams. Do not construct/solve the equilibrium equations.

Figure 5.2: Problem B used in pre- and post-test evaluation. From [20].

pre-test and B as a post-test, and 40 students who had the reverse problem order.

5.1.1 Measuring Learning Gains

We measured learning gains by measuring performance on the pre- and post-test in terms of problem-solving errors. We consider eight major kinds of conceptual errors:

- Roller modeled incorrectly: The normal force is missing, the normal force is not actually normal to the surface, or a friction force is included.
- Pivot modeled incorrectly: One or more of the two force components for the pivot support is missing.
- Weight force incorrect: The weight force is missing or in the wrong direction.
- Moment modeled incorrectly: On Problem A, the moment is not included on the free body diagram, or is drawn in the wrong direction.

- Two-force member error: There is no free body diagram for the two-force member (and a correct model of the two-force member is not implied by the other free body diagrams); there is a free body diagram but the body is modeled with two pivots each having two unknown force components; or there is an apparent attempt to model the body as a two-force member, but forces are not equal and opposite.

- Third law error: A pair of reaction forces does not satisfy the third law of action and reaction. For example, one reaction has more components than the other, the forces are not in opposite directions, or the forces have been given different names.

- Internal forces included: Forces internal to the system are erroneously included on the free body diagram. For example, the free body diagram represents two connected objects, and a force is included at the joint connecting the two.

- Other extra forces: The free body diagram erroneously contains other forces not considered in the other error categories.

These errors are assessed for each free body diagram necessary to solve the problem. For Problem A, we observed three different correct solutions. The first included three free body diagrams, one for member AO, one for member BC, and one for member CD. The second is similar to the first, except that the free body diagram for CD was implicit: this diagram was not drawn, but the other two free body diagrams were consistent with CD being correctly modeled as a two-force member. The third solution included only two free body diagrams, one for AO and one for BCD (i.e., the combination of two members). Even for the third approach, a correct solution requires the student to model point D as the end of a two-force member. If the student omitted one of the free body diagrams necessary for the solution approach selected, the maximum possible number of errors was assessed. For example, if the free body diagram for BC was omitted, three errors were assessed: an incorrectly modeled pivot, a missing moment, and a violation of the third law.

For Problem B, we again observed three different correct solutions. The first included three free body diagrams, one for member OAC, one for member BCD, and one for member AB. The second is similar to the first, except that the free body diagram for AB was implicit. The third solution included only two free body diagrams, one for OAC and one for ABCD (i.e., the combination of two members). For the third approach, a correct solution requires the student to model point A as the end of a two-force member. As before, if the student omitted one of the free body diagrams necessary for the solution approach selected, the maximum possible number of errors was assessed.

Many students began the solution of the problems by constructing a free body diagram for the complete system containing all of the members. This "composite" free body diagram was unnecessary for a correct solution, thus if it was not drawn, no errors were assessed. Because many students did not draw the composite free body diagram, we tabulated errors for it separately from those for the other free body diagrams.

After examining the tests, we discovered that students performed substantially better on Problem A than on B. (Comparison of Figures 5.3 and 5.4 shows that students made far fewer errors on Problem A than on B). It appears that the latter problem was more difficult for the students. As a result, making within-subject comparisons of student performance from pre- to post-test does not produce useful insights: Students who had A as a pre-test faced a harder post-test, while those who had B as a pre-test faced an easier post-test. Thus, we find it more useful to examine across-subject performance, comparing the performance on a problem as pre-test, to the performance on the same problem as a post-test. Note that in this analysis, we are comparing the performance of two different groups of students on the same problem.

The performance results for Problem A are shown in Figure 5.3, while those for Problem B are shown in Figure 5.4. The errors for the required free body diagrams are averaged over the number of students who were assigned the particular problem (i.e., those who did not complete the tests with a digital pen were excluded). The errors for the composite free body diagrams, which are unnecessary for a correct solution, are averaged over the number of students who attempted to draw them. Coincidentally, for both problems, 70% of students drew composite free body diagrams.

Examination of Figures 5.3 and 5.4 reveals clear learning gains from pre- to post-test. There is a reduction in the average number of errors for nearly every error measure. Error reductions are largest for Newton's Third Law errors, pivot reaction force errors, and erroneous internal forces on the composite free body diagrams. On Problem A, there is a substantial increase in the number of extra forces erroneously included in the free body diagrams. This appears to be due to a misunderstanding of the problem. After completing the tutorial problem with NP2, some students modeled the wheel of the landing gear as a roller with a normal force. However, as the wheel is not touching the ground, there should be no force there.

In addition to the gains in solution correctness, there were also gains in solution speed from pre- to post-test as illustrated in Table 5.1. Students required 32% less time to solve Problem A as a post-test than as a pre-test. Similarly, they required 24% less time to solve Problem B as a post-test than as a pre-test.

Problem	**Pre-test Time**	**Post-test Time**	**Reduction**	**Reduction (%)**
A	7.17	4.90	2.28	32%
B	7.33	5.57	1.76	24%

Table 5.1: Average solution time for pre- and post-test problems in minutes, along with the reduction of average time from pre- to post-test in both minutes and %.

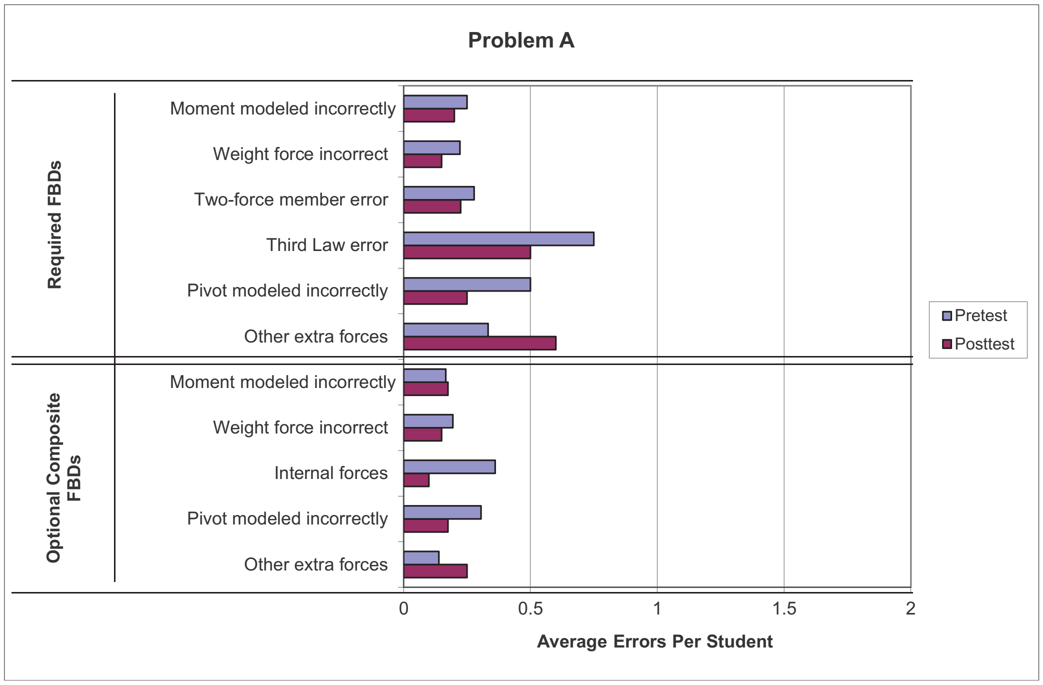


Figure 5.3: Results for Problem A as both a pre- and post-test.

5.1.2 Attitudinal Survey

All questions were answered on a scale from one to ten, with one being the most negative response and ten being the most positive. The median response on all questions was on the positive side of the scale except for two of the questions. Students experienced problems with the system's recognition and interpretation functionality, which likely led to some frustration. We observed that some of the frustration was due to the system's conventions. For example, some students drew arrows head to tail rather than tail to head as the program expected. Similarly, some students wrote equations in unexpected ways, for example, writing the word "SUM" rather than the symbol "Σ" for the equation type.

At the end of the quarter, students were asked to complete a 25 question survey of their opinions about NP2. The survey questions and responses are listed in Table 5.2 in which students rated various aspects of the system including ease of drawing, usability, educational usefulness, and

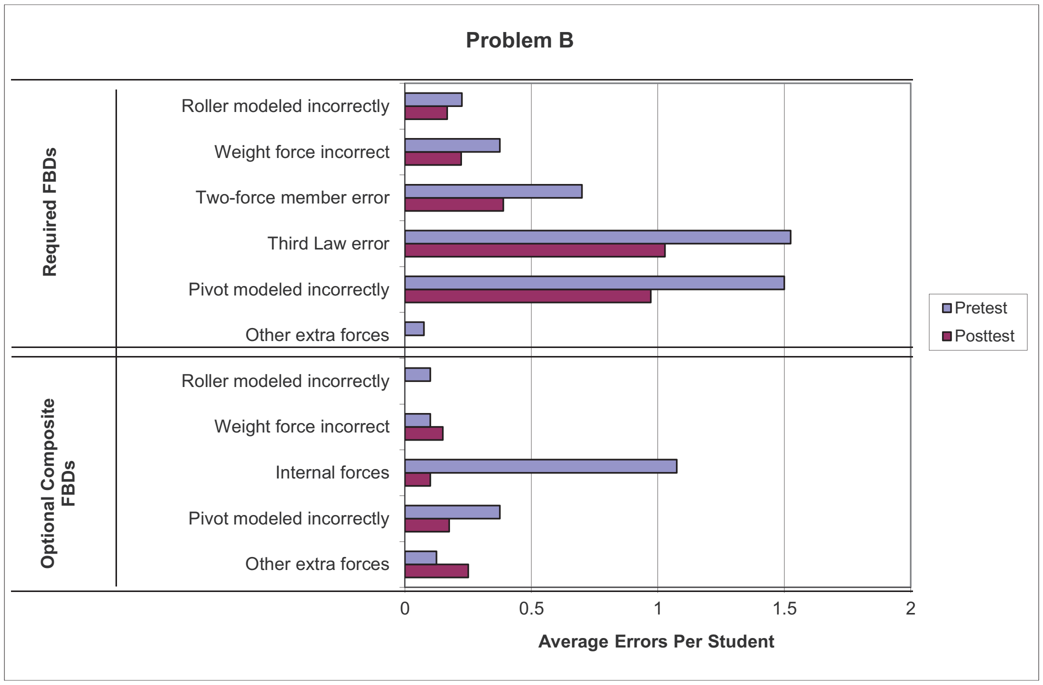


Figure 5.4: Results for Problem B as both a pre- and post-test.

overall reaction. The questions for the latter topic were taken from the user study reported in [13]. The first 11 questions were related to the usability of the system, and considered ease of drawing, recognition accuracy, and overall usability. In one question, students were asked to compare the interface to a traditional WIMP interface. Their answer was based on their general familiarity with WIMP interfaces as the students were not provided with a WIMP-based statics tutoring system. The next 10 questions focused on the usefulness of the system for learning various statics concepts and the overall instructional usefulness. The final four questions concerned the students' overall reaction to the system.

Because this was the first deployment of NP2, students also discovered some bugs in the recognition algorithms. These issues were quickly addressed resulting in releases of improved versions of the system throughout the quarter. We believe that the survey results were negatively affected by students' experiences with the earlier versions of the system. We expect that future deployments of the improved system will result in more favorable opinions.

	Survey Question	Scale	Ave	Med
1	How easy is it to trace and drag system boundaries?	Hard (1) — Easy (10)	6.8	7
2	How easy is it to draw force arrows?	Hard (1) — Easy (10)	6.2	6
3	How easy is it to label force names?	Hard (1) — Easy (10)	5.6	6
4	How easy is it to draw correctly-interpreted equation types?	Hard (1) — Easy (10)	6.4	7
5	How easy is it to write correctly-interpreted equations?	Hard (1) — Easy (10)	5.5	6
6	How easy is it to correct interpretation errors in equations?	Hard (1) — Easy (10)	5.9	6
7	Are the program's interpretation capabilities sufficiently accurate?	Insufficient (1) — Sufficient (10)	5.2	5
8	How easy is it to learn to use the program?	Easy (1) — Hard (10)	2.9	2
9	What is your overall opinion of the usability of this program?	Not usable (1) — Very usable (10)	6.1	6
10	How similar is this system to working with paper and pencil?	Very Dissimilar (1) — Very similar (10)	5.3	5
11	Is this interface preferable to WIMP interfaces?	Prefer traditional interface (1) — Prefer pen-based interface (10)	6.5	7
12	Is this program's tutorial feedback presented in a usable form?	Not usable (1) — Very usable (10)	6.8	7
13	How useful is this for learning selection of system boundaries?	Not useful (1) — Very useful (10)	6.9	7
14	How useful is this for identifying forces on free body diagrams?	Not useful (1) — Very useful (10)	7.1	7
15	How useful is this for application of Newton's 3rd Law?	Not useful (1) — Very useful (10)	6.8	7
16	How useful is this for analysis of two-force members?	Not useful (1) — Very useful (10)	7.0	7
17	How useful is this for force equations?	Not useful (1) — Very useful (10)	6.4	7
18	How useful is this for moment equations?	Not useful (1) — Very useful (10)	6.5	7
19	What is your overall opinion of the usefulness of this program in learning to solve equilibrium problems?	Not useful (1) — Very useful (10)	6.5	7
20	How likely would you be using this system to study for statics courses?	Unlikely (1) — Likely (10)	6.1	7
21	How likely would you be using this kind of system for other subjects?	Unlikely (1) — Likely (10)	6.2	7
22	Rate your overall reaction to the system.	Terrible (1) — Wonderful (10)	6.2	6
23	Rate your overall reaction to the system.	Difficult (1) — Easy (10)	6.7	7
24	Rate your overall reaction to the system.	Frustrating (1) — Satisfying (10)	5.4	5
25	Rate your overall reaction to the system.	Dull (1) — Stimulating (10)	6.4	7

Table 5.2: Survey of student opinions about Newton's Pen II. All questions are answered on a scale from one to ten, with one being the most negative response and ten being the most positive.

Chapter 6

Conclusion

We have developed Newton's Pen II, a pen-based tutoring system for engineering statics. The tutoring system scaffolds students in the construction of free-body diagrams and equilibrium equations for planar devices comprised of one or more rigid bodies.

Different pen-based hardware platforms have different application programming interfaces (APIs). Therefore, applications designed for one platform will typically have to be rewritten to work on another. As an alternative, we created a toolkit that provides a standard API that abstracts away the details of the hardware-specific APIs. A software developer can create an application using the standard API and the toolkit can automatically translate the API calls to work with the device-specific APIs. The toolkit also provides a variety of functions that are typically used when building pen-based applications such as shape recognition and stroke resampling. By building NP2 with this toolkit, we will be able to easily port it to other pen-based platforms, such as the digital pen and paper platform, enabling us to evaluate the suitability of these platforms for tutoring applications.

Our tutorial system embodies several innovations: (1) a novel instructional technique that focuses students' attention on a system boundary as a tool for constructing free body diagrams. (2) A tutorial feedback system based on "buggy rules" that attempts to diagnose and correct problem-solving errors typical of novice students. (3) A progressively more detailed feedback system which promotes independent problem-solving skills.

In the evaluation of the system, we have deployed the system in the an undergraduate statics course with an enrollment of just over 100 students. We administered tests before and after the student's used the tutorial system to determine whether the system helped students construct free-body diagrams for frame and machine problems.

We found that after using the system to solve only a single tutorial problem, there were significant learning gains from pre- to post-test. Furthermore, in a formal survey, students expressed somewhat favorable opinions about the usefulness of the system for learning statics and for the user interface design. However, students did experience frustration with some of the program's features, particularly its interpretation capabilities.

We are encouraged by the results of this first deployment of the Newton's Pen II system in a large undergraduate course. The students did achieve significant learning gains after only a brief exposure to the system. Nevertheless, this is a first step, and there are clear opportunities for improving the system.

Bibliography

[1] Christine Alvarado and Randall Davis. Resolving ambiguities to create a natural sketch based interface. In *Proceedings of Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pages 1365–1371, 2001.

[2] Richard Anderson, Luke McDowell, and Beth Simon. Use of classroom presenter in engineering courses. In *Proceedings of Frontiers in Education '05*, pages T2G–13–18, 2005.

[3] Anoto. Development Guide for Service Enabled by Anoto Functionality, February 2006.

[4] Lisa Anthony, Jie Yang, and Kenneth R. Koedinger. Evaluation of multimodal input for entering mathematical equations on the computer. In *CHI '05 extended abstracts on Human factors in computing systems*, 2005.

[5] Ronald Chung, Petrut Mirica, and Beryl Plimmer. Inkkit: a generic design tool for the tablet pc. In *CHINZ '05: Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction*, pages 29–30, New York, NY, USA, 2005. ACM.

[6] Ruwanee de Silva, David Tyler Bischel, WeeSan Lee, Eric J. Peterson, Robert C. Calfee, and Thomas F. Stahovich. Kirchhoff's pen: a pen-based circuit analysis tutor. In *SBIM '07: Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, pages 75–82, New York, NY, USA, 2007. ACM.

[7] Robert G. Farrell, John R. Anderson, and Brian J. Reiser. An interactive computer-based tutor for LISP. In *Proceedings of the Fourth Annual Conference on Artificial Intelligence (AAAI'84)*, pages 106–109, 1984.

[8] Kenneth D. Forbus, Ronald W. Ferguson, and Jeffery M. Usher. Towards a computational model of sketching. In *Proceedings of the 6th International Conference on Intelligent User Interfaces*, pages 77–83, 2001.

[9] Richard M. Guest. Pentools - a matlab toolkit for on-line pen-based data experimentation. In *ICDAR '09: Proceedings of the 2009 10th International Conference on Document Analysis and Recognition*, pages 1221–1225, Washington, DC, USA, 2009. IEEE Computer Society.

[10] Tracy Hammond and Randall Davis. Tahuti: A geometrical sketch recognition system for UML class diagrams. In *Proceedings of 2002 AAAI Spring Symposium on Sketch Understanding*, pages 59–68, 2002.

[11] James I. Hong and James Landay. A toolkit supporting pen-based interfaces. Technical report, Berkeley, CA, USA, Berkeley, CA, USA, 2002.

[12] Josiah William Jordan. An Intelligent Tutoring System For Static Equilibrium Equations. Master's thesis, University of California, Riverside, USA, March 2011.

[13] Joseph J. LaViola Jr. An initial evaluation of MathPad^2: A tool for creating dynamic mathematical illustrations. *Computers & Graphics*, 31(4):540–553, 2007.

[14] Levent Burak Kara. An image-based trainable symbol recognizer for sketch-based interfaces. In *in AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural*, pages 99–105. AAAI Press, 2004.

[15] James A. Landay. Silk: sketching interfaces like krazy. In *CHI '96: Conference companion on Human factors in computing systems*, pages 398–399, New York, NY, USA, 1996. ACM.

[16] James A. Landay and Brad A. Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer*, 34(3), 2001.

[17] WeeSan Lee, Ruwanee de Silva, Eric J. Peterson, Robert C. Calfee, and Thomas F. Stahovich. Newton's pen - a pen-based tutoring system for statics. In *2007 Eurographics Workshop on Sketch-Based Interfaces and Modeling*, August 2007.

[18] WeeSan Lee, Ruwanee de Silva, Eric J. Peterson, Robert C. Calfee, and Thomas F. Stahovich. Newton's pen: A pen-based tutoring system for statics. *Computers & Graphics*, 32(5):511 – 524, 2008.

[19] Nicholas Matsakis. Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.

[20] J.L. Meriam and L.G. Kraige. *Engineering Mechanics - Statics*. John Wiley & Sons, Inc., 6th edition, 2007.

[21] Sharon Oviatt, Alex Arthur, and Julia Cohen. Quiet interfaces that help students think. In *Proceedings of the 19th annual ACM symposium on User interface software and technology (UIST '06)*, pages 191–200, 2006.

[22] Eric Jeffrey Peterson, Thomas F. Stahovich, Eric Doi, and Christine Alvarado. Grouping strokes into shapes in hand-drawn diagrams. In Maria Fox and David Poole, editors, *AAAI*. AAAI Press, 2010.

[23] James A. Pittman. Handwriting recognition: Tablet pc text input. *Computer*, 40:49–54, 2007.

[24] James A. Pittman. Handwriting recognition: Tablet PC text input. *Computer*, 40(9):49–54, September 2007.

[25] Robert J. Roselli, Larry Howard, Bryan Cinnamon, Sean Brophy, Patrick Norris, Megan Rothney, and Derek Eggers. Integration of an interactive free body diagram assistant with a courseware authoring package and an experimental learning management system. In *Proceedings of American Society for Engineering Education Annual Conference & Exposition*, 2003.

[26] Sue Vi Rosser and Christine Valle. InTEL: Interactive toolkit for engineering learning. `http://intel.gatech.edu/`, 2007.

[27] Dean Rubine. Specifying gestures by example. *Computer Graphics*, 25:329–337, 1991.

[28] Georges Span. LITES, an intelligent tutoring system for legal problem solving in the domain of Dutch civil law. In *Proceedings of the 4th International Conference on AI and Law*, pages 76–81, 1993.

[29] Siriwan Suebnukarn and Peter Haddawy. A collaborative intelligent tutoring system for medical problem-based learning. In *Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI '04)*, pages 14–21, 2004.

[30] Kurt VanLehn, Collin Lynch, Kay Schulze, Joel A. Shapiro, Robert Shelby, Linwood Taylor, Don Treacy, Anders Weinstein, and Mary Wintersgill. The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education*, 15(3), 2005.

[31] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 159–168, New York, NY, USA, 2007. ACM.

[32] Robert Zeleznik, K.P. Herndon, and J.F. Hughes. SKETCH: An interface for sketching 3D scenes. In *Proceedings of the 23rd annual conference on Computer Graphics and interactive techniques (SIGGRAPH '96)*, pages 163–170, 1996.