A simple and efficient markup tool to generate drawing-based online assessments

Nicolas Nytko, University of Illinois at Urbana - Champaign

Nicolas Nytko is a M.S. student in the department of Computer Science at the University of Illinois at Urbana-Champaign. His current research interests are in computer science education and scientific computing.

Prof. Matthew West, University of Illinois at Urbana - Champaign

Matthew West is an Associate Professor in the Department of Mechanical Science and Engineering at the University of Illinois at Urbana-Champaign. Prior to joining Illinois he was on the faculties of the Department of Aeronautics and Astronautics at Stanford University and the Department of Mathematics at the University of California, Davis. Prof. West holds a Ph.D. in Control and Dynamical Systems from the California Institute of Technology and a B.Sc. in Pure and Applied Mathematics from the University of Western Australia. His research is in the field of scientific computing and numerical analysis, where he works on computational algorithms for simulating complex stochastic systems such as atmospheric aerosols and feedback control. Prof. West is the recipient of the NSF CAREER award and is a University of Illinois Distinguished Teacher-Scholar and College of Engineering Education Innovation Fellow.

Prof. Mariana Silva, University of Illinois at Urbana-Champaign

Mariana Silva is a Teaching Assistant Professor in Computer Science at the University of Illinois at Urbana-Champaign. She has been involved in large-scale teaching innovation activities, such as the development of online course content and assessments for the mechanics course sequence in the Mechanical Science and Engineering Department and the numerical methods class in Computer Science. Silva is currently involved in two educational projects involving the development of online assessments for computer-based testing and creation of collaborative programming activities for computer science classes. She is also involved in a project that aims to create a software that facilitates collaborative problem-solving activities in classrooms, through which both the instructors and students learn more about collaboration skills. Silva is very passionate about teaching and improving the classroom experience for both students and instructors. She has been included in the List of Teachers Ranked as Excellent five times and has received the Engineering Council Outstanding Advisor Award every year since 2014.

A Simple and Efficient Markup Tool to Generate Drawing-Based Online Assessments

Abstract

Due to an increase in student enrollment in engineering programs, many instructors are now adopting automated computer-based systems to deliver homework and exams to students. Commercial and free online learning systems such as Gradescope, Pearson MasteringEngineering and PraireLearn give instructors the ability to import or create auto-graded questions involving a mix of multiple-choice, multiple-select, numerical, and symbolic input. However, the ability to auto-grade questions that involve graphing or sketching is still very limited. This constraint has great impact in introductory engineering classes where the ability to hand-draw diagrams and graphs is an important learning objective. For example, drawing free-body diagrams in Statics, bending-moment diagrams in Strength of Materials, and circuit diagrams in Electronics. In this study, we present an online tool that uses a simple HTML markup language to create automated drawing-based questions, allowing students to draw diagrams, graphs and design solutions on the computer that are instantly auto-graded by the computer. While solving homework questions, students receive immediate feedback about their drawing, and can practice this skill until they achieve mastery. Instructors can also use the drawing tool to generate randomized drawing questions for computer-based exams and homework. A key advantage of this new tool over previous work is that the question author does not need to write any explicit programming code.

Introduction

For many engineering disciplines, it is important for students to be able to visualize concepts graphically. For example, students are expected to model complicated systems using free-body diagrams (FBD) in mechanics courses, or create state and logic diagrams in computer science courses. To best master these skills, students should be able to attempt these questions multiple times, with a variety of different forms to ensure a breadth of knowledge in the topic.[1] It is also essential that students receive prompt and meaningful feedback on their submissions, so that they may improve and learn from their mistakes. Research has confirmed the importance of building drawing skills in undergraduate mechanics courses. Shryock and Haglund[2] stress that providing ample practice to draw free-body diagrams helps students understand key concepts in physics and mechanics, and can clear up conceptual misconceptions they may have. Serral et al.[3] have investigated the importance of instant feedback and individualized feedback to students, where

giving such feedback to students can "significantly improve the quality and speed of learning"[3].

With growing enrollment of students in engineering courses, the grading of these problems becomes an increased strain on instructional staff, and hence an alternative method is needed to assess student progress. The current trend of moving student homeworks to online platforms, and indeed even quizzes and tests to similar computerized platforms[4], has helped professors and teaching staff generate automatically graded questions that assess student knowledge using multiple-choice, multiple-select, or numerical inputs. Existing proprietary tools such as McGraw-Hill Connect[5] and Pearson MasteringEngineering[6] have been improving their functionality in recent years to become more user friendly for students and provide learner feedback. These commercial systems, however, often only define one "correct" solution and due to their nature do not give access to instructors to define their own questions.

To circumvent the limitation of commercial tools, some authors have proposed different approaches to allow students to draw FBD using online platforms. Mechanix[7] and Newton's Pen[8] allow free-hand drawing of the diagrams. In *Newton's Pen*, students use a computerized pen and work through a special worksheet. *Mechanix* is a desktop application that must be installed on the computer. In both of these applications, students are required to do some pre-installation or use special resources provided by the instructor. Roselli et al.[9] present a web-based platform for creating and grading mechanics free body diagrams. Their *FBD Assistant* is implemented using the Adobe (then Macromedia) Flash platform and thus prevents the use of mobile devices. The drawing tool presented by Silva et al.[10] uses HTML canvas and provides instant feedback on student answers. However the question implementation requires extensive JavaScript programming, involves substantial boilerplate code for each question, and necessitates some programming background for question authors.

To overcome the above limitations, we implemented a tool for drawing-based questions using the open-source learning management system (LMS) PrairieLearn[11]. Using our tool, question authors are able to create drawing assessments without any explicit programming code. In addition, students are able to use the tool from any browser (including mobile devices), without the need for installation or the use of any special device, such as digital pens or tablets. The proposed drawing tool has been used to generate homework and exam questions in four large courses (Statics, Solid Mechanics, Dynamics, and Numerical Methods) at a Midwestern University, and it can be easily extended to other courses. This paper will focus on three aspects of the drawing tool: (1) description of the markup tags that create mechanics objects, such as pins, rods, pulleys, forces and moments; (2) how these objects can be created using random parameters, so that each student receives a different version of the same question; and (3) the algorithm used to grade student drawings and the type of feedback provided.

This paper is organized as follows: we first provide background information about the PrairieLearn LMS and describe the design requirements of the drawing tool. Then we present how a drawing question can be created using HTML to display from a list of built-in objects, and how they can be randomized to prevent cheating and promote skill mastery. We describe the built-in grading algorithm, and conclude with examples and discussion.

```
<p> What is $2 + 2$? </p>
<pl-number-input correct-answer="4" answers-name="answer"
    label="$2+2=$"></pl-number-input>
```

Figure 1: A code snippet and corresponding visual output for a numerical input field, without parameter randomization.

Background information

To describe the process of creating a question with the drawing tool, we first explain how the PrairieLearn LMS generates questions, such as fill-in the blank or multiple-choice. PrairieLearn is an online problem posing system that permits the specification of *problem generators* that are capable of generating randomly parameterized problem instances. A *problem generator* is defined by the following files: `info.json`, `question.html`, and the `server.py`.

`info.json` is a required file containing miscellaneous metadata about the question, including things such as a unique identifier (UUID), title, topic, and more. `server.py` is an optional file containing specialized logic to be executed server side, which, among other things, can be used to generate random parameters for question instances. The use of this file to randomize questions will be detailed in a following section.

The main file we will focus on is the required `question.html`, which holds the structure of the question itself and is used to render the web page that is seen by the student. It is important to note that this file is not written in the traditional HTML markup language, but using an extended domain-specific language that has elements particular to PrairieLearn.

The example in Figure 1 shows how one can use a built-in element (`pl-number-input`) to quickly create a question expecting some numerical input. One can get away with using these included elements to create large, complex questions (not shown here). Note the ability to render mathematical formulae by enclosing an equation in LaTeX format with dollar signs ($).

Design requirements

The drawing tool was implemented based on the following design requirements:

(a) question generators are specified by markup (and optional code), and not as a graphical tool,

(b) question generators allow for randomization,

(c) the grading algorithm supports multiple correct answers,

(d) questions are easily accessible to students with no special tools.

(a) Question generator specified entirely as markup (and optional code)

In PrairieLearn, question generators are specified entirely as HTML markup (and optional Python code for randomization). While this gives the obvious downside that there is a slight learning curve to creating questions, especially to those not familiar with HTML or Python, there are substantial benefits from using a text format:

- Rapid and precise development: question authors do not have to use a graphical interface and align elements visually, but instead can precisely specify coordinates and angles for objects.
- Questions and question elements (parts that make up a question) can be easily duplicated to create new questions.
- Ease of use with existing versioning control systems. Multiple people can work on the same problems and check-in changes and handle merge conflicts.

(b) Allow built-in randomization

For computerized testing where students take exams asynchronously, some sort of randomization is needed to generate different question variants and reduce cheating. Chen, West, and Zilles[12] reported that having both randomized questions and larger (at least 4) pools of questions to draw from is effective in curbing student cheating. The drawing tool described in this paper takes advantage of the randomization features from PrairieLearn.

(c) Allow multiple correct answers

The drawing tool should be robust enough to handle multiple correct answers and correctly grade any student input according to those answers. For example, when completing free-body diagrams, a force vector can be assumed as correct based only on its direction, but both orientations are accepted. Moreover, some students may prefer to place a force vector with the "tail" at the origin, and others with the "head" at the origin. There are also cases where multiple answers are truly correct, or some others where an answer can be optional.

(d) Easily accessible by students

Ideally the drawing tool should be able to run on multiple operating systems and platforms to allow the best accessibility for students. This can be best realized by developing a platform using web technologies so that students can use any device with a (modern) web browser.

Markup for Drawing Questions

In the spirit of using HTML elements to create questions, we decided to maintain consistency with the rest of the PrairieLearn format and have a similar workflow of creating drawing based questions. A drawing "element" (an empty canvas) is thus initialized by a `pl-drawing` tag, and then made up of three parts, defined by child tags:

- `pl-drawing-initial`, containing the "initial" state of the drawing canvas. This is used to display initial drawings that are given to students.

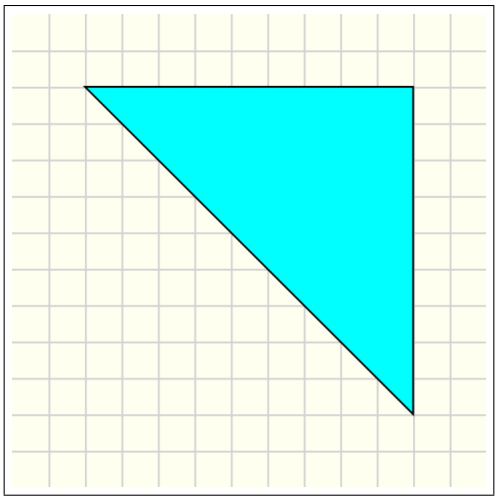


Figure 2: Using the drawing tool to display a triangle

```
<pl-drawing width="260" height="260">
  <pl-drawing-initial>
    <pl-triangle x1="40" y1="40" x2="220" y2="220" x3="220" y3="40"
        color="cyan"></pl-triangle>
  </pl-drawing-initial>
</pl-drawing>
```

Figure 3: HTML markup to generate the triangle in Figure 2

- `pl-controls`, containing buttons and controls available for the student to draw on the canvas with.
- `pl-drawing-answer`, containing a correct answer for this given question.

In order to populate the answer and initial state sections, one can pick from a list of specially crafted *drawing elements*. A single drawing element represents one object or shape that can be drawn on the canvas. At the moment, this list of elements[1] is specially tailored towards generating diagrams for mechanics courses, and includes things such as lines, points, and rectangles; more complex objects can be placed as well like springs, pulleys, and force vectors.

A single drawing or answer is composed of many of these drawing elements. Each element has its own HTML attributes such as the position, width, etc. Figure 2 illustrates an example where a triangle is placed at coordinates $\boldsymbol{x}_1 = (40, 40)$, $\boldsymbol{x}_2 = (220, 220)$, and $\boldsymbol{x}_3 = (220, 40)$. The canvas is initialized with dimensions (260×260) and the background grid uses the default square of size 20. Note that the drawing canvas has origin at the top-left corner, as typically used in computer graphics applications. The HTML markup that generates the triangle is shown in Figure 3.

The `pl-drawing-initial` is useful to generate static figures and drawings. To allow student interactions with the canvas, an additional child in the form of a `pl-controls` tag can be added to the root drawing element. These control buttons allow students to place objects inside the

[1] https://prairielearn.readthedocs.io/en/latest/pl-drawing/: list of available drawing elements

canvas to construct their own drawings. For example, one can use

```
<pl-drawing-button type="object_name"></pl-drawing>
```

to add an object with name `object_name` inside the canvas. In case these interactions require grading and feedback, the expected interactions (answers) should be defined inside the `pl-drawing-answer` child element, which has the same syntax as `pl-drawing-initial`. In the results section, we show more complex examples that combine multiple drawing elements, and also illustrate the use of control buttons and auto-grading features.

Question Randomization

Randomization of parameters is important to allow students to retry variants of the same questions, so they can get many practice opportunities to help mastering skills, and is also important to reduce cheating during exams. The ideas introduced in the previous section can be extended to allow question generators to create different variants of questions where objects can vary their position, angle, size, etc. In the `server.py` file, question authors can define the parameters used by the question generators inside the function `generate()`, which is called automatically by the PrairieLearn server implementation. The `generate()` function has the form:

```
def generate(data):
    data["params"]["my_parameter"] = random.randint(0, 11)
```

where the question author can add any relevant code and store parameters in the dictionary `data["params"]`. Anything added to this dictionary will be "visible" from the question HTML file.

For HTML templating, PrairieLearn uses the Mustache[13] system to inject data and values into HTML pages. After some value is placed into the `data["params"]` dictionary in the server code, it can be used by an HTML question by using the appropriate Mustache syntax. In the example above, the parameter `my_parameter` can be accessed in the corresponding question HTML file with {{ `params.my_parameter` }}. Thus, the HTML markup can be combined with the templating engine and some server-side code to generate randomized question variants. A drawing example that uses randomized parameters is shown in Figure 4. A square object is placed inside the canvas with center at position (x, y), where the coordinates x and y are generated at random.

Question Grading

Perhaps the most important design requirement is the robust grader that can handle more complex problems where multiple or optional solutions exist. The built-in grader compares the expected position of the object, defined by the question author, with the actual submitted position of the object inserted by the user. This is accomplished by defining bounding boxes around the reference position of the correct object. Submitted (user-placed) objects are checked against the reference answers by means of point-rectangle intersections, and this method provides a general

question.html:

```
<pl-drawing width="200" height="200">
  <pl-drawing-initial>
    <pl-rectangle x1="{{params.x}}" y1="{{params.y}}" width="40" height="40">
    </pl-rectangle>
  </pl-drawing-initial>
</pl-drawing>
```

server.py:

```
import random

def generate(data):
    data["params"]["x"] = random.randint(20, 180)
    data["params"]["y"] = random.randint(20, 180)
```

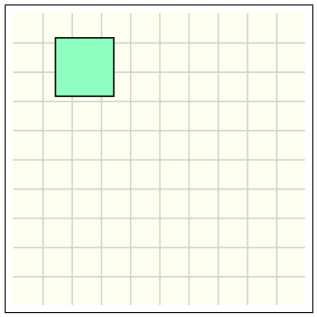


Figure 4: An example of a question generator that uses randomized parameters defined in server.py.

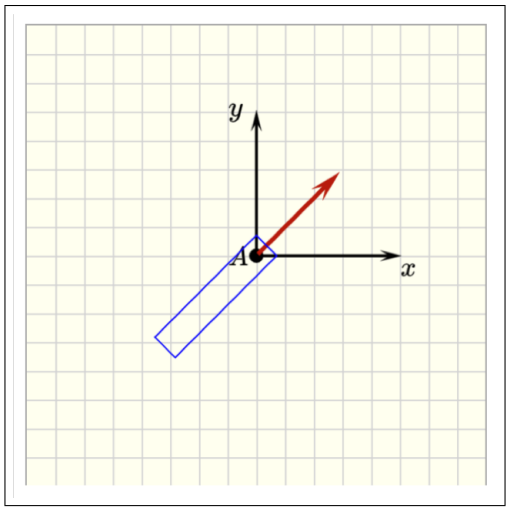


Figure 5: A vector and its corresponding bounding box for grading purposes.

implementation that works well in most cases. When question authors need more flexibility with the grading algorithm, they can write their own grading code in the `server.py` file, which overrides the built-in grading implementation described above. Figure 5 shows a vector and its corresponding bounding box. Note the generous length of the box to allow the placing of the vector with either the head at the origin or the base at the origin. We will introduce more features of the built-in grader in the results section.

Example questions from courses

The drawing tool described in this paper was integrated into PrairieLearn in Summer 2019, and instructors from two different courses (Statics and Strength of Materials) used the tool during the Fall 2019 semester. Here we show examples from these courses.

Find the centroids

In the example illustrated in Figure 6, students are asked to mark the centroid of a triangle and a rectangle by inserting "point" objects inside the canvas. The question gives three options of control buttons: insert a point object (small circle dot), delete any object in the canvas (X sign) and a "help line" that can be used as support grid lines for better visualization. The position of the objects placed on the canvas, in this example the points, will be graded against the expected position defined by the correct answer within the tolerance given by the bounding boxes.

Free-body diagrams

In this next example, students are asked to complete the free-body diagram (FBD) of a cantilever beam, by entering the objects that represent all the applied loads and support conditions, as shown in Figure 7. The control buttons allow students to enter force vectors, which can be rotated and placed at any position in the canvas, and moment vectors in both orientations. Students can also delete any placed object using the "delete" button (here represented with the X sign).

In many applications, creating an FBD is the first step in the process of completing the given

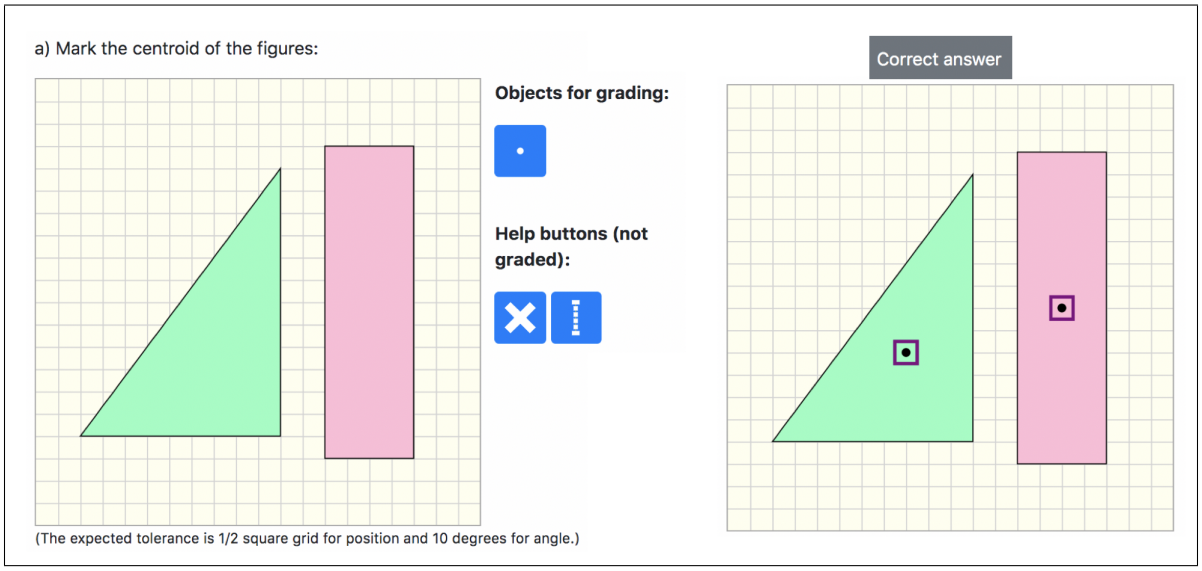


Figure 6: Example question where students are expected to mark the centroid of the figures. The right diagram shows the expected correct answers and their corresponding tolerance bounding boxes.

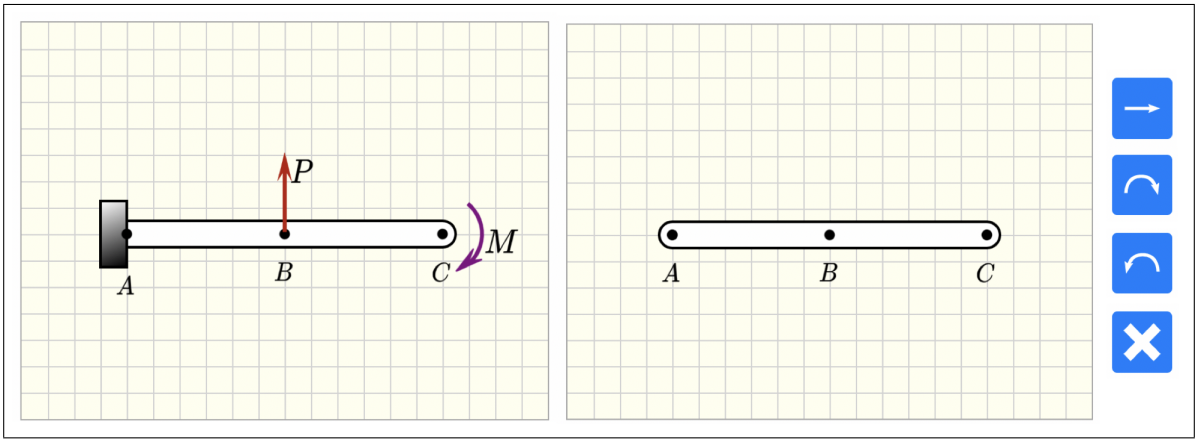


Figure 7: Cantilever beam example where students need to complete the free-body diagram.

problem. Consequently, orientation and magnitude of forces are often not known at this step. Questions that are given to students in multiple-choice format usually represent the "correct" answer, which a student will only have available after they complete the problem. Using this drawing tool, instructors are able to make assessments of FBD skills only, and can separate this step from the actual solution of equilibrium equations. Figure 8 shows the HTML markup definition of the correct answer defined by the question author, and the corresponding rendering of objects and bounding boxes. The labels are further explained below:

(1) Moment at A: this is a required object. The submitted object must be placed inside the bounding box. In this problem, both clockwise and counter-clockwise moments are accepted as correct. This is achieved using the attribute `disregard-sense="true"`.

(2) A vertical force at A: this is a required object. The tail of the arrow should be inside the bounding box (this is the default setting that can also be changed by another attribute). In this problem, the orientation of the force is not relevant. This is achieved using the attribute `disregard-sense="true"`.

(3) A horizontal force at A: this is an optional object, that can be placed or not within the bounding box, without affecting the question score. This is achieved using the attribute `optional-grading="true"`.

(4) A vertical force at B: this is a required object. The tail of the arrow should be inside the bounding box. Since this is a given applied force with known orientation, the correct answer checks for the correct orientation.

(5) A moment at C: this is a required object. The center of the submitted object must be inside the bounding box. Since this is a given applied moment, the correct clockwise orientation is enforced in the correct answer.

Figure 9 shows two possible student submissions for the FBD. Note that the correct submission (figure on the left) does not include the horizontal force at A, and the other two required objects at A are placed with the opposite orientation of the ones defined by the question author. However, the question is marked as 100% correct. The partially correct submission (figure on the right) is missing the moment vector at A. Since this problem expects 4 required objects, each one contributes for 25% of the grade, and hence this incorrect submission gets a partial score of 75%.

Shear and bending moment diagrams

To allow the drawing of shear and bending moment diagrams, the drawing tool has two special objects: a controlled straight line and a controlled curved line. Figure 10 shows how these two objects can be used to create graphs. Once controlled lines are placed in the canvas, they can be moved to any location by adjusting the position of the *end points*. Curved lines have an additional *control point* that modifies the shape of the curve (they are implemented as a Bézier curve).

In the next example, students need to sketch a shear diagram for a simple supported beam. Figure 11 illustrates the drawing canvas where students complete their solution using the controlled lines. The correct answer is defined by the coordinates of the end points with respect to the origin of the diagram (and not the origin of the canvas), which is given as `V_origin =`

```
<pl-drawing-answer draw-error-box="true" >
(1)<pl-arc-vector x1={{params.xA}} y1={{params.yA}}  disregard-sense="true"></pl-arc-vector>
(2)<pl-vector     x1={{params.xA}} y1={{params.yA}}  angle="90" disregard-sense="true" ></pl-vector>
(3)<pl-vector     x1={{params.xA}} y1={{params.yA}}  angle="0"  disregard-sense="true" optional-grading="true"></pl-vector>
(4)<pl-vector     x1={{params.xB}} y1={{params.yB}}  angle="-90" ></pl-vector>
(5)<pl-arc-vector x1={{params.xC}} y1={{params.yC}}  ></pl-arc-vector>
</pl-drawing-answer>
```

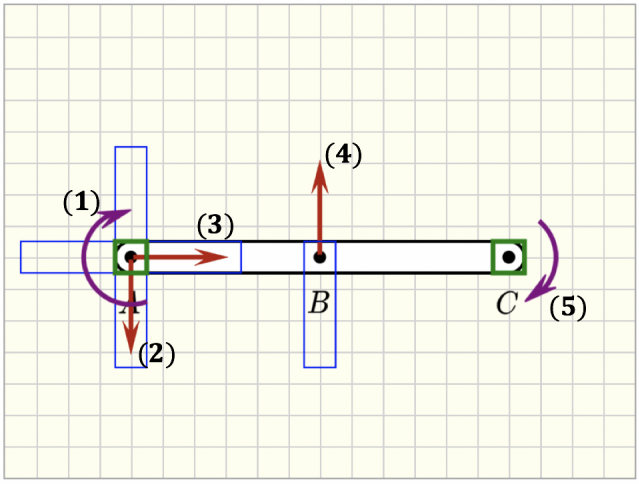
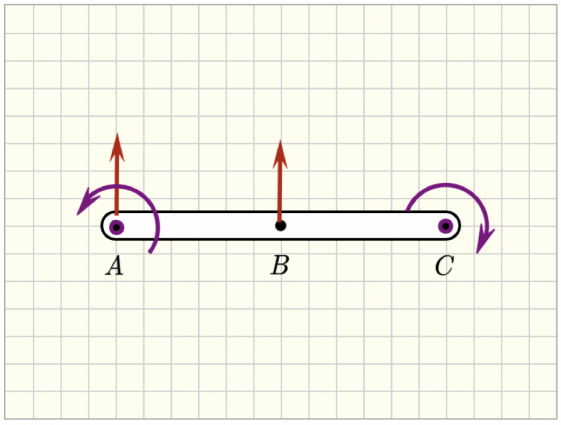
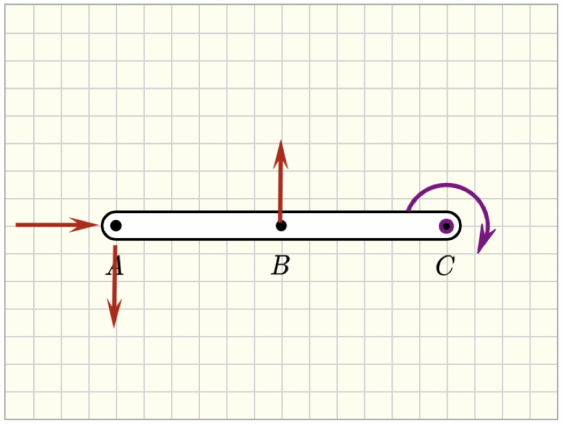


Figure 8: Correct answer defined by the question author in HTML markup, and the corresponding rendering. The labels (1)–(5) are further explained in the body of the text.



correct: 100%



partially correct: 75%

Figure 9: Examples of submitted answers for the question from Figure 7, illustrating the ability to include optional objects and vectors where the orientation is not graded.

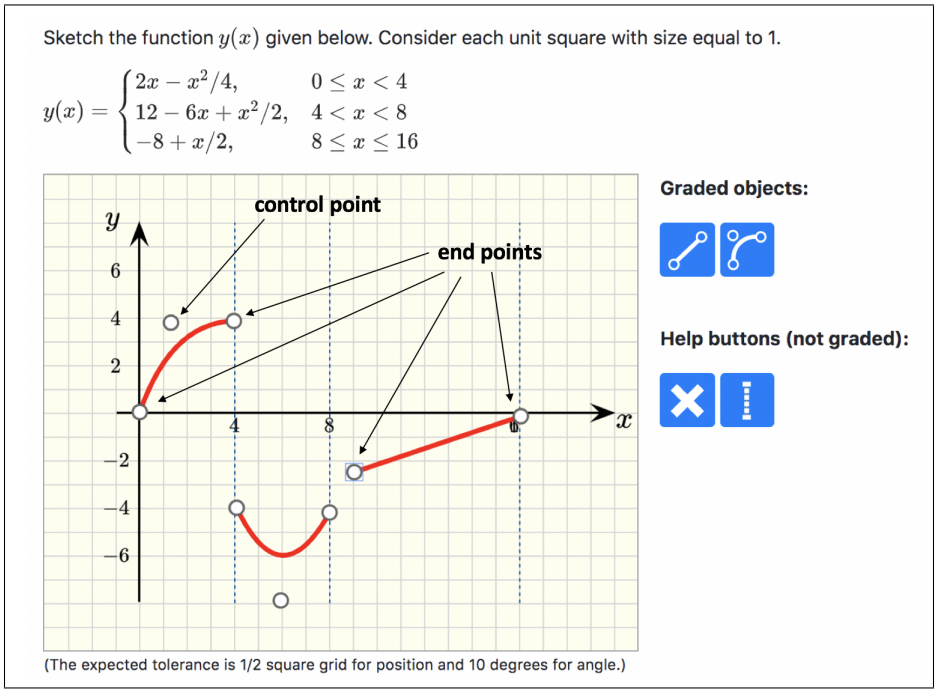


Figure 10: Simple sketching problem illustrating the use of controlled lines. Both straight and curved lines can assume different locations in the canvas by moving the position of their end points. The curved lines can have their shape modified by moving the position of the control point.

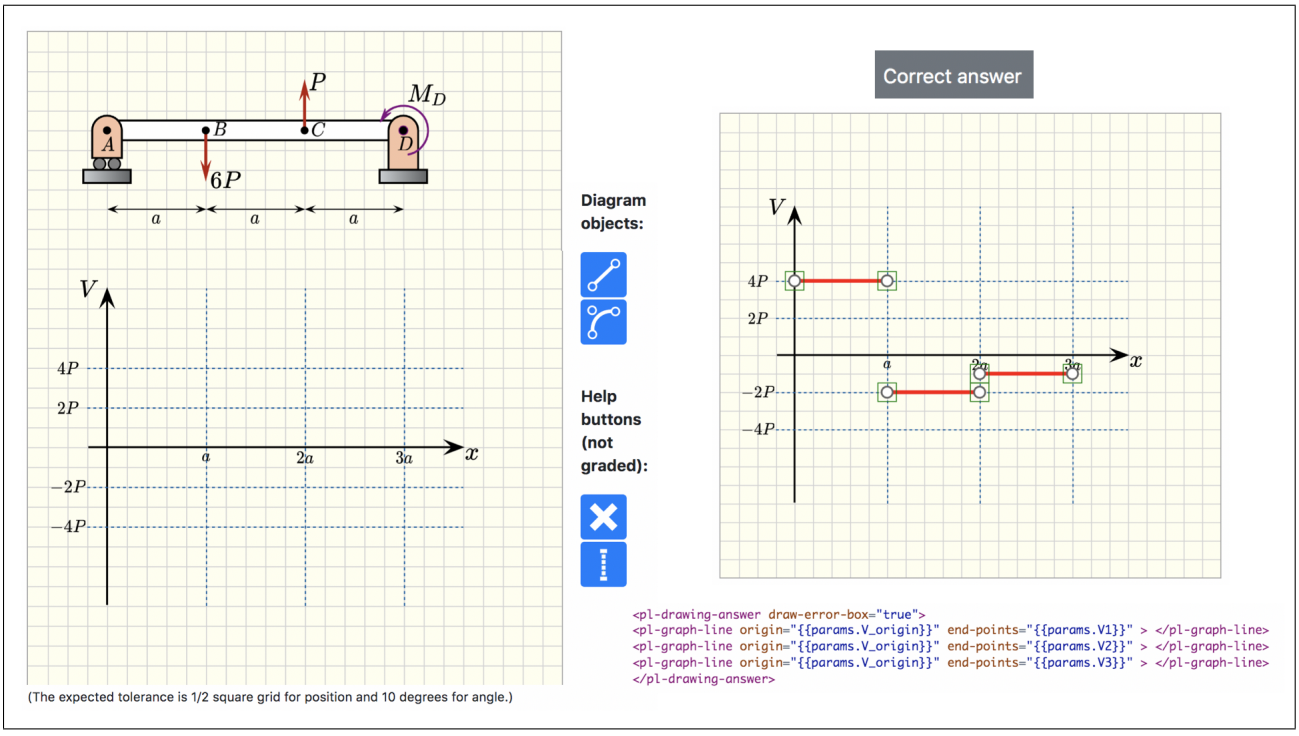


Figure 11: On the left, the drawing canvas where students need to complete the shear diagram corresponding to the simple supported beam. On the right, the correct answer defined by the question author, and the HTML markup that generates the correct answer.

server.py:

```
import random

def generate(data):
    data["params"]["V_origin"] = '{"x":80,"y":420}'
    data["params"]["V1"] = '[{"x":0,"y":80},{"x":100,"y":80}]'
    data["params"]["V2"] = '[{"x":100,"y":-40},{"x":200,"y":-40}]'
    data["params"]["V3"] = '[{"x":200,"y":-20},{"x":300,"y":-20}]'
```

Figure 12: Python code to generate the correct positions for the shear diagram lines illustrated in Figure 11

`(80,420)` assuming the default square size of 20. Consequently, line A-B has end points `(0,80)` and `(100,80)`, line B-C has end points `(100,-40)` and `(200,-40)` and line C-D has end points `(200,-20)` and `(300,-20)`. To add a controlled straight line, the HTML markup has the following syntax (example for line A-B):

```
<pl-graph-line origin='"x":80,"y":420'
end-points='["x":0,"y":80,"x":100,"y":80]'> </pl-graph-line>
```

Figure 12 shows the Python code that defines the end points as dictionaries for each expected line from the example of Figure 11. Note that this question can be easily modified to have randomized values for the end points, given different initial conditions for the applied loads (magnitude, orientation and position).

Results from student usage

In Fall 2019, students in a Solid Mechanics course were asked to solve the shear diagram question from Figure 11 in one of their exams. A total of 143 students completed the question. One of the big advantages of this drawing tool is that it allows students to try the question again for partial credit if they get it marked as incorrect after the first attempt. About 70% of the students had one submitted attempt, 14% had 2 submitted attempts, and the remaining of the students had 3 or more submitted attempts. After the first attempt, the average score for this question was 76%. At the end of all submissions, the final average jumped to 88%. Anecdotal feedback indicates that students appreciate being able to get partial credit in these type of questions, when compared with other online assessments where similar content was assessed using multiple-choice questions without partial credit.

Discussion and Conclusion

In this paper a platform for developing drawing-based questions is introduced and the necessary steps for developing a question using this tool are described. This is then extended to provide randomization for extra mastery by students, and examples specific to mechanics are presented. The platform has been integrated into the open-source PrairieLearn system[11] and is freely available for use.

This platform makes several advances over previously available tools for automatically assessing student drawing. It is entirely based on markup (HTML) and code (Python), it allows tremendous flexibility in randomization of questions, the grading algorithm supports multiple correct answers, and the system is web-based and accessible to students without any specialized software or hardware. All of these features make it very easy to share content created in the platform and to make it available to students.

While formal feedback from instructors and students was not yet available, initial testing resulted in positive anecdotal feedback from both instructors and students. Instructors and question authors reported relative ease of use in creating questions and noted that question development time was much shorter compared to previous methods. Students liked the ability to receive partial credit in drawing-based questions built on the platform, as compared to multiple-choice questions used in previous assessments.

Future work on this platform may include providing a visual way of designing these questions, extending the platform to work for additional types of graphical questions, and adding a system to provide more custom tailored feedback to students who are first learning a concept.

References

[1] M. West, C. Zilles, and G. Herman. Prairielearn: Mastery-based online problem solving with adaptive scoring and recommendations driven by machine learning. *Proceedings of the American Society for Engineering Education (ASEE) 2015 Annual Conference*, 2015. doi: 10.18260/p.24575.

[2] Kristi J. Shryock and John Haglund. Instrument for assessing skills related to free body diagrams in a sophomore engineering mechanics course. In *2017 ASEE Annual Conference & Exposition*, Columbus, Ohio, June 2017. ASEE Conferences. https://peer.asee.org/28542.

[3] Estefania Serral, Jochen De Weerdt, Gayane Sedrakyan, and Monique Snoeck. Automating immediate and personalized feedback taking conceptual modelling education to a next level. In *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*. IEEE, June 2016. doi: 10.1109/rcis.2016.7549293. URL `https://doi.org/10.1109/rcis.2016.7549293`.

[4] C. Zilles, M. West, D. Mussulman, and T. Bretl. Making testing less trying: Lessons learned from operating a computer-based testing facility. *IEEE Frontiers in Education Conference*, 2018.

[5] McGraw-Hill. McGraw-Hill Connect. URL `http://connect.mheducation.com/`.

[6] Pearson. MasteringEngineering. URL `https://www.masteringengineering.com/`.

[7] Olufunmilola Atilola, Cheryl Osterman, Francisco Vides, Erin M. McTigue, Julie S. Linsey, and Tracy Hammond. Mechanix: The development of a sketch recognition truss tutoring system. In *2012 ASEE Annual Conference & Exposition*, San Antonio, Texas, June 2012. ASEE Conferences. https://peer.asee.org/21684.

[8] WeeSan Lee, Ruwanee de Silva, Eric J. Peterson, Robert C. Calfee, and Thomas F. Stahovich. Newton's pen: A pen-based tutoring system for statics. *Computers & Graphics*, 32(5):511–524, October 2008. doi: 10.1016/j.cag.2008.05.009. URL `https://doi.org/10.1016/j.cag.2008.05.009`.

[9] Robert J. Roselli, Larry Howard, Bryan Cinnamon, Sean Brophy, Patrick Norris, Megan Rothney, , and Derek Eggers. Integration of an interactive free body diagram assistant with a courseware authoring package and an experimental learning management system. In *2003 Annual Conference*, Nashville, Tennessee, June 2003. ASEE Conferences. https://peer.asee.org/11837.

[10] M. Silva and M. West. Algorithmic grading strategies for computerized mechanical drawing assessments. *Proceedings of the 124th American Society for Engineering Education Annual Conference and Exposition (ASEE 2017)*, 2017.

[11] University of Illinois at Urbana-Champaign. Prairielearn. URL `https://prairielearn.readthedocs.io`.

[12] B. Chen, M. West, and C. Zilles. How much randomization is needed to deter collaborative cheating on asynchronous exams? *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, 2018. doi: 10.1145/3231644.3231664.

[13] Jan Lehnardt. Mustache (mustache.js). URL `https://github.com/janl/mustache.js`.