



# WPI

## **PMKS+: Recreating a Legacy Application**

*Major Qualifying Project*

Written By:

Praneeth Appikatla (CS)

Griffin Cecil (CS)

Michael Taylor (CS)

Dimitrios Tsiakmakis (CS)

Advisors:

Professor David C. Brown (CS)

Professor Pradeep Radhakrishnan (ME)

April 25, 2019

# Abstract

The goal of this project was to recreate the Planar Mechanism Kinematic Simulator (PMKS), a legacy, open-source web application, on a modern web platform with an enhanced user experience. The conversion included support for multiple browsers and improvements to the graphical user interface. This application was developed using the latest technologies in web development. Through multiple evaluations testing the interface and functions, we were able to create a similar application to PMKS that has an improved user interface experience.

# Acknowledgements

We would like to thank Professor Radhakrishnan and Professor Brown for supporting us through this project. We would also like to thank Dr. Matthew Campbell of Oregon State University for creating the open-source application, PMKS, on which our work is built. Finally, we would like to thank the students of the Mechanical Engineering department of WPI who participated in our user studies and helped us improve the user experience in our application.

# Table of Contents

<b>I.</b>	<b>Abstract .....</b>	<b>1</b>
<b>II.</b>	<b>Acknowledgments.....</b>	<b>2</b>
<b>III.</b>	<b>Table of Contents.....</b>	<b>3</b>
<b>IV.</b>	<b>Authorship.....</b>	<b>6</b>
<b>1.</b>	<b>Introduction.....</b>	<b>7</b>
<b>2.</b>	<b>Research Statement &amp; Goals.....</b>	<b>9</b>
<b>3.</b>	<b>What is PMKS.....</b>	<b>10</b>
<b>4.</b>	<b>Literature Review.....</b>	<b>11</b>
	4.1. Interface Development.....	11
	4.2. Interface Evaluation.....	15
<b>5.</b>	<b>Methodology.....</b>	<b>19</b>
	5.1. Migration Options.....	19
	5.2. Web App Justification.....	19
	5.3. User Interface Evaluation.....	20
	5.4. Interface Design.....	21
	5.5. Linkage Creation Methods.....	21
	5.6. System Design.....	22
	5.7. Simulator Class.....	25
	5.8. Animation.....	25
	5.9. Application Testing.....	25
	5.10. PMKS+ User Evaluations.....	26
<b>6.</b>	<b>Migration Options.....</b>	<b>27</b>
	6.1. Migration Methods.....	27
	6.2. Migration Platforms.....	28
	6.3. Migration Tools.....	29
<b>7.</b>	<b>Web App Justification.....</b>	<b>32</b>
	7.1. Rationale.....	33
	7.2. Results.....	36
<b>8.</b>	<b>User Interface Evaluation.....</b>	<b>37</b>
	8.1. Evaluation Strategy.....	37
	8.2. Evaluation Procedure.....	40
	8.3. Data Analysis.....	41

<b>9.</b>	<b>Interface Design.....</b>	<b>44</b>
9.1.	Preliminary UI Mockups.....	44
9.2.	Iteration 1.....	45
9.3.	Iteration 2.....	46
9.4.	Iteration 3.....	48
9.5.	Iteration 4.....	50
9.6.	Iteration 5.....	53
9.7.	Iteration 6.....	58
9.8.	Iteration 7.....	62
<b>10.</b>	<b>Linkage Creation Methods.....</b>	<b>65</b>
10.1.	Linkage Table.....	65
10.2.	Grid.....	66
10.3.	Final Decision.....	69
10.4.	Context Menu.....	70
<b>11.</b>	<b>System Design.....</b>	<b>72</b>
11.1.	Analysis of Existing Application Design.....	72
11.2.	New System Architecture.....	73
11.3.	Grid Component States.....	75
<b>12.</b>	<b>Simulator Class.....</b>	<b>79</b>
12.1.	Simulation Loop.....	79
12.2.	Joint Traversal Algorithm.....	81
12.3.	Circle Intersection Solver.....	85
<b>13.</b>	<b>Animation.....</b>	<b>87</b>
13.1.	CSS Animations.....	88
13.2.	Custom Typescript Solution.....	89
13.3.	Animation Controls.....	91
<b>14.</b>	<b>Application Testing.....</b>	<b>94</b>
14.1.	Browser Compatibility.....	94
14.2.	Software Testing.....	100
<b>15.</b>	<b>PMKS+ User Evaluations.....</b>	<b>104</b>
15.1.	Individual Evaluation.....	104
15.1.1.	Procedure.....	101
15.1.2.	Target Users.....	101
15.1.3.	Evaluations Script.....	102
15.2.	Final User Evaluations.....	105
15.3.	Differences with PMKS Evaluation.....	105
15.4.	Data Analysis.....	106

<b>16. Conclusions.....</b>	<b>111</b>
<b>17. Discussion.....</b>	<b>113</b>
<b>18. Future Work.....</b>	<b>115</b>
<b>19. References.....</b>	<b>120</b>
<b>Appendix A: PMKS User Evaluation Response Form.....</b>	<b>124</b>
<b>Appendix B: PMKS+ User Evaluation Response Form.....</b>	<b>131</b>
<b>Appendix C: Individual Evaluation Script.....</b>	<b>137</b>
<b>Appendix D: Individual Evaluation Paper Examples.....</b>	<b>140</b>

# Authorship

The following table describes to which sections and chapters of the report each team member contributed. After initial drafts, all team members conducted peer reviews for all the sections.

<b>Chapter</b>	<b>Author(s)</b>
1	Cecil
2	Appikatla, Tsiakmakis
3	Tsiakmakis
4	Appikatla, Cecil, Taylor, Tsiakmakis
5	Appikatla, Cecil, Taylor, Tsiakmakis
6	Taylor
7	Appikatla
8	Cecil
9	Tsiakmakis
10	Taylor
11	Taylor
12	Appikatla, Cecil, Taylor
13	Cecil
14	Taylor
15	Cecil
16	Taylor, Tsiakmakis
17	Cecil, Tsiakmakis
18	Taylor

# Chapter 1

## Introduction

Following global standards is key when developing any website. Not only can they help determine what tools to use to expand website functionalities, but they can also define the design of the interface that is going to improve the overall user experience. By applying globally accepted standards, one can ensure that a website will be compatible with current and future operating systems and browsers.

The Planar Mechanics Kinematic Simulator (PMKS) is a web application that is neither compatible with multiple browsers nor future-proof. PMKS is currently used by the Mechanical Engineering Department at Worcester Polytechnic Institute (WPI) to study the behavior and usage of planar linkages. With Apple's recent update to the Safari 12 browser, support for Netscape Plugin Application Programming Interface (NPAPI) plugins has come to an end (*Safari Technology (2018)*). This includes Microsoft Silverlight, a plugin required to run PMKS. With no support for Mac, running PMKS is a much greater burden for Apple product owners. In order to remain a useful tool that aides student's learning, PMKS must migrate to a more accessible platform. This requires conforming to modern web standards and technology. We have named this new version PMKS+.

The second chapter of this report describes our team's goals and objectives while completing this project. Chapter 3 provides an overview of PMKS. Chapter 4 details the research our team conducted on interface evaluation and development. Chapter 5 describes the methods our team used to create, test, evaluate, and implement PMKS+. Chapters 6 and 7 discuss our research and findings for migrating options to PMKS+. Chapter 8 describes the testing conducted to identify which aspects of the user interface to improve, while chapter 9 discusses the iterative process for developing our user interface. Chapter 10 details the various methods considered for creating linkages in PMKS+. Chapter 11 presents the system architecture and design of PMKS+. Chapter 12 describes the Simulator class that performs calculations on linkages, while Chapter 13 discusses the techniques used for animation in PMKS+. Chapter 14 reports the testing our team performed to ensure that PMKS+ met our compatibility and performance standards. Chapter 15 details the evaluation methods used during and after



development. Chapter 16 concludes the development of PMKS+ while chapter 17 includes discussion on our team's findings as well as future improvements that should be made to the system. Finally, Chapter 18 discusses the future work that can be implemented to improve the usability and efficiency of PMKS+.

## Chapter 2

# Research Statement & Goals

In order for a website to reach its full potential, a developer must follow the global standards of web development. Prior to beginning, a developer must consider what tools they will use, what design their interface will have, and what functions their website will provide. Application of the global standards that are provided by the World Wide Web Consortium (W3C) can ensure compatibility and accessibility of a website.

PMKS has useful functions but limited compatibility with modern browsers. As of now, PMKS can only be used in Internet Explorer and does not provide support for any alternatives. This is a major problem for its users. Since many Mechanical Engineering students thoroughly study linkages for classes and their Major Qualifying Projects, it would be ideal to have software that could be useful to them for a long time.

Our goal was to develop a similar application that offered the same functionalities as PMKS, but that conformed to modern web standards and had a new interface that would improve the user experience. We developed these improvements by gathering feedback from Mechanical Engineering students on the current PMKS application and designing an interface to address any points of concern. Our focus was not to develop an entirely different application but to provide the Mechanical Engineering department with a new tool that improves the user experience. In summary, our goals were to:

- Incorporate beneficial features from PMKS into PMKS+, so that previous users feel comfortable with it.
- Ensure PMKS+ is compatible with most major browsers.
- Enhance the user experience by redesigning and improving both the features and the user interface of PMKS, based on user feedback.
- Deliver a new application that is appropriately documented and maintainable.

## Chapter 3

# What is PMKS?

The Planar Mechanical Kinematic Simulator (PMKS) is an open source software used to simulate the kinematics and kinetics of planar mechanisms (also referred to as linkages) through user defined joints, links, and forces. The initial kinematics version of the software was designed by Dr. Matthew Campbell of Oregon State University (that included parts of Prof. Radhakrishnan's dissertation) and is maintained by its Design Engineering Lab. A previous Major Qualifying Project group at WPI integrated the kinetics aspect into PMKS, which provided new features such as the ability to apply forces as well as to calculate and export static and dynamic joint reaction forces. Currently, PMKS is used in two Mechanical Engineering courses at WPI and some courses at Oregon State University.

## Chapter 4

# Literature Review

### 4.1 Interface Development

#### 4.1.1 Interface Design

When first considering the design of a graphical interface, developers work to establish standards prior to creating prototypes. These baselines help them make decisions about various aspects of the design of a system; from the fonts and color choices of text to the organization of menus. Alongside more general design rules for user interface design, this task analysis governs the choices developers make to create a system that allows access to the widest audience without inhibiting or frustrating them. The origins of the PMKS web application include very little documentation about the creation process and even less detailing decisions with regard to its design. It is unknown whether any considerations to design for a target audience, or to organize menus in a simple and coherent way were taken. As a consequence, our team must design and evaluate the existing system while creating design criteria and analysis methods from scratch.

#### 4.1.2 General Design Rules

Though we based the majority of our changes to the system based on feedback and evaluations conducted with user testing, there were certain, near-universal practices that we also considered. Through years of research, many practices in design have come to be expected of applications, especially those on the web. These principles can be applied to any system and improve various aspects of design that help users navigate, perform, and understand the information of applications.

According to Terry Felke-Morris in his book *Basics of web Design: HTML5 and CSS3* (2016), the basic principles for designing an interface are as follows:

- *Repetition: Repeat visual elements throughout the design*

When applying the principle of repetition, the web design should repeat one or more

elements throughout the page. The repeating aspect ties the work together. Whether it is color, shape, font, or image, repetition of elements helps to unify a design.

- *Contrast: Add visual excitement and draw attention*

To apply the principle of contrast, emphasize the differences between page elements in order to make the design interesting and direct attention. For example, there should be good contrast between the background color and the text color on a web page.

- *Proximity: Group related items*

When applying the principle of proximity, related items are placed physically close together. Unrelated items should have space separating them. Placing interface items close together gives visual clues to the logical organization of the information or functionality.

- *Alignment: Align elements to create visual unity*

When applying this principle, each element on the page should line-up (vertically or horizontally) with another element. This helps to reduce layout complexity.

Following these basic visual design principles can help develop an interface that is targeted for the audience and help them accomplish their tasks. Morris (2016) also gives significant importance to considering the accessibility of a website when developing it. Some of his key points are:

- *Content must be perceivable*

Perceivable content is easy to see or hear. All the graphic and multimedia content that is used in a website should also be available in text, such as descriptions for images, closed captions for videos, and transcripts for audio.

- *Interface components in the content must be operable*

Operable content has interactive features that can be used or operated with either a mouse or a keyboard.

- *Content and controls must be understandable*

Content must be easy to read, organized in a consistent manner, and provide helpful error messages when appropriate.

- *Content should be robust enough with current and future user agents*

Robust content is written to follow *World Wide Web Consortium* recommendations and should be compatible with multiple operating systems, browsers, and assistive technologies such as screen reader applications.

The *World Wide Web Consortium* (W3C) is active in the cause of promoting accessibility in websites by creating guidelines and standards that are applicable to web content developers, authoring-tool developers and browser developers. A web page that is designed to be accessible is typically more usable for all. The following design principles guide W3C's work:

- *Web for All*

The social value of the Web is that it enables human communication, commerce and opportunities to share knowledge. One of W3C's primary goals is to make these benefits available to all people, whatever their hardware, software, network infrastructure, native language, culture, geographical location, or physical or mental ability.

- *Web on Everything*

The number of different kinds of devices that can access the Web has grown immensely. Mobile phones, smart phones, personal digital assistants, interactive television systems, voice response systems, kiosks and even certain domestic appliances can all access the Web.

- *Web for Rich Interaction*

The Web was invented as a communications tool intended to allow anyone, anywhere to share information. For years, the Web was considered a "read-only" tool by many. Blogs and wikis brought more authors to the Web, and social networking emerged from the flourishing market for content and personalized Web experiences. W3C standards have supported this evolution thanks to strong architecture and design principles.

- *Web of Data and Services*

Some people view the Web as a giant repository of linked data while others as a giant set of services that exchange messages. The two views are complementary, and which to use often depends on the application.

- *Web of Trust*

The Web has transformed the way we communicate with each other. In doing so, it has also modified the nature of our social relationships. People now “meet on the Web” and carry out commercial and personal relationships, in some cases without ever meeting in person. W3C recognizes that trust is a social phenomenon, but technology design can foster trust and confidence. As more activity moves on-line, it will become even more important to support complex interactions among parties around the globe.

Shneiderman and Plaisant in their book *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (2016) present a comprehensive list of some additional essential guidelines for developers to follow without needing to evaluate their audience or tasks. The following is a summary of some of these guidelines for interface design that are applicable to our objective:

- *The display should be familiar to the user and related to the tasks*
- *Users should not be required to remember information between pages*
- *Only present data that is relevant to the user*
- *Minimize user input*

In addition, Shneiderman and Plaisant (2016) present several design principles that hold more weight than the aforementioned guidelines. These principles may be summarized as follows:

- *Cater to the user's skill level* - User expertise may vary from beginners to experts. Beginners require assistance and introductions to functionality while experts want to exert control over the system.
- *Define tasks* - Creating a task analysis that helps the designer understand the task frequency of the system leads to development with users in mind.
- *Golden Rules* - These general principles include consistency in presentation, functionality feedback, error prevention, and ease of undoing mistakes.

Following these guidelines will produce an appealing, agreeable system for users of any skill level. Our first objective in evaluating the current PMKS system constitutes ensuring that these principles have been enforced throughout the user experience. Although user testing will aid in understanding their effectiveness, we can evaluate many of these principles through performing our own evaluations using associated quantitative metrics. This point will be further discussed within the *Interface Evaluation* section of the methodology.

## **4.2 Interface Evaluation**

In order to improve a graphical user interface, there must be data pointing to aspects of the design which require improvement. In other words, there must be evidence to show that users are struggling or dissatisfied with the interface. Without this evidence, developers are liable to design a system based on their personal beliefs and preferences, regardless of what users want. To avoid this situation, testing must be done in order to collect user opinions on a system. This testing involves creating an evaluation strategy, conducting testing sessions, and evaluating the collected data.

Prior to testing, creating an evaluation strategy involves establishing and prioritizing usability requirements, defining data to collect, and identifying the constraints of the testing environment (Stone et al., 2005). Rubin (1994) describes that evaluation requires multiple testing periods throughout the development lifecycle to validate any changes made. This iterative testing pattern allows teams to continuously improve the design of a system and adapt to changes in design standards and evolving technologies while also simplifying the handoff of design improvement to other teams.

### **4.2.1 Usability Requirements**

Usability requirements identify the actions that a system should be able to accomplish

When assessing PMKS, our team identified several requirements that are necessary for the interface to satisfy, which the current iteration may or may not accomplish. Our evaluations were developed to address these goals and our progress toward accomplishing them. They are as follows:



- Users should be able to create basic linkages easily and with a small number of actions.
- Students should be able to apply prior knowledge and practices with regard to linkages.
- Users should be able to reset the system and begin from scratch.
- Users should be able to learn to create different types of linkages directly through the site.
- Users should feel in control of animations.

#### 4.2.2 Usability Dimensions

Usability dimensions help to prioritize system requirements and can be broken up into five categories: effective, efficient, engaging, error-tolerant, and easy to learn (Stone et al., 2005). Evaluations of user interfaces should consider the impact of each of these dimensions on the ability of the target audience to use the system. By assigning each category a weight based on their relevance and importance in an application, the interface can be refined to include necessities and omit unnecessary aspects. For example, if the efficiency of a system is given a weight of 30% while engaging is given 15% then the design of the system would focus on performing tasks quickly and requiring fewer actions by users but would not necessarily incorporate superfluous stylings or animations to engage the user.

#### 4.2.3 Usability Metrics

As previously discussed, design evaluation constitutes an iterative process that requires multiple testing sessions in order to re-evaluate changes to the interface. Performing multiple evaluations at different stages requires consistency in metrics used in order to compare and determine the success of attempted improvements. Common metrics that may be gathered, regardless of changes made to the system, include *the time it takes to complete specific tasks*, *the error rate of users when performing a task*, and *the user's subjective satisfaction*. Each of these metrics, both qualitative and quantitative, can be used as evidence to show improvement, worsening, or stagnation in a design.

Constantine and Lockwood (1999) describe several “Advanced Metrics” which gather information through user testing and self-evaluation of a system. These metrics provide a

thorough understanding of the capabilities and limitations of an interface. The metrics our group deemed relevant to this study are *essential efficiency*, *task visibility*, and *layout complexity*.

Essential efficiency compares the number of steps a user must take to complete a task to the ideal number of steps previously defined based on the number of necessary actions (Constantine and Lockwood, 1999). This measurement produces quantitative evidence regarding the effectivity and efficiency of a system. The following formula produces the essential efficiency (*EE*) based on  $S_{essential}$ , the number of ideal steps to accomplish a task compared to  $S_{enacted}$ , the number steps actually taken to accomplish a task:

$$EE = 100 * \frac{S_{essential}}{S_{enacted}}$$

Task visibility describes the user's ability to see functionality related to the tasks they must perform (Constantine and Lockwood, 1999). Determining the visibility of the system also helps evaluate how easy an application is to learn to use, and its overall efficiency. The following formula determines the task visibility (*TV*) based on the total number of enacted steps to complete a use case,  $S_{total}$ , multiplied by the sum of visibility for each task,  $\Sigma V_i$ :

$$TV = 100 * \left( \frac{1}{S_{Total}} * \Sigma V_i \right)$$

Layout complexity evaluates the placement of objects on screen by measuring the distance and differences (size, alignment, angle, etc.) between their positioning (Constantine and Lockwood, 1999). This measurement assesses a system's efficiency and can aid in creating an engaging, easy-to-use interface. Layout complexity defines the following formula using the variables listed below:

$$LU = 100 * \left( \frac{(N_h + N_w + N_t + N_l + N_b + N_r) - M}{6 * N_c - M} \right)$$

$LU$  = Layout Complexity

$N_h$  = Number of objects with different heights

$N_w$  = Number of objects with different widths

$N_t$  = Number of objects aligned to the top of the page

$N_l$  = Number of objects aligned to the left side of the page

$N_b$  = Number of objects aligned to the bottom of the page

$N_r$  = Number of objects aligned to the right side of the page

$N_c$  = Total number of objects

$M$  = Variable to ensure quantity does not fall below zero

## Chapter 5

# Methodology

In this chapter we detail the steps our team took to accomplish the goals established in Chapter 2. Each section describes the actions taken by one or more team members in a chronological order. The following chapters include detailed explanations on the decisions made in each section of the methodology.

### 5.1 Migration Options

Before deciding between a web and desktop application, we considered the options available to us behind each approach. This included the programming language, framework, and method of migration. Through research, and by considering recommendations on technologies provided by Professor Lane Harrison of the Computer Science department at WPI, we finalized our migration strategy. We decided to use Angular, a web framework developed by Google, as our development platform. This framework utilizes HTML5, CSS, and the TypeScript language to assist in building and designing website interfaces (Angular, 2018). Lastly, we decided to do a partial redesign of the application as our migration method because it would keep the existing structure of the PMKS code the same, as well as allow us to redesign parts of the application as needed. Section 6.1, Migration Methods, further discusses the various migration methods we looked at and Section 6.3, Migration Tools, discusses various tools and technologies for those methods.

### 5.2 Web App Justification

After considering each migration option, we needed to choose between creating a web or desktop application. First, we listed important criteria for developing an application. These criteria were then given weights based on priority. Each migration platform option was given a score for each criteria which, in turn was multiplied by the criteria weight and summed for comparison. The criteria and scores were represented in a decision matrix shown in Figure 5.2. In the end, we decided on a web application because its assets met our project goals better, as

indicated by its higher score. Section 6.2, Migration Platforms, lists the various advantages and disadvantages of web and desktop applications, which assisted in justifying the scores for each platform. Section 7, Web App Justification, further details our process in selecting a platform for PMKS+ and provides rationales for criteria weights and the values assigned to each.

### 5.3 User Interface Evaluation

Before beginning the conversion, we evaluated the current state of the PMKS system and determined the initial changes that needed to be made to the user interface. Based on our research into evaluations of user interfaces discussed in Section 4.2, our team developed a process to gather quantitative and qualitative feedback from users in order to improve the design of our system. The creation and design of our evaluation is detailed in Section 6, User Interface Evaluation.

Evaluations were completed during two sessions that took place on October 3rd and 10th, 2018. Each session included 16 subjects placed at individual computers in a single room. All subjects started the evaluation at the same time and completed both the assigned tasks and accompanying questionnaire at their own pace.

In addition, six one-on-one sessions were conducted which paired each individual subject with a proctor who recorded the subject’s observations. This allowed subjects to interact with the system in an uninhibited manner and express their opinions immediately. These sessions were scheduled at separate times from the larger group sessions. The subjects were given as much time as necessary to complete each step of the evaluation.

Our team compared the results gathered through these sessions and the user evaluations performed with PMKS+, described in section 5.10, to find evidence of improvement in our new application. Figure 5.1 details the timeline for evaluation.

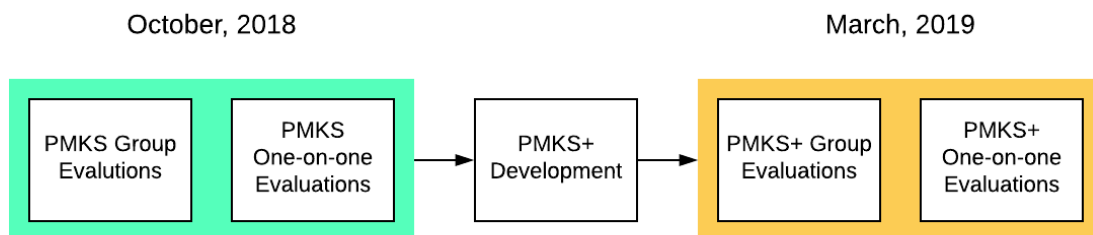


Figure 5.1: A timeline of the evaluation process between PMKS and PMKS+.

## 5.4 Interface Design

After evaluating the previous interface of the application and conducting the interface evaluation survey, we worked to design a new user interface for PMKS+. Although we had some ideas about the design before the two tasks mentioned, we decided it was best to complete those two tasks first and design the interface around the feedback received.

As mentioned in our research statement, our goal was to create an application that is similar to PMKS, in functionality and design. Such an application should feel familiar to existing PMKS users while providing a better user experience for all users.

During B-term, we incorporated our design choices into weekly iterations of our application. Every week we made changes to the interface of the application, sought opinions from our advisors, and incorporated their feedback in the next iteration of the interface while still adding more interface components. Chapter 9 details the various iterations of the interface leading up to the current design. In that section, we will also discuss the justifications for each interface element of our application.

## 5.5 Linkage Creation Methods

We quickly identified linkage creation as a major area in need of improvement in PMKS. The process of specifying links in a table using the coordinates of attached joints, though accurate, proved cumbersome and unintuitive to the user. In an effort to fix this issue, our team sought a way to create joints and links that mimicked the method for drawing linkages by hand.

Our team found that keeping a table which tracks the location and details of links and joints allows users to identify items on the grid more easily and edit the position of these objects accurately. This table was also expanded to distinguish between joints and links, and also to include information on the forces affecting the linkage. We also created a method to help the user identify what actions they may perform in PMKS+. Initially, we bound many of the user actions, such as connecting joints and setting a joint to ground, to specific key presses. Later, we assigned these actions to descriptive options in a context sensitive menu that is accessed by right-clicking on a part of the linkage. The design and implementation of these creation methods are discussed in Chapter 10.

## 5.6 System Design

The system architecture of PMKS was used as a source of inspiration for PMKS+. To accomplish this, we analyzed and mapped out the architecture in PMKS and looked for ways to mirror it in our new design. We found that Silverlight’s “page component” system (Figure 5.1) could be emulated through Angular’s own variation of components called “Angular components” (Figure 5.2). This allowed us to maintain the overall structure of PMKS within PMKS+ using the more modern Angular Framework.

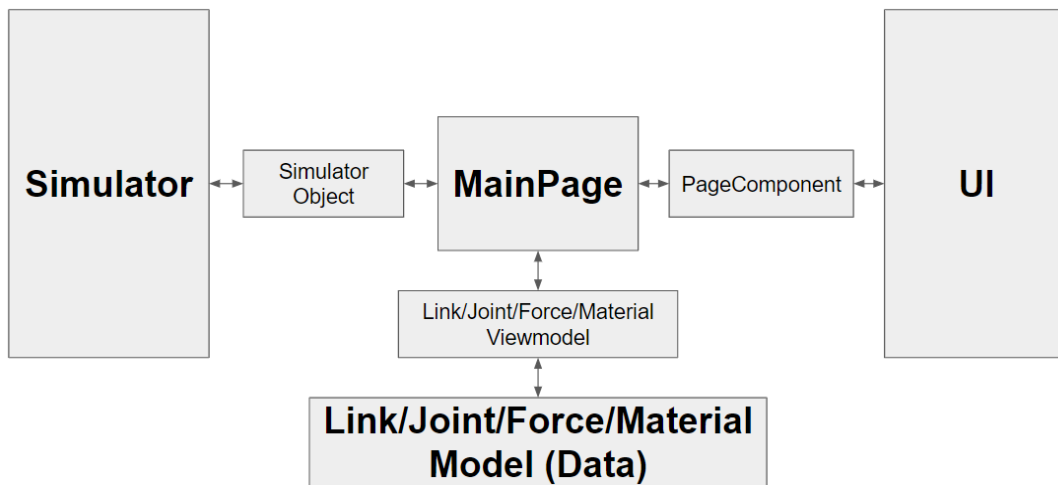


Figure 5.1: System architecture of PMKS

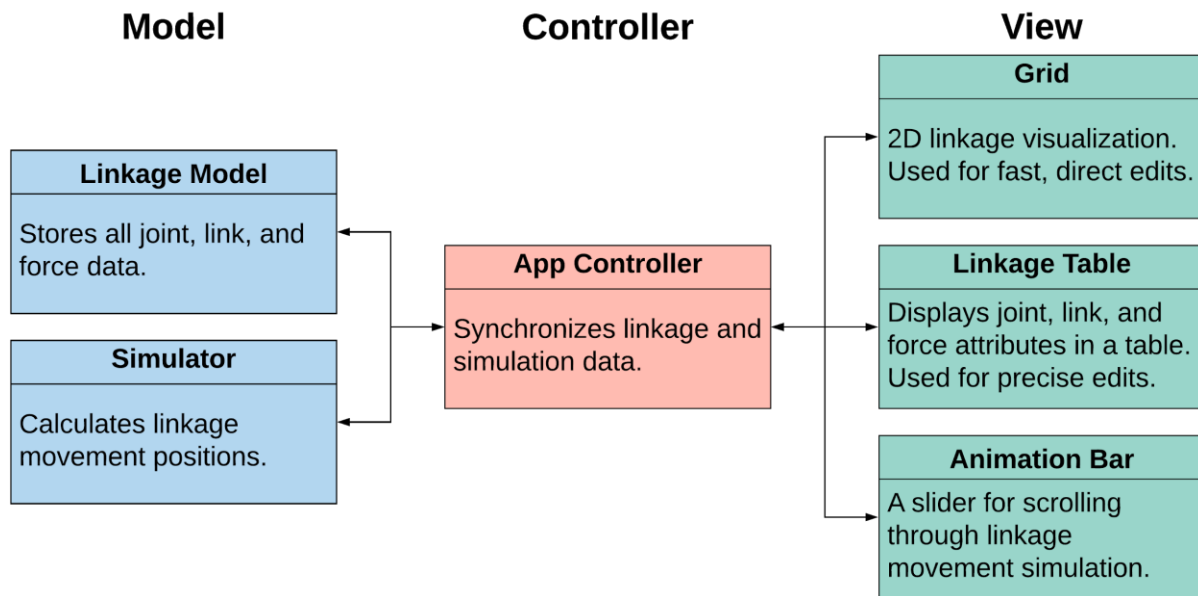


Figure 5.2: System architecture of PMKS+

As functionality in the grid component expanded, our team began facing issues with overlapping user input. It was difficult to distinguish when a left mouse click should allow a user to drag a joint, and when it should be used to set a link between joints. To overcome this obstacle, we refactored the code in the Grid Component so that it followed a branching sequence of states to determine which action would get triggered on user input. By simplifying these conditionals to a state the component was in, it would be easier to rearrange and remap actions to user input. A diagram of this sequence can be seen below in Figure 5.3.





## **5.7 Simulator Class**

The Simulator Class is responsible for calculating kinematic data from user defined joints, links, and forces. Our team used the structure and algorithms from the old PMKS to construct a new simulator for PMKS+. We identified three major components of the simulator. These were the position solver, joint traversal algorithm, and the simulation loop. The position solver calculates a joint's position, the joint traversal algorithm runs the position solver at each joint, and the simulation loop runs the joint traversal algorithm for each time step in the simulation. We divided these components between group members and coded them individually. After each component was completed, they were joined together and connected to the rest of the application. The details on each component can be found in Chapter 12.

## **5.8 Animation**

Creating animations in PMKS+ began with identifying the tools necessary to produce proper linkage animations. Using feedback from the user evaluations, and the previous version as a model, we determined aspects of the animations that were good and those that needed to be improved.

Using the JavaScript Keyframes library, our team attempted to create CSS keyframe animations to move the links in our system. This method of animation was unsuccessful due to incompatibilities between the imported jQuery module and our Angular environment. There were also discrepancies between CSS animations and the scalable vector graphics used in our system. The initial process for creating these keyframes and the reasons for its incompatibility are detailed in Section 13.1.

Based on our findings from using the library, we decided that the best solution going forward was to develop the functionality manually using TypeScript. This process involved taking in coordinates of each point and relocating each SVG element accordingly. The rationale behind this decision and its implementation are discussed in section 13.2.

## **5.9 Application Testing**

While developing each component, unit tests were written to test the current functionality and as a safeguard for future changes. Once a component was completed, integration test cases were written to ensure that the component was always sending and receiving the correct data.

Once the three main components of our application (the linkage table, grid, and mainpage) were completed and linked, more advanced test cases were employed. In these tests, we not only tested if the application was working, but also if it would still work under unusual actions. For example, even if degrees of freedom is 1, there are some instances where our implementation of the simulator class cannot simulate linkage movement. In this case, PMKS+ should stop simulating and not attempt to animate.

## **5.10 PMKS+ User Evaluations**

Following the completion of iteration five of our interface design (as described in Section 9.6) our team performed short user evaluations of PMKS+ to gather feedback and further improve our design. We performed these evaluations with five subjects in separate, one-on-one sessions. The details of these individual evaluations are discussed further in Section 15.1.

Upon completing iteration 6 of interface design, our team performed an additional user evaluation in order to determine whether our application accomplished the task of improving on the PMKS+ design. This evaluation mimicked the first evaluation described in Section 5.3 and was performed with 19 users. The results were then compared with our findings from PMKS in order to determine whether our design improved user satisfaction, accuracy, and understanding.

## Chapter 6

# Migration Options

In this section, we analyze the different options for migrating PMKS. First, we look into the migration methods. Afterwards, we look into the platforms that support these methods.

### 6.1 Migration Methods

Our team identified 2 potential methods of migration for PMKS: porting and rewriting.

**Porting** the application was the most straightforward option. It meant recycling the source code when possible and “tweaking” it into a working state. In order to port PMKS, we needed to find tools that supported C#/XAML, the Silverlight programming languages. We also needed to look into modern substitutes for any deprecated functionality that was directly provided by the Silverlight framework.

**Rewriting** the code meant using a new coding language to recreate the software. This option would be best if the migration platform did not offer adequate support for C#/XAML. Rewriting the code offered two varying levels of redesign.

- A *complete redesign* of PMKS would call for a full evaluation of the PMKS source code. This would involve analyzing the code and looking for areas of structural improvement. In doing this, we could create a more organized and optimized software. This choice also comes with significant risks. Poor design choices could require us to return to the planning phase several times.
- A *partial redesign* of PMKS would be a mix between a complete redesign and a port. In this option, we would rewrite the new PMKS in a modern language while keeping the structure and flow of the original source code. This would allow us to focus our attention on specific areas of the user experience rather than the underlying code. To accomplish this, we would need to identify the components in PMKS and have a conversion plan for each. Compared to a complete redesign, a partial redesign is much quicker. It is also less prone to major structural issues since our team would have a working model of the

software to reference. However, this option would lack the opportunity for low-level structural optimizations that a complete redesign would provide.

## 6.2 Migration Platforms

We looked into two potential platforms for migrating PMKS: a web app and a desktop app. A desktop app is a piece of software hosted and accessed locally on a computer. Desktop applications are typically limited by the hardware in which they run on. These applications must be developed for particular operating systems and may have additional hardware requirements to function properly (Bychkov, 2013). On the other hand, a web app is a piece of software that is stored on a server and presented through a web browser (Pop, 2005). While both web and desktop applications are used today, they have innate differences that are used to justify their use in specific cases. We will discuss the advantages and disadvantages of these two platforms.

### Web App Advantages

- **Accessibility:** Web apps have the advantage of being quickly reachable via a link in a web browser. Additionally, no additional installations are required to run HTML5 web applications.
- **Compatibility:** Since web apps can be accessed through any web browser connected to the internet, it is possible to target a wide variety of devices and operating systems simultaneously.

### Web App Disadvantages

- **Language Restriction:** Since the advent of HTML5, many web apps have conformed to using HTML5/JavaScript as their front end language. Plugins like Silverlight, and Adobe Flash that allowed the use of other programming languages are being deprecated in favor of HTML5/JavaScript. Because of this, it has become very difficult to reuse legacy Silverlight code in modern web apps.
- **Computing Power:** Due to the language restrictions defined above, computing power on web apps is more limited than desktop alternatives. JavaScript has a very difficult time leveraging hardware during computation intensive operations (Mozilla Contributors, 2019).

- **Performance:** A study comparing the performance of web applications and desktop applications looked at the time for different tasks to complete (Pop, 2005). On average, the tasks were 2.3 times slower on a web application than a desktop application.
- **Connectivity:** A web application relies on a constant connection to the Internet. Poor connection or absence of one can cause performance issues (Avesta Group, 2015).

### Desktop App Advantages

- **Language Restriction:** Desktop apps are supported by a wide range of languages ranging from C to JavaScript. This makes porting a much more straightforward operation.
- **Computing Power:** Due to the support of low-level languages such as C, desktop apps are better able to utilize local system hardware.

### Desktop App Disadvantages

- **Accessibility:** Desktop apps require an installation process which may require administrative rights (something that isn't always an option on public computers). Once installed, they live on the user's computer and take up space. Updates must also be distributed to each computer on which the program is installed (Avesta Group, 2015).
- **Compatibility:** Since desktop apps are run and managed locally, allowing for cross-platform support requires tailoring development to each operating system's standards.

## 6.3 Migration Tools

In this section, we look into the specific tools available for desktop and web apps.

### Web App Porting

1. **CSHTML5** is a paid solution for developers wanting to continue coding web apps in C#. This solution works as a plugin for Visual Studio and allows developers to code in C# while taking advantage of the latest technology in HTML5. The software is available via subscription for \$119 per month or \$1,999 for a perpetual license (C#/XAML for HTML5, 2018).
2. **Blazor** is an experimental Microsoft framework aimed at allowing developers to code client-side C# code for web apps. Unlike Silverlight, Blazor takes advantage of a new

web technology known as Web-Assembly. Web-Assembly is a low-level web language that developers leverage to create web compilers for languages other than JavaScript. While promising, Blazor is still in a very early, experimental phase. Microsoft does not recommend using it in production-level apps (Roth, 2018).

## **Web app Rewriting**

1. **KnockoutJS:** KnockoutJS is a JavaScript library used to create rich, client-side user interfaces. It is structured using the Model-View-ViewModel design pattern to create a development environment similar to Silverlight. After its release, it was used by many as a tool for Silverlight migration (Sanderson, 2011). A major disadvantage to Knockout is its future support. KnockoutJS was released in 2010 and hasn't received an update since March of 2017 (Best, 2017). The creator has moved on to other projects such as the Blazor framework described above.
2. **Angular:** Angular is a Google framework used to structure code and build user interfaces. Angular focuses on organization and the concept of component based programming. This concept relies on splitting the application into different components that work together. Like KnockoutJS, Angular is structured using the Model-View-ViewModel design pattern (Darwin, 2017). Angular is best used in websites that contain complex user interfaces with many interconnected components.
3. **React:** React is a Facebook developed framework used to create large applications where data is constantly changing and needs to be updated on screen. React accomplishes this by using web page templates rather than static text. This allows web pages to be reused and refitted with different content. Unlike Angular, React does not enforce a specific design pattern. Instead, it leaves structure up to the developer. (Vaughn, 2013). React is best used in websites that require displaying constantly changing data over templates.
4. **VueJS:** Vue is a framework that combines the templating of React with the component based programming of Angular. It is designed to be lightweight and easy to learn. A major risk to using Vue is that it is new and not supported by a company (You, 2016).

## **Desktop App Porting**

1. **Universal Windows Platform (UWP):** UWP is the standard platform for developing apps on Windows 10. Microsoft plans to support UWP for the foreseeable future. However, UWP is not supported on older Windows versions (7, 8), as well as competing operating systems (Mac, Linux). UWP apps are distributed and updated through the Windows store. UWP also provides support for utilizing the Mobilize.NET Silverlight Bridge. This bridge is a Microsoft funded tool that scans source code and converts Silverlight functionality to UWP (Satran, 2018).
2. **Windows Presentation Foundation (WPF):** Windows Presentation Foundation is the older way of developing Windows apps. It supports all current Windows operating systems since Windows XP. It is a superset of Silverlight, making a port to WPF much more practical. WPF support is being phased out in favor of UWP (Warren, 2018).

## **Desktop App Rewriting**

1. Both UWP and WPF provide the tools to rewrite PMKS. Since the platform supports C# code, it would be possible to reuse portions of the existing code when needed. This allows for an incremental redesign which would in turn help isolate the development of components.



## Chapter 7

# Web App Justification

The first major decision that our team made was whether to create a desktop app or web app. In order to choose between the two, we created a decision matrix, shown below in Table 5.2. This process was used to identify the advantages and disadvantages of each platform. Through this process, we identified various criteria that were important for the development of the application. Each criterion was given a weight to correlate to its level of importance. The team then discussed the options after conducting preliminary research and assigned each type of application a value for each criteria.

<b>Development Criteria</b>	<b>Weight</b>	<b>Web App</b>	<b>Web Total</b>	<b>Desktop</b>	<b>Desktop Total</b>
Ease of conversion	10	4	40	4	40
Optimization	4	3	12	4	16
User Convenience	6	5	30	3	18
Team Experience	8	4	32	4	32
Documentation	4	3	12	2	8
Total			126		114

Table 5.2: Decision matrix for comparing web and desktop applications

## 7.1 Rationale

The weight of each criterion of the decision matrix was assigned by its importance to the goals of the project. The process of assigning scores for each criteria took into account the benefits of each type of application relative to the criteria. The team deliberated together on each value that was assigned. For the team experience criterion, each member rated their experience with each type of application. The ratings were then averaged to produce a more accurate evaluation of team experience.

### **Ease of conversion:**

Conversion was the most important criterion because it could impact the time it takes to complete the project as well as compatibility with multiple platforms. This criterion takes into account the compatibility of each type of application with different operating systems and hardware because ensuring compatibility can be more difficult for different options. A lower value for this field considers the option more difficult for conversion while a higher value would be easier to convert.

**Web App** - PMKS is written using the C# language. We assigned a web app a value of 4 for ease of conversion to due to its poor compatibility with C#. Creating a web app would require the code to be rewritten in another language, most likely JavaScript. On the other hand, converting code from one language to another is still easier than having to write the application from scratch. Since a web application runs through a browser and is not restricted to certain hardware or operating system, the conversion would only involve creating one application.

**Desktop** - We assigned a desktop app a value of 4 because the conversion process from the current web-based application to a locally-run application would not require the least amount of code to be rewritten. Most of the current C# code would not need to be rewritten. The XAML files could also be reused but would need substitutions for any Silverlight dependencies. However, the conversion process would be problematic when developing for multiple operating

systems. As a result, this process would have similar complexity to creating a web app, and therefore a similar amount of work.

### **Optimization:**

Optimization refers to the act of making an application run more efficiently and use fewer resources. Considering the scope of the application, this is not a major priority. As a result, we assigned a weight of 4 for this criteria.

**Web App** - We assigned the web app a 3 because web applications perform slower than their desktop counterparts. This is due to the limitations of JavaScript when compared to lower level languages such as C. The speed is also dependent on the strength of the internet connection. Poor network connection (or having none at all) could affect the use of the app.

**Desktop** - We gave the desktop app a value of 4 due to its ability to access more of a computer's resources. Desktop applications typically work faster to complete certain tasks and the application is also not dependent on a network connection.

### **User Convenience:**

User convenience refers to the application's accessibility to end users and the difficulty to update the application. This includes obtaining initial access to the application as well as keeping it updated with future releases. User convenience is one of the main goals of our project which justifies its weight of 6.

**Web App** - Web app received a 5 because of the convenience of accessing it through a web browser. Developing an app with cross-browser support would allow any computer with an Internet connection to access it. Users also would not have to worry about updating local files as any updates would be applied upon reconnecting with the server. However, this also means that the application cannot be run locally on a computer without Internet connection.

**Desktop** - We assigned desktop a 3 because of the downloading process required and the difficulty of distribution. If made available to everyone, the desktop application would require an initial download and local storage in order to use it. These local applications would also require users to download updates when needed, making the program harder to maintain. However, one benefit is that the application can be run without an Internet connection. In terms of providing the application to the university, if the software is stored on WPI's servers and requires connection to the WPI VPN or use of a WPI computer, the ease of access becomes much lower.

### **Team Experience:**

Team experience takes into account the skills that this project team can apply to both types of applications. This includes experience with languages and with creating similar applications. Since our experience can influence the quality of the final product, it was given a weight of 8. In this section we take into account the experience of all team members and average it to create an overall score.

**Web App** - Web app received a 4 because the team has some experience working with them although not in the scale or context that this project requires. Three out of the four team members have taken a course on web development and have completed various projects including web applications. While the team does not have experience with C# and Silverlight which are used in the current application, we are familiar with creating web applications specifically using HTML5 and JavaScript.

**Desktop** - Desktop app received a 4 because all members of the team have worked on creating desktop apps. We have completed a Software Engineering course which entailed creating a complex desktop application using Java. However, the team does not have much experience with the C# language which would be used for developing this type of application. Therefore, the team experience rating for the desktop app was the same as the web app.

### **Documentation:**

Proper documentation would aid our development when working with new languages and frameworks. In many cases, documentation exists for very similar projects, as well as about

methods for migrating Silverlight applications. Although it can be useful, additional documentation is not necessary as proper documentation exists for languages and libraries associated with C#, Silverlight and the potential languages we might work with. Due to the benefits that proper documentation can have give, the criterion received a weight of 4.

**Web App** - Many Silverlight conversion projects have been performed in the past few years due to the lack of cross-browser support and the announcement of Silverlight's discontinuation. This has led to documentation, articles, and questions supporting further conversion of these apps. This criteria received a value of 3 as a result.

**Desktop** - The lack of conversion projects from Silverlight web apps to desktop applications resulted in less documentation. The documentation of C# still exists to help with development. As a result, desktop app received a value of 2 in this category.

## 7.2 Results

After assigning each option a value for all criteria, they were multiplied by their respective weights and summed. The overall score for the web app was 124, while the score for the desktop app was 114. As a result, decided that the web app was more suitable for our project. According to Table 5.2, the main advantage of the web app over the desktop app was user convenience as a web app can be accessed using a web browser with internet connection.

## Chapter 8

# User Interface Evaluation

Improving the graphical user interface of PMKS first required our team to know how well the current version performed. Although there are tests to evaluate an interface's performance through direct measurement, the best method of gathering feedback on a system is through testing with subjects. Our team performed a total of three separate evaluations, the first was performed using PMKS and occurred before development began. The second and third were conducted using PMKS+ and were performed during development. These evaluations served as an indicator of our users' preferences and opinions about the system as well as how well we were meeting their needs.

The evaluation process required that we create a procedure that could be used with different versions of the system at various stages in development in order to objectively measure the benefits of any changes made in design. We developed an evaluation strategy that detailed the most important aspects to focus on when evaluating PMKS and PMKS+. This strategy was based on relevant usability dimensions, the objectives we would accomplish by evaluating the system, the metrics that would be collected and analyzed, and our target audience.

### 8.1 Evaluation Strategy

In order to properly direct our focus on improving the design of PMKS, our team assigned the following weights to each usability dimension defined in Section 4.2.2. An explanation for the attribution of each value follows the breakdown below.

- Effective - 25%
- Efficient - 20%
- Engaging - 10%
- Error Tolerant - 25%
- Easy to Learn - 20%

**Effectiveness** refers to the application's ability to complete its assigned job with few user steps as errors. Our main objective when recreating PMKS was to create an application capable of replacing the older version. PMKS+ should provide all required functionality to fulfil the same tasks as PMKS and the conversion must function to the same standard.

**Efficiency** describes the speed with which users can accomplish their goals within a system. The target audience of PMKS are engineering and robotics students who use the application to complete class assignments. While not as important as its effectiveness, an efficient application will prevent user frustration and aid in completing work quickly, a trait desired by students.

**Engaging** applications serve to draw users in and entice them to use the service. PMKS serves as a simulation that students will use because of their assigned work, regardless of the appeal of the system. Our focus is on providing proper functionality and not on retaining users.

**Error tolerance** works conjointly with effectiveness and describes the system's ability to respond to user errors. Students must be expected to make mistakes when interacting with the system and PMKS must be ready to handle these gracefully. In addition to preventing errors, the system should fix problems users encounter or help users to restore the system to a working state.

**Ease of learning** is a significant dimension for any system that must cater to new users. PMKS is currently used in multiple classes throughout the school year which requires many new students to begin using the system. This leads to an audience of many beginners with very few experts who continue to use the system. The interface must, therefore, be tailored to teaching new users. This may include implementing tutorials, reducing the user's short-term memory needs, and pre-filling certain fields as examples.

Assigning values to these five dimensions aided our team in specifying the goals we needed to accomplish to create a successful application. Our team sought to test each fully-functional aspect of the PMKS interface from a semi-experienced user's perspective. We also

attempted to establish a standard in linkage modeling by gathering feedback based on the subjects' use of other modeling software. These evaluations helped our team establish a starting point and track our development progress toward completing the following objectives:

1. Allow users to effectively (accurately and with few/no errors) create complex linkages in minimal time (i.e., create a four bar mechanism with DoF=1 in 2 minutes)
2. Minimize navigational errors when creating, deleting, exporting, and saving within PMKS.
3. Identify aspects of other software, that PMKS is lacking, which may satisfy users more.

In order to evaluate our progress in accomplishing these three objectives, we assigned subjects tasks related to those in the objectives to gather information based on associated metrics. The questions were given in a questionnaire that served to gather data on the following metrics:

- Time taken
- Error rate
- Frequency of help
- Number of operations
- User satisfaction

These task evaluations required a pool of test subjects that reflected the target audience of PMKS. Our team determined that the main demographic of PMKS are Mechanical Engineering and Robotics Engineering students at Worcester Polytechnic Institute. Specifically, the simulation sees the most use from the class *Modeling and Analysis of Mechatronic Systems*. The subjects we chose for our evaluations were mechanical and robotics engineering majors enrolled in the aforementioned class who had some experience with PMKS through the assigned curriculum. This pool of subjects allowed our team to design toward the skill level of our target audience and receive relevant feedback from the pertinent users themselves.



## **8.2 Evaluation Procedure**

The PMKS evaluation consisted of two different sessions. The first involved a large group of subjects performing assigned tasks and completing a survey. The second involved one-on-one sessions in which a proctor observed a single subject performing assigned tasks while describing their thought process. Both evaluations followed the methods stated below and collected information based on the metrics described above in the Evaluation Strategy (Section 8.1).

### **Session 1:**

Subjects were seated at a computer and directed to a Google Form. Subjects answered the questions found in the “Pre-Tasks Survey” section of this questionnaire. Once the subject finished, they clicked “continue” in the form and moved on to the “Tasks” section. This section of the form instructed students to complete two tasks using the software. First, subjects recreated a linkage based on a given image, applied a force to it, and then exported the data. When finished with each part, they answered the associated questions in the form. Next, subjects continued to the second task which required them to open a preloaded file containing a diagram. They were instructed to adjust the diagram within the software to match a provided image. When they completed this task, the subjects moved on to the final section of the Google Form. They then answered the “Post-Tasks Survey” questions and submitted the form when finished.

### **Session 2:**

Session 2 involved a one-on-one observation between a proctor and a test subject. The session followed the same procedures as Session 1 and used the same online form for subject responses. The session included new subjects that were not participants in Session 1. During this session, subjects were asked to state their thought process out loud while working with the software. Proctors recorded the subject’s remarks and actions on paper and intervened only if the subject was unable to proceed with the given tasks.

The initial questions asked were used to gather data in order to measure the time taken and error rate of users interacting with PMKS. This corresponds to the effectiveness, efficiency,

and error tolerance of the system. The questions that subjects answered while performing the assigned tasks aimed to gather information about:

- Completion time
- Subject error rate
- Completion rate

Subjects ended by completing the Post-Tasks Survey. The questions in this survey aimed to gather qualitative data on our target user's preferences. This corresponds to the ease of learning and engaging elements of the system. The survey asked for the following information:

- Subject thoughts
- Subject reactions (frustrations, impatience, etc.)
- Subject satisfaction
- Recommendations

The observers for the second session gathered data to establish the number of operations a user made in the old system to accomplish tasks. This sections corresponds to gathering further information on the effectiveness and efficiency of the system. Proctors that performed one-on-one observations with subjects collected the following data in addition to the metrics above:

- Enacted steps
- Functionality frequency of use
- Subject's remarks

### **8.3 Data Analysis**

We used the resulting data from these evaluations to form design choices for PMKS+ based on the metrics gathered. Users experienced a very high number of actions in order to accomplish the most frequently used functionality of the system, creating a linkage. Based on this finding, our new system must allow users to create links with a minimal number of steps. We also found, based on the data shown in Figure 8.1, that there was not a correlation between the users' experience and the amount of time taken to complete the assigned tasks as we had previously predicted to be true.

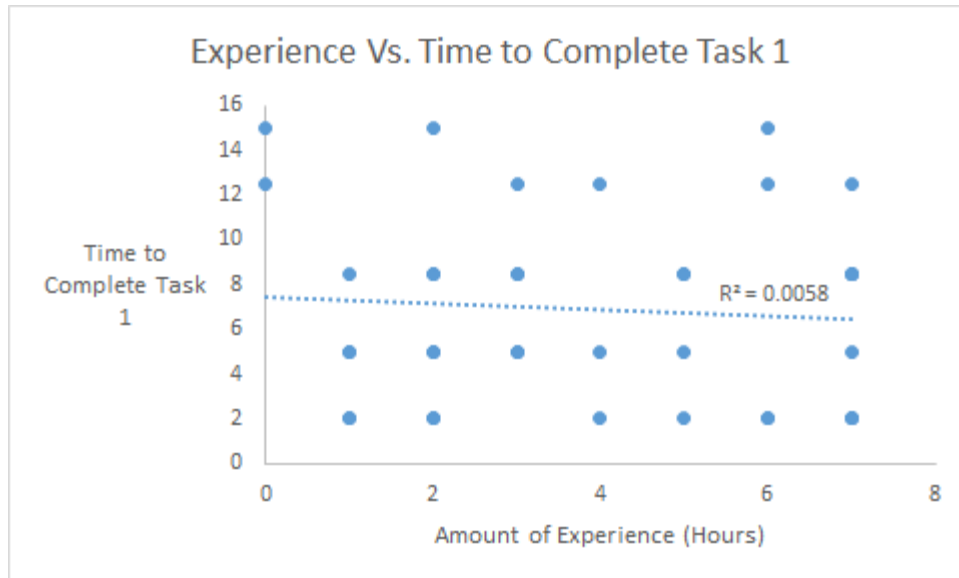


Figure 8.1: Subject experience with PMKS vs. time to complete the assigned task.

The majority of users also rated their usage of the system positively, giving an average of 4 out of 5 for their usage of the interface. They also rated their experience as on-par with other modeling software. Figures 8.1 and 8.2 depict subject’s experience with other modeling software and their comparison of that software to PMKS.

These findings aided our design of the PMKS+ interface during development. In addition, these results were compared to evaluations performed on PMKS+ after iteration 6 of development in order to evaluate the effectiveness of our changes. These comparisons and their value are detailed in Chapter 15.

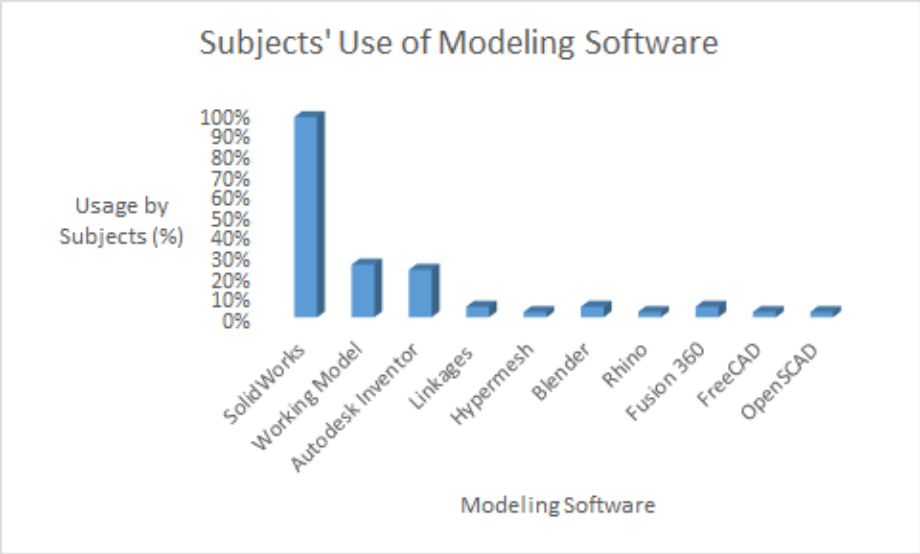


Figure 8.2: User experience with other simulation software.

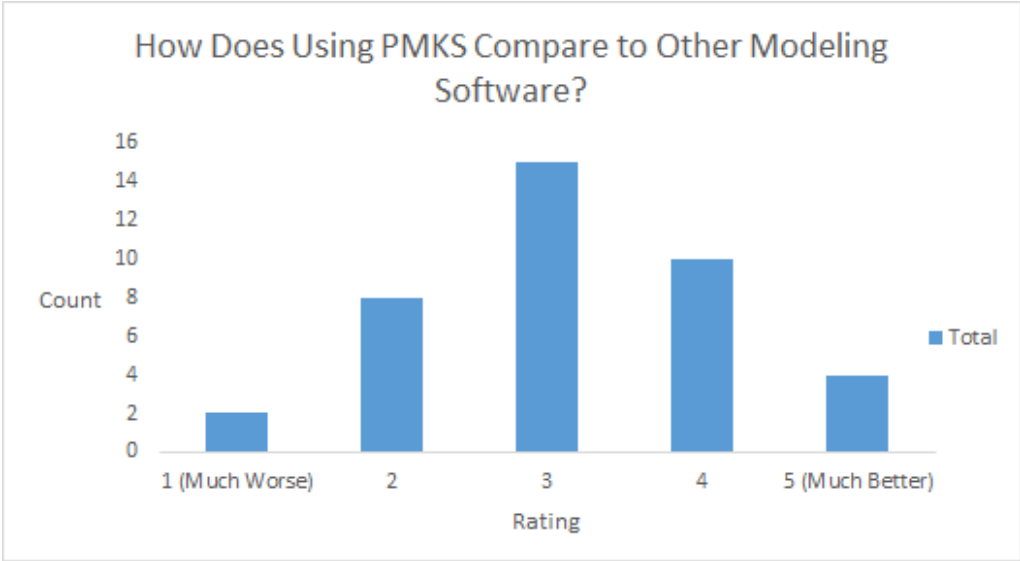


Figure 8.3: User comparison of PMKS with other modeling software.

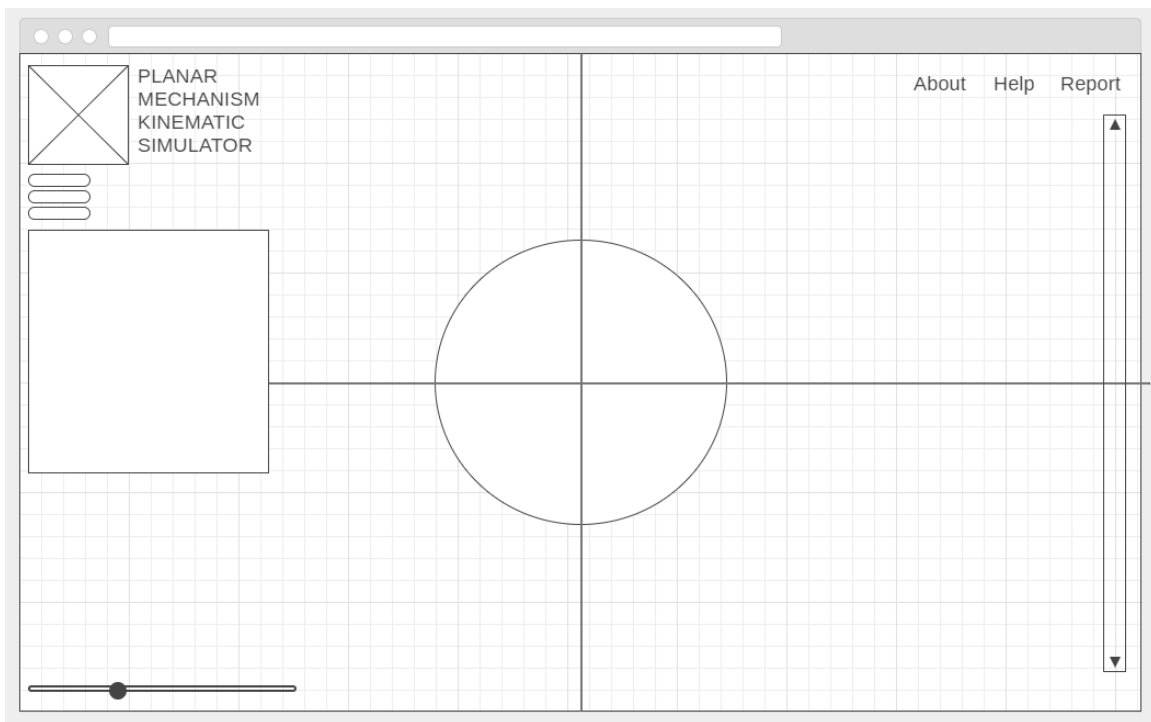
## Chapter 9

# Interface design

In this section, we describe the new interface design of PMKS+. First, we present preliminary UI mockups developed before the initial interface evaluation of PMKS. Then, we discuss the different iterations of the design, examine the changes made at every iteration, and discuss the rationale for each choice and component.

### 9.1 Preliminary UI Mockups

Figures 9.1 and 9.2 show two UI mockups made during the preliminary stages of the interface design. These mockups were made as a baseline to visualize design choices that we would make in the future.



*Figure 9.1: First mockup of the new interface design. Created using Balsamiq, a web app for creating mockups*

Figure 9.1 represents the first UI mockup that was developed for our project. We tried to recreate the aesthetic of PMKS, but also add some new components such as buttons and a logo. The toolbar, animation timeline and zoom were kept in the same location as was in PMKS in order to be familiar for previous users.

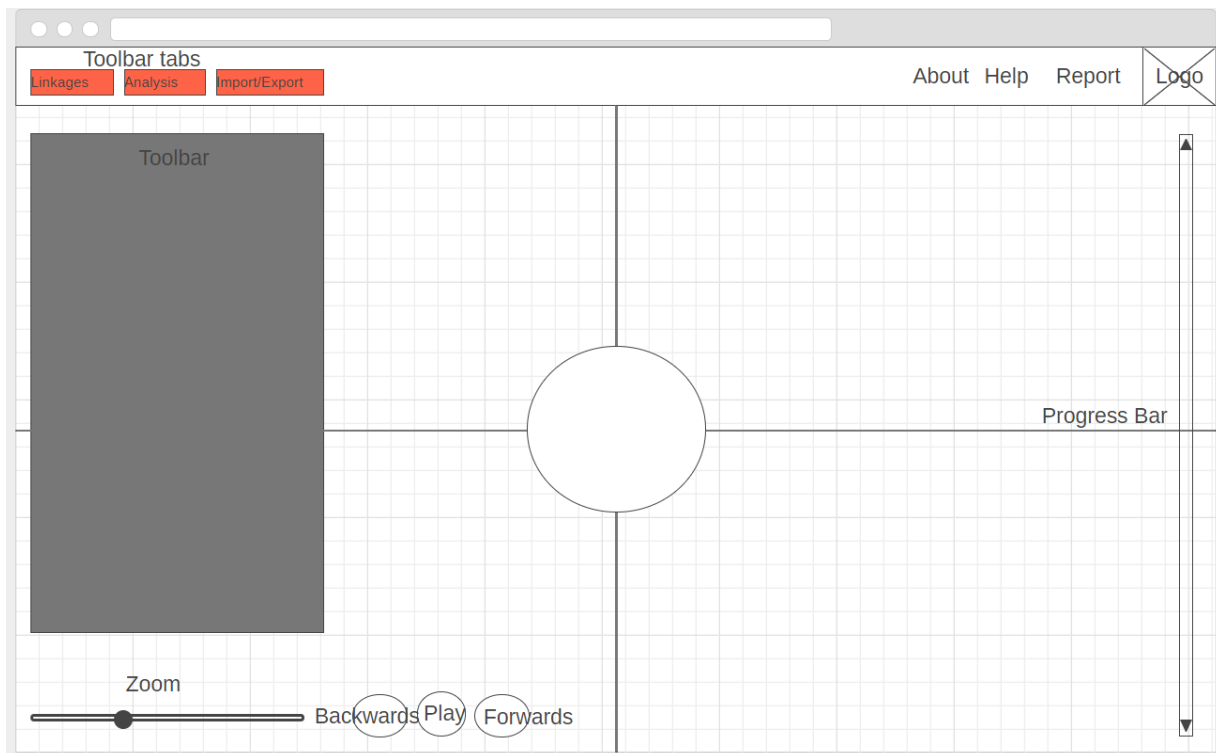


Figure 9.2: Second mockup of the new interface design. Created using Wireframe.cc.

Figure 9.2 represents the second UI Mockup developed for our project. We added more buttons for controlling the animation, moved the logo to the upper-right corner, changed the toolbar to a static component at the left side of the screen, and added the toolbar tabs to provide more space for functionality.

## 9.2 Iteration 1

Once development started, we began using HTML and CSS to design the interface. At this stage, we prioritized implementing functionality over design. This led to our initial design including very little styling (Figure 9.3).

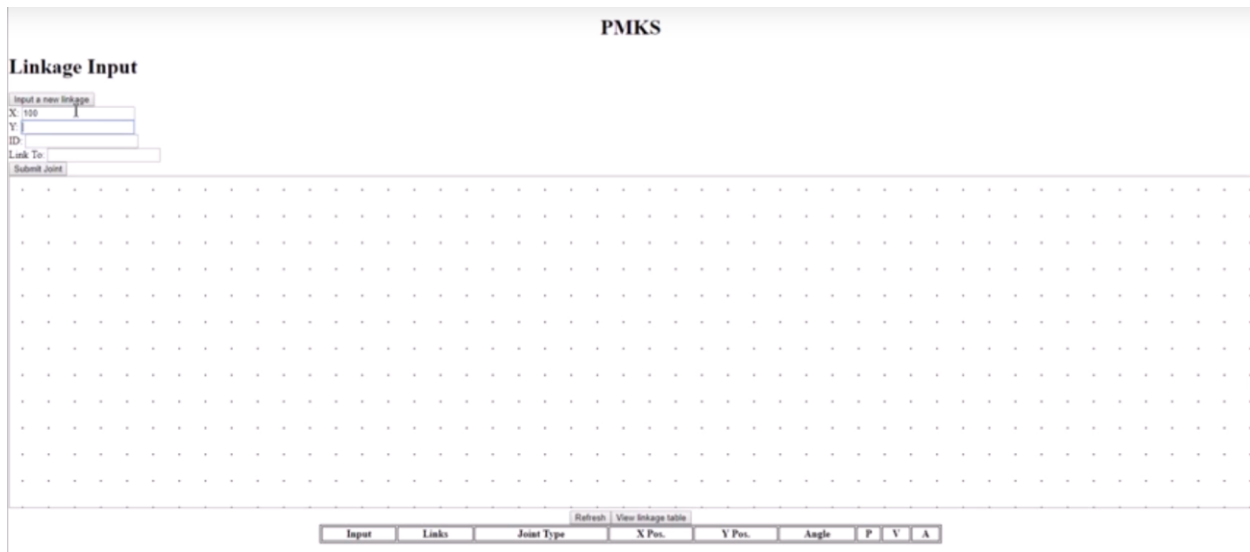


Figure 9.3: First interface design iteration of PMKS+

In this design, we used a bare minimum of color and effects. Cascading Style Sheets had not been used at this iteration. It was composed of only 4 components, which are the header of the page titled “PMKS”, the linkage input form, the grid, and the linkage table. At this iteration, we did not have any rationale for the interface components and wanted to start adding functionalities that existed in the previous PMKS application. During the next iteration and onward, we made significant improvements to the styling and design of elements.

### 9.3 Iteration 2

Figure 9.4 represents our second iteration of the interface. During this iteration, we added more components and some initial styling. Below we describe the changes made in this iteration.

## PMKS

The screenshot displays the PMKS+ interface. At the top, a dark navigation bar contains five tabs: "Linkage Input", "Open file", "Save configuration", "Export Kinematic Data", and "Get as URL". Below this, the "Linkage Input" section is active, featuring a form with fields for "X", "Y", "ID", and "Link To:", along with a "Submit Joint" button. The main area is a large, empty grid of dots. At the bottom, a table header is visible with columns: "Input", "ID", "Joint Type", "X Pos.", "Y Pos.", "Angle", "P", "V", and "A". A "View linkage table" button is positioned above the table header.

Figure 9.4: Second interface design iteration of PMKS+

### Tabs Above the Grid:

These tabs were also a part of the original PMKS. We decided to implement a similar tab system in PMKS+ in order to provide familiarity to previous users. These tabs acted as placeholders for future content. The tabs were simply buttons that would make each of their contents visible. The tabs added were:

- *Linkage input*
- *Open file*
- *Save configuration*
- *Export Kinematic Data*
- *Get as URL*

### View Linkage Table Button:

We decided to add a button to hide the linkage table (located at the bottom of the page). . We felt that seeing the full size of the grid was crucial and having too many joints could make this difficult. While we wanted this table to float on top of the grid (as was in PMKS), we were unable to get it to work this iteration.



### Styling the Tabs and Linkage Table:

During this iteration, we began working with CSS. We added some colouring to the tabs and also added some visual feedback such as highlighting the tab you are currently in, as shown in the tabs in Figure 9.4. This initial styling was added in order to establish an overall theme for our application.

This iteration, we faced some issues with the design. The buttons that made tab content visible would not hide their contents if they were clicked again. If a user wanted to hide the contents of a tab, he would have to refresh the entire page. Another issue we faced was from tab contents being of varying size. This caused certain elements of the interface to shift up and down when cycling between them. This issue was solved in a following iteration.

## 9.4 Iteration 3

In Iteration 3, we decided to add more components and styling to give the interface a theme. In this iteration, we made the most changes so far in order to give a “*personality*” to the application.

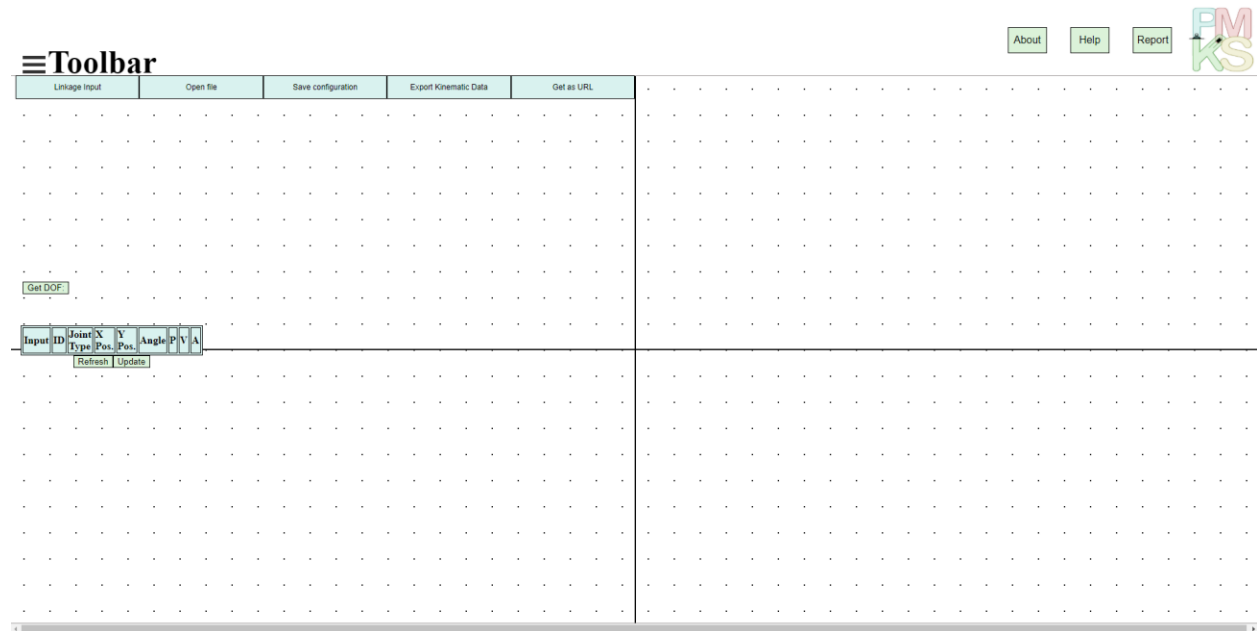


Figure 9.5: Third interface design iteration of PMKS+

Figure 9.5 shows the many changes we made in the positioning of various elements. These changes are described below:

### **Logo:**

In this iteration we decided to add a placeholder logo to the application. A logo is considered one of the most recognizable elements when it comes to a web application since it serves as its identity. When considering popular web pages such as Facebook, Youtube and Twitter, any users can clearly recognize the logo wherever they see it. When a user sees the PMKS logo, they can clearly understand what web application they are using. The logo was taken from the original website of the previous PMKS software.

### **About, Help, and Report Buttons:**

The About, Help, and Report buttons did not exist in the original PMKS. While these buttons did not function yet, we felt it was important to place them in order to get feedback on position and spacing. All three buttons would either redirect you to a new page or provide new functionalities to the application. The About page gives information on version history, authors, and repository access. The Help page would provide help tools, for new users such as visual feedback, step by step actions and a user manual. The Reports page would provide a form where users can fill out bug reports. All three buttons are just placeholders for now and their functionality has not been implemented at this iteration.

### **Toolbar Header and Burger Button:**

We also made many changes to the toolbar this iteration. The dashed lines next to the “Toolbar” title serves as a button to hide the tab system in order to view the grid underneath.

### **Toolbar:**

The toolbar was moved this iteration and placed above the grid. When clicking on a tab, the contents appear floating above the grid. This was done to solve the issue of elements shifting down when tabs were changed.

**Linkage table:**

The linkage table's position was changed to be on top of the grid and made to be always visible. Users may view data relating to the links and joints that they input directly without needing to take extra actions.

**Get DoF Button:**

We also added a "Get DOF" (Degrees of Freedom) button in this iteration. This button displays the degrees of freedom of the current linkage mechanism. The button was placed above the linkage table until we could find a better position for it.

**Theme Colors**

We also added additional coloring for the interface components. The color choices came from the PMKS logo. We found that web pages such as Facebook and Youtube, style their default coloring theme around the colors of their logo.

**X and Y Axis**

We added an X and a Y axis on the grid to help the user navigate the coordinate system more easily. This axis was centered to match a Cartesian coordinate system.

**9.5 Iteration 4**

During iteration four, we solidified many of our design choices, added more components, and added other functions to improve user interaction.

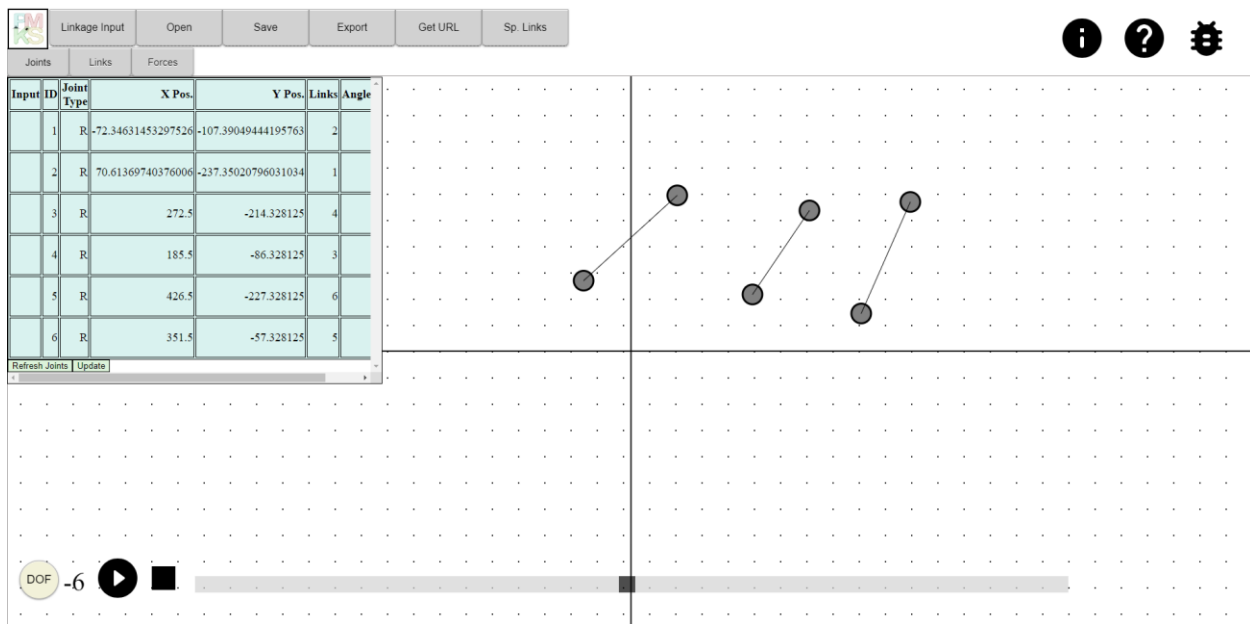


Figure 9.5: Fourth interface design iteration of PMKS+

A major focus during this iteration was the proximity of elements, an interface design principle we discussed in Chapter 4.1 Section B. Items in close proximity to one another should have related functionality. The following discusses the specific changes made to each element:

### Toolbar:

Toolbar has been moved to the upper left side of the screen and is no longer attached to the grid. We made this decision because similar applications such as SolidWorks and AutoCAD include toolbars at the top of the screen. A user from those applications should expect a toolbar to be placed at that position so we followed those expectations as closely as possible. The style of the toolbars has also been changed in order to separate them from other buttons that exist in the application.

### Logo:

In this iteration, we also considered the positioning of the logo. A logo placed in the top center of the screen acts as a title, implying that the user will spend the majority of their time reading while on the page. This style is similar to a newspaper and can be seen on news sites such as the New York Times and the Wall Street Journal. A logo placed to the top-right or top-left implies that the webpage is focused more on interactions. Examples of this include Youtube

and Facebook. The logo was placed in the top left based on cultural expectations of websites. In the west, information is read from left to right and other modeling software follow the practice of keeping their logo in the top left and in-line with the highest-level menu, which in our case is the toolbar. We also added a border to the logo for emphasis, but reduced its size so it wouldn't occupy too much space.

### **About, Help, and Report Buttons:**

These buttons had their text replaced with icons. We felt that easily distinguishable icons could be more visually appealing and help to not overwhelm the user with too much text. The icons used were found at the Material Design website.

### **Tables:**

The table showing all joint information now occupied the left side of the screen and stretched from the top menu to the timeline on the bottom. This design mimics Solidworks' use of a side menu that holds important functionality and information. Users will be able to view information about the grid in the closest proximity to it as possible. Tabs were added to the table to cycle it between joints, links, and forces. The additional tables were added to provide more information about the linkage system created by the user.

### **Animation Timeline:**

The timeline slider, which was previously located on the right side of the screen, was moved to the bottom. This was done to mimic a chronological timeline, which is commonly read left to right. . The default style was used initially and changed during future iterations. At this stage it provided no functionality.

### **Animation Buttons:**

We also added buttons that would play or stop linkage animation. We decided that it would help users control the animation itself, in case they want to make changes dynamically as the animation plays.

## DOF Button

The DOF button was placed in the bottom left alongside the timeline because the two correspond to similar functionality: the animation of the system. A linkage can only animate if its degrees of freedom is equal to 1, therefore it made sense to put the DOF button next to the animation timeline. . The DOF button was displayed through text because we had difficulty representing its functionality through an icon.

## Tooltips

This iteration, we added tooltips, a textual hint that appears when hovering over a button. These were added to help users understand the functionality behind a button without having to click on it. Since we added the tooltips, some words from the tabs have been reduce or removed in order to, as mentioned above, not overwhelm the user with too much text.



*Figure 9.6: An example of a tooltip*

## 9.5 Iteration 5

Iteration 5 was the next large update during our work in C-Term. Many new components were added. Previous components were updated or changed to better scale and improve their functionality. This iteration lasted longer than the previous ones, so the progress made was more significant than any previous iterations.

As can be seen in Figure 9.6, the overall theme of the application has been changed and updated to match our “activity-driven” design.

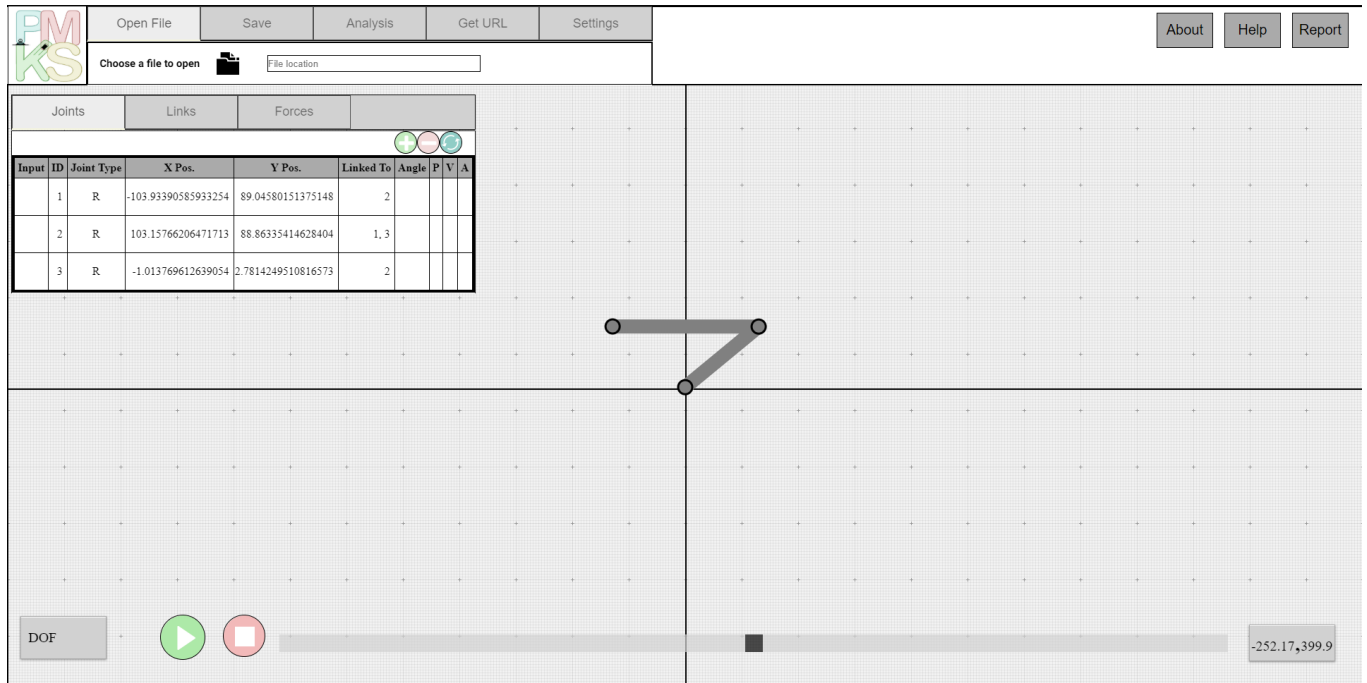


Figure 9.6: Fifth interface design iteration of PMKS+

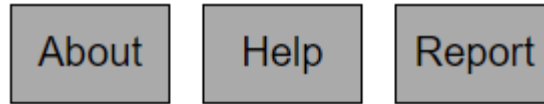
Our main focus in Iteration 5 was to scale the components better and update the design even further than the previous iterations. To achieve this, we needed to slightly redesign all the components on the user interface to function and scale better. We shall go into further detail as to the steps that were taken and the components that were changed and added.

### Logo:

The logo was slightly enlarged and we increased the border around it in order to distinguish it better. Since the name of the application was changed to PMKS+, a new logo needed to be designed. For this iteration we kept the same one. Having a logo return users to a home page is a common practice and seen in websites such as Youtube, Facebook, and Twitter. Since PMKS+ is a single-page application, clicking the logo should therefore refresh the page. We decided to add this functionality to PMKS+ to cater to these user expectations.

### About, Help and Report Buttons:

The About, Help and Report buttons were also changed again. The black color and pictures were really confusing and did not match the overall theme of the website so we decided to revert to our previous design with words and neutral colors (Figure 9.7).



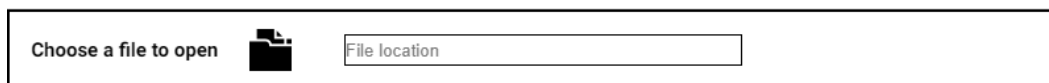
*Figure 9.7: About, Help and Report buttons*

### **Toolbar:**

Scaling and positioning the toolbar was one of the biggest issues we had in previous iterations. Besides the overall design, we had to reconsider the way it was implemented in the code. Before Iteration 5, the toolbar were just buttons to hide or show their contents. The buttons would just change the display of these components into none or block. This was a simple and a fast implementation but it turned out to be hard to scale and even harder to position. We decided to disregard this implementation and use the built-in Material Tabs in the Angular framework. These new tabs performed better in terms of scalability and positioning, and they even had built-in transition animations that made the toolbar feel better when interacting with it. This update to the toolbar also made us think about the “hide” functionality of the tabs. Looking at other toolbar designs from software such as Microsoft Word, we came into the conclusion that the toolbar should always be visible and that the tabs should change between the contents of the toolbar. In terms of style changes, the toolbar tabs were redesigned to feel more like actual tabs. We added more visual feedback such as changes to colors and depth that better indicate to the user which tab is active, which tabs can be clicked, and how many tabs there are. For this iteration, the toolbar components are as follows:

- **Open File tab**

Tab where a user can open a previous saved file (Figure 9.8).

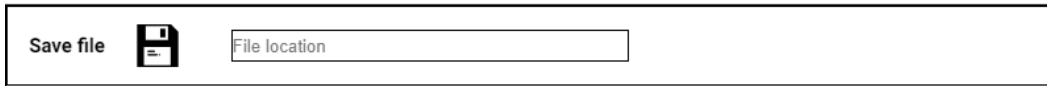


*Figure 9.8: Open file tab*



- **Save File tab**

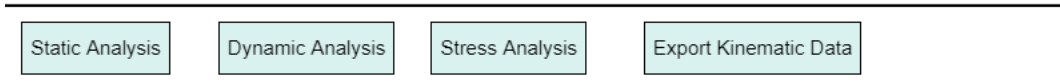
Tab where a user can save the current file (Figure 9.9).



*Figure 9.9: Save file tab*

- **Analysis tab**

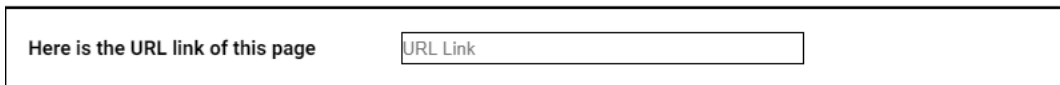
Tab where a user can access the different analyses that can be produced (Figure 9.10).



*Figure 9.10: Analysis tab*

- **Get URL tab**

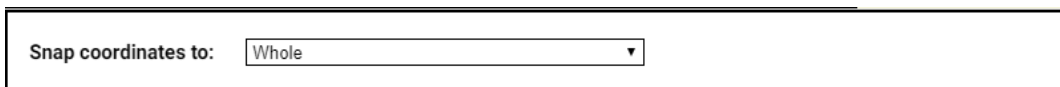
Tab where a user can get and copy the URL link of the current page (Figure 9.11).



*Figure 9.11: Get URL tab*

- **Settings tab**

Tab where a user can adjust different settings for the application. For now there is only one setting that can adjust the units size (Figure 9.12).



*Figure 9.12: Settings tab*

### **Loading animation screen:**

In this iteration, we also added a loading screen to provide more visual feedback to the user (Figure 9.13). The loading screen is a circle spinner that has an animation effect to visualise loading state. The inspiration was drawn from other loading animations that exist in the Windows operating system and in other websites such as YouTube.



*Figure 9.13: Loading screen spinner*

### **DOF textbox:**

In this iteration, the DOF button was changed into a text box. This change was done because we had recently implemented an automatic DOF calculation each time a user modified the linkage. PMKS had this feature and we felt it should be in PMKS+ as well.

### **Animation Buttons:**

The animation buttons were changed to match the theme of the application. This was done by making the play and stop animation buttons round and give them a color similar to the logo. The timeline was left unchanged this iteration.

### **Location textbox:**

Since majority of interactions with the grid were to be done with the cursor, we felt it would help to display the current coordinates of the cursor at the bottom right of the grid. Our goal with this was to improve cursor precision when drawing linkages on the grid.

### **Grid :**

The grid background was changed in this iteration to match the theme of an actual grid. The background before was just dots. For this iteration we added grid lines.

**Table:**

The table was moved slightly further away from the toolbar to separate it since their functionalities are different. The tab system implemented in the toolbar was also implemented here. In this iteration, only the Joints table was implemented because the other two tables didn't have functionality yet. The previous buttons of "clear table" and "refresh joints" were also removed. The refresh joints buttons was removed because joint synchronization between the grid and table was now done automatically. The buttons that remained, which are "add row", "delete row" and "update" were also changed to match the theme of the grid buttons. They were made into rounds buttons with colors that matched the theme and the logo of the application. Furthermore, the table was changed to a more of a minimalistic design without the color that it had before. The font style and sizes were also changed to increase readability.

Following this iteration, we decided that the user interface design was in a good state and a more detailed design critique would be necessary for further improvements. To collect such feedback, we conducted a design critique session with our advisors. During this session, we presented our design to our advisors and went through all the elements of the design one by one. The session produced valuable feedback that was implemented in Iteration 6 of the user interface.

## **9.5 Iteration 6**

This iteration was done after the design critique session in order to further improve the design based on the feedback our team received. After looking back at the basic design rules, we re-examined each element of the interface to make sure they were properly placed and aligned. Iteration 6 was smaller in duration compared to Iteration 5, but resulted in big improvements overall as shown in Figure 9.14.

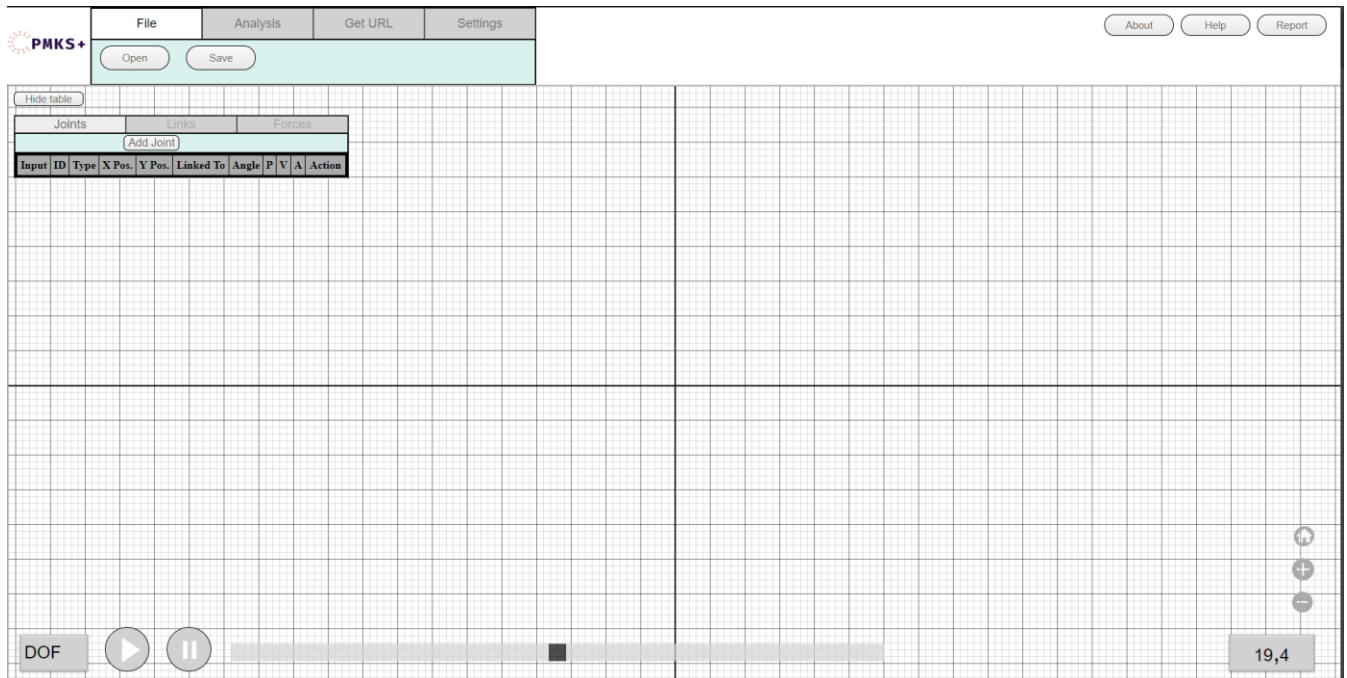


Figure 9.14: Sixth interface design iteration of PMKS+

In this iteration, we redesigned many elements and also added some new elements that we will describe below. Our goal was to change the overall theme once again to match our project standards and our previously established design rules. We also used this iteration to conduct a user evaluation, as described in Chapter 15. In the following subsections, we go into more detail about all the elements that were changed or added.

## Logo

Since our new application is called PMKS+, we felt it necessary to replace the old logo. The new one was still a placeholder and subject to change in a future iteration.

## Buttons

The theme of all buttons were changed once again. We wanted to make sure that users could tell them apart from the tabs so we decided to give them their own design. All buttons were given rounded corners, visual feedback when hovering and clicking, a new font and new size. The only buttons that were different from these were the animation controls.



Figure 9.15: The new button design

### File tab

The previous open file and save file tabs on the toolbar were merged in order to create a new tab called “File”. After some discussion, we decided to merge the “Open File” and “Save File” tabs under a single, unified tab on the toolbar. These buttons had similar functionality and could therefore be grouped to conserve space. We also added buttons to open and save linkage files under this tab. When clicked, these buttons would bring up the native file manager application on the user’s computer.

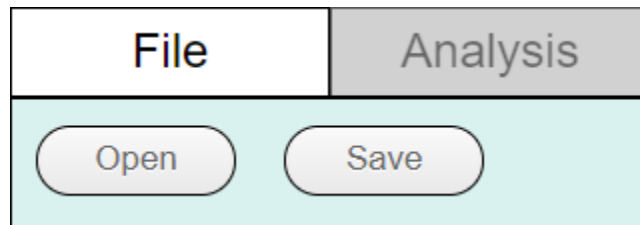


Figure 9.16: The new file tab

### Analysis Tab

In this iteration, we added a pop-up window to display further information about the current linkage. This information appears when clicking on one of four buttons in the analysis tab. Figure 9.17 shows the contents of the tab. Each button corresponds to a different type of evaluation: static, dynamic, stress, and kinematic. In this iteration, these windows were just placeholders and did not yet display information about the current linkage. As shown in Figure 9.18, the popup windows are tables with analysis information. They also include buttons that would allow a user to export the data in the tables. When an analysis window is visible, the rest of the page elements cannot be accessed and are greyed out until a user decides to close it by clicking on the close button at the top right of the page.



Figure 9.17: The analysis tab

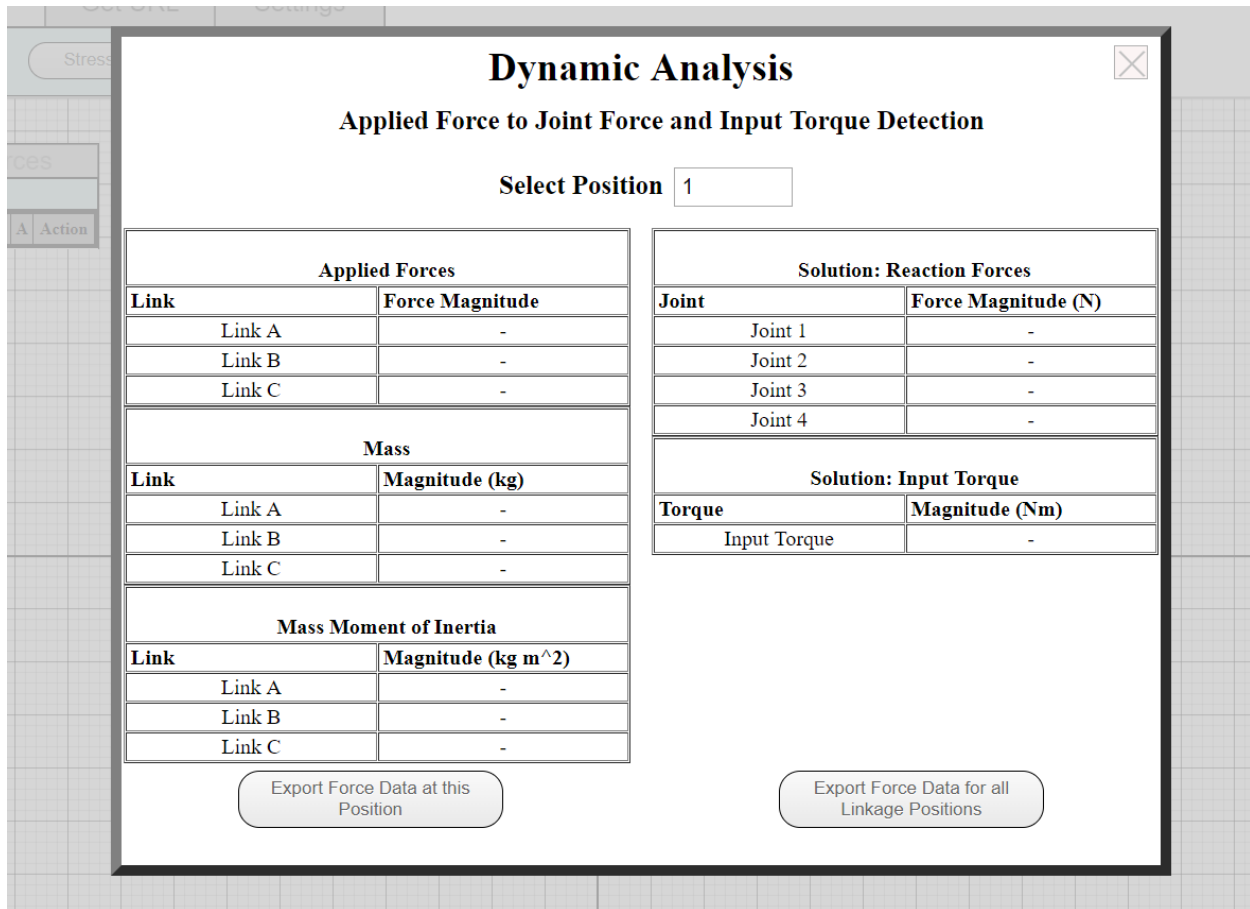


Figure 9.18: The analysis popup window

## Table

The tabs in the table were also resized. This was done to differentiate them from the toolbar tabs since they each serve a different purpose. The Links and Forces tabs were greyed out and disabled because they had no functionality at this iteration and we did not want them confuse users during user evaluations. The previous buttons that existed in the table, besides the add joint button, were also disabled and removed due to bugs in their functionality.

## Animation Buttons

The colors of the animation buttons were changed to a neutral grey. Feedback received before the evaluation indicated that their colors were difficult to distinguish on screen. Thus, we changed them to grey color to blend in with the other elements lining the bottom of the window.

## **Grid Control Buttons**

In this iteration, we also added new buttons for zooming and re-centering. These buttons were placed at the bottom right of the grid because they relate to panning and zooming while the other nearby buttons control animation. A user can now control the zooming of the grid with the plus and minus buttons and can re-center the grid by clicking on the home button.

## **Grid**

The grid lines now properly correspond with the coordinates of the grid, making the grid a more accurate representation of position compared to previous iterations. The axis lines were also given a bigger stroke width to differentiate them from grid lines.

In addition, we also worked on improving the alignment of elements and fixing visual bugs that we found in previous iterations. We made sure that all elements were aligned properly in their groups and that their sizes were appropriate.

## **9.5 Iteration 7**

This iteration was the final iteration of the user interface design. It was done after the PMKS+ user evaluations. As can be seen in Figure 9.19, not many changes were made since users were quite satisfied with the changes to the interface. More details about the evaluation results can be found in Chapter 15. At this point, we were finished with the user interface iterations and no further changes were made to the user interface in this MQP.

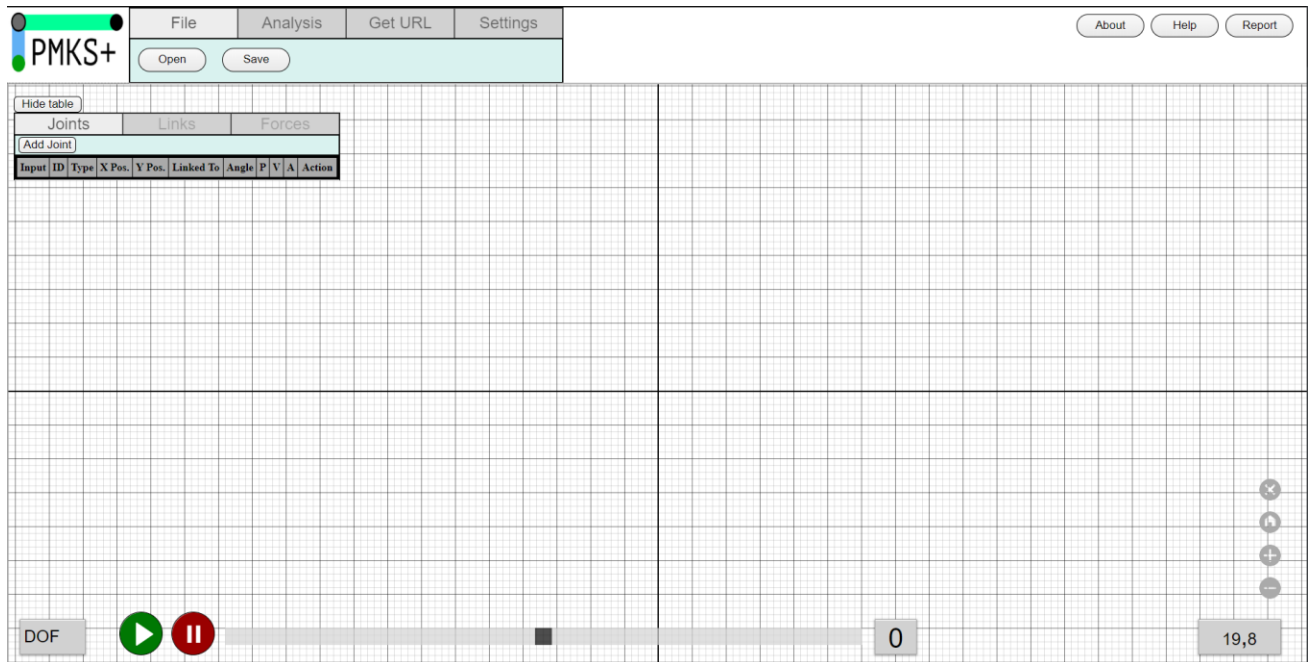


Figure 9.19: Final interface design iteration of PMKS+

In this iteration, we mostly applied feedback received from the PMKS+ user evaluations. These changes included alignment issues, font color, button color and some extra elements. Detail about each change are described in the following subsections.

## Logo

In this iteration we changed the logo to its final design. We decided to change it since our previous logo was not obvious to the user in the application and did not match the design of the application.

## Grid Buttons

The buttons were changed slightly to match in size and color. We also added a new button that clears the grid without refreshing the page.

## Animation Buttons

We added some colors to the animation play and pause buttons. Based on feedback we received, the grey color was implying that the buttons were disabled and blending in with the rest



of the grid elements. Thus, we changed the colors to green and red to that imply that they are usable.

### **Animation Position**

According to feedback from our advisors, it was not clear which animation position a user was currently at in the analysis tab. We decided to add a new text box next to the animation timeline to show the current angle of the animation. With this, users can always see in which animation position they are at and can select it in the analysis window.

# Chapter 10

## Linkage Creation Methods

In this section, we describe the methods of creating linkages, their individual components, and the rationale for their inclusion in PMKS+.

### 10.1 Linkage Table

The linkage table contains three separate tabs for creating elements. Links and joints can be created through their respective tabs by appending a new row and manually specifying the parameters for new elements. Forces are applied to an existing link through their own tab. Using the linkage table is the most precise way to create and position elements because it requires exact x and y values. Adding joints through the linkage table requires a minimum of six actions, one to add the row, and one for each required linkage attribute (id, type, x, y, and connected link).

Joints			Links			Forces				
Add table row		Remove table row		Update canvas						
Input	ID	Type	X Pos.	Y Pos.	Linked To	Angle	P	V	A	Action
	1	R	10	4	3					DELETE
	3	R	367	88	1					DELETE

Figure 9.7: Adding a new joint via the linkage table.

We decided to incorporate this linkage creation method in PMKS+. It added a level of precision to linkage creation that was otherwise not possible. Previous users of PMKS should also find it as the most familiar method when migrating to the new system. Since it does not conflict with any other functionality, there is low risk that the linkage table method will compromise the usage of other methods.

## 10.2 Grid

The grid refers to the section of the interface that displays linkages on a cartesian coordinate plane. Creating joints, links, and forces directly on the grid is a new feature provided through PMKS+. This system was designed to mimic the actions performed when drawing a link by hand. As such, it should feel natural and intuitive to the user.

We initially prototyped three methods of linkage creation: Click-Drag, Click-Click-Connect, and Click-Click. In Click-Drag, a user holds down the cursor at the first endpoint, moves it to the location of the second other endpoint, and releases. In Click-Click-Connect, the user clicks to create individual, unlinked joints. These joints are then manually linked by the user. In Click-Click, users click once to create a joint and again to specify the other. The joints are linked automatically.

Due to technical constraints at the time, it was not possible to have all three methods run simultaneously. Instead, we designated one as the default method, and the others as toggleable options in the user settings. We crafted a list of criteria to rank each method and determine a default. The criteria are:

- *Hand Drawing Similarity*

This refers to the method's similarity to drawing a link by hand. A method that scores high here should feel natural and incorporate the actual hand motions of drawing a link by hand.

- *Efficiency*

Creating linkages is the primary action a user must perform in PMKS+. Reducing the number of actions this takes is essential in creating an alternative to the linkage table.

- *Trackpad Usability*

This refers to the method's usability using laptop trackpads. An advantage PMKS+ has as a web app is that it can be run on any modern computer in any location. Therefore, any actions the user must perform should feel natural and smooth on both laptop trackpads and with a mouse.

Using these criteria, we evaluated each of these linkage creation methods.

### **Click-Drag:**

In Click-Drag, a user holds down the cursor at the first endpoint, moves it to the location of the second endpoint, and releases.

Score:

- *Similarity to hand-drawing: 5/5*

Click-Drag was designed to mimic the actions of drawing a line on paper. When hand drawing a line, the pencil is held down on one endpoint and dragged towards the other, where it is released. This is similar to Click-Drag, where the cursor is held down from one endpoint to the other.

- *Efficiency: 5/5*

Click-Drag is accomplished by a single click to create both joints and a mouse movement to specify the distance between them.

- *Trackpad Usage: 2/5*

A drawback of Click-Drag is that long links need to be dragged long distances. Since laptop trackpads are confined to a small range of motion, it is difficult and sometimes impossible to create these links.

Final Score: 12/15

### **Click-Click-Connect:**

In Click-Click-Connect, links are created by creating two joints and linking them. This method removing dragging so that it could be more usable on trackpad devices.

Score:

- *Similarity to hand-drawing: 2/5*

Click-Click-Connect is very disconnected from hand drawing links. Creating two points and connecting them is not as natural and efficient as dragging from one point the other.

- *Efficiency: 2/5*

A problem with Click-Click-Connect is that it requires a minimum of 4 actions to complete. Two clicks to create the joints, a mouse movement to specify the distance, and an action to manually link the two together.

- *Trackpad Usage: 5/5*

Click-Click-Connect does not involve holding and dragging the cursor. This means that trackpad users can easily place the second joint anywhere on screen.

Final Score: 9/15

### **Click-Click:**

In Click-Click linkage creation, users click once to create a joint and again to specify the other. The process automatically links the two joints, bringing the total number of clicks down to two. Click-Click behaves similar to drawing, just without the obstacle of dragging in order to help trackpad users. The process is detailed in Figure 10.2.

Score:

- *Similarity to hand-drawing: 4/5*

Click-Click functions similar to Click-Drag, just with the drag being replaced by a second click. While the actions are similar, it does lack the feeling of dragging a pencil across a paper.

- *Efficiency: 4/5*

Click-Click requires three actions to complete. One click for each joint and one mouse move to specify the distance between them. The fourth action of linking is done automatically after the second click. These three steps are illustrated in Figure 10.2 below.

- *Trackpad Usage: 5/5*

Click-Click requires no dragging, meaning trackpad users can comfortably place the second joint anywhere on screen.

Final Score: 13/15

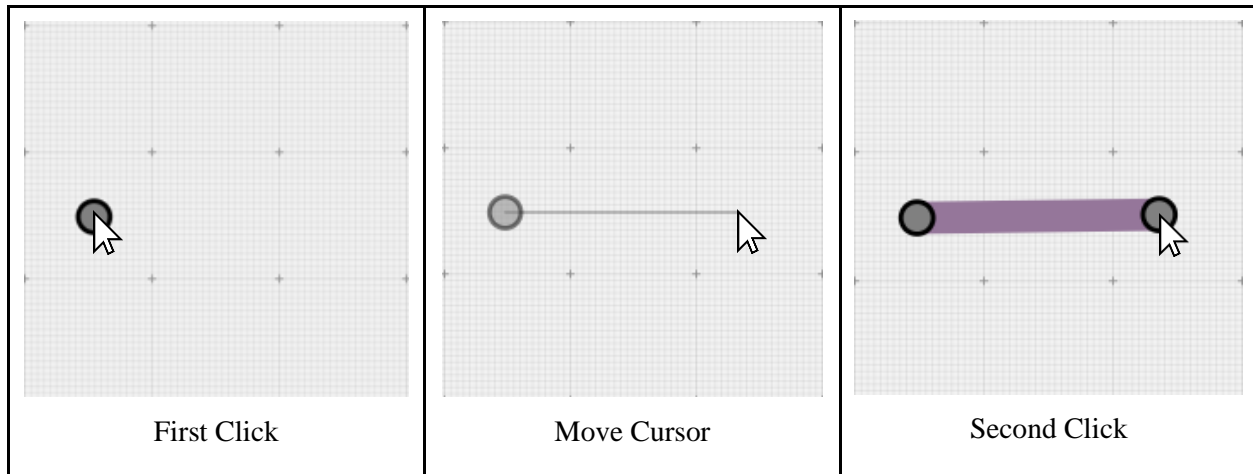


Figure 10.2: The three stages of creating a link in Click-Click

### 10.3 Final Decision

While Click-Drag was difficult to use on a trackpad, we did find it to be the fastest and most natural way to drag links while using a mouse. Therefore, we decided to preserve its code so that it could be used as a toggleable alternative in the future.

Click-Click-Connect scored low in all areas except trackpad usage. The disconnect from hand drawing and the efficiency required made it a very unnatural method. We also discovered that Click-Click-Connect introduced another problem outside of the three criteria being measured. By using Click-Click-Connect, it was possible for the user to leave unlinked joints in grid. This is a critical flaw because the kinematic calculations performed in PMKS+ assume that the system is comprised of only a single linkage. Such an error could lead to incorrect calculations being performed. Due to these flaws, we felt it was necessary to reject Click-Click-Connect completely.

We decided that Click-Click should be the default option in PMKS+ because it proved to be the most usable linkage creation method that worked on both mouse and trackpad. While Click-Drag performed slightly better in efficiency and hand drawing similarity, we felt it was more important to set the method that was most usable on all devices to default.

## 10.4 Context Menu

In this section, we describe the context sensitive menu, a feature in PMKS+ that allows users to perform additional actions from a right mouse button click. The context menu in PMKS+ extends a user's interactivity with the Grid Component. This allows users to edit joint and link attributes directly from the grid. The action can be performed in as few as two clicks (one to open the menu and another to select the action).

### 10.4.1 Advantages

We outlined three advantages that a context menu brings.

- **Straightforward.** The context menu is the most direct way to modify elements of a linkage in the grid. Rather than searching for the joint or link in the linkage table, a user can right click on it in the grid and modify its properties directly.
- **Descriptive.** The context menu textually lists out features it can perform. It is easy for a user to understand what actions are available to them and what each action does.
- **Intuitive.** Many web and desktop interfaces such as Google Chrome and the Windows home screen incorporate context menus. Users expect such functionality to exist on a right mouse click.

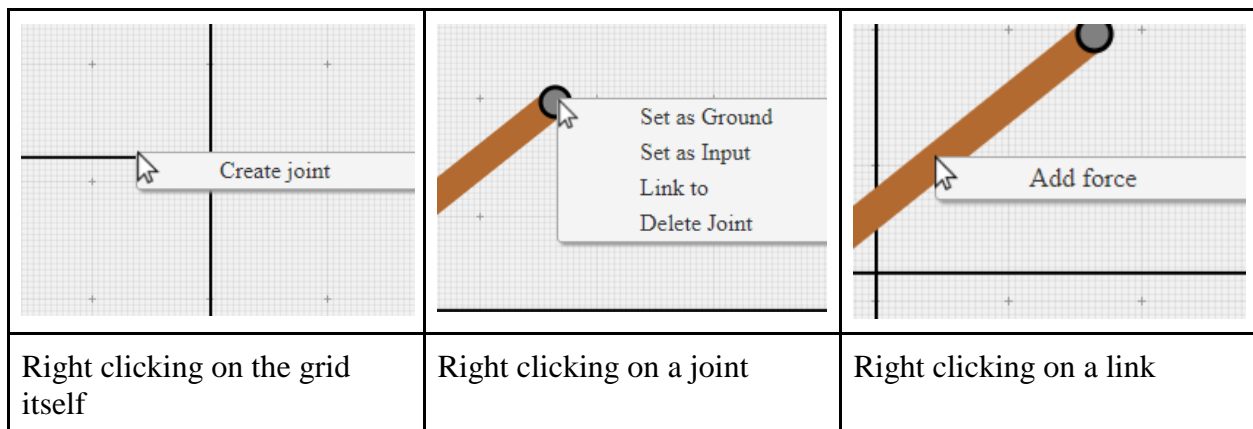
### 10.4.2 Disadvantages

- **Speed.** Because each interaction takes at least two clicks, repetitive tasks can become tedious. Users will have to click at least twice to complete each action. Hotkeys are much better for performing frequent tasks. Hotkeys can be held down meaning that the minimum number of clicks required is only once per task.
- **Simplicity.** Features provided through the context menu should be very simple. Complex actions such as setting a joint's precise x and y coordinates are better performed in the linkage table.

### 10.4.3 Activities Bound to the Context Menu

Based on the strengths and weaknesses outlined above, we decided to bind the following actions to the context menu:

- **Creating Joints:** User feedback on the Click-Click linkage creation method indicated that users were unintentionally clicking on the screen and creating links. By binding link creation to a context menu, we are minimizing this error by requiring them to confirm the action through a click. A downside to this is that link creation will require two more clicks than what was previously possible. We felt that this change was necessary to prevent accidental link creation. It also frees up the mouse click to perform other features such as screen panning.
- **Deleting Joints:** Deleting a joint is a one time action. All the data required for the deletion process can be gathered by the initial right-click on a joint.
- **Ground/Input Toggle:** Setting a joint as a ground or input joint is a simple toggle action. Since there can be many grounded joints in a linkage, a hotkey could also exist as an alternative for this action.
- **Linking Joints:** The context menu is able to trigger only the start of the linking state. The action of selecting a second joint to link to follows the “Click-Click” style for connecting joints.
- **Adding Forces:** Adding a force requires 2 clicks. The first click sets its coordinates and the second one sets its magnitude and direction. Similar to linking joints, the context menu is used to trigger the first action. .



*Figure 10.3: The context menu when interacting with three different objects in the grid.*



# Chapter 11

## System Design

### 11.1 Analysis of Existing Application

In this section, we examine the design and structure of the PMKS source code and its relation to the structure of PMKS+.

#### 11.1.1 What is Microsoft Silverlight?

PMKS was developed using the Microsoft Silverlight framework. Silverlight is a deprecated web framework used to develop internet applications. Silverlight applications are accessed via an internet browser and enabled through a plugin. Silverlight leverages C# and XAML to provide an alternative to the standard JavaScript and HTML web development environment. XAML allows for rich media visuals and C# enables powerful, type-enforced computations. Silverlight applications are structured around the Model-View-ViewModel design pattern. This pattern separates user interface (View) and data (Model) while still allowing for intercommunication via a translation entity (ViewModel) (Stieglitz, 2017).

#### 11.1.2 Code Structure

The PMKS Source code is structured around three components. The User Interface, Simulator, and Model. Each of these components are called through the MainPage class in PMKS (As seen in Figure 11.1).

- The **User Interface (UI)** refers to the visuals displayed on screen to the user. This includes the coordinate grid, animation timeline, and any other user input fields.
- The **Simulator** is a set of mathematical functions used to calculate degrees of freedom, movement curves, and all other kinematic analysis. These functions are all abstracted into a single Simulator Object that PMKS can call upon when necessary.
- The **Model** refers to the data structures used to store all user defined data. This includes the forces, joints, links and materials of the linkage.

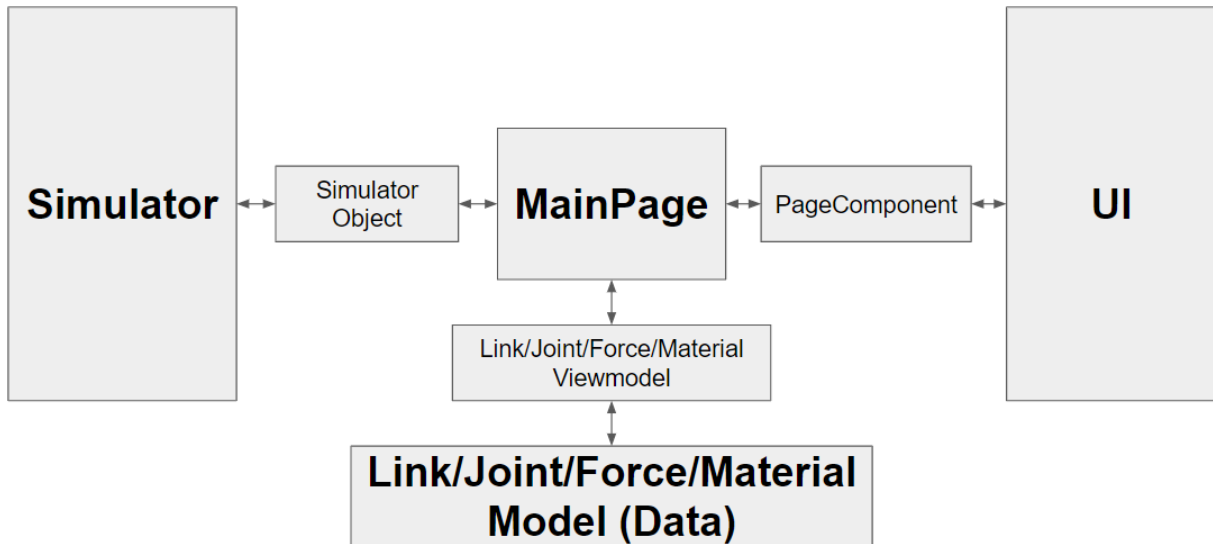


Figure 11.1: A diagram showing the three main components in PMKS and their parental relationship towards MainPage class.

### 11.1.3 Code Flow

Whenever the set of on-screen linkages are modified, PMKS will execute a set of steps to calculate and reanimate the linkages. These steps are:

1. Save the new data to the Model
2. Call the Simulator object to check the degrees of freedom of the linkage.
3. If it returns 1, ask the simulator to calculate the full movement of the linkages and calculate the curves each joint moves along. Else, end the function and draw the static linkages on screen.
4. Draw the movement curves as lines on the coordinate grid (UI).
5. Animate the linkages by moving them along the movement curves.

## 11.2 New System Architecture

In this section we describe the system architecture behind the PMKS+ application.

### 11.2.1 Design Pattern

PMKS+ follows the Model-View-Controller architectural design pattern, In Model-View Controller, software components are organized by functionality. Components in the View directly control what is shown on screen to the user. These include the Grid, LinkageTable,

Toolbar and AnimationBar components. Components in the Model are responsible for storing persistent data and logic in the program. The Model is represented by the Simulator Class, which stores the linkage data and runs all kinematic calculations. Components in the Controller mediate the flow of data between the model and all other components. The MainPage also acts as this controller by using a 2-way data binding, (a data synchronization system provided by the Angular Framework), to synchronize the array of joints and linkages between components. This means that any time the user modifies the linkage in one component, the change will immediately be reflected in all other components. This is especially important when synchronizing data between the LinkageTable and Grid component.

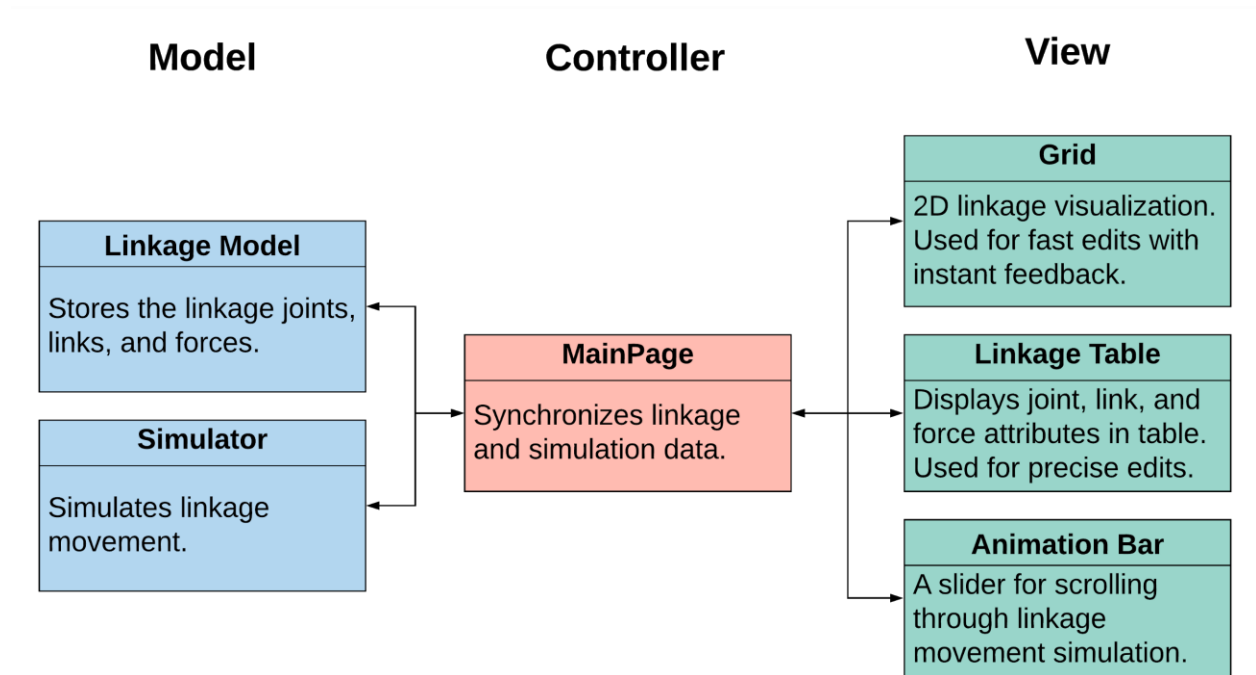


Figure 11.2: PMKS+ Component Diagram. Components are grouped by their functionality in the Model-View-Controller design pattern

### 11.2.2 Primary Classes

There are two main data types stored in PMKS+: Joints and Links. Every instance of these classes created by the user are defined in the MainPage component via an array.

- A **Joint** represents a single point of connection between two or more links. It contains the following five attributes:
  - *Location*: x and y coordinates
  - *Links*: A list of different links the joint is connected to.
  - *Type*: The type of joint (revolute, prismatic, pin-in-slot, or gear).
  - *Ground*: A boolean that indicates whether the joint is connected to the grounded link.
  - *SVG*: A reference to the graphic image of the Joint on screen.
- A **Link** represents the connection of two or more Joints. It contains the following two attributes:
  - *Joints*: An array representing the number of joints that make up the link.
  - *SVG*: A reference to the graphic image of the Link on screen.

### 11.3 Grid Component States

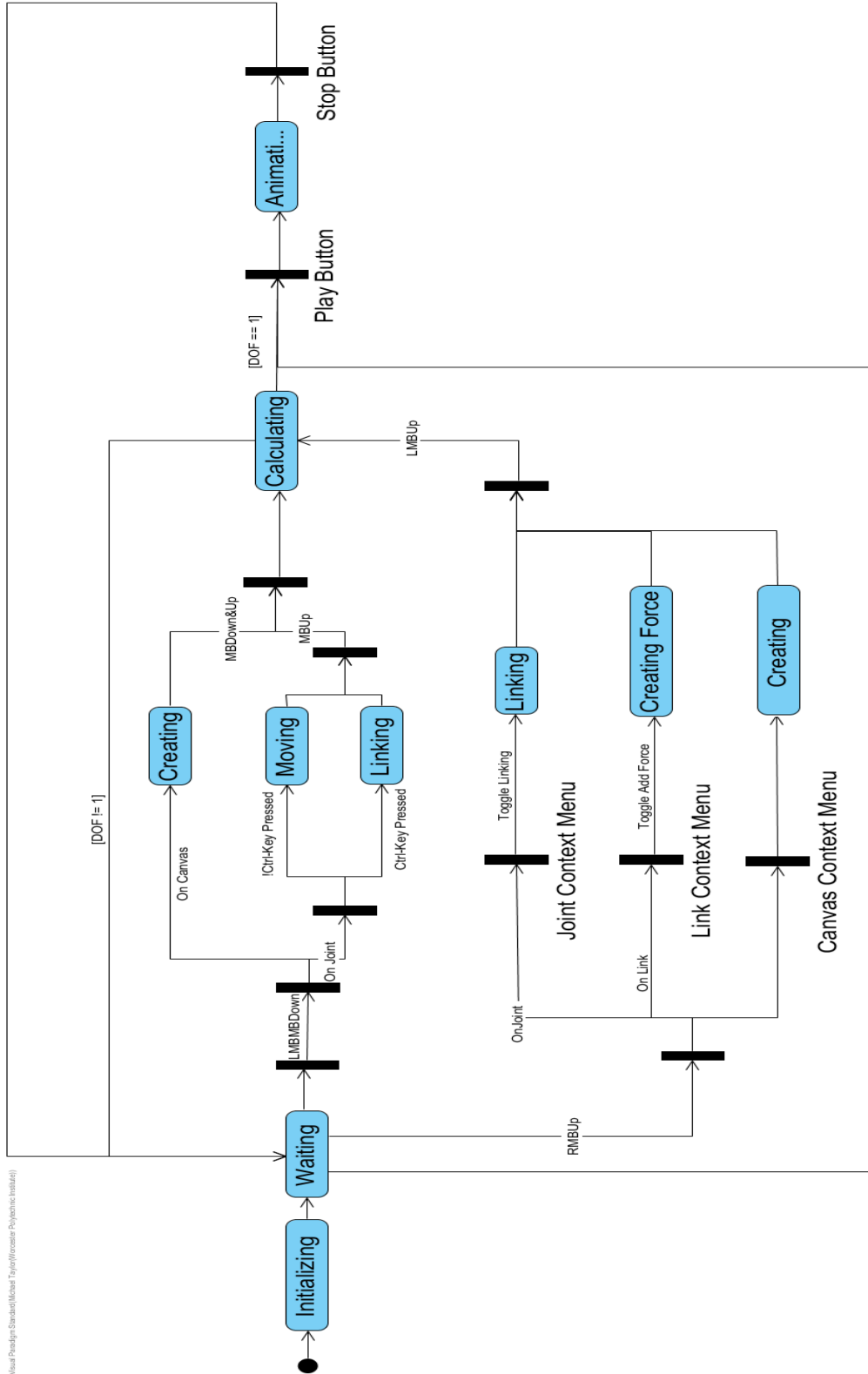
Overlapping user input binds became an issue as functionality in the Grid Component expanded. An example of this is the way the left mouse button interacts with joints. When a user presses the left mouse button down on a joint, they can drag it across the grid to change its location. However, if the user had started creating a link (with click-click link creation), pressing on a joint would instead create a link between the joint being created and the joint clicked on. Complex user input conditions such as these needed to be more readable and organized in the code. And since more functionality was going to be added, it would only become more difficult to track which actions were bound to which button and if any new actions would cause overlap.

To overcome this obstacle, the code in the grid component was refactored to incorporate states. Each state would limit the amount of functionality a user could perform at a given time, making it difficult for overlapping actions to exist. To model this, we created a state diagram (figure 11.4) and mapped out each potential state and the actions it takes to get there.

#### 11.3.1 Benefit to the Developers

When a user interacts with the grid, our new design checks which state the program is in and runs the corresponding function. For instance, if a user clicks on a joint, the program will check the state. If it is in the waiting state, the program will run the function to begin moving the

joint. If it is in the linking state, the program will instead run the function to link the clicked on joint with the previously selected joint. This refactor improved the readability of our code and also helped ensure that we were not developing features with overlapping user input.



Visual Paradigms Standard (Michael Turner/Worcester Polytechnic Institute)

Figure 11.3: The state component diagram for the Grid Component.

### 11.3.2 Benefit to the User

States are represented by visual indicators to the user. These indicators help the user understand what functionality is available to them without exposing the inner workings of the code. For example, the linking state is represented by a thin preview line connecting the user's mouse cursor to the joint they are linking from. This line signals to the user that their next action will be connected to the joint on the other end of the preview line. Figure 11.4 demonstrates this signal as shown to the user.

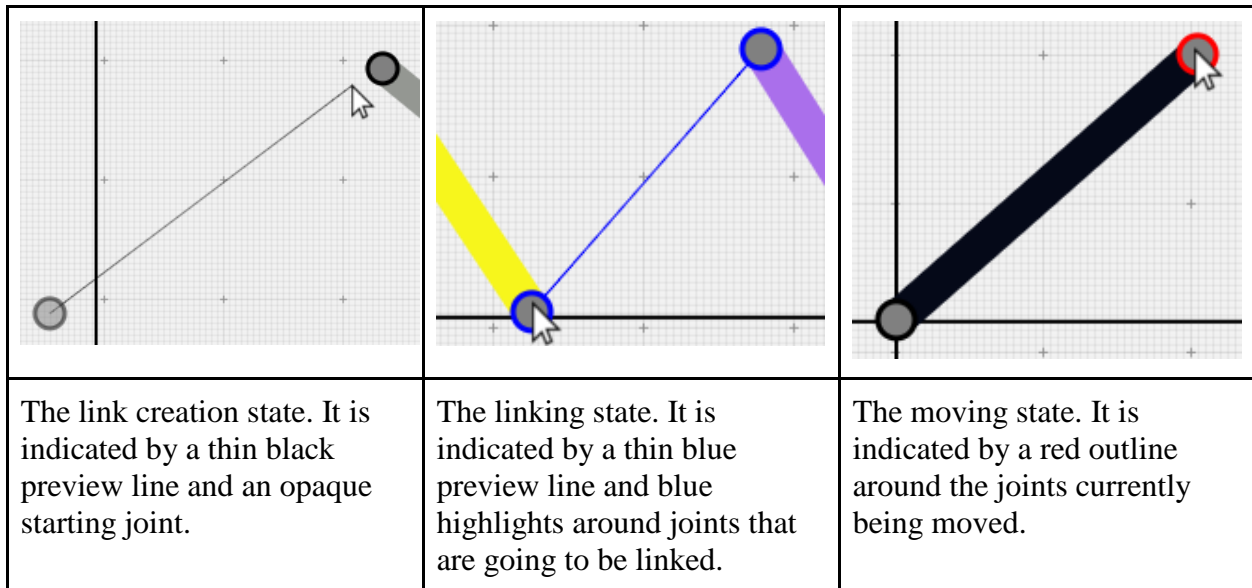


Figure 11.4: Visual hints of states for users.

## Chapter 12

# Simulator Class

PMKS utilizes a standalone Simulator Class to perform kinematic analysis. Such analysis includes finding the degrees of freedom and the full movement of linkages. Once calculated, these analysis values are sent to the PMKS+ interface for linkage animation, or directly to the user for manual kinematic analysis.

### 12.1 Simulation Loop

The simulation loop is the main function executing during linkage movement calculations. A properly formed linkage rotates links either partially or fully around each grounded joint in the system. The simulation loop runs an algorithm that determines the position of each joint every 16.7 milliseconds (each of these millisecond intervals is referred to as a time step) during its motion and saves its output in a Time-Sorted List object. By rendering the linkage's position every 16.7 milliseconds we were able to run the linkage animations at 60 FPS.

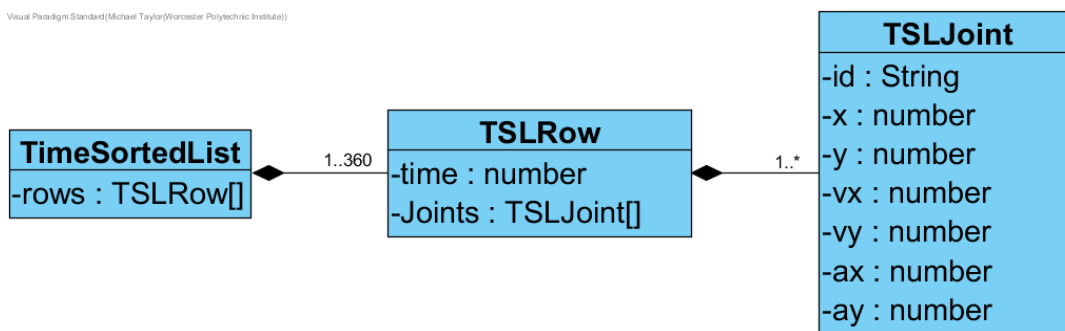


Figure 12.1: Object Diagram for the Time-Sorted List.

A Time-Sorted List is a matrix comprised of rows of positional data. Each row (referred to as a TSLRow in Figure 12.1) contains a time step and a list of each joint in the linkage (referred to as a TSLJoint). Each TSLJoint stores a joint's id, position, velocity, and acceleration at the current time step. Depending on the range of a linkage, a Time-Sorted List may contain 1 to 360 TSLRows. This represents the 360 time-steps required to calculate a full 360° rotation.



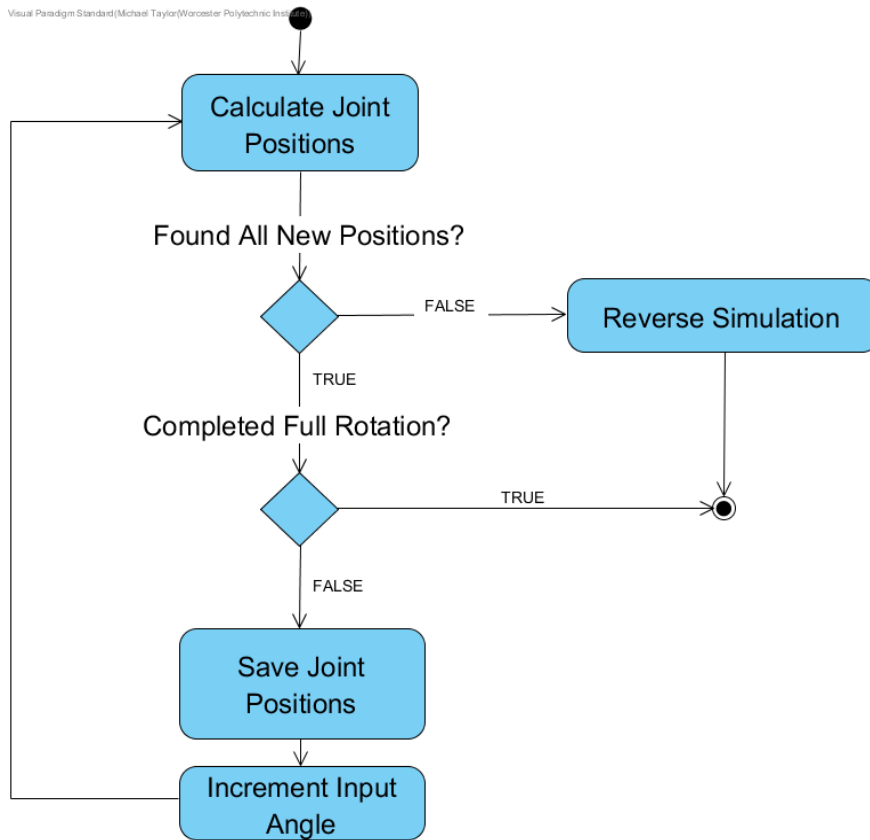


Figure 12.2: Activity diagram for the simulator loop.

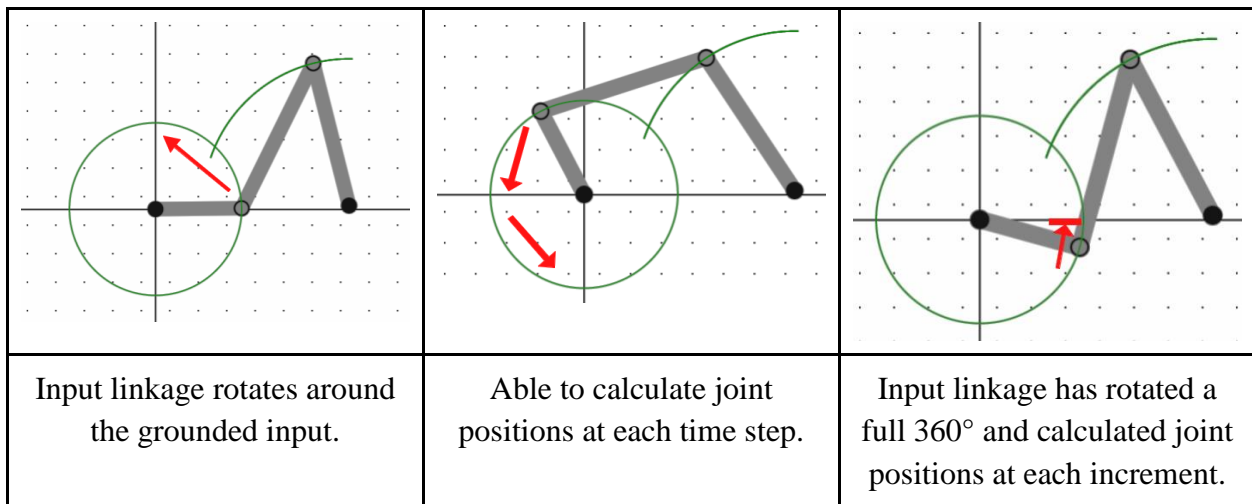
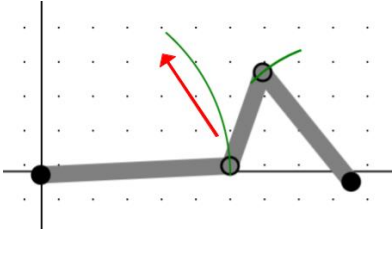
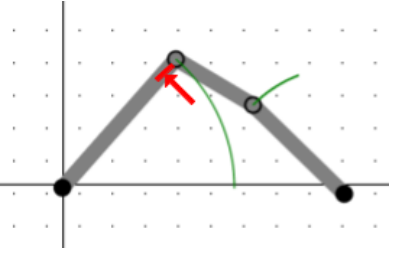
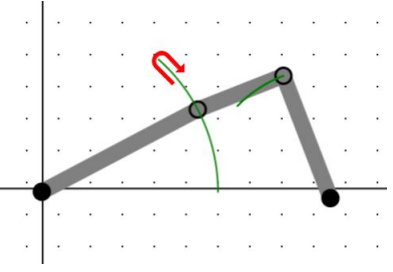


Figure 12.3: Scenario A for simulator termination.

		
<p>Input linkage rotates around the grounded input.</p>	<p>Linkage is unable to move further in the current direction.</p>	<p>Simulation reverses until reaching the starting position.</p>

*Figure 12.4: Scenario B for simulator termination.*

For each time step of the input linkage, the Simulator will attempt to calculate all joint positions. Figure 12.2 details the activity of the simulator at each time step. If calculations were successful, it will save the locations of each joint to the output matrix, increment the angle rotation, and continue. The simulation will terminate when either:

- A: The input linkage has made a full 360° rotation. (Figure 12.3)
- B: An input linkage is unable to rotate any further. (Figure 12.4)

In the former case, the simulation will end because all joint positions have been calculated. For the latter, the linkage cannot go further and can only move backwards. In order to simulate a full range of motion, the simulation will reverse direction until it reaches its starting position. Since the new joint positions will be the same as they were going forwards (only in reversed order), the time sorted list will append a reversed copy of itself.

## 12.2 Joint Traversal Algorithm

To calculate each joint's position, the Simulator visits every joint in the linkage and attempts to derive their new position using one of four techniques.

1. Grounded joints never move and are always located at their starting position.
2. Joints connected to the input joint are solved by rotating themselves one degree around the input.

3. Joints connected to at least two known joints can be solved using circle intersection (described in Section 12.3)
4. Joints not connected to two known joints can be solved using the non-dyadic position solver. This equation was included in PMKS but was not implemented in PMKS+ during the course of this project.

### **Joint Traversal Steps**

Before running through the algorithm, we move each joint into the unknown joint array. This array keeps track of which joints have yet to be solved. Since grounded joints do not move, we can begin by moving all grounded joints out of this array and setting their current position to their starting position. Next, we examine each joint connected to the grounded input. The new positions of these joints can be solved by rotating their position one degree around the input. Now we begin looking at the remaining unknown joints. If a joint is connected to two known joints, it can be solved using the circle intersection solver. If not, it is moved into a new array for ignored joints. Once all unknown joints are set as solved or ignored, we run the algorithm again on the ignored joints. This loop will continue running as long as a single joint is able to be solved at each time step. This process is outlined in Figure 12.5.

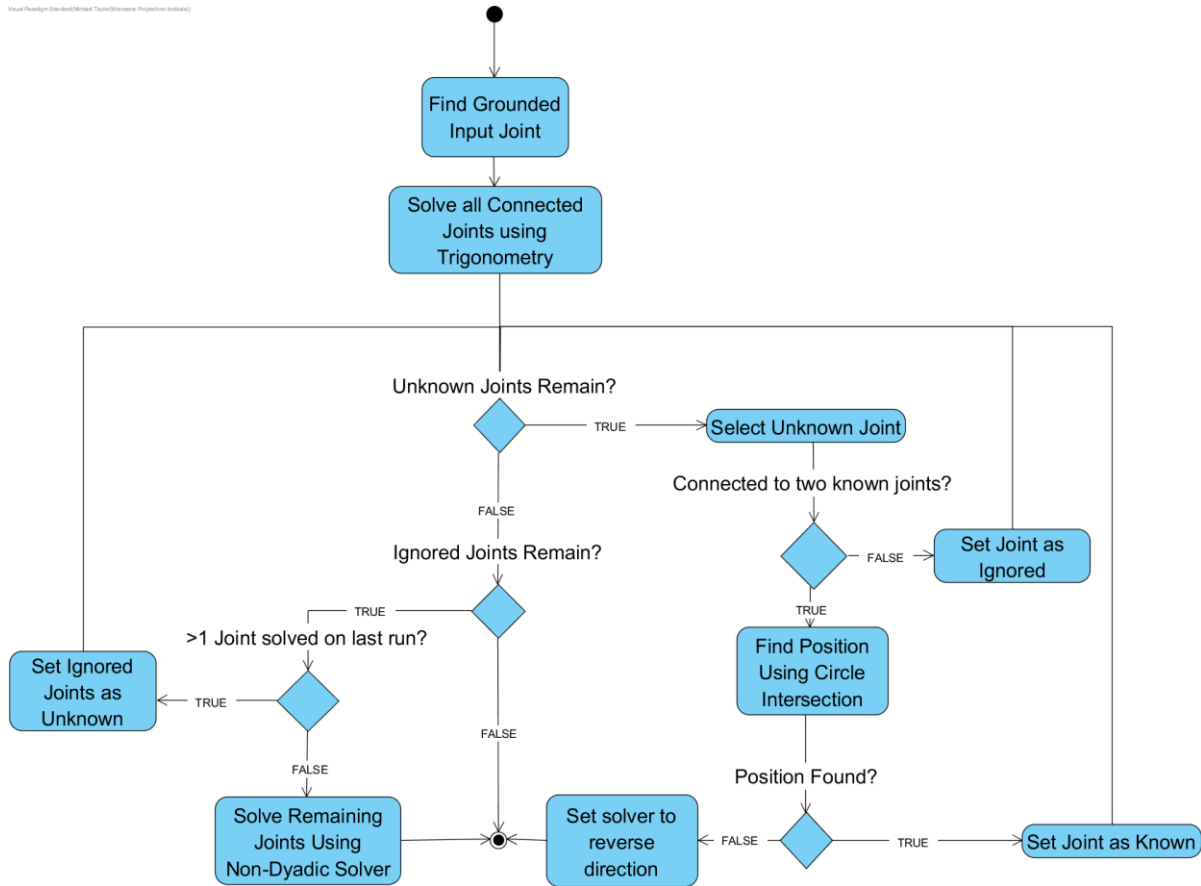


Figure 12.5: Joint Traversal Activity Diagram

Figures 12.6 a-b demonstrate the process using a 6-bar linkage with multiple grounded and ungrounded joints. If the position of any joints cannot be determined during a time step, a special, “non-dyadic” solver is used to find the remaining joints.. In PMKS, this solver was very slow and computation intensive, so it is only run if the joints could not be found via the other methods. Due to the complexity of this solver, it deferred as a task for future developers of PMKS+.

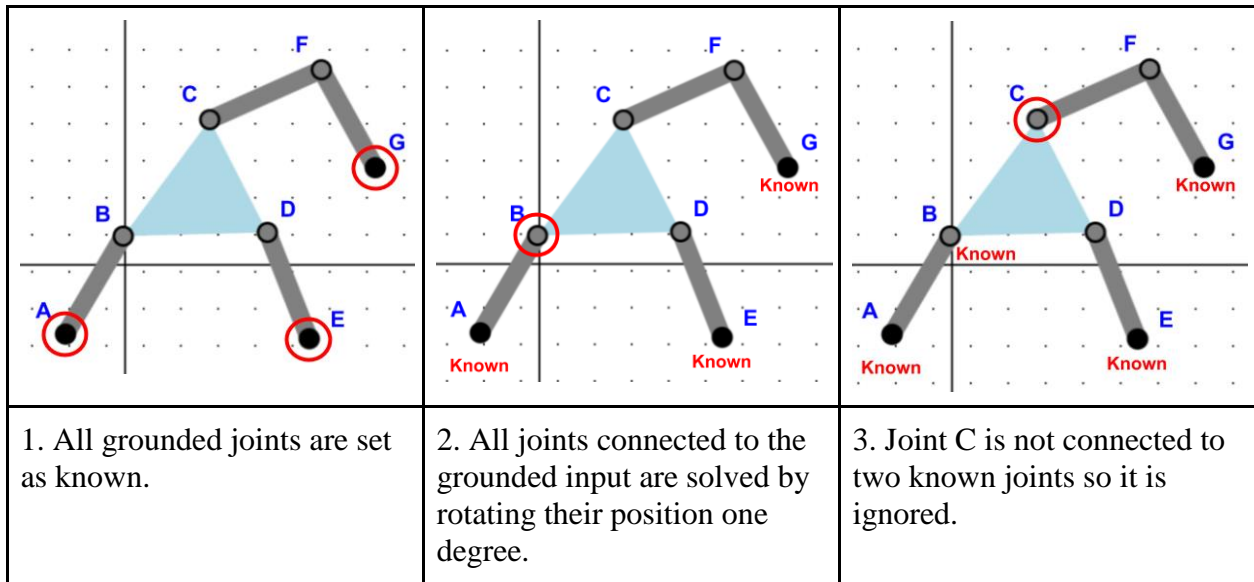


Figure 12.6-a: Example of the Joint Traversal Algorithm on a 6-bar linkage.

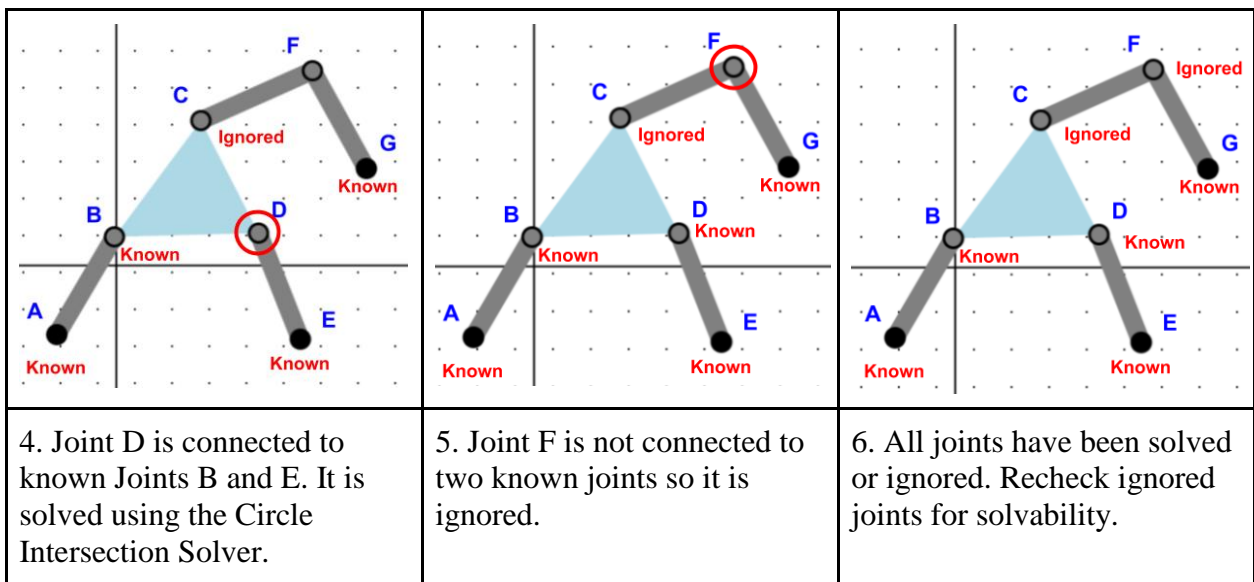


Figure 12.6-b: Example of the Joint Traversal Algorithm on a 6-bar linkage.

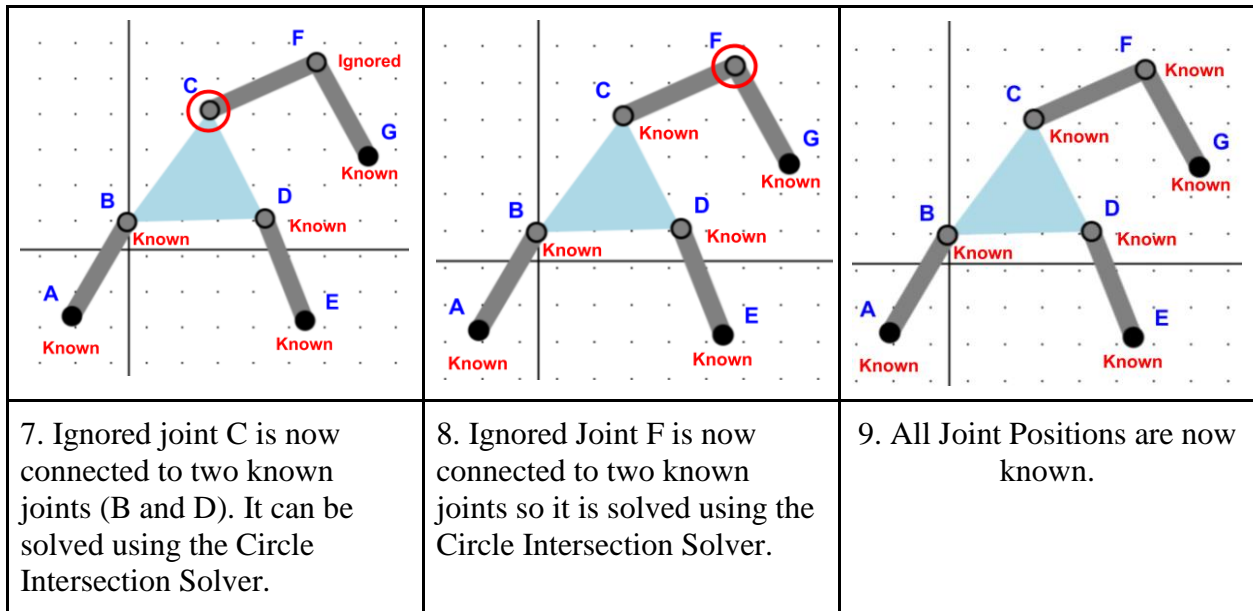


Figure 12.6-c: Example of the Joint Traversal Algorithm on a 6-bar linkage.

## 12.3 Circle Intersection Solver

If a Joint is connected to two or more Joints whose positions have been solved at the current time step, the Joint's new position can be derived using the Circle Intersection Method. In this method, circles are drawn around the two known joints. The radius of each circle is the length of the linkage connecting the known Joint to the unknown one. The point of circle intersection that is closer to the unknown Joint is the unknown Joint's new position. If there is no circle intersection at all, the linkage has reached its maximum rotation in its current direction and must reverse.

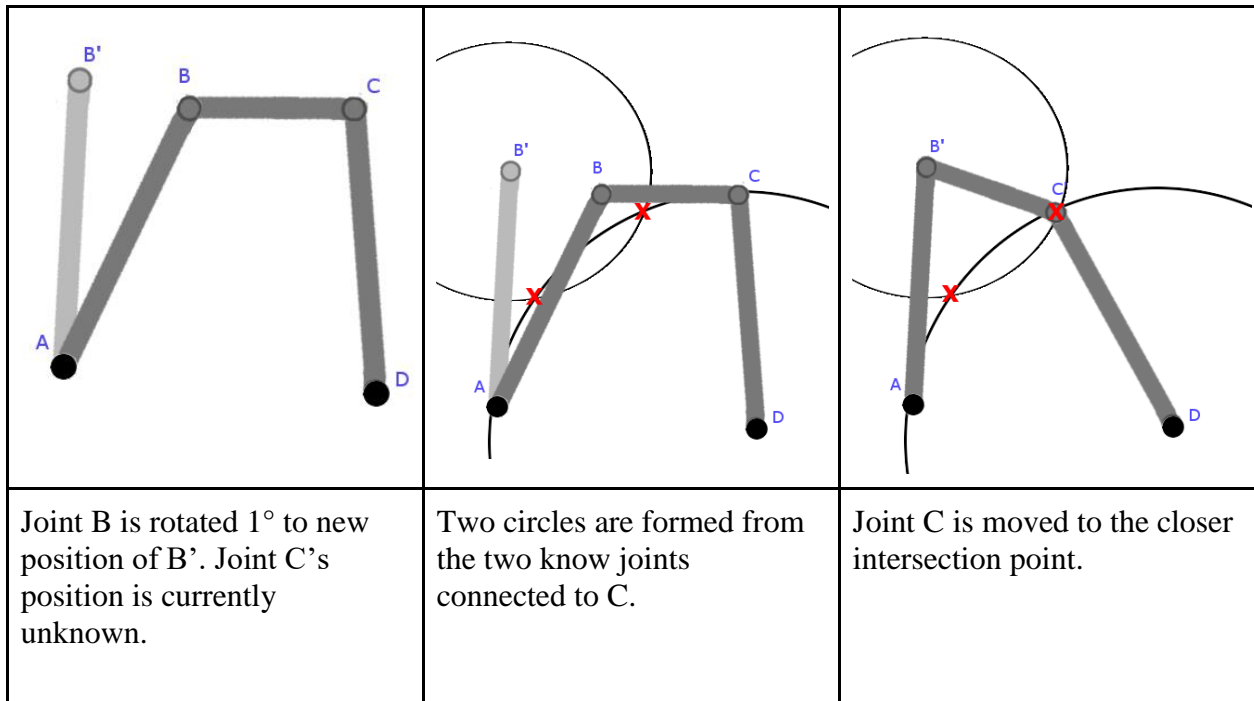
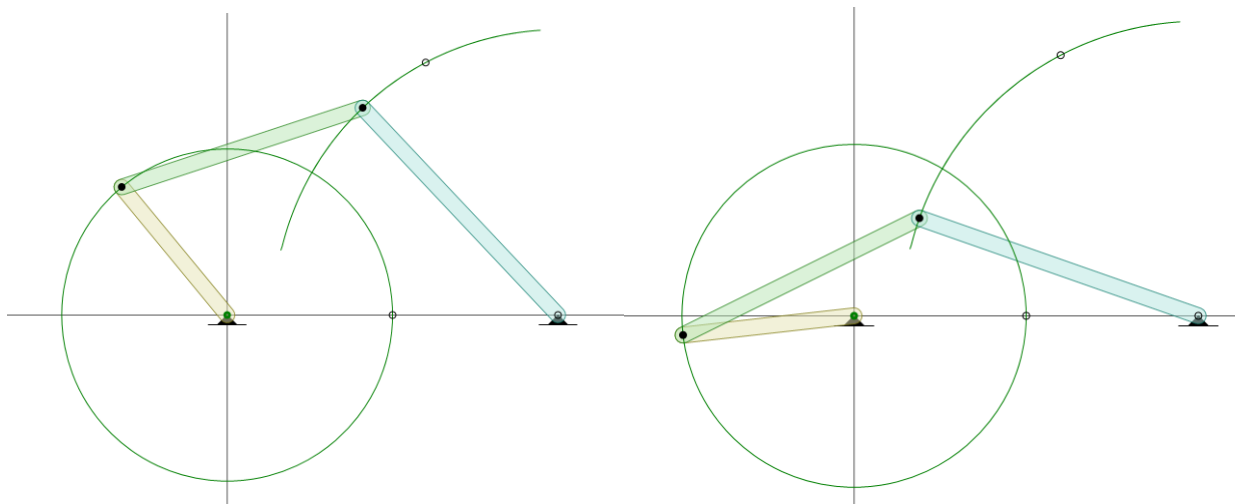


Figure 12.7: Example of using the intersection of two circles to find the new position of Joint C.

## Chapter 13

# Animation

PMKS featured animations which moved links along a path for a set period of time and repeated that movement until the user paused the animation. Figures 13.1 and 13.2 show two frames in the animation of a four-bar system. The ungrounded endpoints of each link move across the drawn lines while the grounded points remain at their respective locations.



*Figures 13.1 and 13.2: Two instances in the animation of a four-bar system provided by PMKS.*

Based on the functionality of PMKS, our team created a list of requirements that PMKS+ must accomplish in order to adequately animate linkages. This list includes:

- Moving the endpoints of a single link along two separate paths for a period of time.
- Drawing the rotation curves that each ungrounded endpoint moves along.
- Pausing and manually advancing the animation.
- Produce smooth animations based on the output of the simulator.

This led our team to consider several options for animating our cross-browser system. The main options included utilizing CSS animations by defining keyframes and creating custom animation functions to move each SVG element. In addition, our team researched the built-in SMIL animations which act on SVG. The option we chose needed to fit each of the requirements



described above. For this reason, we decided against using SMIL animations. Although they were built to handle SVG, this functionality could not animate based on the coordinate positions produced by the simulator. Our group chose to attempt to implement CSS animations before ultimately creating our own custom functionality.

## **13.1 CSS Animations**

The first method our group attempted to implement was CSS animations using an imported jQuery library. CSS animations include keyframe definitions which specify points in time during an animation where an object would be located. Using each of these points, the system moves the object to each location while interpolating the in-between points in order to create a smooth animation. CSS keyframes provide a means to create animations based on specific points in time that circumvents the asynchronous behaviours of Javascript and Typescript. Normally these keyframes are static but our system required the dynamic creation of the coordinates of each link based on the user's input.

### **13.1.1 JQuery.Keyframes**

jQuery.Keyframes is a jQuery library that provides functionality to dynamically create and assign keyframes to elements with CSS stylesheets. SVG elements contain their own stylesheet and may therefore have their own specific keyframes. jQuery.Keyframes also includes functionality to pause and update keyframes, further incentivising its use. Our original plan was to use the coordinates produced by the Simulator Class for each link to define and attach keyframes to their SVG stylesheets. We would then update these stylesheets each time a joint or link is changed by the user. The problems with this solution came from the interactions between CSS and SVG elements as well as discrepancies between the imported library and our Angular environment.

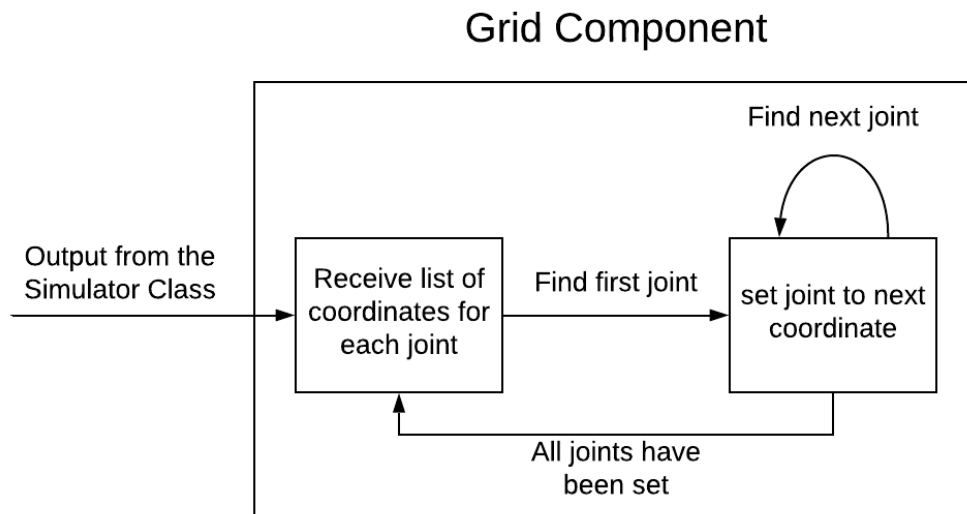
### **13.1.2 Issues with CSS Animations**

Although CSS animations use keyframes to define states of an object at a certain point in time, they cannot act on the properties of objects in the same way. When defining the keyframes of an SVG element, the x and y positions are required for movement. Inputting these values will move the entire element to the specified coordinate. When specifying a keyframe in PMKS+, we needed only a single endpoint to move to the given location, while the other endpoint would

move to a separate location. This requires editing the properties representing each coordinate in the SVG, something that CSS cannot do. It is possible to produce something similar by combining translation of the element with rotation, but the math required to produce the animation in a dynamic environment produces a far greater problem. In addition, the jQuery.Keyframes library is incompatible with Typescript. Importing the library's functions results in unresolved references that cannot be fixed. These problems led our team to approach animation through a different method.

### 13.2. Custom Typescript Solution

The final option for animating links in a way which meets the established requirements was to create the functionality using Typescript. Designing our own functions allowed our team to create animations that perform exactly as needed, though with much more development time and effort required. This solution also creates the possibility for further improvements to the animation system including obtaining data about elements at specific points in time.



*Figure 13.3: The animation process after the Grid Component receives the output of the simulator class*

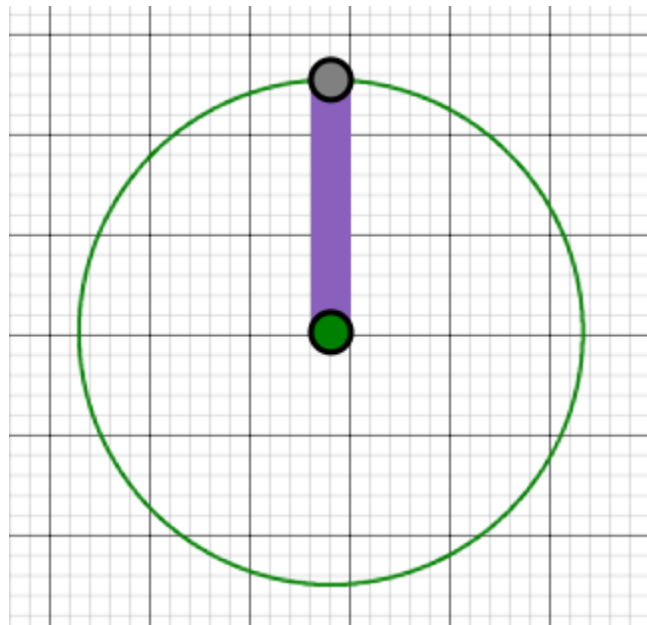
As shown in Figure 13.3, the animation process begins with a list of the coordinates that each joint in the system moves through while in motion. Each joint is then relocated based on its next position in the list. This relocation is repeated until the list is exhausted, by which point the joints have returned to their original positions. The entire process then repeats with each joint's list restarting at their initial coordinates and the joints are relocated in the same manner. Although each step in time is not interpolated, the animation is made smooth by providing a small enough interval between relocations to draw the links at 60 frames per second.

This process uses the output of the simulator class (discussed in Chapter 12) as the positioning data for each joint. Once the simulator has determined how the linkage moves, the animation simply displays these results to the user. This process also handles when linkages reverse their direction. In these situations, the simulator output includes the reversed positions of each joint in the proper order. The animation only needs to loop through each time step in order to properly show reversing.

Drawing each link at a set rate required making the naturally-asynchronous Typescript work in a synchronous manner. This involved creating a timeout function which changed the position of a link, waits a set period of time, and then repeats to move the link to the next position. Although the code succeeds in animating multiple links based on the provided coordinates for each endpoint, the process is very intensive and involves several instances of a recursive function running indefinitely. If the animation is allowed to play forever in this way, the browser will eventually crash. The solution to this problem is to play the animation recursively until it returns to its starting position and then run the same function again, thus resetting the recursion. An alternative to forcing the asynchronous function to run in this way is to play the animation for a certain amount of cycles and then stopping until the user chooses to start again.

In order to initially test the validity of our animation functions, our team used exported data from PMKS as inputs for the linkages in our current system. This data included the coordinates for each link's endpoints at a 0.0167 second interval (60 per second). As previously discussed in Chapter 11, the Simulator in PMKS+ produces the same position values that are calculated in PMKS, therefore this data acted as a viable substitute preceding the accurate output of the PMKS+ Simulator.

Drawing the curves which simulate the movement of each endpoint of a link involved a similar process to moving the individual endpoints themselves. The system creates a new SVG for each curve and places them onto the grid on a layer beneath the links and joints. The endpoint of a link and its rotation curve share a common path so in order to create the SVG, we provided the same inputs used to animate the connected link. Each curve was created as a path element which took the input series of coordinates and connected them using lines. Figure 13.3 shows the rotation curve of a joint in an animated linkage.



*Figure 13.4: A single animated link, the top joint rotates around the input joint on the given circle.*

Our Typescript animations provided the functionality to accomplish our objectives in animating linkages. When comparing the visuals between PMKS and PMKS+, there is very little difference between the quality of animation. In the future, the next improvements that should be made to the system include highlighting key positions in the animation and optimizing performance when large numbers of components are being animated.

### **13.3 Animation Controls**

As previously discussed in Chapter 9, the PMKS+ interface includes an animation component with several controls. These controls include a play, start, and slide input, as shown in Figure 13.4.

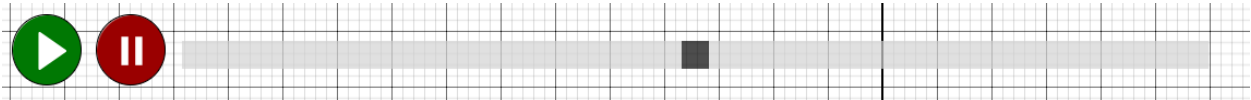


Figure 13.5: From left; the start, pause, and slide input animation controls.

The play and pause buttons set the state of the animations. Pressing play will start the animation of a linkage only if, for each joint in the system, the simulator has calculated a degrees of freedom of one and returned valid positional data. The pause button will halt the animation whenever it is playing at its current position, as shown in Figures 13.5 and 13.6, and pressing play again will make the animation reset. The animation will also halt when a user adds to the linkage, repositions an existing joint, or makes any other changes to the system.

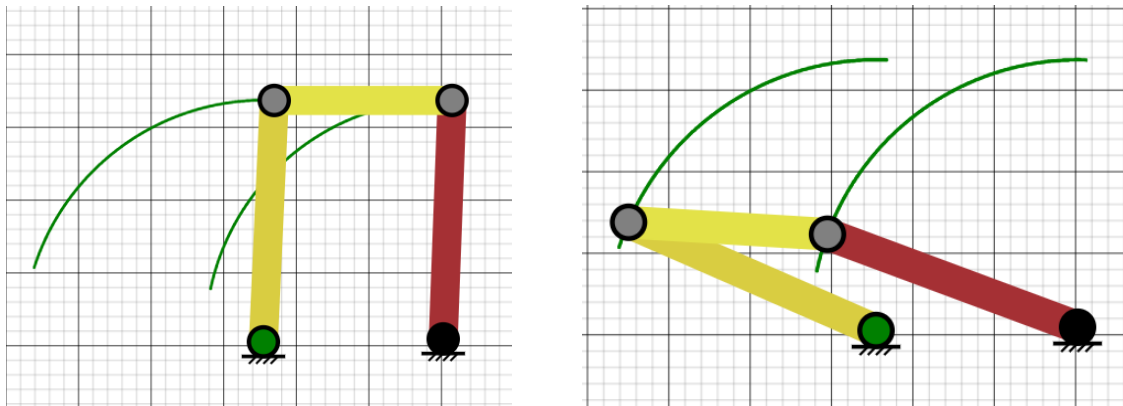
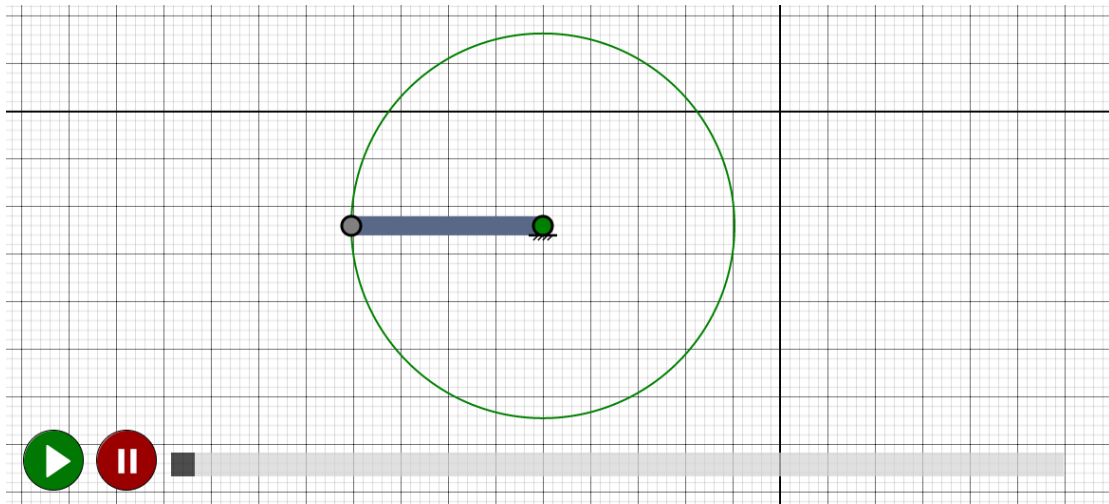
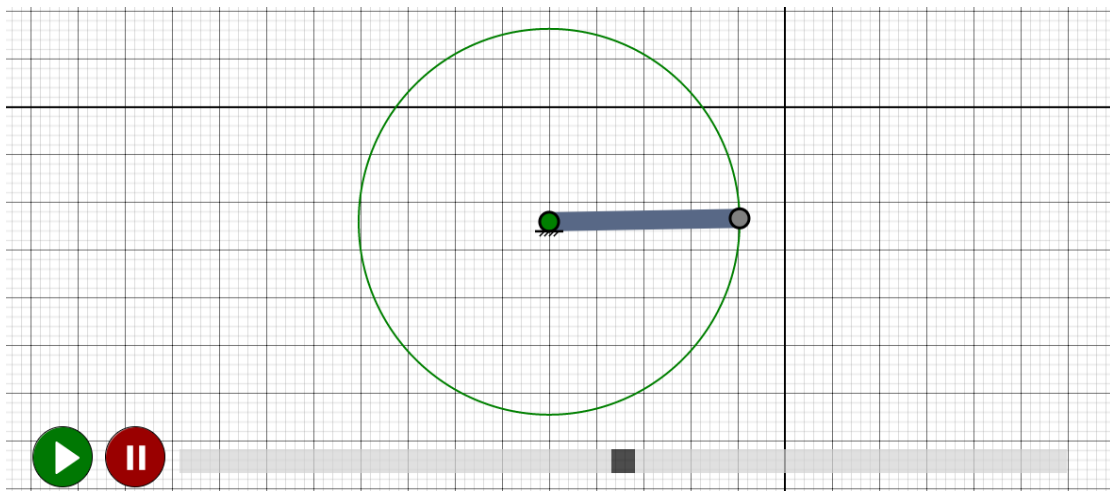


Figure 13.6 (Left) and Figure 13.7 (Right): The linkage moves from its initial position when the play button is pressed, shown left, and stops when the pause button is pressed, shown right.

The slide input allows the user to control the set time of the animation. When the simulator calculates that the linkage has a DoF of one, setting the slide bar to any position will also set the linkage to its positioning at a corresponding angle. The bar takes in an input of 0 through 1 and multiplies this number by the largest angle of the animation. As an example, if the animation includes 300 different angles and the user sets the slider to 0.5 then the linkage will be set to its position at the 150th angle. Figure 13.8 demonstrates the slide input set to 0 degrees while Figure 13.9 shows the same animation set to 180 degrees.



*Figure 13.8: The animation of a linkage set to 0 degrees using the slide input.*



*Figure 13.9: The same animation as in Figure 13.7 set to 180 degrees using the slide input.*

## Chapter 14

# Application Testing

In this section we discuss all the testing our team conducted with PMKS+.

### 14.1 Browser Compatibility

In this section, we go into detail about the tests we performed to check for browser compatibility for PMKS+. The browsers tested were Google Chrome, Mozilla Firefox, Microsoft Edge, Opera, Safari and Internet Explorer. The tests we conducted were:

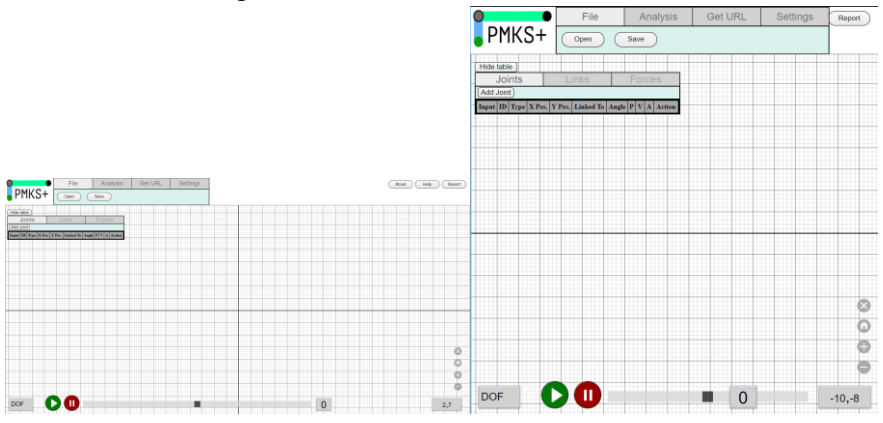
1. Load up the application.
2. Click on all elements, such as tabs and buttons and get the expected result.
3. Make sure that elements of the interface scale to fit the space, so that they are all at their intended positions when the browser is in full window mode.
4. Draw on the grid.
5. Animate a simple linkage to make sure animation works.

In the following subsections, we go into more detail about each browser test. For each browser, we did not stress test all possible drawings or animations. We tested a four bar linkage and its animation to see how it performed in different browsers and checked to see if there were compatibility issues. Further testing was conducted and is discussed in details in section 14.2.

#### 14.1.1 Chrome

Since the application was primarily developed for Google Chrome, the tests we conducted for this browser were all successful. Since this test was our most successful one, we decided to use this test as our standard for comparison with other browsers.

## Google Chrome Compatibility Test



Test steps	Compatibility test comments
<b>First</b>	The application loaded and goes through the loading animation. All interface elements loaded as well.
<b>Second</b>	After selecting all elements, we concluded that elements worked as expected.
<b>Third</b>	<p>The application scaled appropriately in full window size. There were some issues when the window size becomes too small as some elements disappeared. For full window size, the scaling was as expected in 1920x1080 resolution of 100% scaling (Windows OS). As you can see in the two pictures below, the About, Help and Report buttons hid behind the toolbar when the window size changed.</p> 
<b>Fourth</b>	The drawing worked as intended. All drawings showed up and the grid mouse interactions worked as intended
<b>Fifth</b>	Animation worked as intended. All animations were smooth and quick.
<b>Conclusion</b>	PMKS+ is fully compatible with Google Chrome.

### 14.1.2 Mozilla Firefox

Mozilla Firefox was the second browser we decided to test for compatibility. We faced a lot of issues with it as can be seen below from the comments on each step.



## Mozilla Firefox Compatibility Test

Test steps	Compatibility test comments
<p><b>First</b></p>	<p>Application loaded. All elements loaded apart from the grid X and Y axis. Some elements had a slightly different styling like the animation timeline, as can be seen in the pictures below. After further development, the X and Y axis were fixed.</p> <p><b>Chrome:</b></p>  <p><b>Firefox:</b></p> 
<p><b>Second</b></p>	<p>All elements, besides the grid context menu, worked as intended. Mozilla Firefox was not detecting the right clicking on grid in an older version therefore the context menu was not appearing. In a new version the context menu appeared so we could test its compatibility</p>
<p><b>Third</b></p>	<p>Everything scaled as intended in full size window same as Google Chrome but the browser had issues when run with a smaller window size.</p>
<p><b>Fourth</b></p>	<p>We could not draw in the grid using the mouse since the context menu could not appear in an older version. In a newer version of Firefox, the drawing worked but the Y-axis was reversed so all drawings would appear in the opposite y-axis position.</p>
<p><b>Fifth</b></p>	<p>Since we could not draw in the grid, we could not test the animations in an older version. In a newer version of Firefox, animations worked as intended.</p>
<p><b>Conclusion</b></p>	<p>The application loaded up and scaled, but the grid was not fully compatible with Mozilla Firefox in older versions. In newer versions, PMKS+ was compatible with the browser but has issues with the Y-axis translation and all drawings are reversed. This was due to SVG height and width properties being defined in a different format than in other browsers. This was fixed with an updated Mozilla Firefox browser version.</p>

### 14.1.3 Opera

Opera (Version 60) was the third browser we tested and was as successful as Google Chrome.


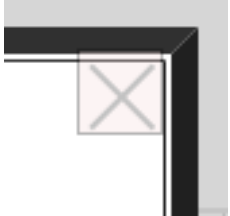
#### Opera Compatibility Test

<b>Test steps</b>	<b>Compatibility test comments</b>
<b>First</b>	Application loaded but had a slight delay compared to Google Chrome. All elements loaded as well
<b>Second</b>	All elements worked as intended.
<b>Third</b>	The application scaled as intended. Opera had same issues with smaller window sizes like Chrome
<b>Fourth</b>	Drawing worked as intended and was similar to Chrome.
<b>Fifth</b>	Animation worked as intended.
<b>Conclusion</b>	PMKS+ is fully compatible with Opera Version 60 and is the second fully compatible browser.

### 14.1.4 Microsoft Edge

Microsoft Edge (Version 44) was the fourth browser we tested in our browser compatibility tests. Microsoft Edge proved to be as successful as Chrome and Opera.

## Microsoft Edge Compatibility Test

Test steps	Compatibility test comments
<p><b>First</b></p>	<p>Application loaded similarly to Google Chrome. All elements loaded as well. We noticed some differences in alignment.</p> <p><b>Chrome:</b></p>  <p><b>Edge:</b></p> 
<p><b>Second</b></p>	<p>All elements worked as intended.</p>
<p><b>Third</b></p>	<p>The applications scaled as intended. There were some issues with the close window button on the analysis windows. The same window issues applied here as well.</p>
<p><b>Fourth</b></p>	<p>Drawing worked as intended.</p>
<p><b>Fifth</b></p>	<p>Animation worked as intended. The animation slider performed slightly different from other browsers. The slider, instead of going only forwards from 1 to 100 percent, goes backwards to 1 when it reaches 100 and then forward again. This is likely a style of sliders specific to Microsoft Edge..</p>
<p><b>Conclusion</b></p>	<p>PMKS+ is fully compatible with Microsoft Edge Version 44. For each test, Edge performed similarly to Chrome.</p>

### 14.1.5 Internet Explorer

Internet Explorer (Version 11) was the fifth browser we tested for compatibility. PMKS was only working in Internet Explorer. Unfortunately, PMKS+ does not load up at all in this browser and hence, all the tests failed. Since official support for this browser is going away in January 2020, we do not need to ensure its compatibility for PMKS+.

#### Internet Explorer Compatibility Test

Test steps	Compatibility test comments
First	Application did not load and was stuck at the loading animation.
Second	Since application did not load up, this could not be tested.
Third	Step could not be completed.
Fourth	Step could not be completed.
Fifth	Step could not be completed.
Conclusion	PMKS+ is not compatible with Internet Explorer Version 11.

### 14.1.6 Safari

Safari (Version 12) was the last browser we tested for compatibility. While some features worked, there were a lot of issues with the grid as mentioned below.

## Safari Compatibility Test

Test steps	Compatibility test comments
<b>First</b>	Application loaded all elements.
<b>Second</b>	The application did not calculate the correct mouse coordinate values for the y-axis. This created an unexpected result when creating a link through the context menu or when dragging joints. The direction of vertical panning was also reversed. Joint and link creation methods through the table worked correctly. All other elements worked as intended.
<b>Third</b>	The application scaled as intended. Safari had the same issues with smaller window sizes similar to Chrome.
<b>Fourth</b>	Due to the error with the mouse coordinates, links created through the context menu are created in the wrong location. Joint and link creation through the table work as intended.
<b>Fifth</b>	Animation works as intended. Animations are smooth and quick similar to Chrome.
<b>Conclusion</b>	PMKS+ was only partially compatible with Safari Version 12 because of the unexpected behavior of the grid. The browser passes a negative value used in the function for calculating screen coordinates. This issue was corrected after the test was conducted and the application now works on Safari as intended.

## 14.2 Software Testing

In this section, we outline the various methods of software testing that were used for PMKS+.

### 14.2.1 Unit Testing

Unit testing is a type of testing that ensures that functions at the lowest level are returning correct values. Unit testing is run in a standalone interface-less environment that mirrors the application environment. All interaction is coded in a separate test file that creates the application and runs through test cases sequentially. Because there is no interface, unit testing is

fast and can be run quickly to ensure new functionality does not break previously developed systems (Shore & Warden, 2008).

Our program utilized the Jasmine test framework, the official Angular testing environment. In this framework, Angular creates specific testing files for each component created. Each testing file mirrors the environment created in the regular application (Angular, 2018).

An example of unit testing in our application is with joint creation and movement. In this test, we create a joint at (1, 1) and call the joint move function to move the joint up two units and to the right three. Afterwards, we test to ensure that the joint is now located at (4, 3).

### **14.2.2 Integration Testing**

Integration testing is a type of software testing that takes the individual components tested in unit testing and tests the interaction between each other (Shore & Warden, 2008). Each integration test should be limited to testing a single action (i.e. creating a joint, moving a joint, etc.). There are three main approaches for Integration Testing:

- **Bottom-Up:** Bottom up testing tests the lowest levels of the program first before moving to higher, parent-level components. This is advantageous if errors are more likely to happen in child components.
- **Top-Down:** Top down testing tests the highest levels of the program first and then moves to lower levels. Top-Down is the opposite of Bottom-Up approach and should be used if errors are more likely to occur in parent components.
- **Big-Bang:** Big-Bang testing tests the entire application at once. Big bang is the most time consuming approach and is only used if components are tightly coupled together.

(Software Development Fundamentals, n.d.)

For PMKS+, we utilized a bottom-up testing approach. We chose bottom up testing because all user interaction is performed in child components. The parent component (the MainPage) is invisible to the user and only exists to synchronize data between other components. Because of this, most sources of error originated from a child component so logically, we decided to check these first.

The Jasmine testing framework was also used to ensure that correct values were being recorded before being sent to separate components. These tests can be viewed as checkpoints that ensure correct data has been gathered before being sent and allows us to identify points of failure quickly. This process is detailed in Figure 14.1.

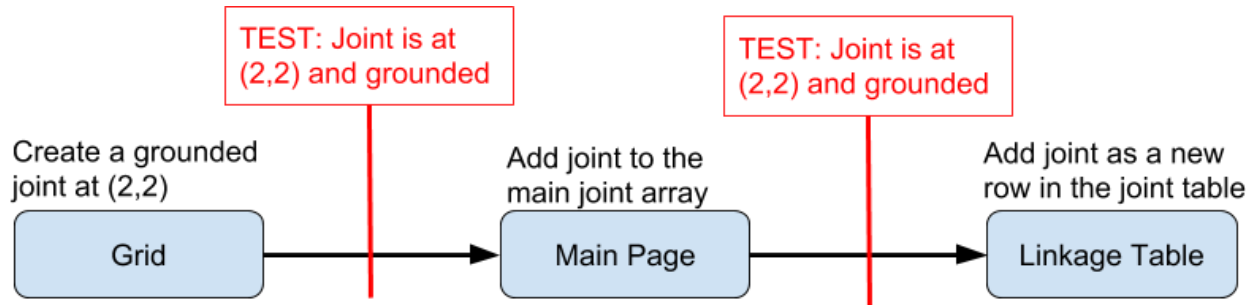


Figure 14.1 A diagram showcasing how test cases can be used to ensure the correct data is being passed between components.

Since Jasmine tests are run through the Angular command line, we found it difficult to test the interaction between the GUI and the application. For this, we used Augury, an unofficial Angular testing extension for the Google Chrome browser. Augury lists variables of Angular components in real time while the application is running (Angular Augury, 2018). This allows us to interact with the user interface and check to ensure the correct values were being set. For example, we can create a joint via the grid and reference Augury to ensure the joint was created and synchronized between components (Figure 14.2).

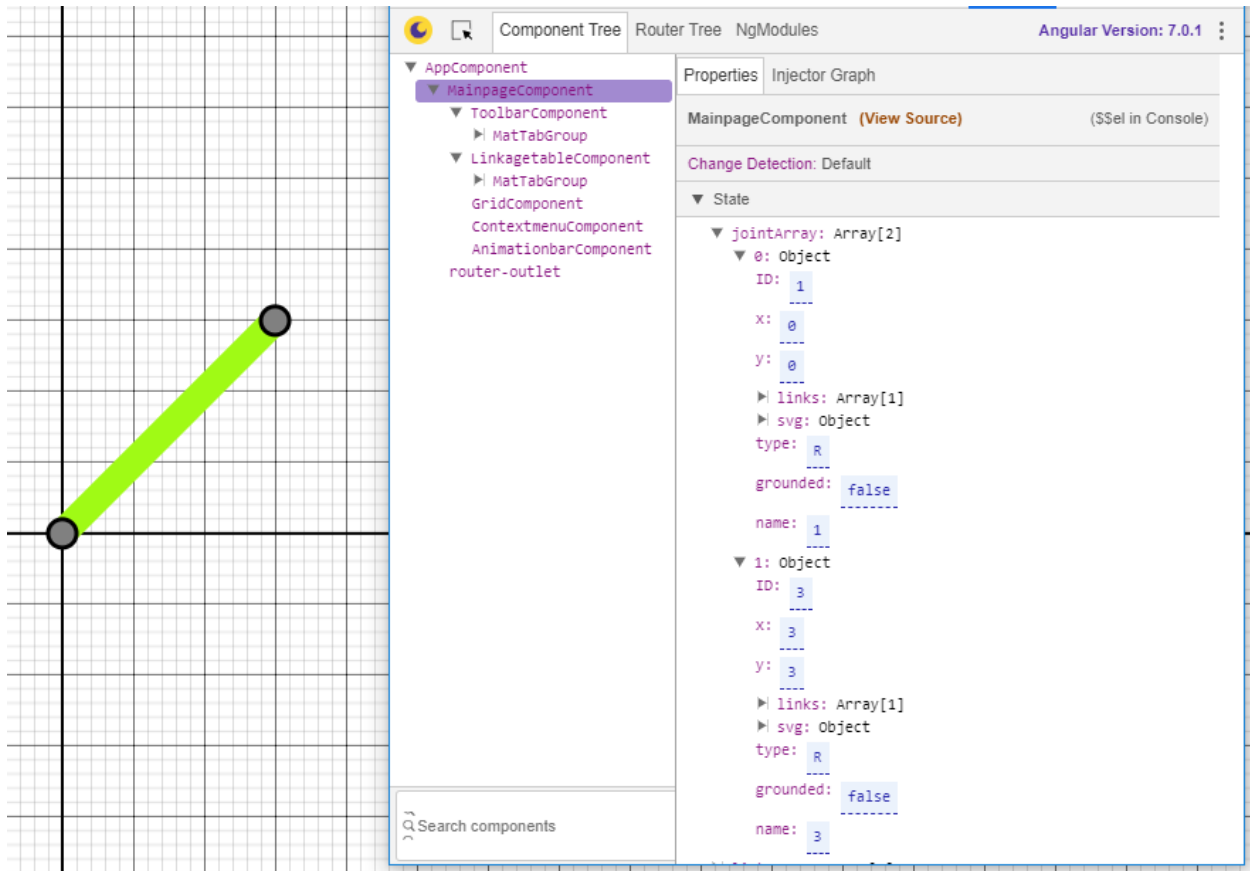


Figure 14.2 The Augury interface displaying detailed joint information about PMKS+ while it is running.



## Chapter 15

# PMKS+ User Evaluations

This chapter will focus on the user interface evaluations conducted during development of PMKS+. These evaluations include testing performed with individual subjects after iteration five of development and the large-group sessions performed after iteration six of development.

### 15.1 Individual Evaluations

In order to gather feedback about the interface design of PMKS+, our team conducted evaluations with individual participants. The focus of these evaluations was to determine whether the results of the users' interactions with the interface matched their expectations for what would happen.

#### 15.1.1 Procedure

The evaluations were conducted with a single proctor and subject. The proctor issued the evaluation in two separate parts. The first had the subject look at sets of two diagrams each showing different versions of the interface with modifications to color and layout. The subjects would then indicate which version they preferred. The second had the proctor instruct the subject to complete tasks directly within PMKS+. Before each task, the proctor would ask the subject what actions they were expecting to perform to complete the task. After each task, the proctor asked the subject for their opinion on the actions they performed and how those actions differed from their expectations. The proctor recorded each of the subject's answers by hand.

#### 5.1.2 Target Users

The target users for this evaluation were identified based on their college major and past experience with PMKS. This pool included Mechanical Engineering and Robotics Engineering

students as these are the main target audience of PMKS+. We also chose subjects with no prior experience with PMKS in order to eliminate pre-existing biases toward the system's design.

### **15.1.3 Evaluation Script**

During evaluations, the proctor read a script (included in Appendix B) detailing the questions and instructions for the subject. The proctor did not answer any questions relating to PMKS+ or make further actions not detailed by the script. If the subject determined that they could not continue, the proctor would complete the step, make a record of the assistance, and have the subject continue from there. The full script can be found in Appendix A. The accompanying diagrams can be found in Appendix B.

## **15.2 Final User Evaluations**

Our team set up the final evaluation of PMKS+ to mirror the evaluation we performed on the original PMKS application as discussed in Chapter 8. The evaluation consisted of two separate sessions. The first involved a large group of subjects completing tasks within the application. The second included a proctor that observed individual subject's interactions with PMKS+. A description of these methods can be found in the Session 1 and Session 2 descriptions of Chapter 8. Subjects were able to access PMKS+ on a temporary Heroku cloud server (Heroku). These servers allowed us to quickly patch any outstanding bugs found prior to the evaluation.

## **15.3 Differences with PMKS Evaluation**

Although our team used the same structure and methods as used for the PMKS eval for our evaluation of PMKS+, there were several changes made to the tasks and questions in the Google Form that subjects were required to complete. Some questions and tasks were excluded due to time constraints during development while others were included to gather feedback on aspects of the new system.

The first difference between the two evaluations is the exclusion of forces, and subsequently, the output of kinematic analysis. The original evaluations included tasks that required users to apply a force to and perform a static analysis of the linkage they built. Forces

were not implemented in PMKS+ prior to the evaluations. Similarly, many of the aspects of analysis (forces, torque, acceleration, etc.) were omitted as well. The questions relating to these tasks were removed from the existing evaluation and replaced with questions relating to the new aspects of PMKS+.

Our team included additional questions to determine which elements of the system subjects interacted with and the process by which they made those decisions. These questions included asking users about their expectations when creating links and what influenced these expectations. We also asked users for feedback on the new methods for creating linkages, as well as their comparison between PMKS and PMKS+.

## **15.4 Data Analysis**

The group evaluation of PMKS+ included a much smaller sample size than the previous evaluations conducted on PMKS. The results include only 18 subjects, a steep drop compared to the previous 39 participants. The number of possible subjects available in March was significantly less than in October because the number of Mechanical Engineering classes that use PMKS as part of their curriculum was higher in October. Subjects were required to have some experience with PMKS which limited the pool of possible subjects to students from these classes. This disparity resulted in less reliable data for the group evaluation, thereby hindering our ability to draw conclusions. Instead, our analysis aims to identify possible areas of improvement or decline between PMKS and PMKS+. The one-on-one evaluations maintained a pool of five subjects similarly to the one-on-one sessions conducted in October. The results from these were used to modify the user interface for iteration seven of design.

The experience level of subjects throughout our evaluations ranged from only slight experience with PMKS to around 6 hours of experience. This variance gave us a good distribution of past experience providing some evidence of subject performance based on time spent with PMKS. For both the PMKS and PMKS+ evaluations, subjects were given the same task to complete. Comparing the times that subjects took to complete this task to the amount of experience subjects have shows evidence of past users being comfortable and able to transfer skills from PMKS to PMKS+. Figure 15.1 details the amount of time subjects of both the PMKS and PMKS+ evaluations took to complete the given task.

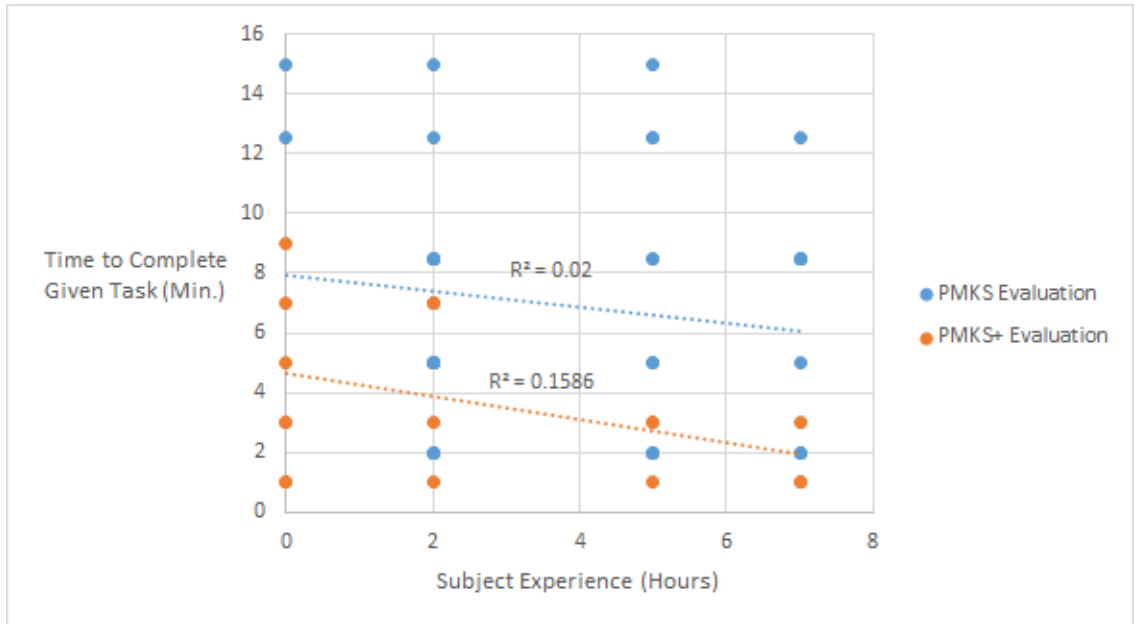
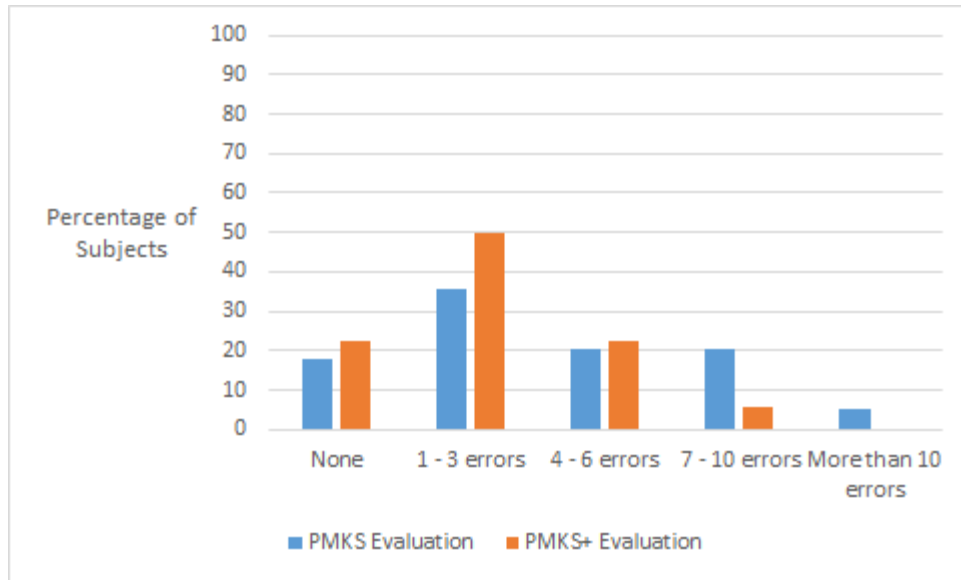


Figure 15.1: Subject experience vs. time to complete task for both sets of evaluations.

This data suggests an overall decrease in the time required to complete the given task. On average, the amount of time subjects using PMKS+ required was 3.66 minutes while those using PMKS required 6.87 minutes on average. The trend lines provided show a similar decrease between evaluations in time needed to complete the given task as subject experience increases. These trends suggest that skills acquired in PMKS may transfer to PMKS+. This could make previous users more comfortable and familiar with PMKS+.

In addition, more than 70% of subjects that used PMKS+ made three or fewer mistakes while performing tasks during the evaluation. In contrast, about 50% of subjects that used PMKS made a similar number of mistakes with almost 45% making 4 or more. These mistakes included creating too many links or joints, resetting the application, and similar actions. Figure 15.2 shows a comparison between the number of mistakes subjects made in the two evaluations. These data points suggest that PMKS+ could help users make less mistakes and therefore has improved usability over PMKS.



*Figure 15.2: Subject error rates while performing tasks with PMKS and PMKS+.*

Our evaluations also gathered some information about subject preferences and opinions. First, subjects in both evaluations were asked to rate the difficulty of creating links in their given applications. The average rating for PMKS+ is 4.06, which is slightly higher than the 3.64 given to PMKS. Although slight, this increase could suggest an improvement in creation methods catering to the expectations and experience of users. The data relating to this question is found in figure 15.3.

Furthermore, figures 15.4 and 15.5 show data comparing PMKS+ to PMKS and other simulation software including SolidWorks and Working Model. The majority of subjects rated using PMKS+ as a better experience than PMKS. In addition, these subjects also rated their use of PMKS+ as either as good or better than their use of other simulation software. These results show that PMKS+ may meet or exceed some standards set by its predecessor and other leading software.

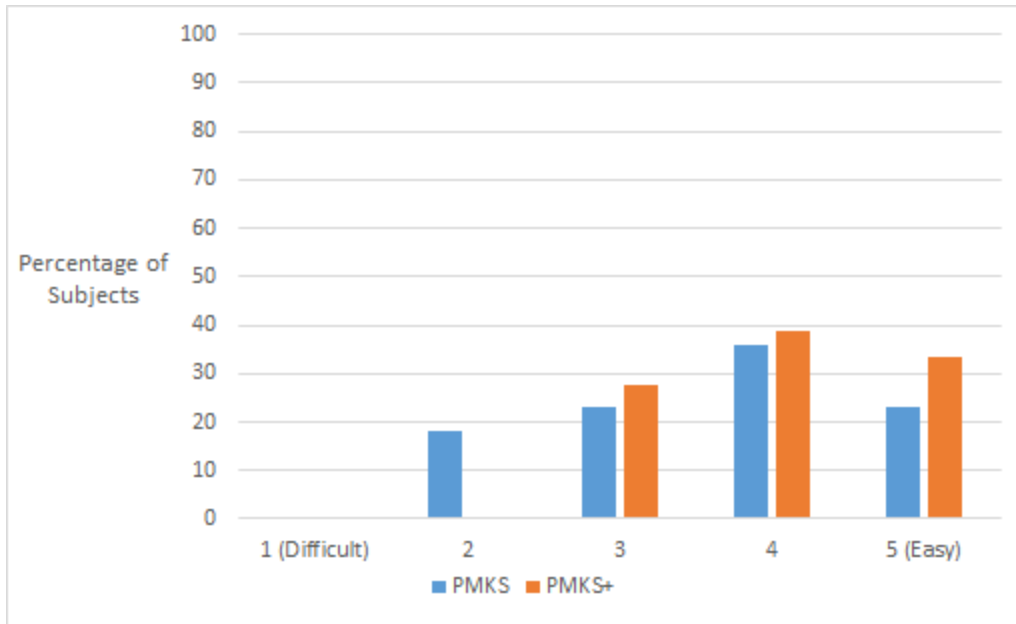


Figure 15.3: Subject rating of the difficulty of creating links between PMKS and PMKS+.

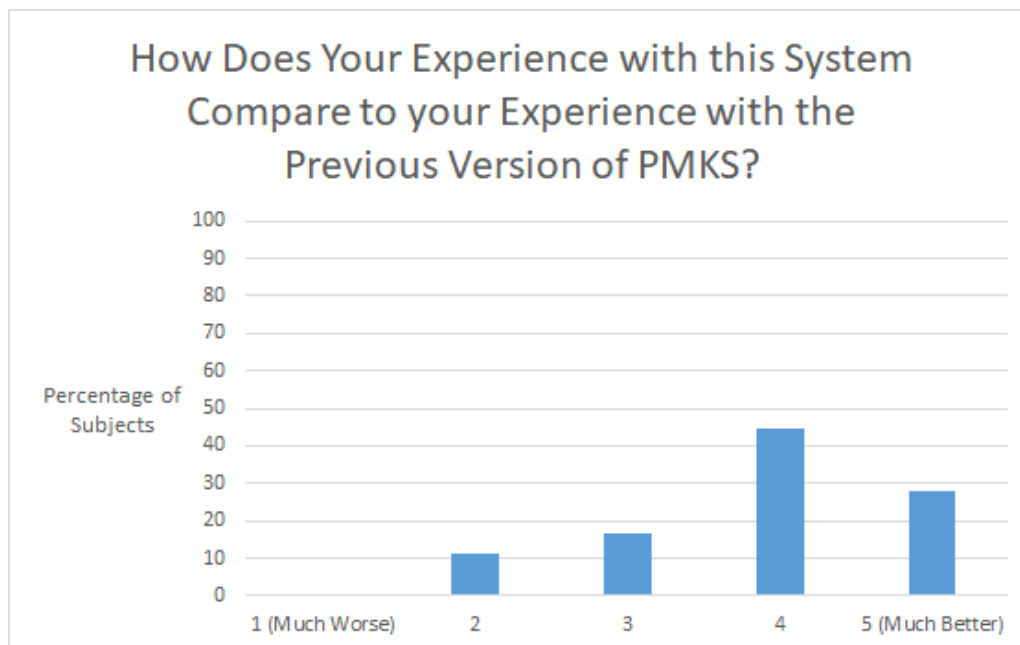
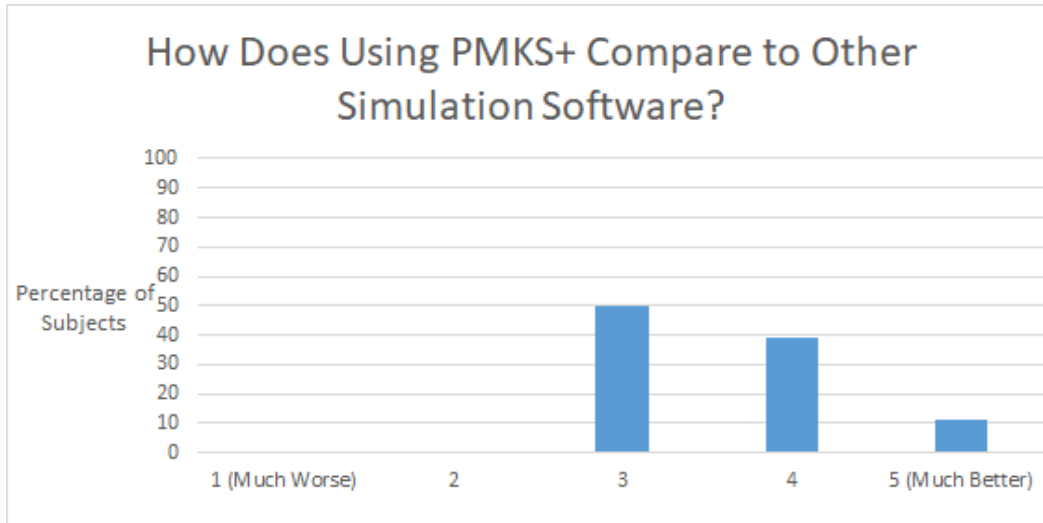


Figure 15.4: Subject preference between PMKS and PMKS+.



*Figure 15.5: Subject comparison between PMKS+ and other simulation software.*

In addition to the data collected above, our team also found several recurring opinions provided through written feedback from subjects. Many reported that although creating joints was easy, subjects wanted to position joints using exact values. The most prominent suggestions to achieve this were to simply forgo using the mouse for creation methods and use only the table as well as allowing joints to “snap” to the grid so that they automatically align with common increments. Subjects took issue with selecting exact coordinates only when editing and not when creating joints.

Some subjects also made comparisons between creating objects in other simulation software such as Working Model and CAD to PMKS+. The most common suggestion stems from creating objects in these software. Within these programs, creating objects or shapes takes place in a creating state where many user actions change to focus only on creating while other actions are limited until the state ends. There were also several subjects that suggested the joints and links shown on the grid could be better connected to the table displaying their information by showing a number or symbol on the grid to identify them. Also, subjects wanted better labels and more explanations for buttons. For example, subjects found that the button “Set as Ground” which could be used to set a joint as grounded or ungrounded did not convey both options clearly.

## Chapter 16

# Conclusions

The objective of this project was to create a modern successor to the PMKS application. To accomplish this, we defined four major goals.

1. Develop PMKS+ to be a similar application to PMKS, so that previous users feel comfortable with it.
2. Work on the compatibility issues that PMKS had to make PMKS+ work on most major browsers.
3. Enhance the user experience by redesigning and improving both the features and the user interface of PMKS based on user feedback.
4. Deliver a new application that is appropriately documented and maintainable.

PMKS+ successfully met all of these goals. First, it carried over important features from PMKS such as the linkage table and grid. This allowed users of the previous system to feel comfortable using the new system. Our user evaluations suggest that users with more experience in PMKS were more proficient with PMKS+ and completed tasks faster than those with less experience.

Next, we developed PMKS+ in Typescript and HTML using the Angular framework. This framework is supported by most major browsers including Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari.

In addition, we enhanced the user experience by adding mouse interactions and a context menu to the grid as well as a toolbar. The context sensitive menu and the mouse interactions allowed users to directly modify the linkage from the grid visualization itself. The toolbar provided a familiar way for users to interact with program functions. User evaluations show that subjects who used PMKS+ were, on average, faster than those who used PMKS. The majority of these subjects also preferred PMKS+ to its predecessor. We also implemented the Model-View-Controller design pattern to ensure that the system architecture is straightforward and simple. We set up a unit test environment using Angular to ensure that future changes to the codebase do not break existing functionality.



Finally, we structured the system architecture to be easy to understand by designing it around the Model-View-Controller framework. This framework is fairly common in web applications and also taught to WPI students in most Software Engineering courses. Additionally, we provided code documentation through inline comments and a developer manual to make sure that PMKS+ is maintainable.

The usability requirements detailed in Section 4.2.1 describe the goals we hoped to accomplish through our evaluations. First, we sought to allow users to easily create linkages with few actions required. The number of minimum steps required to create a linkage decreased from 30 steps in PMKS to 13 in PMKS+. Additionally, based on the previously discussed evaluation results, we found that users could effectively apply prior experience from PMKS into PMKS+ in order to more effectively use the application. We also ensured that users had the ability to reset the system, learn to create linkages, and efficiently control the animations.

## Chapter 17

# Discussion

This section describes the pedagogical value of this project and how our experiences studying at WPI contributed to its success.

Our first challenge as a team was to gather the appropriate knowledge to complete this project by researching migration options and web-design frameworks. We also sought advice from experts to make sure we selected the right tools to facilitate the development of PMKS+. Through this research and preparation, we gained valuable insight into the implications and process behind choosing the proper framework and environment for developing web applications. Our decision to use Angular also provided us valuable experience with a common framework in professional environments and large-scale applications. The experience we gained through “Webware: Computational Technology for Network Information Systems” course greatly influenced our initial decisions.

The majority of projects our team members have worked on in the past have had very short timespans. Work done for computer science classes is limited to one or two terms of eight weeks each. Prior to this project, our experience with applications in a development cycle spanning multiple years was very low. We were able to gain experience in developing not only for short-term goals but also for an application that can be transferred to a new team in the future. Our having taken the “Software Engineering” class at WPI provided experience with group projects in a major development environment that aided our cohesion and synergy as a team.

We decided to have group meetings and meetings with our advisors to keep track of the project’s progress. Our meetings with our advisors were held weekly where we discussed and presented work that was done the previous week and talked about the schedule for the next week to keep our advisors informed. We also gathered expert feedback on interface and linkage design during these meetings. Additionally, in order to keep track of our work, all of the team members wrote individual reports for every step of the project that was being done. For group meetings,

we followed a similar format to the advisor meetings, where we would inform each other of the progress done, problems faced and future work to be done. We also allocated the work in these meetings to make sure every team member was contributing to the project.

During this project, we followed a model-view-controller architectural pattern in designing the main components of the application. Our time in the “Object-Oriented Analysis and Design” class helped us to separate the functions of each component properly and restrict access to classes to their associated components. After using this design pattern, we found that maintaining proper encapsulation helped us protect our objects from unintentional changes that could cause errors in the system.

The evaluations our team conducted throughout the project were invaluable in gaining feedback for improving PMKS+. Creating these evaluations gave us a clearer understanding of the importance of gathering user feedback and developing for the user. At the beginning of development, parts of our initial designs included features that worked well from a developer’s perspective. After consultations with experts and conducting evaluations with prospective users, we found that our perspective of what worked best in PMKS+ did not match the users’ expectations. The design process requires that developers consider the user from the very beginning. During the user evaluations, we were able to incorporate the evaluation techniques we learned through the “Human-Computer Interactions” course.

Through this project, we learned more about web development and design frameworks, how to properly redesign a legacy application and how to drive development through user evaluations, contextual inquiries and heuristic evaluations from experts. Although it does not have all the features that PMKS has, PMKS+ has the proper foundation and documentation to ensure future development can generate a worthy successor of PMKS that can further help with the studies of linkages.

## Chapter 18

# Future Work

Due to time and knowledge constraints, our team was unable to develop certain features in PMKS+. These features were not necessary to meet the original goals of our project and were therefore left for future development. Below we discuss the features that we think will further improve the functionality and usability of PMKS+.

### **Mechanical Analysis**

PMKS had the ability to perform static, dynamic, and stress analysis on a linkage. This analysis includes calculations for calculating Centers of Mass, Moments of Inertia, and Torque. While our current application does not provide the calculations for such analysis, it does show a preview window for the values before downloading. Further steps for the future include performing the calculations for each type of analysis, populating the preview windows with important values, and then allowing the download of the full analysis report to a spreadsheet.

### **Saving and Sharing Linkages**

PMKS provided the option of saving a linkage either through a spreadsheet or a web URL. PMKS + would require us to develop a system to externally store linkage data that can later be reinterpreted by the application. PKMS did this by directly storing the linkage data in the URL through url variables.

### **Non-Dyadic Solver**

While our current simulator class can solve movement for most simple linkages, it will occasionally run into situations where the linkage has a correct degree of freedom but can't be solved. This situation arises when two unknown joints are unable to be solved via circle intersection. This problem is illustrated below in figure 18.1.

<p>Identify all ground and input positions and label their positions as known. Note that Joint G is the input. A previous example (Shown in Figure 12.6) used Joint A as the input.</p>	<p>Joint B is connected to the input so it can be solved by revolving it around the input.</p>	<p>Joint C can be solved by circle intersection only if it is connected to two known joints. There is only 1 known joint adjacent to it so it is unsolvable using circle intersection.</p>

*Figure 18.1: An example of a linkage that cannot be solved using circle intersection.*

To solve this linkage, an alternative to the circle intersection method must be applied. In PMKS this method was called the Non-Dyadic solver. Since this method is much more complicated than traditional position solving, we decided to leave this method out. In the future, Professor Radhakrishnan (who wrote the original position solver) plans to implement this method.

### Force Simulation

While forces can be applied in PMKS+, their effect on the simulation is ignored. We chose to omit force simulation because it was not required for us to simulate basic movement. The current implementation provides ample room to integrate forces into simulator calculations.

### Custom Link Geometry

One of the more complex features we left out was the use of custom link geometry. In reality, a link's geometry is not restricted to the joints that define it. Rather it can be any shape and joints can exist anywhere. We chose to restrict link geometry to joints similar to the old

system as we did not want to run into any major design problems implementing a new feature such as this. We also felt there was no reason to introduce custom link geometry unless there was proper collision simulation behind it. Collision simulation would require us to add more complex physics simulation to the simulator class, something we did not have the time for. Without collision simulation, custom linkage geometry would only exist to add extra visual fidelity to the linkage in the grid.

Before deciding to postpone development, we designed a system architecture that could incorporate custom link geometry. Links would be defined not by joints, but by guiding points known as nodes. A full description of the classes can be seen below. Figure 18.2 shows the class diagram containing each class.

- A **Node** represents a geometry guiding point in a link. These are used to help define the shape of a single linkage without acting as a point of connection between other links. While the simulator does not currently take into account geometry, Nodes provide a platform for such functionality in the future.
- A **Vertex** is the the abstract parent class of Nodes and Joints.
- **L, T, and C-Link** classes are special instances of Links that use nodes to define their geometry. Since they are defined by classes, we are able to define new functions and attributes on top of the existing ones in the Link class. Such functionality includes the ability to easily adjust the curve of a C-Link and the fillet inside T and L-Links.

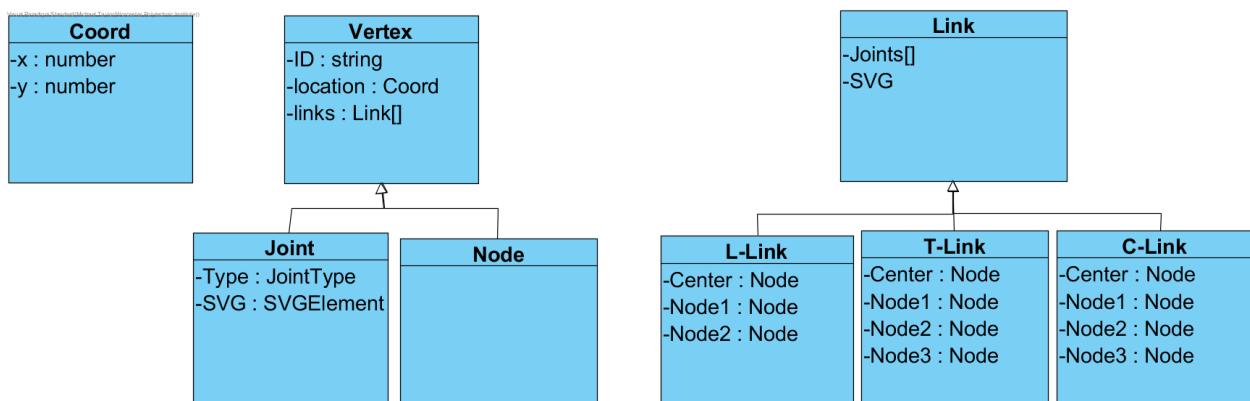


Figure 18.2: A class diagram representing the class hierarchy.

## Interactive tutorial

An interactive tutorial would provide a guided approach to help users learn PMKS+. An interactive tutorial would run through the basic steps of PKMS+ and blur out any parts of the interface not currently in use to help focus the user on certain tasks. Additionally, an instruction list could be displayed to help inform the user of the exact steps that need to be taken to complete each task. A mock-up of what such a tutorial could look like can be seen below in Figure 18.3.

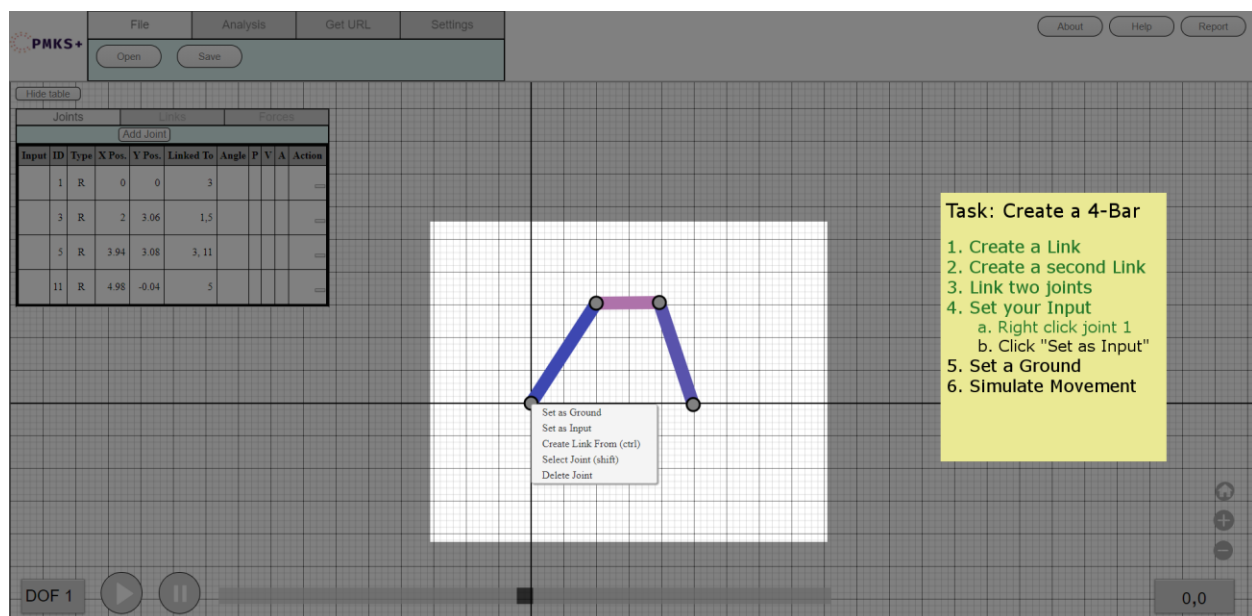


Figure 18.3.: A mock-up of what an interactive tutorial of PMKS+ would look like.

An interactive tutorial would need to be updated when any major changes are done to the system. Due to the rapidly changing elements of our interface during development, we decided that it would be better to leave out an interactive tutorial until more features are finalized for PMKS+.

## Implementation

PMKS+ was designed to be iterated on by Professor Radhakrishnan and WPI students in the future. Additions to the simulator class could be implemented by ME students. These include mechanical analysis, linkage collision, non-dyadic loop solving, and force simulation.

Meanwhile, additions to the UI and architecture could be implemented by Computer Science students. These include saving linkages, user settings, linkage geometry, and an interactive tutorial.



## Chapter 19

# References

- Albert, W., & Tullis, T. (2013). *Measuring the user experience: Collecting, analyzing, and presenting usability metrics* (2nd ed.) Newnes.
- Angular. (2018). Retrieved from <https://angular.io/>
- Angular Augury. (2018). Retrieved from <https://augury.rangle.io/>
- Avesta Group. (2015). Desktop applications vs. web apps. Retrieved from <https://www.avestagroup.net/DetailsEN.aspx?PostID=1006&CataType=5&CataID=1006>
- Best, M. (2017). Knockout release version 3.4.2. Retrieved from <https://github.com/knockout/knockout/releases/tag/v3.4.2>
- Bychkov, D. (2013). Desktop vs. web applications: A deeper look and comparison. Retrieved from <https://www.seguetech.com/desktop-vs-web-applications/>
- Castilho, B. (2015). Migrating from XAML to HTML5 with wijmo. Retrieved from <https://www.grapecity.com/en/blogs/migrating-from-xaml-to-html5-with-wijmo/>
- Constantine, L. L., & Lockwood, L. A. D. (1999). *Software for use: A practical guide to the models and methods of usage-centred design*. New York: ACM Press.
- Darwin, P. (2017). Angular architecture overview. Retrieved from <https://angular.io/guide/architecture>
- Eberhardt, C. (2012). KnockoutJS vs. silverlight. Retrieved from <https://www.codeproject.com/Articles/365120/KnockoutJS-vs-Silverlight>
- Felke-Morris, T. A. (2016). *Basics of web design*. (Third ed.) Boston [u.a.]: Pearson.

- Fracker, M. L. (2010). Scaling usability in terms of requirements: A method for evaluating user interfaces. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 54(6), 576-580. doi:10.1177/154193121005400607
- Furda, A., Fidge, C., Zimmermann, O., Kelly, W., & Barros, A. (2018). Migrating enterprise legacy source code to microservices: On multitenancy, statefulness, and data consistency. *Software, IEEE*, 35(3), 63-72. doi:10.1109/MS.2017.440134612
- Gaherwar, M. (2018). Blazor: Running C# in the browser using web assembly. Retrieved from <https://dzone.com/articles/blazor-running-c-on-browser-using-web-assembly>
- Google. (2019). Angular Docs. Retrieved February 4, 2019, from <https://angular.io/>
- Heroku. Cloud Application Platform, [www.heroku.com/](http://www.heroku.com/)
- James, J. (2011). How to replace flash and silverlight with HTML5. Retrieved from <https://www.techrepublic.com/blog/web-designer/how-to-replace-flash-and-silverlight-with-html5/>
- Knockout JS: Helping you build dynamic JavaScript UIs with MVVM and ASP.NET (2011, Mar 23,).[Video/DVD] Las Vegas, Nevada: MIX11.
- Krug, S. (2013). *Don't make me think: A common sense approach to web usability*. Pearson Education.
- Mozilla Contributors (2019) Intensive JavaScript *MDN Web Docs*. Retrieved from [developer.mozilla.org/en-US/docs/Tools/Performance/Scenarios/Intensive\\_JavaScript](https://developer.mozilla.org/en-US/docs/Tools/Performance/Scenarios/Intensive_JavaScript).
- Microsoft (Producer), & Sardo, G. (Director). (2011). HTML5 for Silverlight Developers. [Video/DVD] Microsoft. Retrieved from <https://channel9.msdn.com/events/MIX/MIX11/HTM14>
- Pop, P. (2002). Comparing Web Applications with Desktop Applications: An Empirical Study. [http://orbit.dtu.dk/en/publications/comparing-web-applications-with-desktop-applications-an-empirical-study\(a3dccc2-12ed-4cb4-97cc-c89a73eca245\).html](http://orbit.dtu.dk/en/publications/comparing-web-applications-with-desktop-applications-an-empirical-study(a3dccc2-12ed-4cb4-97cc-c89a73eca245).html)

- Ramel, D. (2018). Blazor, for .NET Web Apps using WebAssembly. Retrieved from <https://visualstudiomagazine.com/articles/2018/03/23/blazor-alpha.aspx>
- Roth, D. (2018). Blazor FAQ. Retrieved from <https://github.com/aspnet/Blazor/wiki/FAQ>
- Rubin, J., & Chisnell, D. (2011). *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. John Wiley & Sons.
- Radhakrishnan, P., and Campbell, M. (2012), *An Automated Kinematic Analysis Tool for Computationally Synthesizing Planar Mechanisms* ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE); August 12-15., Chicago, IL
- Safari Technology Preview Release Notes (2018).  
<https://developer.apple.com/safari/technology-preview/release-notes/>
- Satran, M., Whitney, T., Jacobs, M., Weston, S. & Das, D. (2018). What's a Universal Windows Platform (UWP) App? Retrieved from <https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide>
- Shneiderman, B., & Plaisant, C. (2016). *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (Fifth ed.). Reading, MA: Addison-Wesley.
- Shore, J., & Warden, S. (2008). *The art of agile development* (1. ed. ed.). 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly. Retrieved from <https://www.jamesshore.com/Agile-Book/>
- Silverlight support roadmap. (2014). Retrieved from <https://blogs.msdn.microsoft.com/webapps/2014/01/16/silverlight-support-roadmap/>
- Software Development Fundamentals. (n.d.). *Integration Testing*. Retrieved from <http://softwaretestingfundamentals.com/integration-testing/>

- Stieglitz, N. (2017). Silverlight Migration Strategies. Atlanta, GA: Wintellect LLC. Retrieved from <https://www.wintellect.com/wp-content/uploads/2017/05/SilverlightMigration-1.pdf>
- Stone, D. L., Jarrett, C., Woodroffe, M., & Minocha, S. (2005). *User Interface Design and Evaluation*. Amsterdam: Elsevier.
- Vaughn, B. (2013). React: Main concepts. Retrieved from <https://reactjs.org/docs/hello-world.html>
- Warren, G., Hogenson, G., Cai, S., Robertson, C., Casey, L., Jones, M., Al-Hashmi, B. (2018). Get started with WPF. Retrieved from <https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2017>
- White, S., Satran, M. & Koren, A. (2017). Move from windows phone silverlight to UWP. Retrieved from <https://docs.microsoft.com/en-us/windows/uwp/porting/wpsl-to-uwp-root>
- Whitney, T., Satran, M., Jacobs, M., Das, D. & devfables. (2018). What's a universal windows platform (UWP) app? Retrieved from <https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide>
- You, E. (2016). What is Vue.js. Retrieved from <https://vuejs.org/v2/guide/>

# Appendix A: PMKS User Evaluation Response Form

## PMKS

Please respond to the following questions. Once you have completed the section, press the "Next" button to begin the assigned tasks.

1. Please list your current major(s)

---

2. Please list your current minor(s)

---

3. What year are you in college? (If none, choose nearest approximation)

*Mark only one oval.*

- Freshman
- Sophomore
- Junior
- Senior
- Graduate

4. Do you have any past experience with the Planar Mechanism Kinematics Simulator (PMKS)?

*Mark only one oval.*

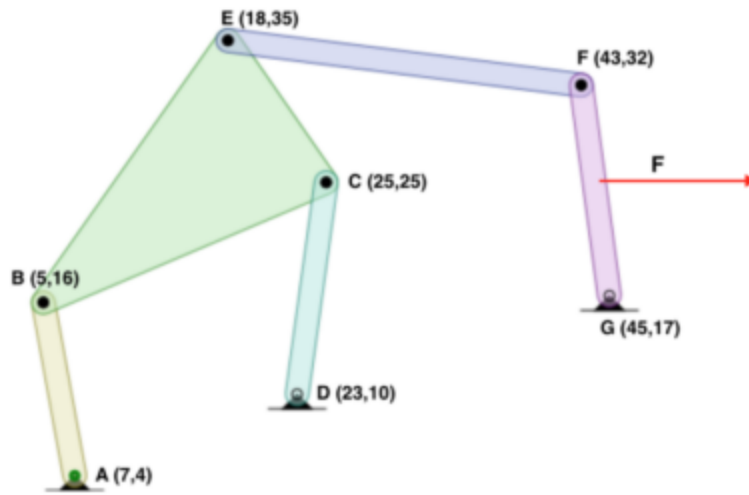
- None
- Less than 1 hour
- 1 - 3 hours
- 4 - 6 hours
- More than 6 hours

5. What modeling or simulation software have you used in the past?

*Check all that apply.*

- SolidWorks
- Autodesk Inventor
- Working Model
- SAM
- Blender
- Other: \_\_\_\_\_

6. Do you have any experience with joints, linkages, or other mechanics similar to that shown in the image below?



Mark only one oval.

- Yes
- No
- Maybe

## Tasks

In this section you will be asked to complete two tasks using the application found at the following link: [bit.ly/WPI-PMKS-MQP](http://bit.ly/WPI-PMKS-MQP)

Click on the link to open the application in a new tab.

## Task 1

---

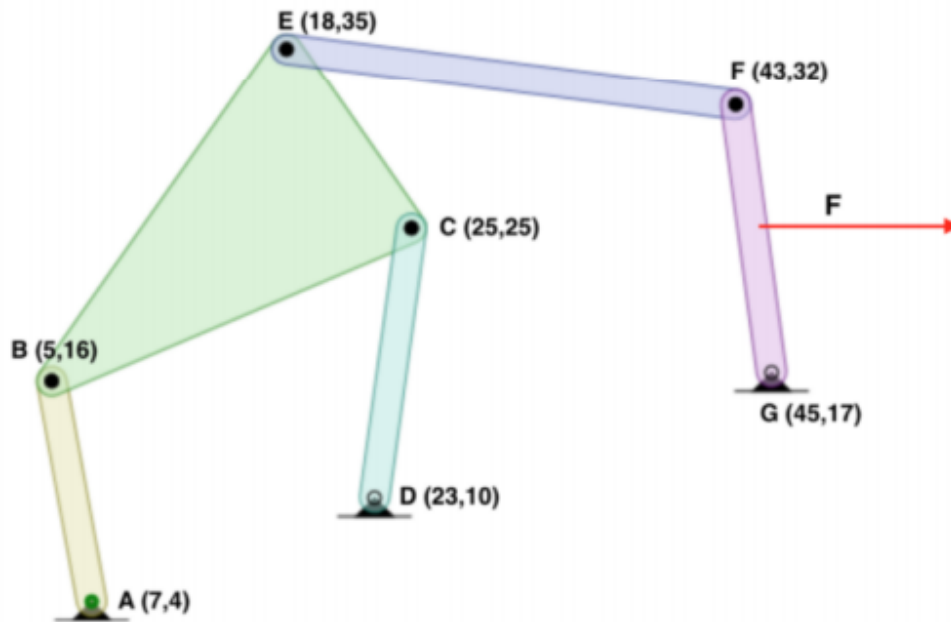
Complete parts 1, 2, and 3 below. After completing each part, answer the given questions before moving onto the next part.

### Part 1

---

Recreate the linkage found below in "Image 1". Do not apply the force F.

### Image 1



7. How much time did it take you to complete Task 1 Part 1?

Mark only one oval.

- 1 - 3 minutes
- 4 - 6 minutes
- 7 - 10 minutes
- 10 - 15 minutes
- More than 15 minutes

8. What is the DOF of the system?

\_\_\_\_\_

9. Approximately how many errors did you make while creating the linkage? (Examples: created an extra link, had to delete a link)

Mark only one oval.

- None
- 1 - 3 errors
- 4 - 6 errors
- 7 - 10 errors
- More than 10 errors

## Part 2

---

Apply a fixed force to the position shown as force F in "Image 1" with an X component of 5 and a Y component of 0. If your DOF is not one, first follow this url:

<http://users.wpi.edu/~pradhakrishnan/pmks/PMKS.html?set=f1&mech=ground,a.R,7.000,4.000,tfff|a,b,R,5.000,16.000,tfff|b,c,R,25.000,25.000,tfff|c,ground,R,23.000,10.000,tfff|b,e,R,18.000,35.000,tfff|e,f,R,43.000,32.000,tfff|f,ground,R,45.000,17.000,tfff>

10. Are you able to see the force by selecting "display" next to the force in the table?

Mark only one oval.

- Yes  
 No

11. How much time did it take you to complete Task 1 Part 2?

Mark only one oval.

- 1 - 3 minutes  
 4 - 6 minutes  
 7 - 10 minutes  
 10 - 15 minutes  
 More than 15 minutes

12. Approximately how many errors did you make while applying a force? (Examples: received message "not on a link")

Mark only one oval.

- None  
 1 - 3 errors  
 4 - 6 errors  
 7 - 10 errors  
 More than 10 errors

## Part 3

---

Attempt to export the system data as a static analysis. If the analysis successfully exports, open the file and answer the following question. If the file could not be exported, leave the next question blank.

13. What is the angle of the input at position 1? If you are unsure, please respond with "unsure"

\_\_\_\_\_

## Task 2

---

Complete parts 1 and 2 below. Answer the questions that follow before continuing onto the next section.

## Part 1

---

Enter the following link into the browser URL:

<http://users.wpi.edu/~pradhakrishnan/pmks/PMKS.html?>



[set=f1&mech=ground,input,R,0.000,0.000,0.000.tfff|input,coupler,R,0.000,8.000,0.000.tfff|coupler,follower,R,20.000,8.000,0.000.tfff|follower,ground,R,20.000,0.000,0.000.tfff|input,coupler1,R,0.000,8.000,0.000.tfff|coupler1,follower2,R,-20.000,8.000,0.000.tfff|follower2,ground,R,-20.000,0.000,0.000.tfff](#)

## Part 2

Adjust the joints of the system to match the image shown below.

### Image 2

ground,input	R	0.000	0.000	0.000			
input,coupler	R	8.000	0.000	0.000	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
coupler,follower	R	8.000	20.000	0.000	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
follower,ground	R	0.000	20.000	0.000			
input,coupler1	R	8.000	0.000	0.000	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
coupler1,follower2	R	8.000	-20.000	0.000	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
follower2,ground	R	0.000	-20.000	0.000			

ow   **DOF = 1**

14. How much time did it take you to complete Task 2 Part 2?

Mark only one oval.

- 1 - 3 minutes
- 4 - 6 minutes
- 7 - 10 minutes
- 10 - 15 minutes
- More than 15 minutes

15. What is the DOF of the system?

\_\_\_\_\_

16. Approximately how many errors did you make while editing the linkage? (Examples: started over, had to delete a link, etc.)

Mark only one oval.

- None
- 1 - 3 errors
- 4 - 6 errors
- 7 - 10 errors
- More than 10 errors

**17. What method did you use to edit the joints?**

*Mark only one oval.*

- Drag and Drop
- Input values into table
- Both
- Other: \_\_\_\_\_

**Post-Tasks Survey**

After completing both sets of tasks, please complete the questions below. You may return to the PMKS site to review its components.

**18. How difficult is it to create a link?**

*Mark only one oval.*

	1	2	3	4	5	
Very Difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very Easy

**19. How difficult is it to move existing joints?**

*Mark only one oval.*

	1	2	3	4	5	
Very Difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very Easy

**20. What changes should be made to make creating linkages and joints easier?**

---

---

---

---

---

**21. How difficult is it to navigate the menu in the top-left corner?**

*Mark only one oval.*

	1	2	3	4	5	
Very Difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very Easy

**22. What changes should be made to make the menu easier to navigate?**

---

---

---

---

23. How difficult is it to read the exported data?

Mark only one oval.

	1	2	3	4	5	
Very Difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very Easy

24. How does using PMKS compare to using similar simulation software?

Mark only one oval.

	1	2	3	4	5	
Much Worse	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Much Better

25. Do you have any recommendations for improving parts of the interface?

---

---

---

---

---

## Appendix B: PMKS+ User Evaluation Response Form

### PMKS+

Please respond to the following questions. Once you have completed the section, press the "Next" button to begin the assigned tasks.

\* Required

1. Please list your current major(s)

---

2. Please list your current minor(s)

---

3. What year are you in college? (If none, choose nearest approximation)

*Mark only one oval.*

- Freshman
- Sophomore
- Junior
- Senior
- Graduate

4. Do you have any past experience with the Planar Mechanism Kinematics Simulator (PMKS)?

*Mark only one oval.*

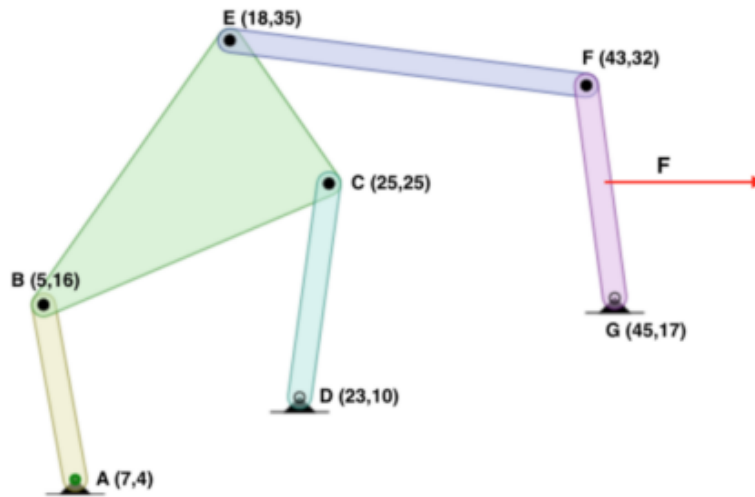
- None
- Less than 1 hour
- 1 - 3 hours
- 4 - 6 hours
- More than 6 hours

5. What modeling or simulation software have you used in the past? (choose all that apply)

*Check all that apply.*

- SolidWorks
- Autodesk Inventor
- Working Model
- SAM
- Blender
- Other: \_\_\_\_\_

6. Do you have any experience with joints, linkages, or other mechanics similar to that shown in the image below?



Mark only one oval.

- Yes
- No
- Maybe

## Tasks

In this section you will be asked to complete two tasks using the application found at the following link: <http://pmksplus.herokuapp.com>

Click on the link to open the application in a new tab.

### Task 1

---

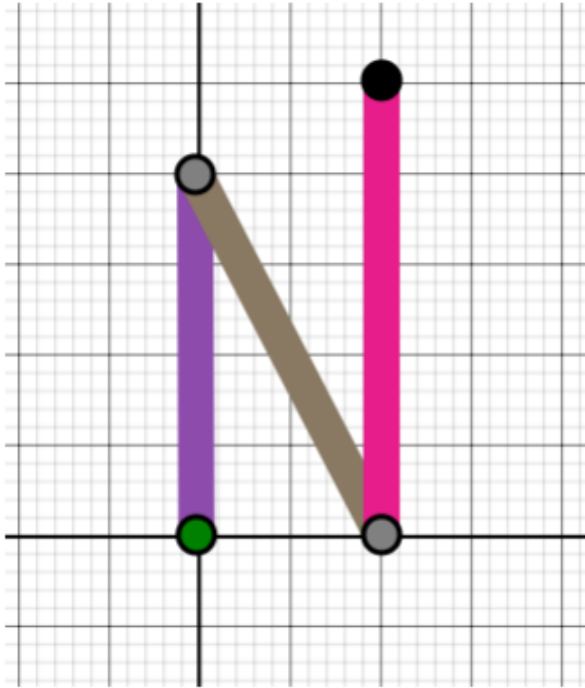
Complete parts 1, 2, and 3 below. After completing each part, answer the given questions before moving onto the next part.

#### Part 1

---

Recreate the linkage found below in "Image 1". If at any point you cannot proceed with the system, ask a proctor for assistance.

#### Image 1



7. How much time did it take you to complete Task 1 Part 1?

Mark only one oval.

- 1 - 2 minutes
- 3 - 4 minutes
- 5 - 6 minutes
- 7 - 8 minutes
- 9-10 minutes
- More than 10 minutes

8. What is the DOF of the system?

\_\_\_\_\_

9. Approximately how many errors did you make while creating the linkage? (Examples: created an extra link, had to delete a link)

Mark only one oval.

- None
- 1 - 3 errors
- 4 - 6 errors
- 7 - 10 errors
- More than 10 errors

**10. How did you initially expect to create joints/links?**

*Mark only one oval.*

- The table on the left side of the page
- Drawing on the coordinate grid
- Right-clicking on the coordinate grid to open a menu
- Drag-and-dropping elements onto the coordinate grid
- Other: \_\_\_\_\_

**11. Did you use the "Help" button located in the top right corner?**

*Mark only one oval.*

- Yes
- No

**12. If yes, was the information you found useful for creating the linkage?**

*Mark only one oval.*

- Yes
- No
- Somewhat

**13. Write a short description of what you expect to happen if you were to press the "Play" button now.**

---

---

---

---

---

**14. Now press the play button. Did the outcome match your expectations? (You may press the pause button to stop the animation.) \***

*Mark only one oval.*

- Yes
- No
- Somewhat

## **Task 2**

---

Complete parts 1 and 2 below. Answer the questions that follow before continuing onto the next section.

### **Part 1**

---

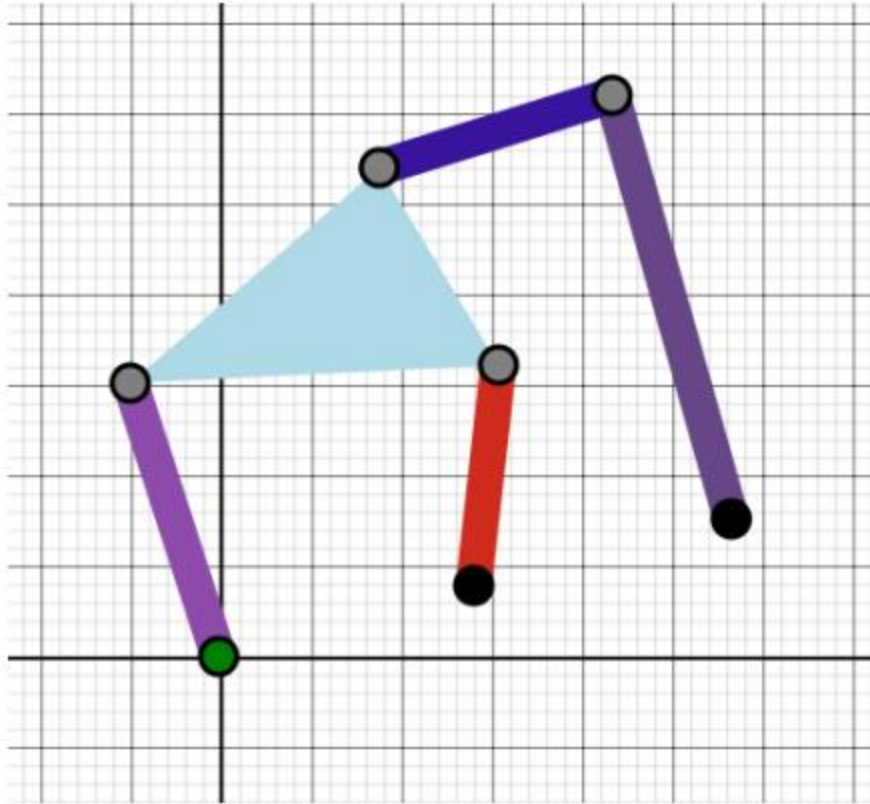
Ensure that the linkage you've made matches the linkage given in Task 1. If you are unsure, ask a proctor to make sure the linkage matches.

## Part 2

---

Without reloading the web page, adjust the joints of the system to match the image shown below. If you find that you are unable to proceed, ask a proctor for assistance.

### Image 2



15. How much time did it take you to complete Task 2 Part 2?

*Mark only one oval.*

- 1 - 2 minutes
- 3 - 4 minutes
- 5 - 6 minutes
- 7 - 8 minutes
- 9 - 10 minutes
- More than 10 minutes

16. What is the DOF of the system?

---



17. Approximately how many errors did you make while editing the linkage? (Examples: started over, had to delete a link, etc.)

Mark only one oval.

- None
- 1 - 3 errors
- 4 - 6 errors
- 7 - 10 errors
- More than 10 errors

18. How did you figure out how to create the triangle link?

Mark only one oval.

- The "Help" page
- Through experimenting with the system
- From a proctor
- Other: \_\_\_\_\_

## Post-Tasks Survey

After completing both sets of tasks, please complete the questions below. You may return to the PMKS site to review its components.

19. How difficult is it to create a link?

Mark only one oval.

	1	2	3	4	5	
Very Difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very Easy

20. How difficult is it to move existing joints?

Mark only one oval.

	1	2	3	4	5	
Very Difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very Easy

21. What changes should be made to make creating linkages and joints easier?

---

---

---

---

---

## Appendix C: Individual Evaluation Script

1. When responding to questions, be as specific as possible and include all parts of your thought process. When performing tasks in the application, state any thoughts that you have out loud. The proctor will not be able to answer any questions you have. If you believe you cannot accomplish a task, inform the proctor and they will complete the current step for you. The proctor will record your answers in writing. If you need a question repeated ask the proctor.
2. For your reference, Image A shows a linkage. This linkage is composed of three links, the colored bars, and connected by 4 joints, the circles. We will refer to the 2D plane that these objects are on as the “Grid”.
3. What is the significance of the black colored joint? In other words, what does the joint being black signify?
4. What is the significance of the green colored joint? In other words, what does the joint being green signify?
5. (If the subject could not tell the purpose of the joints, the proctor will explain with the following) The black joint signifies a grounded joint. The green joint signifies the input joint.
6. Look at images B and C. Which design do you prefer?
7. We will now work with the application. Navigate to the opened tab in Google Chrome.
8. Direct your attention to the table on the left side of the screen. Describe what the table is displaying and what parts of the grid relate to the table. You may interact with the table while answering.
9. What do you expect would happen if you were to click on the grid? Drag the grid? Right-click on the grid?
10. We will now ask you to create a single link in the application. First, describe the process you believe you will need to follow in order to create a link. For this task, a link is composed of two joints which act as endpoints and are connected with a bar.

11. Click on an empty space on the grid. Once you have done that, click on another empty space to create a link. Was this process different than what you expected? What did you like about the process? What did you dislike?
12. What do you expect would happen if you were to click on the link at this moment?
13. What do you expect would happen if you were to click on a joint at this moment?
14. What do you expect would happen if you were to drag a joint at this moment?
15. Now, hold down the shift key and click on a joint on the grid. Describe what happened when you clicked on the joint. What does this action signify?
16. Now, click on an empty space on the grid so that the joint is un-highlighted. Then, right click on a joint. Which method, shift-clicking or right-clicking, do you prefer to select a joint? Which method is clearer in what it does?
17. Next, you are going to create another link and attach one of its endpoints to an existing joint from the original link. Before using the application, how do you expect to attach a new link to the existing one? Now, try to attach a new link.
18. Add another link attached to the end of the newly created link so that it is similar in shape to the diagram shown in Image D. Also, try to match the colors of the colors of the joints. Don't forget to share your thought process out loud.
19. (If the system needs to be reset due to the user input too many joints/reaching an error state, the proctor should take control of the system, refresh the page, and create a 4-bar linkage, recording the subject's trial as a failure)
20. Direct your attention to the icons at the bottom of the screen.
21. Click on the box that says DoF at the bottom left of the screen. This is a temporary step to perform necessary calculations.
22. What do you expect to happen when you press the green button to the right of the DoF box?
23. What do you expect to happen when you press the red button to the right of that?

24. What do you expect the slider that runs along the length of the screen to do?
25. Now, press the green button. How does what happen differ from what you expected?
26. Now, press the red button. What do you expect to happen if you press the play button again?
27. Move the slider at the bottom of the screen. Does the movement of the linkage match what you expected?
28. What is your opinion on the design and layout of these elements, the buttons and slider?

# Appendix D: Individual Evaluation Paper Examples

Screenshot demonstrating PMKS+ with colored elements.

The screenshot shows the PMKS+ software interface. At the top, there is a menu bar with options: Open File, Save, Analysis, Get URL, Settings, About, Help, and Report. Below the menu bar is a file selection area with a 'Choose a file to open' button and a text input field for the file location.

The main workspace is a grid with a horizontal axis. A mechanism is displayed with four colored links: a light green link, a dark green link, a teal link, and a red link. The joints are represented by black dots. A 'DOF' (Degrees of Freedom) control panel is located at the bottom left, featuring a play button, a stop button, and a slider. The current DOF value is displayed as 500.69, 262.04.

Input ID	Joint Type	X Pos.	Y Pos.	Linked To	Angle	P	V	A
1	R	-297.95432684647305	-26.814796394991834	2				
2	R	-169.0594744121715	186.31660056213127	1, 3				
3	R	-58.01244813278009	-30.708295703428913	2, 4				
4	R	91.92946058091285	188.3414968276914	3				

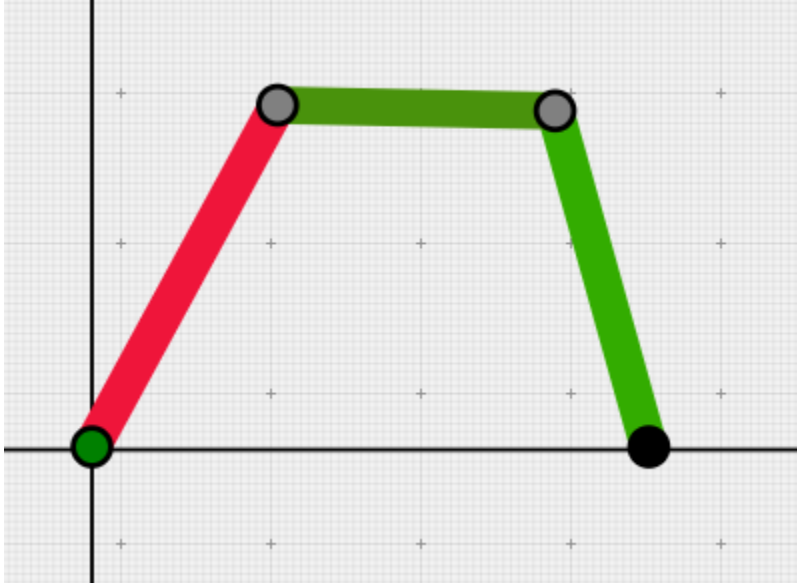
Screenshot demonstrating PMKS+ with monochrome elements.

The screenshot shows the PMKS+ software interface in a monochrome theme. The layout is identical to the first screenshot, but the mechanism's links are all gray. The 'DOF' control panel at the bottom left shows a value of -736.02, 300.74.

Input ID	Joint Type	X Pos.	Y Pos.	Linked To	Angle	P	V	A
1	R	-310.00553250345786	-47.76915324146487	2				
2	R	-216	221.3000030517578	1, 3				
3	R	-126.00553250345781	-44.75808823454924	2, 4				
4	R	-18	235.3000030517578	3, 5				
5	R	76.03457814661134	-50.76500386387151	4				

# Image A

Screenshot used to ask subjects whether they could identify grounded and input joints.



## Image B

Screenshot showing an example of the PMKS table.

Input	Links	Type of Joint	X Pos.	Y Pos.	Angle	P	V	A
<input checked="" type="radio"/>	ground, input	R	0.000	0.000				
<input type="radio"/>	input	R	25.000	0.000		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

## Image C

Screenshot showing an example of the PMKS+ table.

Input	ID	Joint Type	X Pos.	Y Pos.	Linked To	Angle	P	V	A	Action
	1	R	0	1.5333404541015625	2					DELETE
	2	R	99	48.53334045410156	1					DELETE

# Image D

A screenshot of a standard linkage built in PMKS+.

