

Gesture Navigation System  
Micro Student Community KiiT Chaper  
POC: Kartik Agarwal & Animesh Pathak

**Software Requirements Specification**  
**Document**

**Version: (1.0)**

**Date: (09/06/2021)**

## Table of Contents

<b>1. Introduction</b>	<b>4</b>
1.1 Purpose	4
1.2 Scope	4
1.3 Definitions, Acronyms, Abbreviations, and References	<i>Error! Bookmark not defined.</i>
1.4 Overview	4
<b>2. The Overall Description</b>	<b>4</b>
2.1 Product Perspective	5
2.1.1 System Interfaces	5
2.1.2 Interfaces	5
2.1.3 Hardware Interfaces	6
2.1.4 Software Interfaces	6
2.1.5 Communications Interfaces	6
2.1.6 Memory Constraints	7
2.1.7 Operations	7
2.1.8 Site Adaptation Requirements	7
2.2 Product Functions	7
2.3 User Characteristics	8
2.4 Constraints	8
2.5 Assumptions and Dependencies	9
2.6 Apportioning of Requirements	9
<b>3. Specific Requirements</b>	<b>9</b>
3.1 External interfaces	10
3.2 Functions	11
3.3 Performance Requirements	11
3.4 Logical Database Requirements	12
3.5 Design Constraints	12
3.5.1 Standards Compliance	12
3.6 Software System Attributes	13
3.6.1 Reliability	13
3.6.2 Availability	13
3.6.3 Security	13
3.6.4 Maintainability	14
3.6.5 Portability	14
3.7 Organizing the Specific Requirements	15
3.7.1 System Mode	15
3.7.2 User Class	15

## Software Requirements Specifications Document

3.7.3 Objects	15
3.7.4 Feature	15
3.7.5 Stimulus	16
3.7.6 Response	16
3.7.7 Functional Hierarchy	16
3.8 <i>Additional Comments</i>	16
<b>4. Change Management Process</b>	
<b>5. Document Approvals</b>	
<b>6. Supporting Information</b>	16

## 1. Introduction

*The basic idea is to create a language-independent library that can enable programmers to add gesture control to their web or android applications just by command line arguments or using a database. Let's say, we generally use google jam board but if someone is missing a stylus then he/she could have written on it just by gestures, as soon as Jamboard is integrated with our application. This project will make navigation a hell easier, imagine being able to surf the web without even touching the mouse.*

### **1.1 Purpose**

*This SRS document is prepared for MSC-KiiT's Project Wing, to give developers a basic idea about the project, its features, its requirements, etc.*

### **1.2 Scope**

*The Gesture Navigation System uses Mediapipe for Hand Tracking in Python. The main goal of this project is to simplify Gesture Navigation for Developers. Developers can add Gesture navigation to their webpage just by Cloning our project and connecting it to database like (MySQL, Microsoft Excel) OR even use command line arguments. We will enable few default gestures whose function can be decided by developer himself, moreover in a further version we will enable user to add a gesture manually.*

### **1.3 Definitions, Acronyms, Abbreviations, and References**

Google Sheets API : <https://developers.google.com/sheets/api/quickstart/python>  
MediaPipe : <https://google.github.io/mediapipe/solutions/hands.html>  
OpenCV : [https://docs.opencv.org/4.5.2/d6/d00/tutorial\\_py\\_root.html](https://docs.opencv.org/4.5.2/d6/d00/tutorial_py_root.html)  
MySQL : <https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html>

### **1.4 Overview**

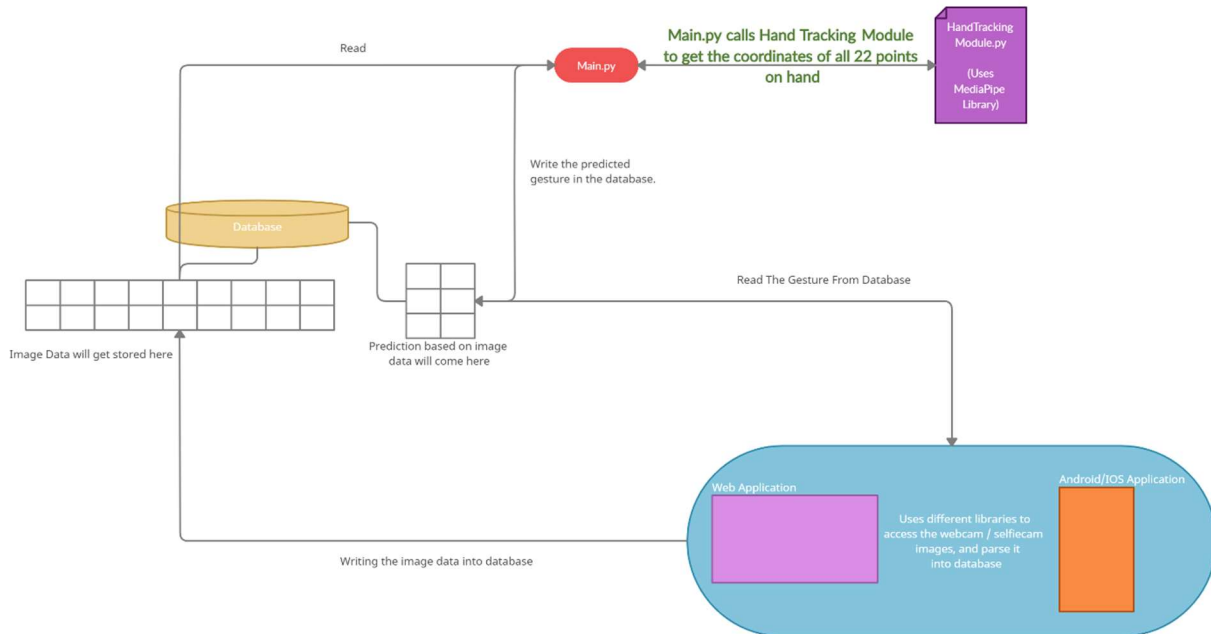
*Moving forward to Section 2, we will be discussing about Software and Hardware stack required for the project. Interface we will be providing Developer/User, and we will also discuss about features and limitations of this project.*

*Then moving to Section 3 we will be discussing about the database and logical requirements. Moreover we will be discussing about the feature and Hierarchy in the project and an expected timeline for the project. We will be discussing about the performance issues that are needed to be addressed.*

## 2. The Overall Description

*Gesture Navigation System uses Mediapipe for hand tracking. Users of the library Just need to clone the repository from the MSC-KiiT Chapter's Official GitHub, and setup the database by following the steps mentioned in README.md file. Now User can use predefines gestures in the library, just by passing webcam images in to database and fetching the gestures decriction from database which is predicted by main.py .*

## **2.1 Product Perspective**



### **2.1.1 System Interfaces**

*As shown in above figure, Dveloper/ User need to connect with the SQL server. The server setup can either be done by user himself or can use the default one available online, but using online one misght slowdown the process a little bit. The language used by the Developer/User should be able to intract with database and and should be able to access astore and process camera images. As discussed earlier in this project we have used mediapipe for hand tracking, opencv for image formatting, so user need to install all the dependencies to the project pesent in REQUIREMENTS.txt.*

### **2.1.2 Interfaces**

*Approaches:*

- (1) First way for Developer to do Gesture navigation is via Database, and it will be recommended one.*
- (2) As of an second approach user can use command line arguments to do Gesture Navigation.*

*Library will interact with user via database or command line, which will consist of two tables one for image data and second for prediction for the image data respectively.*

### **2.1.3 Hardware Interfaces**

*As of hardware interface is concerned, we will be requiring just a webcam access or mobile selfie cam access.*

**\*\*To Be Added later on after Project Completion\*\***

### **2.1.4 Software Interfaces**

*Specify the use of other required software products and interfaces with other application systems. For each required software product, include:*

- (1) Name*
- (2) Mnemonic*
- (3) Specification number*
- (4) Version number*
- (5) Source*

*For each interface, provide:*

- (1) Discussion of the purpose of the interfacing software as related to this software product*
- (2) Definition of the interface in terms of message content and format*

*Here we document the APIs, versions of software that we do not have to write, but that our system has to use. For instance if your customer uses SQL Server 7 and you are required to use that, then you need to specify i.e.*

*2.1.4.1 Microsoft SQL Server 7. The system must use SQL Server as its database component. Communication with the DB is through ODBC connections. The system must provide SQL data table definitions to be provided to the company DBA for setup.*

*A key point to remember is that you do NOT want to specify software here that you think would be good to use. This is only for **customer-specified systems** that you **have to** interact with. Choosing SQL Server 7 as a DB without a customer requirement is a Design choice, not a requirement. This is a subtle but important point to writing good requirements and not over-constraining the design.*

### **2.1.5 Communications Interfaces**

*Specify the various interfaces to communications such as local network protocols, etc. These are protocols you will need to directly interact with. If you happen to use web services transparently to your application then do not list it here. If you are using a custom protocol to communicate between systems, then document that protocol here so designers know what to design. If it is a standard protocol, you can reference an existing document or RFC.*

### **2.1.6 Memory Constraints**

*Specify any applicable characteristics and limits on primary and secondary memory. Don't just make up something here. If all the customer's machines have only 128K of RAM, then your target design has got to come in under 128K so there is an actual requirement. You could also cite market research here for shrink-wrap type applications "Focus groups have determined that our target market has between 256-512M of RAM, therefore the design footprint should not exceed 256M." If there are no memory constraints, so state.*

### **2.1.7 Operations**

*Specify the normal and special operations required by the user such as:*

- (1) The various modes of operations in the user organization*
- (2) Periods of interactive operations and periods of unattended operations*
- (3) Data processing support functions*
- (4) Backup and recovery operations*

*(Note: This is sometimes specified as part of the User Interfaces section.) If you separate this from the UI stuff earlier, then cover business process type stuff that would impact the design. For instance, if the company brings all their systems down at midnight for data backup that might impact the design. These are all the work tasks that impact the design of an application, but which might not be located in software.*

### **2.1.8 Site Adaptation Requirements**

*In this section:*

- (1) Define the requirements for any data or initialization sequences that are specific to a given site, mission, or operational mode*
- (2) Specify the site or mission-related features that should be modified to adapt the software to a particular installation*

*If any modifications to the customer's work area would be required by your system, then document that here. For instance, "A 100Kw backup generator and 10000 BTU air conditioning system must be installed at the user site prior to software installation". This could also be software-specific like, "New data tables created for this system must be installed on the company's existing DB server and populated prior to system activation." Any equipment the customer would need to buy or any software setup that needs to be done so that your system will install and operate correctly should be documented here.*

## **2.2 Product Functions**

*Provide a summary of the major functions that the software will perform. Sometimes the function summary that is necessary for this part can be taken directly from the section of the higher-level specification (if one exists) that allocates particular functions to the software product.*

*For clarity:*

- (1) The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time.*
- (2) Textual or graphic methods can be used to show the different functions and their relationships. Such a diagram is not intended to show a design of a product but simply shows the logical relationships among variables.*

*AH, Finally the real meat of section 2. This describes the functionality of the system in the language of the customer. What specifically does the system that will be designed have to do? Drawings are good, but remember this is a description of what the system needs to do, not how you are going to build it. (That comes in the design document).*

### **2.3 User Characteristics**

*Describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. Do not state specific requirements but rather provide the reasons why certain specific requirements are later specified in section 3.*

*What is it about your potential user base that will impact the design? Their experience and comfort with technology will drive UI design. Other characteristics might actually influence internal design of the system.*

### **2.4 Constraints**

*Provide a general description of any other items that will limit the developer's options. These can include:*

- (1) Regulatory policies*
- (2) Hardware limitations (for example, signal timing requirements)*
- (3) Interface to other applications*
- (4) Parallel operation*
- (5) Audit functions*
- (6) Control functions*
- (7) Higher-order language requirements*
- (8) Signal handshake protocols (for example, XON-XOFF, ACK-NACK)*
- (9) Reliability requirements*
- (10) Criticality of the application*
- (11) Safety and security considerations*



*This section captures non-functional requirements in the customers language. A more formal presentation of these will occur in section 3.*

## **2.5 Assumptions and Dependencies**

*List each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption might be that a specific operating system would be available on the hardware designated for the software product. If, in fact, the operating system were not available, the SRS would then have to change accordingly.*

*This section is catch-all for everything else that might influence the design of the system and that did not fit in any of the categories above.*

## **2.6 Apportioning of Requirements.**

*Identify requirements that may be delayed until future versions of the system. After you look at the project plan and hours available, you may realize that you just cannot get everything done. This section divides the requirements into different sections for development and delivery. Remember to check with the customer – they should prioritize the requirements and decide what does and does not get done. This can also be useful if you are using an iterative life cycle model to specify which requirements will map to which iteration.*

## **3. Specific Requirements**

*This section contains all the software requirements at a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system and all functions performed by the system in response to an input or in support of an output. The following principles apply:*

*(1) Specific requirements should be stated with all the characteristics of a good SRS*

- *correct*
- *unambiguous*
- *complete*
- *consistent*
- *ranked for importance and/or stability*
- *verifiable*
- *modifiable*
- *traceable*

*(2) Specific requirements should be cross-referenced to earlier documents that relate*

(3) All requirements should be uniquely identifiable (usually via numbering like 3.1.2.3)

(4) Careful attention should be given to organizing the requirements to maximize readability (Several alternative organizations are given at end of document)

Before examining specific ways of organizing the requirements it is helpful to understand the various items that comprise requirements as described in the following subclasses. This section reiterates section 2, but is for developers not the customer. The customer buys in with section 2, the designers use section 3 to design and build the actual application.

Remember this is not design. Do not require specific software packages, etc unless the customer specifically requires them. Avoid over-constraining your design. Use proper terminology:

The system shall... A required, must have feature

The system should... A desired feature, but may be deferred til later

The system may... An optional, nice-to-have feature that may never make it to implementation.

Each requirement should be uniquely identified for traceability. Usually, they are numbered 3.1, 3.1.1, 3.1.2.1 etc. Each requirement should also be testable. Avoid imprecise statements like, "The system shall be easy to use" Well no kidding, what does that mean? Avoid "motherhood and apple pie" type statements, "The system shall be developed using good software engineering practice"

Avoid examples, This is a specification, a designer should be able to read this spec and build the system without bothering the customer again. Don't say things like, "The system shall accept configuration information such as name and address." The designer doesn't know if that is the only two data elements or if there are 200. List every piece of information that is required so the designers can build the right UI and data tables.

### **3.1 External Interfaces**

This contains a detailed description of all inputs into and outputs from the software system. It complements the interface descriptions in section 2 but does not repeat information there. Remember section 2 presents information oriented to the customer/user while section 3 is oriented to the developer.

It contains both content and format as follows:

- Name of item
- Description of purpose
- Source of input or destination of output
- Valid range, accuracy and/or tolerance
- Units of measure

- *Timing*
- *Relationships to other inputs/outputs*
- *Screen formats/organization*
- *Window formats/organization*
- *Data formats*
- *Command formats*
- *End messages*

### **3.2 Functions**

*Functional requirements define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as “shall” statements starting with "The system shall..."*

*These include:*

- *Validity checks on the inputs*
- *Exact sequence of operations*
- *Responses to abnormal situation, including*
  - *Overflow*
  - *Communication facilities*
  - *Error handling and recovery*
- *Effect of parameters*
- *Relationship of outputs to inputs, including*
  - *Input/Output sequences*
  - *Formulas for input to output conversion*

*It may be appropriate to partition the functional requirements into sub-functions or sub-processes. This does not imply that the software design will also be partitioned that way.*

### **3.3 Performance Requirements**

*This subsection specifies both the static and the dynamic numerical requirements placed on the software or on human interaction with the software, as a whole. Static numerical requirements may include:*

- The number of terminals to be supported*
- The number of simultaneous users to be supported*
- Amount and type of information to be handled*

*Static numerical requirements are sometimes identified under a separate section entitled capacity.*

*Dynamic numerical requirements may include, for example, the numbers of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions.*

*All of these requirements should be stated in measurable terms.*

*For example,*

*95% of the transactions shall be processed in less than 1 second*

*rather than,*

*An operator shall not have to wait for the transaction to complete.*

*(Note: Numerical limits applied to one specific function are normally specified as part of the processing subparagraph description of that function.)*

### **3.4 Logical Database Requirements**

*This section specifies the logical requirements for any information that is to be placed into a database. This may include:*

- *Types of information used by various functions*
- *Frequency of use*
- *Accessing capabilities*
- *Data entities and their relationships*
- *Integrity constraints*
- *Data retention requirements*

*If the customer provided you with data models, those can be presented here. ER diagrams (or static class diagrams) can be useful here to show complex data relationships. Remember a diagram is worth a thousand words of confusing text.*

### **3.5 Design Constraints**

*Specify design constraints that can be imposed by other standards, hardware limitations, etc.*

#### **3.5.1 Standards Compliance**

*Specify the requirements derived from existing standards or regulations. They might include:*

*(1) Report format*

- (2) Data naming
- (3) Accounting procedures
- (4) Audit Tracing

For example, this could specify the requirement for software to trace processing activity. Such traces are needed for some applications to meet minimum regulatory or financial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace file with before and after values.

### **3.6 Software System Attributes**

There are a number of attributes of software that can serve as requirements. It is important that required attributes be specified so that their achievement can be objectively verified. The following items provide a partial list of examples. These are also known as non-functional requirements or quality attributes.

These are characteristics the system must possess, but that pervade (or cross-cut) the design. These requirements have to be testable just like the functional requirements. Its easy to start philosophizing here, but keep it specific.

#### **3.6.1 Reliability**

Specify the factors required to establish the required reliability of the software system at time of delivery. If you have MTBF requirements, express them here. This doesn't refer to just having a program that does not crash. This has a specific engineering meaning.

#### **3.6.2 Availability**

Specify the factors required to guarantee a defined availability level for the entire system such as checkpoint, recovery, and restart. This is somewhat related to reliability. Some systems run only infrequently on-demand (like MS Word). Some systems have to run 24/7 (like an e-commerce web site). The required availability will greatly impact the design. What are the requirements for system recovery from a failure? "The system shall allow users to restart the application after failure with the loss of at most 12 characters of input".

#### **3.6.3 Security**

Specify the factors that would protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to:

- Utilize certain cryptographic techniques
- Keep specific log or history data sets

- Assign certain functions to different modules
- Restrict communications between some areas of the program
- Check data integrity for critical variables

### **3.6.4 Maintainability**

Specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices. If someone else will maintain the system

### **3.6.5 Portability**

Specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include:

- Percentage of components with host-dependent code
- Percentage of code that is host dependent
- Use of a proven portable language
- Use of a particular compiler or language subset
- Use of a particular operating system

Once the relevant characteristics are selected, a subsection should be written for each, explaining the rationale for including this characteristic and how it will be tested and measured. A chart like this might be used to identify the key characteristics (rating them High or Medium), then identifying which are preferred when trading off design or implementation decisions (with the ID of the preferred one indicated in the chart to the right). The chart below is optional (it can be confusing) and is for demonstrating tradeoff analysis between different non-functional requirements. H/M/L is the relative priority of that non-functional requirement.

ID	Characteristic	H/M/L	1	2	3	4	5	6	7	8	9	10	11	12
1	Correctness													
2	Efficiency													
3	Flexibility													
4	Integrity/Security													
5	Interoperability													
6	Maintainability													
7	Portability													
8	Reliability													
9	Reusability													
10	Testability													
11	Usability													
12	Availability													

*Definitions of the quality characteristics not defined in the paragraphs above follow.*

- *Correctness - extent to which program satisfies specifications, fulfills user's mission objectives*
- *Efficiency - amount of computing resources and code required to perform function*
- *Flexibility - effort needed to modify operational program*
- *Interoperability - effort needed to couple one system with another*
- *Reliability - extent to which program performs with required precision*
- *Reusability - extent to which it can be reused in another application*
- *Testability - effort needed to test to ensure performs as intended*
- *Usability - effort required to learn, operate, prepare input, and interpret output*

*THE FOLLOWING (3.7) is not really a section, it is talking about how to organize requirements you write in section 3.2. At the end of this template there are a bunch of alternative organizations for section 3.2. Choose the ONE best for the system you are writing the requirements for.*

### **3.7 Organizing the Specific Requirements**

*For anything but trivial systems the detailed requirements tend to be extensive. For this reason, it is recommended that careful consideration be given to organizing these in a manner optimal for understanding. There is no one optimal organization for all systems. Different classes of systems lend themselves to different organizations of requirements in section 3. Some of these organizations are described in the following subclasses.*

#### **3.7.1 System Mode**

*Some systems behave quite differently depending on the mode of operation. When organizing by mode there are two possible outlines. The choice depends on whether interfaces and performance are dependent on mode.*

#### **3.7.2 User Class**

*Some systems provide different sets of functions to different classes of users.*

#### **3.7.3 Objects**

*Objects are real-world entities that have a counterpart within the system. Associated with each object is a set of attributes and functions. These functions are also called services, methods, or processes. Note that sets of objects may share attributes and services. These are grouped together as classes.*

#### **3.7.4 Feature**

*A feature is an externally desired service by the system that may require a sequence of inputs to effect the desired result. Each feature is generally described in as sequence eof stimulus-response pairs.*

### **3.7.5 Stimulus**

*Some systems can be best organized by describing their functions in terms of stimuli.*

### **3. 7.6 Response**

*Some systems can be best organized by describing their functions in support of the generation of a response.*

### **3.7.7 Functional Hierarchy**

*When none of he above organizational schemes prove helpful, the overall functionality can be organized into a hierarchy of functions organized by either common inputs, common outputs, or common internal data access. Data flow diagrams and data dictionaries can be use dot show the relationships between and among the functions and data.*

## **3.8 Additional Comments**

*Whenever a new SRS is contemplated, more than one of the organizational techniques given in 3.7 may be appropriate. In such cases, organize the specific requirements for multiple hierarchies tailored to the specific needs of the system under specification.*

*Three are many notations, methods, and automated support tools available to aid in the documentation of requirements. For the most part, their usefulness is a function of organization. For example, when organizing by mode, finite state machines or state charts may prove helpful; when organizing by object, object-oriented analysis may prove helpful; when organizing by feature, stimulus-response sequences may prove helpful; when organizing by functional hierarchy, data flow diagrams and data dictionaries may prove helpful.*

*In any of the outlines below, those sections called “Functional Requirement i” may be described in native language, in pseudocode, in a system definition language, or in four subsections titled: Introduction, Inputs, Processing, Outputs.*

## **4. Change Management Process**



*Identify the change management process to be used to identify, log, evaluate, and update the SRS to reflect changes in project scope and requirements. How are you going to control changes to the requirements. Can the customer just call up and ask for something new? Does your team have to reach consensus? How do changes to requirements get submitted to the team? Formally in writing, email or phone call?*

## **5. Document Approvals**

*Identify the approvers of the SRS document. Approver name, signature, and date should be used.*

## **6. Supporting Information**

*The supporting information makes the SRS easier to use. It includes:*

- *Table of Contents*
- *Index*
- *Appendices*

*The Appendices are not always considered part of the actual requirements specification and are not always necessary. They may include:*

- (a) *Sample I/O formats, descriptions of cost analysis studies, results of user surveys*
- (b) *Supporting or background information that can help the readers of the SRS*
- (c) *A description of the problems to be solved by the software*
- (d) *Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements*

*When Appendices are included, the SRS should explicitly state whether or not the Appendices are to be considered part of the requirements.*

*Tables on the following pages provide alternate ways to structure section 3 on the specific requirements. You should pick the best one of these to organize section 3 requirements.*