

Which Tool Did I Use There?

Supporting
DIY Crafting Documentation
Using
Tracking
and
Data Visualization

Master's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Mithil Mrutyunjaya Kashipurad

Thesis advisor:
Prof. Dr. Jan Borches

Second examiner:
Prof. Dr. Bastian Leibe

Registration date: 18.04.2023
Submission date: 18.10.2023

Eidesstattliche Versicherung

Statutory Declaration in Lieu of an Oath

Kashipurad, Mithil

Name, Vorname/Last Name, First Name

407537

Matrikelnummer (freiwillige Angabe)

Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/
Masterarbeit* mit dem Titel

I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

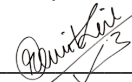
Which Tool Did I Use There? Supporting DIY Crafting Documentation Using Tracking
and Data Visualization

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Aachen, 29.08.2023

Ort, Datum/City, Date



Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:

Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

Para. 156 StGB (German Criminal Code): False Statutory Declarations

Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtet. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence

(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.


(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

I have read and understood the above official notification:

Aachen, 29.08.2023

Ort, Datum/City, Date



Unterschrift/Signature

Contents

Abstract	xv
Überblick	xvii
Acknowledgements	xix
Conventions	xxi
1 Introduction	1
1.1 Outline	3
2 History, Motivation, and Related Work	5
2.1 Historical Precedence	5
2.1.1 DoDoc: A Modular Approach to Record Traces of Activities	8
2.1.2 Protobooth: A Photo Booth for Proto- types	9
2.1.3 Spin: A Photography Turntable Sys- tem for Animated Documentation . . .	11
2.1.4 The Maker Ed Open Portfolio Project	12

2.2	Related Work on Tool Tracking	13
2.2.1	Instrumenting and Analyzing Fabrication Activities	14
2.2.2	ArUco-Based Tool tracking	16
3	Initial Attempts: Sensor-Based Tool Tracking and Tracking with Vicon Motion Capture	17
3.1	Sensor-Based Tool Tracking	18
3.1.1	IMU Sensor with a Wired Setup	18
3.1.2	IMU Sensor with a Wireless Setup	20
3.2	Vicon Motion Capture	22
4	Overview of the System Components	25
4.1	Camera Setup	26
4.2	Tool Tagging	31
5	Web Application Component: Design and Imple- mentation	36
5.1	Data Collection	37
5.2	Data Storage	40
5.3	Data Processing and Transformation	40
5.3.1	Splitting and Grouping	41
5.3.2	Estimating the Tool Usage	42
5.3.3	Suggesting Tool Usage Order	43

5.3.4	Creating Uninterrupted Continuous Time Series Data	46
5.4	Visualizing the Data	47
5.5	Application Run	49
5.5.1	Setup Phase	50
5.5.2	File Creation Phase	51
5.5.3	Preparation Phase	52
5.5.4	Detection Phase	54
5.5.5	Visualization Phase	57
6	Technical Specifications, Tool Testing and Results	60
6.1	Camera Specifications	62
6.2	Computer/Processing Hardware	62
6.3	Software Specifications	63
6.4	Technical Boundary Analysis	64
6.5	System Testing	67
6.5.1	Purpose	67
6.5.2	Procedure	68
6.5.3	Evaluation and Results	69
7	Summary, Limitations, and Future Work	72
7.1	Summary of the work	72
7.2	Related Fields	74

7.3	Limitations and Future work	75
A	Important Code Snippets of the Web Application	80
B	List of Packages and their Versions	93
	Bibliography	95

List of Figures

2.1	DoDoc DIY setup	9
2.2	Protobooth setup	10
2.3	Protobooth entry timeline	10
2.4	Spin system Setup	11
2.5	Simple DIY setups	12
2.6	Multisensorial data collection	14
2.7	Tool sensorization and processing	15
2.8	ArUco-based tracking	16
3.1	Arduino Nano with IMU (wired)	20
3.2	Arduino Nano 33 IoT	21
3.3	Pre-fabricated BLE module	22
3.4	Vicon system and motion detection	23
3.5	Tools with reflective markers	24
4.1	Entire prototype setup	26
4.2	Three components of the system	26

4.3	Low vs High Resolution	27
4.4	Small vs Large marker	27
4.5	Tracking area	28
4.6	iVCam Interface	31
4.7	Tool Tagging	32
4.8	ArUco marker types	32
4.9	Right and wrong tool placement	33
4.10	ArUco marker on curved surface	34
4.11	ArUco marker placement	35
5.1	Overview of the Web application steps	36
5.2	Data Collection	37
5.3	Marker detection	39
5.4	Rest time stamp analysis	45
5.5	Cleaned vs Uncleaned data	46
5.6	Combined plot of all tools	48
5.7	Separated plot of each tool	48
5.8	KDE plot	49
5.9	Overview of the Web application steps	50
5.10	Home Page	50
5.11	Setup up page	51
5.12	New file creation	51

5.13	Append and Delete file	52
5.14	Append and Delete file	52
5.15	Tools affixed with markers	53
5.16	Tools affixed with markers	54
5.17	Constraint Info	54
5.18	Detection window and Console	55
5.19	Constraint pop-up	55
5.20	Detection window	56
5.21	Console	56
5.22	Visualization Page	57
5.23	Interactive graph	58
5.24	Return to home	58
6.1	Phones having cameras	61
6.2	Sales of smartphone vs other cameras	62
6.3	Technical boundary analysis	65
6.4	Saturation in detection limits	67
6.5	A user performing a task using a tool	69
6.6	Manual vs Tool comparison	70
6.7	Manual vs Tool for all 8 trials	71
7.1	ARPen	76

List of Tables

6.1	Varying resolution with constant marker size	66
6.2	Varying marker size with a fixed resolution .	66
B.1	Packages used and their Versions	94

Abstract

DIY (Do-it-Yourself) tutorials are fun to create, but it follows the arduous task of documenting everything extensively and in a coherent way so that the readers are able to understand and replicate the steps themselves. Often this task is daunting, error-prone, and boring. Good documentation is critical to the success and vitality of DIY practices. Good tutorial authorship is one way to maintain and improve the quality of the documentation in DIY processes. While project documentation can help young makers showcase their learning, prototyping skills, and creativity, motivating documentation practices has remained a challenge. Automated collection of basic details like the tool list, materials used, usage times, and sequence of usage can already provide an overview of the entire process. Such first-hand information can assist the author's documentation process by giving him a head start.

In this thesis, we develop an affordable easy-to-reproduce system to support the DIY crafting documentation process by automating some steps as mentioned previously. The system starts off by asking the author to enter the tools that he wishes to use, this helps ensure that we have a list of all the tools to be tracked in real-time. Each tool can be differentiated by affixing a different marker to it, in our case, the ArUco markers. The system detects and tracks the markers using a camera. An iteration is defined as the run from having entered the tool details to visualizing their usage statistics. New tools can be added in any given iteration, simulating the real-life DIY crafting process scenario. The system's main objective is to give the user information about the sequence of tool usage. Other details related to tool usage statistics and several visualizations are also provided for additional assistance.

Technical specifications and best practices for the system usage are described. This is followed by the technical boundary analysis where the operation limits of the system are listed. Multiple iterations of testing done to investigate its functioning, correctness and accuracy are also described.

Überblick

DIY (Do-it-Yourself)-Anleitungen zu erstellen macht Spaß, aber es folgt die mühsame Aufgabe, alles ausführlich und verständlich zu dokumentieren, damit die Leser die Schritte verstehen und selbst nachmachen können. Oft ist diese Aufgabe entmutigend, fehleranfällig und langweilig. Eine gute Dokumentation ist entscheidend für den Erfolg und die Lebendigkeit von Heimwerkerpraktiken. Eine gute Autorenschaft für Anleitungen ist eine Möglichkeit, die Qualität der Dokumentation von DIY-Prozessen zu erhalten und zu verbessern. Während die Projektdokumentation jungen Machern dabei helfen kann, ihr Lernen, ihre Fertigkeiten beim Prototyping und ihre Kreativität zu präsentieren, ist die Motivation für die Dokumentationspraxis nach wie vor eine Herausforderung. Die automatisierte Erfassung grundlegender Details wie der Werkzeugliste, der verwendeten Materialien, der Nutzungszeiten und der Reihenfolge der Nutzung kann bereits einen Überblick über den gesamten Prozess geben. Solche Informationen aus erster Hand können den Dokumentationsprozess des Autors unterstützen, indem sie ihm einen Vorsprung verschaffen.

In dieser Arbeit entwickeln wir ein erschwingliches, leicht zu reproduzierendes System, das den Prozess der DIY-Basteldokumentation unterstützt, indem es einige der oben erwähnten Schritte automatisiert. Das System beginnt damit, dass der Autor aufgefordert wird, die Werkzeuge einzugeben, die er verwenden möchte. Dadurch wird sichergestellt, dass wir eine Liste aller Werkzeuge haben, die in Echtzeit verfolgt werden können. Jedes Werkzeug kann unterschieden werden, indem es mit einer anderen Markierung versehen wird, in unserem Fall mit den ArUco-Markern. Das System erkennt und verfolgt die Marker mit Hilfe einer Kamera. Eine Iteration ist definiert als der Durchlauf von der Eingabe der Werkzeugdetails bis zur Visualisierung ihrer Nutzungsstatistiken. In jeder Iteration können neue Werkzeuge hinzugefügt werden, wodurch das reale Szenario eines Heimwerkerprozesses simuliert wird. Das Hauptziel des Systems ist es, dem Benutzer Informationen über die Reihenfolge der Werkzeugverwendung zu geben. Weitere Details zur Statistik der Werkzeugnutzung und verschiedene Visualisierungen werden als zusätzliche Hilfe bereitgestellt.

Es werden technische Spezifikationen und bewährte Verfahren für die Nutzung des Systems beschrieben. Darauf folgt die Analyse der technischen Grenzen, in der die Betriebsgrenzen des Systems aufgeführt sind. Es werden auch mehrere Iterationen von Tests beschrieben, die durchgeführt wurden, um die Funktionsweise, Korrektheit und Genauigkeit des Systems zu untersuchen.

Acknowledgements

To start with, I would like to thank my thesis advisor Prof. Dr. Jan Borchers for supporting me and providing me with the opportunity to write my master's thesis at the Media Computing Group. In addition, I would like to thank Prof. Dr. Bastian Leibe for examining my thesis as a second examiner.

I would like to thank Marcel Lahaye for supervising me, for the constant guidance and support, and for giving me the freedom to work with my ideas. It was an exciting experience to work under your supervision. While you provided me with valuable feedback and expertise during our meetings, our conversations have always been characterized by problem-solving and open-minded discussions.

I also would like to extend my special thanks to Adrian Wagner who had an important role in shaping this thesis by guiding me and providing relevant veteran insights. I also want to thank all the members of the HCI chair including Lovis Suchmann, who in some or the other capacity have contributed towards my thesis.

I also want to extend my special thanks to all the participants who volunteered for my user studies and spent their time providing me with helpful feedback.

Finally, I want to thank my friends and my family who offered me a comfortable environment while I was working on this thesis.

Conventions

Throughout this thesis, we use the following conventions.

Text conventions

Definitions of technical terms or short excursus are set off in colored boxes.

EXCURSUS:

Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
Excursus

The first person is written in the singular form. Any participant including the user will be referred to by using the pronoun 'he'.

Acronyms are first stated followed by their abbreviations enclosed within parentheses.

Example: DIY (Do-it-Yourself)

The text pertaining to functions and code is written in a different font as shown: `myClass`

The words that need to be highlighted are italicized.

The entire thesis is written in American English.

Marginal notes aim to summarize key points.

Chapter 1

Introduction

A DIY (do-it-yourself) laboratory is characterized as a space where individuals “do stuff”, as in, engage in creative experimentation, crafting, and tinkering within a welcoming, communal environment [Meyer, 2013]. Despite the emergence of the DIY movement in the 1950s and 1960s [Meyer and Vergnaud, 2020], it has gained significant prominence only in the past two decades, evident from the substantial body of literature dedicated to the topic in the recent years. DIY labs can be situated in a variety of accessible settings, such as personal backyards, garages, community areas, warehouses, and similar spaces ([Landrain et al., 2013];[Seyfried et al., 2014]).

The defining features of DIY encompass an informal setup, modest financial investment, a lack of stringent regulations, reduced competition, an absence of specialized marketing personnel [Sarpong and Rawal, 2020], and an absence of documented commercialization strategies. These characteristics stand in contrast to innovation hubs or digital garages, which embody more structured setups, government-backed funding opportunities, regulatory and governance requisites, and access to broader markets [Ng et al., 2020].

Makers are embracing DIY practices to craft items that hold personal significance, ranging from practical alternatives to mass-produced goods [Tseng, 2016]. Upon successful

DIY refers to “doing things”. DIY has gained significant prominence in the past two decades.

DIY setups are hassle-free in many ways compared to the likes of research labs and innovation hubs.

DIY culture is a fusion of individualism, creativity, enthusiasm, and remarkable potential.

expansion, these initiatives offer budding entrepreneurs, novel products, and service innovations akin to those originating within business organizations, carrying the potential to exert a substantial influence on society. Therefore, the DIY culture embodies a fusion of individualism, creativity, and enthusiasm, holding promises of remarkable potential [Dzandu and Pathak, 2021].

As DIY practices increase, authorship and documentation have become equally important.

As a result of the surge in the popularity of DIY setups, the aspect of *DIY tutorial authorship* has become very relevant and pronounced. The topic has gained significant traction and is increasingly being discussed in the context of DIY tutorials and processes in recent years [Kuznetsov and Paulos, 2010]. As the embrace of the DIY culture continues to grow, an imperative arises for the development of solutions and enhancements in the realm of documenting and disseminating knowledge to a broader audience [Tseng and Resnick, 2014].

DIY authorship refers to the step-by-step guides on the DIY process.

Being a DIY tutorial author means creating step-by-step guides and instructions on the DIY process. These authors must possess a unique ability to break down complex tasks into simple, manageable steps that anyone can follow.

DIY authorship is challenging. Challenges range from having to produce good visuals, and documenting steps clearly and comprehensively.

DIY authorship, or self-publishing comes with its own set of challenges [Dzandu and Pathak, 2021]. It can be challenging to convey complex concepts and instructions in a clear and simple manner. DIY tutorials often rely on visuals to demonstrate techniques, materials, and project progress. Authors face the challenge of capturing high-quality images or videos that effectively convey the necessary information [Rigaud et al., 2022]. DIY projects may involve multiple steps, materials, tools, and variations.

DIY authorship challenges in attracting readership and a need for its automation.

Authors need to ensure that their documentation is comprehensive and covers all necessary aspects to attract readership. Documenting DIY tutorials requires significant time and resources. Authors need to plan, execute, and document each step of the project, which can be time-consuming. Documenting physical artifacts and processes also carries a unique set of obstacles, including the restricted ability to manipulate physical documentation interfaces due to the preoccupation of one's hands [Tseng,

2016]. This propels the discussion toward the need for an *automated system* of documentation.

Overcoming these challenges requires dedication, attention to detail, and a commitment to providing valuable and comprehensive documentation that helps readers successfully complete their DIY projects. Due to the multitude of factors that that require consideration, as discussed previously, the process of documentation often becomes arduous and exhausting, often causing additional stress to the author. Hence, there is a need for easier documentation methods and processes and also the possibility to automate such processes. Our focus in this thesis is to provide an automated “document while doing” solution involving the collection of real-time data during a DIY process and thus support DIY authorship efforts. The real-time data is processed in various stages to generate useful insights and visualizations to support the DIY documentation/authorship process.

In conclusion, crafting a comprehensive DIY tutorial necessitates a considerable amount of effort, particularly with a substantial portion dedicated to the documentation process. From our literature survey, we found that the research on providing insights about tool usage in the context of DIY documentation is scarce. Hence, this thesis takes on the task of supporting this documentation endeavor by providing information on elements of DIY processes such as the tool list, order/sequence of their usage, duration of usage, and usage timeline graphs. By providing this preliminary groundwork, the thesis seeks to offer authors a valuable head start as they embark on their documentation journey. The core objective of this system remains around the creation of an automated system that is easily reproducible and supports the documentation efforts of DIY authors.

We propose a “document while doing” approach in an attempt to tackle DIY documentation challenges.

An automated tool-tracking system is proposed.

1.1 Outline

In (chapter 2) we delve into the historical context of DIY processes and their documented evolution. We outline why DIY authorship remains a challenge and elucidate our mo-

History of DIY documentation, its challenges, our motivation, and related work.

tivation for solving it. Furthermore, we will examine a selection of proposed solutions and techniques aimed at facilitating the DIY authoring process to provide the reader with a comprehensive grasp of the ongoing research and the prevalent state-of-the-art.

Chapter 3 discusses our initial approaches

In chapter 3, we discuss our initial attempts since the inception of the thesis and the learning from it that lead to the proposal of our final solution. The chapter particularly aims at giving the reader insights into the strengths and limitations of the approaches tried and builds up a case for how we reached our current proposed solution.

Chapter 4 and 5 discusses the design overview and various components of the proposed prototype.

In chapter 4, we present the design overview of the system and discuss in detail each of the components that comprise the system. We discuss the first two of the three major components that the entire system is divided into, namely 'Camera Setup' and 'Tool Tagging'. The third component, 'Web application' is described in detail in chapter 5, focusing on the design and implementation specifics of the GUI (Graphical User Interface) and the back-end of the application module. Furthermore, we will demonstrate the use of the web application through an example run.

Chapter 6 discusses the technical aspects and specifications along with informal observations of the tool usage by users.

In chapter 6, we provide recommendations on the choice of hardware and software specifications. We also discuss and evaluate the technical aspects of the system by doing a technical study for boundary analysis. We then follow it with the system testing where we test the system for its correctness and accuracy. We summarize the chapter by discussing the results of the tests.

Chapter 7 summarizes the work, and discusses limitations and future work.

In chapter 7, we conclude by summarizing our work and its applications in other related areas. We will conclude the thesis by discussing the limitations of our prototype and suggestions on the potential future work.

Chapter 2

History, Motivation, and Related Work

Due to the increasing popularity and attractiveness of the DIY setups, efforts have been made to provide solutions and techniques to solve the challenges in DIY documentation. In the following chapter, we start by describing the historical precedence of the DIY documentation processes and their evolution, and how it motivated us to work on finding a solution to the challenges posed by it. We then discuss how researchers have identified various facets of DIY documentation and aimed at providing various techniques, and proposals to solve them. In the subsequent subsections, we will focus exclusively on the research done in the area of automated tracking of the tools that are used during the DIY process. The details of the tracked usage order and statistics of these tools form an integral part of the documentation process.

2.1 Historical Precedence

In this subsection, we trace and describe the historical precedence of design documentation. We trace the lineage of non-digital forms of documentation, such as logbooks and journals, to new digital formats including wikis, on-

line portfolios, and video tutorials. We further discuss how software and hardware technological revolutions enables automated documentation processes.

Early Forms of Documentation: Logbooks and Journals

Early forms of documentation include hand sketches and logbooks.

An author's documentation encompasses sketches, calculations, designs, and process descriptions. The earliest manifestations of these documents take shape as tangible journals and logbooks, which hold significant value as legal proof of intellectual property and are therefore indispensable for documentation purposes ([McAlpine et al., 2006];[McAlpine et al., 2017]). Beyond their role in documentation, logbooks play a crucial role in recording a designer's individual decision-making journey, spanning from initial research to design iteration analysis, and act as a tangible reminder of work in progress [Ferguson, 1994].

Information present in the documentation reflects the work of the designer.

By consolidating these types of information in one place, logbooks become objects for reflection in which a designer can review their decision-making process [Lau et al., 2009]. Reflection is considered a critical element of a designer's practice and is commonly integrated into the design curriculum ([Amon et al., 1995];[Adams et al., 2003];[Agouridas and Race, 2007]).

Due to technological improvements we now have 'Electronic logbooks', which are easier to make, maintain, and share.

While logbooks often take the form of physical notebooks, electronic instantiations have been proposed to foster improved communication and information management across teams and corporations [Tichkiewitch and Brisaud, 2004]. Researchers have analyzed the use of physical logbooks with shared virtual file share systems, arguing for hybrid journaling as a means to facilitate shared documentation [Oehlberg et al., 2009], wikis as centralized repositories for design teams ([Walthall et al., 2011];[Yang, 2009]) and electronic notebooks for designers [Hong et al., 1995]. Such documentations leverage the benefits or features of traditional pencil-and-paper documentation while allowing distributed design teams to share digital work. Yet, there is an ongoing research on knowledge capture and "mass collaborative documentation" [Chandrasegaran et al., 2013].

Age of Digital Technology: A Paradigm Shift in Documentation Techniques

There is a rich history of research on the role of technology in supporting documentation and learning. In the 1980s, [Collins and Brown, 1988] described how computers naturally track actions performed by a user. [Lin et al., 1999] promoted reflective practice in which learners monitor, evaluate, and modify their thinking while comparing themselves to peers or experts. Cameras and audio devices play the role of **heart and soul** in modern documentation. Tremendous amounts of research solutions have been proposed using these two devices, forming the backbone of digital narration ([Jacoby and Buechley, 2013];[Raffle et al., 2007]).

A lot of research that aims to provide solutions to visual documentation can be found.

Visuals also play a significant role in the documentation process [Rigaud et al., 2022]. It enables users to see and relate to the DIY process and further stimulates imagination and creativity. A lack of visuals often leads to a gap in understanding and conveyance of the author's ideas. Hence, it is **paramount** that any documentation has ample visuals in them. These issues have always **propelled** researchers to study ways to support users in capturing and navigating documentation, including video authoring tools [Chi et al., 2013] and physical craft processes [Torrey et al., 2009]. The growth of computers enabled and accelerated the process of digital documentation. Digital technology enables sharing of documentations to a wider audience as documented by [Kuznetsov and Paulos, 2010];[Rosner and Bean, 2009];[Torrey et al., 2007];[Wakkary et al., 2015].

Visuals in the documentation are very crucial and enable readers to connect and understand better. Technology can help a lot in this.

Age of Automation

The documentation of a process is **imperative and holds crucial significance**. In contemporary documentation, visuals have become an integral component [Onsès and Hernández-Hernández, 2017]. This is a tedious and demanding task for an author. Consequently, a compelling necessity arises for solutions that eliminate manual toil and introduce automation into the process. Thus, the evolution of digital documentation **must** progress towards automated documentation, reducing the need for substan-

Thanks to technological advances, emphasis is now on automating the documentation process.

tial manual intervention. This is possible with the current technological **proWess** and innovations, and is what serves as our motivation in proposing newer solutions and techniques for supporting automated documentation.

Motivation

The core concept involves leveraging technology and tools to streamline and simplify the creation, management, and maintenance of documentation. It involves automating various tasks involved in documentation, such as content generation, formatting, version control, collaboration, and distribution. **The push in state of the art** is increasingly towards a form of “in-process documentation”, a departure from the traditional post-process approach. Most of the necessary details for documentation are collected during the DIY process itself [Tseng, 2015], to be processed later or in real-time to return a document that adheres to the necessary documentation specifications.

This is one possibility not a single trend. Additionally, only one reference does not support an increased move towards something.

Numerous techniques have been proposed in the area of automated documentation. Learning and inspiration from these general documentation techniques can be adapted and used in the context of DIY documentations, which is the area of focus for our current work. We will now delve into some of the proposed techniques in detail.

2.1.1 DoDoc: A Modular Approach to Record Traces of Activities

DoDoc [Gourlet et al., 2016] is an integrated user interface designed to record traces of hands-on activities. This process of collecting records during embodied tasks serves to enrich reflective practices, both during and after the activities. During the activity, it provides a change of perspective on the ongoing work and after the activity, it enables the review of previous records, offering an overview of the entire process.

The prototype consists of three components: a modular kit with sensors (webcam, microphone, lights), a remote

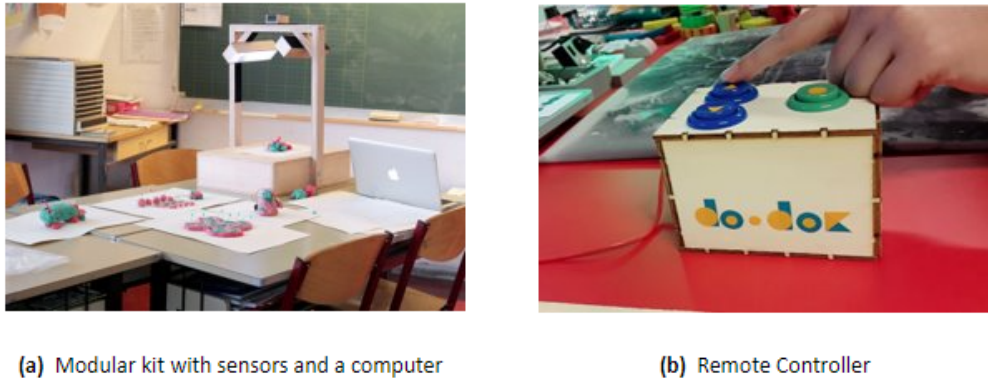


Figure 2.1: The entire DoDoc DIY setup [Gourlet et al., 2016].

controller (Figure 2.1 (b)) with physical buttons for activating picture and video modes, and a computer screen displaying a web-based interface with camera and microphone feedback. Four trace collection modes are offered: images, videos, animation movies, and sound. Its design aligns with the intention of supporting **experiential** learning methodologies by amassing digital materials for reflection during practical activities or play. The setup can be seen in Figure 2.1

2.1.2 Protobooth: A Photo Booth for Prototypes

The *Protobooth* prototype was developed by [Sjöman et al., 2017]. The system functions akin to a conventional photo booth with cameras for capturing images. The system is designed to monitor attributes such as tool utilization, material consumption, user identification, and the mapping between users, tools, and materials.

The users set their prototype inside the Protobooth (Figure 2.2) and authenticate themselves using the RFID (Radio Frequency Identification)¹ card. A successful authentication activates the system and triggers the photographing process of the two webcams. The use of two cameras

The system functions are similar to a conventional photo booth to take pictures for documentation.

Prototype is placed in the booth, pictures are taken using a camera and entered in the database, a timeline of entries is also created for modification at any time.

¹https://en.m.wikipedia.org/wiki/Radio-frequency_identification

is to gain a close to 360-degree view of the prototypes to be photographed. The cameras take pictures of the prototype and subsequently upload them to the repository server (database) and a data entry of the metadata is populated with the information of the time and the user. At any given time, the users can view a timeline of their entries (Figure 2.3) and modify them to add more detailed descriptions in addition to the pictures of the prototype that were automatically added.



Figure 2.2: The setup of the Protobooth where the prototype must be placed. Lighting and camera setup can be seen along with the system status display on the left [Sjöman et al., 2017].



Figure 2.3: Timeline of entries that can be viewed or modified, made using Protobooth [Sjöman et al., 2017].

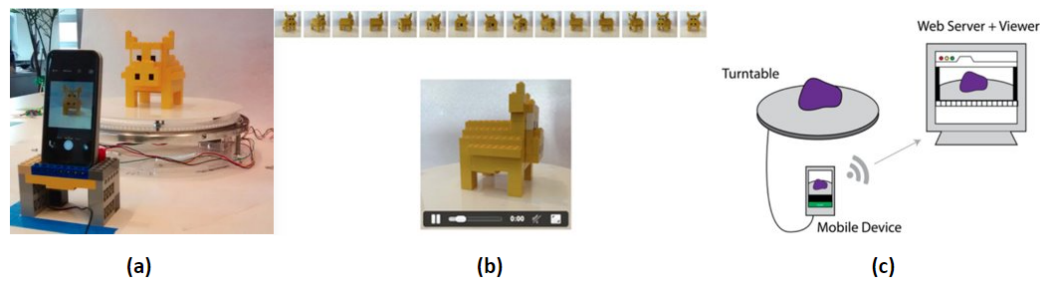


Figure 2.4: (a) The motorized turntable that pairs with a mobile device. This ensures capturing the prototype images in 360°, (b) the images of the prototype taken over time in the form of an animation, (c) a depiction of how the turntable + mobile setup is connected to a web service to be viewed. All images are taken from [Tseng, 2015].

2.1.3 Spin: A Photography Turntable System for Animated Documentation

Spin [Tseng, 2015] is a system that serves as a solution to two key challenges in the project documentation. Firstly, it aims to address the issue of tools for photographing projects often being disruptive to the flow of creating a design project. Furthermore, it also addresses the issues of compilation of documentation into a readable and shareable format being a time-consuming process.

Spin is essentially a photography turntable system for creating animated documentation. It consists of a motorized turntable that pairs with a mobile device (Figure 2.4) to capture 360-degree views of a DIY project at a particular point in time.

These photographs are compiled into an animation of the project called a ‘spin’. As a project is developed over time, these spin animations coalesce into a comprehensive set that illustrates the project’s evolution over time. The mobile device is connected wirelessly to a ‘web service’ for viewing.

Photography disrupts the flow a process, readable and shareable documentations are time-consuming.

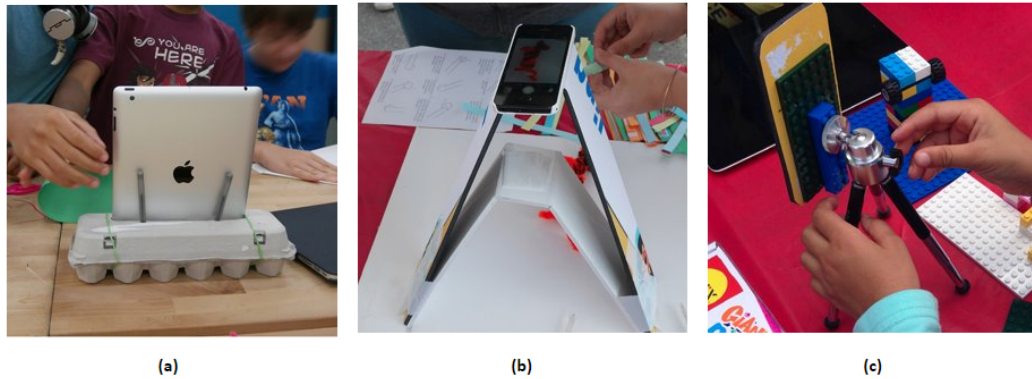


Figure 2.5: (a) The egg carton tablet stand, (b) The poster smartphone stand, (c) The smartphone LEGO® [Wikipedia, 2023] back cover. All images are taken from [Keune et al., 2015].

2.1.4 The Maker Ed Open Portfolio Project

Simplicity and ingenuity of DIY setups

Simple and basic ideas to solve the DIY documentation problem.

It is important to note that the **ingenuity** and popularity of the DIY process lies in the fact that anyone irrespective of age and education can take up the tasks and start applying their individual ideas. Hence, we would also like to discuss a few simple ideas that were proposed by students from a lower age group to demonstrate a contrasting side of simplicity, omnipresence, and ubiquity of the DIY culture.

[Keune et al., 2015] summarizes the possibilities of simple and easy-to-assemble DIY documentation stations. These stations utilize the power of ubiquitously available cameras in smartphones and tablets that can be readily used to capture images and videos.

Creative ideas using everyday materials.

It then discusses coming up with creative and adaptable ways of stably mounting personal tablet computers and smartphone devices, re-appropriating everyday materials for simple assembly as most phones and tablets require a custom mounting device to stay in position and can be costly and difficult to remember to bring along to DIY stations. Figure 2.5 shows a few proposed ideas.

The egg carton tablet stand [Pi'ikea St., 2019] transforms a typical egg carton into a stable, upright cradle for a tablet. It is lightweight, quick to assemble, and easily transportable. With the record stop/start button within close reach, makers can integrate documentation fluidly into their workflow and avoid the overly large data files that come from letting the camera run continuously (Figure 2.5 (a)). The poster smartphone stand [Instructables, 2013] is designed to capture a bird's eye view of projects-in-progress. It is ideal for capturing stop-motion animations and documenting the step-by-step evolution of a product (Figure 2.5 (b)). The smartphone LEGO®² back cover [Recyclelovers, 2014] uses a flat ® piece taped to a smartphone, smartphone cover, or tablet. By attaching another LEGO® piece to a flat surface, the smartphone with the LEGO® back cover can easily be fastened to any place, including a flat wall, the ceiling, the edge of a table, or even a bicycle (Figure 2.5 (c)).

Suggests using (1) Egg carton and tablet, (2) Poster smartphone stand, and (3) LEGO® back cover, as a documentation tool.

LEGO®:

LEGO® is a line of plastic construction toys that are manufactured by the Lego Group. LEGO® consists of variously colored interlocking plastic bricks. LEGO® pieces can be assembled and connected in many ways to construct objects, including vehicles, buildings, and working robots. Anything constructed can be taken apart again, and the pieces reused to make new things [Wikipedia, 2023].

Definition:
LEGO®

2.2 Related Work on Tool Tracking

The focus of our thesis is to support documentation using tool tracking. Tool tracking helps capture all the tools used during the DIY process. As a consequence of this, we can deduce the tool usage order and duration statistics, supporting the documentation process. Object (in our case, tool) tracking has been researched extensively.

During our literature survey, we found papers predomi-

²<https://www.lego.com/>

nantly focusing on techniques that can be classified into two broad categories. The first employs sensors. Multiple sensors are used to collect data with the likes of vibration, audio, heat, and position in order to track objects. The second uses image processing techniques which aim to use visual inputs to build data models that can identify and track objects. We would like to discuss two papers one on each of the before-mentioned categories. Both the papers suggest an 'automated tool tracking' approach.

2.2.1 Instrumenting and Analyzing Fabrication Activities

A modular multi-sensorial system that can capture data from the varied sensors and infer contextual information

This paper [Gong et al., 2019] explores the sensorization of making and fabrication activities, where the environment, tools, and users were considered to be separate entities that could be instrumented for data collection (Figure 2.7). It employs a modular multi-sensorial system that can capture data from the varied sensors and infer contextual information (Figure 2.6).

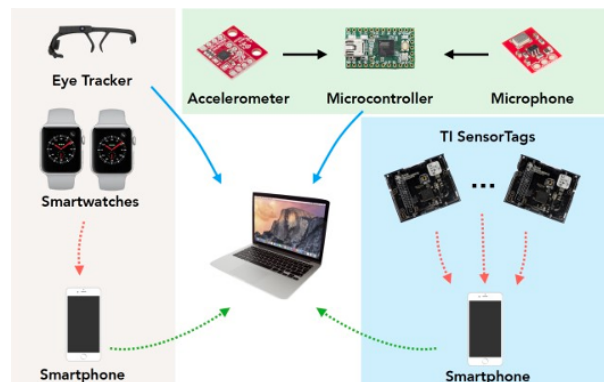


Figure 2.6: Data from multiple sensors being collected to be processed later for gaining insights [Gong et al., 2019].

Video-based sensing is deliberately excluded due to its issue of occlusion.

From the collected data, the authors also predict which activities are being performed, which users are performing the activities, and what expertise the users have (Figure 2.7 (b)). The paper deliberately excludes video-based sensing ([Fails and Olsen, 2003];[Maynes-Aminzade et al., 2007])

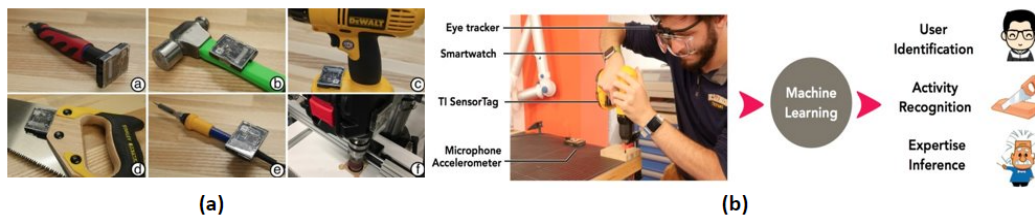


Figure 2.7: (a) The various tools that have been sensorized, (b) the user being sensorized along with tools, and also shows the workflow and the activities involved. Both images are taken from [Gong et al., 2019].

owing to fabrication spaces being large, transitory, and dynamic. Video-based sensing has the issue of occlusion can cause a loss of information from camera streams.

The proposed system exploits audio data for capturing contextual information due to the promising results found in the prior work [Laput et al., 2017]

IMUs (Inertial measurement units)³include an accelerometer, gyrometer, and magnetometer, which can be used to detect orientation, movement, and even subtle vibrations in the objects they are attached to.

The use of machinery may give off some heat, or change how the tools are held. The authors suggest this may cause changes in the amount of ambient light that is detected. While sensing these variations may not be a rich source of information, the authors believe it may provide some useful information at a very low cost.

They also employ a biometric sensor for user identification.

Relevancy?

All the multi-sensorial fusion of data is fed to an ML (machine learning) model for training and prediction. Three elements of contextual information are explored: (1) identification of which user is performing the task, (2) which task the user is performing, and (3) what expertise the user has.

³https://en.m.wikipedia.org/wiki/Inertial_measurement_unit

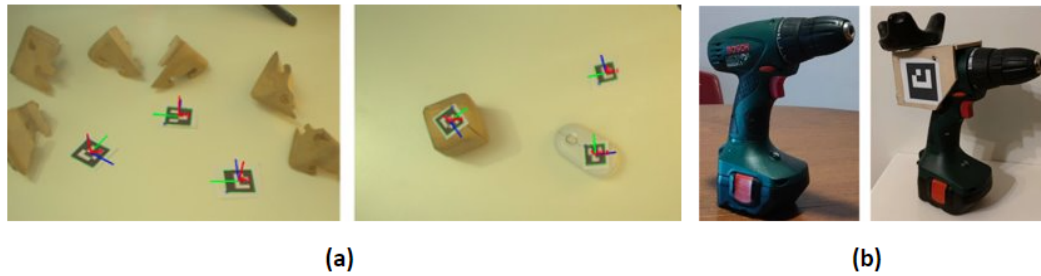


Figure 2.8: (a) The ArUco marker detections and pose estimations, (b) Shows a technique to affix ArUco markers on non-flat tools/surfaces. Both images are taken from [De Feudis et al., 2022].

2.2.2 ArUco-Based Tool tracking

In this study [De Feudis et al., 2022], the estimation of the position of a specific point of interest of the tool that has to be tracked in real-time during an assembly or maintenance procedure using four different vision-based methods namely OpenPose⁴, Azure Kinect Body Tracking⁵, and the YOLO (You Only Look Once) network⁶, and ArUco markers⁷ has been proposed. Results from all four methods are compared with each other. The ArUco marker method can be seen in (Figure 2.8 (a)).

Figure 2.8 (b) shows a technique that has been employed to mount the ArUco marker to the power drill. The proposed approaches were tested on a real scenario with four users handling a power drill simulating three different conditions during an assembly procedure.

To conclude, the advantages and drawbacks in terms of the accuracy and invasiveness of the method were discussed and summarized. This paper familiarized to us, the use of ArUco markers for object tracking.

⁴<https://openposes.com/>

⁵<https://learn.microsoft.com/en-us/azure/kinect-dk/body-joints>

⁶<https://pjreddie.com/darknet/yolo/>

⁷<https://www.uco.es/investigacion/grupos/ava/portfolio/aruco/>

Chapter 3

Initial Attempts: Sensor-Based Tool Tracking and Tracking with Vicon Motion Capture

Sensor-based tracking systems are often tailored for specific applications, yielding **optimal** outcomes when paired with the appropriate sensors. This alignment enhances overall efficiency. With advancements in chip manufacturing, modern sensors are compact and cost-effective. Given the **modest** scale and individualistic nature of most DIY setups [Sarpong and Rawal, 2020], constrained budgets are common [Ng et al., 2020]. **As a result, sensor utilization has emerged as a practical and favored choice for many.**

Moreover, motion-tracking sensors **exhibit resilience against external factors** like motion blur, occlusion, and varying lighting conditions. Leveraging these inherent benefits, we initially embarked on experiments to create a setup using sensors. Subsequently, we uncovered certain drawbacks associated with the sensor approach, which will be elaborated upon in section 3.1.

Sensor-based tracking is a **very popular** means used for object tracking

How does this relate?

Source?

Methods of sensor-based motion-tracking have certain benefits over video-based ones.

Another very popular and widely used device in research for object tracking is a camera. Despite susceptibility to challenges such as inadequate lighting, motion blur, occlusion, and perspective distortion, cameras remain prevalent across a multitude of object-tracking applications.

Source?

Source?

Different types of cameras exist, that can be used for motion tracking and detection.

There are various types of cameras that internally use diverse hardware and correspondingly distinct techniques for image capturing, which can be used for object tracking, examples include: a standard camera that captures pictures and uses image recognition algorithms to identify and track objects, an infrared camera that uses infrared rays to identify and track an object by analyzing the light reflected from the surface of the object, and a thermal imaging camera that captures heat signatures of the object, to name a few.

Source?

We also experimented with the Vicon¹ motion capture cameras which are based on infrared rays. When positioned strategically in the tracking area these cameras can be used for motion detection and tracking of the object. We discovered some limitations when using the infrared-based Vicon system as well. These along with the other setup details are discussed in section 3.2.

3.1 Sensor-Based Tool Tracking

3.1.1 IMU Sensor with a Wired Setup

Our first approach was to use an IMU sensor (MPU-6050) that could be affixed to an object. An IMU (Inertial Measurement Unit) sensor is a device that combines multiple sensors to measure and provide information about an object's orientation, velocity, and acceleration. ^{Why?} Source?

IMU sensors typically include three primary types of sensors:

(1) an accelerometer that measures linear acceleration and detects changes in velocity along different axes. It can de-

¹<https://www.vicon.com/>

termine the direction and intensity of acceleration forces applied to the sensor.

(2) a gyroscope that measures angular velocity or rotational motion around different axes. It detects changes in orientation and helps determine the rate of rotation or angular displacement

(3) a magnetometer that measures the strength and direction of the magnetic field around the sensor. It can provide information about the orientation of the sensor relative to Earth's magnetic field and help determine the heading or compass direction.

By combining data from these sensors, an IMU can track changes in position, orientation, and motion in three-dimensional space.

We also need a microcontroller that can perform the i/o (input and output) operations on the data from the IMU sensor and other processing functions. The Arduino Nano was used for programming. *Why?*

Arduino Nano
[Arduino Official
Store] is used as the
microcontroller.

A breadboard was used as a mounting platform for the Arduino Nano and the IMU sensor. We call the combination of the IMU sensor, the breadboard, and the Arduino Nano the 'breadboard platform setup' (Figure 3.1). Each tool would be mounted with the breadboard platform setup and would be connected to a computer using cables/wires.

Cables were intended for faster live serial data transfer. This approach would work well with fewer tools but was not scalable as we are limited by the number of USB ports on the computer for serial communication. *So wired does not seem to be a good option*

Cables were used for
faster live data
transfer, but had
disadvantages.

That is not a minor issue?

There was also a concern about the cables ending up being tangled and causing disconnection issues, which could affect the tool usage. Other disadvantages included the need to fabricate a casing to keep the sensors and connections safe, making the overall system not so convenient in terms **of ease of reproducibility.**

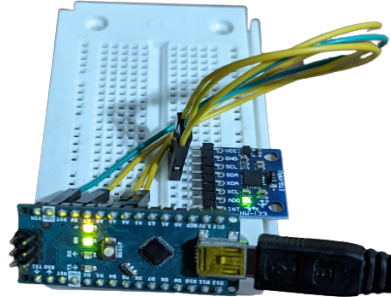


Figure 3.1: Arduino Nano with the IMU sensor in the wired approach.

Wireless methods for live data transfer needed to be used.

This pushed us toward finding compact prefabricated sensors which could communicate with the computer wirelessly. This would eliminate the limitation of the setup being too large due to the breadboard platform setup and allow the possibility of using more tools using a master-slave (also known as: central-peripheral) connection² principle.

Seems obvious and not worth investigating in a scientific work

3.1.2 IMU Sensor with a Wireless Setup

BLE with IMU was used as a slave, and Arduino Nano with Bluetooth module as master.

Our second approach was to use a prefabricated module containing a BLE (Bluetooth Low Energy)³ module for data transmission, an IMU sensor for motion detection, and a 20mm button cell battery to power the entire circuit. This module would act as a peripheral node (also called a slave). It could then be coupled with the module which comprised an Arduino Nano⁴ board and a Bluetooth module (such as the Bluetooth HC-05 module).

²[https://en.m.wikipedia.org/wiki/Master/slave_\(technology\)](https://en.m.wikipedia.org/wiki/Master/slave_(technology))

³https://en.m.wikipedia.org/wiki/Bluetooth_Low_Energy

⁴<https://store.arduino.cc/products/arduino-nano>

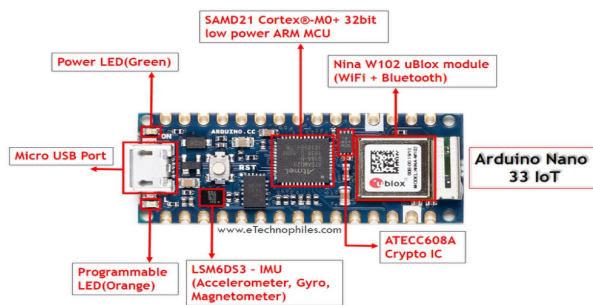


Figure 3.2: Arduino Nano 33 IoT with its important components named [Pinterest, 2021].

We, however, used the Arduino Nano 33 IoT⁵, a microcontroller that comes with an inbuilt low-energy Bluetooth module (Nina W102 uBlox module). This module would act as a central node (master).

It can be noted that two Arduino Nano 33 IoT (Internet of Things) boards can also be used, one with the role of a master and the other as a slave as it also comes with an integrated IMU sensor (LSM6DS3 6 DoF IMU, DoF stands for degrees of freedom) (Figure 3.2).

It is important to ensure the Arduino Nano 33 IoT does not get higher voltages as it works at 3.3V, unlike the Arduino Nano. A higher voltage may damage it.

The prefabricated BLE (Figure 3.3) modules work between 1.8V-3.6V and act like beacons constantly sending data to the master every few intervals (of time). These time intervals can manually be defined. The modules can be mounted onto the tools, each taking up the role of a slave and sending data in the defined intervals to the central master node. This data packet has encoded in it all the information about the connection and the IMU sensor readings.

BLE modules act like a beacon and send readings after every certain time interval.

⁵<https://store.arduino.cc/products/arduino-nano-33-iot>



Figure 3.3: (a) The pre-fabricated BLE module we used [Alibaba, a], (b) shows the internal contents inside the black casing [Alibaba, a], (c) shows the arrangement of button cell battery and the circuit board containing all the necessary sensors [Alibaba, b].

3.2 Vicon Motion Capture

We used the Vicon motion-capturing system (Vicon Bonita Cameras) that was already available at the HCIC (Human-Computer Interaction Center) at RWTH Aachen University, Germany.

Vicon cameras are based on infrared rays.

The Vicon systems utilize high-resolution cameras that are strategically placed around the capture area. These cameras capture the movements of reflective markers placed on subjects or objects. Each camera uses the standard LEDs (Light Emitting Diodes) fitted to its strobe unit and emits infrared (IR) light at 850nm (nanometer) [VICON] (Figure 3.4 (a)).

Either abbreviation and then explanation or the other way around. Not both.

Tool differentiation is realized through distinct arrangements of reflective markers.

As one of our system's objective is to seamlessly accommodate the use of diverse tools within a single process run, a mechanism is needed to differentiate between these tools. In the Vicon system, this differentiation is achieved through the creation of distinct arrangements of reflective markers on each tool.

It is difficult to find unique arrangements for each tool in cases where tools are many or small.

When numerous tools are involved, the task of arranging unique markers can become challenging. A minimum of three markers are required in order for the Vicon tracking system to establish the x,y, and z axes, to be able to identify and later track the object for its position and orientation. This stipulation is difficult to achieve for objects (tools) with smaller dimensions.

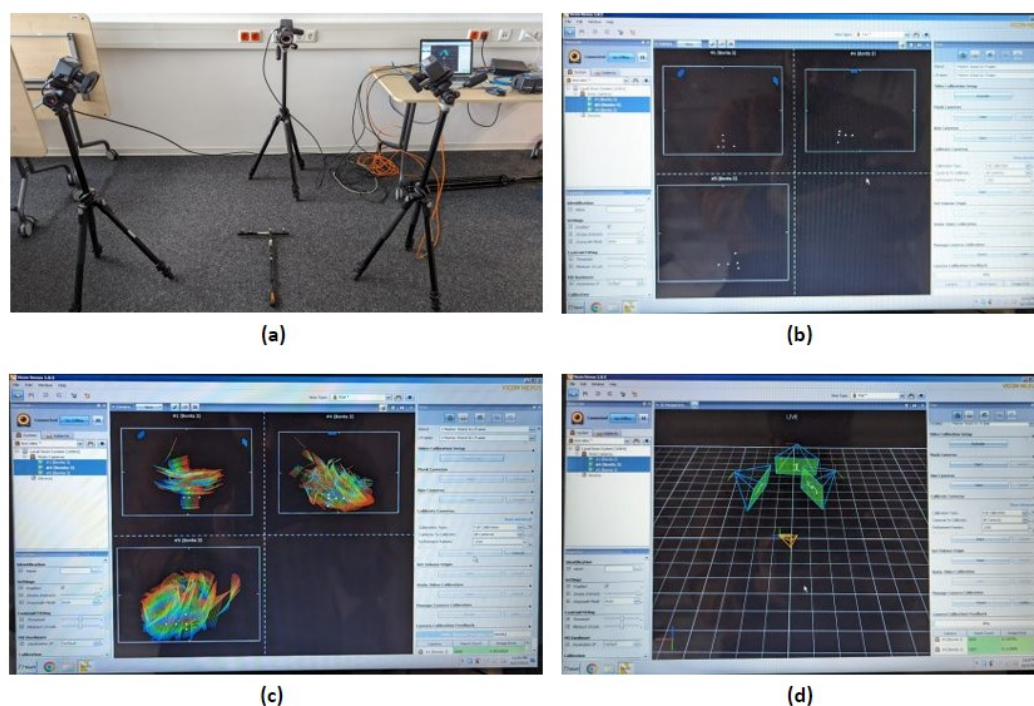


Figure 3.4: (a) The Vicon system setup for our experiments. Strategically placed cameras are being used to detect the 'wand' that has reflective markers attached to it and is lying on the ground, (b) the first step of calibration in the Nexus 1.8.5 tracking software, where we must, in our case, ensure that each of the cameras is able to detect the reflective markers. White dots can be seen in all three black boxes (camera view of each camera used), (c) the second step of calibration where we move the wand around, (d) the cameras successfully detecting the wand.

The Vicon system needs its **fair share** of initial calibrations which can **sometimes** be time-consuming (Figure 3.4 (b)(c)(d)). The presence of occlusions, such as a marker becoming hidden during tool usage, can lead to object detection failure. Either an erroneous result is returned (misidentifying an object due to the undetected reflective marker) or no object is detected.

Initial calibrations of can sometimes be time-consuming.

Furthermore, since the system uses infrared rays and the detection is based on the light that bounces off the reflective markers, shiny surfaces and bright lightning of the working area will in most cases, cause hindrances in detection, again resulting in erroneous or no results (Figure 3.5).

Detection is solely based on light reflection from the markers.



Figure 3.5: The 'wand' along with the custom-made marker arrangements on some tools. It can be seen that the marker arrangements for the pen are not ideal as we are unable to create the distinctive x and y axes needed by the Vicon system to track it in the 3D space. Also, note the need for the objects to be covered in non-reflective tape to avoid unwanted reflections that will cause problems in detection.

These limitations compelled us to opt for a more robust detection method. In our proposed solution, we use a standard camera acting as a central hub, that detects objects via real-time video capture and sends the collected data to a computer connected to it. The computer is responsible for all the data processing. The object (tool) detection is based on the detection of markers (in our case, the ArUco markers) using image processing. These markers are affixed to each tool.

Our solution works on real-time data and even when scaled up only needs the printing of additional markers, which is not cumbersome. In the following chapter, we discuss in detail the design, implementation, and setup of our proposed solution. Source?

Chapter 4

Overview of the System Components

Drawing from the insights and knowledge gained through our preliminary attempts outlined in chapter 3, we proceed to present an overview of the prototype for our developed solution system in this chapter. Furthermore, we will delve into a comprehensive exploration of the distinct components comprising the system.

Figure 4.1 shows the entire setup of the system/prototype. To establish a fully operational prototype, we need to set up three major components of the system. **These components are made by grouping based on the homogeneity and coherence of the tasks and subcomponents that collectively constitute the entire system.**

What does this mean?

The first component is the 'Camera Setup', encompassing the camera arrangement and tripod configuration, a subject we will comprehensively address in section 4.1. The second component of 'Tool Tagging', entails affixing markers to the tools and will be examined in detail in section 4.2. Lastly, the third component is a Flask-based ¹ web application, which **stands as the cornerstone** of the entire system. It provides a GUI for navigating through the DIY documentation process and is responsible for the post-processing of

¹<https://flask.palletsprojects.com/en/3.0.x/>

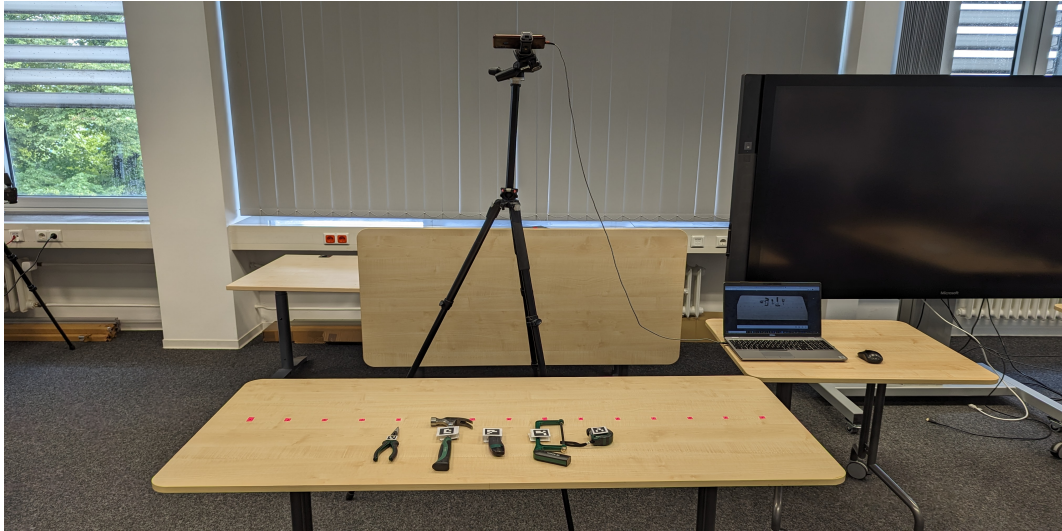


Figure 4.1: The entire setup of our proposed prototype.



Figure 4.2: The three components that make up the entire system of our prototype.

the data collected by the camera. We will discuss this component in detail in chapter 5. Figure 4.2 shows the three components.

4.1 Camera Setup

The camera setup required for the seamless operation of our proposed prototype must fulfill two **vital** requirements.

First, the camera should adeptly detect the markers affixed to the tools at their rest position (when the tools are not being used and are stationary) throughout the entire DIY process without any difficulty. This is an essential prerequisite

and a minimum requirement for an error free functioning of the system. To ensure its fulfillment, the right balance between the choice of camera and the size of the markers becomes critical. We must choose a camera that can capture pictures of the markers in good resolutions. This ensures an adequate number of pixels within each image frame, resulting in a sharp and clear picture, thereby facilitating accurate marker detection during the subsequent image-processing phase (detailed in chapter 5).

The camera must be capable of detecting markers on the tool, when the tool is at rest throughout the entire DIY process, without any difficulty.

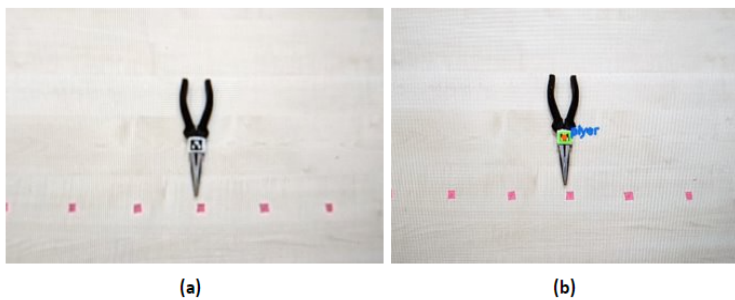


Figure 4.3: (a) The detection is being done at the resolution 640x360. (b) The detection is being done at 2560x1440. Both in (a) and (b) the marker is 25mm x 25 mm. The marker in (b) is detected but not in (a) due to (b) having a higher resolution than (a), indicating using higher resolutions results in better detections.



Figure 4.4: (a) The plier uses a marker of size 25mm x 25mm, while the hammer uses a marker size of 45mm x 45mm. (b) The marker of the hammer is detected while the pliers are not, indicating better detections when larger markers are used.

Better the camera resolution, better the marker detection.

The detection success of a marker, of a given size, is **closely** and positively correlated to the camera's resolution (until a certain threshold, after which the resolution does not have a significant impact, see *Observations*, section 6.4). In simple terms, the better the camera resolution the better the detection (Figure 4.3). Note that changing the size of the markers is also a way to help improve the detection. Larger markers are more easily detected than smaller ones, for a given resolution (Figure 4.4).

The camera is able to tract the entire working area.

Second, the camera is able to produce a frame aspect ratio large enough so that it can capture the entire working area of the setup. Figure 4.5 shows a table which was our working area (area to be tracked). Fulfilling this requirement can again be achieved through the selection of cameras offering higher resolutions. This enables us to choose better and wider aspect ratios for a frame by not having to compromise on higher pixel density. This is essential for marker detection during the image processing step.

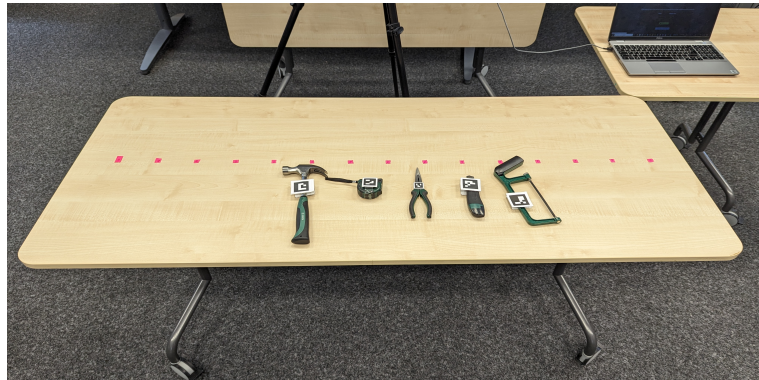


Figure 4.5: The working area or the area to be tracked.

A tripod is useful for camera placement.

A tripod comes in very handy in enabling the user to place the camera strategically to meet this criterion. By utilizing a tripod, users can configure both the horizontal and vertical distances between the camera and the working area, as well as fine-tune the camera's orientation.

Very large working area must be avoided.

In general, we advise against opting for expansive/large working areas. This would require tracking of a large area at all times, meaning a higher or farther placement or both,

of the camera. As the camera-to-working area distance increases, detection becomes more challenging and demands more robust camera specifications.

Additionally, we emphasize the importance of a well-illuminated working area to facilitate the detection process. This directly relates to the camera's aperture size, which determines the amount of light it can take in. A wider aperture allows more light to reach the camera sensor, resulting in **superior** image quality. Modern cameras generally feature adequate aperture sizes to enable enough light to pass through for detection in **decently-lit** environments. However, in instances of damp or low-light conditions, a camera with a larger aperture might be necessary.

Tracking area must be well illuminated.

Because the requisite camera specifications depend on multiple variables such as working area dimensions, marker size, lighting, the nature of DIY tasks (high-speed activities necessitate higher frame rates for marker capture), to name some out of many, we refrain from prescribing specific camera specifications. Discovering the most suitable camera involves trial and error on the user's part to align with the aforementioned requirements.

User must choose a camera with required and suitable specifications.

Another important aspect to consider is the connectivity between the camera and the computer responsible for real-time data collection. This can be established either through wired or wireless means, depending on the user's preference. In our exploration, we conducted experiments using both options. For the wireless approach, we utilized a smartphone's camera alongside a third-party smartphone application named 'IP Webcam'².

The functioning principle of this application involves the computer and the phone being linked to the same Wi-Fi (Wireless Fidelity) network. During our testing, we observed instances where the video stream experienced lags and jitteriness, primarily when the Wi-Fi connection strength was weak. Delays in transmission were observed on several occasions.

Wireless data transmissions tend to have jitters and lags.

²<https://ip-webcam.en.uptodown.com/android>

We chose to transmit data using a USB connection.

After trial and error with different combinations of cameras (example: smartphone camera, standalone camera) and ways of connection (example: wired and wireless with different applications), we narrowed our selection for the final prototype to a smartphone camera connected to the computer via a USB (Universal Serial Bus) connection. Sending the real-time data from the camera to the computer wirelessly was strongly dependent on how good the Wi-Fi connection was, and had occasional jitters and delays. Hence, we decided not to use the wireless mode of data transmission.

For our prototype, we use the integrated camera of the 'Vivo Z1 Pro'³ smartphone. The usage of this specific device is not obligatory. It's worth highlighting that the range of applicable cameras extends beyond smartphones.

During the prototype's development, we also conducted trials using a DSLR (Digital Single-Lens Reflex) camera, a webcam, and the laptop's built-in camera. **When using a DSLR or other comparable cameras, it is required to download the manufacturer's proprietary application in order to be able to access the camera feed.**

We used the Canon EOS 700D⁴ camera along with the 'Canon EOS Webcam Utility Software'⁵ during our experiments. **A DSLR, however, often carries the drawback of being both expensive and bulky, and is not ideal for a easily reproducible and hassle-free solution.**

Why? Comparison between fixed and mobile workspace

To access the smartphone camera via USB from the computer, we use the third-party application 'iVCam' (Figure 4.6). This step was required as the smartphone we used did not allow the direct access of its camera.

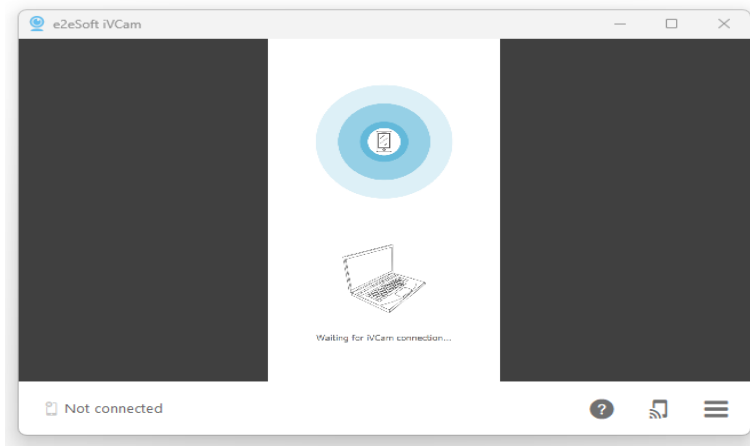


Figure 4.6: The iVCam application interface developed by e2esoft .

4.2 Tool Tagging

Tool tagging refers to the process of mounting/sticking/affixing a marker onto the tools that will be used during the DIY process. This is done so that we can identify and differentiate tools from each other. In our proposed solution we use the ArUco markers (Figure 4.7).

Tool tagging means sticking markers to the tools.

ARUCO MARKERS:

ArUco markers are a type of visual marker commonly used in computer vision and image processing applications. These markers consist of black squares in a grid pattern with a unique identifier encoded within them. The distinctive arrangement allows cameras and software to detect and track them with high accuracy and speed. ArUco markers find extensive application in fields like robotics, augmented reality, and indoor navigation [Romero-Ramirez et al., 2018].

Definition:
ArUco markers

³https://mshop.vivo.com/in/sp/vivo_Z1Pro_launch

⁴https://www.canon.co.uk/for_home/product_finder/cameras/digital_slr/eos_700d/

⁵<https://www.canon-europe.com/cameras/eos-webcam-utility/>



Figure 4.7: Affixing/Tagging tools with different ArUco markers with different IDs to differentiate them.

ArUco markers come in different dimensions.

ArUco markers also **come in different sizes, more specifically dimensions.** The size of ArUco markers can vary depending on the specific application and requirements. However, the most commonly used ArUco markers are square in shape and typically range in size from a few centimeters to a few tens of centimeters.

Why was this not investigated?

The total number of cells in the marker determines its unique identifier

The size is often specified by the number of squares (cells) on each side of the marker, and the total number of cells in the marker determines its unique identifier. For example, a commonly used ArUco marker is the 4x4 marker, which has 4 squares on each side, resulting in a 16-cell marker. There are also larger markers, such as the 5x5, 6x6, and larger, which contain more squares and provide more unique identifier patterns (Figure 4.8).

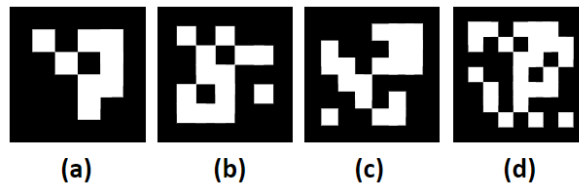


Figure 4.8: ArUco marker with different cell numbers and ID = 0: (a) 16 cells (4x4), (b) 25 cells (5x5), (c) 36 cells (6x6), (d) 49 cells (7x7).



Figure 4.9: (a) The measuring tape is placed on its edge resulting in its marker being obscured from the camera. This must be avoided, (b) Multiple tools are placed over each other again resulting in the obscuring of markers of multiple tools from the camera. This kind of cluttering of tools must be avoided, (c) All the markers are clearly visible to the camera. This is a good practice.

The actual physical size of an ArUco marker can be adjusted based on the application's needs and the distance at which the camera system is placed from it. Smaller markers may be used for close-range applications, while larger markers are more suited for long-range ones.

It is essential to consider the trade-offs between marker size, detection distance, and the available resolution of the camera system to ensure reliable and accurate marker detection and pose estimation. **In this prototype, we experimented with different marker sizes and used them accordingly based on factors like the distance between the tool (with the marker attached) and the camera, the size of the tool, and the area to be tracked. (Figure 4.4).**

How was this tested?

As mentioned previously in section 4.1, **it is an important prerequisite of our proposed solution that the markers are clearly visible to the camera at all times, every time the tool is at rest, during the entire duration of the DIY process.** The details and the reasoning behind this constraint will be discussed in subsection 5.3.3. Figure 4.9 shows good and bad practices.

Special care must be taken about where and how the marker is placed on the tool. A question of how a square marker must be placed on a tool that has curved surfaces arises. This is indeed a challenge. Sticking a marker on a curved surface may lead to partial visibility of it to the camera resulting in a failed detection (Figure 4.10).

The actual physical size of an ArUco marker can be adjusted based on the application's needs. Appropriate marker sizes must be chosen based on the size of the tool and the working area and camera distance from it.

Not entirely true. Workarounds have been found from us here.

Sticking markers on curved surfaces poses a challenge.



Figure 4.10: (a) Two similar ArUco markers are used. One is affixed on the curved part of the tool ('left') and another is just placed on the table ('right'), (b) 'left' has partial visibility to the camera resulting in a failed detection, while 'right' is detected without any problems.

Find flat areas to stick the marker.

One way of tackling this could be to just avoid sticking the marker onto the curved surfaces of the tool and try to find more convenient flatter areas that may exist. In cases where there are no plausible flat areas, a firm square cardboard can be attached to the curved surface and the marker can then be stuck on top of it (Figure 4.11 (b)).

Marker might get damaged if placed in areas that have frequent contact. It can also disrupt the natural intuitive way of doing tasks as the user is conscious about preventing marker damage.

It is also important to avoid sticking the markers in areas of the tool where it is held, like handles. There are three problems with this. First, there is the case of the marker being obscured from the camera when its held, leading to a failed detection. Second, this might damage the marker and hence might result in the marker being undetected.

Lastly, if the user tries to make sure the marker isn't damaged he is likely to consciously focus on doing this. This changes the way the user is using the tool (Figure 4.11 (a)). The user's attention and focus are now diverted away from the DIY task towards making sure the marker isn't damaged.

This is a form of hindrance and breaks away from the natural way of doing a tool-based task. This defeats the aim of our solution to preserve the natural way of tool usage. By recognizing the markers' spatial coordinates, a camera-equipped system can ascertain their position and orientation, thus contributing to accurate object tracking and localization. Due to their simplicity, versatility, and robustness,

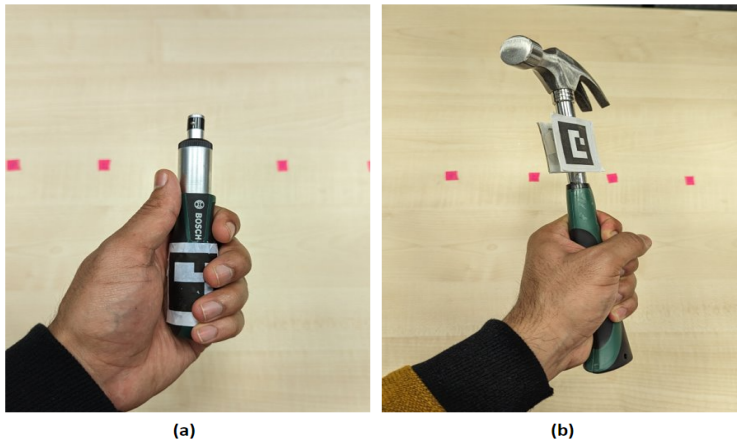


Figure 4.11: (a) The ArUco marker is placed where the tool is most likely to be held causing obscurity from the camera during the tool usage and also has the possibility to get damaged, (b) The ArUco marker is placed in an area that is likely to receive the least amount of contact using cardboard.

ArUco markers have become a popular choice in the realm of computer vision [OpenCV, 2023], enabling a wide range of practical and innovative applications.

Chapter 5

Web Application Component: Design and Implementation

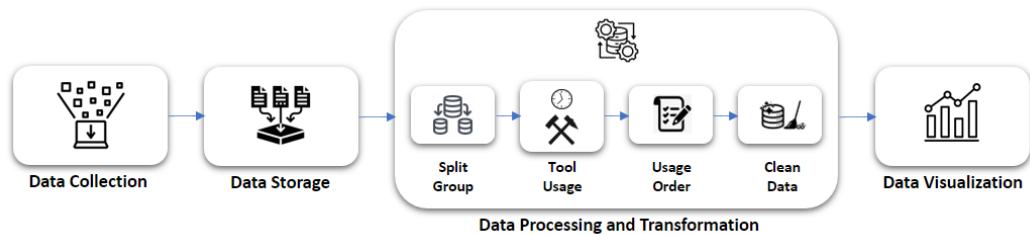


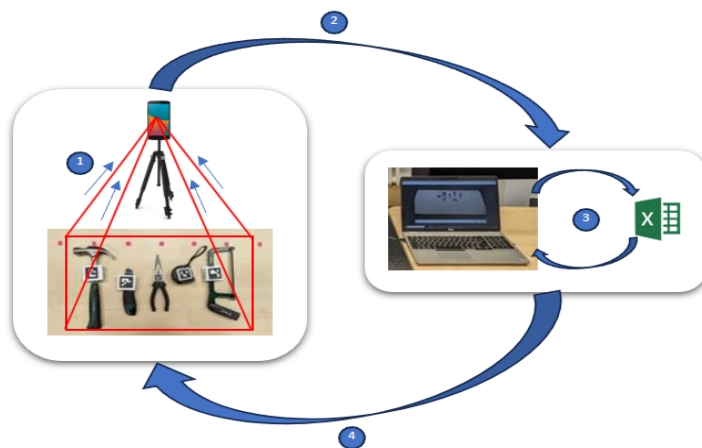
Figure 5.1: Overview and order of execution of all the steps that the web application is involved in.

The camera connected to the computer sends real-time data of the detections made. However, before the final results can be showcased, this data undergoes several phases. This data must be collected, processed and transformed, and stored. We will now discuss the implementation.

Figure 5.1 gives an overview of the order of execution of these various phases. Only after these steps are accomplished can we effectively employ the data for analysis and derive various insights. In this chapter, we will discuss these different phases in detail and also provide the implementation specifics.

5.1 Data Collection

The primary step in this phase is the detection of the ArUco markers by the image processing library, OpenCV ¹, and the collection of this data in a formatted way. Figure 5.2 visualizes on a high level how the data collection happens and the different steps involved in it.



The camera captures a picture frame and sends it to the computer where the marker is detected and this data is recorded. This is repeated in a loop.

Figure 5.2: (1) The camera captures a picture frame of the marker, (2) These picture frames are sent to the computer for detection of markers present in them, (3) The information about the detected markers is stored **in an excel** and the control is returned to the computer, (4) The control goes back to the camera to capture new picture frames. The entire process keeps happening in a loop until detection is stopped manually.

OPENCV:

OpenCV (Open Source Computer Vision Library) is a popular open-source computer vision and machine learning software library. With its versatility, performance, and continuous development, OpenCV has become a fundamental resource for researchers, engineers, and developers in the fields of computer vision, robotics, augmented reality, and other related areas.

Definition:
OpenCV

¹<https://opencv.org/>

We use OpenCV library for marker detections using image recognition

We use the Python library of OpenCV for our prototype. **Since it is an open-source library, contributions are made to it frequently.** The code required for ArUco marker detection is in the library 'opencv-python-contrib'.

Why is this relevant?

Aruco dictionary has a predefined set of markers uniquely designated by different IDs

We start by importing the 'ArUco dictionary'. ArUco dictionary, also known as an ArUco dictionary marker set, refers to a collection of unique visual markers used in the ArUco marker detection and pose estimation system. Each ArUco marker within this dictionary is uniquely designated by an ID or code, facilitating the system's ability to distinguish between multiple markers within the field of view.

The ArUco dictionary typically consists of a predefined set of markers with different IDs and the size and complexity of the dictionary can vary depending on the specific use case and requirements. Having a dictionary of markers with unique IDs is essential to ensure reliable and accurate detection and tracking of markers.

When the camera captures an image or video frame, the computer vision system looks for these markers and matches their IDs to the corresponding dictionary, allowing it to determine the marker's pose and orientation in the 3D (three-dimensional) space.

reference?

The picture frame of tools having the marker is captured by the camera, the markers are detected and the same picture frame is returned with all the details marked.

The function `VideoCapture()` (which is a part of the OpenCV library) lets us capture video frames from a live (real-time) video stream from the camera or video file input. The details about the function can be found in [Appendix A (ArUco Detection)].

These frames are then passed onto the `detectMarkers()` function which is a part of the ArUco module in the OpenCV library. It returns the coordinates of the four corners of the marker along with its ID.

We then use a custom function `aruco_display()` which primarily takes the picture frame and corners as the input. Utilizing these corners as a foundation, we construct edges and subsequently compute the ArUco marker's center. Subsequently, we return the exact picture frame back

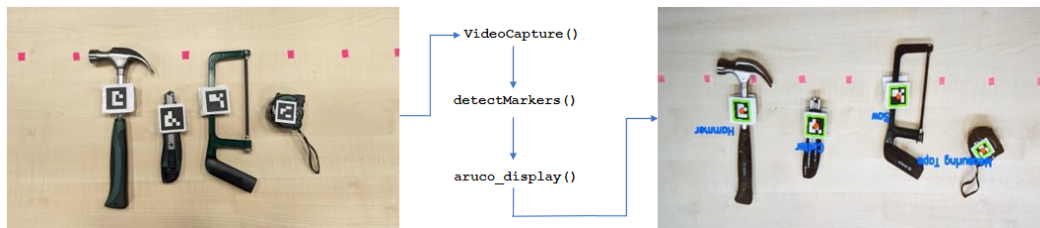


Figure 5.3: The function `VideoCapture()` sees the tools with the markers, captures a picture frame, and sends it to the `detectMarkers()` function that detects the corners of the marker and sends this data further to the `aruco_display()` function that returns the picture frame with marker edges (green), center (red), and marker text (blue).

only now with the marked edges, center, and text indicating the tool associated with each ArUco marker. We use the following color coding for better readability: [edges: green, center: red, marker text: blue] (Figure 5.3).

What is swift and fast in this context? Computer system + actual time

Since this process is swift and happens very fast and in an iterative loop until the detection window is closed, it results in a real-time video stream with detected markers that are marked with edges, center, and marker text.

Capturing of frames keeps happening in a loop.

The details of the tools being used, their names and the IDs assigned to them are all user inputs. This data is garnered via the GUI of the Flask-based web application. The intricacies of how this data is collected will become clear in subsection 5.5.1 during the application run.

FLASK:

Flask is a micro web framework written in Python. It does not require particular tools or libraries.[2] It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself [Flask].

Definition:

Flask

5.2 Data Storage

Raw data must be organized in rows and columns.

What does raw in this case mean?

The data collected in the 'Data Collection' step is raw and needs to be well formatted to proceed further. Hence, the first step is to ensure data is organized and stored correctly as rows and columns. This preliminary dataset will be stored in an Excel file with the filename as provided by the user in subsection 5.5.1.

Why a proprietary file format instead of CSV?

We use four columns, 'Timestamp', 'Tool ID', 'X-coordinate', and 'Y-coordinate' of the marker (where the x and y coordinates pertain to the marker's center coordinates). We are specifically interested in the center coordinates as this is what we will consider to be the location coordinates of the marker from here on.

File handling activities like creating, deleting, and appending are done.

The other operation that needs to be performed is file handling. This includes checking whether the file name provided by the user exists or not and based on this, creating the file, naming it, and setting the folder and file path. Appending data to an existing file is also taken care of as we also allow users to append data of the future iterations to the previous ones if they choose to.

The data generated after each step in the data processing/transformation phase (which will be discussed in section 5.3) is also stored in an Excel file in a formatted way.

5.3 Data Processing and Transformation

We start off by transforming the data entered by the user: the filename (name of the file where the initial data will be stored, discussed in section 5.3), tool names, and their corresponding IDs.

The tools and markers IDs must each be converted to a list and be correspondingly mapped and stored using a hashmap.

The functions `tools_to_list()`, `markers_to_list()`, `create_tools_marker_map_dict()` which are custom defined, do these tasks respectively.

Next, we use the data from the initial Excel file that was generated in the 'Data Collection' step. This file has the x and y location coordinates and the ID of each marker along with the time stamps.

5.3.1 Splitting and Grouping

The aim of this prototype is to be able to depict a timeline that shows the usage, duration statistics, and the order of usage of each tool. To achieve this we need to process and transform the data obtained in the 'Data Collection' phase.

We start off by converting the data in Excel to a DataFrame, for easier data operations using Python libraries like pandas and numpy. We then split the entire DataFrame into sub-DataFrames by grouping tool IDs.

The precision of the timestamps recorded in the initial Excel is in milliseconds, meaning we have multiple timestamp entries for any given second. This is because the camera is capturing multiple picture frames in a single second (as previously explained in section 5.1).

The timestamps are necessary for finding the order of the tool usage and computing the tool usage statistics. We, however, are not looking for such sort of precision in the tool usage durations. A precision of one second is decent and enough for us to provide considerable insights and visualizations.

Hence, we can group these timestamps based on seconds by taking the average of x and y coordinates, eliminating the millisecond precision and also reducing unnecessary computation. This grouping is done for each sub-DataFrame individually, and later these splits are recombined to return the entire data back. The function `split_and_group()` takes care of these tasks.

The data is split based on tools and then each data chunk is grouped by timestamps after reducing its precision from milliseconds to seconds. These data chunks are recombined and returned.

Definition:
DataFrame

DATAFRAME:

A DataFrame is a data structure that organizes data into a 2-dimensional table of rows and columns, much like a spreadsheet

5.3.2 Estimating the Tool Usage

To find out whether the tool is being used or not, we primarily focus on instances when the tool is stationary. This can be estimated based on the x-y coordinates of the tool for a given timestamp. We consider and assign equal weights to both the x and y coordinates.

Let the current timestamp be T_{curr} and its corresponding x and y coordinates be X_{curr} and Y_{curr} respectively.

Similarly, let the next timestamp be T_{next} and its corresponding x and y coordinates be X_{next} and Y_{next} respectively.

Why 1?

If either the modulo differences of X_{next} and X_{curr} at time T_{curr} or Y_{next} and Y_{curr} is **greater than 1**, then it implies either the x or the y coordinates of the tool changed between the time stamps T_{curr} to T_{next} , inferring the tool has moved.

This now gives us a foundation to estimate whether or not the tool was used. We create a new Excel sheet to record this data. Every time the modulo difference is greater than 1, we add an entry '1' to a newly created column called 'used' and '0' if the modulo difference is less than 1 (Equation 5.1). These tasks are accomplished by the custom function `calculate_tool_usage()`.

$$'used' = \begin{cases} 1, & \text{if } |X_{next} - X_{curr}| > 1 \quad \text{or} \quad |Y_{next} - Y_{curr}| > 1, \\ 0, & \text{otherwise.} \end{cases} \quad (5.1)$$

For an accurate assessment and estimation of tool usage, a pragmatic approach is essential, accounting for how tools are genuinely utilized in real-world scenarios. Merely altering the tool's position cannot be regarded as a genuine indicator of its usage.

Hence, in our study, we employ the rule of a tool considered as being used, if there are more than two continuous occurrences of 1s in the 'used' column. Hence, a count of three consecutive 1s is considered as the tool being used. This consideration is based on our user tests where we found that it takes on an average a minimum of two seconds to pick up the tool and place it back down, without performing any action.

Also, there are situations such as: the tool accidentally being picked up and immediately put back down (happens within two seconds), the table on which the tools are placed receiving a jerk due to accidental contact, accidental and unintentional contact with the tool during a task which resulted in it shifting its position by a small amount, errors frames were recorded by the camera, and so on. All such cases result in data recordings where the number of consecutive 1s in the 'used' column is either one or two.

Hence, we consider a tool as having been used to perform some real-life task only if it was used for more than two seconds and in all other cases where it is less than or equal to two seconds we simply ignore and remove all such instances from the data.

This consideration inherently also implies that the consecutive/continuous occurrence of more than three 0s in the 'used' columns means that the tool's position was unchanged for more than three seconds indicating that it wasn't used or was at rest.

5.3.3 Suggesting Tool Usage Order

The `suggest_order()` function is used to first calculate the tool usage duration of each tool and then to calculate

Which user tests? What was the structure of these tests?

A realistic approach must be used. Just a coordinate change cannot be considered tool usage as these might be accidental. Only genuine tool usage must be considered. We come up with a solution for this based on a few basic user experiments.

Why not mark it to provide feedback to the user?

the tool usage order. For this, we need to identify the start and end timestamps for each tool. This is done on the basis of the first and last occurrences of more than three consecutive 1s in the 'used' column respectively.

The algorithm used to find the start and end timestamps can be found in the Appendix A. We then sort the tool usage based on their start timestamps to order it. This is now the order of tool usage. A DataFrame with the start timestamp, end timestamp, tool ID, and tool usage duration is returned and also stored in an Excel file.

Marker must be visible to the camera for data to be recorded.

An evident constraint of our prototype revolves around the prerequisite that the marker affixed to the tool must remain within the camera's visual range for successful detection, thereby initiating the data collection process. If the marker is not detected, no data will be recorded.

For the majority part, the marker goes undetected during tool usage.

Based on the observations from our comprehensive experiments and analysis, we discovered that in the majority of instances, the marker remains undetected while the tool is being used. This is surely a limitation of our system which will be addressed and discussed extensively later in section 7.3.

This can be attributed to several factors including fast movements of the tool resulting in motion blur and the camera being unable to capture the image of the marker, situations where the marker partially or fully gets obstructed/covered by the user's hand or other objects causing occlusion and the marker detection to fail (ArUco markers need all the corners and patterns inside the square to be visible for the correct detection as seen previously in section 5.1), to name a few.

To tackle this issue, we propose the idea of a constraint that is a necessary requirement for the solution to work.

We focus primarily on the tool rest timestamps and not the tool usage timestamps. We must find a way to get the right start and end timestamps of the tool, which are the two rest states.

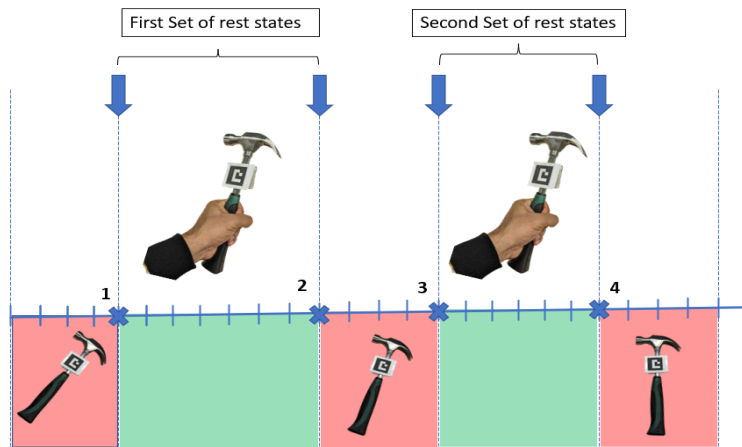


Figure 5.4: The figure represents the tool usage timeline. The red areas depict times when the tool is at rest and the green areas depict the times when the tool is being used. We need to look for consecutive 'rest-to-use' and 'use-to-rest' transition pairs. The sum of differences of all such pairs gives the entire tool usage duration. In this case $[(\text{time}(2) - \text{time}(1)) + (\text{time}(4) - \text{time}(3))]$

This can be ensured by making sure the markers that are attached to the tools are clearly visible to the camera without any obscurity, at all times whenever the tool is at rest, during the entirety of the DIY process. The time between the two rest states (time between start and end timestamps) of the tool is considered as when the tool is being used (Figure 5.4).

Hence, in conclusion, the tool not being detected during usage is not a matter of great concern as we propose to primarily focus on the start and end (the two rest states) timestamps, which indirectly helps us in deducing if the tool was being used or not.

However, although as mentioned previously about the high chances of the tool marker going undetected, it is important to still keep tracking the tool at all times during its usage, as there could be occasional detections which will help in asserting that the tool was indeed being used based on position changes corresponding to the timestamps.

This also becomes essential if the user decides to use a recorded video as the input instead of real-time tracking. Either a slow-motion video where the frame rate is higher or a regular video with a frame rate setting in the code can be used as the input.

This will ensure that the OpenCV image recognition algorithms get enough additional time to analyze each frame often resulting in better detections than when done in a real-time scenario. During our testing, we observed a higher rate of successful marker detections for slowed-down videos shot at high resolutions as input compared to live stream input.

5.3.4 Creating Uninterrupted Continuous Time Series Data

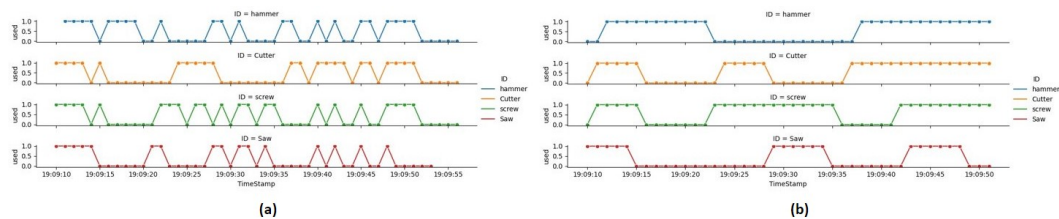


Figure 5.5: (a) The data used for visualization is unclean, meaning there are a lot of missing data points that are just considered as '0', which is evident from the time series visualization. (b) The data used for visualization is cleaned to remove inconsistencies and missing data points resulting in a continuous smooth time series visualization.

Times when the marker wasn't detected results in data gaps.

As mentioned in the previous section, the consequence of the tool going undetected is that no data is recorded at that particular instance of time. This naturally creates data gaps in the timeline resulting in discrete and sporadic data at the end of the DIY process. Gaps in data result in a fluctuating timeline visualization with constant dips (gaps are missing data and are considered as '0') and rise (Figure 5.5 (a)).

Data gaps must be taken care of for insightful visualizations.

This needs to be fixed as visualization of the whole timeline of the process necessitates a time series data block with uninterrupted contiguous timestamps. We take care of this in the `create_clean_data()` function.

We first create a new empty DataFrame and then populate it with the 'start timestamp' of the first tool used as the first entry. We then enter the timestamp in increments of one second until we reach the 'end timestamp' of the last tool used. This now gives us the timestamp interval to generate a timeline of the entire DIY process.

The DataFrame also has a column named 'Used' which describes if the tool was used or not using '1' and '0' respectively (conventions similar to the ones used in subsection 5.3.2). All entries in the 'used' column are initially set to the default of '0'.

We create such DataFrames for each tool and create a list of these DataFrames. Let's call these DataFrames *clean_data_t* where *t* is the tool ID and the list of these data blocks is *clean_data_list*.

The original data block containing the actual recorded data is also split into smaller data blocks on the basis of each tool and a list of these data blocks is created. Let's call these original data blocks *original_data_t*, *t* being the tool ID and the list of these data blocks as *original_data_list*.

We loop over entries in *original_data_t* and map (copy) it to the corresponding entries in *clean_data_t*. For each *original_data_t* in *original_data_list* a mapping for the corresponding *clean_data_t* in *original_data_list* is made. Note that the correspondence is based on the tool ID, *t*.

The code can be found in [Appendix A (Data Processing and Transformation)]

The entire timeline is first filled with 0s. This data chunk is replicated for each tool. Next, for each tool, 1s are filled based on the tool usage data recorder. All the data chunks are then combined. This gives us smooth contiguous non-sporadic time series data, well-suited for visualizations.

5.4 Visualizing the Data

We now have continuous non-sporadic time series data that is ready to be plotted. Visualizing the data helps us better understand it and gain interesting insights about the DIY process. We use this data to visualize three different plots.

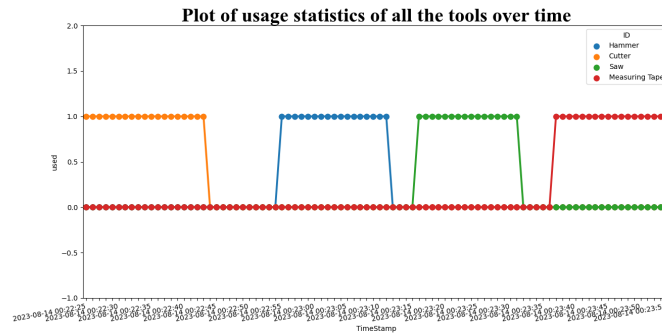


Figure 5.6: Combined time series plot of all the tools for the entire process.

First is a time series plot which has the usage timeline of all the tools in one single plot stacked on top of one another. In the graph, '1' indicates the tool being used, and '0' indicates the tool being at rest.

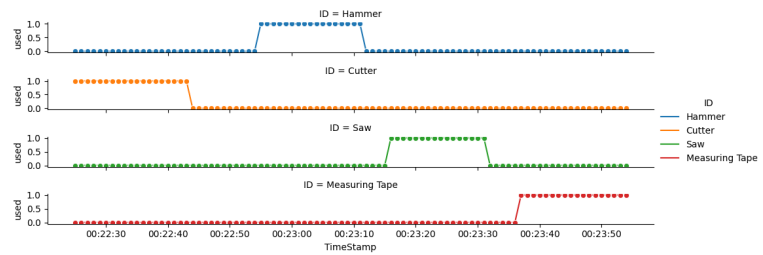


Figure 5.7: Separated time series plots of each of the tools for the entire process.

Second gives a separate plot of the timeline of each tool separately for better visualization and individual tool analysis.

Third is a KDE (Kernel Density Estimate) plot. With a KDE plot, we are able to visualize the tool usage density and also estimate at what point of time in the entire timeline the tool usage was the maximum.

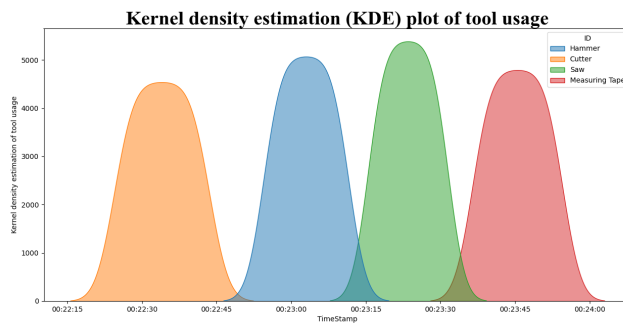


Figure 5.8: KDE (Kernel Density Estimate) plot that visualizes the probability density estimate using peaks. Higher the peak, the more the usage density.

KDE PLOT:

A KDE plot is a method for visualizing the distribution of observations in a dataset, analogous to a histogram. KDE represents the data using a continuous probability density curve in one or more dimensions. It produces a plot that is less cluttered and more interpretable, especially when drawing multiple distributions. [seaborn.kdeplot - seaborn 0.12.2 documentation].

Definition:
KDE Plot

5.5 Application Run

In this subsection, we will do an example run for an entire DIY process in order to demonstrate how the application must be used from start to finish. We have developed a flask-based web application for our proposed prototype. Figure 5.9 shows all the phases involved in the application run.

The details of the flask-based configurations, files, plugins, and package versions will not be discussed in this subsection as the scope of this subsection is to focus more on the code and implementation of the application. It is however important to note that using different OpenCV package versions might result in the code not functioning due

Flask-based configurations and package version details are not discussed here.

to possible compatibility issues. These could be a result of the changes done to these packages/libraries. More on this will be discussed in section 6.3.

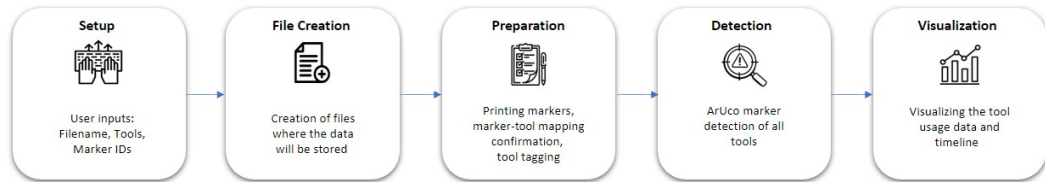


Figure 5.9: Overview and order of execution of all the steps that the web application is involved in.

5.5.1 Setup Phase

We start at the ‘home’ page. We call this the ‘Setup Phase’. This is the page where we enter all the basic details. The user enters the name of the file where the real-time data recorded from the camera should be stored. The user then enters the name of all the tools with the corresponding ArUco marker IDs that he wishes to use during the DIY process. The user can add or delete as many tools as he wishes (Figure 5.10).

User inputs like file name, tool list, and their corresponding marker IDs are collected.

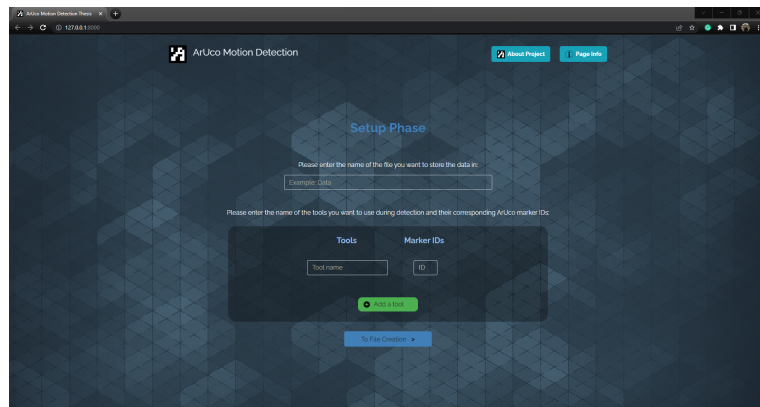


Figure 5.10: The GUI of the home page. This is what users see when the application boots up.

For this example run, we name the file ‘Example_run’ and add the tools: Hammer with ArUco marker ID 1, cutter

with ID 0, Saw with ID 2, and Measuring Tape with ID 3. We then click on the 'To File Creation' button to proceed to the 'File Creation Phase' (Figure 5.11).

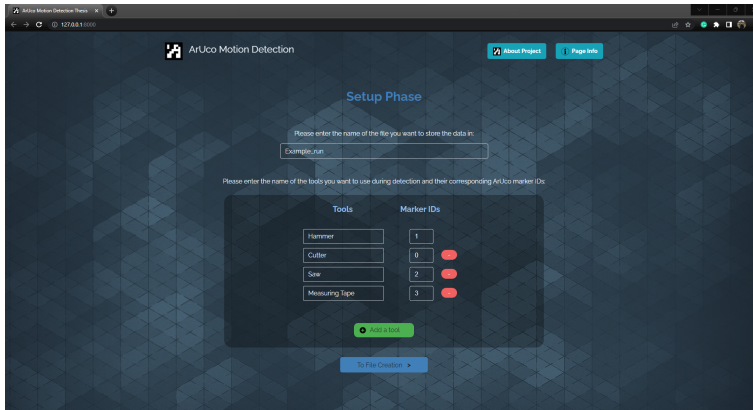


Figure 5.11: The user has filled in all the setup details (file-name, tools, and their corresponding ArUco marker IDs).

5.5.2 File Creation Phase

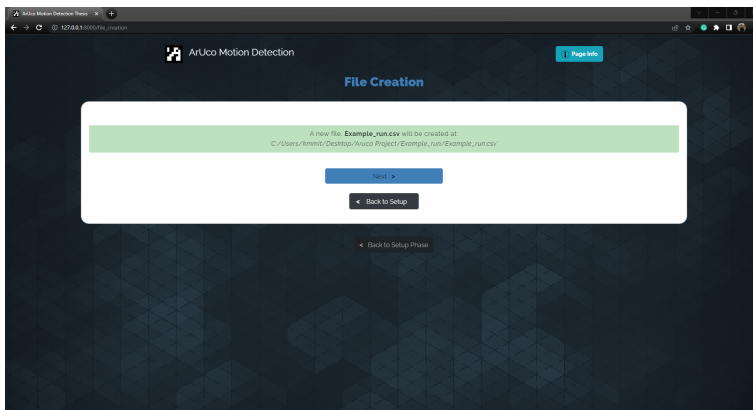


Figure 5.12: The message shows the file does not exist and has been created successfully in the mentioned path.

If the file does not exist, a new file will be created. We click on the 'next' button to proceed to the 'Preparation Phase' (Figure 5.12).

New file is created.

Append or delete to create a new file options are given.

If the file already exists, the user is given an option to either append the data that will be generated in the current run to the existing file or delete the existing file and start with a new file. The user can either click on the 'delete' or the 'append' button to proceed to the 'Preparation Phase' (Figure 5.13).

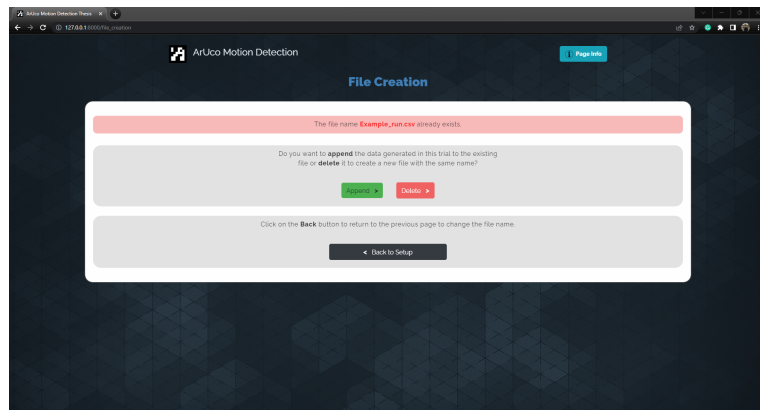


Figure 5.13: The user has the option to either append the data generated in the current run to an existing file or delete the existing file and create a new one.

5.5.3 Preparation Phase

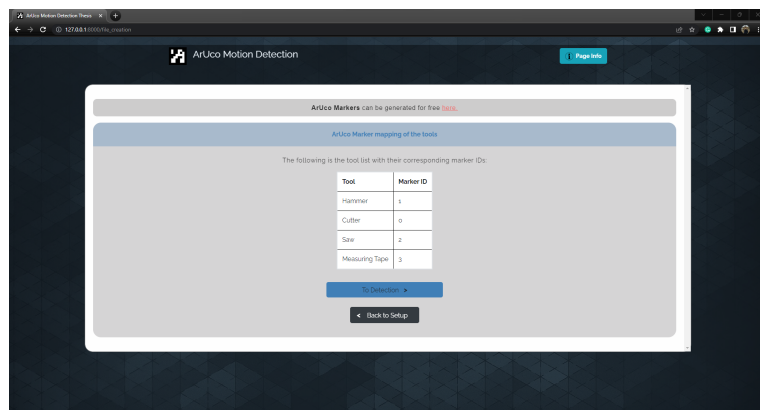


Figure 5.14: The user has the option to either append the data generated in the current run to an existing file or delete the existing file and create a new one.

In this phase, we display a table with the tool list and their corresponding markers as entered by the user, so that he can confirm it before proceeding further to the detection/-capturing step. The user has the option to go back using the 'Back to Setup' button to change the entries (Figure 5.14).

Confirmation of the tools and their corresponding marker IDs.

We also provide the link to a website from where the user can download and print the ArUco markers for the tools for free [Online ArUco markers generator]. The user has to enter the dictionary type, marker ID, and the dimensions of the marker to be generated. He can then print it out to affix them to the tools.

Print ArUco markers based on required type, ID, and size.

Since we use the markers with IDs '0', '1', '2', and '3', we print the markers as seen in the Figure 5.15.

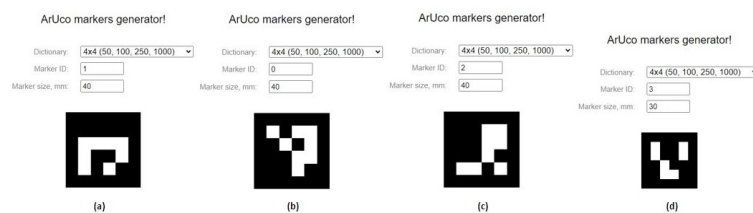


Figure 5.15: (a) The ArUco marker with ID '1' and size 40mm x 40mm to be affixed to the hammer, (b) The ArUco marker with ID '0' and size 40mm x 40mm to be affixed to the cutter, (c) The ArUco marker with ID '2' and size 40mm x 30mm to be affixed to the saw, (d) The ArUco marker with ID '3' and size 30mm x 30mm to be affixed to the measuring tape. All markers are generated using [Online ArUco markers generator]

The tools with their markers attached are placed on the table and can be seen in Figure 5.16.

Once we are done with affixing the tools with the markers we click on the 'To Detection' button to proceed to the 'Detection Phase'.



Figure 5.16: The printed ArUco markers with their IDs have been affixed to the corresponding tools respectively.

5.5.4 Detection Phase

Inform the user about our prototype constraint.

In this phase, we start off by informing the user about the minimum requirement/constraint of our prototype as discussed in subsection 5.3.3, about the markers attached to the tool being visible to the camera at all times when the tool is at rest, throughout the DIY process. We click on the 'Understood' button to proceed to the next step (Figure 5.17).

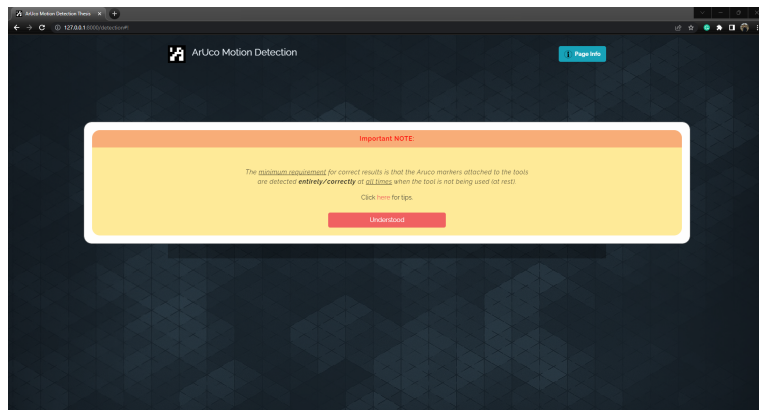


Figure 5.17: Pop-up showing the minimum requirement/-constraint of our prototype.

We then click on the 'Start Video Stream' button to initiate the capturing process. A window that will capture frames in real-time pops up (Figure 5.18).

Click on "Start Video Stream" to start the detection process.

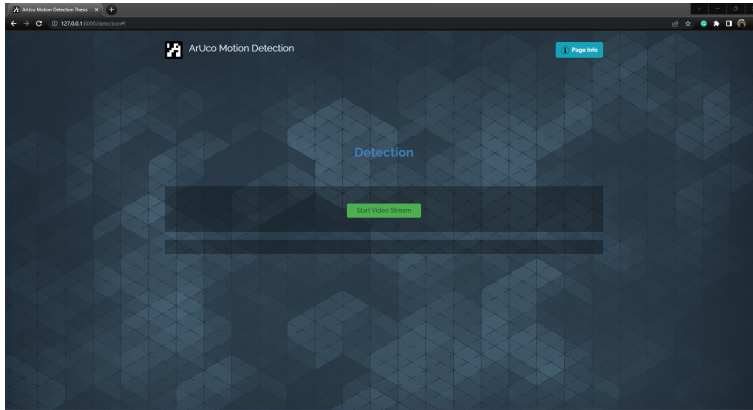


Figure 5.18: The page shows the 'Start Video Stream' button which opens with the detection/capturing window, (b) The pop-up informing the users to click on the close (x) icon of the capturing window to close it.

A decision to include a pop-up informing the user necessitating the clicking of the 'X' icon on the capturing window to end the capturing process was made as during our user studies we found on multiple occasions that the users did not know how to stop the capturing process (Figure 5.19).

Pop-up informing the user on how to end the detection process.

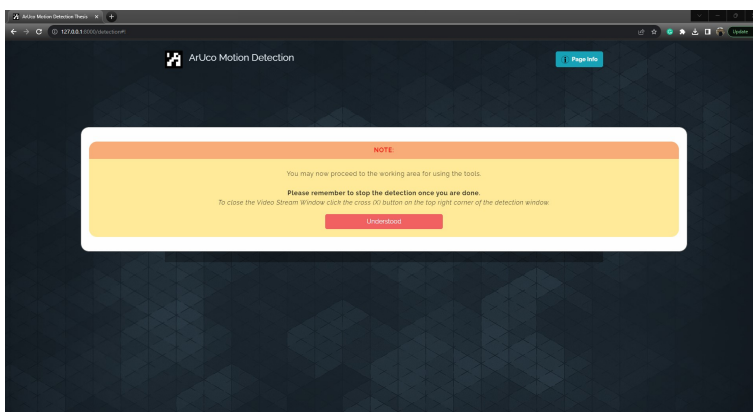


Figure 5.19: The pop-up informing the users to click on the close (x) icon of the capturing window to close it.

Also, notice a message prompting the user to now proceed to the working/tracking area to use the tools. This was added as we observed users not knowing what to do next and had to be reminded to move away from the web application and start with the tool usage tasks. (Figure 5.19).

Console output
printing detection
details.

The console prints out details about the real-time detections that are being made. Figure 5.20 shows the detection window showing detections of the ArUco markers affixed to the tools. Figure 5.21 shows the console output of the detections.

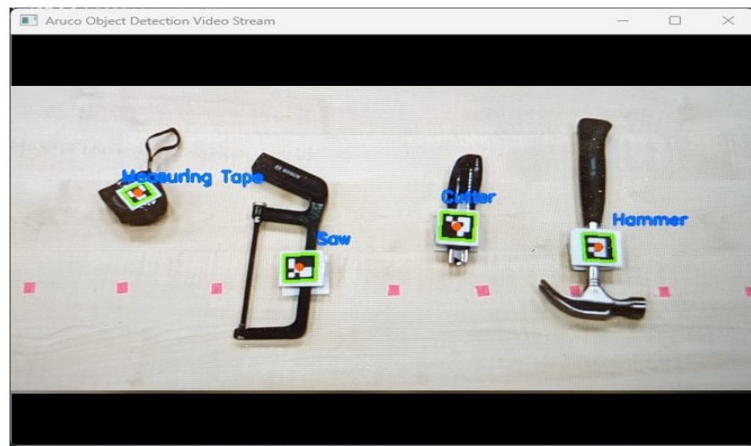


Figure 5.20: The detection window showing the detections of the ArUco markers affixed to the hammer, the cutter, the saw, and the measuring tape.

```

Run: app
Detected 'Measuring Tape' with Aruco Marker ID: 3 | X Co-ordinate: 238, Y Co-ordinate: 294
Detected 'Hammer' with Aruco Marker ID: 1 | X Co-ordinate: 353, Y Co-ordinate: 189
Detected 'Saw' with Aruco Marker ID: 2 | X Co-ordinate: 233, Y Co-ordinate: 186
Detected 'Cutter' with Aruco Marker ID: 0 | X Co-ordinate: 352, Y Co-ordinate: 297
Detected 'Measuring Tape' with Aruco Marker ID: 3 | X Co-ordinate: 246, Y Co-ordinate: 296
Detected 'Hammer' with Aruco Marker ID: 1 | X Co-ordinate: 359, Y Co-ordinate: 190
Detected 'Saw' with Aruco Marker ID: 2 | X Co-ordinate: 241, Y Co-ordinate: 188
Detected 'Cutter' with Aruco Marker ID: 0 | X Co-ordinate: 359, Y Co-ordinate: 299
Detected 'Cutter' with Aruco Marker ID: 0 | X Co-ordinate: 370, Y Co-ordinate: 301

```

Figure 5.21: The console output gives information about the detected ArUco marker ID, its associated tool, and the coordinates.

We then end the capturing process by closing the capturing window and proceed to click on the 'To visualizations' button to proceed to the 'Visualization Phase'.

5.5.5 Visualization Phase

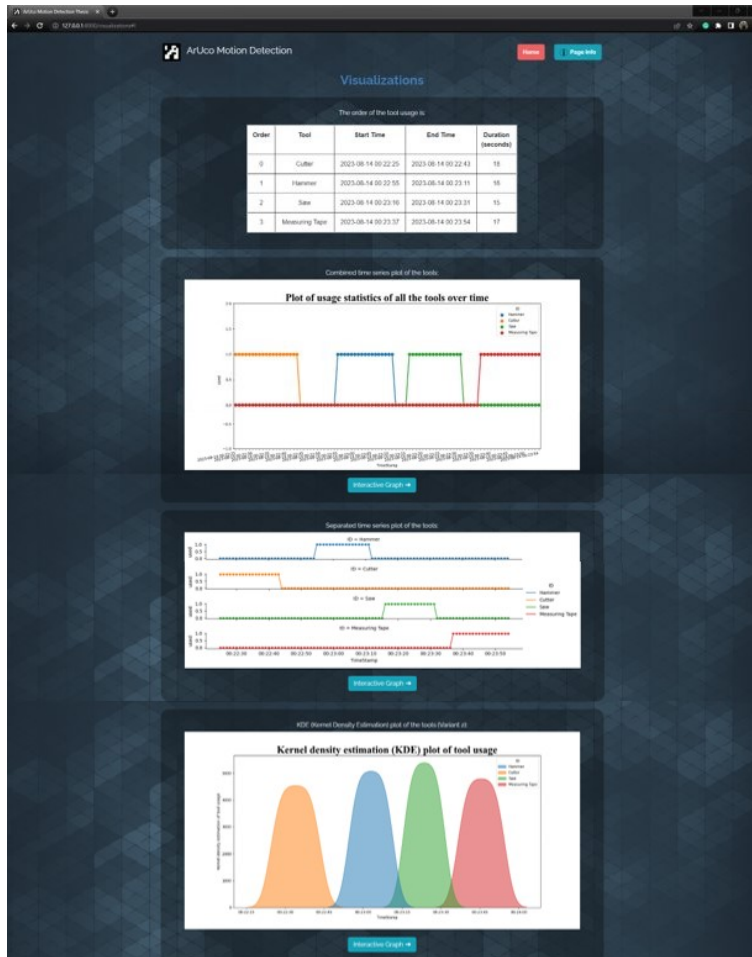


Figure 5.22: Combined image of the tool usage order and duration statistics along with all three visual plots.

In this phase, we visualize the time series data and plot the tool usage graphs for the entire timeline of the DIY process. We also display the tool usage order along with the duration statistics. The three different graphs are: (1) The combined tool usage plot of all the tools, (2) The separated tool usage plots of each tool, and (3) The KDE plot of all the tools (Figure 5.22).

There is also an interactive graph option for each graph, on clicking of which the user is taken to a new webpage where

The three visualization graphs with tool usage statistics.

Interactive graph.

he can interact with the graph by zooming in, sliding, and scrolling for possible further and deeper analysis of data points (Figure 5.23).

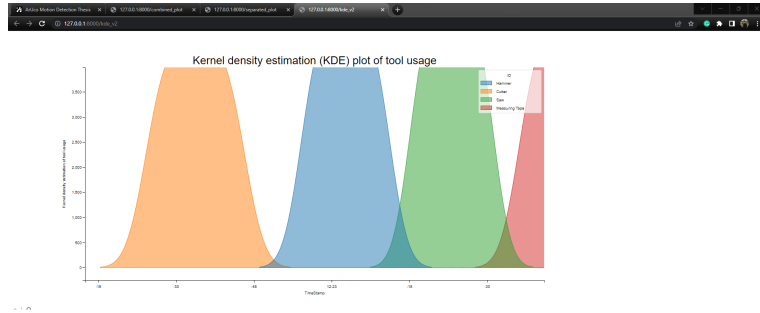


Figure 5.23: It can be seen that the KDE plot is zoomed in.

Click on the 'Home' button followed by the 'Yes' button to begin a new iteration.

Clicking on the 'home' button located at the top right, allows the user to exit the 'Visualization Phase'. On clicking the 'Yes' button the user goes back to the 'home' page (setup phase) to start the next run/iteration (Figure 6.1). On clicking the 'No' button the user stays on the same page (visualization page).

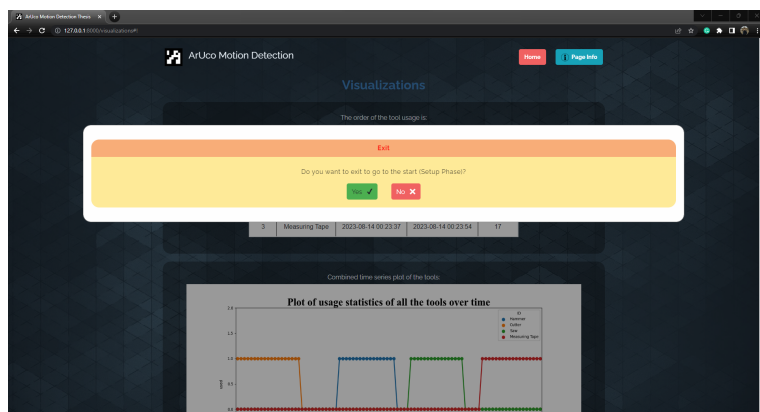


Figure 5.24: The application allows you to go to the home screen (Setup phase) to start a new run/iteration.

Doing multiple application runs through user studies also helped us in evaluating the system to find out the flaws and

inadequacies in the GUI interface. Fixing these helped us to improve the prototype. **We discuss more about the user study in section 6.5.**

Chapter 6

Technical Specifications, Tool Testing and Results

Our goal for the proposed solution is to support the users in their DIY authorship process. In doing so, we also aim for the solution to be reproducible with minimal effort to ease the entire prerequisite setup process for the user. Hence, it is essential the technical aspects of the prototype are discussed to give the user an idea of the software and hardware requirements. This helps the user to estimate the effort and the financial means necessary to assemble such a system himself. This chapter will provide insights on these specifics.

We divide the chapter into five subsections. The first two subsections discuss the hardware requirements: the first, discusses the *camera specifications* and the second discusses the *computer specifications*. The third subsection discusses the *software requirements*. In the fourth subsection, we summarize the *technical boundaries* of our prototype for different combinations of the two parameters: the *resolution of the camera* and the *size of the ArUco markers*. We conclude this chapter with the fifth subsection where we conduct *Tool testing trials* and report our results from the these trials.

As mentioned in the section 4.1, the camera required is dependent on factors including dimensions of the working area, the size of the markers used, and the lighting to name

a few. It is at the user's discretion to decide on the camera requirements based on the needs of the DIY setup and the tasks involved. We, however, want to provide an analysis of the statistics of different cameras available on the market.

We look at the analysis done by the German online platform 'Statista' that gathers and visualizes data offering statistics, reports, market and product insights [eDesiderata, 2022]. According to the report [Statista Daily Data, 2012], 81% of all phones in 2012 had built-in cameras in them. In 2013, the analysis done by [ThinkTAP, 2013] already shows an increase of 2% from 2012 and reports that 90% of the people preferred to take pictures using smartphones compared to standalone cameras. The analysis also predicts that this trend is most likely to continue. [Oberlo, 2023] also reports an annual average increase of 10.3% in smartphone users between 2016 to 2023.

Most smartphones today have built-in cameras. Majority of the people prefer using phones compared to standalone cameras.

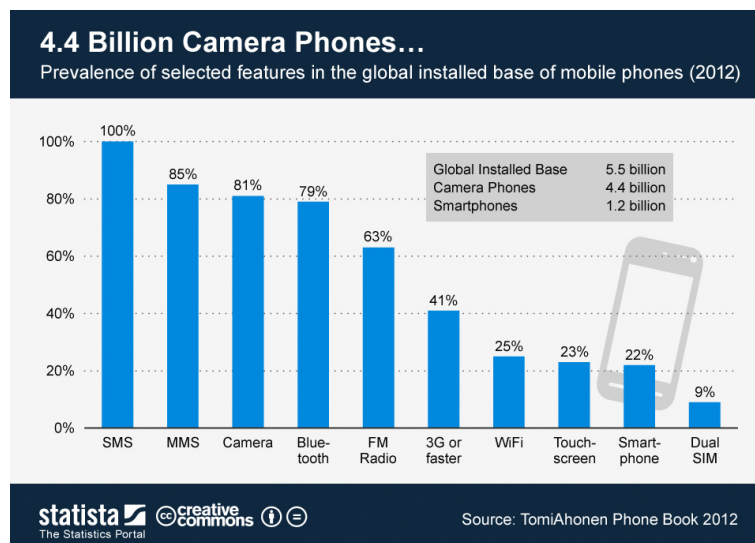


Figure 6.1: Statistics about the percentage of smartphones that come with a built-in camera [Statista Daily Data, 2012].

[Statista, 2022] reports worldwide standalone camera shipments drop by 93% between 2010 and 2021 attributing to the smartphone's ubiquity, smaller size, and innovation in the built-in cameras over standalone cameras, while [DX-MARK, 2021] visualizes the explosive growth of smartphones compared to standalone cameras (Figure 6.2) and

Smartphones have seen an explosive growth.

attributes it to the convenience and ease-of-use that smartphones offer over standalone cameras. Both use data collected from [CIPA (Camera & Imaging Products Association), 2023] as their source.

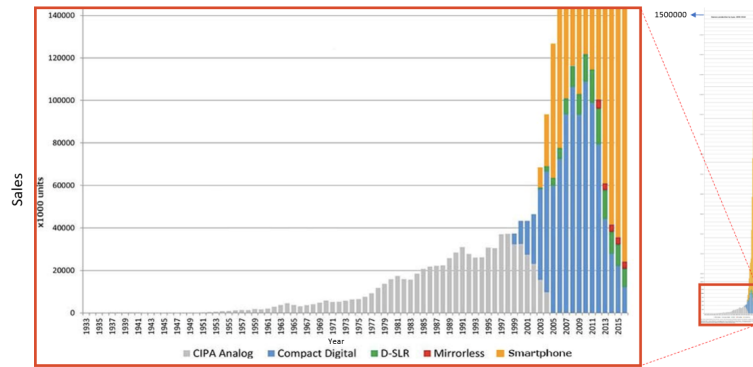


Figure 6.2: Standalone camera and smartphone sales by year, showing the explosive growth of smartphones compared to standalone cameras. By 2015, smartphone sales dwarfed traditional standalone camera sales, and the numbers have only become more extreme since then. [CIPA (Camera & Imaging Products Association), 2023]

We recommend using smartphones with built-in cameras for our prototype.

Accounting for the factors of ubiquity, popularity, affordability, and accessibility, we recommend the use of smartphones with built-in cameras over other standalone camera types for our prototype as we envision a easily reproducible and hassle-free solution.

6.1 Camera Specifications

6.2 Computer/Processing Hardware

Our tool is a locally hosted Flask-based (Flask) web application, meaning the web-application is compiled and run locally on the computer. All the computations happen on the computer. The web application uses Python libraries (Appendix B) for the post-processing of data, computations, and visualizations. The computer vision OpenCV library is

used for image processing (used for detecting ArUco markers). While Python isn't very demanding when it comes to computational power, image processing on the other hand is. The official OpenCV documentation does not mention anything with regard to the minimum hardware requirements for its usage of the computer, however, the main requirements of the OpenCV library are a suitable processing unit and memory. Processing a 640x480 pixel image from a webcam needs little memory while processing a 4K video will fill the RAM (Random Access Memory) more quickly. Hence, the choice is at the user's discretion, based on the needs of his DIY setup and the tasks involved in it.

The basic OpenCV 3 release needs at least 1GB of RAM to compile with an additional 2GB swap memory. A lesser RAM will lead to OpenCV not being able to compile on that system to generate the object code. Hence, a system with 3GB RAM is a minimal requirement ([Laptop: Minimum System Requirement for OpenCV - OpenCV Q&A Forum, 2023]). We recommend using 8GB to avoid crashes and out-of-memory errors. If the user plans to work with high resolution images, then an upgrade to a higher memory is recommended for similar reasons as previously mentioned.

The web application generates data about the detected markers and other computed data throughout the application run, that needs to be stored. Storage devices such as HDDs (Hard Disk Drives) can be used. SSDs (Solid State Drives) can be used for faster read and write speeds improving the performance when working datasets involving large or high resolution images.

6.3 Software Specifications

In this section, we discuss the software specifications that are necessary for the web application to run. Since the web application is Flask-based (refer to chapter 4 for more details) the code is predominantly written in Python. We recommend working with a Python 3.x interpreter (we used Python 3.9), as other versions might result in compilation

The web application is locally hosted, meaning it needs to be compiled and run locally on a computer. Image processing library of OpenCV is used, which is computationally very demanding.

OpenCV needs a RAM of 3GB and more. We recommend using 8GB for smooth operation of the application void of crashes and compile-time errors.

An SSD may be used over an HDD for faster read-write operation speeds, but isn't obligatory.

We recommend using Python versions 3.x

errors due to versioning conflicts (A situation where the python interpreter is unable to compile the code due the incompatibility of code in the package versions as a result of modified or missing code).

Recommended software package versions.

As previously mentioned during the application run in section 5.5, it is important to note that using different package versions, especially the 'opencv-contrib-python' library, might result in the code not functioning due to the possible compatibility issues as a result of the changes done to these packages/libraries. Thus, it is recommended to use the package version combinations as summarized in Appendix B.

iVcam was used to access the smartphone camera from the computer.

The third-party software, IVCam that enables the access of the smartphone camera via the computer is used for the application. Any stable version of IVCam can be used. We used versions x64 v7.2.0.1722 and 7.0.8.1598 for the computer (laptop) and the smartphone respectively. Other alternatives that enable smartphone camera access via the computer can also be used.

6.4 Technical Boundary Analysis

Measure the distance at which the prototype fails to detect the ArUco marker.

To record the limitations/boundaries of our prototype we conducted a technical study of the system which included several iterations of testing and measurements. The goal was to measure the distance at which the prototype fails to detect the ArUco marker affixed to the tool, for different parameter combinations of camera resolutions and marker sizes.

how? Why not just have everything fixed?

ArUco marker was tilted at the same angle as that of the camera from the ground for accurate line of sight measurements.

During the testing and measurements **special care was also taken** to make sure the marker was in line of sight with the camera. Hence, the markers were angled such that their face was directly in line with the camera, and did not lie flat on the ground. Doing this ensures the accuracy of measurements. If the markers lie flat, due to the angle generated between the marker and the camera, the marker gets occluded. Hence, we consider the absolute distances for our measurements.

Let d_a be the absolute distance between the camera and the marker that needs to be calculated, d_m the manually measured distance between the camera and the marker (the distance between the perpendicular projection of the center of the camera on the ground, to the marker), h the height of the camera from the ground, and α be the angle of tilt between the camera and the ground. d_a is calculated using the Pythagoras theorem:

$$d_a^2 = h^2 + d_m^2 \quad (6.1)$$

$$d_a = \sqrt{h^2 + d_m^2}$$

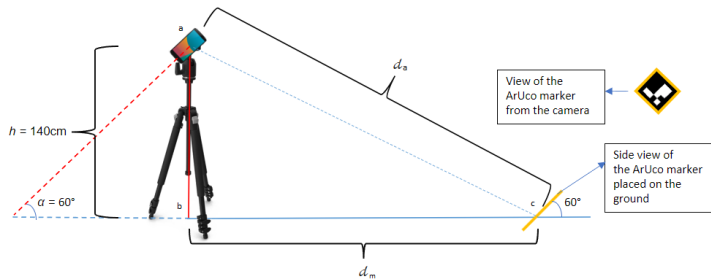


Figure 6.3: Visualization of the placement of the camera and the Aruco markers for calculating the absolute distance d_a between the two.

The height h was 140cm and the angle α was 60° as seen in Figure 6.3. We now can calculate d_a using Equation 6.1.

Two sets of experiments were carried out, one where the resolution of the camera was varied whilst the marker size was fixed/kept constant (Table 6.1), and the other, the vice-versa wherein the size of the marker was varied whilst the resolution of the camera was kept constant (Table 6.2).

Camera Resolution	d_m	$d_a = \sqrt{h^2 + d_m^2}$
640x360	83cm	≈162 cm
640x480	86cm	≈164 cm
960x540	215cm	≈256 cm
800x600	100cm	≈172 cm
1280x720	240cm	≈277 cm
1280x960	120cm	≈184 cm
1920x1080	265cm	≈299 cm
2560x1440	267cm	≈301 cm
3840x2160	267cm	≈301 cm

Table 6.1: The table summarizes d_a and d_m for different camera resolutions for an ArUco marker of size of 40mm x 40mm and the height of the camera from the ground, $h=140$ cm. d_m is the farthest distance between the perpendicular projection of the center of the camera on the ground, to the Aruco marker before it stops being detected. It is obtained by manually placing the marker away from the camera in small distance increments until it finally stops being detected. The value of d_m is the measured distance between the marker and the camera at this point. d_a is the farthest absolute distance between the camera and the ArUco marker before it stops being detected and is obtained by substituting values of h and d_m in Equation 6.1.

Marker Size	d_m	$d_a = \sqrt{h^2 + d_m^2}$
25mm x 25mm	70cm	≈156 cm
30mm x 30mm	105cm	175cm
35mm x 35mm	139cm	≈197 cm
40mm x 40mm	180cm	≈ 228 cm
45mm x 45mm	203cm	≈ 246 cm

Table 6.2: The table summarizes d_a and d_m for different marker sizes with a fixed camera resolution of 1920x1080. The definitions and calculations of d_a and d_m are the same as described in Table 6.1.

Observations

We see from Table 6.1 that in general as the resolution of the camera improves, the marker of the same size gets detected at farther distances, inferring a positive co-relation between the two. Another observation is that after a certain resolution threshold the detection distance gets saturated and doesn't push limits of detection (distance at which the marker gets detected) any further. This occurs as fitting too many pixels in a given screen size after a certain threshold saturates object detection. This is also reported by [Borel-Donohue and Young, 2019] where the author tries to detect objects by adding noise and builds a co-relation graph between the number of objects detected vs the noise added (increased noise results in lower resolutions of the image) This can be seen in Figure 6.4.

That is obvious

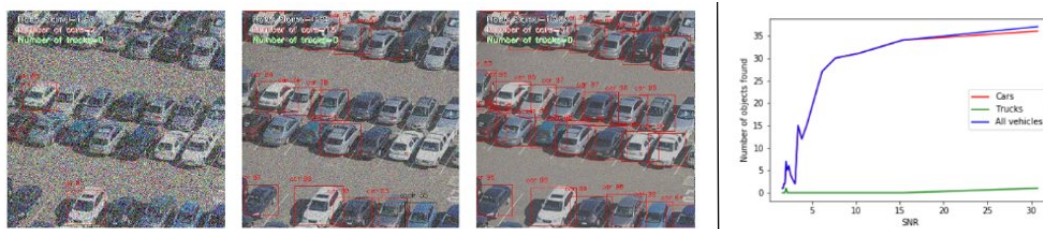


Figure 6.4: *Left:* Vehicle detections for additive noise with signal to noise ratio, SNR=1.81,4.39,10.24, *Right:* Number of cars detected as a function of the Gaussian noise added with a SNR=1.62, ..., 30.76, Borel-Donohue and Young [2019]

From Table 6.2 it can be inferred that as the marker sizes increase the detection gets better. For a given camera resolution a larger marker allows the freedom for it to be placed farther than a smaller one and still be detected by the camera due to it being large and easily recognizable by object detection algorithms.

6.5 System Testing

6.5.1 Purpose

After having established the technical boundaries and limitations in section 6.4, we next focused on the correctness

and accuracy of our system/tool. The goal in this section was to check the correctness of the *tool usage statistics* and its *timeline visualizations*.

6.5.2 Procedure

System testing involved having to do a *complete run-through* which involved performing the tasks of the DIY process. This included using a set of tools to **perform basic tasks** with them in combination with navigating through the web application. The entire procedure was recorded.

What tasks?

Tasks to be performed

The system testing process involved starting at the 'home page' of the web application. The instructions as given by the web interface were followed to proceed further accordingly. At some stage, the web application would prompt to go to the tool usage area to perform a few basic tasks with the tools (Please refer to section 5.5, where an application run is demonstrated, for details and visuals of the web application).

After having finished performing a few basic tasks relevant to the tools provided (example: cutting an object when using a cutter, Figure 6.5), we would proceed back to the web application to view the tool usage statistics and visualizations (subsection 5.5.5). At this stage, we had the option to either stop the application or do another iteration and repeat all the before-mentioned steps. **Once we were satisfied with the iterations, we would quit the application.** This also ends the system testing process. A single testing procedure run is termed as a *trial*.

What is the satisfaction criteria?

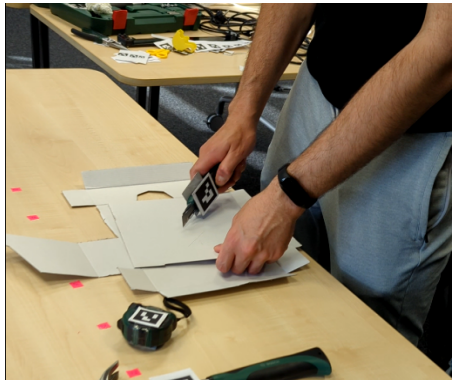


Figure 6.5: A user performing a task using a tool, more specifically using a cutter to cut a piece of paper.

6.5.3 Evaluation and Results

Evaluation Method

We did a total of 8 trials. To verify the correctness of the *tool usage statistics* we need to analyze the *tool usage order* and *duration*. Since the entire system testing process of each trial was recorded (subsection 6.5.2), we can use it to manually analyze and make a note of the order of the tool usage and also estimate the usage duration of each tool. This can be compared to the results displayed by our web application to verify its correctness.

Plotting of the tool usage times to visualize the timeline is done using 'Seaborn'¹, a Python data visualization library based on matplotlib². The visualizations are a direct consequence of the input data and hence is a part of verification process of correctness of the results generated by the tool.

The manually recorded tool usage order details and the data of the usage duration of each tool was entered into an **Excel sheet** and compared with the results generated by our tool. An example of this analysis done for *Trial 3* can be seen in Figure 6.6. This was done for each trial.

Spreadsheet

¹<https://seaborn.pydata.org/>

²<https://matplotlib.org/>

Trial 3				
Manual		Our Tool		
Tool order	Duration	Tool Order	Duration	Delta
Hammer	6	Hammer	6	0
Spanner	3	Wrench	3	0
Plier	4	Plier	4	0
Plier	10	Plier	11	1
Hammer	9	Hammer	10	1
Spanner	11	Wrench	11	0
Hammer	3	Hammer	3	0
Spanner	4	Wrench	3	1

Figure 6.6: A table containing tool order and tool usage duration (both done manually using the recorded videos of the trials during the system testing procedure) of Trial 3 and the same data generated by our tool for comparison.

Results

After manually comparing the data, we found that our tool was successfully able to detect the right tool usage order for all 8 trials without any errors. Delta in the Figure 6.6 indicates how many seconds of deviation was observed between manual estimations and our tool calculations. A summarized report of this comparison for all 8 trials has been visualized in Figure 6.7. Although the focus of our prototype is primarily on finding the *tool usage order* and not the accuracy of the tool usage duration, we see that our tool only had ± 1 -second delta in all the cases except for a ± 2 seconds in one of the cases. In general, the tool has an accuracy of ± 2 seconds, as in cases where it is more than 2, it is specially handled by the algorithm (see section 5.3).

Easy with simple tasks. What about actual usage?

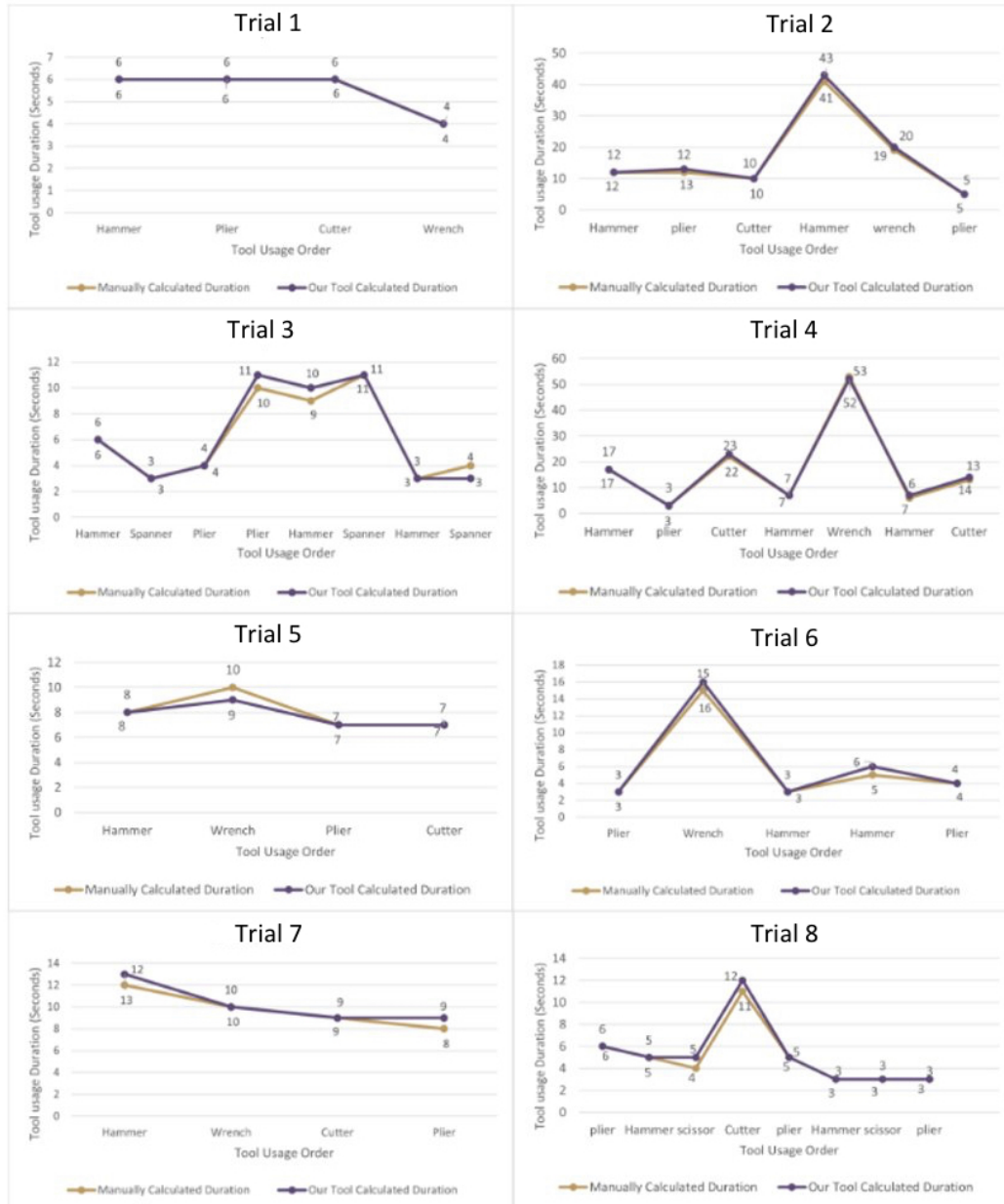


Figure 6.7: Line chart showing the trend between the manual readings of the tool usage duration from the video recordings and the output generated by our tool. It can be seen that the two lines are close to each other for almost all the trials, suggestive of the tool being able to perform well in calculating the tool usage duration.

Chapter 7

Summary, Limitations, and Future Work

7.1 Summary of the work

The thesis aimed at assisting and supporting the DIY creators in their documentation process. The focus was to provide a real-time solution that is affordable and easily reproducible. It must also be ubiquitous so that it applies to most use case scenarios and any special requirements are easily satisfiable and hassle-free without causing hindrance.

Initial attempts using sensor-based (wired and wireless) approach along with its drawbacks.

We, therefore, initially looked at sensor-based solutions in section 3.1 as they are quite affordable and not complicated to work with. The idea was to go for a multi-sensorial approach in hopes of collecting different types of data including audio, vibration, acceleration, tilt, etc. These sensors would be stuck to the tools that would be used in the DIY process. Collection of data from these different sensors during usage of the tool and at rest would help us estimate the tool usage statistics by comparing the two data sets. The methods and sensors used were discussed in subsection subsection 3.1.1 and subsection 3.1.2 along with some of their drawbacks regarding increased overall sensor system size along with the need for wireless data transfer, and the affordability and data handling concerns when too many

sensors are used.

Owing to the drawbacks of the sensor-based system, we decided to use a camera-based system called the VICON Motion Capture. The high-resolution infrared cameras that are strategically placed in order to cover the entire work area, capture the motion of the tools by detecting and tracking the reflective markers placed on the tools. We discussed the use of the VICON system in subsection section 3.2 along with its drawbacks. We, however, had a lot of learning from our experiments with the VICON system. Its **forthcomings** and limitations compelled us to direct our focus on realizing a system that eliminated these drawbacks, eventually leading to our proposed prototype.

In chapters 4 and 5, we discussed the overview of the system components describing in detail the three broad homogeneous components of the entire system. chapter 4 was focused on the physical aspects of the setting-up process majorly involving camera setup and tool tagging, while in chapter 5, we discussed in detail the design and implementation of the software side of the system, the flask-based web application. We explained in detail how the data pertaining to the tool usage is collected, stored, transformed and processed, and finally visualized. Various sub-tasks and processes were also explained in detail for a concrete understanding of the reader. In the final subsection of chapter 5, we did an example run of the web application to demonstrate its usage. Each step was elaborated in detail along with screenshots from the application.

In chapter 6 we discussed and summarized the technical aspects and specifications of the system. We discussed the technical requirements of both the software and the hardware aspects for the smooth and reliable functioning of our prototype. In the hardware requirements, the camera and the computer specification requirements were discussed, while in the software requirements, the versions of the packages and libraries along with the configuration requirements of the web application were discussed. This was followed by a technical boundary analysis in which we measured the distances at which the prototype fails to detect the ArUco marker on the tool, for different param-

After learning from the sensor-based approach and finding its drawbacks we experimented with the Vicon motion-tracking system. The drawbacks and limitations of the Vicon system were discussed.

The overall system architecture and its components were discussed in detail along with the implementation specifics.

Technical aspects, hardware, and software requirements, boundary analysis, and user studies were discussed.

eter combinations of camera resolutions and marker sizes. These measurements were summarized in a table. Finally, we gave some insights into how we conducted our user studies which were very helpful in retrospectively and improving the prototype.

7.2 Related Fields

Construction projects
and manufacturing
facilities.

In this section, we describe how solutions to our problem are not just limited to the DIY processes but also can aid many other fields. In construction projects, a wide range of tools and equipment are used. Tool tracking helps prevent loss or theft, improves accountability, reduces downtime due to missing tools, and aids in managing equipment maintenance schedules. Manufacturing facilities often utilize numerous specialized tools and machinery. Effective tool tracking ensures tools are available when needed, reduces production downtime, optimizes maintenance schedules, and improves overall operational efficiency.

Hospitals and
medical facilities.

Hospitals and medical facilities rely on a range of specialized medical instruments and equipment. Tool tracking helps prevent loss, ensures that instruments are sterilized and properly maintained, and enhances patient safety by minimizing the risk of using defective or improperly calibrated tools.

Distribution centers
and warehouses.

Distribution centers and warehouses use various tools and equipment for loading, unloading, and storage. Tool tracking enhances inventory accuracy, streamlines workflows, and reduces operational costs, thus helping in cost-effective and smooth logistics.

Educational
institutions,
laboratories, and
research centers.

Educational institutions, especially those offering technical or vocational courses, require tools and equipment for hands-on training. Tool tracking helps manage inventory, ensures availability for students, and maintains the condition of teaching aids. Laboratories and research centers rely on precise instruments for experiments and analyses. Tool tracking ensures that equipment is properly calibrated, re-

duces the chances of contamination, and maintains the integrity of research results.

Event planners and organizers rely on various equipment for setting up events. Tool tracking helps streamline the event setup process, ensures equipment is in good condition, and minimizes the risk of missing items during tear-down.

Event planners and organizers.

7.3 Limitations and Future work

In this section, we will discuss the limitations and possible solutions, future work, and ideas to extend our prototype.

Limitation

Our prototype does not perform well for large industrial-scale workspaces. There are two challenges here. Since we are using a single camera, to detect the markers in a very large space we would either need very large ArUco markers or we must have very high-resolution cameras which are expensive. Both are not an ideal choice. Large markers enable easier detection but in large working spaces, we might end up in a situation where the marker necessitates such a size that it is no longer convenient.

Does not perform well for large industrial-scale workspaces due to marker size and camera constraints.

Possible solutions

A wireless sensor-based approach is better suited as the size of the work /tracking area isn't a significant deterring factor for sensor data transmissions. Similar to a multi-sensorial approach a multi-camera approach in which the cameras are strategically placed and tweaked to the exact DIY setup requirements will be effective in accurate and reliable detection and data collection.

Sensor-based approach eliminates the drawbacks mentioned.

Limitation

Our prototype **smartly** tries to determine the tool usage duration by considering the rest states of the tool, meaning we are not tracking the tool at all times during its usage. This might not be favorable in all cases and applications where

Does not track the marker at all times.

accurate tracking of position and duration is required. We also encountered situations during our experiments with the prototype where the marker got covered by the user's hand and consequently went undetected.

Possible solutions

Strategically position marker so that at least one is always visible.

To be able to fix and handle such situations and track the tool at all times, multiple markers can be used for a single tool. They must be strategically placed such that at least one of the markers is always visible to the camera. A possible solution to this problem is suggested in [Wacker et al., 2019] as seen in Figure 7.1.

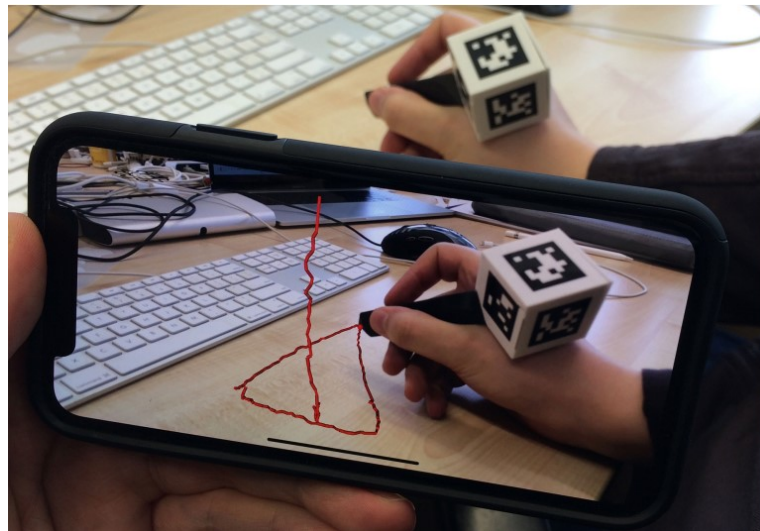


Figure 7.1: Shows a way to strategically place ArUco markers such that at least one of them is always detected.

Limitation

The prototype relies completely on visual perceptions. Other forms of perceptions like heat, sound, and vibrations which give substantial insights on tool usage are not considered.

We are using image detection and processing using OpenCV as the central and only source of data collection. OpenCV is primarily designed for visual tracking based on image analysis. This means it heavily relies on the appearance and characteristics of the objects being tracked. It might struggle with objects that lack distinctive features or are similar in appearance to the background. Hence, our prototype is only limited to visual perceptions. Other sensory factors like heat, sound, vibrations, etc. which are crucial in estimating the tool usage become irrelevant. It

is a known fact that the tracking algorithms might struggle in challenging environments with low lighting conditions, occlusions, cluttered backgrounds, or abrupt changes in object appearance. Hence, our prototype doesn't perform well in very damp and poorly lit working areas.

Possible solutions

A multi-sensorial approach to capture various forms of data. A sensor-based approach that does not rely on and is independent of the lighting might be a better choice.

A multi-sensorial approach.

Limitation

A major drawback of the ArUco markers is that it suffers from the problem of occlusion and noise. If a large part of the marker is occluded, no detection information can be acquired at that moment. Noises due to hand shaking also affect the quality of the resulting detection. This is not desirable in a real-time environment. This also means that the ArUco markers must always be entirely visible to the camera to ensure a high detection rate. A lot of tools in the DIY processes have curved surfaces. ArUco markers cannot be attached to these areas of the tool as the markers will end up being curved resulting in it being obscured. Hence, we are always forced to look for areas/places on the tools where the makers can be placed flat.

ArUco markers suffer from the problem of occlusion and noise

Possible solutions

Various solutions to improve the detection of ArUco markers have been suggested in the literature. [Kam et al., 2018] provides solutions to the problem of occlusion and noise using a linear Karman filter. [Oščádal et al., 2020] suggests ways to develop a library to combine the pose data from the individual markers for higher accuracy and stability. [Sin et al., 2022] focuses on providing a solution specifically targeting the flat nature of the ArUco markers. It proposed a move towards enabling its usage on uneven and curved surfaces for wrapping around objects. [Romero-Ramirez et al., 2018] proposes an approach that claims to outperform the state-of-the-art methods without sacrificing accuracy or robustness and is up to 40 times faster in detection.

Why was this not used if the solution exists?

Using Karman filter to reduce noise, develop a library to combine the pose data from the individual markers for higher accuracy and stability, solutions to enable ArUco marker use on curved surfaces.

The prototype performs poorly in real-time high-speed tracking applications.

Limitation

Our prototype performs poorly in real-time high-speed tracking applications where the objects undergo significant changes in scale, rotation, or perspective. Fast-moving objects or shaky camera movements can introduce motion blur, making it difficult for OpenCV's algorithms to accurately track objects. In general ArUco markers are susceptible to fast movements and often the detection fails.

Usage of high-speed cameras to tackle motion blur.

Possible solutions

While a plethora of research can be found in the literature that addresses and propose solutions trying to improve the detection process itself as mentioned in the previous paragraph, no considerable solutions to the problem of detection of markers moving at high speed exist. Possible solutions to this are using high-speed cameras (these are often very expensive) with higher frame rates to reduce motion blur. Deblurring algorithms targeted to improve out-of-focus and blurred images during post-processing can also be experimented with as a solution. [Romero-Ramirez et al., 2018] suggests one such approach.

Use a slow-motion video as input for detection.

Another approach is to record the DIY process in slow motion (or slowing down the frame rate of a normal video shot at a decently high resolution) and later use the video as the input for detection and post-processing (note that here the detection is not happening in real time).

Using a sensor-based approach.

A contrasting solution is a sensorial approach in which sensors sensitive to motion are used. This offers a very simple straightforward yet highly effective approach to address this challenge.

OpenCV library is computationally intensive.

Limitation

The OpenCV library is computationally intensive, especially when applied to high-resolution videos or real-time applications. This can impact the efficiency and responsiveness of the tracking system, especially on resource-constrained devices. Achieving real-time tracking performance with OpenCV might require optimization and careful parameter tuning. High frame rates and low-latency

tracking can be demanding, especially on slower hardware.

Possible solutions

A solution to this use suitable hardware with enough computation and graphical capabilities based on the scale of the DIY setups.

Using suitable hardware.

Future work and Prototype extension

Machine learning algorithms can be used to create predictive models that take tool usage statistics as input and predict what the DIY process was about. It can further be extended to identify user and tool characteristics.

Another suggestion to extend our current prototype would be to take pictures of the tools at different phases during the DIY process and add them to the documentation template. This adds an additional visual dimension to the documentation.

Conclusion

To conclude, the aim of our tool was to be able to support and assist DIY authorship by providing useful information such as tool usage order and usage timeline. This gives the author a head start and an initial groundwork to move further in his documentation process. Our tool shows that it performs well on the small sized DIY processes not involving very high-speed motions and is able to gather and deliver the intended data and statistics.

Appendix A

Important Code Snippets of the Web Application

NOTE:

Please note that only the core functionalities (functions) have been mentioned here. This is not the entire code base. The HTML, CSS, and Javascript code is excluded.

ArUco detection

```
def tools_to_list(tools):  
    """  
    :param tools: A string containing the tools , given as input by the user .  
    :return: tool_list: A list containing the tools .  
  
    The function converts string of tools to list of tools and returns it .  
    """  
    stripped_tool_list = []  
    tool_list = tools.split(',')  
    for tool in tool_list:  
        tool = tool.strip()  
        stripped_tool_list.append(tool)  
    tool_list = stripped_tool_list  
    print(tool_list)  
    return tool_list  
  
def markers_to_list(markers):
```

```

"""
:param markers: A string containing the markers, given as
input by the user
:return: marker_list: A list containing the markers.

The function converts string of markers to list of markers
and returns it.
"""

```

```

stripped_marker_list = []
marker_list = markers.split(',')
for marker in marker_list:
    marker = marker.strip()
    stripped_marker_list.append(marker)
marker_list = stripped_marker_list
print("The Aruco marker list is: ", marker_list)
return marker_list

```

```

def create_tools_marker_map_dict():

```

```

"""
The function takes config variables: marker_list and
tool_list, to create a dictionary: tool_marker_map_dict
with keys as markers IDs (string) and values as Tool
names (string).
"""

```

```

marker_list = config.marker_list
tool_list = config.tool_list
tool_marker_map_dict = dict(zip(marker_list, tool_list))
print("\nCreating marker and tool dictionary...")
print("The dictionary is: ", tool_marker_map_dict)

```

```

return tool_marker_map_dict

```

```

def aruco_display(corners, ids, image):

```

```

"""
:param corners: The corners detected and returned by
the function: detectMarkers() from the ArUco Library
:param ids: The IDs detected and returned by
the function: detectMarkers() from the ArUco Library
:param image: The image frame returned by the camera
:return: image: with marked center, edges, and text
indicating the tool associated with each ArUco marker

```

```

The function builds edges from the received corners
and then calculates the center. It then returns an
image with marked center, edges, and text indicating

```

the tool associated with each ArUco marker. All of this information along with the x and y coordinates are recorded and stored in an Excel sheet.

```

"""
if len(corners) > 0:
    ids = ids.flatten()

    for (markerCorner, markerID) in zip(corners, ids):
        corners = markerCorner.reshape((4, 2))
        (topLeft, topRight, bottomRight, bottomLeft) = corners

        topRight = (int(topRight[0]), int(topRight[1]))
        bottomRight = (int(bottomRight[0]), int(bottomRight[1]))
        bottomLeft = (int(bottomLeft[0]), int(bottomLeft[1]))
        topLeft = (int(topLeft[0]), int(topLeft[1]))

        cv2.line(image, topLeft, topRight, (0, 255, 0), 2)
        cv2.line(image, topRight, bottomRight, (0, 255, 0), 2)
        cv2.line(image, bottomRight, bottomLeft, (0, 255, 0), 2)
        cv2.line(image, bottomLeft, topLeft, (0, 255, 0), 2)

        centerX = int((topLeft[0] + bottomRight[0]) / 2.0)
        centerY = int((topLeft[1] + bottomRight[1]) / 2.0)
        cv2.circle(image, (centerX, centerY), 4, (0, 0, 255), -1)

        tool_name = config.tool_marker_map_dict.get(str(markerID))

        cv2.putText(image, tool_name, (topLeft[0], topLeft[1]-10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

        print(" Detected \'' + str(tool_name) + '\'' with Aruco
            Marker ID: " + str(markerID) + " | X Co-ordinate: {}"
            .format(centerX) + ", Y Co-ordinate: {}"
            .format(centerY))

        headers = ['TimeStamp', 'ID', 'X-co', 'Y-co']
        rows = [{
            'TimeStamp': datetime.now().replace(microsecond=0),
            'ID': tool_name,
            'X-co': float(centerX),
            'Y-co': float(centerY) }]

with open(config.file_path, "a", encoding="UTF8", newline='') as csv_file:
    writer = csv.DictWriter(csv_file, fieldnames=headers)
    writer = csv.DictWriter(csv_file, fieldnames=headers)
    writer.writerow(rows)

return image

```

```

def detection():
    """
    This is the primary function for the detection of ArUco markers.
    Includes importing the ArUco dictionary, creating detector
    parameters, setting up video capture, and the actual detection.
    """

    print("Starting detection...\n")
    aruco_type = "DICT_4X4_50"

    arucoDict = cv2.aruco.Dictionary_get(ARUCO_DICT[aruco_type])

    arucoParams = cv2.aruco.DetectorParameters_create()

    capture = cv2.VideoCapture(0)

    # Setting frame resolution: 2650 x 1140
    capture.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
    capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
    fps = capture.get(cv2.CAP_PROP_FPS)
    print("Frames per second (fps) of the video: " + str(fps))

    print("Start Capturing...\n")

    while capture.isOpened():
        ret, img = capture.read()
        h, w, _ = img.shape

        corners, ids, rejected = cv2.aruco.detectMarkers(
            img, arucoDict,
            parameters=arucoParams)

        detected_image = aruco_display(corners, ids, img)

        cv2.imshow("Image", detected_image)
        cv2.imshow("Image", img)
        cv2.setWindowTitle("Image", "Aruco Object Detection Video
                               Stream")

        cv2.waitKey(1)

        value = cv2.getWindowProperty("Image", cv2.WND_PROP_VISIBLE)
        if value == 0:
            break

    capture.release()
    cv2.destroyAllWindows()
    print("Aruco Marker detection has Ended.\n")

```

Data Processing and Transformation

```
def split_and_group_data(entire_data):
    """
    :param entire_data: A data frame containing the data read from
        the CSV file generated during the detection process.
    :return: data_list_time_grouped: A list of data frames grouped
        by time, one for each tool.

    The function splits the entire data into separate data frames
    one for each tool. The data frames are then individually
    grouped by the time column and mean of the x and y coordinate
    columns is taken and shortened to two decimal places. These
    data frames are stored in a list and the list is returned.
    """

    if entire_data.empty:
        print("The data sent to split and group was empty.")
        return []

    # Splitting data into data frames for each tool
    data_list = []
    for key, value in config.tool_marker_map_dict.items():
        data_list.append(entire_data[entire_data['ID'] == value])

    # Remove empty data frames for cases when a certain tool is
    # not used for each df in data_list if it is not empty then add
    # it to data_list (syntax is called list comprehension)

    data_list = [df for df in data_list if not df.empty]

    # Grouping the columns by time
    data_list_time_grouped = []
    for data in data_list:
        grouped_data = data.groupby(['TimeStamp', 'ID'],
                                    as_index=False)[['X-co', 'Y-co']].mean()
        .round(2)
        data_list_time_grouped.append(grouped_data)

    combined_data_for_excel = pd.DataFrame()
    empty_row = {'TimeStamp': "", 'ID': "", 'X-co': "", 'Y-co': ""}

    for data in data_list_time_grouped:
        combined_data_for_excel = combined_data_for_excel.
            _append(data, ignore_index=True)
        combined_data_for_excel = combined_data_for_excel._append(
            empty_row, ignore_index=True)
```

```

generated_file_path = config.file_path[:-4] + "_split_group.xlsx"
generated_file_path = generated_file_path.replace("/", "\\")
combined_data_for_excel.to_excel(generated_file_path, index=False)

```

```

return data_list_time_grouped

```

```

def calculate_tool_usage(data_list_time_grouped, flag):

```

```

    """
    :param flag: flag is sent when an override of empty appended
    data is desired
    :param data_list_time_grouped: A list containing data frames
    grouped by time, for each tool.
    :return: data_with_tool_used_calc: A data frame with calculated
    'used' column.

```

In this function we add an entry of 1, if there is a coordinate change between the current and next entry in the data frame (suggesting movement) or 0 if there was no change in the coordinates. The coordinate is calculated as changed if the difference between either the x-coordinates or the y-coordinates or both are greater than 1. The entries of 0s and 1s are made in the 'used' column of the data frame. The data frames of different tools are combined and a new single data frame is returned.

The flag argument comes into play when there is a need to run with an empty data append.

```

    """

```

```

    if len(data_list_time_grouped) == 0:
        print("Data in the calculate usage step is empty")
        return

```

```

    for data in data_list_time_grouped:
        for index in data.index:
            cur_x = float(data['X-co'].iloc[index])
            cur_y = float(data['Y-co'].iloc[index])
            try:
                next_x = float(data['X-co'].iloc[index + 1])
            except IndexError:
                next_x = float(data['X-co'].iloc[index])
            try:
                next_y = float(data['Y-co'].iloc[index + 1])
            except IndexError:
                next_y = float(data['Y-co'].iloc[index])

            if abs(next_x - cur_x) > 1 or abs(next_y - cur_y) > 1:

```

```

        data.loc[index, 'used'] = int(1)
    else:
        data.loc[index, 'used'] = int(0)

data_with_tool_used_calc = pd.DataFrame()

for data in data_list_time_grouped:
    data_with_tool_used_calc = data_with_tool_used_calc.
        _append(data, ignore_index=True)

# appending 4 zeros (meaning the tool is at rest) after each
# tool to prevent errors in duration calculations to take off the
# the time gap between two appends of data to the same file

last_index = len(data) - 1
last_timestamp = data['TimeStamp'].iloc[last_index]
last_timestamp = datetime.strptime(last_timestamp,
    '%Y-%m-%d %H:%M:%S')

counter = 0
while counter < 4:
    timestamp = last_timestamp + dt.timedelta(seconds=(counter+1))
    row_to_add_unused_entry = {'TimeStamp': timestamp,
        'ID': data['ID']
        .iloc[last_index], 'X-co': 0,
        'Y-co': 0, 'used': int(0)}
    data_with_tool_used_calc = data_with_tool_used_calc._append(
        row_to_add_unused_entry,
        ignore_index=True)

    counter += 1

generated_file_path_xlsx = config.file_path[:-4] +
    "_with_tool_usage_stats.xlsx"
generated_file_path_xlsx = generated_file_path_xlsx
    .replace("/", "\\")
generated_file_path_csv = config.file_path[:-4] +
    "_with_tool_usage_stats.csv"

if flag == 0:
    if os.path.exists(generated_file_path_csv):
        prev_data_with_appened_zeros = pd.read_csv
            (generated_file_path_csv,
            sep=',')
        cur_data_with_appened_zeros = prev_data_with_appened_zeros.
            _append(data_with_tool_used_calc,
            ignore_index=True)
        cur_data_with_appened_zeros = cur_data_with_appened_zeros.
            reset_index(drop=True)
        data_with_tool_used_calc = cur_data_with_appened_zeros

```

```

    data_with_tool_used_calc = data_with_tool_used_calc .
                                reset_index(drop=True)
    data_with_tool_used_calc.to_excel(generated_file_path_xlsx ,
                                     index=False)
    data_with_tool_used_calc.to_csv(generated_file_path_csv ,
                                    index=False)

    return data_with_tool_used_calc

def suggest_order(data):
    """
    :param data: The data frame returned from the function:
    calculate_tool_usage(). Data with 'used' column entries.

    :return: tool_usage_duration_filtered: Data frame with
    tool usage duration in order.

    The function calculates the duration for which each tool was
    used. The entries of 1 in the 'used' column are counted to
    calculate the duration. The count indicates the number of
    seconds a particular tool was used. The goal is to filter the
    data and remove the occasional 1's and 0's (errors in the
    readings due to a lot of factors such as fast tool movements
    resulting in the marker going undetected leading to a 0 between
    1's, the camera itself fails to record and detect some frames,
    etc.) that occur in between a series of continuous 0'S and 1's
    respectively.

    A tool is considered to be at rest when a series of more than 3
    continuous zeros are encountered. A series of 4 continuous zeros
    (tool has not moved for 4 seconds) is assumed as the tool is at
    rest/unused. Similarly, a tool is considered as being in use if
    there are 2 or more occurrences of continuous 1's. From basic
    user testing, we found that it takes on average a minimum of 2
    seconds to pick up the tool and place it back down. We remove
    such cases as picking up and immediately putting the tool down
    to rest isn't considered it being used (might be accidental).

    The continuity of 0's and 1's are monitored using 'next_of_cur_used'
    and 'next_of_next_of_cur_used'. The function returns a data frame
    with the duration of usage of each tool along with the sequence of
    the tool.
    """

    if data is None:
        print("Data in the suggest order step is empty")
        return pd.DataFrame() # returning an empty data frame

    counter = 0

```

```

tool_usage_duration = pd.DataFrame(columns=['StartDate', 'EndDate',
                                           'ID', 'Duration'])

for index in data.index:

    # Status of the current tool being used. check if is it a 1 or 0.
    cur_used = int(data['used'].iloc[index])
    cur_tool = data['ID'].iloc[index]

    if index == 0:
        prev_tool = cur_tool
    else:
        prev_tool = data['ID'].iloc[index - 1]

    # Checking and assigning the 'next' and 'next to next' entries
    in 'used' column
    try:
        next_of_cur_used = float(data['used'].iloc[index + 1])
    except IndexError:
        next_of_cur_used = cur_used
    try:
        next_of_next_of_cur_used = float(data['used'].iloc[index+2])
    except IndexError:
        next_of_next_of_cur_used = next_of_cur_used

    # checking all the conditions for 1's
    if cur_used == int(1) and cur_tool == prev_tool:
        counter += 1
    elif cur_used == int(1) and counter > 1 and cur_tool != prev_tool:
        start_time = data['TimeStamp'].iloc[index - counter]
        end_time = data['TimeStamp'].iloc[index - 1]
        duration = end_time.timestamp() -
            start_time.timestamp()

    counter = 0

    # timestamp() function converts time to seconds
    row = {'StartDate': start_time, 'EndDate': end_time,
          'ID': data['ID'].iloc[index - 1], 'Duration': duration}
    tool_usage_duration = tool_usage_duration._append(row,
              ignore_index=True)

    elif cur_used == int(1) and counter <= 1 and cur_tool != prev_tool:
        counter += 1

    # checking all the conditions for 0's
    if cur_used == int(0) and counter > 1 and cur_tool == prev_tool
        and next_of_cur_used == int(0) and
        next_of_next_of_cur_used == int(0):

```

```

start_time = data['TimeStamp'].iloc[index - counter]
end_time = data['TimeStamp'].iloc[index - 1]
duration = end_time.timestamp() - start_time.
            timestamp()

counter = 0

row = {'StartDate': start_time, 'EndDate': end_time,
      'ID': data['ID'].iloc[index - 1], 'Duration': duration}
tool_usage_duration = tool_usage_duration._append(row,
            ignore_index=True)

elif cur_used == int(0) and counter > 1 and
     cur_tool == prev_tool and
     (
         (next_of_cur_used == int(0) and
          next_of_next_of_cur_used != int(0)) or
         (next_of_cur_used == int(0) and
          next_of_next_of_cur_used != int(0)) or
         (next_of_cur_used != int(0) and
          next_of_next_of_cur_used != int(0))
     ):
    counter += 1

elif cur_used == int(0) and counter > 1 and
     cur_tool != prev_tool:

start_time = data['TimeStamp'].iloc[index - counter]
end_time = data['TimeStamp'].iloc[index - 1]
duration = end_time.timestamp() - start_time.timestamp()

counter = 0

row = {'StartDate': start_time, 'EndDate': end_time, 'ID':
      data['ID'].iloc[index - 1], 'Duration': duration}
tool_usage_duration = tool_usage_duration._append(row,
            ignore_index=True)

# Discarding tool usage of less than 2 seconds
tool_usage_duration_filtered = tool_usage_duration
            .query("Duration > 2.0")

if tool_usage_duration_filtered.empty:
    print("The tools were not used for long enough to generate an order")

```

```

ordered_data = tool_usage_duration_filtered
                .sort_values(by=['StartDate'])
ordered_data = ordered_data.reset_index(drop=True)

order_file_path = config.file_path[:-4] + "_tool_order.xlsx"
order_file_path = order_file_path.replace("/", "\\")
ordered_data.to_excel(order_file_path)

print(ordered_data)
return ordered_data

```

```

def create_clean_data(data):
    """
    :param data: A data frame containing data from the function:
        suggest_order().
    :return: clean_time_series_data: A data frame with clean
        continuous time series data.

```

The aim of this function is to create clean continuous uninterrupted, time series data. Due to the fast motion of the tool being used and sometimes the camera failing to record some frames, there are missing values in the timeline.

Example: Hammer was correctly detected between 10:30:20 and 10:30:30, but due to user's fast motions it wasn't detected between 10:30:30 and 10:30:35. This will result in 5 seconds of missing data. This is not a problem in the overall tool usage duration calculation as, when the tool does get detected again, we simply add these missing 5 seconds to the duration (the suggest_order() function takes care of this). However, this data with missing values is discrete and hence not good when we want to visualize a time series graph/plot of the tool usage which is continuous in nature. This function fixes this issue and makes the data continuous by intelligently filling 1s or 0s accordingly.

Steps:

- 1. An empty data frame (clean_data) with start date of the first entry in the data (ordered data) and the end date of the last entry is created. All entries in the 'ID' column are set to null and in the 'used' column are set to 0.*
- 2. The data is split into data frames for each tool and a list of data frames called data_list are created.*
- 3. Create a list of clean_data data frames for each tool called clean_data_list.*

-
4. A mapping is created for each data frame *df* in *data_list* and *clean_df* in *clean_df_list* a mapping is created.
 5. The start date of *df* is searched for in *clean_df*. 1's are filled from that index for a count of the duration. This is done for each row in the *df*.
 6. This is done for each *df* and *clean_df* and a concatenated data frame of all data frames in *clean_df_list* is returned.

```

"""
if data.empty:
    print("Data in the create clean data step is empty")
    return

df_start_date = data['StartDate']
df_end_date = data['EndDate']

min_of_start_date = min(df_start_date)
min_of_end_date = min(df_end_date)
max_of_start_date = max(df_start_date)
max_of_end_date = max(df_end_date)

start_date = min(min_of_start_date, min_of_end_date)
end_date = max(max_of_start_date, max_of_end_date)

index = start_date

# Create an empty data frame with continuous time stamps
in steps of one second.

clean_data = pd.DataFrame(columns=['TimeStamp', 'ID', 'used'])
while index != (end_date + dt.timedelta(seconds=1)):
    row = {'TimeStamp': index, 'ID': '', 'used': int(0)}
    clean_data = clean_data.append(row, ignore_index=True)
    index = index + dt.timedelta(seconds=1)

# Split data into different data frames for each tool
data_list = []
for key, value in config.tool_marker_map_dict.items():
    data_to_append = data[data['ID'] == value]
    data_to_append = data_to_append.reset_index(drop=True)
    data_list.append(data_to_append)

entire_clean_time_series_data = pd.DataFrame()

```

```
for key, value in config.tool_marker_map_dict.items():
    clean_data['ID'] = value
    entire_clean_time_series_data = entire_clean_time_series_data
        .append(clean_data,
                ignore_index=True)

ectsd = entire_clean_time_series_data # for convenience

for df in data_list:
    for index_in_df in df.index:
        tool = str(df['ID'].iloc[index_in_df])
        duration = int(df['Duration'].iloc[index_in_df])
        start_date = df['StartDate'].iloc[index_in_df]
        try:
            start_index_to_fill = ectsd[(ectsd['TimeStamp']
                                        == start_date) &
                                        (ectsd['ID'] == tool)].index[0]

        except IndexError:
            break
        for index_clean_data in ectsd.index:
            ectsd.loc[start_index_to_fill + index_clean_data, 'used'] = 1
            if index_clean_data > duration - 1:
                break

clean_time_series_data_file_path = config.file_path[:-4] +
    "_filtered_clean.xlsx"
clean_time_series_data_file_path = clean_time_series_data_file_path
    .replace("/", "\\")
ectsd.to_excel(clean_time_series_data_file_path, index=False)

return ectsd
```

Appendix B

List of Packages and their Versions

(Table on the next page)

Package	Version
Flask	2.3.2
Jinja2	3.1.2
MarkupSafe	2.1.2
Pillow	9.5.0
PyYAML	6.0
Werkzeug	2.3.4
blinker	1.6.2
bokeh	3.1.1
click	8.1.3
colorama	0.4.6
contourpy	1.0.7
cycler	0.11.0
et-xmlfile	1.1.0
fonttools	4.39.4
future	0.18.3
importlib-metadata	6.6.0
importlib-resources	5.12.0
itsdangerous	2.1.2
kiwisolver	1.4.4
matplotlib	3.7.1
mpld3	0.5.9
numpy	1.24.3
opencv-contrib-python	3.4.11.45
openpyxl	3.1.2
packaging	23.1
pandas	2.0.2
pip	23.1.2
yparsing	3.0.9
python-dateutil	2.8.2
pytz	2023.3
scipy	1.10.1
seaborn	0.12.2
setuptools	67.8.0
six	1.16.0
tornado	6.3.2
tzdata	2023.3
xyzservices	2023.5.0
zipp	3.15.0

Table B.1: List of all the packages and their corresponding versions used in the web application.

Bibliography

Robin S Adams, Jennifer Turns, and Cynthia J Atman. Educating effective engineering designers: The role of reflective practice. *Design studies*, 24(3): 275–294, 2003. URL [https://doi.org/10.1016/S0142-694X\(02\)00056-X](https://doi.org/10.1016/S0142-694X(02)00056-X).

Vassilis Agouridas and Phil Race. Enhancing knowledge management in design education through systematic reflection practice. *Concurrent Engineering*, 15(1):63–76, 2007. URL <https://doi.org/10.1177/1063293X07076267>.

Alibaba. long life ble 3 axis accelerometer beacon e9, a. URL https://www.alibaba.com/product-detail/long-life-ble-3-axis-accelerometer_62028544931.html.

Alibaba. Bluetooth 4.0 itag beacon anti lost alarm key finder ibeacon, b. URL <https://turkish.alibaba.com/product-detail/Bluetooth-4-0-iTag-Beacon-Anti-60794336990.html>.

Cristina H Amon, Susan Finger, Daniel P Siewiorek, and A Smailagic. Integration of design education, research and practice at carnegie mellon university: A multidisciplinary course in wearable computer design. In *Proceedings Frontiers in Education 1995 25th Annual Conference. Engineering Education for the 21st Century*, volume 2, pages 4a1–14. IEEE, 1995. URL <https://ieeexplore.ieee.org/document/483164>.

Arduino Official Store. Arduino nano. URL <https://store.arduino.cc/products/arduino-nano>.

Christoph Borel-Donohue and S Susan Young. Image quality and super resolution effects on object recognition using deep neural networks. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, pages 596–604. SPIE, 2019.

Senthil K Chandrasegaran, Karthik Ramani, Ram D Sriram, Imré Horváth, Alain Bernard, Ramy F Harik, and Wei Gao. The evolution, challenges, and future of knowledge representation in product design systems. *Computer-aided design*, 45(2):204–228, 2013. URL <https://doi.org/10.1016/j.cad.2012.08.006>.

Pei-Yu Chi, Joyce Liu, Jason Linder, Mira Dontcheva, Wilmot Li, and Bjoern Hartmann. Democut: generating concise instructional videos for physical demonstrations. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 141–150, 2013. URL <https://doi.org/10.1145/2501988.2502052>.

CIPA (Camera & Imaging Products Association). Camera & imaging products association: Home, 2023. URL <https://www.cipa.jp/e/index.html>.

Allan Collins and John Seely Brown. The computer as a tool for learning through reflection. In *Learning issues for intelligent tutoring systems*, pages 1–18. Springer, 1988. URL https://doi.org/10.1007/978-1-4684-6350-7_1.

Irio De Feudis, Domenico Buongiorno, Stefano Grossi, Gianluca Losito, Antonio Brunetti, Nicola Longo, Giovanni Di Stefano, and Vitoantonio Bevilacqua. Evaluation of vision-based hand tool tracking methods for quality assessment and training in human-centered industry 4.0. *Applied Sciences*, 12(4):1796, 2022. URL <https://doi.org/10.3390/app12041796>.

DXOMARK. Smartphones vs cameras: Closing the gap on image quality, 2021. URL <https://tinyurl.com/yp7v5j6u>.

Michael D Dzandu and Buddhi Pathak. Diy laboratories, their practices, and challenges—a systematic literature review. *Technology Analysis & Strategic Management*, 33(10):

- 1242–1254, 2021. URL <https://doi.org/10.1080/09537325.2021.19683735>.
- eDesiderata. Statista, 2022. URL <https://edesiderata.crl.edu/resources/statista#crl-review>.
- Jerry Fails and Dan Olsen. A design tool for camera-based interaction. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 449–456, 2003. URL <https://doi.org/10.1145/642611.642690>.
- Eugene S Ferguson. *Engineering and the Mind's Eye*. MIT press, 1994. URL <https://tinyurl.com/5er8n27v>.
- Flask. Welcome to flask. URL <https://flask.palletsprojects.com/en/3.0.x/>.
- Jun Gong, Fraser Anderson, George Fitzmaurice, and Tovi Grossman. Instrumenting and analyzing fabrication activities, users, and expertise. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2019. URL <https://doi.org/10.1145/3290605.3300554>.
- Pauline Gourlet, Sarah Garcin, Louis Eveillard, and Ferdinand Dervieux. Dodoc: A composite interface that supports reflection-in-action. In *Proceedings of the TEI'16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction*, pages 316–323, 2016. URL <https://doi.org/10.1145/2839462.2839506>.
- Jack Hong, George Toye, and Larry J Leifer. Personal electronic notebook with sharing. In *Proceedings 4th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'95)*, pages 88–94. IEEE, 1995. URL <https://ieeexplore.ieee.org/document/484552>.
- Instructables. Iphone scanner, 2013. URL <http://www.instructables.com/id/IPhone-Scanner/>.
- Sam Jacoby and Leah Buechley. Drawing the electric: storytelling with conductive ink. In *Proceedings of the 12th International Conference on Interaction Design and Children*, pages 265–268, 2013. URL <https://doi.org/10.1145/2485760.2485790>.

Ho Chuen Kam, Ying Kin Yu, and Kin Hong Wong. An improvement on aruco marker for pose tracking using kalman filter. In *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 65–69. IEEE, 2018. URL <https://ieeexplore.ieee.org/document/8441049>.

Anna Keune, Kylie Pepler, Stephanie Chang, Lisa Regalla, and Maker Education Initiative. Diy documentation tools for makers, 2015. URL https://makered.org/wpcontent/uploads/2015/02/OPP_ResearchBrief3_DIYDocumentationToolsForMakers_final.pdf.

Stacey Kuznetsov and Eric Paulos. Rise of the expert amateur: Diy projects, communities, and cultures. In *Proceedings of the 6th Nordic conference on human-computer interaction: extending boundaries*, pages 295–304, 2010. URL <https://doi.org/10.1145/1868914.1868950>.

Thomas Landrain, Morgan Meyer, Ariel Martin Perez, and Remi Sussan. Do-it-yourself biology: challenges and promises for an open science and technology movement. *Systems and synthetic biology*, 7:115–126, 2013. URL <https://link.springer.com/article/10.1007/s11693-013-9116-4>.

Laptop: Minimum System Requirement for OpenCV - OpenCV Q&A Forum. Laptop: Minimum system requirement for opencv edit, 2023. URL <https://answers.opencv.org/question/179923/laptop-minimum-system-requirement-for-opencv/>.

Gierad Laput, Yang Zhang, and Chris Harrison. Synthetic sensors: Towards general-purpose sensing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 3986–3999, 2017. URL <https://doi.org/10.1145/3025453.3025773>.

Kimberly Lau, Lora Oehlberg, and Alice Agogino. Sketching in design journals: An analysis of visual representations in the product design process. *The Engineering Design Graphics Journal*, 73(3), 2009. URL <http://www.edgj.org/index.php/EDGJ/issue/view/30>.

- Xiaodong Lin, Cindy Hmelo, Charles K Kinzer, and Teresa J Secules. Designing technology to support reflection. *Educational technology research and Development*, 47:43–62, 1999. URL <https://doi.org/10.1007/BF02299633>.
- Dan Maynes-Aminzade, Terry Winograd, and Takeo Igarashi. Eyepatch: prototyping camera-based interaction through examples. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 33–42, 2007. URL <https://doi.org/10.1145/1294211.1294219>.
- Hamish McAlpine, Ben J Hicks, Greg Huet, and Steve J Culley. An investigation into the use and content of the engineer’s logbook. *Design Studies*, 27(4):481–504, 2006. URL <https://doi.org/10.1016/j.destud.2005.12.001>.
- Hamish McAlpine, Philip Cash, and Ben Hicks. The role of logbooks as mediators of engineering design work. *Design Studies*, 48:1–29, 2017. URL <https://doi.org/10.1016/j.destud.2016.10.003>.
- Morgan Meyer. Domesticating and democratizing science: a geography of do-it-yourself biology. *Journal of Material Culture*, 18(2):117–134, 2013. URL <https://doi.org/10.1177/1359183513483912>.
- Morgan Meyer and Frédéric Vergnaud. The rise of biohacking: Tracing the emergence and evolution of diy biology through online discussions. *Technological Forecasting and Social Change*, 160:120206, 2020. URL <https://doi.org/10.1016/j.techfore.2020.120206>.
- Wilson Ng, Félix Arndt, and Tori Y Huang. Do-it-yourself laboratories as integration-based ecosystems. *Technological Forecasting and Social Change*, 161:120249, 2020. URL <https://doi.org/10.1016/j.techfore.2020.120249>.
- Oberlo. How many people have smartphones? [jul 2023 update], 2023. URL <https://www.oberlo.com/statistics/how-many-people-have-smartphones#:~:>

text=The%20latest%20figures%20show%20an,
2016%2C%20just%20seven%20years%20ago.

Lora Oehlberg, Kimberly Lau, and Alice Agogino. Tangible interactions in a digital age: Medium and graphic visualization in design journals. *AI EDAM*, 23(3): 237–249, 2009. URL <https://doi.org/10.1017/S0890060409000213>.

Online ArUco markers generator. Aruco markers generator! URL <https://chev.me/arucogen/>.

Judit Onsès and Fernando Hernández-Hernández. Visual documentation as space of entanglement to rethink arts-based educational research. *Research in Arts and Education*, 2017(2):61–73, 2017. URL <https://journal.fi/rae/article/view/118837>.

OpenCV. Detection of aruco markers, 2023. URL https://docs.opencv.org/3.4/d5/dae/tutorial_aruco_detection.html.

Petr Oščádal, Dominik Heczko, Aleš Vysocký, Jakub Mlotek, Petr Novák, Ivan Virgala, Marek Sukop, and Zdenko Bobovský. Improved pose estimation of aruco tags using a novel 3d placement strategy. *Sensors*, 20(17):4825, 2020. URL <https://doi.org/10.3390/s20174825>.

Píikea St. Diy ipad stands, 2019. URL <https://www.piikeastreet.com/diy-ipad-stands/>.

Pinterest. Arduino nano 33 iot pinout, specs, schematic[detail board layout], 2021. URL <https://in.pinterest.com/pin/pinterest--802344489872655396/>.

Hayes Raffle, Cati Vaucelle, Ruibing Wang, and Hiroshi Ishii. Jabberstamp: embedding sound and voice in traditional drawings. In *Proceedings of the 6th international conference on Interaction design and children*, pages 137–144, 2007. URL <https://doi.org/10.1145/1297277.1297306>.

Recyclelovers. Diy iphone lego back cover - 5 useful features, 2014. URL [http:](http://)

[//www.recyclelovers.com/2014/01/diy-iphone-lego-back-cover-5-useful.html](http://www.recyclelovers.com/2014/01/diy-iphone-lego-back-cover-5-useful.html).

Clara Rigaud, Yvonne Jansen, and Gilles Bailly. Automating documentation considered harmful (some of the time). In *CHI'22 Workshop: Reimagining Systems for Learning Hands-on Creative and Maker Skills*, 2022. URL <https://hal.sorbonne-universite.fr/hal-03808703/>.

Francisco J Romero-Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and vision Computing*, 76:38–47, 2018. URL <https://doi.org/10.1016/j.imavis.2018.05.004>.

Daniela Rosner and Jonathan Bean. Learning from ikea hacking: i'm not one to decoupage a tabletop and call it a day. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 419–422, 2009. URL <https://doi.org/10.1145/1518701.1518768>.

David Sarpong and Amit Rawal. 23 from open labs to diy labs-harnessing 'the wisdom of crowds' for innovation. *Innovating in the Open Lab: The new potential for interactive value creation across organizational boundaries*, pages 263–274, 2020. URL <https://www.researchgate.net/publication/363767859>.

seaborn.kdeplot - seaborn 0.12.2 documentation. [seaborn.kdeplot](https://seaborn.pydata.org/generated/seaborn.kdeplot.html). URL <https://seaborn.pydata.org/generated/seaborn.kdeplot.html>.

Günter Seyfried, Lei Pei, and Markus Schmidt. European do-it-yourself (diy) biology: Beyond the hope, hype and horror. *Bioessays*, 36(6):548–551, 2014. URL <https://doi.org/10.1002/bies.201300149>.

Zackary PT Sin, Peter HF Ng, and Hong Va Leong. Curvable image markers: Toward trackable markers for every surface. In *International Conference on Advances in Mobile Computing and Multimedia Intelligence*, pages 57–70, 2022. URL https://doi.org/10.1007/978-3-031-20436-4_6.

Heikki Sjöman, Jorgen AB Erichsen, Torgeir Welo, and Martin Steinert. Effortless capture of design output a prerequisite for building a design repository with quantified design output. In *2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 564–570. IEEE, 2017. URL <https://ieeexplore.ieee.org/document/8279935>.

Statista. Infographic: Smartphones wipe out decades of camera industry growth, 2022. URL <https://www.statista.com/chart/15524/worldwide-camera-shipments/>.

Statista Daily Data. Infographic: 4.4 billion camera phones..., 2012. URL <https://tinyurl.com/ykwj7sxf>.

ThinkTAP. 902013. URL <https://tinyurl.com/4ahzstyc>.

S Tichkiewitch and D Brissaud. Introduction to methods and tools for co-operative and integrated design. *Methods and Tools for Co-operative and Integrated design*, 2004. URL <https://doi.org/10.1007/978-94-017-2256-8>.

Cristen Torrey, David W McDonald, Bill N Schilit, and Sara Bly. How-to pages: Informal systems of expertise sharing. In *ECSCW 2007: Proceedings of the 10th European Conference on Computer-Supported Cooperative Work, Limerick, Ireland, 24-28 September 2007*, pages 391–410. Springer, 2007. URL https://doi.org/10.1007/978-1-84800-031-5_21.

Cristen Torrey, Elizabeth F Churchill, and David W McDonald. Learning how: the search for craft knowledge on the internet. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1371–1380, 2009. URL <https://doi.org/10.1145/1518701.1518908>.

Tiffany Tseng. Spin: a photography turntable system for creating animated documentation. In *Proceedings of the 14th International Conference on Interaction Design and Children*, pages 422–425, 2015. URL <https://doi.org/10.1145/2771839.2771869>.

Tiffany Tseng. *Making make-throughs: documentation as stories of design process*. PhD thesis, Massachusetts Institute of Technology, 2016. URL <http://hdl.handle.net/1721.1/106764>.

Tiffany Tseng and Mitchel Resnick. Product versus process: representing and appropriating diy projects online. In *Proceedings of the 2014 conference on Designing interactive systems*, pages 425–428, 2014.

VICON. Documentation. URL <https://docs.vicon.com/m/mobile.action#page/107484037>.

Philipp Wacker, Oliver Nowak, Simon Voelker, and Jan Borchers. Arpen: Mid-air object manipulation techniques for a bimanual ar system with pen & smartphone. In *Proceedings of the 2019 CHI conference on human factors in computing systems*, pages 1–12, 2019. URL <https://doi.org/10.1145/3290605.3300849>.

Ron Wakkary, Markus Lorenz Schilling, Matthew A Dalton, Sabrina Hauser, Audrey Desjardins, Xiao Zhang, and Henry WJ Lin. Tutorial authorship and hybrid designers: The joy (and frustration) of diy tutorials. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 609–618, 2015. URL <https://doi.org/10.1145/2702123.2702550>.

Carolynn J Walthall, Srikanth Devanathan, Lorraine G Kisselburgh, Karthik Ramani, E Daniel Hirleman, and Maria C Yang. Evaluating wikis as a communicative medium for collaboration within colocated and distributed engineering design teams. 2011. URL <https://doi.org/10.1115/1.4004115>.

Wikipedia. Lego, 2023. URL <https://en.wikipedia.org/wiki/Lego>.

Maria C Yang. Observations on concept generation and sketching in engineering design. *Research in Engineering Design*, 20:1–11, 2009. URL <https://doi.org/10.1007/s00163-008-0055-0>.

