

NORTHWESTERN UNIVERSITY

Understanding and Critiquing Multi-Modal Engineering Design
Explanations

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

by

Jon Wetzel

EVANSTON, ILLINOIS

August 2014

Copyright © 2014, Jon Wetzel

All Rights Reserved

ABSTRACT

Understanding and Critiquing Multi-Modal Engineering Design Explanations

Jon Wetzel

Designers often use a series of sketches to explain how their design goes through different states or modes to achieve its intended function. Instructors find that learning how to create such explanations is a difficult problem for engineering students, thus giving the impetus to create a design coach which allows students to practice explaining their designs and give feedback on said explanations. Given the complexity and wide scope of engineering design sketches, creating this design coach is a challenging AI problem. When communicating with sketches, humans act multi-modally, using language to clarify what would often be ambiguous or crude drawings. Because these drawings can be ambiguous even to human viewers, and because engineering design encompasses a very large space of possible drawings, traditional sketch recognition techniques, such as trained classifiers, will not work. This dissertation describes a coaching system for engineering design, CogSketch Design Coach. The claims are that (1) descriptions of mechanisms, specified multi-modally using sketches and restricted natural language must be understood via qualitative reasoning, and (2) the validity and clarity of an explanation

of a mechanical design can be evaluated using a combination of design teleology and qualitative reasoning. These claims are supported by a set of evaluations of the system on sketched explanations.

The work results in the following contributions: (1) new extensions to qualitative mechanics, a qualitative model of physics reasoning, for use with sketched input, (2) two new algorithms for identifying spatial mechanical relationships, (3) two new algorithms for generating items for critique of engineering design explanations, and (4) a teleology for describing and critiquing explanations of engineering designs. Through a classroom intervention in a first year engineering design course the work resulted in (5) a corpus of 240 multi-modal explanations and (6) data on a phenomenon we describe as *sketching anxiety*. These contributions form a set of resources for building new engineering design tools like CogSketch Design Coach and provide insight into future challenges for AI and intelligent coaching systems for classroom settings.

Acknowledgements

First of all, I am grateful to Ken Forbus. He has been a wonderful advisor, growing my knowledge of AI and training me in the practice of scientific research. He continually supported me: from my first year at NU as I applied to the National Science Foundation fellowship and writing our first publication in 2008, to completing this dissertation and finding my new post-doctoral home, and everything in between. Through success and failure he encouraged me onward, and I shall be forever grateful.

I could not have gotten this far without the other members of QRG lab as well. Many thanks to Matt McLure and Subu Kandaswamy for their collaboration with me in designing and implementing the surface contact algorithm. Additional thanks to Matt for adding scribble-fill detection and for collaborating with me on the cord connection analysis. Special thanks to Maria Chang, for being my educational-software partner-in-crime; I learned much from all the ideas we bounced around. Thanks to my good friend, David Barbella, for creating the initial version of the language generation facilities used by Design Coach. Special thanks to Andrew Lovett, for his sage advice. Many, many, many thanks to Jeff and Tom, for all their time spent answering my questions about CogSketch and our reasoning engine, respectively. And thanks to everyone else in the lab, including alumni, for their support over the years.

I also am grateful to the National Science Foundation's Spatial Intelligence and Learning Center (SILC) for funding my research (NSF SLC Grant SBE-0541957) and

providing me with a network of collaborators and colleagues. My experiences with SILC expanded my knowledge of learning science and cognitive science. I also am thankful to my colleagues at SILC for their support, especially Sian Beilock, Gerardo Ramirez, and Nina Sims for their help with the psychological components of my research.

There are many others who supported me throughout my time in grad school. Thanks to my family for their continual support, love, and encouragement. Thanks to everyone at the NU Graduate Christian Fellowship for your friendship and care. Thanks to my peers in the EECS, Learning Science, and Psychology departments. Thanks to my church family at Evanston Bible Fellowship, my home away from home. And finally, praise to my heavenly Father, for making all the aforementioned people and groups so loving, supportive, and awesome at what they do.

Table of Contents

Acknowledgements.....	5
Table of Contents.....	7
Table of Figures	13
Table of Tables	20
Chapter 1 Introduction	21
1.1 Motivation.....	21
1.2 Claims and Contributions	22
1.3 Overview.....	25
Chapter 2 Background	27
2.1 CogSketch.....	27
2.1.1 Knowledge Base and Reasoning.....	27
2.1.2 Sketching.....	30
2.1.3 Spatial relationships and representations	32
2.2 Qualitative Mechanics	33
Chapter 3 Force-Centered Qualitative Mechanics.....	34
3.1 Qualitative Vector Representation.....	34
3.2 Object Representations	40
3.3 Constraint and Motion	43

3.3.1 Transmission of Constraint via Surface Contact	45
3.3.2 Transmission of Constraint via Direct Connection.....	47
3.3.3 Predicting the Next Motion.....	48
3.4 Force and Torque Representation	50
3.4.1 Transmission of Force through Surface Contact	51
3.4.2 Transmission of Force via Direct Connection	54
3.4.3 Assumed Forces and Torques	55
Chapter 4 Extensions to Qualitative Mechanics	57
4.1 Surface Contact Detection	57
4.2 Springs	68
4.3 Gears	69
4.4 Cords.....	71
4.4.1 Cords and Pulleys	74
4.4.2 Cord System Topology	75
4.4.3 Effects of Kinematic Cord Connections	81
4.4.4 Force Transfer to Movable Pulleys.....	84
4.5 Cord Connection Analysis	86
Chapter 5 CogSketch Design Coach.....	93
5.1 Overview.....	94
5.1.1 Example	94
5.1.2 Architecture.....	102

5.2 User Input.....	104
5.2.1 Sketches	104
5.2.2 Structured Language Input.....	105
5.3 State Transition Verification.....	109
5.4 Sequential Explanation Analysis	112
5.5 Teleological Ontology	121
5.5.1 Way-of-Function Ontology.....	122
5.5.2 Teleological Representations of Design Coach.....	123
5.5.3 Functions and Ways.....	124
5.6 Feedback Generation	126
Chapter 6 Evaluation.....	129
6.1 Mechanism Corpus	131
6.2 Design Coach In-Class Interventions.....	133
6.3 Explanation Corpus.....	135
6.3.1 Surface Contact.....	139
6.3.2 Sentence Critiquing.....	143
6.3.3 Detection of Other Spatial Relationships.....	146
6.3.4 Other Error Sources	148
6.4 Impact on Students.....	149
6.4.1 Sketching Anxiety.....	149
6.4.2 Course Performance.....	155

6.4.3 Effect of Feedback	158
Chapter 7 Related Work.....	160
7.1 Qualitative Mechanics	160
7.2 Teleological Ontologies for Engineering Design	162
7.3 Structured Language Input.....	164
7.4 Critiquing Explanations and Designs.....	165
7.5 Sketch Recognition and Understanding.....	167
Chapter 8 Conclusion.....	170
8.1 Discussion of Claims and Contributions.....	171
8.2 Limitations and Future Work.....	173
8.2.1 Three-Dimensional Spatial Reasoning and Representations	174
8.2.2 Richer Qualitative Reasoning	175
8.2.3 Improving Surface Contact Detection	177
8.2.4 Classifying Taut and Slack Cords in Sketched Input.....	178
8.2.5 Improving the Range of Language Understanding.....	179
Bibliography	182
Appendix A Rules for the Teleological Ontology	187
Attachment Rules.....	187
Detachment Rules	188
Adapts to Change in Size Rules.....	188
Containment Rules.....	188

Increase Comfort Rules.....	189
Appendix B Mechanism Corpus Sketches.....	190
DTC Designs.....	191
Book Holder.....	191
Detachable Paint Roller	192
Mini Baja	193
Double-Sided Switch	193
One-Handed Nail Clipper	194
Tobeggan.....	196
Abigator	198
Wheelchair Stabilizer.....	199
Recliner.....	199
Under-Armrest Rotatable Cup Holder	200
Gamma Handle	201
Dynamic Dropper.....	201
Basic Machines	202
A Runner.....	203
Gun Tackle.....	204
Luff Tackle.....	205
Luff Upon Luff	206
Other Examples.....	206

Spring Button	207
Gear Train	207
Appendix C Design Coach Assignment Handouts	208
Appendix D Sketching Anxiety Measure Questionnaire	217

Table of Figures

Figure 1: A simple overview of Design Coach.....	22
Figure 2: An overview of the architecture of Design Coach.	24
Figure 3: The reasoning architecture of CogSketch	29
Figure 4: A sketch drawn in CogSketch.	30
Figure 5: A screenshot of the metalayer in CogSketch. Relation arrows can be used between the subsketch glyphs to create a comic graph.....	31
Figure 6: An example of edge segmentation. Edges are colored lines, junctions are circles.	32
Figure 7: The Two-Dimensional Qualitative Translational Directions	36
Figure 8: Open Half Plane vs Half Plane for the direction, Left	37
Figure 9: The saloon door rotates about the center of x , which marks its rotational origin.	39
Figure 10: Examples of object types in action.....	41
Figure 11: The surfaces which touch the wedge are highlighted.....	43
Figure 12: The set of motions constrained for an object (the ball) via contact a sufficiently constrained obstacle (the wall).	46
Figure 13: Constraints (solid arrows) transfer from the ramp to the wedge, but not from the saloon door to the arm.....	47

Figure 14: Directly connecting objects (double arrow) causes constraints to be shared. The arm also shares the saloon door's rotational origin.	48
Figure 15: Surface contact detection algorithm, top level	58
Figure 16: Edges and junctions found using edge decomposition (<code>edgeDecomp</code>) on the ramp example.....	59
Figure 17: Routine for creating the surface contact decomposition	60
Figure 18: The original edge decomposition of the Ramp (left) gets new junctions at the points where junctions of wedge intersect it (right). (<code>contactDecomp</code>).....	61
Figure 19: Routine for finding the pairs of junctions in contact between two surface contact decompositions.....	61
Figure 20: An edge-edge contact is present when the junctions at the endpoints of the edges are in contact.....	62
Figure 21: The highlighted contact edge of the arm rest is an example of an inward facing contact.....	63
Figure 22: The direction surface normal of an edge contact is the edges direction (from end to end) rotated 90 degrees toward the direction of contact.....	64
Figure 23: Routines for determining if a contact is inward or not. Two different <code>inwardContact?</code> methods are needed for edge-edge and edge-junction contact.	65
Figure 24: Angle enclosure can be used to determine if an edge-to-junction contact is inward or not.....	67

Figure 25: An example of a mechanism involving a spring.	68
Figure 26: An example of a mechanism involving gears.	71
Figure 27: A sketch with a kinematic cord connection (left) and one with a movable pulley (right).	73
Figure 28: The cord glyph is divided into two segments.....	74
Figure 29: The segments of the left cord and right cord glyphs connect through the pulley... ..	77
Figure 30: Examples of normal vectors of cords inside pulleys.....	78
Figure 31: If the three sets of constraints hold for the wall (above) then the three constraints hold for the ball (below).	81
Figure 32: The three possible force transfers for a kinematic cord connection.....	83
Figure 33: A luff tackle. The bottom pulley is movable.....	85
Figure 34: The two types of edge contact relationships.	87
Figure 35: Choosing the set of contact glyphs.....	88
Figure 36: The cord is decomposed using surface contact against all three contacting glyphs at once.	89
Figure 37: A zoomed-in view of the connection between block A and the cord.....	90
Figure 38: The algorithm for characterizing edge contacts.	92
Figure 39: A basic overview of Design Coach.....	93
Figure 40: A photograph of the Under Armrest cup holder design.....	94

Figure 41: A student’s subsketch depicting the side view of the cup holder. The student chose the “fixed object” and “rigid object” labels for the selected part.	95
Figure 42: The metalayer after cloning the subsketch, editing it, and adding a <code>causes</code> relation.	96
Figure 43: The student expands the second feedback item to reveal its explanation. Selecting feedback highlights referenced parts of the sketch in green.	97
Figure 44: The student revises the sketch.....	98
Figure 45: The student constructs sentences. Green check marks indicate that no fields are left blank.	99
Figure 46: Design Coach gives feedback on the new version with sentences.....	100
Figure 47: Design Coach gives feedback on the third version.	101
Figure 48: An overview of the architecture of Design Coach. Each arrow means one subsystem is providing data to the other.....	102
Figure 49: The Structured Language Input.....	105
Figure 50: A visual description of the state transition verification algorithm	109
Figure 51: State transition verification algorithm, top level.....	110
Figure 52: Sequential Explanation Analysis, Top Level: The facts representing the sentences are sequentially passed to the appropriate verification routine.	113
Figure 53: Sequential Explanation Analysis, <code>verifySimpleFact</code> : Simple facts are proven according to the relation they represent.....	114

Figure 54: Sequential Explanation Analysis, <code>verifyCausesFact</code> : A causes B if both are proven and A justifies B.	118
Figure 55: Sequential Explanation Analysis, <code>verifyPreventsFact</code> : A prevents B if A is proven, B is contradicted by C, and C is justified by A.....	120
Figure 56: A close-up view of the Design Coach feedback pane from Figure 43	127
Figure 57: Design Coach feedback for a teleological sentence	128
Figure 58: The Dynamic Dropper Design, an Orthopedic Device	132
Figure 59: Overall feedback error rates for STV and SEA published in Wetzel & Forbus (2012) (STV = 23.5%, SEA=14.3%) vs running on the explanation corpus in 2014 (STV = 12.4%, SEA = 5.85%).....	137
Figure 60: A surface contact causes the system to conclude that the rotation is blocked.	140
Figure 61: A surface contact causes the system to conclude that the motion is blocked.	141
Figure 62: An example of a messy figure.	142
Figure 63: An example of our scribble fill detection	143
Figure 64: A cup holder drawn in perspective.....	144
Figure 65: A cup holder which clamps the cup using a spring	145
Figure 66: A feedback error in STV (highlighted in the suggestion list)	146
Figure 67: A feedback error caused by an incorrect topological relationship.	148
Figure 68: Sketching Anxiety Survey Results, Fall 2012.....	152

Figure 69: Sketching Anxiety Survey Results, Winter 2013.....	153
Figure 70: Sketching Anxiety Survey Results, Fall 2013.....	154
Figure 71: Plots of scores.....	157
Figure 72: Erroneous surface contact resulting from when tolerances are too high.....	177
Figure 73: Book Holder (The page is modeled as a RigidObject.).....	191
Figure 74: Releasing the Detachable Paint Roller	192
Figure 75: Mini Baja.....	193
Figure 76: Double-Sided Switch.....	193
Figure 77: Picture of the One-Handed Nail Clipper Prototype.....	194
Figure 78: One-Handed Nail Clipper sketch	195
Figure 79: Tobeggan (One-Handed Egg Cracker).....	197
Figure 80: Abigator (Abdominal Exerciser).....	198
Figure 81: Wheelchair Stabilizer (for playing baseball).....	199
Figure 82: Recliner	199
Figure 83: Under-Armrest Rotatable Cup Holder	200
Figure 84: Gamma Handle.....	201
Figure 85: Dynamic Dropper	201
Figure 86: A Runner	203
Figure 87: Gun Tackle	204
Figure 88: Luff Tackle	205
Figure 89: Luff Upon Luff.....	206

Figure 90: Spring Button207

Figure 91: Gear Train.....207

Table of Tables

Table 1: Lists of Available Concepts in Design Coach Sketches.....	104
Table 2: List of verbs, and their available objects in the Tell window; Italicized verbs are teleological.....	107
Table 3: Ways of Achieving Teleological Functions	125
Table 4: Mechanism Corpus Sources and Features	131
Table 5: Explanation Corpus Sources.....	135
Table 6: Error Rates for Feedback Algorithms.....	137
Table 7: Precision and Recall of Feedback Algorithms.....	138
Table 8: Breakdown of Explanation Corpus by Design	138
Table 9: Sources of Error for Explanation Corpus Sketches	139
Table 10: Number of Sketches with Surface Contact Errors by Type (Note: some sketches contained more than one type of error)	140
Table 11: Number of students who responded to survey in Design Coach/Control group by quarter	151
Table 12: Correlation between students' scores on the Design Coach assignments and grades in the Design.....	156
Table 13: Correlation between scores on the Design Coach assignment and number of times feedback was requested.....	158

Chapter 1

Introduction

1.1 Motivation

The motivation for this work comes from the Design Thinking and Communication course¹ (DTC) at Northwestern University. DTC is the introductory course for freshman in engineering majors. DTC students work in teams to solve a design problem for a real world client over the course of each quarter. The course is two quarters long, and in the first quarter students are taught to use sketching as part of the design process. However, instructors in DTC report that students have trouble learning to communicate with sketches. Sketching is both an important form of representation in mechanical design, and the preferred method of external representation for engineers (Ullman et al., 1990). Given the complexity and wide scope of engineering design sketches, creating a software coach that could let students practice explaining their designs and give feedback on said explanations is a challenging AI problem.

¹ <http://segal.northwestern.edu/programs/undergraduate/design-thinking-communication/index.html>

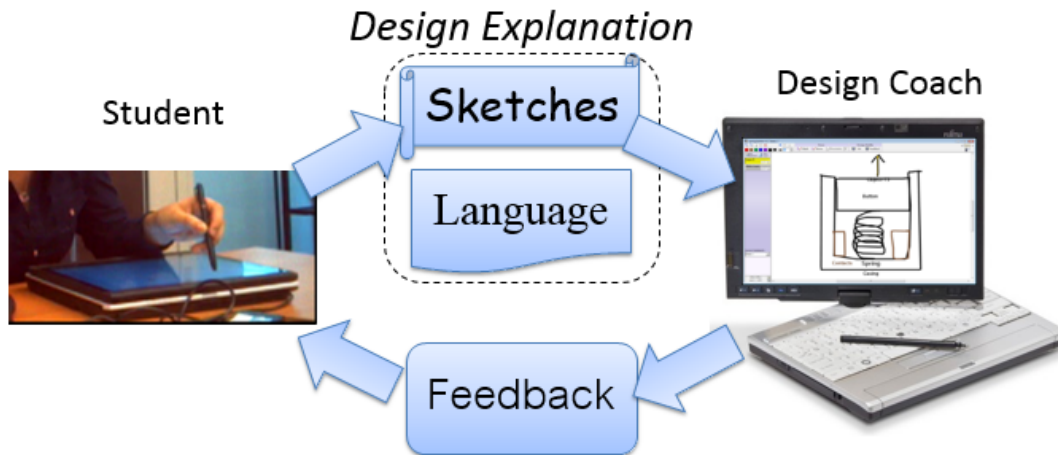


Figure 1: A simple overview of Design Coach

When communicating with sketches, humans act multi-modally (see Figure 1), using language to clarify what would often be ambiguous or crude drawings. Because these drawings can be ambiguous even to human viewers, and because engineering design is a very large space of possible drawings, traditional sketch recognition techniques such as trained classifiers (de Silva et al., 2007; Plamondon & Srihari, 2000; Wobbrock et al., 2007) will not work. The goal of this thesis is to show that using qualitative reasoning and engineering design teleology, an AI system can understand and critique multimodal explanations of early stage engineering designs involving mechanisms.

1.2 Claims and Contributions

In this thesis, we claim that:

1. Descriptions of mechanisms, specified multi-modally, using sketches and restricted natural language, must be understood via qualitative reasoning.
2. The validity and clarity of an explanation of a mechanical design can be evaluated using a combination of design teleology and qualitative reasoning.

Next we clarify the terms of in these claims. In the first claim, the *descriptions of mechanisms* we are interested in are explained at the conceptual stage of design, lacking precise numerical dimensions and quantities. *Understanding* descriptions of mechanisms entails the system predicting the behavior of the mechanism given the description and maintaining knowledge of the causation of the behavior suitable for explaining the behavior to the user. A numerical physics simulator or engine would not satisfy this claim, both because the description may be missing necessary numerical information and because such simulators do not maintain a knowledge representation which explains the outcome. The *restricted natural language* is human-readable structured language input of the kind introduced by Thompson et al. (1983). The user may use the structured language to add qualitative statements to the descriptions, such as statements about the mechanism's behavior (e.g. The ball can move; The ball moves to the left; etc.) or about its properties (e.g. An object is compressed; An object touches another object; etc.). Understanding and critiquing descriptions with these statements further necessitates the use of qualitative reasoning.

In the second claim, *design teleology* entails understanding the function of a design. Qualitative reasoning allows us to see if the behavior of the mechanism helps it

accomplish its function. We say *can* here instead of *must* (as in the first claim) because the design teleology extends the range of statements and explanations we can critique, rather than enabling critiques entirely.

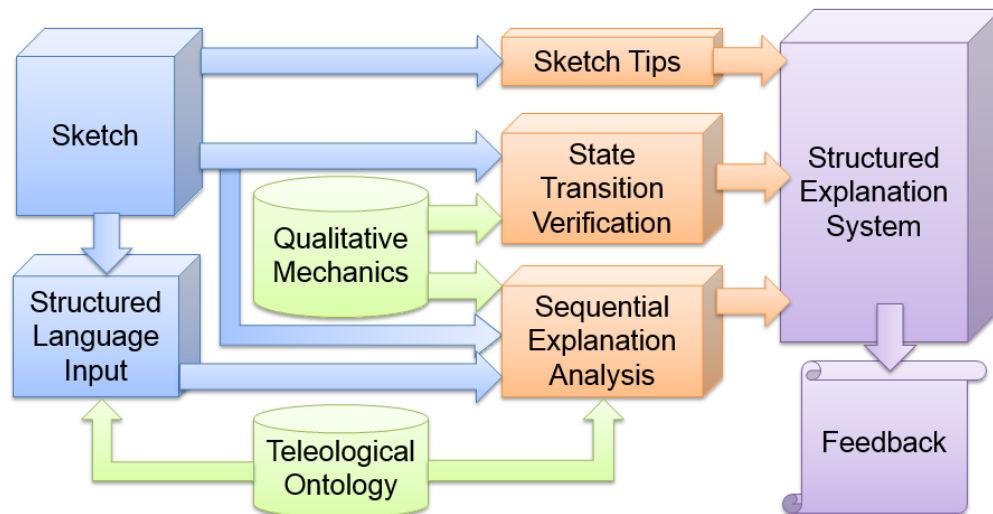


Figure 2: An overview of the architecture of Design Coach.

To support these claims, we present *CogSketch Design Coach*, a system which uses qualitative reasoning to critique explanations of mechanisms consisting of freehand sketch and structured language input. We will evaluate the performance of Design Coach on two corpora of sketched mechanisms, one demonstrating the range of designs it can understand, and another collected from a series of classroom interventions where students from DTC used Design Coach in a homework assignment. Thus the contributions of this thesis are:

1. New extensions to qualitative mechanics for use with sketched input, including a force-centered model of rigid body mechanics and representations for springs, gears, cords, and pulleys

2. Two new algorithms for extracting mechanical spatial relationships from sketched input: surface contact detection and cord connection analysis
3. Two new algorithms for generating items for critique of engineering design explanations: State Transition Verification and Sequential Explanation Analysis
4. A teleological ontology for representing higher level functions of designs involving mechanisms in the context of a first-year undergraduate engineering design course

Additionally, our classroom intervention resulted in the following contributions:

5. A corpus of 240 multi-modal explanations from engineering design students.
6. Data from the results of a sketching anxiety survey, including the detection of a significant decrease in anxiety in two out of three quarters in sections where the Design Coach assignment was introduced and one out of three quarters in control sections.

1.3 Overview

This thesis begins with background information on the AI systems and theory used by Design Coach: CogSketch, our sketch understanding system, and qualitative mechanics, a qualitative model of physics (Chapter 2). Chapter 3 introduces Design Coach QM, a new, force-centered version of QM for use with sketched input and multi-modal explanations. Chapter 4 introduces further extensions to that QM, including algorithms for deriving important relationships from freehand sketched input and

representations for springs, gears, and cords. Chapter 5 describes the implementation of Design Coach, including its multimodal input, teleological ontology, and algorithms for critiquing explanations and giving feedback. Design Coach was then evaluated over two corpuses: one was a variety of mechanisms, and the other a set of explanations produced by students during the classroom intervention. This evaluation is described in Chapter 6, along with a description of the intervention and its impacts on students. Chapter 7 discusses related work, and Chapter 8 concludes with a summary of how the claims were met and a discussion of the limitations and future work.

Chapter 2

Background

This chapter describes prior work that the Design Coach builds upon. The first section describes CogSketch, the sketch understanding system that takes in the user's drawing and computes spatial relationships useful for reasoning. The second section introduces Qualitative Mechanics, the model of physics used by the system to predict motions in mechanisms.

2.1 CogSketch

CogSketch is a publicly available, open-domain, sketch understanding system (Forbus et al., 2011). This section describes the core features of CogSketch that existed before Design Coach and are relevant to its workings. The user interface additions for Design Coach are described in Chapter 5.

2.1.1 Knowledge Base and Reasoning

CogSketch uses a customized version of the OpenCyc knowledge base which includes extensions for qualitative reasoning and analogy added by the Qualitative Reasoning Group. Facts and axioms in the knowledge base are represented as predicate calculus statements, e.g.

```
(isa Gear1 Gear)
```

states that the entity named `Gear1` is an instance of the concept `Gear`. These statements are used to define *collections*, *relations*, and *rules*. Collections are classifications of entities (e.g. `Gear`, `Spring-Device`, `Cord`, `RigidObject`) and they are arranged in an inheritance hierarchy (e.g. all members of the collection `Gear` are members of the collection `RigidObject`). Relations convey relationships between or properties of entities. For example, the `touchesDirectly` relationship is used when two entities are in physical contact. Rules allow the inference of a consequent fact from a list of antecedent facts and are represented using Horn clauses. For example, when reasoning about a sketch we might assume that if two objects are gears and touch directly, then they are enmeshed. This would be written as:

```
(<== (enmeshed ?obj1 ?obj2)
      (touchesDirectly ?obj1 ?obj2)
      (isa ?obj1 Gear)
      (isa ?obj2 Gear))
```

The collections, relations, and rules in the knowledge base are partitioned into logically consistent sets called *microtheories*. These microtheories are also arranged in an inheritance hierarchy such that all the facts believed in a microtheory are believed by all of its children.

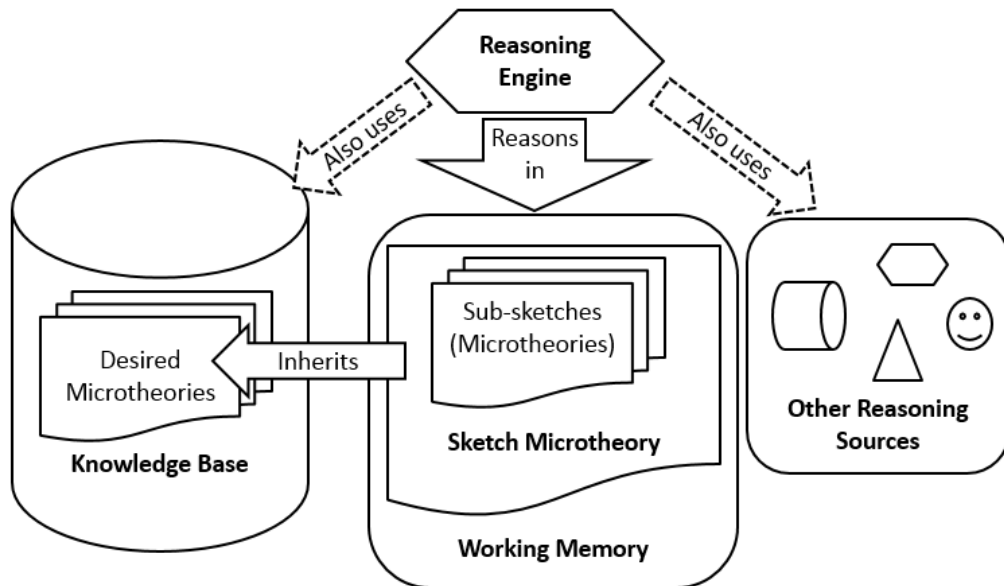


Figure 3: The reasoning architecture of CogSketch

CogSketch includes a reasoning engine which can perform queries. When working with a sketch, the reasoning engine primarily performs queries in a context called *working memory* which includes the microtheory of the active sketch (see Figure 3). The sketch's microtheory inherits a set of desired microtheories, usually those relevant to spatial reasoning and any topics at hand, such as the qualitative mechanics knowledge. In addition to looking up facts in the selected microtheory, the engine can use backchaining to use rules to infer answers to the query, or use alternative reasoning sources. One example of an alternative reasoning source would be the use of geometric calculations to derive the topological relationship between to objects, or the use of a Voronoi diagram to derive adjacency. Design Coach uses a combination of knowledge and reasoning sources to understand the user's sketch and generate feedback.

2.1.2 Sketching

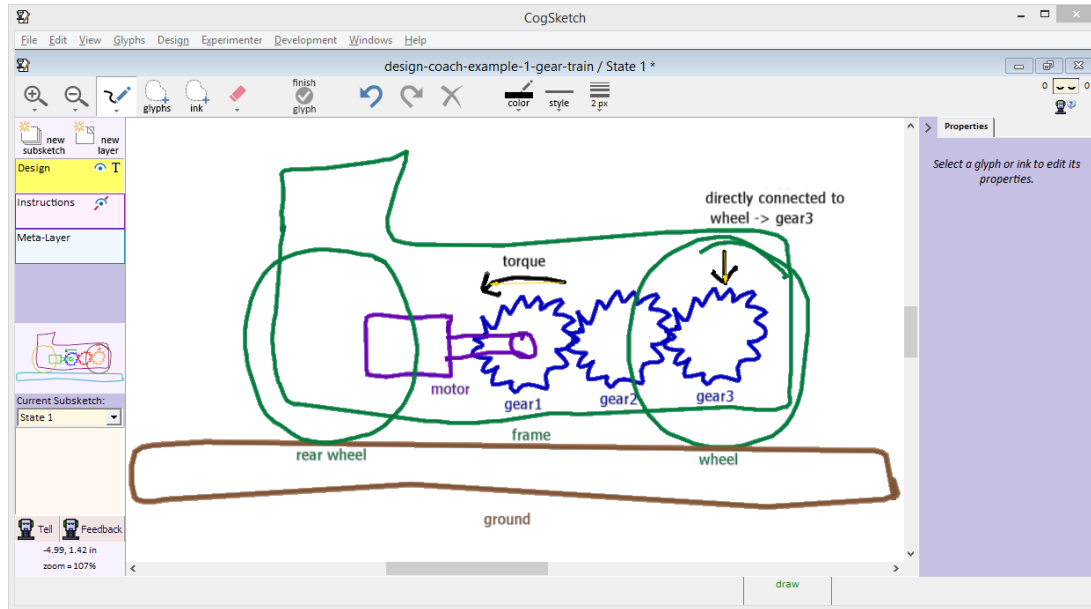


Figure 4: A sketch drawn in CogSketch.

In CogSketch the user draws sketches using digital ink. This ink is segmented by the user into *glyphs*. Glyphs consist of their ink and one or more conceptual labels drawn from its knowledge base. As the user (or the system) adds and removes labels, an underlying representation of the sketch is updated by adding and removing corresponding facts to the working memory of the sketch. There are three types of glyphs that the user can choose from:

1. *Entity glyphs* are the standard type of glyph, and represent instances of one or more concepts. Some examples of conceptual labels relevant to this work include the collections `RigidObject` and `Gear`.

2. *Relation glyphs* represent a relationship between two glyphs. They are assumed to be drawn as an arrow. An example is the rightmost arrow in Figure 4, which represents the direct connection between gear3 and the wheel.
3. *Annotation glyphs* specify additional information about glyphs such as the value of quantity like height or weight. An example is the leftmost arrow in Figure 4, which represents the rotational force applied to gear1. For annotations involving a quantity, the user may supply a precise value (e.g. 10 Newtons) if they wish. Design Coach does not use these values at present, relying only on the direction of the arrow instead. A single annotation may be applied to multiple glyphs; doing so applies the same information to all of them.

Design Coach sketches use all three types of glyphs. The user may also give each glyph a name—a string which will be used to refer to it in explanations.

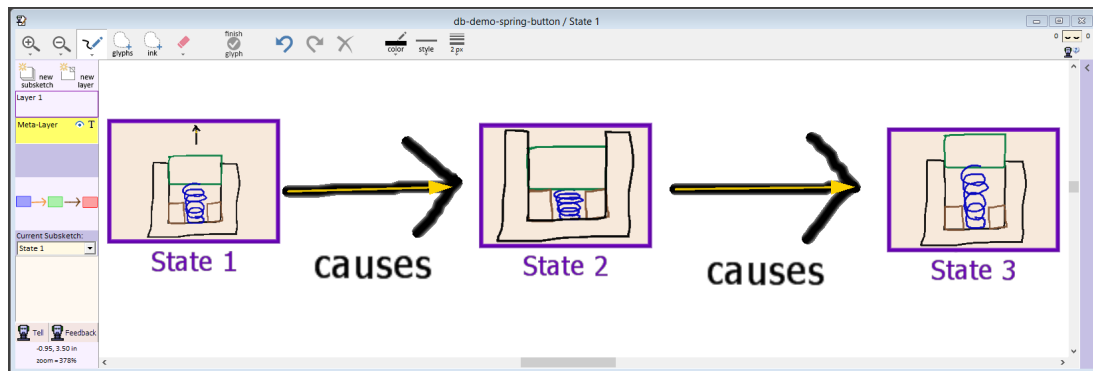


Figure 5: A screenshot of the metalayer in CogSketch. Relation arrows can be used between the subsketch glyphs to create a comic graph.

Each sketch can contain one or more separate drawing areas, called *subsketches*. By selecting the metalayer view (see Figure 5), the user can see all of their subsketches

and even add glyphs to provide information about how subsketches relate. The metalayer is important for describing mechanisms operating in multiple states in Design Coach (see Chapter 5).

2.1.3 Spatial relationships and representations

As glyphs are added and edited, spatial relationships are computed and asserted into the working memory of the sketch. CogSketch computes some spatial relationships automatically as glyphs are changed, including topological relationships based on the RCC8 relational vocabulary (Cohn, 1996). Other relationships, such as positional relationships (e.g. *above*, *rightOf*), may also be computed as needed. Design Coach uses both topological and positional relationships to detect surface contacts and perform other spatial reasoning tasks.

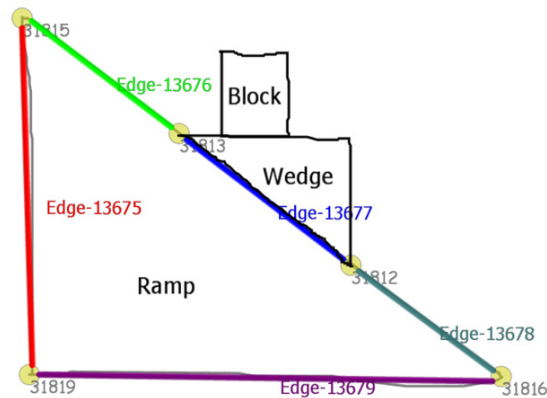


Figure 6: An example of edge segmentation. Edges are colored lines, junctions are circles.

CogSketch also includes routines for segmenting a glyph into edges and junctions (Lovett et al., 2006). Chapter 4 describes how Design Coach uses this segmentation to find the surface contacts between objects.

2.2 Qualitative Mechanics

Qualitative Mechanics (QM) is a qualitative model of physics used to represent and predict the behavior of physical mechanisms (Kim, 1993; Nielsen, 1988; Pearce, 2001). Given a description of objects, their surface contacts, and the forces active in a mechanism, Nielsen's QM allows a system to predict where objects in a mechanism will move qualitatively in the next instant of time and to envision the future states of interaction between said objects. The QM of Design Coach builds upon the former capability. Nielsen used scanned images of actual machined parts for input, whereas Design Coach takes in hand drawn sketches. This, combined with the additional goal of critiquing the users' explanation, leads to some differences in the approach to Design Coach QM, as explained in Chapter 3 and Chapter 7.

Chapter 3

Force-Centered Qualitative Mechanics

Qualitative Mechanics (QM) is a qualitative model of physics used to represent and predict the behavior of physical mechanisms (Kim, 1993; Nielsen, 1988; Pearce, 2001; Stahovich et al., 1997). The Design Coach QM builds upon and adapts the work of Nielsen (1988) for use with engineering design sketches. *Force-Centered Qualitative Mechanics* is the core of Design Coach QM, and its name reflects a major difference from Nielsen's QM. The end goal has changed from predicting motion to explaining (and critiquing explanations of) mechanical behavior, therefore the underlying representations require more detail at the level of forces.

This chapter describes the representations at the core of Design Coach QM. We begin with qualitative vectors, then define objects, constraint, motion, force and torque.

3.1 Qualitative Vector Representation

We represent direction and orientation using qualitative vectors. In Nielsen's QM a one-dimensional vector has three possible values: + (greater than zero), - (less than zero), and 0 (zero). Design Coach QM adds a fourth value, *Ambig* (ambiguous), to the

original three to represent the case where it is not clear which direction the vector points.

Definition 1 (Senses of One-Dimensional Vectors): *A Sense is a symbol denoting a qualitative value. The set of senses for a one-dimensional vector is $\{+, 0, -, \text{Ambig}\}$.*

The senses + and - have an inverse relationship, defined as follows.

Definition 2 (Inverse Senses): *$\text{inverseSense}(s_1, s_2)$ is true if $s_1=+$ and $s_2=-$ or $s_1=-$ and $s_2=+$.*

The inverse and ambiguous senses allow us to define the sum of senses.

Definition 3 (Sums of Senses): *$\text{senseSum}(s_1, s_2, s_R)$ is true if the sum of s_1 and s_2 is s_R . The following rules define the sums of senses:*

1. *The sum of any sense s and 0 is s .*
2. *The sum of any sense s and Ambig is Ambig.*
3. *The sum of any sense s and itself (e.g. + and +, Ambig and Ambig, etc.) is s .*
4. *The sum of a sense s and its inverse (Definition 2) is Ambig.*

With one-dimensional directions defined, we now move to the two-dimensional case.

For two-dimensional space there are nine translational directions (zero, up, down, left, right, and quadrants 1 through 4 for diagonals) and three rotational directions (zero, clockwise, and counterclockwise). We begin with the translational vectors. To the first nine, we again add an ambiguous vector:

Definition 4 (Two-Dimensional Translational Vectors): *A $2DQVector$ denotes a translational direction. The set of two-dimensional qualitative vectors is $\{Up,$*

Down, Left, Right, Quad1, Quad2, Quad3, Quad4,
ZeroQVector, AmbigQVector}.

The translational directions are two-dimensional qualitative vectors with components in both the x and y axis of a Cartesian coordinate plane. See Definition 5 and Figure 7.

Definition 5 (Senses of Two-Dimensional Translational Vectors): $xSense(v, s)$ is true when Sense s is the direction of the component of v along the x axis. Similarly, $ySense(v, s)$ is true when s is the direction of the component of v along the y axis.

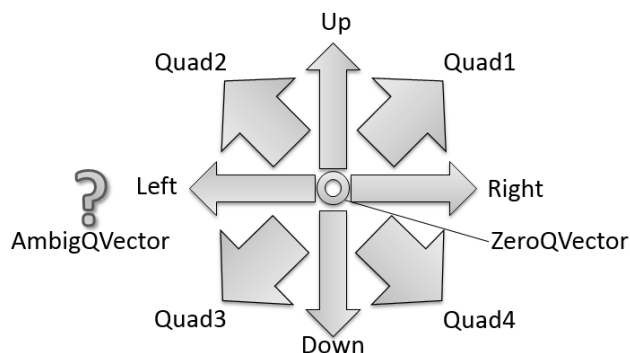


Figure 7: The Two-Dimensional Qualitative Translational Directions

We add qualitative vectors by summing their component senses. For example, Right $(+,0)$ and Up $(0,+)$ sum to Quad1 $(+,+)$. When we add opposite directions $(+ \text{ and } -)$, the resulting component is Ambig. AmbigQVector represents any two-dimensional vector with an ambiguous component. ²

² Alternatively, we could define a set of four ambiguous vectors (e.g. leftwards is $(+,Ambig)$, upwards is $(Ambig,+)$, etc.). This would allow for more specific feedback on the direction of ambiguous vectors and allow the user to the more limited ambiguous vectors in their explanations. However, there was never a need for this feature so it remains unimplemented in Design Coach.

The two dimensional vectors also have inverse relationships with each other, derived from their senses:

Definition 6 (Inverse Vectors): $\text{inverseVector}(v1, v2)$ is true when all of the senses of $v1$ are the inverse of their respective counterparts in $v2$.

In QM it is often important to know which vectors have overlapping direction, (e.g. to know if two forces push together). As in Nielsen's QM, the concepts of *half plane* and *open half plane* are used to solve these problems. Thus, Definition 7 and Definition 8 correspond to Definitions 12 and 13 in Nielsen's QM. Figure 8 also displays the difference between the open half plane and the half plane graphically.

Definition 7 (Half Plane): $\text{halfPlane}(v1, v2)$ is true when the sign vector dot product of the first argument and the second argument is + or 0.

Definition 8 (Open Half Plane): $\text{openHalfPlane}(v1, v2)$ is true when the sign vector dot product of the first argument and the second argument is +.

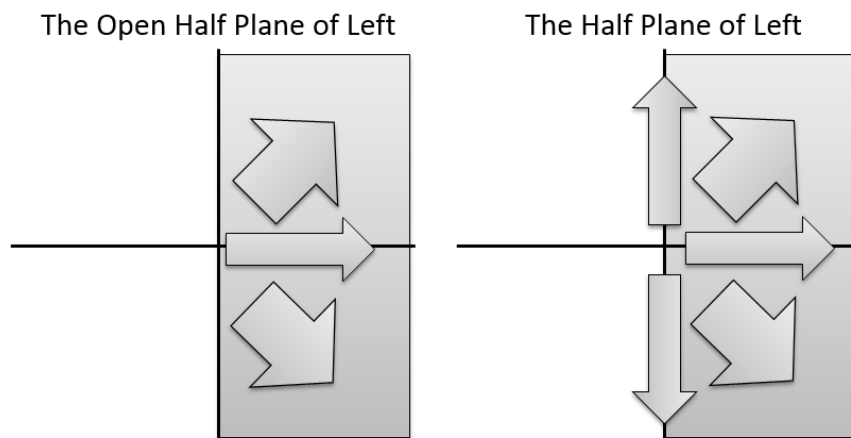


Figure 8: Open Half Plane vs Half Plane for the direction, Left

Having introduced translational vectors, we now turn to rotational vectors. We define these similarly to the way they are defined in Definitions 4-6 of Nielsen's QM, with the addition of an ambiguous vector.

Definition 9 (Rotational Vectors): *A `RotVector` denotes a rotational direction.*

The set of rotational vectors is $\{\text{CW}, \text{CCW}, \text{ZeroRot}, \text{AmbigRot}\}$.

Definition 10 (Senses of Rotational Vectors): *`rotSense(v, s)` is true when Sense s corresponds with the direction of the rotational vector v .*

Rotating a translational vector produces a new translational vector. The following relationships determine which rotational direction must be used to perform specific rotations (or will be produced as a result of rotation in that direction):

Definition 11 (Rotating Translational Vectors): *`rotate90(tv1, tv2, rv)` is true when $tv1$ when rotated 90 degrees in direction rv produces $tv2$. Similarly, `rotate45(tv1, tv2, rv)` is analogously true for 45 degree rotations.*

We use `Rotate90` in rules to infer rotational consequences of translational facts, such as determining the direction of a torque produced by a translational force.

We use `Rotate45` to quickly recall a translation vector's neighboring members in the open half plane without getting the vector itself as a result; we also use it to find the direction between a cord and a pulley it runs through.

At any instant of time, an object may only rotate around one point. We call this point the object's *rotational origin*.

Definition 12 (Rotational Origin Relationship): $\text{rotationalOrigin}(o, p)$ is true when p is the point that object o rotates around in the given reference pane.

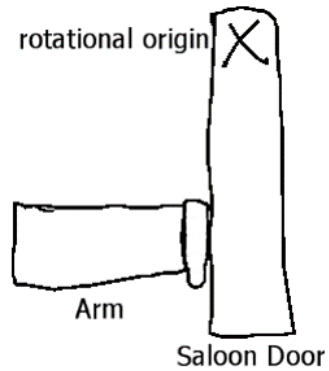


Figure 9: The saloon door rotates about the center of x , which marks its rotational origin.

To find the rotational origin, we first look to see if one is given (e.g. sketched by the user). If it is not, we use the object's center of mass. Cords and gears creates special cases that are addressed in Chapter 4.

The last definition we need concerning vectors is the use of a vector in a spatial relationship: the direction between two points:

Definition 13 (Qualitative Vector Between):

$\text{qualitativeVectorBetween}(p1, p2, tv)$ is true when tv is the direction from point $p1$ to point $p2$.

With vectors defined, we now turn to the problem of representing the objects involved in the engineering design explanations.

3.2 Object Representations

Nielsen's QM worked on systems of rigid objects. Design Coach QM extends it with additional types (presented in Chapter 4) in order to accommodate a wider variety of mechanisms, including those involving non-rigid objects such as springs and cords. The most general category of object is the set of all physical objects, which includes all rigid objects:

Definition 14 (Physical Objects): *A `PhysicalObject` is a member of the set of physical objects.*

Definition 15 (Rigid Objects): *A `RigidObject` is a member of the set of rigid objects. Every `RigidObject` is a `PhysicalObject`.*

In Design Coach QM, forces may be applied to any physical object, which may cause them to move in response. Rigid objects differ in that they may have their motion blocked or redirected by making contact with other rigid objects. Since all motion is relative to some frame of reference, two additional object types are useful:

Definition 16 (Fixed Objects): *A `FixedObject` is a member of the set of physical objects which neither rotate nor translate in the current frame of reference.*

Definition 17 (Fixed-Axis Objects): *A `FixedAxisObject` is a member of the set of physical objects which do not translate but may rotate around its rotational origin (Definition 12) in the current frame of reference.*

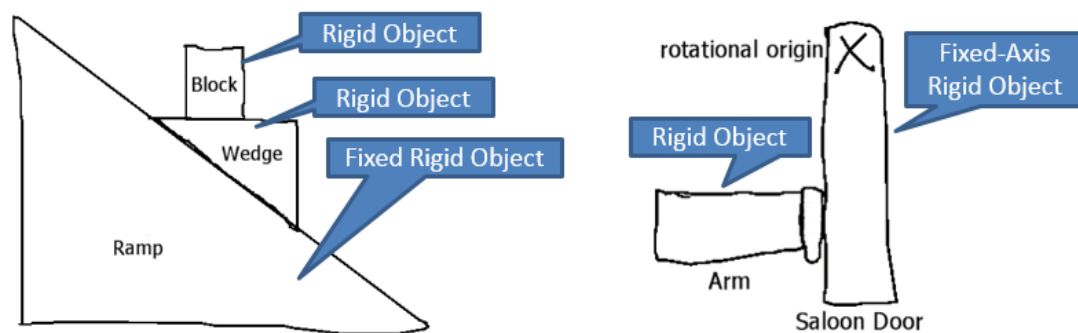


Figure 10: Examples of object types in action

Figure 10 presents two examples of mechanical situations we discuss in this chapter. One is a block sitting on a wedge on a ramp, viewed from the side. (A similar example appears in Nielsen (1988). The other is a human arm pushing open the door of a saloon. All of the physical objects in these diagrams are rigid. The “x” labeled “rotational origin” is a `RotationalOrigin` and is not a physical object. In addition, the ramp is a fixed object, and the saloon door is a fixed-axis object.

Surfaces of objects are represented similarly to Definitions 15-19 of Nielsen’s QM:

Definition 18 (Surfaces): *A Surface is a member of the set of the surfaces of a rigid object.*

Definition 19 (Normal of a Surface): `surfaceNormal(s, tv)` is true when the surface normal vector of surface, s , is pointed in the direction of translational vector, tv .

Definition 20 (Surface Contact): `surfaceContact(s1, s2)` is true when surfaces $s1$ and $s2$ are in physical contact with each other.

Definition 21 (Has a Contact Surface): $\text{hasContactSurface}(o1, o2, s)$ is true when s is a surface of $o1$ and s has surface contact with a surface of $o2$.

The hasContactSurface relationship (Definition 21) replaces Nielsen's surface relationship (Nielsen's Definition 15), and takes its role in linking the surface to its parent object. In Nielsen's QM, all of the surfaces of each object were reified, but in this work only the contact surfaces are reified. The non-contact surfaces of the objects do not need to be reified for an analysis of a single instant of time, as is the case in Design Coach. In contrast, Nielsen's CLOCK program used envisionment to find all possible future states. Such an envisionment requires the system to have knowledge of all the non-contact surfaces, to see if they may come in contact in the future. Choosing to reify only the contact surfaces can increase efficiency by reducing the search space of surfaces. In our ramp example, four surfaces must be reified: the bottom surface of the block, a top right surface of the ramp, a top surface of wedge, and the bottom left surface of the wedge. The former two surfaces are shown in Figure 6. We discuss the method for detecting surface contacts in sketched input in Chapter 4.

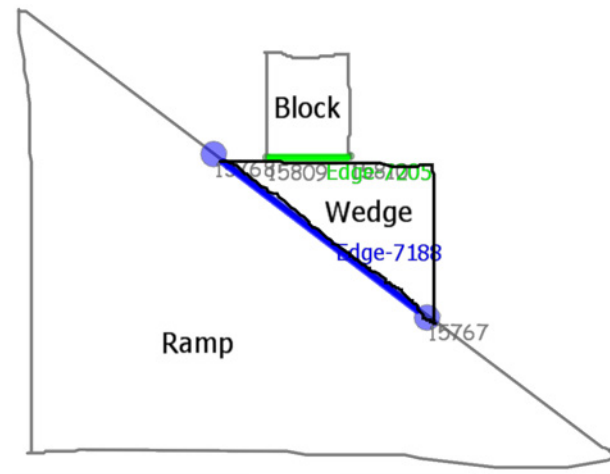


Figure 11: The surfaces which touch the wedge are highlighted.

3.3 Constraint and Motion

The most common goal in analyzing the behavior of a mechanism is to determine where its parts will move (or not move). In this section we describe the process of determining the next motion of an object. To start, at any particular instant of time, each object possesses one translational motion vector and one rotational motion vector:

Definition 22 (Motion): $\text{translationalMotion}(o, tv)$ is true when object o moves in the direction of vector tv in the current frame of reference. tv may be any of the vectors in **Definition 4**, including ZeroQVector .

$\text{rotationalMotion}(o, rv)$ is true when object o rotates in the direction of vector rv in the current frame of reference. rv may be any of the vectors in **Definition 9**, including ZeroRot .

In QM *constraint* is the property of being unable to move in one or more directions. Like motion, it has translational and rotational components:

Definition 23 (Constraint): $\text{translationalConstraint}(o, tv)$ is true when object o cannot move in the direction of vector tv in the current frame of reference.

(tv cannot be `ZeroQVector` or `AmbigQVector`.) Similarly,

$\text{rotationalConstraint}(o, rv)$ is true when object o cannot rotate in the direction of vector rv in the current frame of reference. (rv cannot be `ZeroRot` or `AmbigRot`.)

While an object can only have one translational motion and one rotational motion at a time, it may have multiple constraints at once.

There are five cases in which an object O may become constrained:

1. O is a `FixedObject` or a `FixedAxisObject` (see Definition 16 and Definition 17).
2. O is a `RigidObject` in contact with blocking obstacle, B , such that B prevents motion of O in one or more directions.
3. O is directly connected to (e.g. glued to) another object, P , such that it shares the constraints on P .
4. O is tied to another object by a taut cord.
5. O is a gear, enmeshed with another gear G such that it shares in the some of the constraints on G .

The first case is self-explanatory; cases 2 and 3 will be discussed next; and cases 4 and 5 will be addressed in the sections on cords and gears in Chapter 4.

Finally, it is also sometimes useful for explanatory or teleological purposes to be able to state that an object is able to be moved or rotated at all.

Definition 24 (Movable and Rotatable): *movable*(\circ) is true when object \circ is not constrained from translating in at least one direction in the current frame of reference. Similarly, *rotatable*(\circ) is true when object \circ is not constrained from rotating in at least one direction in the current frame of reference.

3.3.1 Transmission of Constraint via Surface Contact

Constraint transfer occurs through surface contact, as it does in Nielsen's QM. Nielsen's law of contact constraint (Nielsen 1988, pg 23) says that an obstacle prevents a contacting object from moving if the obstacle is *sufficiently constrained* as follows:

1. Translational motion into the open half plane centered on the contacting object surface normal at the point of contact.
2. Rotational motion clockwise about any point which lies in the open half plane centered ninety degrees clockwise from the contacting object's surface normal at the point of contact.
3. Rotational motion counter-clockwise about any point which lies in the open half plane centered ninety degrees counter-clockwise from the object's surface normal at the point of contact.

If an obstacle is sufficiently constrained as above, then the above three constraints are transmitted to the object blocked by said obstacle. Figure 12 shows this graphically.

The ball cannot translate $Left$, $Quad1$, or $Quad4$. Also note that while the ball may not rotate around points in the halfplanes above and below it, it may rotate around its center.

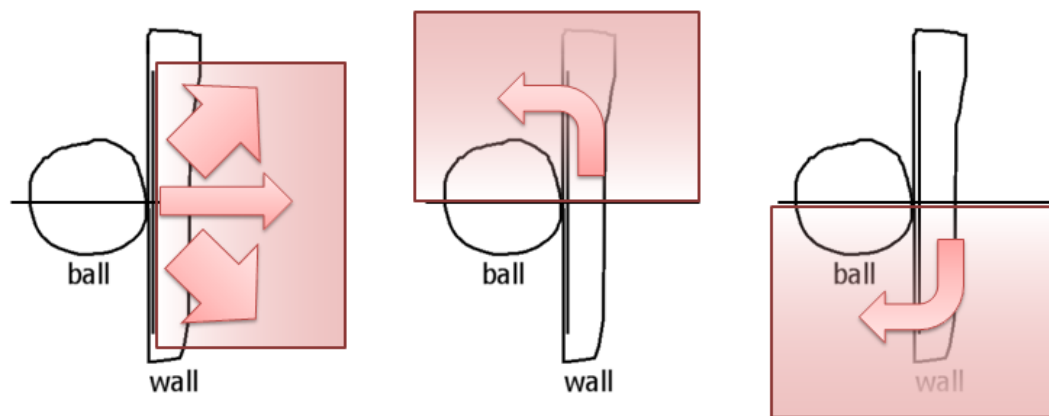


Figure 12: The set of motions constrained for an object (the ball) via contact a sufficiently constrained obstacle (the wall).

In our block-and-wedge-on-ramp example (Figure 13), the ramp is constrained in all directions. At the surface contact between the ramp and wedge, the ramp is sufficiently constrained, so the wedge inherits those constraints. At the contact between the block and wedge however, the wedge is not sufficiently constrained in the downward direction (it may slide down and to the right), so there are effectively no constraints on the block. Similarly, while the saloon door cannot translate anywhere to the right, it can rotate counter-clockwise out of the arm's way. Therefore it is not sufficiently constrained, and the arm can move freely in any direction.

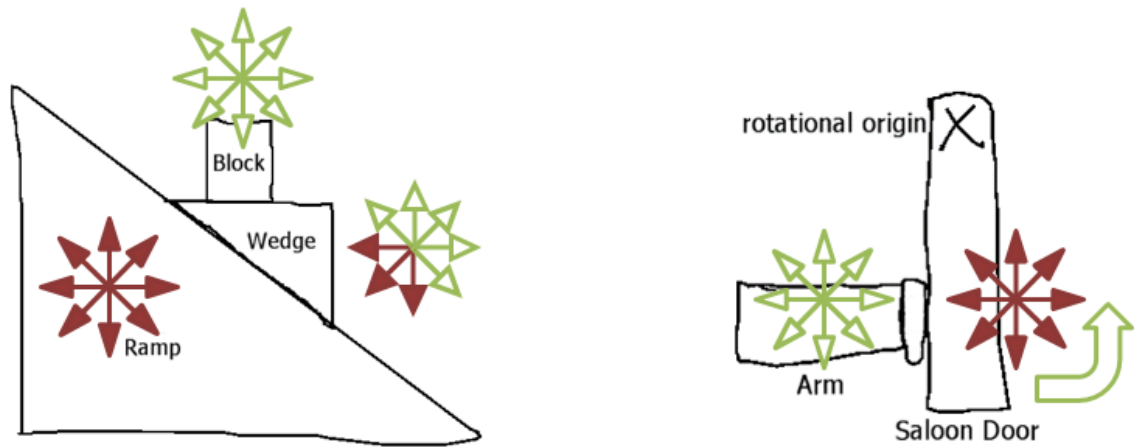


Figure 13: Constraints (solid arrows) transfer from the ramp to the wedge, but not from the saloon door to the arm.

3.3.2 Transmission of Constraint via Direct Connection

Another new addition of Design Coach QM is the concept of directly connected objects:

Definition 25 (Direct Connection): $\text{directlyConnected}(o_1, o_2)$ is true when o_1 is rigidly connected to o_2 (for example, glued together) such that o_1 and o_2 share their constraints and rotational origin.

Figure 14 illustrates how the constraints in our examples change if we directly connect some of the parts in Figure 13 together. In the case of the block and wedge on the ramp, if we directly connect the wedge and ramp then the wedge inherits the ramp's constraints and can no longer move. Consequentially, the wedge becomes sufficiently constrained at the surface contact with the block and transmits its constraints to the block accordingly. If we directly connect the arm to the fixed-axis saloon door (ouch!),

the arm is constrained from translating, but now will instead rotate around the door's rotational origin.

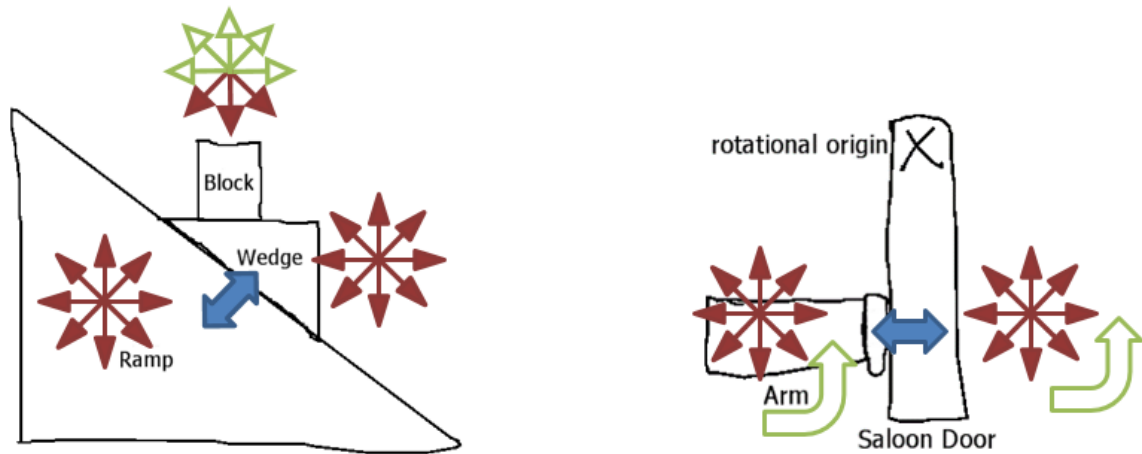


Figure 14: Directly connecting objects (double arrow) causes constraints to be shared. The arm also shares the saloon door's rotational origin.

The concept of direct connection can be contrasted with the concept of linkage in Nielsen's QM (Nielsen's Definition 21). A linkage asserts that two different representations of one object are indeed the same object. Direct connection is for two different entities that effectively become one physical entity, from the perspective of mechanical constraint.

3.3.3 Predicting the Next Motion

Now that constraint has been defined, we return to the problem of predicting the next motion of an object. In Design Coach QM, the next motion of an object can be determined if the net force applied to it is known, as follows:

Definition 26 (Predicting Translation): Given an object o , and the direction of the net force applied to it, d :

1. If d is `AmbigQVector`:

The result is `TranslationalMotion(o, d)`

2. Otherwise, if not `TranslationalConstraint(o, d)`:

The result is `TranslationalMotion(o, d)`

3. Otherwise, if not `TranslationalConstraint(o, d')` s.t.

`openHalfPlane(d, d')`:

The result is `TranslationalMotion(o, d')`

4. Otherwise the result is:

`TranslationalMotion(o, ZeroQVector)`

At first glance the clause in line 3 might seem like it could produce either of two results because d' could be either neighboring vector in the open half plane of d . However, whenever a translational constraint is introduced, (either through fixed property or surface contact) it is applied to at least an open half plane, guaranteeing that there will be at most one free direction within an open half plane of any constraint. This procedure is therefore correct.

The procedure for determining rotation is simpler:

Definition 27 (Predicting Rotation): Given an object o , and the direction of the net torque applied to it, d :

1. If d is `AmbigRot`:

The result is `RotationalMotion(o, d)`

2. *Otherwise, if not* `RotationalConstraint(o, d)` :

The result is `RotationalMotion(o, d)`

3. *Otherwise the result is:* `RotationalMotion(o, ZeroRot)`

In Design Coach QM, all motions are the result of a net force and torque on the object. This is a major departure from Nielsen's QM, in which motion is found through transmission between objects. In Design Coach QM, forces/torques are transmitted between objects instead. The next section describes the force/torque transfer process and how the direction of the applied net force and torque are found.

3.4 Force and Torque Representation

In Design Coach QM, the core of the force representation is the following relationship:

Definition 28 (Force Applied to Object):

`forceAppliedToObject(o, tv, src)` *is true when* `PhysicalObject o` *receives a force in direction* `tv` *from source* `src`.

Torques³ are represented similarly:

Definition 29 (Torque Applied to Object):

`torqueAppliedToObject(o, rv, src)` *is true when* `object o` *receives a torque in direction* `rv` *from source* `src`.

³ In this document we use the more colloquial definition of torque, a turning or twisting force. However, "rotational force" is a more accurate term for a mechanical engineering context, so the Design Coach user interface uses that term instead.

The source is used to distinguish between multiple forces acting in the same direction. The force/torque applied to an object is the second important connection point for QM extensions (the first being constraints). Springs, cords, and gears can apply forces and torques to objects using this relationship. Those applied forces/torques then feed into the net force/torque which make each object move. The net force and torque applied are defined as follows:

Definition 30 (Net Force): $\text{netForceApplied}(o, tv)$ is true when the direction of the net force applied to o is the translational vector tv .

Definition 31 (Net Torque): $\text{netTorqueApplied}(o, rv)$ is true when the direction of the net force applied to o is the rotational vector rv .

These are computed by summing up the qualitative vectors using their component senses (**Definition 3**). As noted previously, there are several ways a force/torque may be applied to an object. Some forces may just be given, such as gravity. Next, we describe transmission through surface contact and direct connections.

3.4.1 Transmission of Force through Surface Contact

When a force is applied to an object, it produces a force at all of its contact surfaces which have their normals in the open half plane of that force:

Definition 32 (Force at a Surface): $\text{forceAtSurface}(s, tv, o)$ is true when:

$\text{hasContactSurface}(o, o', s)$ AND

$\text{forceAppliedToObj}(o, tv', src)$ AND

surfaceNormal(s, tv) AND
 openHalfPlane(s, tv')

The force is transmitted between surfaces like so:

Definition 33 (Force Applied to a Surface):

forceAppliedToSurface(s, tv, src) is true when:

hasContactSurface(o, src, s) AND ;; s belongs to o
 surfaceContact(s, s') AND
 forceAt(s', tv, src) ;; s' belongs to object src

Note that in this definition the direction, tv , is guaranteed to be directed at s because it is in the open half plane of the surface normal of s' due to the definition of *forceAt*.

Finally, the force applied to the surface will become a force on the object if the object is not sufficiently constrained along that direction. If it is constrained, we can assume that a reaction force from the obstacle cancels the force out and thus not bother applying it to the whole object. We assume that rigid objects are not deformable.

Definition 34 (Force from Surface to Object):

forceAppliedToObject(o, tv, src) is true when:

hasContactSurface(o, src, s) AND
 forceAppliedToSurface(s, tv', src) AND
 ;; case 1: o is free to move directly that way
 ((NOT transConstraint(o, tv')) AND equal(tv, tv'))
 OR

```
;; case 2: o can't move directly, but can slide
;;           in the general direction
(transConstraint(o,tv') AND openHalfPlane(tv',tv))
```

In the ramp example (Figure 13), the force of gravity applies to both the block and the wedge. The block's contact surface normal is `Down`, which is in the open half plane of the direction of gravity so a force appears at that surface. The wedge is not sufficiently constrained in the `Down` direction, so it receives the force applied to its upper surface, giving it two forces in the downward direction; one from the block, and one from gravity. The sum of `Down` and `Down` is `down`, so the net force applied to the wedge is `Down`. The ramp is fixed and thus sufficiently constrained, so it does not receive a force from contact with the wedge. The wedge has no forces to transmit to the block across its upward pointing surface normal, so the net force applied on the block is just that of gravity--`Down`. According to prediction of translation (Definition 26), the block will instantaneously move down since it is not constrained in that direction, while the wedge will instantaneously move down and right (`Quad4`) since it is constrained from moving down and from moving down and left (`Quad3`). (Friction would normally cause the block to move `Quad4` as well, but neither friction nor gravity are assumed by default.)

A force at a surface may also apply a torque if the rotational origin lies on either side of the point of contact:

Definition 35 (Torque from Surface to Object):

$\text{torqueAppliedToObject}(o, rv, src)$ *is true when:*

$\text{hasContactSurface}(o, src, s)$ AND
 $\text{hasContactSurface}(src, o, s')$ AND
 $\text{surfaceContact}(s, s')$ AND
 $\text{forceAtSurface}(s, tv, src)$ AND
 $\text{rotationalOrigin}(o, p)$ AND
 $\text{qualitativeVectorBetween}(s, p, tv')$ AND
 $\text{surfaceNormal}(s', tv'')$ AND
 $\text{openHalfPlane}(tv''', tv')$ AND
 $\text{rotate90}(tv'', tv''', rv)$

In the saloon door example (Figure 13), the force at the arm points `Right`, and the direction from the contacted surface of the door to its rotational origin is up and right (`Quad1`), which is in the open half plane counterclockwise from `Right`, therefore the door receives a CW torque from the arm.

3.4.2 Transmission of Force via Direct Connection

Forces are transferred through a direct connection in the same method as constraints. If two objects are `directlyConnected`, a force/torque on one becomes a force/torque on the other.

3.4.3 Assumed Forces and Torques

Sometimes a force or torque should just be assumed to be working on an object, as in our ramp example where we assume gravity is acting on the block and wedge. In Design Coach, this happens when a user draws a force arrow annotation. In this case, the system creates an assumed force relationship.

Definition 36 (Assumed Force or Torque): $\text{forceAssumed}(o, tv, src)$ means the system may apply a force on object o in the direction of tv . Similarly, $\text{torqueAssumed}(o, rv)$ means the system may apply a force on object o in the rotational direction, rv . src is the source of the assumed force/torque, e.g. gravity or a force/torque annotation.

Note the word “may” in Definition 36. This is because it is still unclear if a $\text{force/torqueAppliedToObject}$ relationship should be present. If the object is constrained, then we can assume that a reaction force will cancel out a component or even the entire the assumed force. As with motion, if an assumed force encounters a constraint, it will be redirected elsewhere in the open half plane, or be entirely zeroed out.

Definition 37 (Application of Assumed Force):

$\text{forceAppliedToObject}(o, tv, src)$ is true when:

$$\begin{aligned} & \text{forceAssumed}(o, tva, src) \text{ AND} \\ & ((\text{NOT transConstraint}(o, tva) \text{ AND equals}(tva, tv)) \\ & \text{OR } (\text{NOT transConstraint}(o, tv) \text{ AND} \end{aligned}$$

`openHalfPlane(tv, tva)))`

The torque case is simpler, since there is no optional extra directions for the torque to redirect into.

Definition 38 (Application of Assumed Torque):

`torqueAppliedToObject(o, rv, src)` *is true when:*

`torqueAssumed(o, rv, src)` AND

NOT `rotConstraint(o, rv)`

Chapter 4

Extensions to Qualitative Mechanics

This chapter introduces more significant extensions to QM for Design Coach. These extensions fall into two categories. The first are algorithms which allow QM to use sketched input. The surface contact detection algorithm is one of two algorithms we shall present. The second are representations for new types of objects. Early in the development of the Design Coach QM, we performed a survey of past mechanical projects from the engineering design course. The survey revealed that QM could predict the behavior of a significant proportion of the mechanical designs using rigid objects alone, and with the inclusion of springs and gears, we could cover nearly 40% more of the designs surveyed (Wetzel & Forbus, 2008).

The chapter begins with a description of the process used to identify surface contact information from a hand-drawn sketch. It then describes three new types of objects: springs, gears, and cords, including the process used to identify different types of cord connections in a hand-drawn sketch.

4.1 Surface Contact Detection

For QM, we need to identify each contact surface and the direction of its normal.

Recall that glyphs can be decomposed into edges and junctions, which gives us three possible types of contact between them: edge-to-edge, edge-to-junction, and junction-

to-junction. The `detectSurfaceContact` algorithm in Figure 15 takes two glyphs as input and returns these three types of contacts and their normal.

```

1. detectSurfaceContact (Glyph1, Glyph2) :
2.   Decomp1 <- edgeDecomp (Glyph1)
3.   Decomp2 <- edgeDecomp (Glyph2)
4.   Contact1 <- contactDecomp (Decomp1, Decomp2)
5.   Contact2 <- contactDecomp (Decomp2, Decomp1)
6.   JunctionPairs <- contactJunctionPairs (Contact1, Contact2)
7.   E-EContacts <- edgeEdgeContacts (Contact1, Contact2,
8.                                     JunctionPairs)
9.   E-ENormals <- []
10.  For EdgePair in E-EContacts:
11.    E-ENormals.add(edgePairSurfaceNormal (EdgePair,
12.                                           JunctionPairs))
13.  J-Contacts <- filterJContacts (JunctionPairs)
14.  J-EContacts <- junctionEdgeContacts (J-Contacts)
15.  J-ENormals <- []
16.  For EdgePair in J-EContacts:
17.    J-ENormals.add(edgePairSurfaceNormal (EdgePair) )
18.  J-JContacts <- setDifference (J-Contacts, J-EContacts)
19.  Return [E-EContacts, E-ENormals, J-EContacts, J-ENormals,
20.          J-JContacts]

```

Figure 15: Surface contact detection algorithm, top level

Here `edgeDecomp` is the edge decomposition algorithm described in Lovett et al. (2006). We also employ a limited scribble fill detection to eliminate excess edges from the initial decompositions. This is discussed further in Section 6.3.1. Figure 16 shows the resulting edges and junctions when `edgeDecomp` is used on the ramp example.

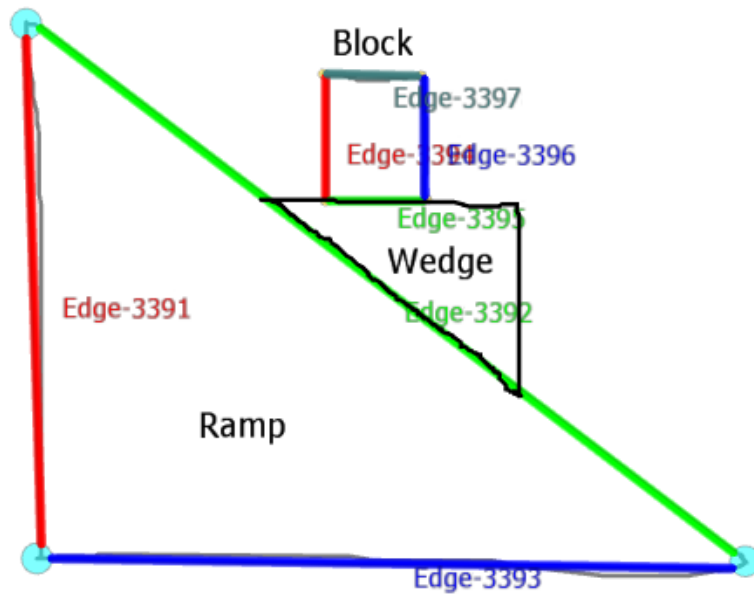


Figure 16: Edges and junctions found using edge decomposition (`edgeDecomp`)

The routine shown in Figure 17 computes the surface contact decomposition between a pair of glyphs by adding new junctions at the points of intersection between the perimeters edge cycles of their original decompositions. `intersection` uses geometry to find the point of overlap between the edge of the first object and the intersecting edge or junction from the second object. A graphical example is given in Figure 18.

```

1. contactDecomp(Decomp1,Decomp2):
2.   ContactDecomp1 <- Decomp1
3.   IntersectingPoints <- []
4.   For edge1 in perimeterEdges(Decomp1):
5.     IntersectingJunctions <- []
6.     For junction in perimeterJunctions(Decomp2):
7.       If rcc8Connected?(edge,junction) and
8.         not(rcc8Connected?(jctA(edge1),junction) and
9.           not(rcc8Connected?(jctB(edge1),junction):
10.      IntersectingJunctions.add(junction)
11.   IntersectingEdges <- []
12.   For edge2 in perimeterEdges(Decomp2):
13.     If rcc8Connected?(edge1,edge2) and
14.       not setIntersection(endJunctions(edge2),
15.                             intersectingJunctions)
16.         and
17.           not(rcc8Connected?(jctA(edge1),edge2) and
18.             not(rcc8Connected?(jctB(edge1),edge2):
19.      IntersectingJunctions.add(edge2)
20.   For object in IntersectingJunctions + IntersectingEdges:
21.     IntersectingPoints.add(intersection(edge1,object))
22.   For Intersection in IntersectingPoints:
23.     ContactDecomp1.addNewJunction(Intersection)
24.   Return ContactDecomp1

```

Figure 17: Routine for creating the surface contact decomposition

In these routines, `rcc8Connected?` means the two junctions/edges are close enough together to be considered topologically connected.

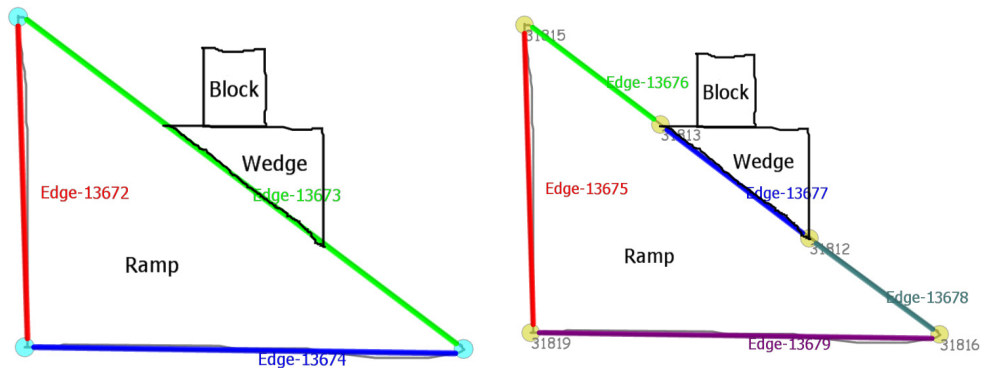


Figure 18: The original edge decomposition of the Ramp (left) gets new junctions at the points where junctions of wedge intersect it (right). (contactDecomp)

Once the surface contact decompositions have been computed for both glyphs, the junctions which overlap are involved in points or regions of contact. First, all of the pairs are collected (Figure 19). `uniqueClosestPairs` sorts the pairs by the distance between their junctions and then guarantees that each junction is only party to one pair by removing any more distant pairs involving that junction.

```

1. contactJunctionPairs (Decomp1, Decomp2) :
2.   JunctionPairs <- []
3.   For Junction1 in junctions(Decomp1) :
4.     For Junction2 in junctions(Decomp2) :
5.       If rcc8Connected?(Junction1, Junction2) :
6.         JunctionPairs.add([Junction1, Junction2])
7.   Return uniqueClosestPairs(JunctionPairs)

```

Figure 19: Routine for finding the pairs of junctions in contact between two surface contact decompositions.

The next step is to find the edge-edge contacts, as done by the routine in Figure 20. For example, an edge-edge contact is detected between ramp and wedge on the right-hand

side of Figure 18 because the endpoint junctions of the ramp's new edge contact the endpoints of the ramp's bottom-left edge.

```

1. edgeEdgeContacts(Decomp1,Decomp2, ContactJunctionPairs):
2.   E-EContacts <- []
3.   For edge1 in exteriorEdges(Decomp1):
4.     For edge2 in exteriorEdges(Decomp2):
5.       If ([edge1.J1,edge2.J2] in ContactJunctionPairs and
6.         [edge1.J2,edge2.J1] in ContactJunctionPairs) or
7.         ([edge1.J1,edge2.J1] in ContactJunctionPairs and
8.         [edge1.J2,edge2.J2] in ContactJunctionPairs):
9.         E-EContacts.add([edge1,edge2])
10.  Return E-EContacts

```

Figure 20: An edge-edge contact is present when the junctions at the endpoints of the edges are in contact.

The angle of the contacting edge will determine the direction of its normal. Since the edges are exterior edges, the simplest method for finding the normal would be to rotate the edges angle 90 degrees towards the outside of the shape. However in some cases that will produce the wrong result. Figure 21 shows an example of an inward contact with an exterior edge. If the normal was taken to be the exterior of the armrest's edge it would face away from the contact rather than towards it.

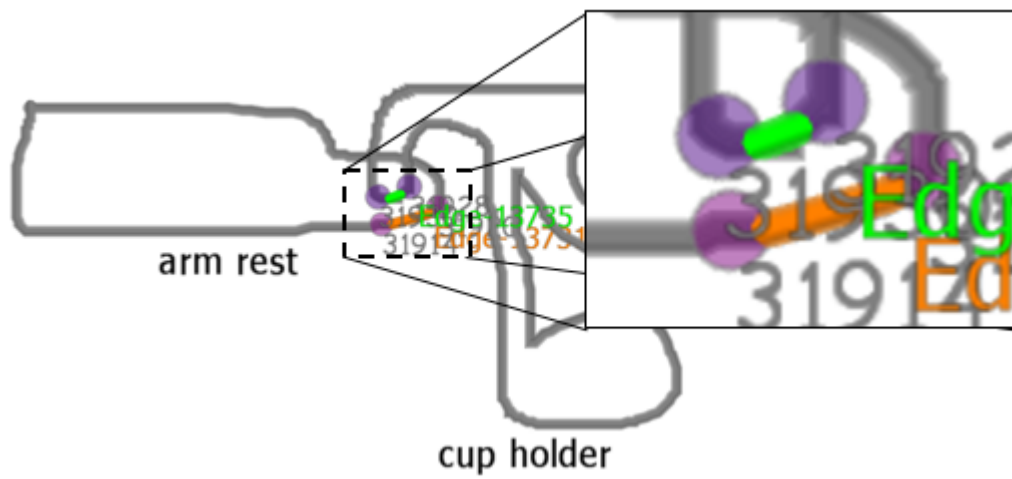


Figure 21: The highlighted contact edge of the arm rest is an example of an inward facing contact.

To compensate for this, our surface normal calculations take into account if the contact surface is inward or not as done in the routine in Figure 22.

```

1. edgePairSurfaceNormal(edgePair, JPairs):
2.   # if this is a pair of edges, base normal off the
3.   # straighter one
4.   If type(edgePair[0]) == Edge and type(edgePair[1]) == Edge:
5.     straighterEdge <- straightest(edgePair[0], edgePair[1])
6.     otherEdge <- edgePair.remove(straighterEdge)[0]
7.     If inwardContact?(straighterEdge, otherEdge, JPairs) and
8.         getInwardContact(straighterEdge, otherEdge) ==
9.             straighterEdge:
10.      Return (angle(straighterEdge) - 90) mod 360
11.     Else:
12.      Return (angle(straighterEdge) + 90) mod 360
13.   Else If type(edgePair[0]) == Edge:
14.     If inwardContact?(edgePair[0], edgePair[1], JPairs):
15.      Return (angle(edgePair[0]) - 90) mod 360
16.     Else:
17.      Return (angle(edgePair[0]) + 90) mod 360
18.   Else If type(edgePair[1]) == Edge:
19.     If inwardContact?(edgePair[1], edgePair[0], JPairs):
20.      Return (angle(edgePair[1]) - 90) mod 360
21.     Else:
22.      Return (angle(edgePair[1]) + 90) mod 360

```

Figure 22: The direction surface normal of an edge contact is the edges direction (from end to end) rotated 90 degrees toward the direction of contact.

The angle of edges is always measured clockwise, and the exterior edges are directed, such that they go clockwise around the shape. So to rotate the angle 90 degrees towards the outside of the shape, 90 is added. If the contact faces inwards, then the angle must be rotated counterclockwise to the normal, thus 90 is subtracted.

To determine if the contact faces inwards, the routine in Figure 23 is used.


```

1. inwardContact?(Edge1,Edge2,ContactJunctionPairs):
2.   Return [edge1.J1,edge2.J1] in ContactJunctionPairs and
3.     [edge1.J2,edge2.J2] in ContactJunctionPairs
4.
5. getInwardContactEdge (Edge1,Edge2):
6.   J1A <- Edge1.J1
7.   J1B <- Edge1.J2
8.   J2A <- Edge2.J1
9.   J2B <- Edge2.J2
10.  incomingEdge1 <- perimeterEdgeEnters (J1A)
11.  incomingEdge2 <- perimeterEdgeEnters (J2A)
12.  departingEdge1 <- perimeterEdgeLeaves (J1A)
13.  departingEdge2 <- perimeterEdgeLeaves (J2A)
14.  EvidenceA <- 0
15.   If incomingEdge1 != nil and incomingEdge2 != nil:
16.     EvidenceA <- (angle(incomingEdge1 - incomingEdge2))
17.  EvidenceB <- 0
18.   If departingEdge1 != nil and departingEdge2 != nil:
19.     EvidenceB <- (angle(departingEdge2 - departingEdge1))
20.  If (EvidenceA + EvidenceB) > 0:
21.   Return Edge1
22. Else:
23.   Return Edge2
24.
25. inwardContact?(SuperEdge, Junction):
26.  closestJunction <- closestJunctionTo (SuperEdge, Junction)
27.  subEdgeA <- subedgeEnters (SuperEdge,closestJunction)
28.  subEdgeB <- subedgeLeaves (SuperEdge,closestJunction)
29.  angleA <- angleFrom(closestJunction,subEdgeA)
30.  angleB <- angleFrom(closestJunction,subEdgeB)
31.  otherEdges <- [perimeterEdgeEnters (Junction),
32.                perimeterEdgeLeaves (Junction)]
33.  otherAngles <- []
34.  For edge in otherEdges:
35.    otherAngles <- angleFrom(closestJunction,edge)
36.  angleEnclosureDir <- angleEnclosure (angleA,angleB,
37.                                       otherAngles)
38.  If angleEnclosureDir == CW:
39.    Return False
40.  Else:
41.    Return True

```

Figure 23: Routines for determining if a contact is inward or not. Two different inwardContact? methods are needed for edge-edge and edge-junction contact.

Determining the inwardness of an edge-to-edge contact is straightforward; because the exterior edges move clockwise around the shape if the contact junctions at the end of the edges are paired first-to-first and second-to-second, we know one of the edges must contact on its inward side. To figure out *which* edge is the inward one (`getInwardContactEdge`), an estimate is made by comparing the angles of the neighboring pairs of edges on either side of that edge. The edge whose neighboring edges form a larger concave around the opposing edge's neighbors is assumed to be the inward facing edge.

Determining inwardness in the edge-to-junction contact case is a bit more complex. Recall the original surface contact decomposition added a junction at the point of contact, segmenting the surface of the ramp into two edges. When it was determined that this junction was a junction-edge contact, the edges were merged into a single super-edge. Once again we take advantage of the invariant that exterior edges are directed clockwise around the outside of the shape. Assuming these are solid objects, the two exterior edges of the block leaving the junction should both extend away from the super edge (rather than crossing it). Let us call the angle of the inbound sub-edge as measured away from the junction "angle A"; the angle of the outbound sub-edge as measured away from the junction "angle B"; and the edges of the block "1" and "2". (See Figure 24.)

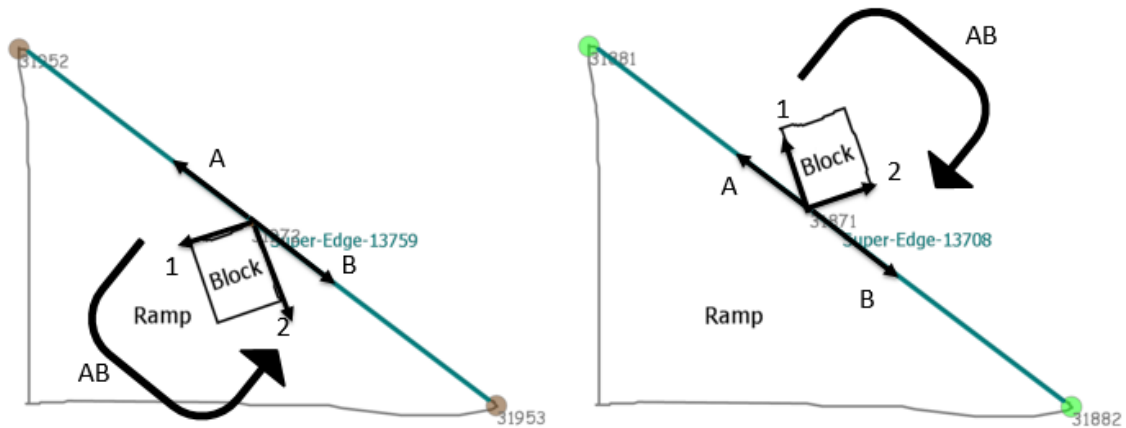


Figure 24: Angle enclosure can be used to determine if an edge-to-junction contact is inward or not. If the perimeter moves clockwise from A to B, then the contact is inward whenever enclosing angle AB travels counterclockwise.

Since 1 and 2 are both on the same side of the super edge, there exists an angle from angle A to angle B, “angle AB”, which encloses 1 and 2. Since going from edge A to edge B travels clockwise about the shape, if the angle AB travels clockwise, then 1 and 2 will be outside the shape. If angle AB travels counterclockwise, then 1 and 2 will be on the inside of the shape. Therefore, if `angleEnclosureDir` returns the direction of enclosing angle AB, then whenever it returns a counterclockwise result, the edge contact faces inward.

The inward contact detection adds a limitation; the edges in question must be part of a closed loop to guarantee that the perceptual edge decomposition system assigns the edges in clockwise order around the outside of the shape. If the shape is open, then this is not guaranteed. Instructors preferred that their students sketch without using stick figures, so making inward contact detection work with stick figures was not

necessary for our intervention. Not all students followed those instructions, however; examples can be found in Section 6.3. Ways to address these cases in the future can be found in Section 8.2.3.

4.2 Springs

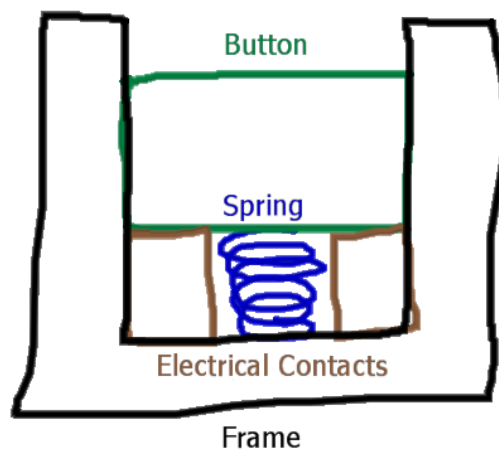


Figure 25: An example of a mechanism involving a spring.

Springs are one of the new non-rigid objects added in Design Coach QM. Springs can be connected to up to two other objects, one at each end. Springs may be in one of three states: compressed, stretched, or relaxed. The state is specified by the user via assigning collections from the KB. Depending on the state, a spring may apply a force to an object:

Definition 39 (Force Applied by Compressed Spring):

`forceAppliedToObj(o, tv, spg)` is true when:

`connectedToDirectly(o, spg)` AND

`qualitativevBetween(o, spg, tv) AND`
`isa(spg, Compressed)`

Definition 40 (Force Applied by Stretched Spring):

`forceAppliedToObj(o, tv, spg)` *is true when:*

`connectedToDirectly(o, spg) AND`
`inverseVector(tv, tvi) AND`
`qualitativevBetween(o, spg, tvi) AND`
`isa(spg, Stretched)`

Relaxed springs do not apply any force to their connected objects. Unlike rigid objects, the springs in this model do not create any constraints on motion for their connected objects. In the example pictured in Figure 25, the button will have a force applied upwards from the spring because the spring been labeled `Compressed`.

4.3 Gears

In the simplest sense, gears are merely rigid objects of a certain shape, and could be modeled as such. They were modeled this way in the Clock program using Nielsen's QM (Forbus et al., 1991). However in sketching it is difficult to draw a gear such that the surfaces would interact as desired. A more abstract representation for gears avoids the need for the gear to be drawn accurately.

Definition 41 (Gear): *A Gear is a member of the set of objects which are gears.*

Additionally, for convenience the system assumes the origin of rotation for a gear is its center, unless another origin is given.

When two gears interlock to rotate against one another, they are *enmeshed*. For sketched input, we infer enmeshment based on contact alone, rather than based on the geometry of the shapes.

Definition 42 (Enmeshed): $\text{enmeshed}(g1, g2)$ is true when the teeth of gear 1 are interlocked with the teeth of gear 2.

Gears transfer rotational constraints to their enmeshed neighbors.

Definition 43 (Constraint Transfer through Enmeshment):

$\text{rotConstraint}(g1, rv)$ is true when:

$\text{enmeshed}(g1, g2)$ AND
 $\text{fixedAxisObject}(g1)$ AND
 $\text{inverseVector}(rv, rvi)$ AND
 $\text{rotConstraint}(g2, rvi)$

Similarly, enmeshed gears transfer torque to each other.

Definition 44 (Torque Transfer through Enmeshment):

$\text{torqueAppliedToObject}(g1, rv)$ is true when:

$\text{enmeshed}(g1, g2)$ AND
 $\text{fixedAxisObject}(g1)$ AND
 $\text{inverseVector}(rv, rvi)$ AND
 $\text{torqueAppliedToObject}(g2, rvi)$

An example is shown in Figure 26. A counterclockwise torque is applied to gear 1, which causes an opposite torque on gear 2. Gear 2 is enmeshed with gear 3, so gear 3 receives a counterclockwise torque. The wheel is connected to gear 3, and thus receives the same counterclockwise torque.

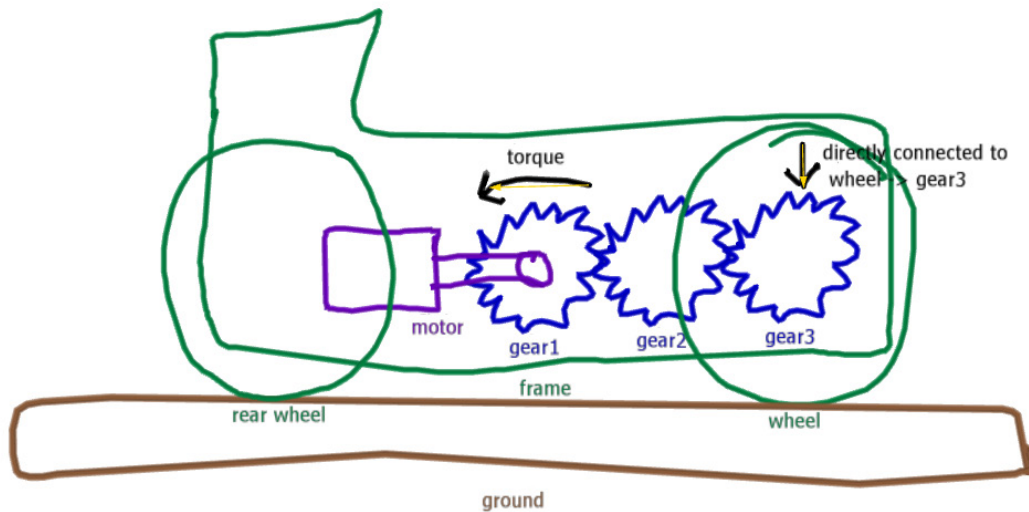


Figure 26: An example of a mechanism involving gears.

4.4 Cords

Cords represent rope, strings, or wires that can create taut connections between objects. These connections can transfer force and constraint; unlike surface contact, cords pull rather than push. Because the goal of Design Coach QM is to predict instantaneous motion and a slack cord has no immediate effect on constraints or forces, we will focus on the case where cords are taut. Detecting if a cord is slack or not is important to understanding designs involving cords. Because the interesting case for QM is the taut

case, that will be the focus of this work. Distinguishing slack cords from taut ones is discussed in Chapter 8. Given they are always taut, it would simplify things if the system could assume all cord connections produce the same constraint/force transfer; however, tautness alone is not sufficient. For instance, if the cord is arranged tautly around an object which is free to move in response to the force on the cord, the force would move the intermediate object instead, leaving the fate of the object at the far end ambiguous. To simplify things, the cord representation in Design Coach QM is organized around handling two cases:

1. A taut cord connects two objects, and does not travel through or around any objects such that the cord can push them away from itself. Since this arrangement is similar to a kinematic pair, it will be referred to as a *kinematic cord connection*.
2. A taut cord connects two objects, and there are one or more movable pulleys between them.

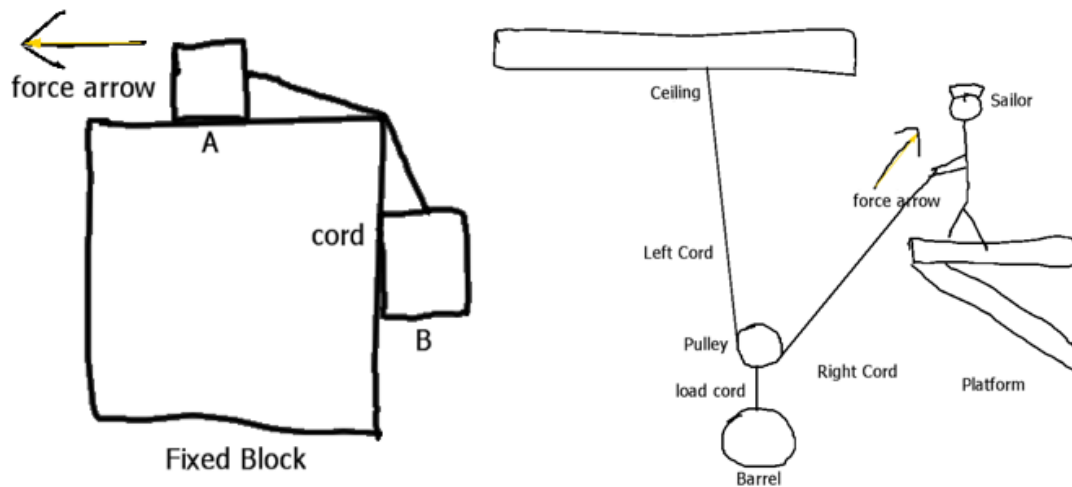


Figure 27: A sketch with a kinematic cord connection (left) and one with a movable pulley (right).

Figure 27 shows an illustration of both cases. The second case is limited to pulleys because this includes all the cases in the mechanism corpus for this thesis. Generalizing to objects other than pulleys will be discussed in future work (Section 8.2.2, pg. 175). With these restrictions, the problem of finding the forces and constraints resulting from cords becomes:

1. Identify the topology of the cord system.
2. Find any kinematic cord connections, and their forces/constraints
3. Find any movable pulleys, and their forces/constraints

The next sections describe the new object representations, cord system topology, and force and constraint transfers for the two cases handled.

4.4.1 Cords and Pulleys

The following definitions form the representation for a cord system in Design Coach QM. The first new category of object required is cords.

Definition 45 (Cords): *A Cord⁴ is a thin, unstretchable length of substance similar in behavior to a string, rope, or wire.*

When a cord goes around a corner or a pulley, it forms two cord segments with different directions. Determining the direction of the last length of cord is crucial to determining how that cord affects the connected object. To enable this, the cord is divided into cord segments.

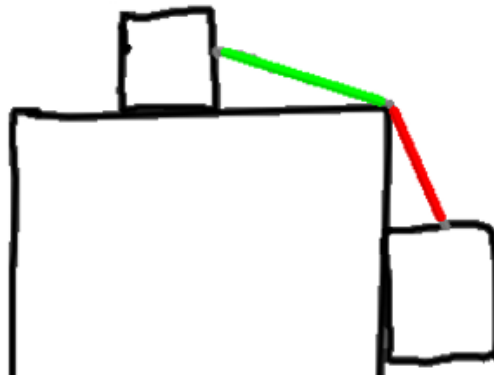


Figure 28: The cord glyph is divided into two segments.

Definition 46 (Cord Segments): *A CordSegment is a portion of a Cord whose ends are each either: where the cord terminates in an object, where the cord bends around a corner, or where the cord enters a pulley.*

⁴ For generality, the actual Cyc collection used is `CordLikeArtifact`. We use `Cord` in this document for brevity.

We derive cord segments from the edges in the edge decomposition of glyphs labeled `Cord`. An illustration is given in Figure 28. The algorithm for finding edges and their connections will be described in Section 4.5.

The final category of object required is the pulley.

Definition 47 (Pulleys): *A Pulley is a RigidBody with one or more grooves for Cords to travel around it.*

At the level of the QM theory, the number of blocks is not limited. However, all the pulleys presented in the mechanism corpus of this thesis take either one cord (i.e. the pulley is a single block) or two cords (i.e. a double block). UI restrictions make it challenging but not impossible to rig up larger blocks as explained in the next section.

4.4.2 Cord System Topology

The following relationships describe the topology of cord systems. The first relationship keeps track of which cord is the parent of a given segment. This is determines which cord segments are part of a kinematic cord connection.

Definition 48 (Cord Segment of Cord): *`cordSegmentOf(cs, c)` is true when `cs` is a `CordSegment` that is part of `Cord c`.*

The end of a cord segment could connect to an object, or be the point where the cord enters a sheave (one of the grooved slots) of a pulley.

Definition 49 (Connected at End of Cord): $\text{connectedAtEnd}(cs, o)$ is true

when one end of $\text{CordSegment } c$ is connected to object o .

Definition 50 (Cord Segments Enter Pulley): $\text{cordSegEntersPulley}(cs, p)$ is

true when one end of $\text{CordSegment } cs$ enters $\text{Pulley } p$.

For both of the above relations, it is useful to know what direction the cord segment enters or arrives at the object, giving us the following relationships.

Definition 51 (Direction to and from a Cord Connection):

$\text{qvFromCord}(cs, o, tv)$ is true when: cs is connected to or enters object o from

the direction of translational vector tv . $\text{qvToCord}(o, cs, tv)$ is true when cs is

connected to o and leaves it in the direction of translational vector tv .

For pulleys, there are two other necessary topological facts. First, since a pulley might have more than one sheave (and thus more than one cord), the following relationship is used to keep track of which cord segments connect to each other inside a sheave.

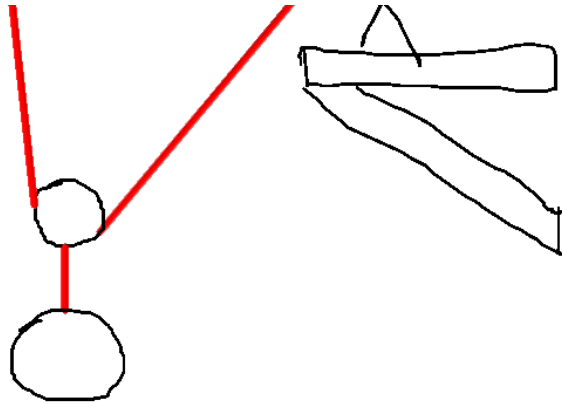


Figure 29: The segments of the left cord and right cord glyphs connect through the pulley, despite the part of the cord inside the pulley not being drawn, and the two cords being separate glyphs. (The system also concludes that the bottom segment does not enter the pulley.)

Definition 52 (Cord Segments Connecting through a Pulley):

$\text{cordSegsConnectThroughPulley}(cs1, cs2, p)$ is true when $cs1$ and $cs2$ are CordSegments of a single cord passing through Pulley p .

Second, as the cord segments connect through a pulley, the cord wraps around the sheave. If both ends of the cord were pulled and/or fixed, the cord would exert a force (and/or confer a constraint) on the pulley. Therefore, the system needs to know the direction from the cord to the pulley.

Definition 53 (Normal of Cord inside a Pulley):

$\text{normalOfInnerCord}(p, cs1, cs2, tv)$ means $cs1$ and $cs2$ connect through pulley p , and the direction from the cord inside p to the center of the p is tv . It is true when:

$(qvToCord(p, cs1, tv) \text{ AND } qvToCord(p, cs2, tv))$

OR $(qvToCord(p, cs1, tv) \text{ AND } qvToCord(p, cs2, tv2) \text{ AND}$

`rotate45(tv, tv2, rv) AND diagonal(tv)`

OR `(qvToCord(p, cs1, tv1) AND qvToCord(p, cs2, tv2) AND`

`rotate45(tv1, tv, rv1) AND rotate45(tv2, tv, rv2)`

The above formula for determining the normal can be explained as: if the directions to both cord segments is 0 or 90, the normal is the vector between them. Otherwise, it is the diagonal vector between them. This way, normal can be determined even if the interior cord is not drawn (Figure 29). Graphical examples are given in Figure 30. Note that the normal is not defined for the case where the difference between the vectors is greater than or equal to 180 degree, because in that case pulling the cord won't affect the pulley.

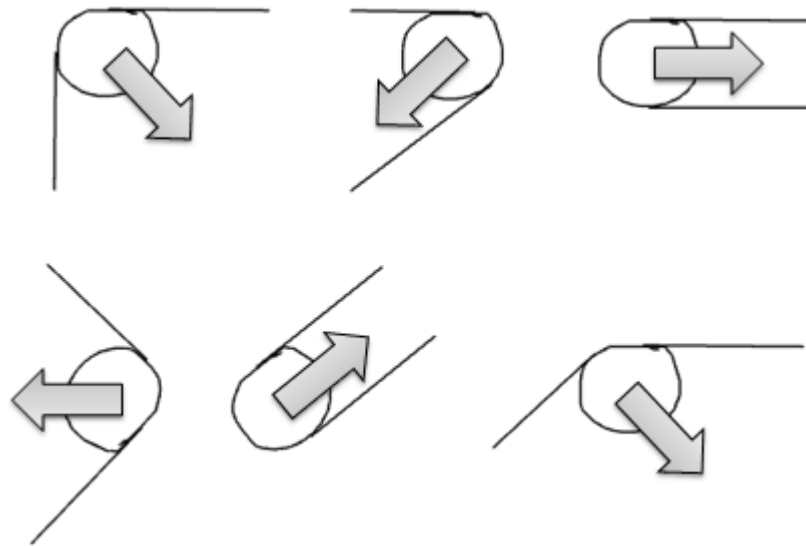


Figure 30: Examples of normal vectors of cords inside pulleys

With the aforementioned relationships available, the system can use the following three relationships to find the kinematic cord connections and movable pulleys. The first identifies when two cord segments are part of a larger cord.

Definition 54 (Same Cord): $\text{sameCord}(cs1, cs2)$ means cord segments $cs1$ and $cs2$ are both part of the same cord. This is true when:

$\text{cordSegsConnectThroughPulley}(cs1, cs2)$

OR $(\text{sameCord}(cs1, cs3) \text{ AND } \text{sameCord}(cs2, cs3))$

For example, the two upper cord segments in Figure 29 are sameCord , but the bottom cord is not. The second relationship identifies a cord connection.

Definition 55 (Cord Connection): $\text{cordConnection}(o1, o2, cs1, cs2)$ means objects $o1$ and $o2$ are connected at opposite ends of a cord and the connecting cord segments are $cs1$ and $cs2$. It is true when:

$\text{sameCord}(cs1, cs2) \text{ AND}$

$\text{connectedAtEnd}(cs1, o1) \text{ AND}$

$\text{connectedAtEnd}(cs2, o2)$

Note this cord connection relationship does not depend on what happens to the cord between the two objects. If a movable pulley is present, then the connection is not a kinematic cord connection. Thus, the third relationship the system needs to know is when a movable pulley is on the cord.

Definition 56 (Movable Pulley on Cord): $\text{movablePulleyOnCord}(cs, p)$ means cs belongs to a cord which goes through p . It is true when:

$\text{sameCord}(cs, csp)$ AND
 $\text{cordSegEntersPulley}(csp, p)$ AND
 $\text{normalOfInnerCord}(p, csp, csp2, dir)$ AND
 NOT $\text{sufficientlyConstrained}(p, dir)$

With these two relationships available, detecting a kinematic cord connection becomes trivial.

Definition 57 (Kinematic Cord Connection):

$\text{kinematicCordConnection}(o1, o2, cs1, cs2)$ means that $o1$ and $o2$ form a kinematic pair through cord segments $cs1$ and $cs2$. It is true when:

$\text{cordConnection}(o1, o2, cs1, cs2)$ AND
 NOT $\text{movablePulleyOnCord}(cs1, p)$

The algorithm for deriving connectedAtEnd and $\text{cordSegmentEntersPulley}$ relationships from sketched input is discussed in Section 4.5.

4.4.3 Effects of Kinematic Cord Connections

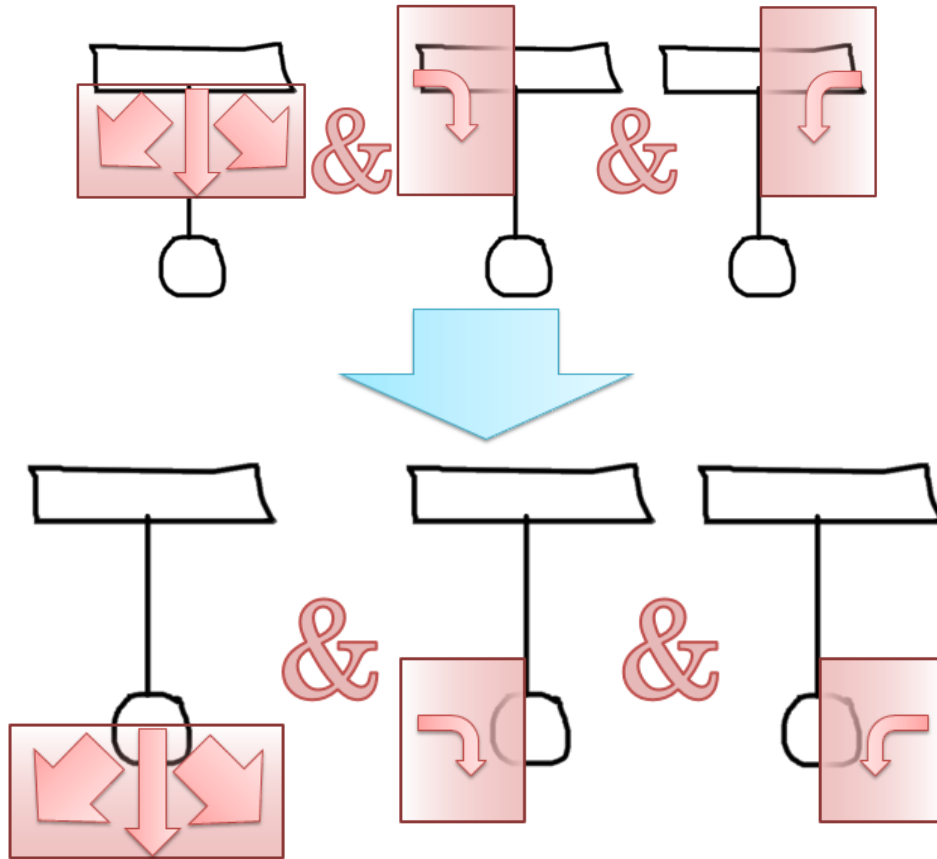


Figure 31: If the three sets of constraints hold for the wall (above) then the three constraints hold for the ball (below).

A kinematic cord connection between objects A and B creates the following effects:

1. If A is sufficiently constrained such that it cannot move in response to the cord being pulled, then B is constrained from moving in directions that would pull the cord.
2. If a force is applied to A such that the cord is pulled, then that force is applied to B if B is not sufficiently constrained.

A kinematic cord connection may also create the following effects:

3. If a force is applied to A such that the cord is pulled, then that force may apply a torque to B.
4. If A is sufficiently constrained such that it cannot move in response to the cord being pulled, then it may create a new rotational origin for B.
5. A force applied to A may create a torque on A about the new rotational origin (case 4) if B is sufficiently constrained.

The constrained directions for sufficient constraint at a cord connection are analogous to those of surface contact:

1. Translational motion into the open half plane centered on the vector out along the cord segment from the point of connection (i.e. the direction of the $q_{vToCord}$ relation).
2. Rotational motion clockwise about any point which lies in the open half plane centered ninety degrees clockwise from the vector out along the cord segment from the point of connection.
3. Rotational motion counter-clockwise about any point which lies in the open half plane centered ninety degrees counter-clockwise from the vector out along the cord segment from the point of connection.

As before, the first set of motions (the translational motions) are referred to as being *sufficiently constrained from translating*. The second and third sets (rotational motions) are *sufficiently constrained from rotating*. A kinematic cord connection to an object

with the above three properties will prevent the motion of the connected object as pictured in Figure 31.

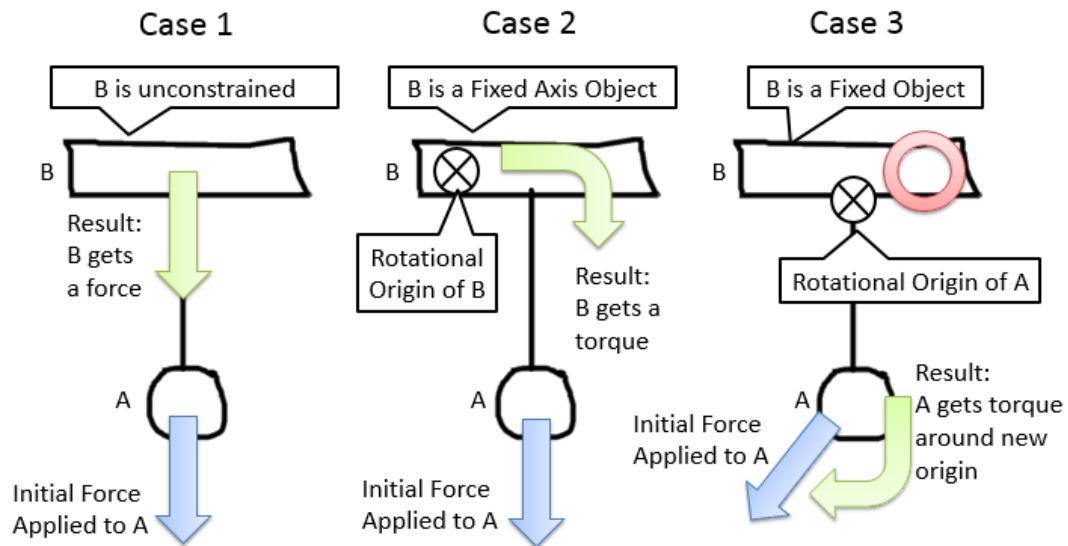


Figure 32: The three possible force transfers for a kinematic cord connection.

The next class of effects are the force transfers. Assuming object A is not sufficiently constrained in the direction of the force, a force applied to A should have the following consequences as pictured in Figure 32:

1. Create a force on B if B is not sufficiently constrained from translating
2. Create a torque on B, if B is not sufficiently constrained from rotating
3. Create a torque on A, if B is sufficiently constrained and A is free to rotate

The remaining effect of a kinematic cord connection is a new rotational origin. If the object does not already have a rotational origin given by the user, the center of the junction at the far end of the object's cord segment in the cord connection relationship becomes its rotational origin (see the rightmost example in Figure 32).

4.4.4 Force Transfer to Movable Pulleys

Movable pulleys may receive force from the cord that moves through them if one end of that cord is pulled and another end is constrained or connected to the pulley itself. This requires two new relationships not defined thus far. The first is the pulling of the cord. The cord may be pulled by an assumed force or another segment of the cord through a pulley.

Definition 58 (Force Assumed on a Cord Segment Directly):

$\text{forceAppliedToObject}(cs, tv, src)$ means that CordSegment cs receives a force in direction tv from source src . This is true when:

$$\begin{aligned} & \text{forceAssumed}(cs, tv, src) \text{ AND} \\ & \text{qvFromCord}(cs, o, tv2) \text{ AND} \\ & \text{axisAligned}(tv, tv2) \end{aligned}$$

Definition 59 (Force Applied to a Cord Segment through Pulley):

$\text{forceAppliedToObject}(cs, tv, src)$ is also true when:

$$\begin{aligned} & \text{cordSegsConnectThroughPulley}(cs, cs2, p) \text{ AND} \\ & \text{forceAppliedToObj}(cs2, tv2, src) \text{ AND} \\ & \text{qvToCord}(p, cs2, tv2) \text{ AND} \\ & \text{qvFromCord}(cs, p, tv) \end{aligned}$$

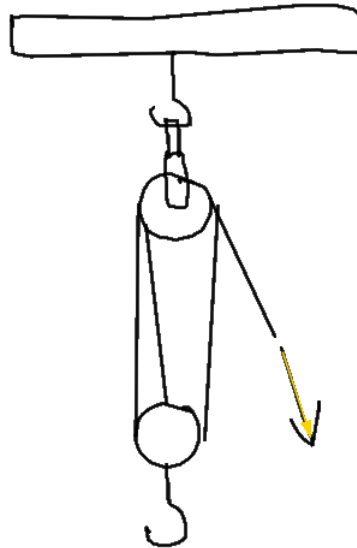


Figure 33: A luff tackle. The bottom pulley is movable.

Now that forces can be applied to the cord segments traveling through our movable pulley, the last thing needed is a relationship to find out if the cord going through the pulley terminates at a constrained point. For a non-ambiguous result, the terminus must either be the pulley itself (as in a luff tackle configuration, see Figure 33) or another object which is sufficiently constrained so that the pulley can be forced against it (see the movable pulley in Figure 27 on pg. 73).

Definition 60 (Movable Pulley Cord Terminus):

$\text{movablePulleyCordTerminus}(p, c, tms)$ means Cord c terminates travels through pulley p and terminates at terminus. It is true when:

$\text{cordSegsConnectThroughPulley}(cs1, cs2, p)$ AND

$\text{cordSegmentOf}(cs1, c)$ AND

$\text{cordSegmentOf}(cs, c)$ AND ## find any cord segment of c

```

connectedAtEnd(cs, o) AND
  (equal(o, p) OR
    (qvToCord(cs, o, tv) AND
      sufficientlyConstrained(o, tv)))

```

With one last definition, the pulley can be forced.

Definition 61 (Force Applied by Cord to Movable Pulley):

forceAppliedToObject(p, tv, src) means pulley p is being forced in direction tv. This true when:

```

normalOfInnerCord(p, cs1, cs2, tv) AND
cordSegmentOf(cs2, c) AND
movablePulleyCordTerminus(p, c, tms) AND
forceAppliedToObj(cs1, tv1, src) AND
qvToCord(p, cs1, tv1) AND
NOT sufficientlyConstrained(p, tv)

```

Examples of Design Coach understanding pulley systems can be found in the evaluation in Chapter 6. A contrast of this method for modeling rope and pulley systems with an earlier method (Pearce, 2001) can be found in Chapter 7.

4.5 Cord Connection Analysis

As was the case for surface contact, cord topology knowledge must be derived from the sketch. This is done by dividing the cord glyph into edges, which then become cord

segments. For each cord segment which touches another object the system needs to know if that segment should assert a `connectedAtEnd` or `cordSegEntersPulley` relationship (or neither). This is done using a new pair of relationships:

Definition 62 (Edge Contacts Tangentially):

$\text{edgeContactsTangentially}(e, o)$ means edge e touches object o tangentially.

Definition 63 (Edge Contacts Non-Tangentially):

$\text{edgeContactsNonTangentially}(e, o)$ means edge e touches object o non-tangentially.

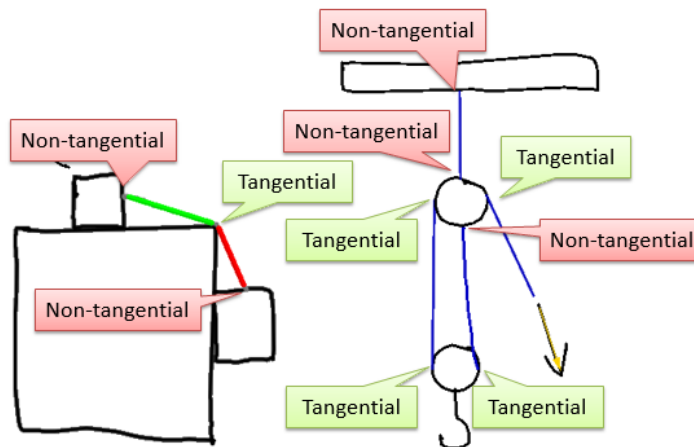


Figure 34: The two types of edge contact relationships.

Figure 34 shows the edge contact relationships graphically. Once these have been established, the topological facts can be derived simply: if a cord makes a non-tangential contact, it creates a `connectedAtEnd` relationship. If a cord makes a tangential contact with a pulley, then it creates a `cordSegEntersPulley`

relationship. Thus in Figure 34, the cords to the sides of the upper pulley will enter it, while the cords above and below it will be `connectedAtEnd` to it.

Figure 38 contains the algorithm for characterizing cord contacts. It takes as input the cord glyph and a set of glyphs which contact it. The set of contacting glyphs is the set of all glyphs representing physical objects that directly touch the cord glyph, minus any glyphs that have a `doesNotTouchDirectly` relationship with the cord glyph (this relation must be added by the user using a relation arrow). Figure 35 shows a graphical example of which glyphs get picked.

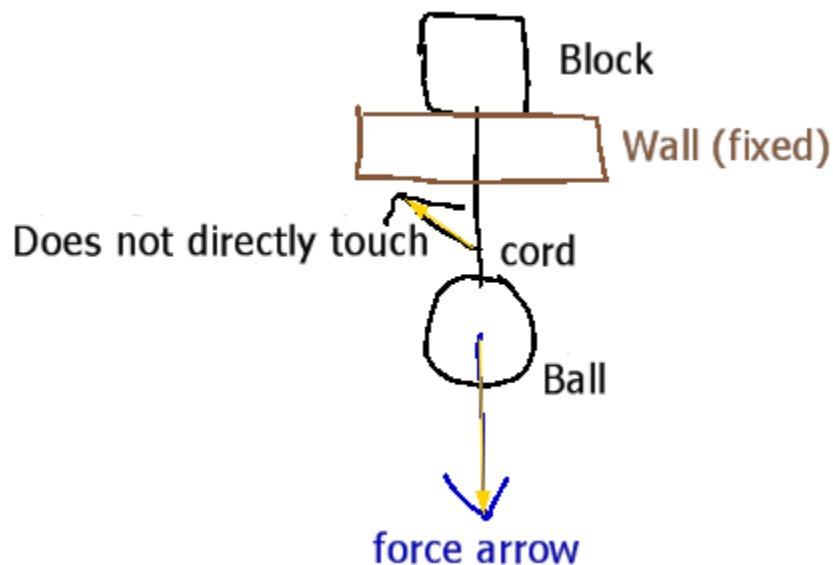


Figure 35: Choosing the set of contact glyphs. In this case, Block and Ball are chosen because they directly touch cord. Wall is ignored because of the “Does not directly touch” relationship, and the contact with the relationship arrow itself is ignored because it does not represent a physical object.

Next the surface contact decompositions are computed. Recall that surface decompositions are created by inserting new junctions at points of contact with other

objects. This is demonstrated on the diagram to the right of Figure 36, where blocks A and B have a new junction added where the cord touches them. For the surface contact decompositions from the cord to the contacted glyphs, each contacted glyph gets its own decomposition. For the reverse (from the objects to the cord) decompositions, the new junctions are accumulated into a single joint decomposition for the cord. This guarantees that the cord is partitioned into one set of edges by the junctions. (If multiple separate decompositions of the cord were used, the same portions of ink would get split different ways depending on the junction placement and there would be no single common set of edges.)

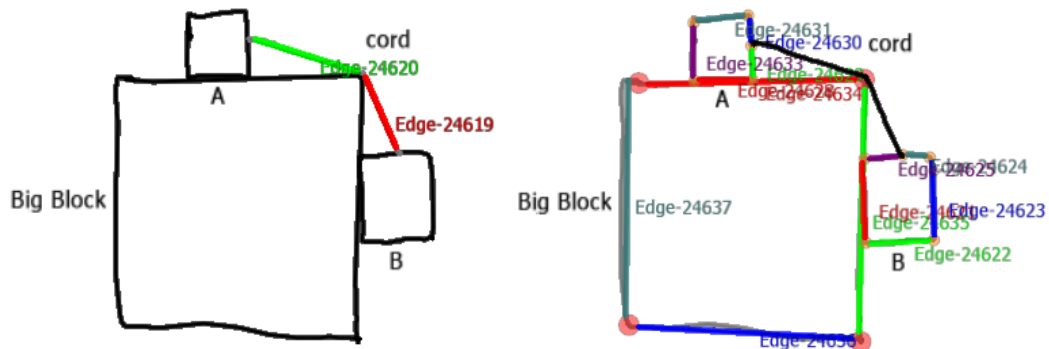


Figure 36: The cord is decomposed using surface contact against all three contacting glyphs at once.

Once the glyphs are decomposed, the algorithm iterates through the pairs of contact junctions. In the example in Figure 36, there are three such pairs: one between the cord and A, one between the cord and B, and one between the cord and the Big Block. For each cord edge entering the junction pair, the difference in angle is calculated between the cord edge and the contacted exterior edges leaving the junction.

For example, Figure 37 shows the junction pair between A and the cord. The cord has one edge entering this junction, Edge-5. The exterior edges of Block A leaving that junction are Edge-1 and Edge-2. To determine if Edge-5 meets Block A tangentially, the difference in angles between Edge-5 and Edges 1 and 2 are computed in the algorithm used in `tangentialContact?` (Figure 38) which checks if the difference in angle between the cord's edge and each of the axes of the contact edges is less than a threshold. For this work 18 degrees was used based on previous perceptual models created within CogSketch for salient angle differences (Lovett, 2012). By this criterion, the contact of Edge-5 in Figure 38 with Block A is non-tangential, while the contact with the corner of the big block is tangential.

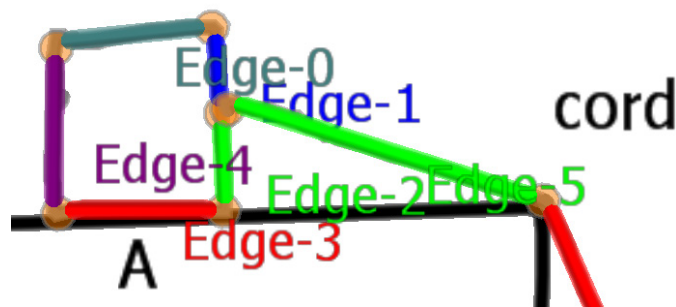


Figure 37: A zoomed-in view of the connection between block A and the cord.

Once the tangentialness of each connection is established, `connectedAtEnd` and `cordSegmentEntersPulley` relationships can be determined as discussed earlier in this section, and the system can use that knowledge to reason out the topology of the cord system.

Three rules are used to determine which pair(s) of cord segment(s) connect through a given pulley.

1. If there are only two cord segments which enter the pulley, they connect through the pulley.
2. If two cord segments belong to the same cord, and they are the only two belonging to that cord entering the pulley, they connect through the pulley.
3. If four cord segments enter a pulley, and one is from cord A, one is from cord B, and two are from cord C, then the segment from cord A and the segment from cord B connect through the pulley.

In Design Coach, if segments are part of the same glyph, then they belong to the same cord. So with the above rules the user may connect any number of cord segments through a pulley provided each pair belongs to a different glyph. Rule #3 only scales up to four cords, so while QM can handle any number of `cordSegmentsConnectThroughPulley` expressions per pulley, with Design Coach there is a soft two-cord maximum. This was sufficient because the examples chosen for the mechanism corpus had at most two cords running through each pulley (i.e. all were single or double blocks). We also do not detect impossible cord arrangements, such as one cord segment having multiple `cordSegmentsConnectThroughPulley` relationships in a single pulley.

The next chapter will explain how these QM representations are used to give feedback on engineering design sketches.

```

1. characterizeEdgeContacts (CordGlyph, ContactingGlyphs) :
2.   EdgeRelations <- [] ##results
3.   CordDecomp <- edgeDecomp (CordGlyph)
4.   ContactingDecomps <- []
5.   For Glyph in ContactingGlyphs:
6.     ContactingDecomps.add(contactDecomp (
7.       CordDecomp, edgeDecomp (Glyph))
8.   CordSurfDecomp <- contactDecomp (ContactingDecomps, Cord)
9.   For Contactor in ContactingDecomps:
10.    JunctionPairs <- contactJunctionPairs (Contactor, Cord)
11.    For JPair in JunctionPairs:
12.      ContactorEdgePair <-
13.        edgesEnteringJunction (Contactor, JPair[0])
14.      CordEdges <- edgesEnteringJunction (Contactor, JPair[1])
15.      For CordEdge in CordEdges:
16.        If tangentialContact? (CordEdge, ContactorEdgePair)
17.          EdgeRelations.add(tangentialRel (CordEdge,
18.                                             Contactor))
19.        Else:
20.          EdgeRelations.add(nonTangentialRel (CordEdge,
21.                                             Contactor))
22.    Return EdgeRelations
23.
24. tangentialContact? (CordEdge, ContactorEdgePair) :
25.   contactAngle1 <- angle (ContactorEdgePair[0])
26.   contactAngle2 <- angle (ContactorEdgePair[1])
27.   cordAngle <- angle (CordEdge)
28.   CCWDiff <- mod (cordAngle-contactAngle1, 360)
29.   CWDiff <- mod (cordAngle-contactAngle2, 360)
30.   Return (or (and CCWDiff > 180-Thresh)
31.            (and CWDiff > 180-Thresh))
32.            (and CCWDiff < 180+Thresh)
33.            (and CWDiff < 180+Thresh))

```

Figure 38: The algorithm for characterizing edge contacts.

Chapter 5

CogSketch Design Coach

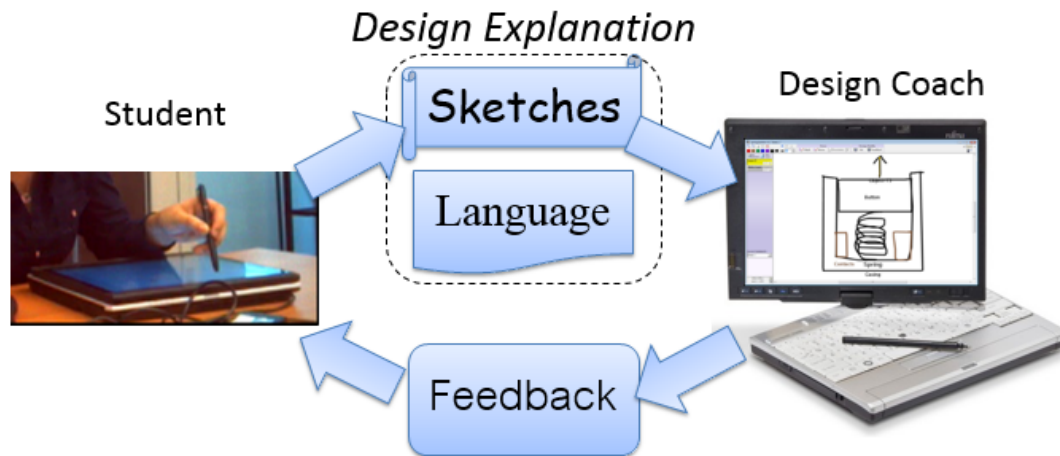


Figure 39: A basic overview of Design Coach

CogSketch Design Coach is a sketch-based educational software system which allows a student to explain their engineering design using a combination of hand-drawn sketches and structured language input, to get on-demand feedback on their explanation. This chapter begins with an example of the system's use and an overview of its architecture. Next, we describe the user input and interface. Then we introduce the two algorithms used to critique the sketch: *state transition verification* and *sequential explanation analysis*. These are followed by a description of a teleological ontology which allows the system to understand input concerning higher-level purposes of engineering designs. The final section describes how the feedback is arranged and displayed to the user.

5.1 Overview

As explained in Chapter 1, we created Design Coach to help first year engineering design students improve their comfort and skill with sketching as a communication tool, as well as a source of feedback on their designs. The following example demonstrates the workflow of Design Coach.

5.1.1 Example



Figure 40: A photograph of the Under Armrest cup holder design

In a homework exercise, a student was asked to explain the design of a detachable cup holder (depicted in the photograph in Figure 40) to Design Coach. (The exercise is described further in Chapter 6.)

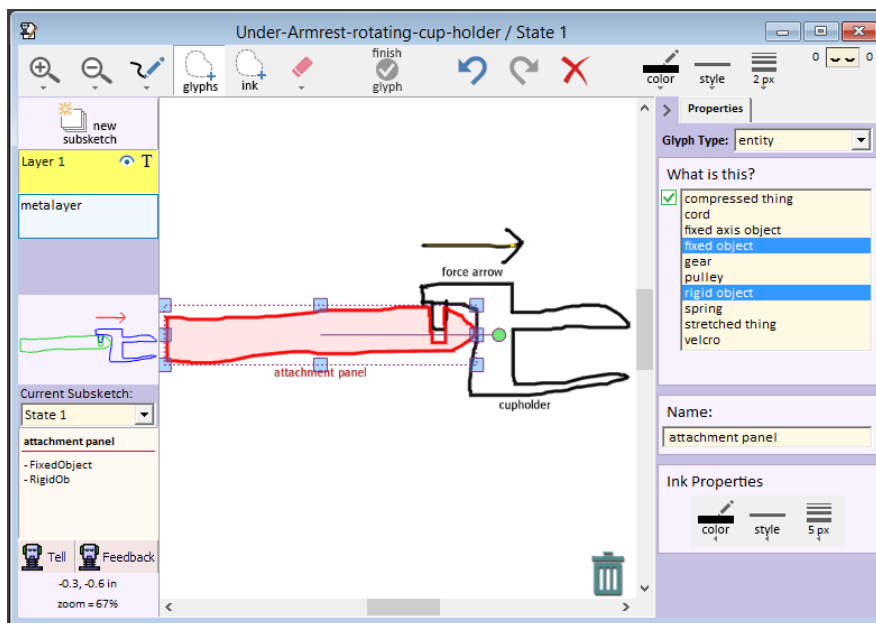


Figure 41: A student’s subsketch depicting the side view of the cup holder. The student chose the “fixed object” and “rigid object” labels for the selected part.

After taking the Design Coach tutorials (which are built into CogSketch), the student begins by drawing the parts of her design as glyphs (Figure 41)⁵. She starts by sketching an orthographic projection of the side view, to keep the motion in a 2D plane. She then adds conceptual labels and names to her glyphs, using the properties pane on the right. She also adds a force arrow to indicate how the cup holder is removed.

⁵ This example is based on a sketch drawn by an anonymous first-year engineering student, but the steps taken have been modified for this example. The original sketch can be found in the explanation corpus.

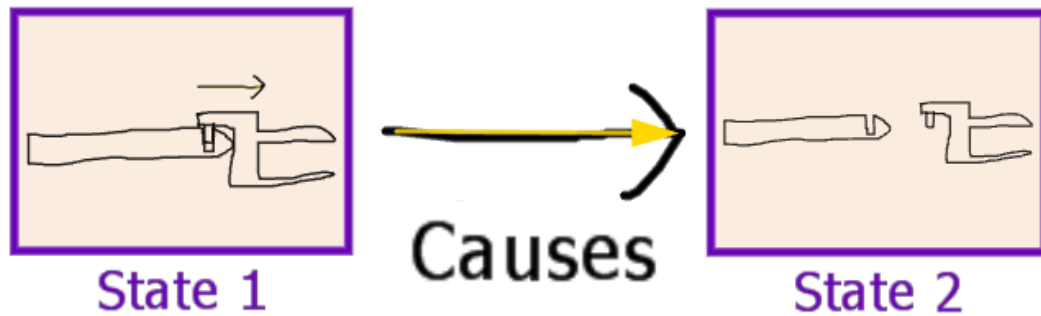


Figure 42: The metalayer after cloning the subsketch, editing it, and adding a `causes` relation.

She then shows Design Coach what should happen next by creating a comic graph. She moves to the metalayer and clones her subsketch to create another state for her design. Once inside her new subsketch, she moves the cup holder glyph to the right to show that it detaches, then returns to the metalayer and adds a `causes` relation arrow between the states, showing that the situation in the first subsketch causes the situation in the second (Figure 42).

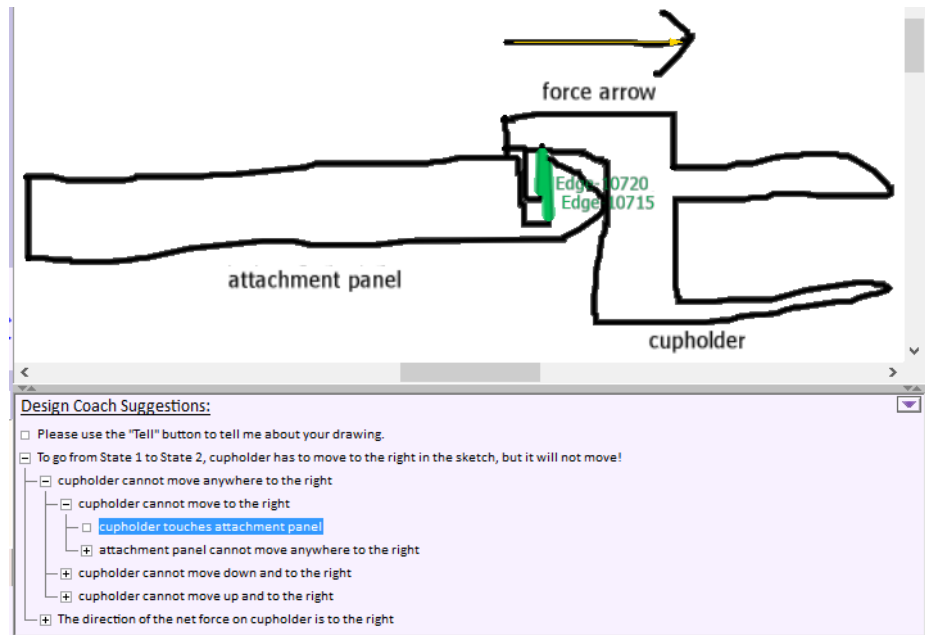


Figure 43: The student expands the second feedback item to reveal its explanation. Selecting feedback highlights referenced parts of the sketch in green.

She then requests feedback using the button in the lower left (Figure 41).

Design Coach sees that the cup holder has been displaced to the right, but by using QM it reasons that, in the first state, the cup holder is sufficiently constrained to its right because of its contact with the attachment panel. Therefore, the Design Coach gives her the following piece of feedback (shown in Figure 43): “To go from State 1 to State 2, cupholder has to move to the right in the sketch, but it will not move!” By clicking the small boxes next to the feedback, she can expand it to display the reasons behind it. Design Coach would also like to know more about the sketch, so it prompts her to use the Tell window to do so. She decides to address this feedback about motion first.

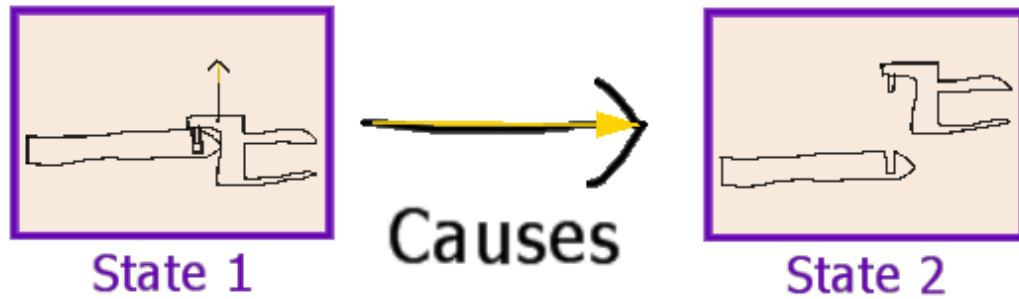


Figure 44: The student revises the sketch

Entering the subsketches again, she changes her sketch so that the cup holder is removed vertically instead by repositioning it in the second state and rotating the force arrow to point upwards in the first state. She then decides to address the other piece of feedback, and tell the Design Coach about how the cup holder detaches from the base and how it rotates.

Tell Design Coach about "Under-Armrest-rotating-cup-holder"...

Type an overall description of your design for human readers in the box below.

The cupholder piece slides and out of the base piece. The cupholder piece can also rotate about that attachment point.

In State 1 ... cupholder detatches from
 attachment pan via interlocking/deinterlocking

In State 1 ... cupholder
 can rotate

Figure 45: The student constructs sentences. Green check marks indicate that no fields are left blank.

She opens the Tell window using the button on the lower right of the interface. She adds two sentences, shown in Figure 45. She builds the sentences by selecting words from the drop down boxes, which are populated with terms from her sketch and the knowledge base. (The box at the top is for students to describe their design in natural language for their instructors to read. We also read these boxes while determining what new vocabulary might be added to the system.)

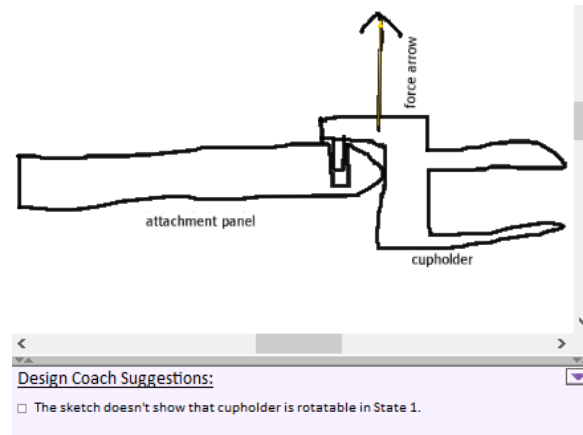


Figure 46: Design Coach gives feedback on the new version with sentences.

She presses the button to request feedback again. Design Coach processes the sentences. The first sentence is about the detachment; Design Coach knows that for detachment, the objects should start out attached and become separated. Design Coach confirms this is happening by checking at the sketch, so it agrees (Figure 46). Also, this time Design Coach sees that the cup holder has been displaced upward and uses QM to confirm that it will move up.

The second sentence says the cup holder is rotatable. Design Coach checks to see if the cup holder is free to rotate. Since no origin of rotation was given, it chooses the center of mass. The contact with the panel sufficiently constrains the cup holder from rotating, so Design Coach responds with feedback saying that it cannot rotate.

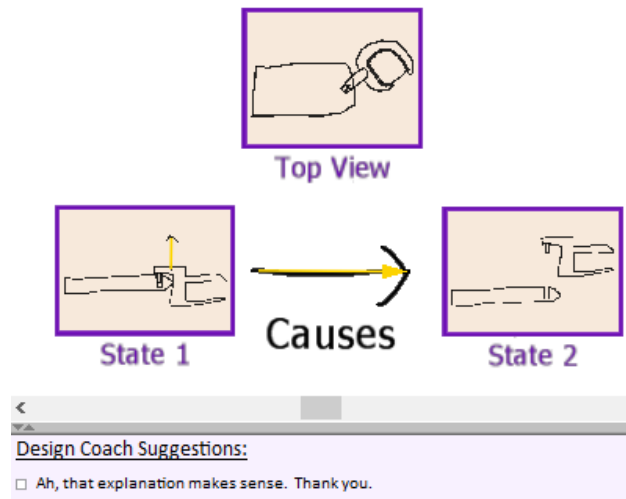


Figure 47: Design Coach gives feedback on the third version.

The student decides to use another view where the cup holder will not be constrained from rotating. She creates a new subsketch called Top View using the button on the left panel and draws the top view. She then returns to the Tell window and changes the context of her sentence from “In state 1” to “In top view” and finishes the sentence. She hits the feedback button, and Design Coach checks the sketch again. This time, every sentence proved to be true and the motion between state 1 and state 2 was also valid, so the system responds that the explanation made sense. (Figure 47)

5.1.2 Architecture

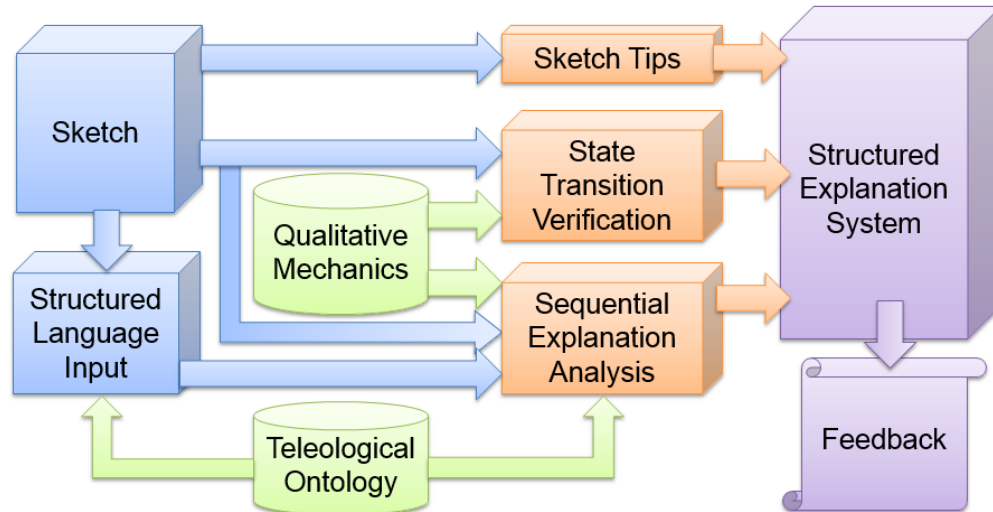


Figure 48: An overview of the architecture of Design Coach. Each arrow means one subsystem is providing data to the other.

Design Coach is organized as shown in Figure 48. This diagram is organized into four columns. The first column (on the far left of the diagram) is user input, which includes the sketch and structured language input. The sketch informs the structured language input because the sketch contains the objects the student will tell the system about.

The second column is the knowledge base, which includes Design Coach QM and the teleological ontology. Design Coach QM was explained in Chapter 3 and Chapter 4. The teleological ontology provides additional vocabulary for use in the structured language input, with the goal of allowing Design Coach to understand and give feedback on the function of the designs as well as their physical behavior. It will be described in Section 5.5.

The third column from the left shows the feedback generators. The one at the top, sketch tips, checks the sketch for user input errors such as disconnected annotation or relation glyphs, forgetting to segment their ink into multiple glyphs, or forgetting to include a fixed object to create a frame of reference. The middle box is *state transition verification*, the first feedback algorithm created for Design Coach (Wetzel & Forbus, 2009a, 2009b). State transition verification looks at the comic graph on the metalayer and uses QM to see if the transitions in the graph make sense. The third box is *sequential explanation analysis*, the newer feedback algorithm which uses multimodal input (Wetzel & Forbus, 2010, 2012). It processes the structured language input sentence by sentence and verifies if each sentence is proven, unproven, or contradictory using QM and the sketched input. The two feedback algorithms will be discussed in Sections 5.3 and 5.4, respectively.

The final column is the feedback generation system, which organizes the feedback and displays it to the user. A key feature is the *structured explanation system* which allows the user to drill down into the feedback and see why the system provided it. The knowledge representations for QM and the teleological ontology stored in the KB include natural language strings which are used to convert the predicate calculus statements into English (connection not shown in diagram). More detail is given in Section 5.6.

5.2 User Input

This section provides details on the input sources Design Coach uses: the CogSketch interface and the structured language input.

5.2.1 Sketches

Design Coach sketches use a custom skin of the CogSketch interface making specific conceptual labels available for the user. The types of available concepts are listed in Table 1. All three types of glyphs (entity, relation, and annotation) are used in Design Coach, as is the metalayer. Each subsketch is expected to have a single layer, therefore the new layer control is hidden in the Design Coach skin. All other user interface features remain available.

Table 1: Lists of Available Concepts in Design Coach Sketches

Glyph Type	Available Labels
Entity Glyph	Rigid Object, Fixed Object, Fixed Axis Object, Spring, Compressed Thing, Stretched Thing, Gear, Cord, Pulley, Velcro
Annotation Glyph	Force Arrow, Rotational Force Arrow, Origin of Rotation
Relation Glyph	Causes (metalayer), Directly Connected To, Does Not Directly Touch

Concepts represented by all of the above labels have been discussed in previous chapters except for one: Velcro. Velcro was added to accommodate more designs from DTC. In the Design Coach ontology, it is a specialization of the collection

AdhesiveMaterial. This concept is relevant in the teleological ontology, which specifies that attachment can be attained using adhesive materials (see Section 5.5).

5.2.2 Structured Language Input

Figure 49: The Structured Language Input

Language is a natural complement to sketching, in the context of communication. The structured language input allows Design Coach to take advantage of multi-modal input without waiting for advances in natural language understanding. In Design Coach, students use the Tell window (Figure 45, pg. 99) to construct restricted sentences using templates that can be rendered into understandable natural language. This structured language and its input interface are based on a similar one used in an earlier system

from the Qualitative Reasoning Group, nuSketch Battlespace (Forbus et al., 2003).

The core structure of these constructed sentences is:

In <context> <subject> <verb> <object>.

An example simple sentence that would be true in our previous example (5.1.1) would be “In State1 cup holder moves up.” Here <context> is the state (i.e. the subsketch) being described. Once a subsketch is selected, the <subject> field is populated with the names of entities represented by glyphs in the sketch. This includes all entity glyphs and none of the annotations/relations. The <verb> field is populated with concepts from QM and the teleological ontology (Table 1). The contents of the <object> field vary depending on the selected verb. If none are available, the <object> field will disappear completely. For instance, in the cup holder example one sentence the student made was “In top view cup holder can rotate.” “Can rotate” is one of the verb options that has no object field. For a list of which verbs allow which object choices, see Table 2. The teleological verb phrases generate an additional <way> field; this will be explained in Section 5.5. Because sentences may introduce new forces or additional information about the types of objects, their order matters. The student can use the green arrow buttons to the left of the sentence to quickly change the order of their statements.

**Table 2: List of verbs, and their available objects in the Tell window;
Italicized verbs are teleological.**

<i>Verb</i>	Available Objects
<i>Moves</i>	Up, Down, Left, Right, Up and Left, Up and Right, Down and Right, Down and Left
<i>Rotates</i>	Clockwise, Counter Clockwise
<i>Is being forced</i>	Up, Down, Left, Right, Up and Left, Up and Right, Down and Right, Down and Left, Clockwise, Counter Clockwise
<i>Is</i>	Rigid, fixed, compressed, stretched, springy, Velcro
<i>Touches</i>	List of entities in the subsketch
<i>Attaches to</i>	
<i>Detaches from</i>	
<i>Adapts to change in size of</i>	
<i>Contains</i>	
<i>Is made more comfortable</i>	None
<i>Does not move</i>	
<i>Does not rotate</i>	
<i>Can Move</i>	
<i>Can Rotate</i>	

Sentences may either be simple or compound. To create a compound sentence, the user clicks the blue arrow button and additional fields are revealed. Structurally, the compound sentence is like joining two simple sentences together.

In <context> <subject1> <verb1> <object1> <conjunct>
<subject2> <verb2> <object2>.

The <context> is shared between the two conjuncts, but they are otherwise separate. The <conjunct> can be either “which causes” or “which prevents”, allowing the user to make causal statements such as “In state 1 cup holder is being forced up which causes cup holder moves up”. Given the user’s choice for <verb1>, there may be

some limitations on which verbs may be used for <verb2>, as well; these are explained in Limitations and Future Work, Section 8.2.5.

As sentences are completed, they are added as facts to the working memory of the sketch. However, Design Coach does not immediately believe the facts in the sentences. What it actually stores is a fact that says the student stated the fact. So in the cup holder example the sentence, “In top view cup holder can rotate.” became a fact like⁶:

```
(studentSays (in-context State1 (rotatable CupHolder)))
```

It is then up to the Design Coach to decide if the fact should be assumed true or not. It uses the feedback algorithm in Section 5.4 to do so, and then runs the following algorithm to check the causal relationships implied by the relation arrows in the comic graph (i.e. on the metalayer).

⁶ In this chapter facts will be formatted in Cyc-style predicate calculus for the benefit of any who want to try the examples out in CogSketch and compare results in the knowledge browser. Note that some terms are abbreviated for conciseness, e.g. *ist-information*.

5.3 State Transition Verification

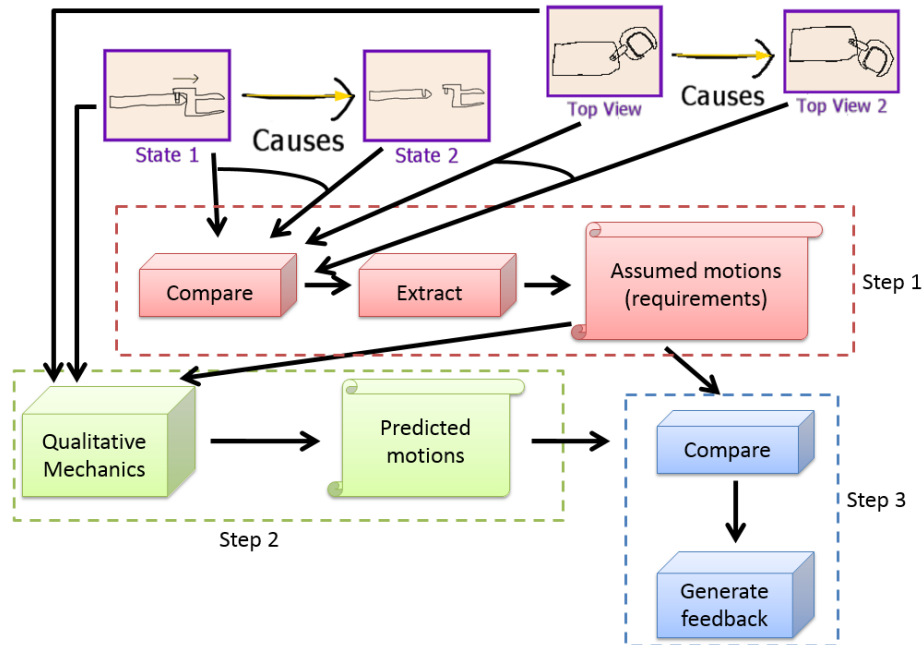


Figure 50: A visual description of the state transition verification algorithm

The first feedback algorithm, state transition verification (STV), is depicted graphically in Figure 50 and in pseudocode in Figure 51. The algorithm begins with a pre-processing step, to ensure all of the surface contact knowledge and cord connection knowledge needed for QM is updated. Then, the subsketch pairs are found by checking working memory for the `causes` relationship created by the relation arrows on the metalayer. In the example pictured in Figure 50, there are two transition pairs (our student became even more ambitious), which are represented by the facts `(causes State1 State2)` and `(causes TopView TopView2)`.

```

1. stateTransitionVerification(Sketch):
2.   For Subsketch in Sketch.subsketches:
3.     updateSurfaceContactKnowledge(Subsketch)
4.     updateCordConnectionKnowledge(Subsketch)
5.   Feedback ← []
6.   For SubsketchPair in getTransitionPairs(Sketch):
7.     For Requirement in deduceRequirements(SubsketchPair):
8.       For Verification in verifyRequirements(Requirement):
9.         If Verification == Requirement:
10.            Feedback.add(recordSuccess(Requirement,
11.                                       Verification))
12.         Else:
13.            Feedback.add(recordFailure(Requirement,
14.                                       Verification))
15.   Return Feedback

```

Figure 51: State transition verification algorithm, top level

Once the list of transition pairs has been made, state transition requirements must be determined for each pair. The requirements are motion facts that would have to hold for the objects to end up in their final positions, e.g. (`transMotion CupHolder1 Right`). To determine if objects translated, `deduceRequirements` first selects a glyph labeled `FixedObject` that is present in both subsketches. Then, the relative position of each non-fixed object to that fixed object is computed between the two subsketches. In the example in Figure 50 it is determined that the cup holder is displaced to the right between the two subsketches, and the aforementioned fact becomes stored as a state transition requirement for the pair.

`deduceRequirements` uses a cognitive model of mental rotation (Lovett et al., 2007) to determine if an object rotated. This model uses edge segmentation to find the edges of the glyphs and then finds a mapping between the edges in two decompositions, matching up their corresponding edges. This mapping is then used to calculate the

shortest angle of rotation between them. While this effectively creates an assumption that objects always rotate through the shortest distance, in practice this has not yet caused problems with any of the design explanations encountered.

Returning to the example at the start of this subsection, the two requirements are stored as facts in working memory:

```
(stateTransitionRequires State1 State2
      (translationalMotion CupHolder Right))
(stateTransitionRequires TopView TopView2
      (rotationalMotion CupHolder CW))
```

The next step, `verifyRequirements`, verifies that each requirement is met by using QM to see if the predicted motion occurs. In this example, the system performs the following queries to check the directions:

```
(translationalMotion CupHolder ?dir)
(rotationalMotion CupHolder ?rdir)
```

These are performed in the context of the first member of the pair, `State1` and `TopView`, respectively. The translational motion query will return `ZeroQVector`, because again, the surface contact with the fixed base of the device prevents it from moving. The rotational motion query will return `ZeroRot`, because there is no rotational force acting on cup holder. (If you check, you'll see there's no force arrow.)

When the result of the verification step does not match the requirement, it is added to a new feedback item, which will be turned into the English statement like the

one seen previously in Figure 43 (page 97): “To go from State 1 to State 2, cupholder has to move to the right in the sketch, but it will not move!” Additionally it would give feedback about the top view pair (pictured in Figure 50) too: “To go from Top View to Top View 2, cupholder has to rotate clockwise in the sketch, but it will not!” To address this feedback, the student could change the cup holder to be removed vertically in both of the side view subsketches as done before, and add a rotational force to the cup holder. After these changes, the results match the requirements, and no feedback is generated by STV.

5.4 Sequential Explanation Analysis

Sequential explanation analysis (SEA) takes in a list of `studentSays` facts derived from the structured language input (5.2.2) and proceeds through them sequentially, checking to see if they can be proven true, can be proven contradictory, or are currently unprovable. This checking is done by a set of verification routines, introduced in Figure 52.

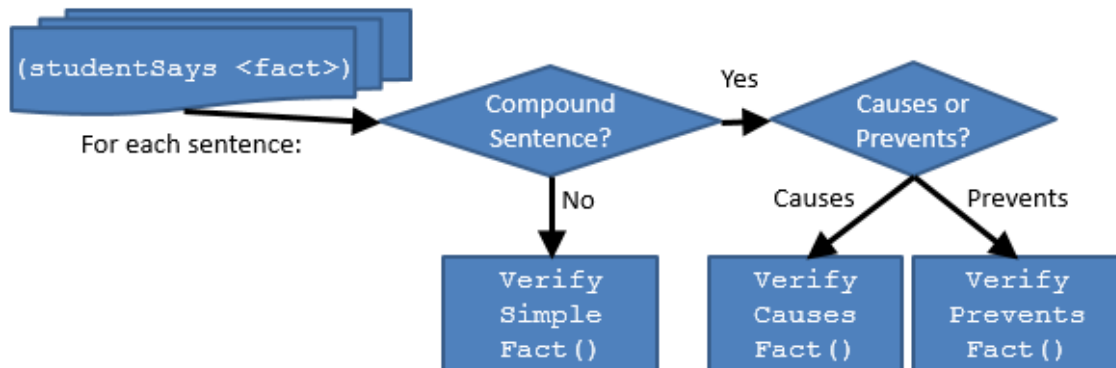


Figure 52: Sequential Explanation Analysis, Top Level: The facts representing the sentences are sequentially passed to the appropriate verification routine.

Non-compound sentences are checked by the routine `VerifySimpleFact`.

Compound sentences are sent to `VerifyCausesFact` or

`VerifiesPreventsFact` depending on their conjunct. Given a fact, `F`, any of the verification routines in this chapter will return one of three values:

1. `(proven F)`
2. `(contradiction F C)`, where `C` is a fact representing the reason that `F` is contradicted
3. `(unproven F)`

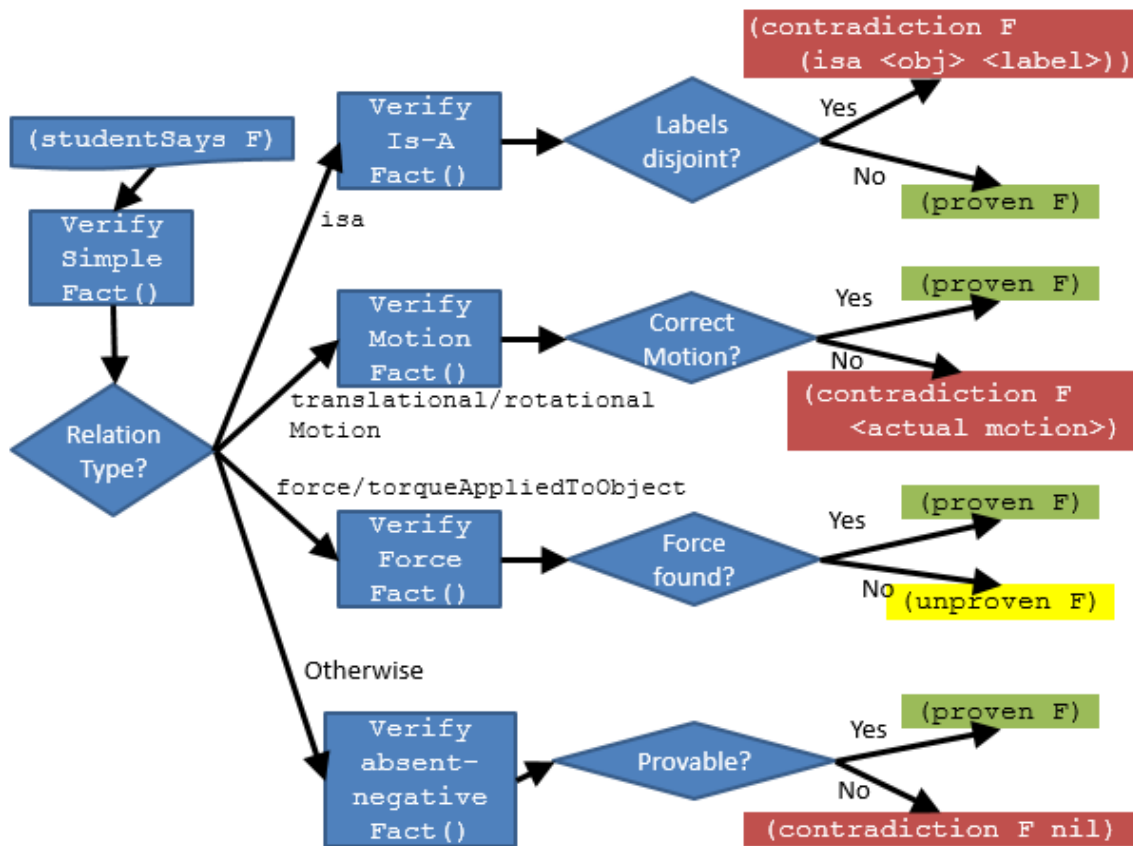


Figure 53: Sequential Explanation Analysis, verifySimpleFact:
Simple facts are proven according to the relation they represent.

verifySimpleFact passes F to one of four subroutines depending on the relation F represents (Figure 53). For example, when our student in the cup holder example told Design Coach that the cup holder can rotate in the side view, the structured language input produced a fact like:

```
(studentSays (in-context State1 (rotatable CupHolder)))
```

Since this fact is not one of the three special cases, it is handled by the default routine, verifyAbsentNegativeFact, which assumes that a fact is negated if it cannot be

proven. Since QM cannot prove the Cup Holder is Rotatable, the routine returns `(contradiction rotatable(CupHolder) nil)`. A more ideal result would be to fill this nil in with the missing facts, but the reasoning engine in CogSketch does not store the justifications for why previous queries failed. Systems which do keep track of this information could have the default verification routine fill in the reason field of the contradiction fact at this point for later processing. In Design Coach, we use the structured explanation system to provide the student with the appropriate details (see Section 5.6, Feedback Generation).

The third time the student requested feedback in our example, the context of the sentence was changed to the top view, where the cup holder was no longer blocked from rotating. In that case, QM concludes that the cup holder can rotate, and the routine simply returns `(proven rotatable(CupHolder))`.

Aside from the default, there are three fact-specific routines for Design Coach:

1. `verifyIsAFact`
2. `verifyMotionFact`
3. `verifyForceFact`

The first, `verifyIsAFact`, looks at facts of the form `(isa <entity> <label>)`. These facts mean that an entity belongs to the Cyc collection (a category of things) specified by label. Entities may have multiple labels but the KB also includes facts about which collections are disjoint, so this routine checks to see if the label in the sentence fact is disjoint with any other labels believed about the object in working

memory. If a disjoint label is found, a contradiction fact is produced and that disjoint label included as the reason. If the labels are not disjoint, the result becomes a proven fact regardless of whether or not it was believed before.

The second routine, `verifyMotionFact`, uses QM to check and see if the translation/rotation in the sentence is indeed the direction the object will travel. Because motion queries are designed to always return one answer (including zero or ambiguous), the result will either match the direction given in the sentence or not. If it does, it is returned proven, and if it does not, a contradiction is returned with the contrary motion fact QM. For example, if the student opened her sketch with the cup holder moving up in state one and added a sentence that said “In state 1, the cup holder moves down.” SEA would return a fact like:

```
(contradiction
  (inContext State1 (translationalMotion CupHolder Down))
  (inContext State1 (translationalMotion CupHolder Up)))
```

The third routine, `verifyForceFact`, also uses QM to check the fact. However, since there may be any number of forces applied to an object, Design Coach can accept the student’s statement as an unproven assumption rather than a contradiction, so here the options are proven or unproven. The unproven fact will prompt feedback asking for more detail, but the system will also assume the fact is true until the student changes or removes the sentence. For example, if the student adds a

sentence “In Top View, the cup holder is being forced clockwise.” the system comes to believe a fact like:

```
(inContext TopView (rotationalMotion CupHolder CW))
```

But it also stores a fact like:

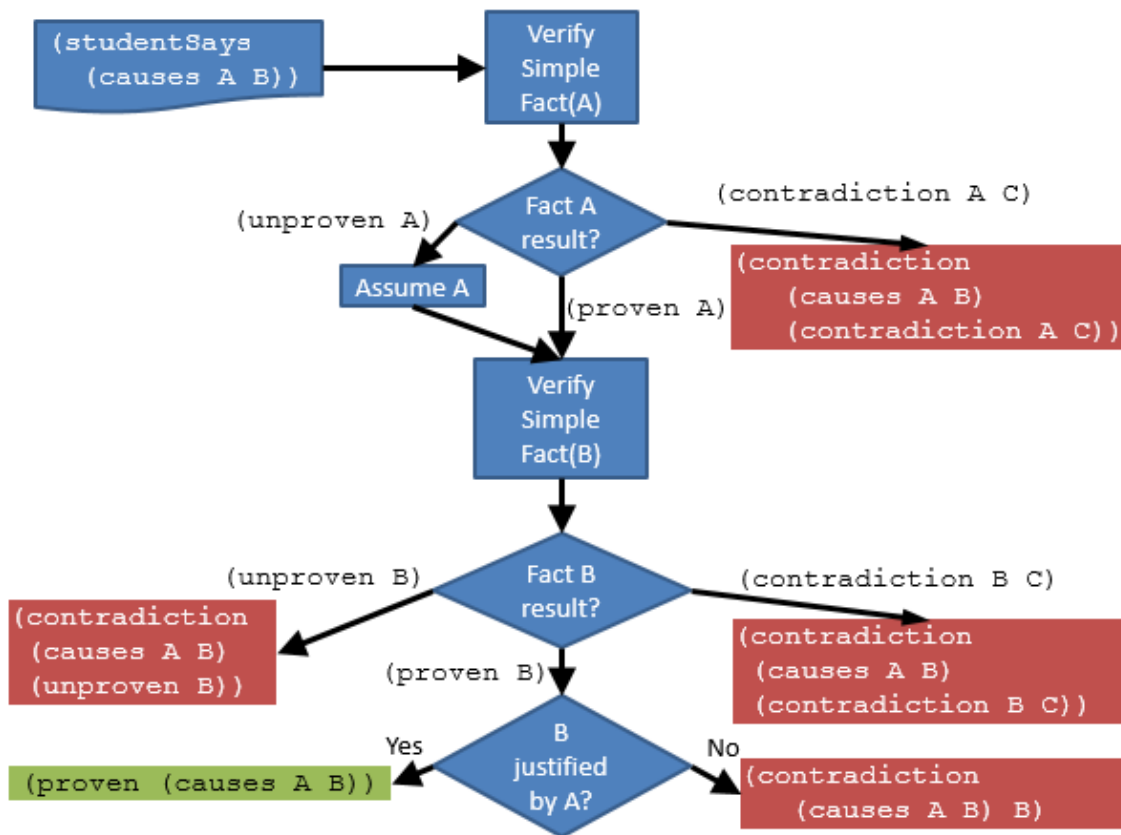
```
(unproven
  (inContext TopView (rotationalMotion CupHolder CW)))
```

The presence of this unproven fact produces a new feedback item: “Why is it the case that cupholder is being forced clockwise in Top View?”

Now that we have covered all of the cases in verifying simple sentences we turn to compound ones. As noted before, compound sentences can be thought of as two simple sentences with a conjunct in between them. They are represented as facts like:

```
(studentSays (<conjunct> factA factB))
```

There are two verification routines for compound sentences, one for when the conjunct is causes (Figure 54) and the other for when the conjunct is prevents (Figure 55). Both procedures make use of `verifySimpleFact` to verify their individual clauses. Both also verify their A (antecedent) clause first, since if this clause is false then the statement is moot—an instant contradiction. If A is unproven, both assume A to be true and proceed with their verification of their B (consequent) clause. From that point on, the behavior of the two diverge.



**Figure 54: Sequential Explanation Analysis, verifyCausesFact:
A causes B if both are proven and A justifies B.**

For `verifyCausesFact` to prove that A causes B, it must show that B is true and that B is true *because* A is true. If B is unproven or contradicted, then “A causes B” is contradicted right away by that fact. In this case the contradiction/unproven fact itself becomes the reason for the contradiction. Returning to the cup holder example (Figure 47, pg. 101), consider the case where the student adds the sentence “In Top View, cup holder is being forced clockwise which causes cup holder rotates counterclockwise.” Because the antecedent is a force statement, and there are no force

arrows in that subsketch, the antecedent is found unproven, and thus assumed true. When SEA verifies the consequent, the clockwise torque applies to the cup holder. However, since the student said that causes the cup holder to rotate counterclockwise, the result is a contradiction. Therefore `verifyCausesFact` returns a fact like:

```
(contradiction (causes
                (torqueAppliedToObject CupHolder CW
                ?source)
                (rotationalMotion CupHolder CCW))
 (contradiction (rotationalMotion CupHolder CCW)
                (rotationalMotion CupHolder CW)))7
```

However, if the antecedent had been another unproven fact, such as saying the rotational force should cause an upward force, then the contradiction returned would have the unproven fact as its reason instead.

```
(contradiction (causes
                (torqueAppliedToObject CupHolder CW
                ?source)
                (forceAppliedToObject CupHolder Up
                ?source))
 (unproven (forceAppliedToObject CupHolder Up)))7
```

⁷ Context wrappers removed for clarity.

Finally, if the consequent is true then there are still two more cases; either A caused B or A didn't cause B. SEA uses the reasoning engine's facilities to check if A is a logical antecedent of B. If it is, then it is proven that A causes B. If not, then the result is a contradiction. There could be any number of reasons why B was true without A, so B itself is added to the reason slot for the contradiction, leaving the calling system to perform its own analysis of B's antecedents to further explain the contradiction.

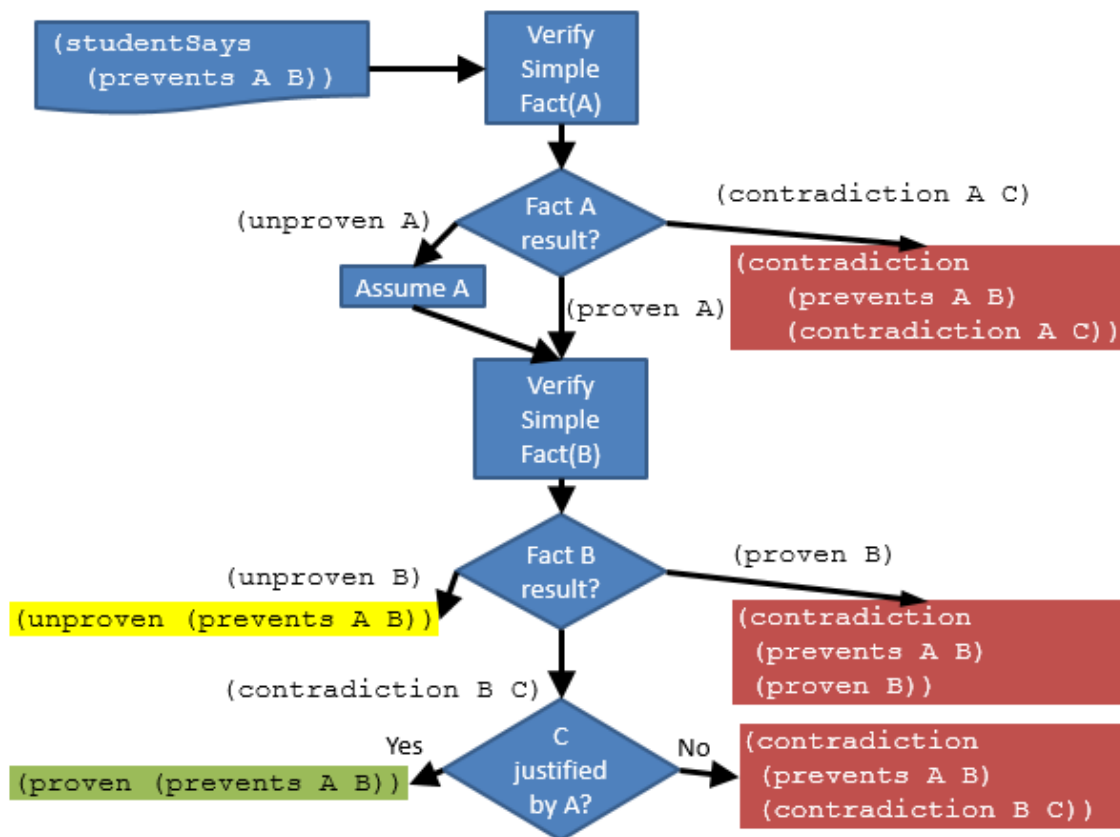


Figure 55: Sequential Explanation Analysis, verifyPreventsFact: A prevents B if A is proven, B is contradicted by C, and C is justified by A.

`verifyPreventsFact` is somewhat analogous to `verifyCausesFact`: A will be shown to prevent B if 1) A is proven, 2) B is contradicted, and 3) the contradiction of B is justified by A. It proceeds identically to the causes version of the routine up until the point after B has been verified; then there are two main differences. First, the consequences for B being a contradiction and B being proven are switched. If B is proven, that contradicts A preventing B. On the other hand, if B is contradicted, then to prove A prevents B, A should play a role in causing C, the immediate reason given for B's prevention. (Naturally, if $A = C$, A prevented B as well.) If A is not equal to or an antecedent of C, then `(prevents A B)` is contradicted by the fact that C contradicts B *instead of A* or some consequent of A. Again, `(contradiction A C)` is placed in the reason slot of the contradiction so the calling system may sort it out.

The second change from the causes version is that if B is unproven, then the sentence is unproven rather than contradicted. This is because if A is true but it is unclear why B is not proven, that signals a lack of knowledge, whereas in the causes case it is clear that A didn't cause B, or B would be proven. Thus the result is unproven instead of a contradiction.

5.5 Teleological Ontology

Design explanations involve more than statements about mechanical behavior, so QM knowledge alone could only take Design Coach so far. With a knowledge of function,

i.e. *teleology*, the system can link the mechanical properties and behaviors it understands to their purposes. The teleological ontology is an organized representation of design function. The ideal is a teleology that will be domain-independent, useable with future systems and any future Design Coach projects (e.g. electronic systems, physical systems outside the current scope of Design Coach QM, etc.). This section begins with a brief description of the ontology on which the teleology for Design Coach is based, then describes the representations used in Design Coach and finally concludes with details of the resulting options available to the user through structured language input from this teleology, and how those options were selected.

5.5.1 Way-of-Function Ontology

After considering several alternatives (see Related Work, Section 7.2), the model of ontology of device function developed by Kitamura et al. (2006) was selected as the basis for the teleological representations in Design Coach. Kitamura's work describes artifacts using function decomposition trees, in which the goal function of a system is recursively decomposed into a tree of method functions which, when performed, achieve their parent goal. In these trees, parents are related to their child functions via a type of relationship called a *way-of-function-achievement*, or a *way* for short. In their ontology, each way has a list of generic functions which accomplish its goal. For example, the Heat Water function can be achieved by a Heat Transfer way which specifies two generic child functions: Transform Electricity to Heat and Give heat to Water. The core concepts of function and way can be easily expressed and understood

as English statements, and recursive nature of this ontology lends it to future expansion, making it a good fit for Design Coach.

5.5.2 Teleological Representations of Design Coach

The core of this adaptation of way-of-function ontology are the following three representations. The first is the concept of teleological function itself.

Definition 64 (Teleological Function): *A TeleologicalFunction is a symbol denoting a purpose a design may (seek to) accomplish.*

One example of a teleological function was in the example at the start of chapter: the cup holder being detachable. The next concept is ways to achieve said functions.

Definition 65 (Way of Achieving Function): *A WayOfAchievingFunction is a symbol denoting a way to achieve some TeleologicalFunction.*

An example of the ways of achieving the aforementioned detachment are through de-interlocking the parts or disconnecting one or both of them from some temporary adhesive.

Definition 66 (Function Achieved Via):

functionAchievedVia (tf, w, parts, contexts) is true when the members of the list of entities, parts, achieve TeleologicalFunction tf through WayOfAchievingFunction w over the course of progression through the list of states, contexts.

The student's sentence was "In state 1 cup holder detaches from attachment panel via interlocking/de-interlocking." This translates into a fact like:

```
(studentsays
  (functionAchievedVia Detachment-TF InterlockingPartsWay
    (TheList CupHolder Panel) (TheList state1)))
```

The full vocabulary of teleological functions and ways follows.

5.5.3 Functions and Ways

As part of our classroom interventions (Section 6.2), students were asked to submit feedback on Design Coach, including vocabulary they would like to be able to use in the Tell window. Additionally, the students' work submitted as part of the interventions included adding novel refinements to the design, which introduced new functions, or alternative means of achieving the original design goal. After collecting feedback from students and instructors, we assembled the following collection of functions and ways (Table 3) into the teleological ontology and added it to Design Coach.

Table 3: Ways of Achieving Teleological Functions

Teleological Function	Ways of Achieving Function
Attachment	Interlocking Parts Adhesive Material Temporary Adhesive Material
Detachment	Interlocking Parts Temporary Adhesive Material
Adapting in Size	Adjustable Parts
Containment	Enclosure Prohibiting Downward Motion
Increased Comfort	Change in Shape Moving within Reach

The teleological functions were added to the structured language input as verbs (Table 2, pg. 107). For four of the five teleological functions, the language like input looks like the kind of sentence discussed at the end of the previous section:

In <context> <subject> <Teleological-Function-Verb> <object>
via <Way-of-Achieving-Function>.

The one exception is the function for increasing comfort, where the object is simply assumed to be the user of the device and is thereby omitted. When a teleological sentence is constructed, facts are added of the form seen in the previous example.

(studentsays

```
(functionAchievedVia <TeleologicalFunction>
  <WayOfAchievingFunction>
  <list of parts> <list of states>)))
```

We added rules to allow the system to prove these facts, so the verification routines of SEA can simply query the reasoning engine to prove if they are true. These rules are

listed in Appendix A. `functionAchievedVia` is assumed to be negated if absent, and thus falls under the default verification routine.

5.6 Feedback Generation

When the student presses the feedback button, the three feedback systems are engaged in turn. SEA is run first, then STV, and then finally the sketch tips system.⁸ SEA is run before STV so that STV can benefit from any new QM knowledge that was proven or assumed from the structured language input. At the end, the sketch tips system searches the sketch for user input mistakes such as disconnected annotations and relations (user forgot to choose (a) target(s) for the glyph), subsketches with only one glyph (user forgot to segment their ink), and subsketches connected by a `causes` arrow which lack a common fixed object (necessary for STV). Each piece of feedback is added to a lisp object which includes the English text string for display, the list of glyphs/edges to highlight, and if relevant, the fact that prompted this feedback (e.g. an STV verification or an SEA contradicts/unproven fact). If the source of the feedback was a fact, the English text is rendered by a language generation system using templates stored in the knowledge base and the namestrings given to entities by the user. Lastly, if no feedback items are found, then the message “Ah that explanation makes sense. Thank you.” is added to the feedback list instead.

⁸ Though the sketch tips system is run last, it is actually independent of the other two algorithms. We also move the sketch tips to the top of the list of feedback displayed to the student, so they may fix their input errors first.

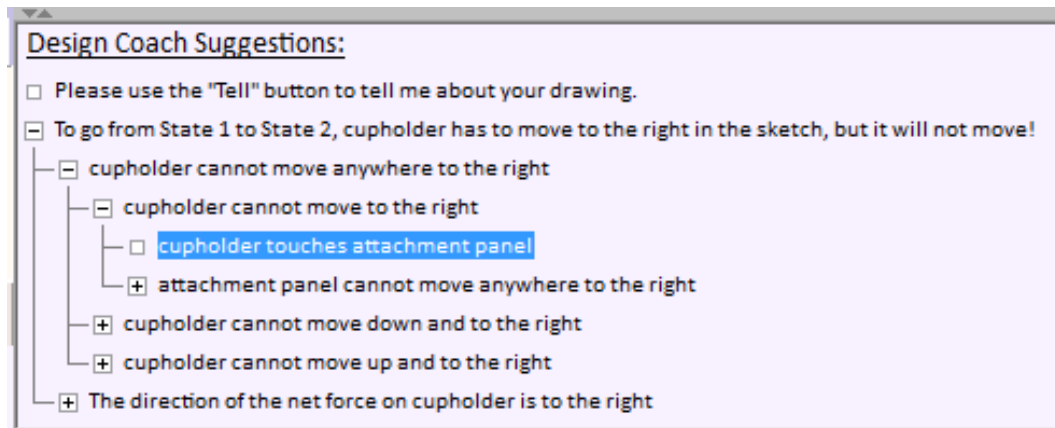


Figure 56: A close-up view of the Design Coach feedback pane from Figure 43

The feedback items are then displayed in the feedback pane below the sketch as shown in Figure 56 (see also Figure 43, Figure 46, and Figure 47 in Section 5.1.1). The expanding tree structure is a structured explanation system similar to one used in CyclePad (Forbus et al., 1999). We developed this to help Design Coach explain its feedback to students, who sometimes could not figure out why Design Coach did not understand their explanations. The structured explanation system is based on two sets of knowledge: a set of axioms that define which facts warrant further explanation, and a set of rules which determine which facts should be used to explain a given fact. The former is used to determine if an item in the outline is expandable or not. The latter is used when a student clicks on the icon to expand the feedback (lazy evaluation).

The default rule for explaining a fact is to state any antecedents of that fact for which the system has an English string template. (In other words, if you can't say it, don't bother.) This works well when facts have justification involving QM facts; for example in Figure 56 when the system is explaining why the fact that the net force on

the cup holder is to the right, some `forceAppliedToObject` fact(s) will be its antecedents, so the student can drill down into the net force fact to see where system thinks the forces are coming from.

Design Coach Suggestions:

- The sketch doesn't show that attachment panel and cupholder achieve detachment via interlocking parts in State 1.
 - For this explanation to make sense, make sure:
 - The two objects must appear in two states, attached via interlocking in one, and detached in the other.
 - The two states must either be connected by a "causes" relation or BOTH be mentioned in the sentence.

Figure 57: Design Coach feedback for a teleological sentence

However, in the case of a contradiction for a `functionAchievedVia` fact, the fact was simply failed to be proved, and it is hard to deduce why. In the case of teleological knowledge, facts have been added to the KB which give explicit instructions on the requirements for the function to be achieved by the selected way. Rules then specify that the teleological facts should expand to show those instructions. For example, if the student removed the `causes` arrow from between State 1 and State 2 in the cup holder example, her sentence explaining the detachment would result in the feedback shown in Figure 57. In other cases, rules are used to filter out redundant information or skip unnecessary levels of detail.

Chapter 6

Evaluation

In this chapter, Design Coach is evaluated on three criterion.

1. The accuracy of Design Coach QM at predicting motion in a range of sketched mechanisms (the mechanism corpus)
2. The accuracy of Design Coach at giving feedback on student's engineering design explanations (the explanation corpus)
3. Impacts of a Design Coach classroom intervention on the education of engineering students, including:
 - a. Effects on their self-reported anxiety and skill with sketching and
 - b. Correlation between success on design coach homework and overall performance in the course.
 - c. Correlation between use of the feedback mechanism and performance of explanatory tasks using Design Coach

The chapter begins with a description of the mechanism corpus evaluation, which shows the breadth of mechanisms that Design Coach QM can understand. It continues with a description of the interventions which created the sources of data for the second and third criteria. The explanation corpus is evaluated next, showing how well the Design Coach did at critiquing engineering design explanations from the intervention

and describing the sources of its limitations. We conclude our evaluation with an analysis of the three measures of impact on students.

6.1 Mechanism Corpus

Table 4: Mechanism Corpus Sources and Features
(sketches and their descriptions can be found Appendix B)

Name	Source	Features	Fig.
Book Holder	DTC	Preventing/Allowing Translation	Figure 73
Detachable Paint Roller	DTC	Preventing/Allowing Translation	Figure 74
Mini Baja Go-Cart	DTC	Rotation and Translation	Figure 75
Double-Sided Switch	DTC	Translation	Figure 76
One-Handed Nail Clipper	DTC	Multiple Springs	Figure 78
One-Handed Egg Cracker	DTC	Rotation and Translation	Figure 79
Abdominal Exerciser	DTC	Springs, Rotation	Figure 80
Wheelchair Stabilizer	DTC	Preventing/Allowing Rotation	Figure 81
Recliner	DTC	Springs, Rotation	Figure 82
Under-Armrest Rotatable Cup-Holder	DTC	Rotation and Translation	Figure 83
Steering Wheel Handle	DTC	Rotation	Figure 84
Orthopedic Trainer	DTC	Springs	Figure 85
Runner (Block and Tackle)	Basic Machines	Cords & Pulley	Figure 86
Gun Tackle	Basic Machines	Cords & Pulleys	Figure 87
Luff Tackle	Basic Machines	Cords & Pulleys	Figure 88
Luff upon Luff	Basic Machines	Cords & Pulleys	Figure 89
Spring Button	Other	Springs	Figure 90
Gear Train	Other	Gears	Figure 91

Table 4 lists the designs in the mechanism corpus. Figures for each design in the corpus can be found in Appendix B. This first twelve designs come from the Design Thinking and Communication Course at Northwestern. Nine of these come from the original group of mechanisms surveyed as we started the project (Wetzel & Forbus, 2009b), and three come from projects selected by instructors for use in our classroom interventions (see the next section). The final pair were created for demonstration purposes, and as of this writing, are included with the CogSketch executable. Together, the designs in this corpus cover a variety of mechanisms, including all aspects of QM: transference of constraint and of force, translational and rotational motion, springs, gears, cords, and pulleys. For each design a sketch was created and the translation and rotation of each rigid object in the sketch was queried with Design Coach QM.

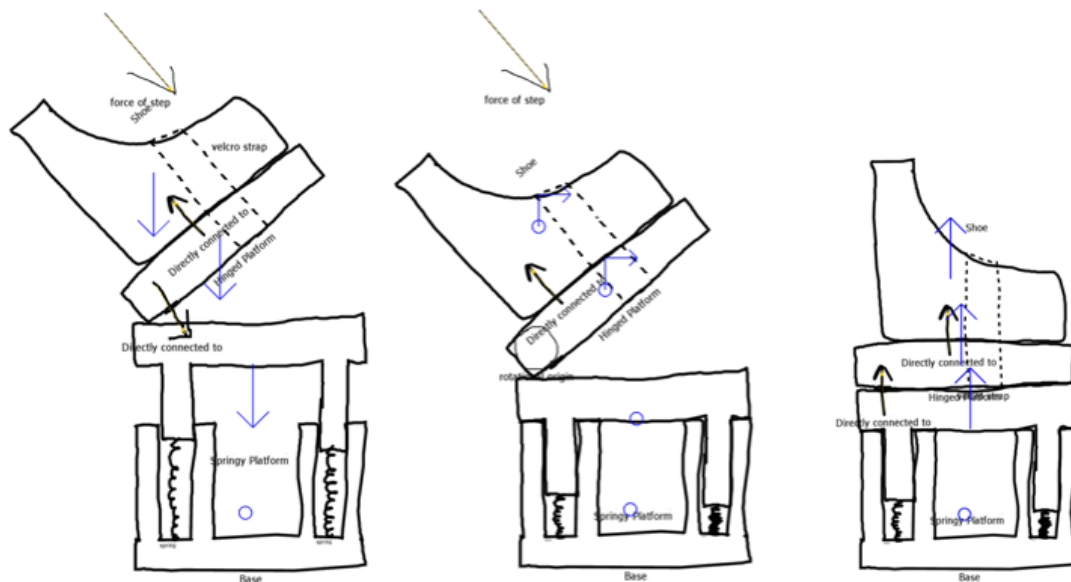


Figure 58: The Dynamic Dropper Design, an Orthopedic Device

For example, Figure 58 shows the Dynamic Dropper Design from DTC, which helps patients with spinal cord injuries avert rapid decline in bone mineral density by allowing them to experience muscle activity. In response to gravity acting on the foot, the mechanism descends. When it can no longer fall straight down, the hinged platform rotates clockwise, and with it the shoe. Finally the springs exert a force upward to send the platform back up.

In summary, Design Coach is able to predict the motion of a variety of mechanisms within a 2D plane. Mechanisms which move in three dimensions will be discussed in future work.

6.2 Design Coach In-Class Interventions

Over the course of a five-quarter series of pilot pull-out studies, the first version of Design Coach, *Design Buddy*, was developed (Wetzel & Forbus, 2008, 2009a, 2009b, 2010) for Northwestern University's Design Thinking and Communication course (formerly known as Engineering Design and Communication). The first in-class interventions took place in the fall 2011 and winter 2012. A small pullout study was performed in the spring of 2012, followed by three more interventions in fall 2012, winter 2012, and fall 2013. All interventions took the form of a mandatory homework assignment, which was developed in collaboration with instructors of the course.

Before each quarter, instructors were recruited to try the intervention in their section. In DTC, instructors are paired up and assigned to one or more sections of about

16 students, so the pool of students which received the homework assignment was determined by which instructors volunteered. The average number was about two sections per quarter, or 32 students. Each quarter the homework assignment was given at a time of the instructors' choosing, and students were given one week to complete the assignment (extensions were allowed upon request). For the final three quarters, students were also given an opportunity to take an online survey on their anxiety and skill with sketching before and after the assignment. The details and results from the survey are given in Section 6.4.1.

The homework assignment consisted of three parts:

1. Complete the in-program Design Coach Tutorials.
2. Explain a given design from a previous quarter to Design Coach using a sketch and sentences entered through the Tell window.
3. Think of one refinement for the design in part 2, and explain the refined version to Design Coach.

The homework assignment also asked the students to write an explanation in English for their instructor in the box at the top of the Tell window, and to share their own feedback about Design Buddy/Coach in the notes window. The first design chosen was a rotating cup holder (Figure 83), primarily because it involved motions in two separate planes without requiring 3D. Additional designs were tried, including the Gamma Handle (Figure 84) and Dynamic Dropper (Figure 85) in the Spring 2012 pilot, and the Gamma Handle in section 5 of the Fall 2012 intervention, but in the end it was decided

to stick to one design (the cup holder) to lower the number of variables. For more details, copies of a homework assignment handouts have been included in Appendix C.

In the first quarter (Fall 2011) there was an additional part between steps two and three, where students were told to give Design Coach an explanation of how the design was used in a new context of the students choosing. It was removed after the first quarter to shorten the overall length of the assignment after feedback from students convinced us it was too long. As a result of the shortening of the assignment, in fall 2011, students were asked to create three files (one for each sketched explanation), while the remaining quarters they were asked for two. These submitted files form the explanation corpus.

6.3 Explanation Corpus

Table 5: Explanation Corpus Sources

Quarter	Section	# of Sketched Explanations	# of Students Submitting
Fall 2011	6	60	22
Winter 2012	10	26	14
Winter 2012	16	24	12
Spring 2012	N/A	4	2
Fall 2012	4	27	14
Fall 2012	5	31	16
Winter 2013	8	21	11
Fall 2013	3	27	13
Fall 2013	12	20	11
Totals	N/A	240	115

The explanation corpus contains 240 sketched explanations (sketch files),

submitted by 115 students. A breakdown of the source by quarter and section number is given in Table 5. To evaluate the explanation corpus, each sketched explanation was opened and feedback was requested from Design Coach. When older sketches had their sentences removed due to changes in the vocabulary of our structured language input, these we re-entered where applicable using the newest vocabulary (presented in 5.2.2) before testing. The ink of the sketches was not modified. Each feedback item was then manually inspected and valid feedback was added to the gold standard. If a piece of valid feedback should have been present but was missing or replaced with an invalid feedback item, the missing valid item was also added to the gold standard. Each sketched explanation was then checked again against the gold standard to create a list of missing and invalid feedback. Instances of errant feedback were then tallied by inspecting the lists and removing duplicates generated by errors with double-entries. For example, if the correct feedback was “Cup holder moves right in the sketch, but it will actually move up!” and the system returned “Cup holder moves right in the sketch, but it will actually move down!”, the former is added to the missing list while the latter is added to the invalid list—two entries for one error. If the system generated no feedback at all for this fact, then the former would be added to the missing list and no addition would be made to the invalid list. Thus when tallying the feedback we ignored errors that are duplicates from the other column. This was done separately for the STV and for the SEA feedback. All errant feedback from the sketch tips system was fixed during debugging, leaving no errors to present here.

35 out of the 240 sketches in the corpus have at least one instance of missing or invalid feedback. Table 6 shows the error rates for both feedback generation algorithms separately. The number of valid feedback responses includes counting cases where the correct response was to give no feedback (i.e. SEA in response to a correct sentence, STV when objects move as depicted between state transitions).

Table 6: Error Rates for Feedback Algorithms

Feedback Source	Number of Valid Feedback Responses	Number of Invalid Feedback Responses	Error %
STV	197	28	12%
SEA	627	39	5.9%
Total	824	67	8.1%

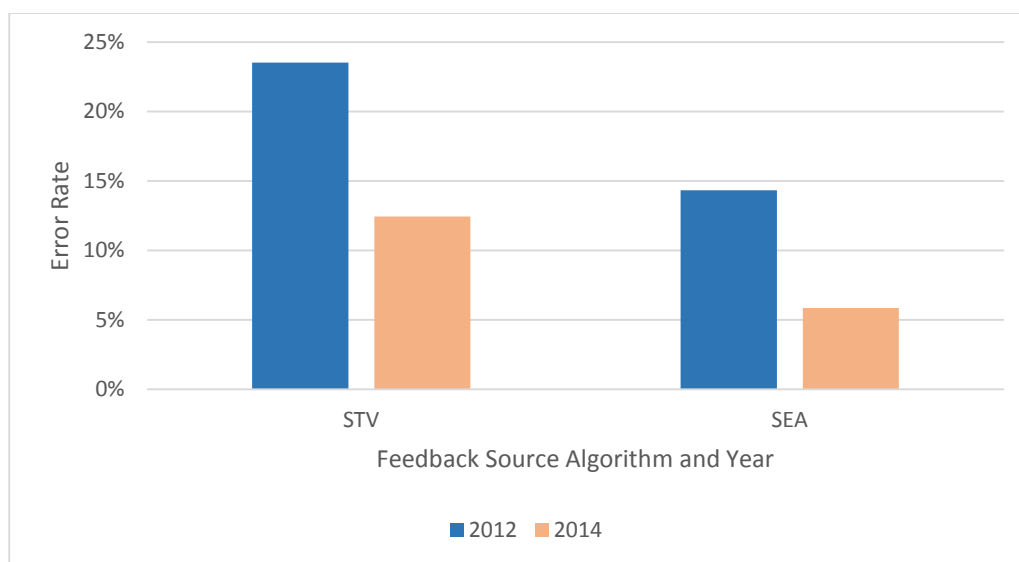


Figure 59: Overall feedback error rates for STV and SEA published in Wetzel & Forbus (2012) (STV = 23.5%, SEA=14.3%) vs running on the explanation corpus in 2014 (STV = 12.4%, SEA = 5.85%)

The error rate for both algorithms has decreased from when the teleological ontology and structured explanation systems were first added (Wetzel & Forbus, 2012).

The corpus for the original experiment was the fall 2011 and winter 2012 data.

Table 7: Precision and Recall of Feedback Algorithms

Feedback Source	Number of Missing Feedback Items	Number of Extra Feedback Items	Precision	Recall
STV	26	2	99.0%	88.3%
SEA	3	36	94.6%	99.5%
Total	29	38	95.6%	96.6%

The invalid feedback consists of extra feedback (false negatives) and missing feedback (false positives). These are shown in Table 7. As explained in the previous section, the sketches in this corpus are from three designs used with a classroom intervention with Design Coach. The breakdown of sketch types and error rates is listed in Table 8.

Table 8: Breakdown of Explanation Corpus by Design

Design	Number of Sections Which Used Design	Number of Sketch Files Received	Number of Files with Incorrect Feedback	% with Incorrect Feedback
Cup Holder	5	205	38	18.5%
Gamma Handle	1 (+ pullouts)	33	2	6%
Dynamic Dropper	0 (pullouts only)	2	1	50%
Total	N/A	240	41	17.1%

Table 9: Sources of Error for Explanation Corpus Sketches

Feedback Error Source	Number of Sketches
Surface Contact	28
Sentence Critiquing	6
touchesDirectly Detection	5
Translation Detection	3
Arrow Recognition	3
Rotation Detection	2
Other	1

The sources of the feedback errors are shown in Table 9. The following subsections will describe the errors found and their causes.

6.3.1 Surface Contact

Surface contact errors have the highest incidence rate, affecting 22 of the 35 sketch files collected with incorrect feedback, see Table 9. We find that these errors are caused by one of three circumstances:

1. Objects drawn overlapping in 3D or drawn in perspective
2. A low threshold for the minimum distance needed for contact
3. False positives due to students drawing messy ink strokes or stick figures

The number of sketches with each type of error is given in Table 10.

Table 10: Number of Sketches with Surface Contact Errors by Type
 (Note: some sketches contained more than one type of error)

Surface Contact Error Type	Number of Sketches
False Positive from 3D	17
Messy Drawing/Stick Figure	9
Bad Threshold Values	7

The most frequent complication involved in surface contact errors were from objects overlapping in three dimensions. For example, in Figure 60, the algorithm detects a contact between the cup holder and the arm (see also Figure 64). This problem is discussed in Section 8.2.1.

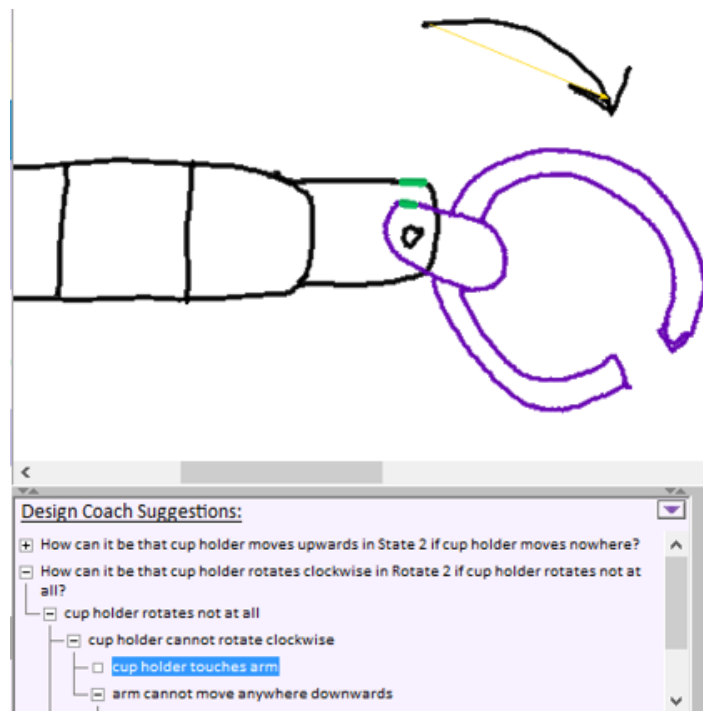


Figure 60: A surface contact causes the system to conclude that the rotation is blocked.

Also visible in Figure 60 is the second source of errors, low thresholds. The gap between the two edges is so large that even if the surfaces were in the same plane they should not be blocked. Another example of this is given in Figure 61.

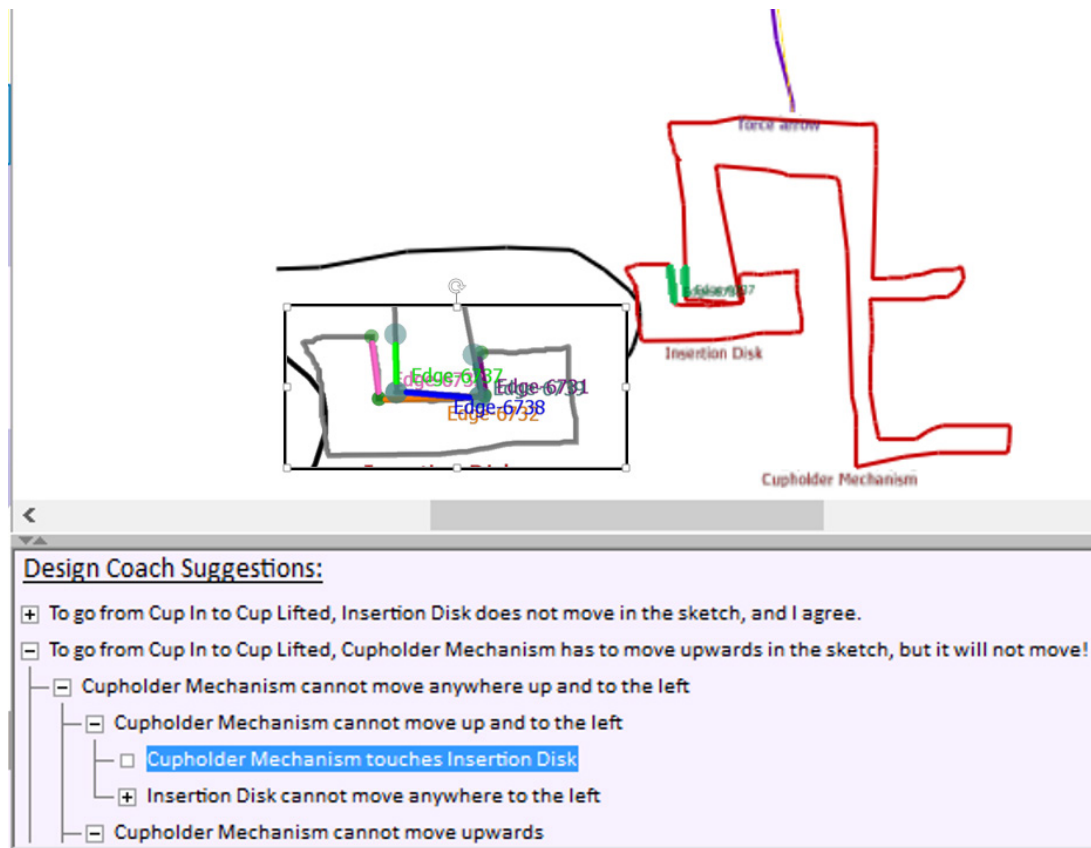


Figure 61: A surface contact causes the system to conclude that the motion is blocked.

In this case, the system finds that the junctions at the end points of the highlighted edges intersect each other, so the cup holder cannot move into the open half plane of left.

However, the normal of the right most edge of the cup holder is over the threshold for pointing Quad1, preventing the cup holder from moving straight up as well. Therefore

Design Coach perceives it to be stuck. Solutions to the issues involving thresholds are discussed in Section 8.2.3.



Figure 62: An example of a messy drawing.

The third type of problem was feedback resulting from surface contacts found (or not found) as a result of messy ink, such as in Figure 62 and Figure 63. While these errors raise important questions about sketch understanding, including how to find the abstract representation of an object with a messy texture, they are considered outside the scope of this work. We do employ a limited scribble fill detection to eliminate excess edges from the initial decompositions created at the start of the surface contact detection algorithm, but its effectiveness is limited (see Figure 63).

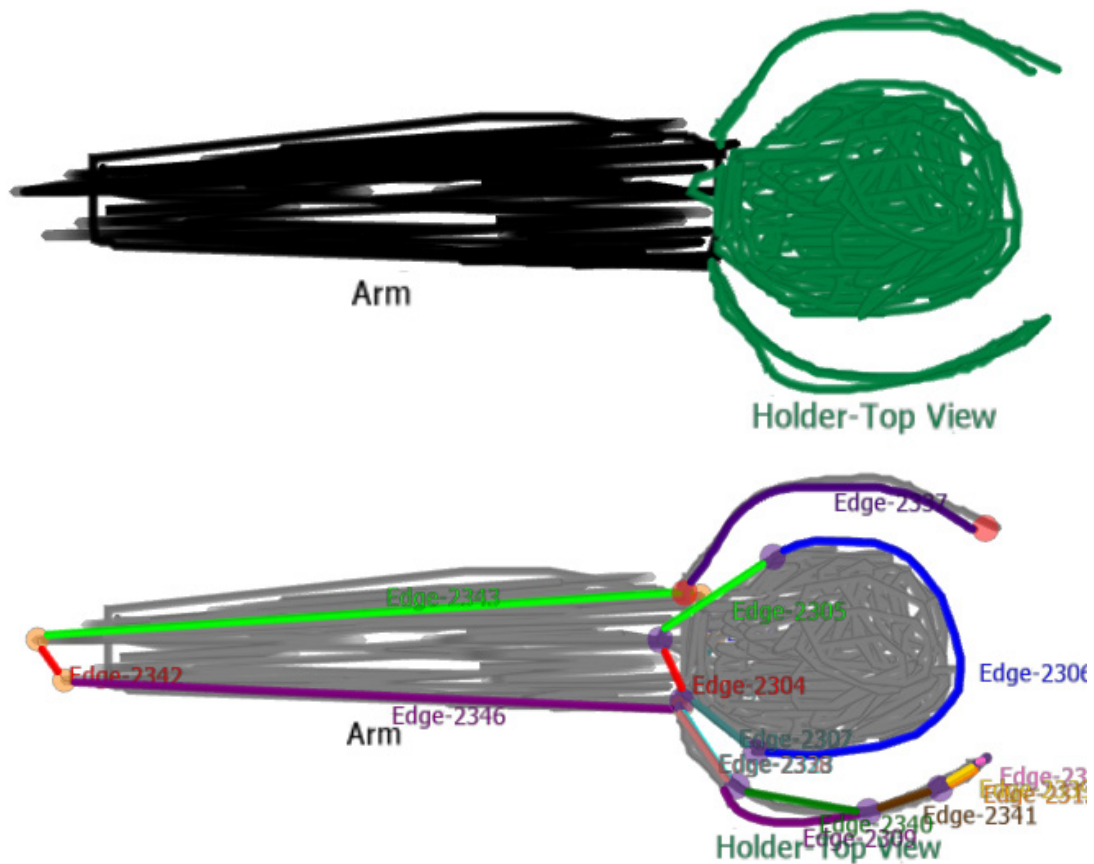


Figure 63: An example of our scribble fill detection

6.3.2 Sentence Critiquing

While the structured input language is designed to only allow the student to input valid statements, in some cases the student's statements carried more meaning than the system gave them. Examples of this included moves-causes-moves sentences, discussed in 8.2.5. Other examples included:

- “In State 2 Velcro strips 1 touches Velcro Strips 2 which causes Cup holder arm is being forced up”

This statement was interesting because in the real world, force could get transferred through such a connection. However, we do not assume Velcro implies a `connectedTo-Directly` relationship, because sometimes the students may want to apply a force to separate the Velcro, and `connectedTo-Directly` our definition implies that the objects will remain together in the specified state.

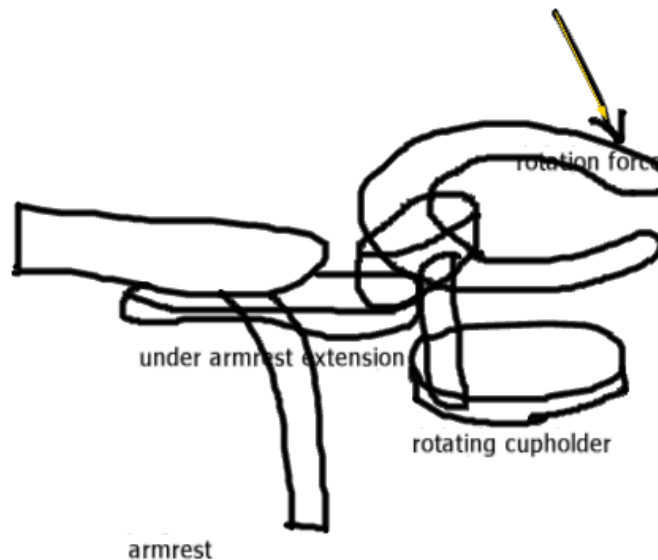


Figure 64: A cup holder drawn in perspective

- “You assert that rotating cupholder is being forced downwards which causes rotating cupholder rotates clockwise in State 1 but it’s false that rotating cupholder rotates clockwise.”

This statement accompanied the drawing in Figure 64 sketched from an oblique perspective; the student probably wanted to say “forced clockwise” instead, but that was not available vocabulary at the time—this example was from the very first quarter. A surface contact found with the adjacent edge of the under armrest extension prevents Design Coach from thinking it will rotate regardless. An alternative approach to giving feedback on perspective is described in Section 8.2.1.

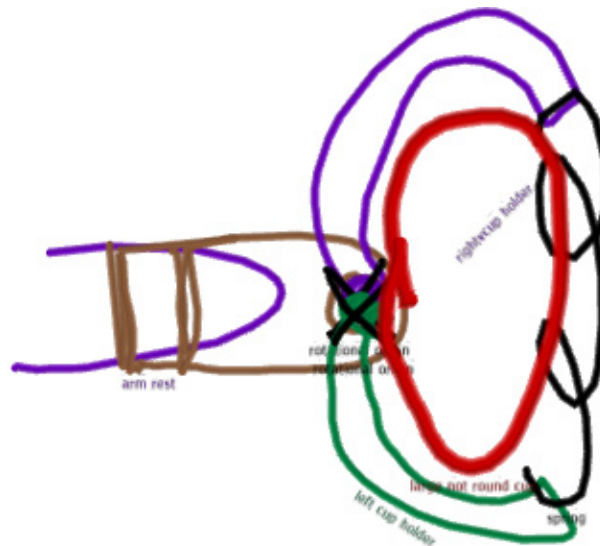


Figure 65: A cup holder which clamps the cup using a spring.

- “You assert that spring is stretched thing which causes large not round cup moves nowhere in State 2 but it’s false that large not round cup moves nowhere.”

This last statement came from the design pictured in Figure 65. The sentence is logical, because the spring will hold the clamps shut around the cup, preventing it from moving. However there are two reasons Design Coach could not understand this; first, in Design

Coach QM motion is inferred from forces, and to infer zero motion, there must be zero force. The idea of a non-zero force causing a zero motion is outside of QM's model, because that entails the force acting over time, either statically as in this case or against momentum. The second and related reason this could not be understood is that Design Coach does not model quantities, including quantities of force. Drilling down using the structured language interface reveals the explanation "large not round cup moves in an ambiguous direction". The opposing forces of the clamps have summed to an ambiguous force. To address this limitation, adding quantities to Design Coach is discussed in Section 8.2.2.

6.3.3 Detection of Other Spatial Relationships

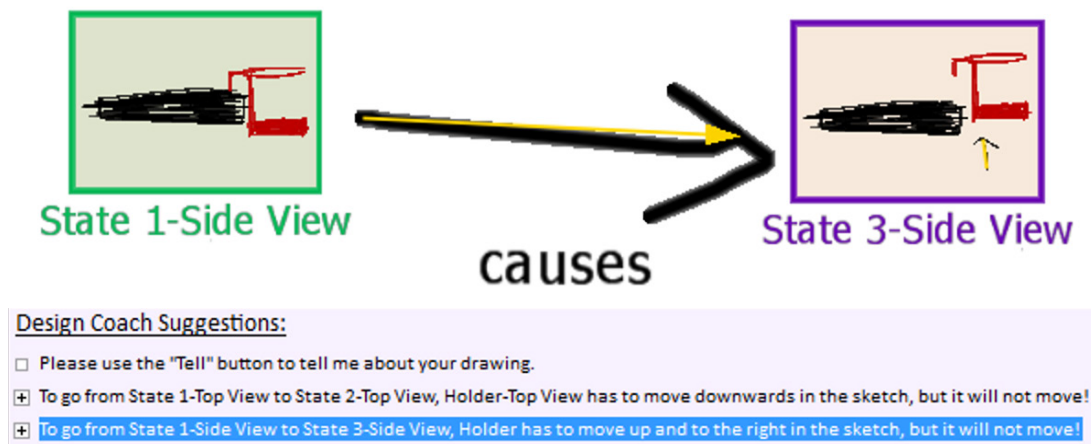


Figure 66: A feedback error in STV (highlighted in the suggestion list)

The systems of detecting translation and rotation between subsketches (STV) both caused a few incorrect pieces of feedback. In STV we assume that students copy and paste their glyphs as instructed. For rotation detection, the weakness of our mental

model of rotation is when the user edits the ink of one of the glyphs, getting a perfect mapping between the edges becomes impossible, and the ability to calculate the rotation of the shape deteriorates. For translation detection, examples like that in Figure 66 raise the question of what is the appropriate threshold; in this case the cup holder moved just enough to the right for the system to decide that it moved Quad1, but to the human eye it could easily be accepted as upwards motion. (The fact that actually it will not move at all comes from the fact that there is no force arrow in state 1.) Similarly, a few errors came from false positives in deriving the `touchesDirectly` relationship. In the example pictured in Figure 67, CogSketch finds the cup holder and beam glyphs to be RCC8 edge connected, and our rules use that to infer that they touch.

Thresholding is discussed further in Section 8.2.3.

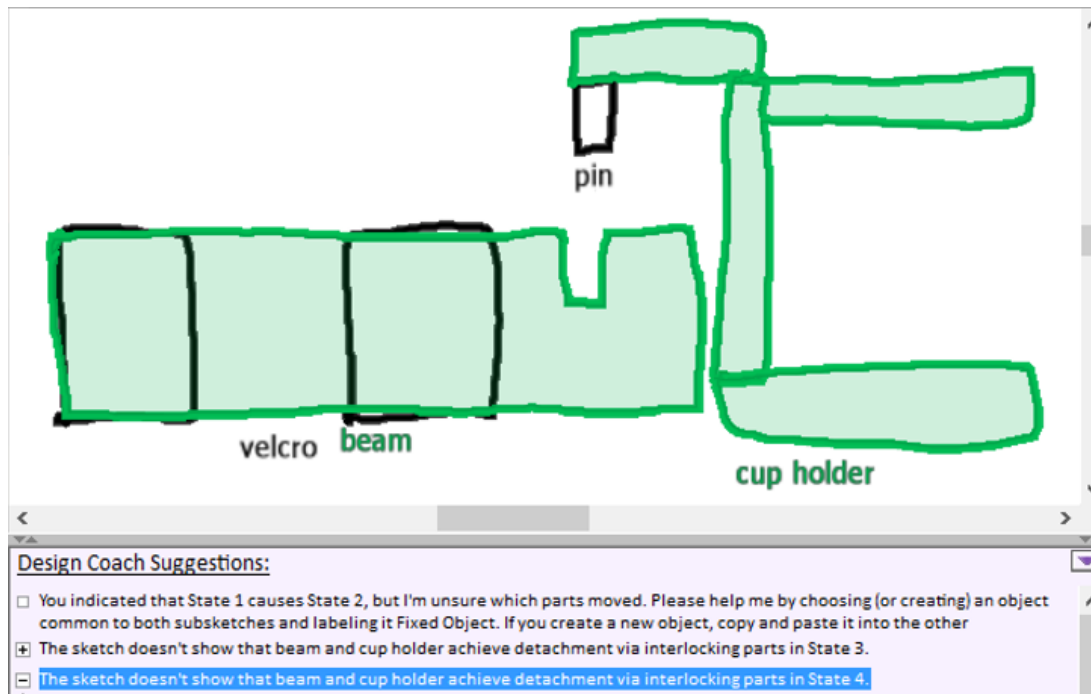


Figure 67: A feedback error caused by an incorrect topological relationship.

6.3.4 Other Error Sources

The final sources of errors involved arrows; in three sketches the head and/or tail of the arrow were misinterpreted resulting in forces being applied in the wrong direction.

CogSketch helpfully overlays the arrow with another arrow indicating its interpretation to prevent such mistakes, but the students did not always correct their arrows. In the final undiscussed case (listed as Other in Table 9), the student somehow managed to create a `connectedTo-Directly` relation arrow that also asserted a force. Because these could not be corrected without modifying the students' ink (or doing some

hacking of the underlying representations) these sketches remain in the corpus as they were found.

6.4 Impact on Students

One educational goal of the Design Coach project was to reduce students' anxiety with sketching. This section details the results of the anxiety measure of the intervention on students.

6.4.1 Sketching Anxiety

When we met with DTC instructors at the start of this work, they convinced us that a major problem for their students was anxiety about sketching. We hypothesized that it might be analogous to math anxiety, a well-known phenomenon which Beilock et al. (2010) have shown has harmful impact on math skills in different situations. To attempt to measure sketching anxiety, I collaborated with a graduate student from Beilock's lab at the University of Chicago to design a measure for sketching anxiety, based on their lab's previous work.

The measure design produced was a survey of questions about anxiety using a 5-point Likert scale. DTC instructors gave us feedback on the questions and final approval before giving the survey to their students. Our original survey included some generic questions about sketching (which were eventually cut for length) and a second set of questions about student's self-perceived skill with sketching. Here we present an

analysis of the questions which were relevant to sketching in the context of the engineering design course.

In the past two weeks, how nervous would the following situation make you feel?

- Joining a group which put you in charge of sketching the design of their engineering concept
- Getting an assignment asking you to draw an engineering design to scale
- Being asked to sketch an idea you have just suggested in brainstorming
- Being asked to draw an oblique view of a model
- Being asked to draw an orthographic projection of a model

Students ranked their anxiety for each item on a five point Likert scale using the following values: Not at all, A little, A fair amount, Much, or Very Much. These were then converted to scores of 0 to 4 and averaged to arrive at the student's anxiety score. The survey also contained other questions which were not part of this analysis. A full copy of the survey questions can be found in Appendix D.

Each quarter, the survey was made available online twice, as a pre-test and a post-test to the Design Coach assignment. The survey was identical in both phases, however timing varied because each quarter the instructors set a different schedule for the assignment, and the survey must close and re-open around it. A link and invitation to the survey were posted on course management system and sent in emails to the entire course. Along with the questions, students were asked for their section number, which was used to determine if they were in a section with the Design Coach group or not. If

they were not, they were assigned to the control group. The number of students surveyed are given in Table 11. The results for each quarter are shown in Figure 68, Figure 69, and Figure 70.

Table 11: Number of students who responded to survey in Design Coach/Control group by quarter

Quarter	Design Coach N=		Control N=	
	Pre	Post	Pre	Post
Fall 2012	16	21	30	16
Winter 2012	15	7	24	12
Fall 2013	17	14	91	27

In the following figures the data was unpaired due to the low paired N. Alas, students who filled out both the pre- and post-test surveys were rare. Because the data is unpaired, a Wilcoxon rank sum test is used to calculate the P value. When considering the timing information, it is helpful to know that the pre-test surveys were posted in week 1 each quarter, and remained opened until the Design Coach assignment was due. After that, the survey was closed and the post-test opened one week later. Students were not required fill out the survey at any particular time within these periods; it was completely optional, at the instructors' request.

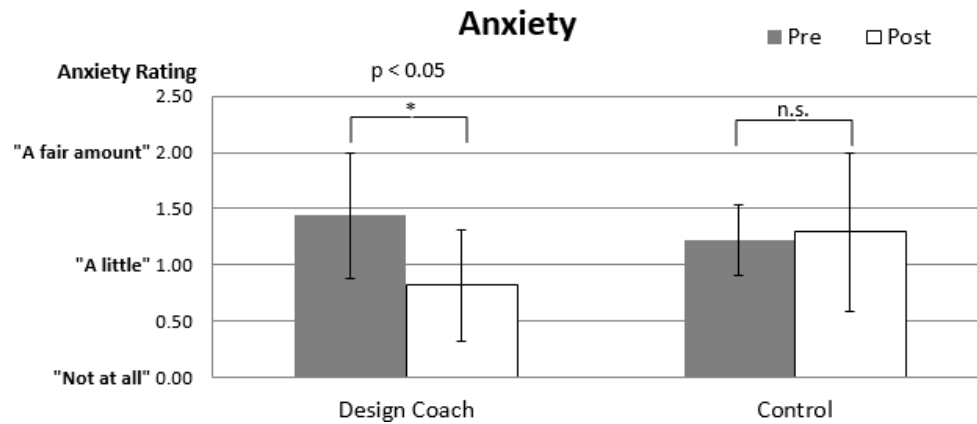


Figure 68: Sketching Anxiety Survey Results, Fall 2012

DC: $N_{pre}=16$, $N_{post}=21$, $p=0.043$ Control: $N_{pre}=30$, $N_{post}=16$, $p=0.644$

In the fall quarter of 2012, the Design Coach group's self-reported anxiety dropped significantly, while the control group's did not. During this quarter students responded to the pre-test between the 3rd and 6th week, and the post-test after the 7th week. The Design Coach assignment was due at the end of the 6th week. This quarter was also the first quarter the Sketch Tips system of Design Coach had its complete set of feedback. After this, no additional Sketch Tips were added.

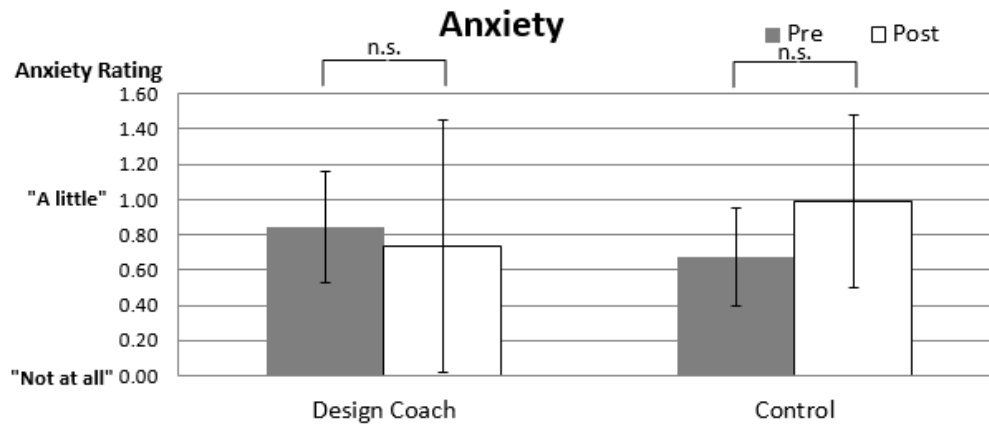


Figure 69: Sketching Anxiety Survey Results, Winter 2013

Design Coach: $N_{pre}=15$, $N_{post}=7$, $p=0.478$ Control: $N_{pre}=24$, $N_{post}=12$, $p=0.211$

Unfortunately in the winter 2013 quarter there was only one section in the Design Coach group, so the Ns were smaller. Students also filled out the pre-test earlier, between the 1st and 4th weeks. The Design Coach assignment was due week 8. They also filled out the survey later, starting in the 9th week. This quarter had the lowest starting scores of the three. Additionally, this quarter was the first quarter where Design Coach could highlight the contact surfaces for students as part of the feedback.

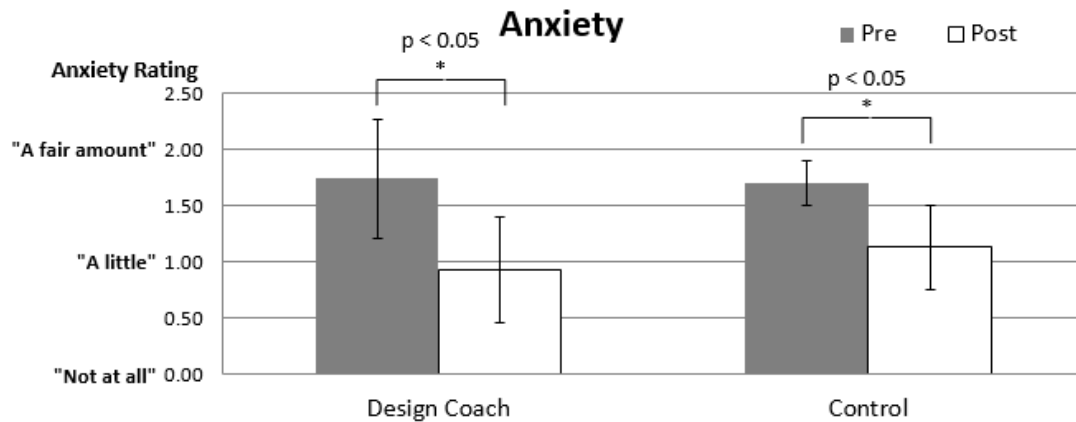


Figure 70: Sketching Anxiety Survey Results, Fall 2013

Design Coach: $N_{pre}=17$, $N_{post}=14$, $p= 0.026$ Control: $N_{pre}=91$, $N_{post}=27$, $p= 0.004$

In the fall 2013 quarter, there were two Design Coach sections again, and the control group had a much higher participation rate than previously. Significant decreases in anxiety were found in both groups, but the difference in change between Design Coach and the control group was not significant. Students filled out the pre-test between weeks 1 and 6. The Design Coach assignment was due Week 6. As in winter, students began to fill out the post-test in week 9. This quarter had the highest mean initial scores of the three for both conditions.

Overall the Design Coach group achieved a significant decrease in two out of the three quarters, while the control group only achieved a significant decrease in the third quarter. In fact, the mean of the control group increased in the two previous quarters, though the change was not significant in either case. There are many variables to consider, including timing and the wide range of instruction styles and emphasis on sketching throughout the different sections.

6.4.2 Course Performance

Another measure for the Design Coach intervention is to see if performance on the homework assignment correlates with students' performance in the course. When they were first administered, each instructor was allowed to grade the assignment on their own. To unify the scoring, a new rubric was created for the most often used version of the Design Coach assignment, the rotating cup holder. The rubric for grading the design worked as follows.

1. Explain rotation feature (up to 4 points)
 - If student drew the cup holder rotating but not rotation/rotatable fact was believed in the sketch's working memory: 1 point
 - If student drew rotated cup holder and the rotation/rotatable fact was believed in the sketch's working memory: 2 points
 - If the student also gave any functional reason for the rotation (using the structured language input), but not all facts involved in that sentence were believed by Design Coach (i.e. proven in working memory): 3 points
 - If the functional explanation was believed: 4 points
2. Explain how the base of the mechanism attaches to the armrest (up to 4 points)
 - If student stated the parts were attached in the structured language input but not all facts involved were believed, 2 points.

- If student stated the parts were attached and Design Coach believed it, 4 points
3. Explain how the cup holder part detaches from the base (up to 4 points)
- If student stated the cup holder detached from the base, but not all facts involved were believed, 2 points.
 - If student stated the cup holder detached from the base and Design Coach believed it, 4 points

Credit was awarded to the student if they completed the task but Design Coach failed to believe their explanation due to reasons other than student errors/omissions. Scores for each quarter are included in Table 13 in the next subsection.

Table 12: Correlation between students' scores on the Design Coach assignments and grades in the Design

Grade Type	N	Correlation Coefficient	Significant given N?
Graphics Grade	61	0.036	n.s.
Final Grade	49	0.225	n.s.

Student's graphics grades, their cumulative score for sketching and drafting exercises, was available for the following Design Coach sections: two sections in Winter 2012, one section in Fall 2012, one section in Winter 2013, and one section in Fall 2013 (five sections total). Their cumulative score for the entire course (final grades) were available for: two sections in winter 2012, one section in fall 2012, and one section in winter 2013 (four sections total). The correlation between Design Coach assignment scores and grades in the course were not significant for either type of grade

(Table 12). Looking at the grading data reveals that this is probably due to the fact the grade data was letter grades, and the students mostly fell into a narrow range of grades (B+ to A for final grades, B- to A for graphics grades). Plots of the grades vs the scores are shown in Figure 71.

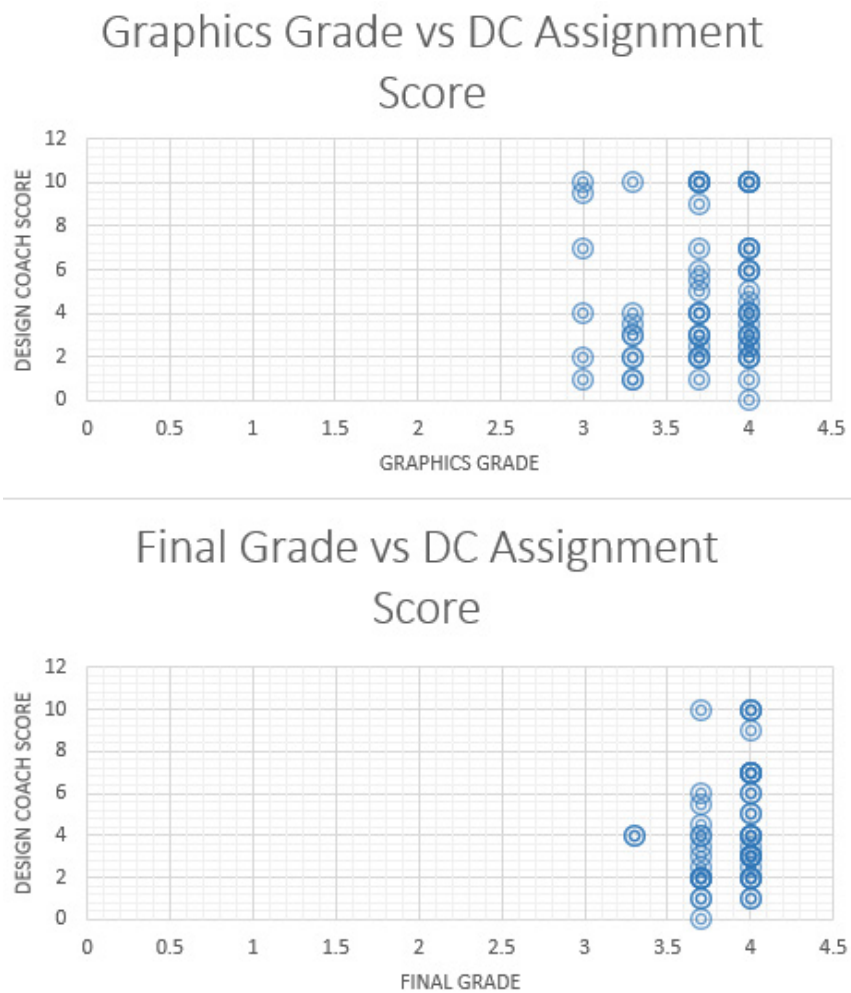


Figure 71: Plots of scores

6.4.3 Effect of Feedback

Another interesting measure is the correlation between the number of times the students requested feedback and their performance on the assignment is in Table 13.

Table 13: Correlation between scores on the Design Coach assignment and number of times feedback was requested. Scores are out of 2 for fall 2011 and 12 for all subsequent quarters.

Quarter	N	Mean # of Feedback Requests	Mean Score	Correlation Coefficient	Significant given N?
Fall 2011	23	8.5	1.26 / 2	0.38	P < 0.05
Winter 2012	26	6.6	3.60 / 12	0.04	n.s.
Fall 2012	14	10.6	4.61 / 12	0.38	n.s.
Winter 2013	11	15.9	5.63 / 12	0.31	n.s.
Fall 2013	24	12.8	6.56 / 12	0.40	P < 0.05
Totals (Winter 2012- Fall 2013 only)	75	10.7	5.03 / 12	0.38	P < 0.05

The correlation was consistently non-negative, and the overall positive correlation after fall 2011 is significant. (Fall 2011 was not included in that sum because the assignment was different that quarter; it had three parts and there was no teleological ontology, so it automatically fails half of the rubric.) One possible explanation for the drop and then increase in correlation starting in winter 2012 is that the teleological ontology had just been introduced, but the structured explanation system had yet not been created. Therefore, students may have been unable to figure why their teleological sentences weren't working regardless of the number of times they requested feedback. (Some might also have given up on feedback more quickly, resulting in the lower mean.) The consistent increase in score over the next three

quarters is also consistent with the presence of UI improvements, including an expanding pool of structured explanations and sketch tips for common errors like unattached relations/annotations.

Chapter 7

Related Work

This chapter compares and contrasts this research with research on qualitative mechanics, teleological ontology, sketch understanding, and intelligent tutoring systems.

7.1 Qualitative Mechanics

Design Coach QM is based on the QM of Nielsen (1988), and introduced several differences between the two. To summarize, in Design Coach QM:

- Force & torque transfers are explicitly represented at the level of each individual object, as forces applied to said objects.
- Force & torques cause motion, but motion does not cause motion.
- Ambiguous vectors are represented.
- New algorithms provide means to extract the necessary qualitative representations from freehand sketched input.
- Only contact surfaces are reified. The rest of the detail about the object is preserved in the sketched input.
- In addition to rigid bodies, object representations have been added for springs, gears, ropes, and pulleys.

The shift of focus from the level of motion to the level of force produces a tradeoff for the Design Coach; the advantage is that Design Coach can use the force representations in explanations. The disadvantage is that statements like “A moves because B moves” become non-trivial to prove because motion is never the antecedent of another motion. This tradeoff is appropriate for the engineering design context where a more detailed account based on forces is preferred. Stahovich et al.’s (1997) *Qualitative Rigid Body Mechanics* also rooted the motions of objects in their respective net forces, but it required contact surfaces to be identified manually.

Pearce’s *ARCHIMEDES* system introduced a qualitative model of ropes, pulleys, axels, levers, and gears (2001). *ARCHIMEDES* took in a list of the system’s components and their interconnections in symbolic form as input. It then used rules to determine how the system will move, either in a resting state (gravity is assumed) or with external forces applied. These rules model parts with specific, constrained connection points: gears, axles, levers, ropes, and pulleys. While both systems model cords at the level of cord segments, *ARCHIMEDES* transfers forces along every individual segment. Design Coach QM introduces the idea of a kinematic cord connection, allowing forces to be propagated without reifying the force on each individual cord segment. The model for pulleys is also significantly different between the two systems. In *ARCHIMEDES* a simplified pulley model is used where the pulley may only have four connection points with ropes (top, left, bottom, and right). While the pulley may be rotated to any orientation desired, the model is restricted to at most

one rope going through the pulley, and two ropes directly attaching the pulley to other objects. In contrast, Design Coach QM can model any number of ropes going through the pulley, and comfortably (from a user-input standpoint) accommodates sketches with two ropes traveling through a pulley at once. With proper glyph segmentation, any number of ropes may be used (see Section 4.5, Cord Connection Analysis, on pg. 86). Finally, Design Coach QM works with sketched input whereas ARCHIMEDES used symbolic input.

7.2 Teleological Ontologies for Engineering Design

A number of ontologies have been proposed for teleology and design rationale. As explained in Section 5.5.1 (pg. 122), Design Coach utilizes a way-of-function ontology based off of the work of Kitamura et al. (2006). Kitamura et al. showed that engineers at a semiconductor manufacturing plant could use their way-of-function ontology to build their own catalogue of functional knowledge covering several kinds of manufacturing tasks. However, in Design Coach the students are never directly exposed to the ontology itself because the goal is for the students to explain their design, not rather than organize a pool of designs.

Two other teleologies we explored included diagrams as part of their representation explicitly. Yaner and Goel (2007) proposed a five level Drawing-Shape-Structure-Behavior-Function model for linking drawings in a diagram to teleological function. In Design Coach, the representations generated by CogSketch are like their

drawing, shape, and structure levels; the metalayer and qualitative mechanics are similar to their behavior level; and the structured language input is similar to their function level. Their work was designed for use in case-based reasoning, and so the focus was on transferring models from previous cases into the current case rather than critiquing an explanation. However, with QM, our system already was able to derive behavior from the drawing without a case library, so all we needed was to add a functional layer of description. Thus we opted for the way-of-function based ontology instead.

Wang and Kim (2007) created an ontology for supporting form-function reasoning. Their representation combined the shape of an object with its proposed generic functions. For example, all containers are assigned the function of limiting downward motion of their contained substance. Such representations bake in assumptions, such as gravity in this example. Given that CogSketch produces spatial representations, we do opt to focus primarily on the functions of mechanisms which involve spatial information, including containment. However, using the way-of-function hierarchy allows us to also tackle non-spatial functions, like improving comfort, by finding ways of achieving that function for which we have representations (i.e. “moving it” and “changing its shape”).

7.3 Structured Language Input

For collecting language-like input, Design Coach uses a structured language input system directly derived from the intent dialogs of nuSketch Battlespace. The progenitor of our interfaces was *NLMENU*, a menu-based natural language interface developed by Thompson et al. (1983) as an easier way for users to construct queries for databases. A semantic grammar, generated from a subset of the system's data the user would be interested in, is used to automatically generate a menu-based interface for the user to input their query. The result is an interface that was both easy to use (the choices were clear to the user via the menus) and robust (the generation process guaranteed that the user's selections would translate into valid queries). For Design Coach, we created a core grammar from predicates and relations in the KB relevant to QM and our teleological ontology. Most of the predicates were binary, so we arrived at a subject-verb-object structure for our grammar. Like *NLMENU*, Design Coach ensures that the student's input produces well-formed input for critiquing.

A knowledge acquisition interface developed by Blythe et al. (2001) combines several techniques to allow a user to input data into an AI system using natural language. The user may define new terms and quantities for the system to take into account. While the structured language input to Design Coach does not yet facilitate the addition of new concepts, the student may add new entities and information about their properties (e.g. forces acting on them, connections between them) using the

sketching interface. We have selected the grammar to complement the sketched input, and its vocabulary is dynamically expanded as the user adds information to the sketch.

7.4 Critiquing Explanations and Designs

Intelligent tutoring system creators may seek to use a student's explanation for input to help detect and/or diagnose misconceptions they have, or to help them improve their communication ability. In the *Belvedere* system (Suthers et al., 2001) students constructed arguments using a visual language and Belvedere critiqued those arguments based on their structure. The feedback focused on the structure, ignoring the specific content. In contrast, Design Coach brings QM (and categorical knowledge in the KB) to bear on the content of the individual sentences and their meaning in the context of the sketched input. While there is not a higher order structure to critique with our structured language input, sentences may introduce new forces or additional information about the types of objects, and thus their order matters. If the information is given late in the explanation, the unproven or contradiction fact produce will cause Design Coach to give feedback, and in response the student can change the order of their sentences to improve their explanation.

The *Why2-Atlas* system (Jordan et al., 2006) critiqued students' natural language solutions to qualitative physics problems. It used a combination of three competing

analysis methods to create a first-order predicate logic representation of each sentence. It then used an assumption-based truth maintenance system tailored individually to the specific problems to diagnose errors. In contrast, Design Coach uses a single larger model to address a range of designs students may try to explain.

A variety of computer critiquing systems have been developed for other domains including software engineering (Ali, Admodisastro, et al., 2013; Qiu & Riesbeck, 2003), architecture (Oh et al., 2004), and furniture design (Oh et al., 2010). One such system, *Design Evaluator*, utilized qualitative reasoning in the form of spatial relationships and rules over sketched input to critique the design of floor plans and web page layouts (2004). Unlike Design Coach, Design Evaluator did not take in any form of language-like input. Oh et al.'s (2010) *Furniture Design Critic* used a set of pre-defined constraints to critique sketched furniture designs. Its critique system also maintained short-term and long-term models of the user which it used to adjust its critiques over time.

Ali et al. (2013) defined a taxonomy of computer-based critiquing tools. In their taxonomy, Design Coach would belong to analytical critiquing family of approaches, because it uses rules to detect problems in the user's solution and guides the user away from recognized problems. In contrast the other family of approaches they define, comparative critiquing, uses a differential analyzer against its own solution, and then guides the user to that known solution. *Sketch Worksheets* (Yin et al., 2010), an open-

domain sketch-based educational software system (also built on CogSketch) would belong to that family.

7.5 Sketch Recognition and Understanding

Sketch-based input systems have been created for tutoring systems in a variety of domains, including chemistry (Cooper et al., 2009), circuits (de Silva et al., 2007; Johnston & Alvarado, 2013), and physics. PhysicsBook (Cheema & LaViola, 2012) uses a 2D physics engine to create animations from sketched input. Design Coach uses QM instead of a numerical approach primarily because it needs to be able to understand and critique the student's *explanation* of the design and to be able to *explain its own reasoning* to the student. Another reason we use QM is at the conceptual design stage, the precise physical parameters are usually not available.

Sketch-based systems have provided tutoring in the mechanical engineering domain as well. Mechanix, a sketch-based tutoring system for statics courses (Valentine et al., 2012), was used to create statics problems involving trusses, and then guide students through them. Newton's Pen II and Newton's Tablet (Garcia, 2014; Lee et al., 2011) guided undergraduate students through the construction of free body diagrams and equilibrium equations. Students draw outlines over a background to identify the bodies, then add force arrows to those bodies individually, and finally write equations describing the situation they sketched. The system gives them feedback by

checking their work against a predefined solution. These systems are limited to specific lists of problems. These limitations exist in part because the sketch recognition techniques used limit the range of understandable input. Trained recognizers limit the user to a pre-defined set of symbols, while comparative systems require known solutions. Newton's Tablet uses the former for understanding handwritten equations and arrows, and the latter to check if the student's free body diagrams fit the boundaries given in the problem description. Mechanics and Sketch Worksheets similarly check input against that of an authored solution. A comparative system is inherently limited to its bank of given solutions, and the breadth of that bank is limited by the flexibility of the techniques used for comparison. Design Coach avoids the need for given solutions by using a general physics model and algorithms (surface contact detection and cord connection analysis) which will produce the input needed by that model.

FEASY (Murugappan & Ramani, 2009) is a sketch-based computer aided design tool for performing finite element analysis. The system was also trained to recognize a set of finite element symbols which the user may draw in their sketch to create regions of constraint, apply forces and more. Design Coach currently uses labeling at the level of whole glyphs to create the initial sources of constraint (i.e. `FixedObject`). However, adding the ability to label parts of glyphs (through a combination of edge decomposition and the concept picker or recognition) could provide the input needed for a future QM of deformable objects. FEASY is also limited to the set of symbols it has been trained to identify, whereas CogSketch gives access to

rich variety of concepts found in the knowledge base. Rather than using trained classifiers over all the strokes in the sketch, Design Coach uses top down information (i.e. glyph labels and user segmentation) to select the right targets, then uses human-like visual processing to identify the relevant qualitative relationships (surface contact, cord connection). While there are limitations to our techniques (see Section 8.2), they tend to be at the level of spatial relations rather than recognizing symbols.

Chapter 8

Conclusion

We have described a system for critiquing multimodal engineering design explanations of mechanisms, Design Coach, which was deployed in an intervention in a first-year undergraduate engineering design course at Northwestern University. Chapter 1 presented an overview of the claims and contributions. Chapter 2 provided background information on the AI systems and theory used by Design Coach: CogSketch, our sketch understanding system, and qualitative mechanics, a qualitative model of physics. Chapters 3 introduced Design Coach QM, a new, force-centered version of QM for use with sketched input and multi-modal explanations. Chapter 4 introduced extensions to that QM, including algorithms for deriving important relationships from freehand sketched input and representations for springs, gears, and cords. Chapter 5 detailed the implementation of Design Coach, including its multimodal input, teleological ontology, and algorithms for critiquing explanations and giving feedback. Design Coach was then evaluated over two corpuses: one was a variety of mechanisms, and the other a set of explanations produced by students during the classroom intervention. This evaluation was described in chapter 6, along with a description of the intervention and its impacts on students. Finally, chapter 7 discussed related work.

This chapter discusses our claims presented in Chapter 1 in light of evaluation of this work. We close with a summary of the limitations of Design Coach and future work.

8.1 Discussion of Claims and Contributions

The first claim of this thesis is:

1. Descriptions of mechanisms, specified multi-modally, using sketches and restricted natural language, must be understood via qualitative reasoning.

To support this claim, we showed that by extending QM with new representations and algorithms for interpreting sketched input, the system could achieve coverage of a meaningful range of mechanisms, including mechanisms from an engineering design course (Wetzel & Forbus, 2009b) and from a chapter on block and tackle mechanisms in a textbook (6 examples out of 10 figures) (*Basic Machines*, 1994).

Qualitative reasoning supports the ability of our system to understand in two ways. First, our new algorithms for detecting surface contact and cord connections use qualitative representations of the sketched input (e.g. perceptually salient edges and junctions) to extract the necessary qualitative spatial relationships for QM. Second, Design Coach uses QM to predict the mechanical behaviors of sketched mechanisms. These qualitative representations are also instrumental in providing the critiques of student's design explanations, leading to the second claim of this thesis:

2. The validity and clarity of an explanation of a mechanical design can be evaluated using a combination of design teleology and qualitative reasoning.

To support this claim, we demonstrated the ability of Design Coach to critique multi-modal explanations over a corpus of 240 sketched explanations created by engineering design students with an accuracy rate of 92.9%. These critiques were done using two new feedback algorithms which used QM, qualitative spatial reasoning, and a teleological ontology to determine 1) if statements made about the mechanism were true, contradictory, or unprovable, and 2) if the state transitions present in a comic graph reflected the behavior of the mechanism over time.

To summarize, the contributions of this thesis are:

1. New extensions to qualitative mechanics for use with sketched input, including a force-centered model of rigid body mechanics and representations for springs, gears, cords, and pulleys
2. Two new algorithms for extracting mechanical spatial relationships from sketched input: surface contact detection and cord connection analysis
3. Two new algorithms for generating items for critique of engineering design explanations: State Transition Verification and Sequential Explanation Analysis
4. A teleological ontology for representing higher level functions of designs involving mechanisms in the context of a first-year undergraduate engineering design course

Additionally, our classroom intervention resulted in the following contributions:

5. A corpus of 240 multi-modal explanations from engineering design students.
6. Data from the results of a sketching anxiety survey, including the detection of a significant decrease in anxiety in two out of three quarters in sections where the Design Coach assignment was introduced and one out of three quarters in control sections.
7. A finding of significant correlation between the amount of feedback requested and performance on the Design Coach assignment.

8.2 Limitations and Future Work

Our work with Design Coach reveals several areas that could be improved through future research. Our initial survey of 39 past projects from the engineering design course, revealed that about half of them (19) involved mechanisms. Adding springs and gears enabled QM to be able to perform STV over descriptions covering 16 of these designs (Wetzel & Forbus, 2009b). Non-mechanical designs in the DTC course include projects such as teddy bears that tell stories, a rooftop garden plaza for the university's student center, and an artificial whale stomach for training aquarium veterinarians. Mechanical projects which involve motion and contact in three dimensions are also difficult for Design Coach to understand, as are certain combinations of structured language input. The following sections address these areas of future research.

8.2.1 Three-Dimensional Spatial Reasoning and Representations

A number of designs encountered in the DTC class projects involved three-dimensional spatial relationships beyond the current representations of our CogSketch platform and QM implementation. Scaling up QM to three dimensions may be relatively straightforward; the vector representation could easily accommodate additional senses. However, determining 3D surface contacts from sketched input appears to require additional information. Even with top and side views provided, ambiguities about depth persisted, preventing us from being able to accurately derive surface contacts based on the sketch alone. One approach would be to ask the user for additional input about the 3D topology (e.g. using additional types of relation glyphs, using the visual-conceptual relationship chooser interface in CogSketch, or by adding a sentence like “The Cup Holder goes through the base.”) Our explanation corpus includes 77 examples where matching labeled glyphs are drawn from at least two different perspectives for future use.

While students were instructed to use orthographic views, sometimes they drew sketches from an isometric or perspective view. When orthographic views are required by instructors, it would be helpful if Design Coach could identify when the student has mistakenly used a non-orthographic view. Fortunately, it is possible to determine when a sketch depicts a three-dimensional view by the presence of certain types of junctions (Lovett et al., 2008). The next step would be to use this information to correctly judge when a glyph is drawn in a three-dimensional view and give feedback to the student

accurately. This will be complicated by students leaving in hidden lines (i.e. edges which are present in the object but obscured in the current view).

8.2.2 Richer Qualitative Reasoning

There are several areas where richer qualitative reasoning would improve feedback and/or expand the range of designs. For one, by predicting only the immediate motion of objects in the sketch, Design Coach encourages students to draw a new state for every qualitative change in behavior. This is reasonable for simple designs, but for complex ones we would prefer to let students leave out intermediate states which might seem obvious to a human. Design Coach will need more qualitative reasoning to fill in those missing states and evaluate when they make assumptions that are inconsistent with a student's explanation. A good starting point might be previous work on configuration space, including Nielsen's (1988) QM and Stahovich et al.'s (1997) work with sketched input.

Student design projects often incorporate quantities which change over time and additional types of non-rigid materials, some of which also have quantity as a property (e.g. fluids). Qualitative Process Theory (QPT) can be used to provide reasoning about quantities (Forbus, 1984). Incorporating Qualitative Process Theory would also allow for the understanding of concepts such as energy, momentum, and work. A QPT model of mechanical advantage could also be introduced to allow critique of additional teleological statements common to machines, such as "In State 1, a luff tackle system makes it easier to lift barrel." Models for the other machines such as levers, gear trains,

axles, and differential hoists could also be added and analyzed using spatial information in the sketch (e.g. relative length from fulcrum, relative difference in radii). Kim's bounded stuff ontology (Kim, 1993) would allow us to handle fluids. Kim's work also includes a model for linkages which would further extend the applicable range of designs.

The model of force in Design Coach QM could itself be enriched to allow for gravity and friction to be applied by default. Currently gravity and friction often work against other forces on an object, resulting in ambiguous force directions. By considering the type of the source of the force (e.g. is it from the actuator of the motion?) a system can disambiguate cases where a mechanism moves against gravity and friction, while still taking into account their effects (Stahovich et al., 1997).

Our model of cords currently does not include the transfer of forces from a cord to an object that contacts its side, except for pulleys that cord runs through. The model of ropes introduced by (Pearce, 2001) also lacks this feature. We hypothesize that, given the normal of the cord to the object, force may be imparted in a way similar to movable pulleys. Similarly, we lack a model for what happens to an object attached to a cord at a location somewhere between the ends of a cord segment. Such a model would allow the system to predict the motion of flags on a flag hoist and the yard and stay tackle (*Basic Machines*, 1994).

8.2.3 Improving Surface Contact Detection

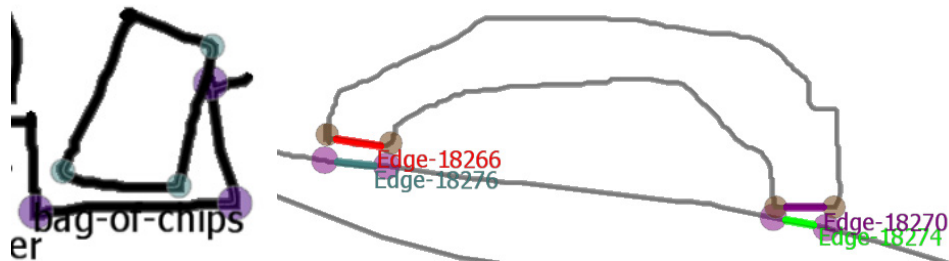


Figure 72: Erroneous surface contact resulting from when tolerances are too high.

These threshold problems came from the fact that the choice to use the same threshold that had been set based on the results of using CogSketch to develop cognitive models of spatial reasoning, to arrive at a threshold for what a human would consider connected. In the context of engineering design sketches, this first choice turned out to be a bit too loose. The two thresholds that are important for determining surface contact are the *RCC8 tolerance*, which determines how far apart edges and junctions must be to be considered disconnected, and *junction radius* which determines how large junctions are. In the example on the left of Figure 72, the RCC8 tolerance and junction radii are such that three junction-junction contacts are made between the bag-of-chips glyph and the holder. We would prefer for this to result in an edge-junction contact between the right side edge of the bag-of-chips and the top right junction found in the holder. Similarly, even though there is a gap between both pairs of edges of the two glyphs in the example on the right of Figure 72, surface contacts are found. Cases like these are the primary source of error in surface contact. We hypothesize that such errors might

be avoided by altering the thresholds dynamically. Lovett (2012) uses various techniques to vary thresholds, including using different thresholds between objects and within objects, and degrading thresholds over space as they get further away along the shape from a set reference point. Another, possibly complimentary, approach would be to use dual thresholds, i.e. to have a separate threshold for contact and non-contact, and let the more confident measure win out.

Detection of inward contacts requires the shapes be closed. In order to handle stick figures, the system could first be augmented to detect if a given edge is part of a stick figure or not. To distinguish between a closed shape with an accidental gap and a stick figure, we could a gap-detection algorithm to close the gaps in the initial edge decompositions before proceeding with surface contact detection.

8.2.4 Classifying Taut and Slack Cords in Sketched Input

In Chapter 4 we described a QM model for cord systems, such as ropes and pulleys. Since slack cords do not affect the immediate motion of attached objects, we ignored them in QM. However, designs involving cords may have states where the cord is slack, therefore it will be useful to be able to detect if a sketched cord is taut or not. One way to do this would be to use the edge decomposition routines used previously in this work. Junctions are placed at perceptually salient points in a contour; if a junction occurs in a contour and it does not contact another object on its concave side, then we can assume there is a bend in the cord and classify it as slack. Similarly, if the edge is curved and there is no object against its concave, then it is slack as well. If setting the

thresholds for inserting junctions and detecting curves proves unworkable, annotations glyphs modified to annotated edges could be used instead.

8.2.5 Improving the Range of Language Understanding

A more ideal way to use the system would entail speech recognition coupled with a full natural language understanding system, but this is beyond the state of the art in those areas currently. Further extensions would be useful to capture a broader range of semantics. In the structured language of Design Coach, compound sentences take the form “In <context>, <subject1> <verb1> <object1>, <conjunct> <subject2> <verb2> <object2>.” In compound sentences, the choices for <verb2> are also affected by the choice of <verb1>. This is because some causes/prevents statements which might make sense to a human do not make sense to the system. The two restrictions in Design Coach are:

1. Moves, rotates, and the teleological functions may only cause or prevent a function. (e.g. “Hand moves up which causes Cup Holder moves up.”)
2. Touches may not be caused or prevented by any verb (e.g. “Cup Holder moves down which causes Cup Holder touches base.”). It is simply omitted from <verb2>.

These restrictions are due to the underlying representations and the way contradictions are proven in compound statements (see Section 5.4, Sequential Explanation Analysis). Because the contradiction facts come from an analysis of the system’s own justification

trail, and because the system does not use moves, rotates, or teleological functions in rules to prove the predicates behind the other verbs, some other logic must be added to understand them. For other contradictions, we can use built-in responses in our structured explanation system, like we do for explaining nil contradictions for teleological statements. The above cases, however, are more complex.

For the first case, we might want to reinterpret the statement such that the ultimate cause of the statement is shared with the second. So, if the student says “In State 1 Hand moves up which causes Cup Holder moves up.” the system could look up the reason for Hand moving (i.e. a force) and then see if that force and the existence of Hand play are in the justification for the Cup Holder. Similarly, the way-of-function model is arranged such that functions are deduced from behaviors, not the other way around. But if the system could re-represent the function as its behavioral causes, or if more rules were added so that behaviors can be produced by some functions, e.g. `connectedTo-Directly` could be deduced from teleological `Attachment` if the way of achievement is `PermanentAdhesiveMaterials`.

The second case is more subtle. `touchesDirectly` facts are derived from visual analysis of the sketch, so for the sentence “In State 1 Cup Holder moves down which causes Cup Holder touches base.” the moves fact cannot be used to justify the cup holder touching the base. But this is technically correct because the context given is a single state. What the student really needs to be able to say is “In State 1 Cup Holder moves down which causes Cup Holder touches base *in State 2*.” Then,

something like the logic in STV's transition requirements (see Section 5.3) could be added to determine if motion occurs between states that supports this statement being true.

Additionally, some statements we allow have multiple meanings in plain English; for example if the student says "The cup holder does not move." she may mean the cup holder is not actively moving *or* that it cannot move at all. Future work is needed to determine when such distinctions are important to the rest of the explanation and to decide which interpretation should be critiqued and how.

Bibliography

- Ali, N. M., Admodisastro, N., & Abdulkareem, S. M. (2013). An Educational Software Design Critiquing Tool to Support Software Design Course. *2013 International Conference on Advanced Computer Science Applications and Technologies*, 31–36. doi:10.1109/ACSAT.2013.14
- Ali, N. M., Hosking, J., & Grundy, J. (2013). A Taxonomy and Mapping of Computer-Based Critiquing Tools, *39*(11), 1494–1520.
- Basic Machines*. (1994). Naval Education and Training Professional Development and Technology Center.
- Beilock, S. L., Gunderson, E. A., Ramirez, G., & Levine, S. C. (2010). Female teachers' math anxiety affects girls' math achievement. *Proceedings of the National Academy of Sciences of the United States of America*, *107*(5), 1860–1863.
- Blythe, J., Kim, J., Ramachandran, S., & Gil, Y. (2001). An integrated environment for knowledge acquisition. In *Proceedings of the 6th International Conference on Intelligent User Interfaces*. Santa Fe, NM.
- Cheema, S., & LaViola, J. (2012). PhysicsBook: A Sketch-Based Interface for Animating Physics Diagrams. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces* (pp. 51–60). Libson, Portugal.
- Cohn, A. (1996). Calculi for Qualitative Spatial Reasoning. *Artificial Intelligence and Symbolic Mathematical Computation, LNCS 1138*, 124–143.
- Cooper, M. M., Grove, N. P., Pargas, R., Bryfczynski, S. P., & Gatlin, T. (2009). OrganicPad: an interactive freehand drawing application for drawing Lewis structures and the development of skills in organic chemistry. *Chemistry Education Research and Practice*, *10*(4), 296. doi:10.1039/b920835f
- De Silva, R., Bischel, D. T., Lee, W., Peterson, E. J., Calfee, R. C., & Stahovich, T. F. (2007). Kirchhoff's Pen: a pen-based circuit analysis tutor. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling* (pp. 75–82). San Diego, CA.

- Forbus, K. D. (1984). Qualitative process theory. *Artificial intelligence*, 24, 84–169.
- Forbus, K. D., Nielsen, P., & Faltings, B. (1991). Qualitative spatial reasoning: The CLOCK project. *Artificial Intelligence*, 51(1-3), 417–471. doi:10.1016/0004-3702(91)90116-2
- Forbus, K. D., Whalley, P. B., Everett, J. O., Ureel, L., Brokowski, M., Baher, J., & Kuehne, S. E. (1999). CyclePad: An articulate virtual laboratory for engineering thermodynamics. *Artificial Intelligence*, 114(1-2), 297–347. doi:10.1016/S0004-3702(99)00080-6
- Forbus, K., Usher, J., & Chapman, V. (2003). Sketching for military courses of action diagrams. In *Proceedings of IUI'03* (pp. 61–68). Miami, Florida.
- Forbus, K., Usher, J., Lovett, A., Lockwood, K., & Wetzel, J. (2011). CogSketch: Sketch Understanding for Cognitive Science Research and for Education. (C. Hölscher, T. F. Shipley, M. O. Belardinelli, J. A. Bateman, & N. Newcombe, Eds.) *Topics in Cognitive Science*, 3(4), no–no. doi:10.1111/j.1756-8765.2011.01149.x
- Garcia, R. D. L. A. (2014). *Intelligent Statics Tutoring Systems as Tools for Undergraduate Mechanical Engineering Education*. UC Riverside.
- Johnston, D., & Alvarado, C. (2013). *Sketch Recognition of Digital Logical Circuits*. University of California, San Diego.
- Jordan, P., Makatchev, M., Pappuswamy, U., Vanlehn, K., & Albacete, P. (2006). A Natural Language Tutorial Dialogue System for Physics. In *Nineteenth International Florida Artificial Intelligence Research Society Conference*. Melbourne Beach, Florida, USA: AAAI Press.
- Kim, H. (1993). *Qualitative reasoning about fluids and mechanics*. Northwestern University.
- Kitamura, Y., Koji, Y., & Mizoguchi, R. (2006). An ontological model of device function: industrial deployment and lessons learned. *Applied Ontology*, 1, 237–262.
- Lee, C., Stahovich, T., & Calfee, R. C. (2011). AC 2011-2252 : A PEN-BASED STATICS TUTORING SYSTEM A Pen-Based Statics Tutoring System.

- Lovett, A. (2012). *Spatial Routines for Sketches: A Framework for Modeling Spatial Problem-Solving*. Northwestern University.
- Lovett, A., Dehghani, M., & Forbus, K. (2006). Efficient learning of qualitative descriptions for sketch recognition. In *Proceedings of the 20th International Qualitative Reasoning Workshop*. Hanover, New Hampshire.
- Lovett, A., Dehghani, M., & Forbus, K. (2008). Building and Comparing Qualitative Descriptions of Three-Dimensional Design Sketches Comparison via Analogy. In *Proceedings of the 22nd International Qualitative Reasoning Workshop*. Boulder, CO.
- Lovett, A., Forbus, K., & Usher, J. (2007). Using Qualitative Representations and Analogical Mapping to Solve Problems from a Spatial Intelligence Test. In *Proceedings of the 21st International Qualitative Reasoning Workshop*. Aberystwyth, U.K.
- Murugappan, S., & Ramani, K. (2009). FEASY: A Sketch-Based Interface Integrating Structural Analysis in Early Design. In *Proceedings of the ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference* (pp. 1–10). San Diego, CA.
- Nielsen, P. E. (1988). *A Qualitative Approach to Rigid Body Mechanics*. University of Illinois at Urbana-Champaign.
- Oh, Y., Do, E. Y.-L., & Gross, M. D. (2004). Intelligent critiquing of design sketches. *American Association for Artificial Intelligence Fall Symposium - Making Pen-based Interaction Intelligent and Natural.*, 127–133.
- Oh, Y., Gross, M. D., Ishizaki, S., & Do, E. Y.-L. (2010). A Constraint-Based Furniture Design Critic. *Research and Practice in Technology Enhanced Learning*, 05(02), 97–122. doi:10.1142/S1793206810000864
- Pearce, J. P. (2001). *Qualitative Behavior Prediction for Simple Mechanical Systems*. Massachusetts Institute of Technology.
- Plamondon, R., & Srihari, S. N. (2000). On-Line and Off-Line Handwriting Recognition : A Comprehensive Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1), 63–84. doi:10.1109/34.824821

- Qiu, L., & Riesbeck, C. K. (2003). Facilitating critiquing in education: The design and implementation of the Java Critiquer. In *Proceedings of the International Conference on Computers in Education*.
- Stahovich, T., Davis, R., & Shrobe, H. (1997). Qualitative rigid-body mechanics. In *American Association for Artificial Intelligence* (pp. 138–144).
- Suthers, D., Connelly, J., Lesgold, A., Paolucci, M., Toth, E. E., & Weiner, A. (2001). Representational and Advisory Guidance for Students Learning Scientific Inquiry. In *Smart Machines in Education: The Coming Revolution in Educational Technology* (pp. 7–35).
- Thompson, C. W., Ross, K. M., Tennant, H. R., & Saenz, R. M. (1983). Building Usable Menu-Based Natural Language Interfaces to Databases. In *Proceedings of the 9th International Conference on Very Large Data Bases* (pp. 43–55).
- Ullman, D. G., Wood, S., & Craig, D. (1990). The importance of drawing in the mechanical design process. *Computers & Graphics*, 14(2), 263–274.
doi:10.1016/0097-8493(90)90037-X
- Valentine, S., Vides, F., Lucchese, G., & Turner, D. (2012). Mechanix: A Sketch-Based Tutoring System for Statics Courses. In *24th Annual Conference on Innovative Applications of Artificial Intelligence* (pp. 2253–2260). Toronto, Canada.
- Wang, E., & Kim, Y. (2007). Form-Function Reasoning for Product Shape Ontology. In *Proceedings of the First International Workshop on Semantic Web and Web 2.0 in Architectural, Product and Engineering Design*. Busan, Korea.
- Wetzel, J., & Forbus, K. (2008). Integrating Open-Domain Sketch Understanding with Qualitative Two-Dimensional Rigid-Body Mechanics. In *Proceedings of the 22nd International Workshop on Qualitative Reasoning*. Boulder, CO.
- Wetzel, J., & Forbus, K. (2009a). Automated Critique of Sketched Mechanisms. In *Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference*. Pasadena, CA.
- Wetzel, J., & Forbus, K. (2009b). Automated Critique of Sketched Designs in Engineering. In *Proceedings of the 23rd International Workshop on Qualitative Reasoning*. Ljubljana, Slovenia.

- Wetzel, J., & Forbus, K. (2010). Design Buddy: Providing Feedback for Sketched Multi-Modal Causal Explanations. In *Proceedings of the 24th International Workshop on Qualitative Reasoning*. Portland, Oregon.
- Wetzel, J., & Forbus, K. (2012). Teleological representations for multimodal design explanations. In *Proceedings of the 26th International Workshop on Qualitative Reasoning*. Los Angeles, California.
- Wobbrock, J. O., Wilson, A. D., & Li, Y. (2007). Gestures without libraries, toolkits or training: a 1 recognizer for user interface prototypes. *Proceedings of the 20th annual ACM symposium on User interface software and technology UIST 07*, 85(2), 159. doi:10.1145/1294211.1294238
- Yaner, P. W., & Goel, A. (2007). Understanding Drawings by Compositional Analogy. In *International Joint Conference on Artificial Intelligence, IJCAI-2007* (pp. 1131–1137).
- Yin, P., Forbus, K. D., Usher, J., Sageman, B., & Jee, B. D. (2010). Sketch Worksheets : A Sketch-based Educational Software System Background : CogSketch. In *Proceedings of the 22nd Annual Conference on Innovative Applications of Artificial Intelligence*. Atlanta, GA.

Appendix A

Rules for the Teleological Ontology

In response to student feedback, these rules were made as loose as possible.

Attachment Rules

```
(functionAchievedVia Attachment AdhesiveMaterials
  (TheList ?obj1 ?obj2 ?adhesive) (TheList ?context))
```

is true when:

```
(and
  (in ?context (touches ?obj1 ?adhesive))
  (in ?context (touches ?obj2 ?adhesive))
  (isa ?adhesive AdhesiveMaterial)))
```

```
(functionAchievedVia Attachment TempAdhesiveMaterials
  (TheList ?obj1 ?obj2 ?adhesive) (TheList ?context))
```

is true when:

```
(and
  (in ?context (touches ?obj1 ?adhesive))
  (in ?context (touches ?obj2 ?adhesive))
  (isa ?adhesive TempAdhesiveMaterial)))
```

```
(functionAchievedVia Attachment InterlockingParts
  (TheList ?obj1 ?obj2 ?adhesive) (TheList ?context))
```

is true when:

```
(in ?context (touches ?obj1 ?obj2))
```

Detachment Rules

```
(functionAchievedVia Detachment TempAdhesiveMaterials
  (TheList ?obj1 ?obj2 ?adhesive) (TheList ?c1 ?c2))
```

is true when:

```
(and
  (functionAchievedVia Attachment TempAdhesiveMaterials
    (TheList ?obj1 ?obj2 ?adhesive)) (TheList ?c1))
  (in ?c2 (doesNotTouch ?obj1 ?obj2))
  (in ?c2 (doesNotTouch?obj1 ?adhesive))
  (in ?c2 (doesNotTouch?obj2 ?adhesive)))
```

```
(functionAchievedVia Detachment InterlockingParts
  (TheList ?obj1 ?obj2 ?adhesive) (TheList ?c1 ?c2))
```

is true when:

```
(and
  (functionAchievedVia Attachment InterlockingParts
    (TheList ?obj1 ?obj2)) (TheList ?c1))
  (in ?c2 (doesNotTouch ?obj1 ?obj2)))
```

Adapts to Change in Size Rules

```
(functionAchievedVia AdaptInSize AdjustableParts
  (TheList ?base ?changingPart) (TheList ?c1 ?c2))
```

Is true when:

```
(and
  (functionAchievedVia Attachment ?way
    (TheList ?base ?changingPart)) (TheList ?c1))
  (functionAchievedVia Attachment ?way
    (TheList ?base ?changingPart)) (TheList ?c2))
  (in ?c2 (movable ?base))
```

```
(functionAchievedVia AdaptInSize AdjustableParts
  (TheList ?base ?changingPart) (TheList ?c1 ?c2))
```

Containment Rules

```
(functionAchievedVia Containment Enclosure
  (TheList ?containedObj ?container) (TheList ?context))
```

Is true when:

```
(or
  (in ?context (rcc8-TPP ?container ?containedObj))
```

```
(in ?context (rcc8-NTPP ?container ?containedObj))
```

```
(functionAchievedVia Containment ProhibitDownwardMotion
  (TheList ?containedObj ?container) (TheList ?context))
```

Is true when:

```
(and
  (in ?context (transConstraint ?containedObj Down))
  (in ?context (transConstraint ?containedObj Quad3))
  (in ?context (transConstraint ?containedObj Quad4)))
```

Increase Comfort Rules

```
(functionAchievedVia IncreaseComfort ChangingShape
  (TheList ?changedPart) (TheList ?c1 ?c2))
```

Is true when:

```
(differentInk ?changedPart ?c1 ?changedPart ?c2)
```

```
(functionAchievedVia IncreaseComfort MoveWithinReach
  (TheList ?part) (TheList ?context))
```

Is true when:

```
(or
  (and (in ?context (transMotion ?part ?dir))
    (different ?dir ZeroQVector))
  (and (in ?context (rotMotion ?part ?rdir))
    (different ?rdir ZeroRotDir)))
```

Appendix B Mechanism Corpus Sketches

The following figures show screen captures of a few mechanisms drawn in CogSketch, to demonstrate the range of mechanisms Design Coach QM can handle. In each sketch QM was used to identify the direction of motion of every rigid object in the sketch (translation and rotation). Blue arrows indicate the direction of each object's motion. Bent arrows indicate rotation, and an empty circle indicates zero translational motion. (These arrows were drawn using the "Draw Next Motion" feature found in the Design menu of CogSketch Design Coach.)

DTC Designs

Book Holder

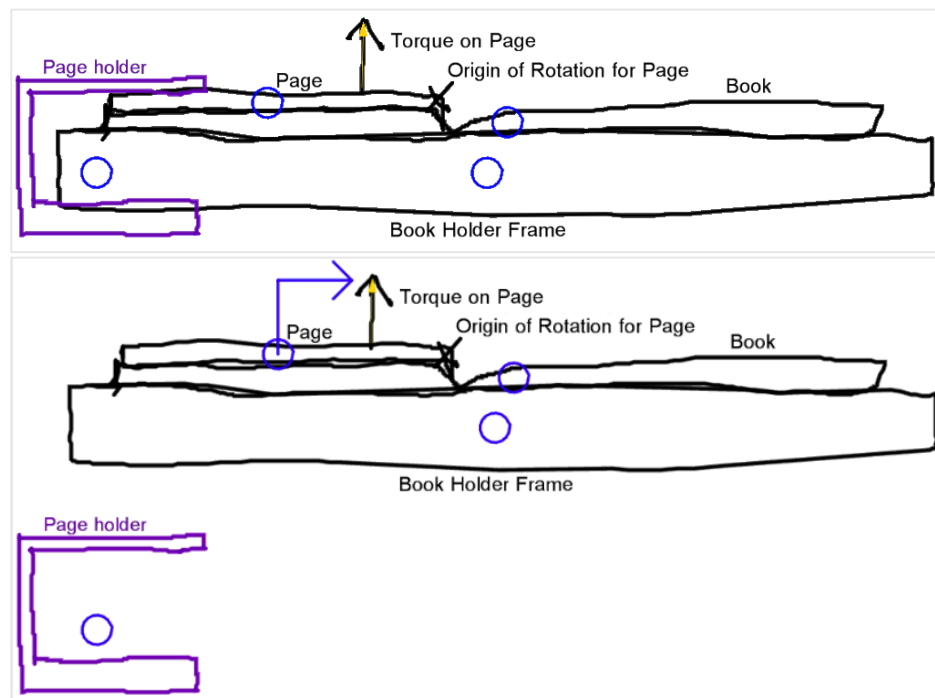


Figure 73: Book Holder
(The page is modeled as a RigidBody.)

The book holder project used page holders to hold the book open. In the sketch in Figure 73, we draw the book holder from the bottom w.r.t. the book being held open. We assumed a clockwise force acting on the page to represent the tendency of the page to attempt to flip upwards. The page will rotate as long as the holder is not in place.

Detachable Paint Roller

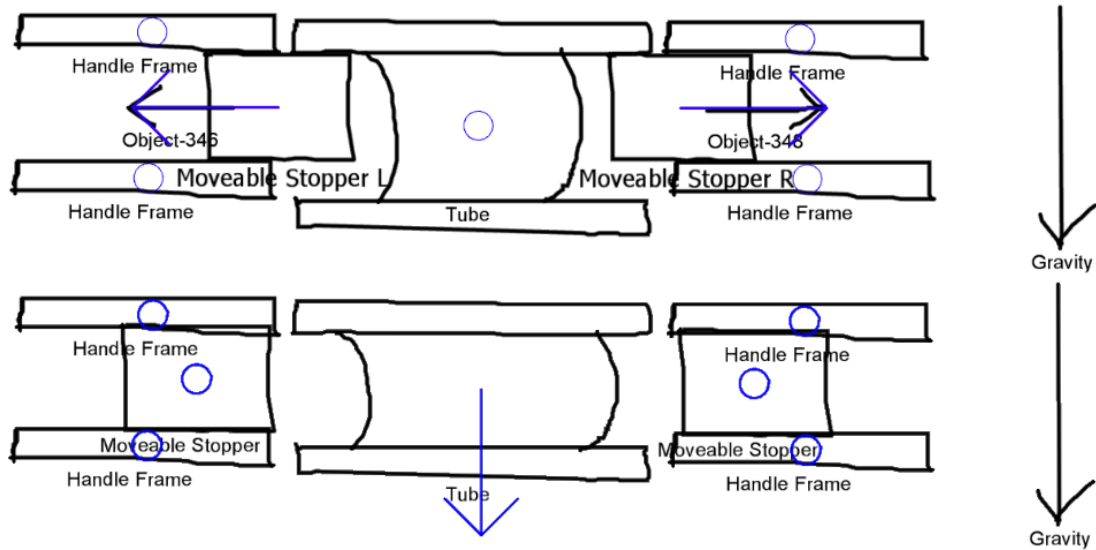


Figure 74: Releasing the Detachable Paint Roller

The detachable paint roller project allowed a paint roller to quickly be released from its handle. Here we drew the roller from the end. In the first state, the stoppers prevent the tube from moving down in response to gravity. The stoppers are pulled out of the way via forces from another mechanism in the handle (not shown) and the tube is then free to fall.

Mini Baja

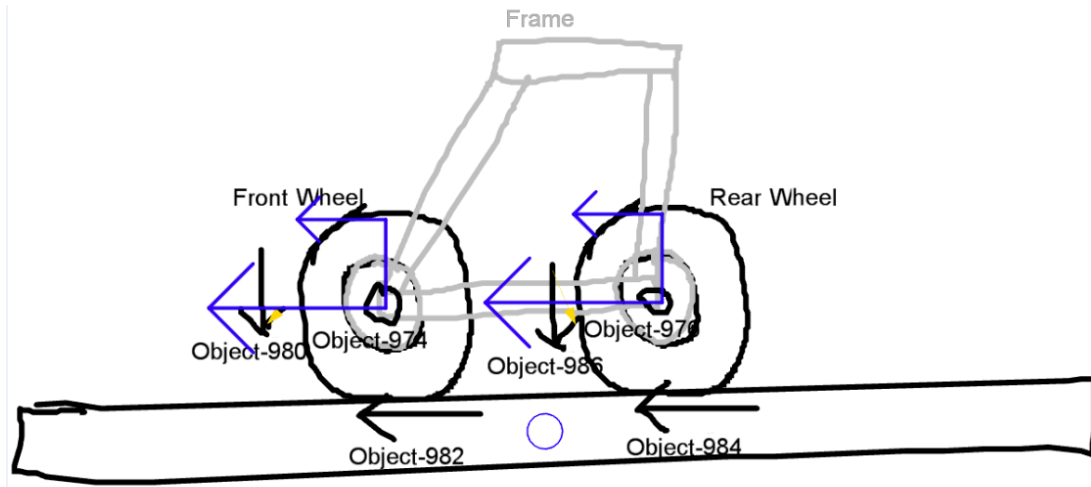


Figure 75: Mini Baja

The Mini Baja is a small 4-wheeled vehicle. Here we show how reactionary friction forces applied to the wheels will cause them to move forward as they roll.

Double-Sided Switch

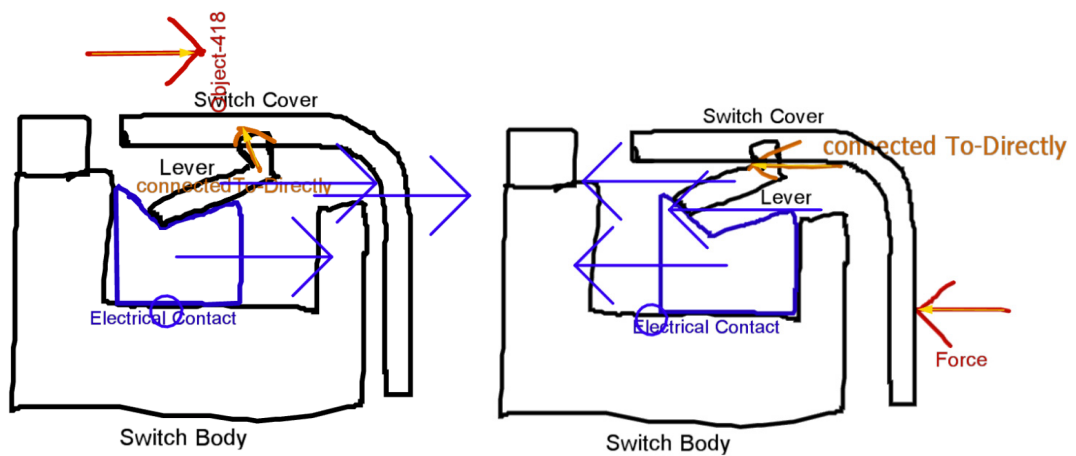


Figure 76: Double-Sided Switch

This switch can be pressed from two different angles, connecting different sides of the electrical contact inside.

One-Handed Nail Clipper

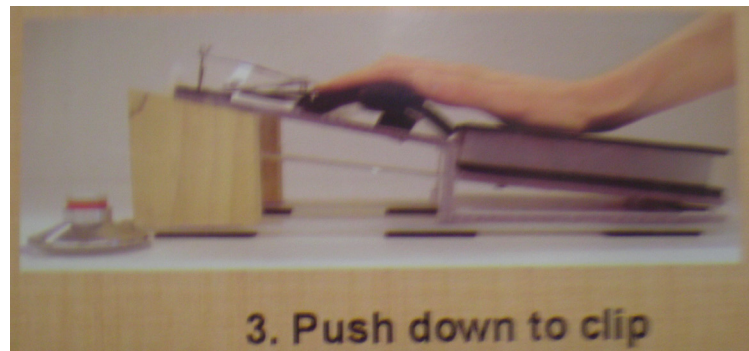


Figure 77: Picture of the One-Handed Nail Clipper Prototype

The One-Handed Nail Clipper allows a user to clip their nails with one hand by using spring forces to actuate a nail clipper.

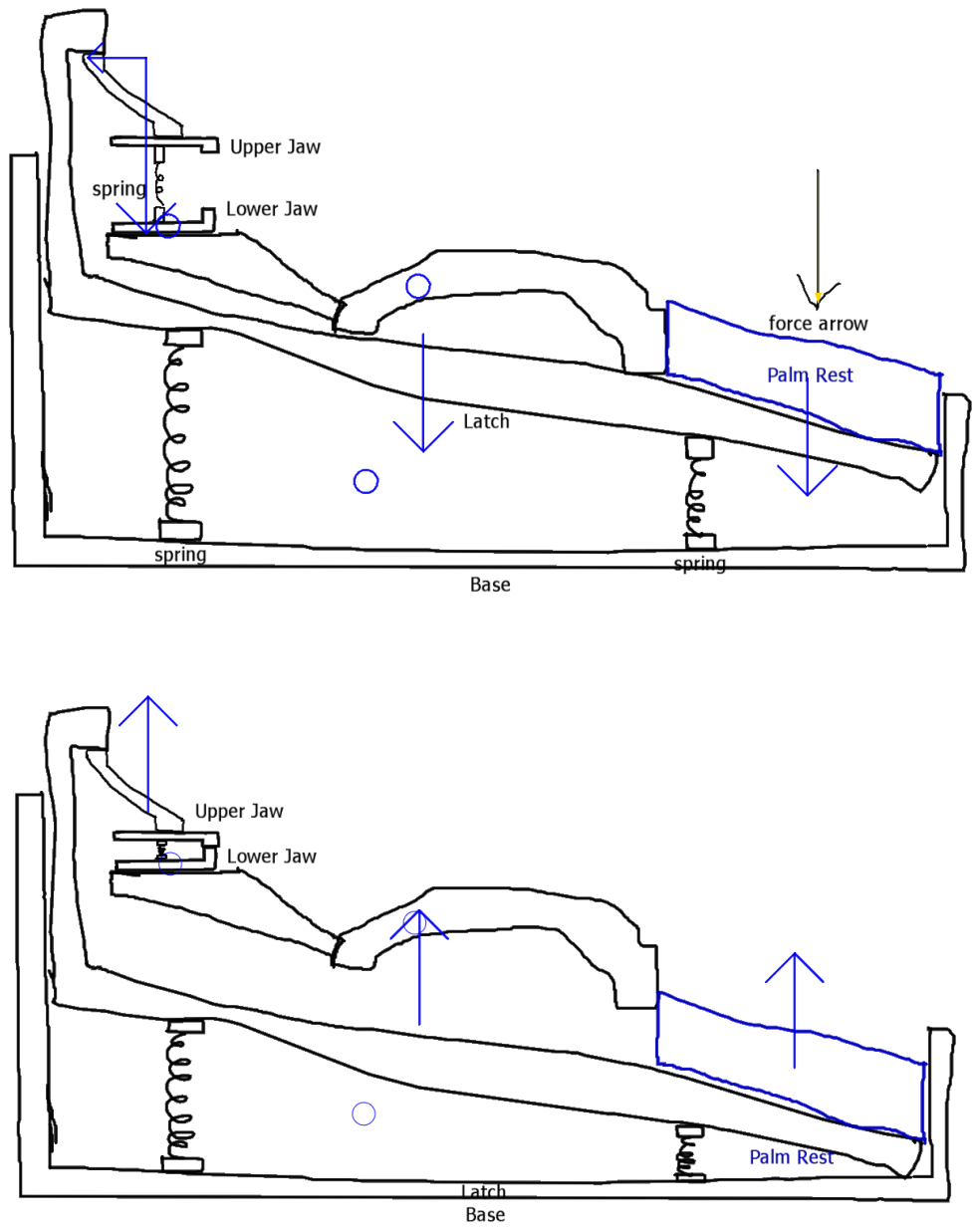


Figure 78: One-Handed Nail Clipper sketch

Tobeggan

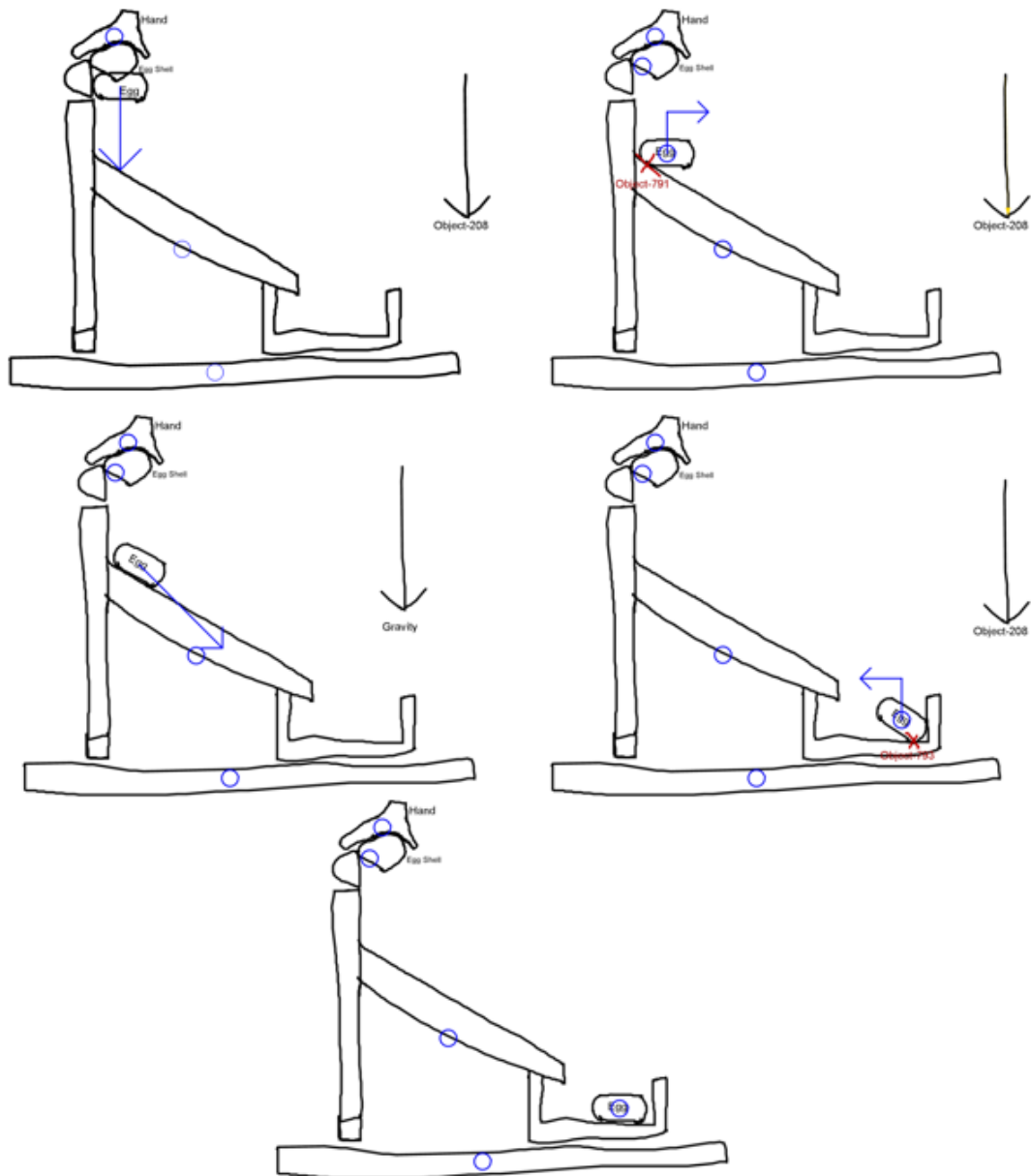


Figure 79: Tobeggan (One-Handed Egg Cracker)

The Tobeggan allows a user to crack an egg with one hand; the contents of the egg then move down a slide into a bowl.

Abigator

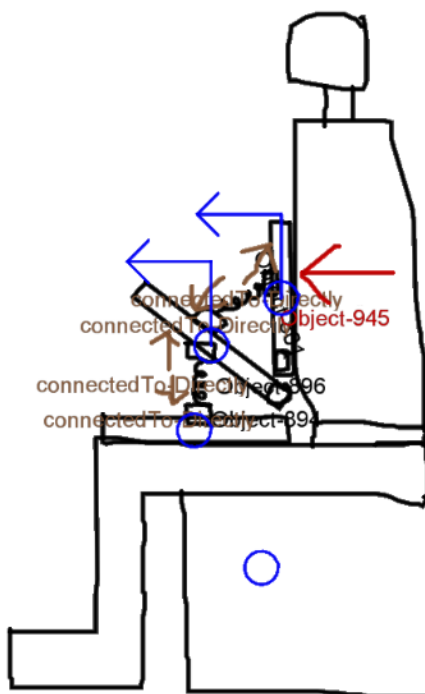


Figure 80: Abigator (Abdominal Exerciser)

The Abigator is a device for exercising one's abdominal muscles while seated or in a wheelchair. Its parts rotate in response to the user leaning forward.

Wheelchair Stabilizer

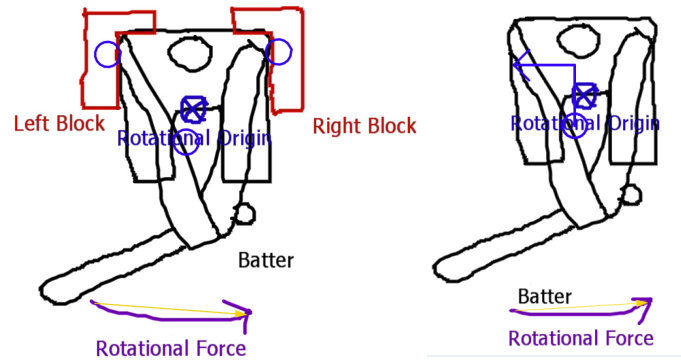


Figure 81: Wheelchair Stabilizer (for playing baseball)

The Wheelchair Stabilizer allows a batter to swing without rotating their chair. Without the blocks, the batter will rotate.

Recliner

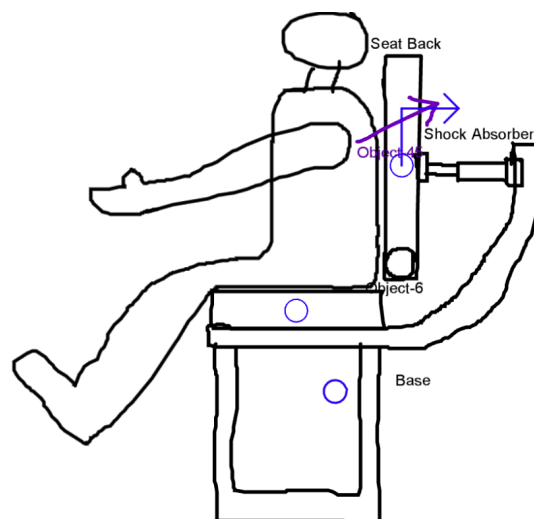


Figure 82: Recliner

The recliner uses rotates backwards in response to the user leaning back.

Under-Armrest Rotatable Cup Holder

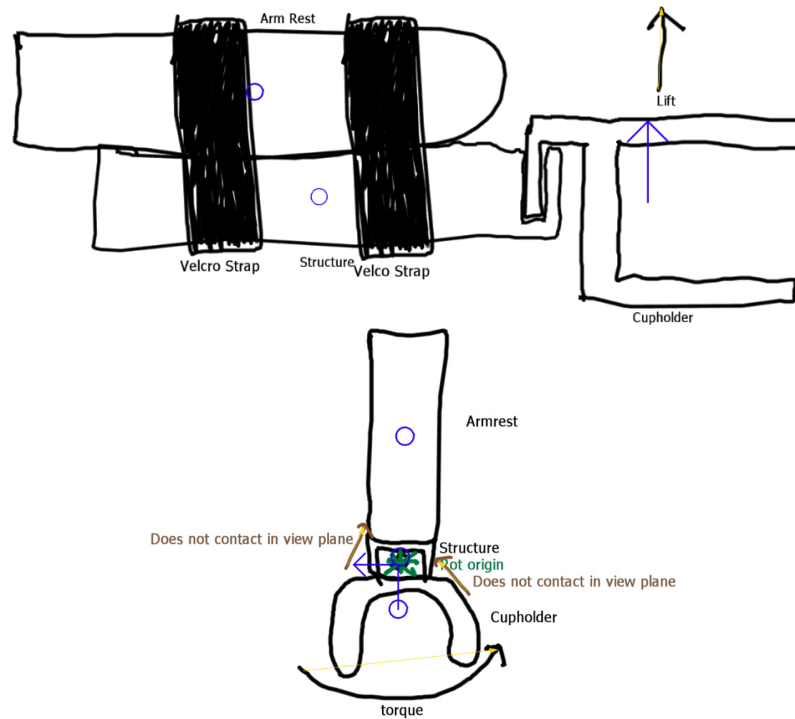


Figure 83: Under-Armrest Rotatable Cup Holder

The Under-Armrest Rotatable Cup Holder attaches to an armrest using Velcro, and the cup holder part can rotate or be removed for easy washing.

Gamma Handle

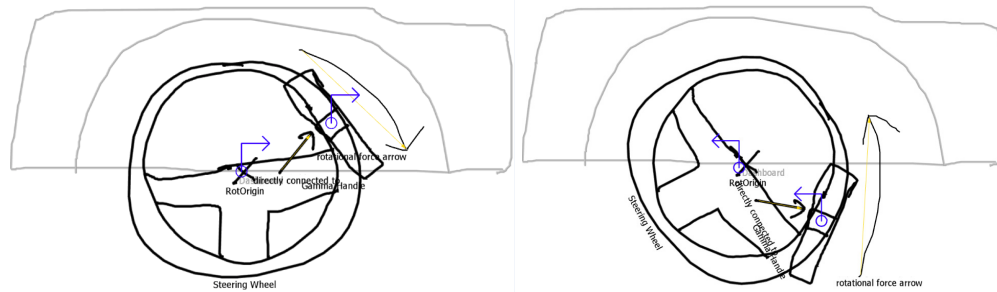


Figure 84: Gamma Handle

The Gamma Handle attaches to a steering wheel, allowing the user to apply force to it to rotate the wheel instead applying force to the wheel directly.

Dynamic Dropper



Figure 85: Dynamic Dropper

The Dynamic Dropper is designed to allow patients with paralysis exercise their leg muscles by simulating the motion of walking. The foot is attached by Velcro to a hinged platform which is connected to a springy platform. As the foot steps down in response to gravity, the platform moves down until the springy platform is in contact with the base, then the hinged platform rotates. Once down, springs will cause the foot to rise up again.

Basic Machines

The following four examples come from Chapter 3 of Basic Machines (#CITE 1994).

A Runner

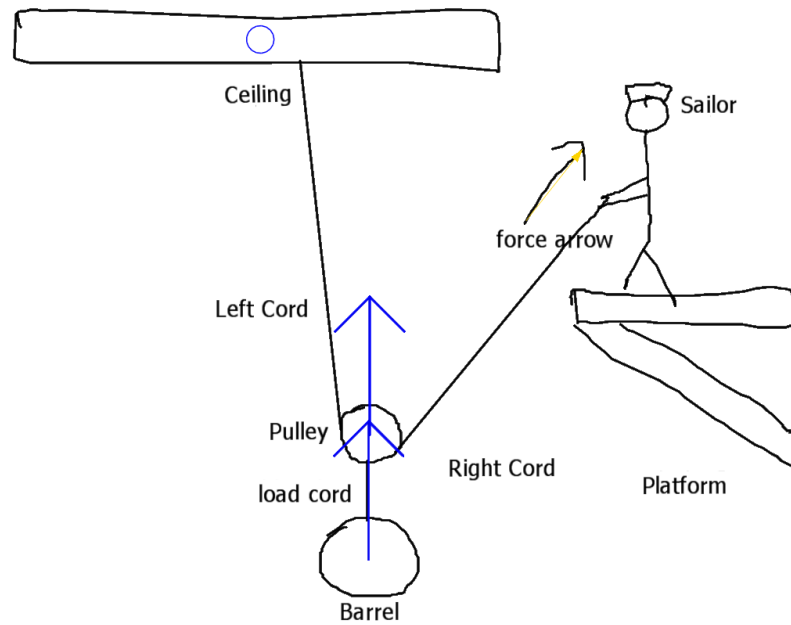


Figure 86: A Runner

A runner is used to lift objects with upward force, and consists of one movable pulley.

Gun Tackle

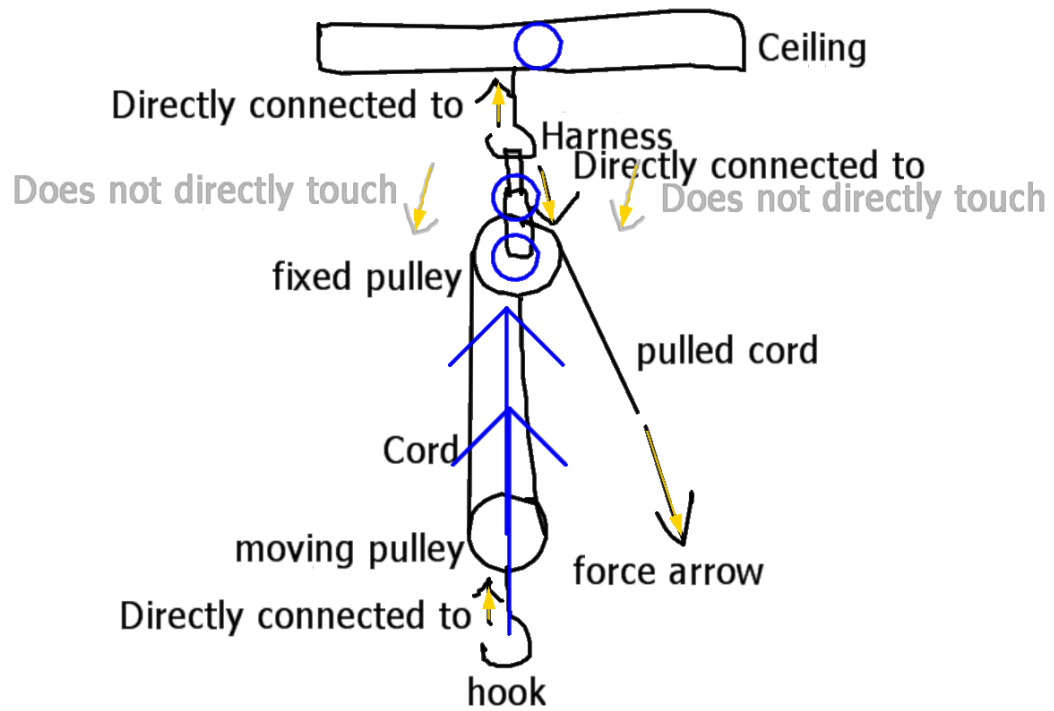


Figure 87: Gun Tackle

A gun tackle lifts objects with a downward force, using one fixed and one movable pulley.

Luff Tackle

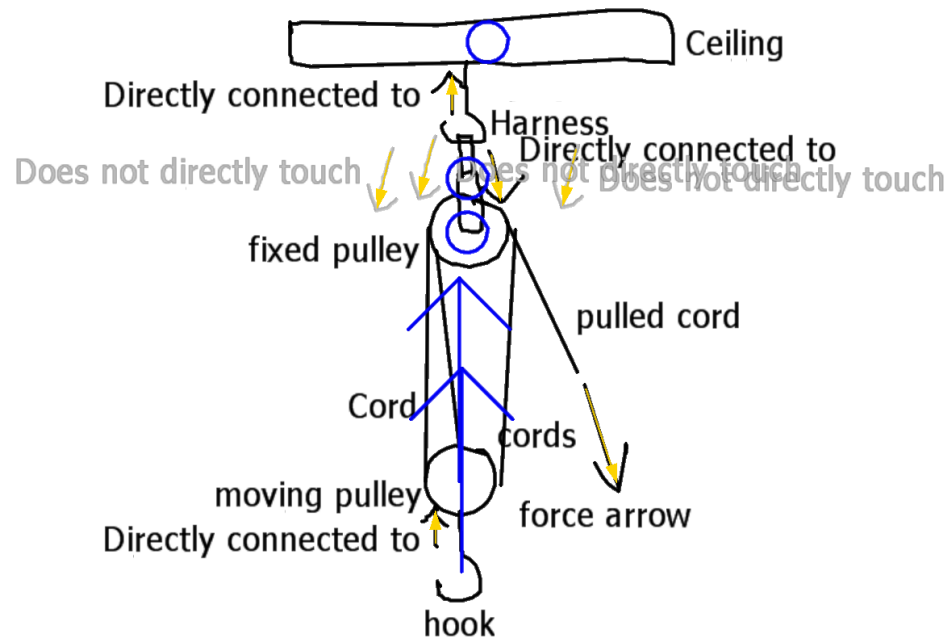


Figure 88: Luff Tackle

A luff tackle uses a downward force to lift a weight with even more mechanical advantage, while using the same number of mechanical pulleys.

Luff Upon Luff

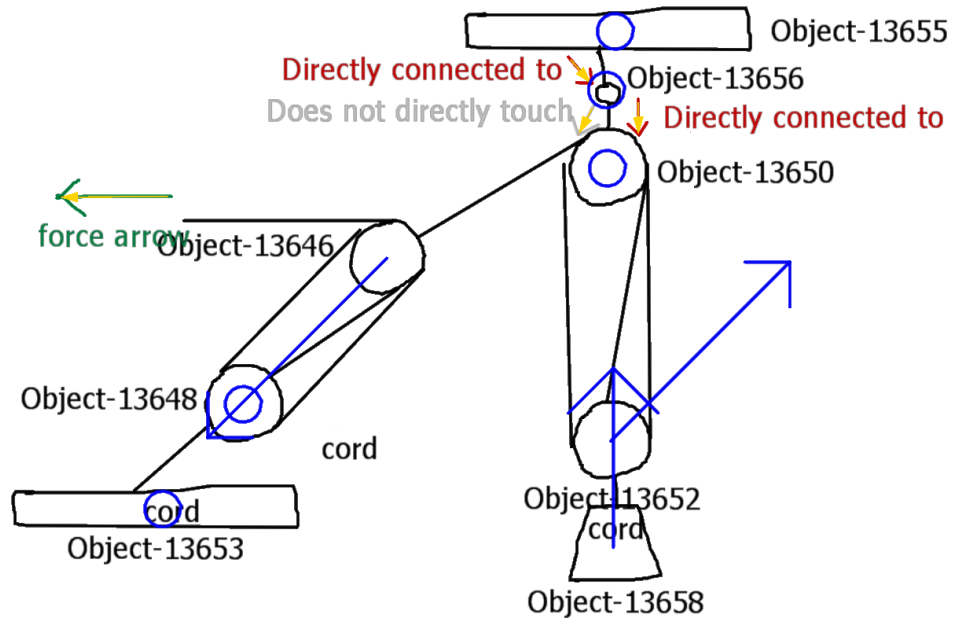


Figure 89: Luff Upon Luff

A luff upon luff configuration uses two luff tackles connected together to get even more mechanical advantage.

Other Examples

The following examples were created for further demonstration purposes.

Spring Button

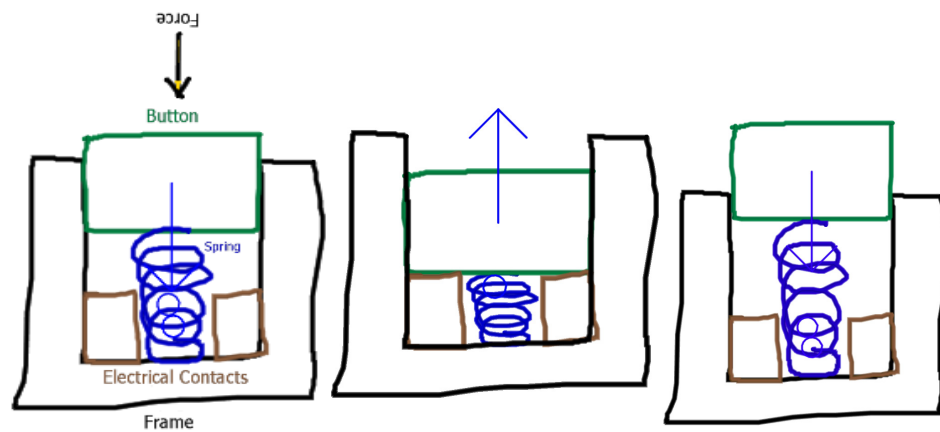


Figure 90: Spring Button

In this spring-loaded button design, the user can press down the button to bridge two electrical contacts. When they let go, the button pops back up into place.

Gear Train

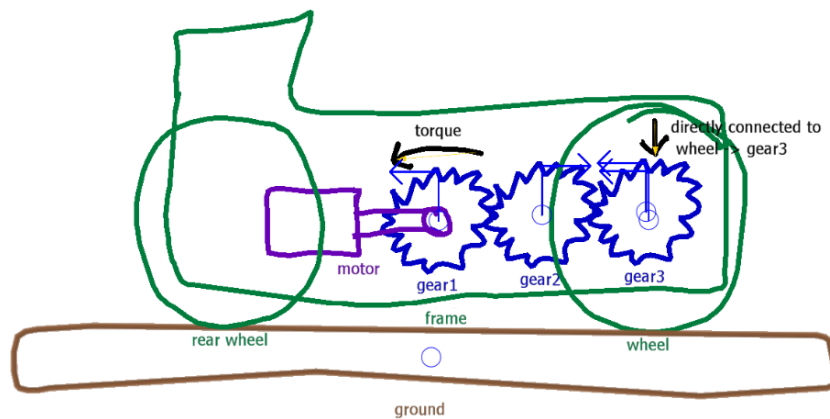


Figure 91: Gear Train

In this gear train, gears transferring rotational torque in series and transfer it to the front wheel.

Appendix C Design Coach Assignment Handouts

The following handouts are included:

- Under Armrest Cup Holder (3-part version; used Fall 2011)
- Dynamic Dropper (2-part version; used Spring 2012)
- Gamma Handle (2-part version; used Spring & Fall 2012)
- Under Armrest Cup Holder (2-part version; used Winter 2011, Fall 2012, Winter 2013, and Fall 2013)

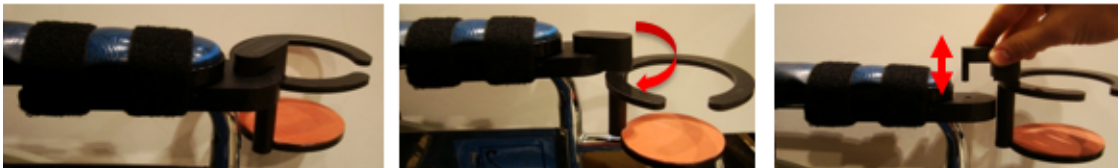
CogSketch Exercises - EDC fall 2011

There are three goals to this exercise:

1. To practice communicating through sketching
2. To use the CogSketch tool to get feedback during the process of communicating through sketching
3. To inform the development of the CogSketch tool

An EDC team designed the Under Armrest Rotating Cup Holder for their client at RIC. The design is shown below.

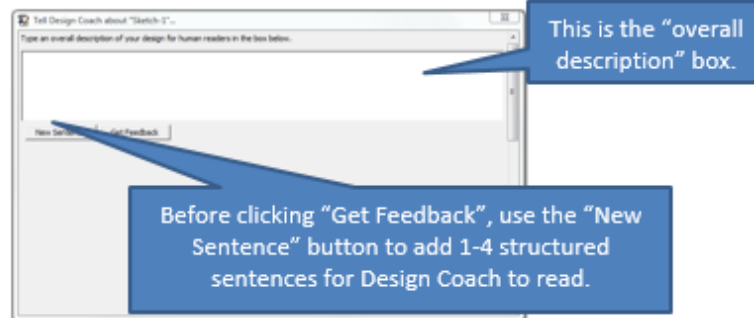
WQ11, Section 5, Team 3: Branden Alegbeleye, Angela Jiang, Jacob Kelter, Patrick Weitzel



- Easy to attach using Velcro
- Doesn't take up user space
- Cup holder rotates giving user flexibility in placement
- Cup holder slides off for quick attachment/removal

0. Use CogSketch for this assignment
 - a. If you have a PC, you may download it here: <http://www.org.northwestern.edu/software/cogsketch/index.html> (or google "download cogsketch")
 - OR-
 - You may use it on computers in the EDC classrooms
 - OR-
 - You may visit the QRG Lab to use it on a tablet PC. E-mail <researcher email> to set up a lab session.
 - b. When you use CogSketch for the first time, check out the Design Coach Tutorial ("Help" menu-> "Tutorials"-> "Design Coach Tutorial") to get an idea of how it works.
1. Create a sketch of the above design using CogSketch:
 - a. Under the "File" menu select "New Design Sketch" to get started
 - b. Sketch it. **Save often.** Include two or more subsketches, as shown in the tutorial, to show how the device operates.

- c. Write a 3-5 sentence description of the design in the “overall description” box at the top of the Tell Window.



- d. Build 1-4 structured sentences, as shown in the tutorial, to tell CogSketch what the device does. (CogSketch cannot understand what is written in the overall description [box](#).) Then, use the Feedback feature to get feedback about the design.
- e. Comment on the feedback it gave you in the overall description box. If you have any trouble using CogSketch, please write about it in the box as well. Be sure to save the sketch one last time before closing it.
2. Add some context for the design, such as the setting or a specific use-case scenario, to help illustrate some of the benefits or limitations of this design. For example, the device might be used in a hospital setting for a range of patients, or it could be used outdoors to carry a cold drink on a hot, humid day, or many other possible scenarios.
- a. **Repeat steps a-e from part 1, sketching the device being used in some context. Remember to save often!**
3. Consider possible refinements to the design. Use CogSketch to draw and describe one variation, and get feedback for it. Also describe the purpose of the refinements in (1-3 sentences) in the “overall description” box at the top of the Tell window.
- a. **Repeat steps a-e from part 1, sketching your refined variation of the device. Remember to save often!**

Your submission should consist of 3 sketch files, one for each of parts 1-3. E-mail them to <instructor e-mail>

If you encounter difficulty using CogSketch, and/or would like to use it on a tablet PC, please e-mail <researcher-email> to set up a lab session.]

CogSketch Activity – EDC2 Spring 2012

The goal of this activity is to gather input from users in order to improve the CogSketch tool.

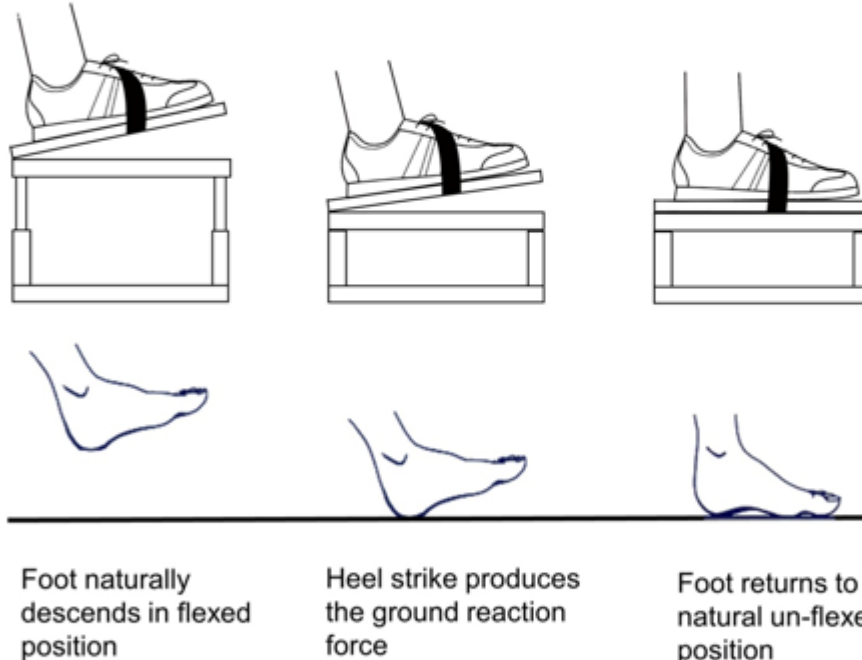
0. Take the CogSketch Tutorials

- a. If this is your first time using CogSketch or you want a refresher, open the “Help” menu-> “Tutorials”-> “Design Coach Basic Tutorial” (make sure the name is correct) to learn how it works.
- b. When you are done, do the “Design Coach Rotation Tutorial”

1. Create a sketch of the following design using CogSketch:

EDC wq09, sec 4, team 2

Daniel Kim, Nashida Alam, Razna Ahmad, Stefano Pianura



Foot naturally descends in flexed position

Heel strike produces the ground reaction force

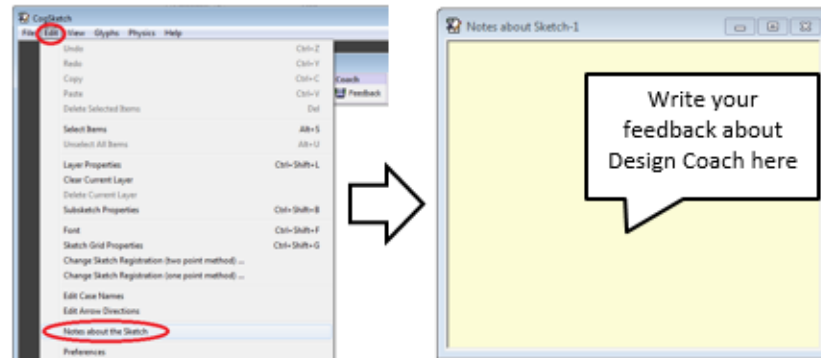
Foot returns to a natural un-flexed position

The Problem: Patients with spinal cord injury (SCI) experience rapid decline in bone mineral density (BMD), particularly in the lower extremities, due to minimal or absent voluntary muscle activity

The Solution: A platform that guides the wheelchair user’s foot to replicate the heel strike in the walking motion. This produces a ground reaction force that strengthen the bone

- a. Under the “File” menu select “**New Design Sketch**” (*not* “New Sketch”) to get started.
- b. Open the **sketch notes window**, where you will write your user feedback:

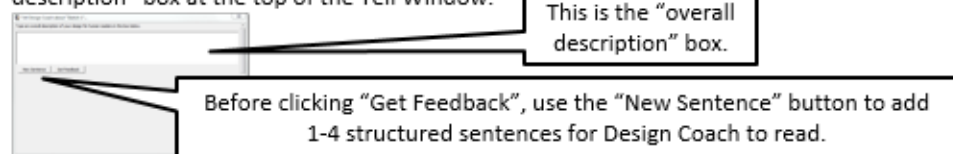
- i. On the menu bar click edit and then “Notes about the sketch”



- ii. You will see a window like the one on the right.

As you work, write your thoughts on CogSketch and the Design coach in this box. This feedback will be used to improve program. The sketch notes are saved whenever you save the sketch associated with them.

- c. **Draw a sketch** to explain the design to the Design Coach. Include two or more subsketches, as shown in the tutorial, to show how the device operates. **Use flat, 2-D views** (no isometric/perspective). **Save often.** If you are unsure what to do at any point, hit the “Feedback” button on the Design Coach.
- d. Hit the “Tell” button and **write a 3-5 sentence description** of the design in the “overall description” box at the top of the Tell Window.



- e. Using the Tell Window, **build 1-4 structured sentences**, as shown in the tutorial, to tell CogSketch what the device does. (CogSketch cannot understand what is written in the overall description box.) Then, use the Feedback feature to get feedback about the design. You can edit your sketch or sentences to address the Coach’s feedback.
- Remember to **write your thoughts** about the program in the **sketch notes window** as you work
 - Be sure to save** the sketch one last time before closing it.

2. Consider possible refinements to the design. Use CogSketch to draw and describe one improvement, and get feedback for it.

- f. In a new **design sketch file**, Repeat steps a-e from part 1, sketching your refined variation of the device. **Remember to save often!**
- You may copy and paste parts from your first sketch to save time.
 - Be sure to describe the purpose of the refinements in the “overall description” box at the top of the Tell window.

Your work should consist of 2 sketch files, one for each part. Save your sketches to the desktop of the tablet you are using.

CogSketch Activity – DTC-1, Fall 2012

The goal of this activity is to gather input from users in order to improve the CogSketch tool.

0. Take the CogSketch Tutorials

- If this is your first time using CogSketch or you want a refresher, open the “Help” menu-> “Tutorials”-> “Design Coach Basic Tutorial” (make sure the name is correct) to learn how it works.
- When you are done, do the “Design Coach Rotation Tutorial”

1. Create a sketch of the following design using CogSketch

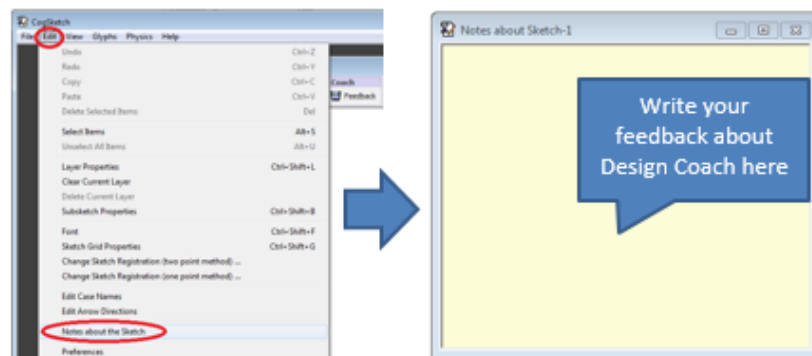
An EDC team designed the Gamma Handle to help people with hemiplegia to operate their car with only one hand. The Gamma Handle is an L-shaped, independently rotating handle that mimics the feel of a steering wheel. The design is shown below.

EDC wq10, sec 1, team 1

Elizabeth Appelt, Frank Cummins, Anvesh Tanuku, & Yee Wai



- Under the “File” menu select “New Design Sketch” (*not* “New Sketch”) to get started.
- Open the sketch notes window, where you will write your user feedback:
 - On the menu bar click edit and then “Notes about the sketch”



- You will see a window like the one on the right. As you work, write your thoughts on CogSketch and the Design coach in this box. This feedback will be used to improve program. The sketch notes are saved whenever you save the sketch associated with them.
- Draw a sketch to explain the design to the Design Coach.

- i. Include two sets of subsketches, as shown in the tutorial, to demonstrate 1) how the handle is used to turn the wheel and 2) how the handle is attached to the wheel.
 - ii. **Each subsketch should use a single orthographic projection.** For example, you might draw a top-down view to depict the rotation in the first set of subsketches, and then a side view to depict the attachment to the wheel.
 - iii. If you are unsure what to do at any point, hit the “Feedback” button on the Design Coach. **Save often.**
- d. Hit the “Tell” button and **write a 3-5 sentence description** of the design in the “overall description” box at the top of the Tell Window.



This is the “overall description” box. (Part d)

Before clicking “Get Feedback”, use the “New Sentence” button to add 1-4 structured sentences for Design Coach to read. (Part e)

- e. Using the Tell Window, **build 1-4 structured sentences**, as shown in the tutorial, to tell CogSketch what the device does. (CogSketch cannot understand what is written in the overall description box.) Then, use the Feedback feature to get feedback about the design. You can edit your sketch or sentences to address the Coach’s feedback.
 - i. Remember to **write your thoughts** about the program in the **sketch notes window** as you work
 - ii. **Be sure to save** the sketch one last time before closing it.

2. Consider possible refinements to the design. Use CogSketch to draw and describe one improvement, and get feedback for it.

In a new **design sketch file**, Repeat steps a-e from part 1, sketching your refined variation of the device. **Remember to save often!**

- i. You may copy and paste glyphs (but not subsketches) from your first sketch to save time.
- ii. Be sure to describe the purpose of the refinements in the “overall description” box at the top of the Tell window. If the structured sentence system is missing the words you’d want to use, that is ok as long as you include them in the “overall description” box.
- iii. **Don’t forget to write your feedback about the program in the “Notes about the sketch” box as you go.** This is your primary goal as a test user!

Your submission should consist of 2 sketch files, one for each part. Email your sketches to your instructor.

Logistics: (estimate 45-90 minutes, depending on your comfort with the software)

1. Schedule a visit to our lab in the Ford building (3-310) to use our **tablets**
 - a. **Email <researcher email> to schedule lab hours (24 hours in advance please!)**
2. Use the DTC lab machines
3. Download it yourself (windows only)
<http://www.grg.northwestern.edu/software/cogsketch/index.html>

For any CogSketch questions, please contact Jon <researcher email> (not your instructors)

CogSketch Activity – DTC-1, Fall 2013

The goal of this activity is to gather input from users in order to improve the CogSketch tool.

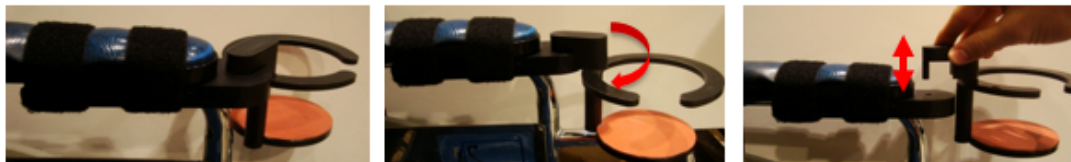
0. Take the CogSketch Design Coach Tutorials

- If this is your first time using CogSketch or you want a refresher, click “Design Coach Basic Tutorial” on the starting screen to learn how it works.
- When you are done, continue on to the “Design Coach Advanced Tutorial”

1. Create a sketch of the following design using CogSketch Design Coach:

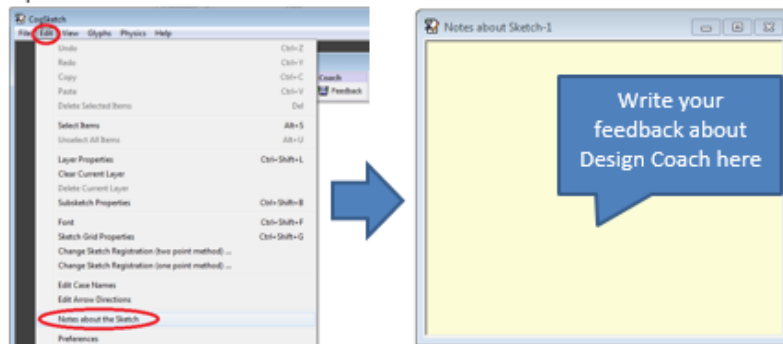
A team designed the Under Armrest Rotating Cup Holder for their client at RIC. The design is shown below.

WQ11, Section 5, Team 3: Branden Alegbeleye, Angela Jiang, Jacob Kelter, Patrick Weitzel



- Easy to attach using Velcro
- Doesn't take up user space
- Cup holder rotates giving user flexibility in placement
- Cup holder slides off for quick attachment/removal

- On the CogSketch home screen, click “New Design Sketch” on the starting screen to create an empty sketch. If you have other sketches open, minimize or close them and the start screen will reappear.
- Open the **sketch notes window**, where you will write your user feedback:
 - Open the menu bar click edit and then “Notes about the sketch”



No stick figures!
Orthographic
drawings only!

- You will see a window like the one on the right.

As you work, write your thoughts on CogSketch and the Design coach in this box. This feedback will be used to improve program. The sketch notes are saved whenever you save the sketchfile associated with them.

- Draw a **sketch** to explain the design to the Design Coach.



- Include at least **two sets of subsketches**, as shown in the tutorial, to demonstrate 1) how cup holder rotates and 2) how the cup holder may be removed from the base of the device and how the base attaches to the chair.

- ii. **Each subsketch should use a *single* orthographic projection.** For example, you might draw top-down views to depict the rotation in the first set of subsketches, and then draw side views to depict the attachment/detachment of the cupholder parts. Do not draw stick figures and do not use isometric views.
 - iii. If you are unsure what to do at any point, hit the “Feedback” button on the Design Coach. **Save often, especially before you hit “Feedback”**
- d. Hit the “Tell” button and **write a 3-5 sentence description** of the design in the “overall description” box at the top of the Tell Window.



This is the “overall description” box. (Part d)

Before clicking “Get Feedback”, use the “New Sentence” button to add 1-4 structured sentences for Design Coach to read. (Part e)

- e. Using the Tell Window, **build 1-4 sentences**, as shown in the tutorial, to tell CogSketch what the device does. (CogSketch cannot understand what is written in the overall description box.) Then, use the Feedback feature to get feedback about the design. You can edit your sketch or sentences to address the Coach’s feedback.
 - i. Remember to **write your thoughts** about the program in the **sketch notes window** as you work
 - ii. **Be sure to save** the sketch one last time before closing it.

2. Consider possible refinements to the design. Use CogSketch to draw and describe one improvement, and get feedback for it.

In a new **design sketch file**, Repeat steps a-e from part 1, sketching your refined variation of the device. **Remember to save often!**

- i. To save time, you may use a copy of your first sketch file as a starting point, or copy and paste glyphs from your first file’s subsketches. Unfortunately copy subsketches from one sketch file to another is not supported at this time.
- ii. Be sure to describe the purpose of the refinements in the “overall description” box at the top of the Tell window. If the structured sentence system is missing the words you’d want to use, that is ok as long as you include them in the “overall description” box.
- iii. **Don’t forget to write your feedback about the program** in the “Notes about the sketch” box as you go. This is your primary goal as a test user!

Your submission should consist of 2 sketch files, one for each part. Email your sketches to your instructor.

Logistics: (estimate 45-90 minutes, depending on your comfort with the software)

1. Schedule a visit to our lab in the Ford building (3-310) to use our **tablets**
 - a. **Email <researcher email> to schedule lab hours (24 hours in advance please!)**
2. Use the DTC lab machines
3. Download it yourself (windows only)
<http://www.grg.northwestern.edu/software/cogsketch/index.html>


For any CogSketch questions, please contact <researcher email> (not your instructors)

Appendix D

Sketching Anxiety Measure Questionnaire

The following is a reproduction of our sketching anxiety measure as it appeared on the web in the fall quarter of 2013. Questions to part 2 (pictured here) were identical to those of part 1).

DTC 1 Fall 2013 Self-Assessment Part II



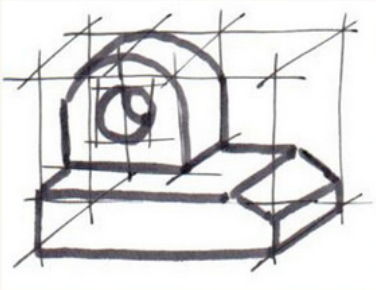
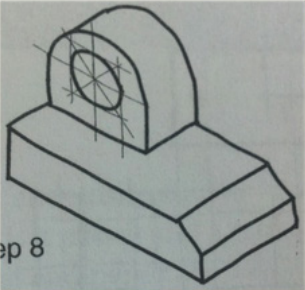
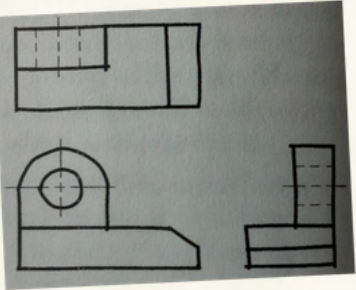
This survey data will be anonymized, and any individual information will NOT be shared with your instructor.
Some of the items in the questionnaire refer to things and experiences that may cause tension, apprehension, or anxiety. For each item, mark the response that describes how much you would be made anxious by it. Work quickly, but be sure to think about each item.

How nervous would the following situations make you feel, right now?

	Not at all	A little	A fair amount	Much	Very much
Being asked to sketch an idea you have just suggested in brainstorming	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Being asked to sketch to communicate your design to your instructor	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Getting an assignment asking you to hand draw an engineering design to scale	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Joining a group which put you in charge of hand drawing the design of their engineering concept	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Being asked to hand draw an oblique view of a model (see below)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Being asked to hand draw an isometric view of a model (see below)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Being asked to hand draw an orthographic projection of a model (see below)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

(Survey continues on next page; it has been cut here to fit.)

Examples of hand drawn views/projections of an object

	Oblique	Isometric	Orthographic
			

How skilled do you feel you are at the following tasks, right now?

	Not at all	A little	A fair amount	Much	Very much
Hand drawing an engineering design	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Hand drawing a picture of a friend's face	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Hand drawing an inanimate object	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Hand drawing a diagram to describe a process	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>