Virtual Computer Mouse using mmWave Radar (TI)

Oscar Chavez Araiza
Greyson Heath
Daniel Lu
Zane Meikle

Final Report

Table of Contents

Concept of Operations..... 3
Functional System Requirements..... 17
Interface Control Document.....34
Schedule and Validation..... 46
Subsystem Reports.....55
System Report.....80

Virtual Computer Mouse using mmWave Radar (TI)

Oscar Chavez Araiza
Greyson Heath
Daniel Lu
Zane Meikle

CONCEPT OF OPERATIONS

REVISION – 1.2
28 April 2025

Concept of Operations
for
Virtual Computer Mouse using mmWave Radar (TI)

Team <71>

Approved by:

_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
-	9/14/2024	Zane Meikle		Draft Release
1.0	9/22/2024	Zane Meikle		Revision 1.0
1.1	12/3/2024	Greyson Heath		Revision 1.1
1.2	4/28/2025	Zane Meikle		Revision 1.2

Table of Contents

Table of Contents **6**
List of Tables **7**
List of Figures **8**
1. Executive Summary **9**
2. Introduction **10**
2.1. Background....10
2.2. Overview....10
2.3. Referenced Documents and Standards....10
3. Operating Concept **11**
3.1. Scope....11
3.2. Operational Description and Constraints....11
3.3. System Description....12
3.4. Modes of Operations....13
3.5. Users....14
3.6. Support....14
4. Scenario(s) **15**
4.1. Presentation....15
4.2. Medical....15
5. Analysis **16**
5.1. Summary of Proposed Improvements....16
5.2. Disadvantages and Limitations....16
5.3. Alternatives....16
5.4. Impact....16

List of Tables

No table of figures entries found.

List of Figures

Figure 1: Functional System Diagram..... 10
Figure 2: Prediction state and measurement state..... 11

1. Executive Summary

This project aims to create a virtual mouse using Texas Instrument's (TI) mmWave radar. Using the mmWave radar, the system will track the user's hand and convert those movements into mouse movements on the computer. Additionally, it will recognize different gestures and convert those into mouse actions. The virtual mouse will have a GUI in which you can change multiple settings. This solution stands out based on its lower power and processing requirements, and a variety of environments in which it can easily operate.

2. Introduction

This project aims to make effective use of TI's mmWave radar technology to eliminate the need for a mouse by tracking the user's hand movements. A virtual mouse would improve a multitude of computing tasks, as opposed to using a mouse, in ways such as the following: enhanced multitasking, improved accessibility leading to a 'hands-free' interaction, increased ergonomics, and hygiene management. The performance of the virtual mouse will not be impacted by variations in ambient brightness or pollutant levels, as the signals can transmit usable data regardless of the environmental conditions.

2.1. Background

Common virtual mouse systems used today rely on technology referred to as computer vision. Computer vision uses machine learning and neural networks to extract important data from video or digital images, and further analyze and compute tasks. A crucial component of the success of these systems involves the storage of countless videos and images. The system will continuously run analyses and train itself using the stored data, to the point where it can distinguish minute differences and learn to recognize patterns. Massive databases store heavy amounts of data used to execute computer vision, which requires vast resources and a significant amount of computing power.

The virtual mouse using TI's mmWave radar technology aims to replicate the same successful results seen using computer vision, but meanwhile address and correct various technical and privacy issues. The mmWave radar does not rely on the following for meaningful data: visual data extracted from videos or photos, sufficient ambient lighting conditions, and high air transparency. The mmWave radar will consume much less power as compared to a traditional camera, and enable an even lower-latency detection and tracking of gestures. Eliminating the need for a camera resolves the growing privacy concern regarding potentially sensitive user data.

2.2. Overview

The mmWave radars use short wavelength electromagnetic waves, detecting the gestures in this project. The data from the radars will be communicated as frames. A gesture-detecting Machine Learning model will process the data to detect hand gestures. Detected gestures corresponding to assigned commands will have a gesture ID which will later be used to map the gesture to the mouse input. A separate algorithm will track the hand and map its movements to mouse movements. The mmWave radar can determine the range, velocity, and angle of the objects, with multiple electromagnetic waves transmitting and reflecting signals. The mmWave radar operates at 60-64 GHz with error in a fraction of a millimeter. The mmWave radar will provide position based data when an object, presumably a hand, enters its tracking area. We will use that data to feed it to the Machine Learning model and the hand tracking subsystem.

2.3. Referenced Documents and Standards

- 60GHz mmWave Sensor EVMs User's Guide
- IWR6843, IWR6443 Single-Chip 60- to 64-GHz mmWave Sensor datasheet
- Texas Instruments mmWave Radar Academy
- IEEE Xplore: An FCNN-Based Super-Resolution Mmwave Radar Framework for Contactless Musical Instrument Interface

3. Operating Concept

3.1. Scope

The virtual mouse using mmWave Radar is a proof of concept that will simulate the functionality of a simple computer mouse. A virtual mouse has been implemented using computer vision, thus this project will test the effectiveness of radar in tasks already done using computer vision. The virtual mouse will be able to control the cursor's position, and mouse actions like left-click, right-click, double left-click, and scroll.

3.2. Operational Description and Constraints

3.2.1 Operational Description

To operate the virtual mouse, the user will connect TI's mmWave radar to the top of the computer via microUSB to USB. The mmWave radar will be recognized as an I/O device and will be continuously monitoring for gestures to interact with the computer mouse. The user should ensure that the mmWave radar is properly oriented so that it is able to detect and capture hand gestures. Additionally, the computer program that interacts and interprets the radar output should be executed and running prior to using the mmWave radar.

3.2.2 Constraints

Correct mouse functionality is expected when the user's hands are within a range of ~1 ½ feet from the mmWave radar, and only one hand is facing the radar.

3.3. System Description

The mmWave Radar Virtual Mouse is divided into 4 different subsystems: Chirp Optimization and Data Collection, Mouse Positioning, Gesture Recognition, and Input Mapping to the Computer.

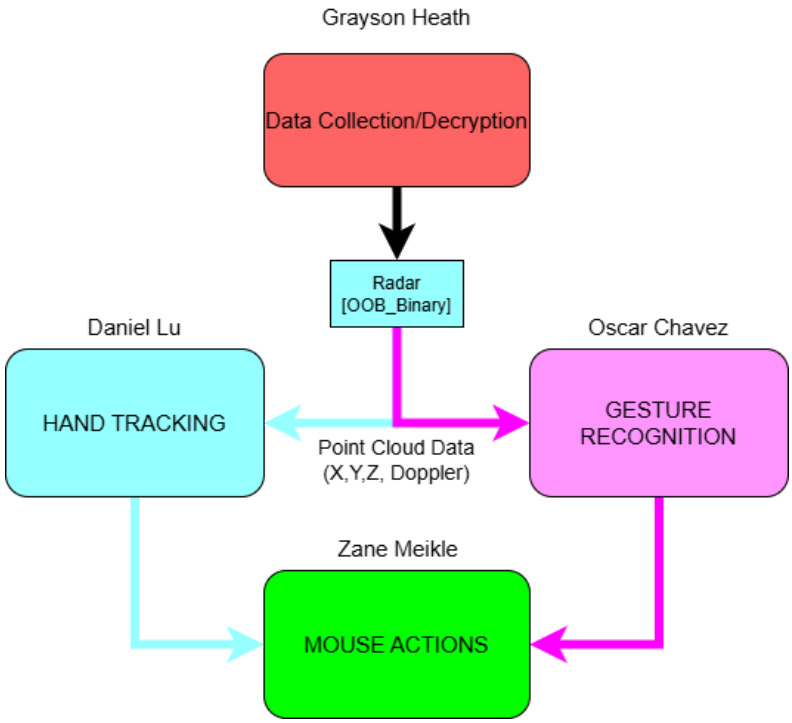


Figure 1: Functional System Diagram

Chirp Optimization and Data Collection

The Chirp Optimization and Data Collection subsystem is responsible for the radar functionality, and establishing a sufficient chirp configuration and outputting useful data to be used for processing. A chirp signal is a pulsed signal that increases frequency over time, and further allows for an accurate range resolution and target detection to be observed. Heavy trial-based experiments and research will be conducted to yield an accurate target resolution for gestures using the chirp signal, and will be continuously modified as needed. Following the optimization of the chirp signal, the output data will be collected and transmitted to the computer in a meaningful format for further processing through python. The processed data will be passed onwards to the next respective subsystems.

Mouse Positioning

The Mouse Positioning subsystem is responsible for calculating a 2D unit vector in the direction of movement of the hand. By creating a group tracker module, which utilizes clustering and gesture recognition in order to identify and monitor the existence of objects in an environment, a 2D position plot of the point cloud will be created. The gesture recognition unit will detect the viewing and remove the return signal data from the surrounding environment.

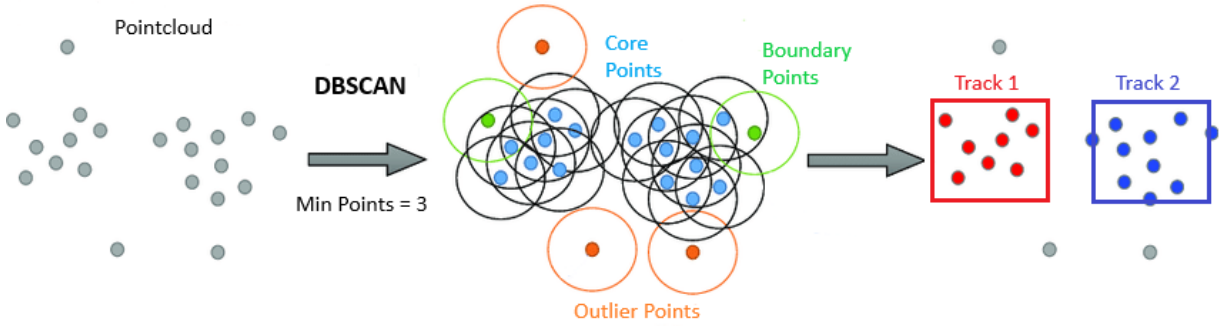


Figure 2: Prediction state and measurement state

There are two tracking algorithms, position tracking and average weight velocity tracking. Average weight velocity tracks the velocity of the user's hand and converts that velocity vector into a movement vector for the cursor. Position tracking algorithm tracks the relative position of the user's hand with respect to the radar and maps an area in front of the radar to the screen. In short, the mouse cursor will move to a position relative to the hand's location.

Gesture Recognition
The Gesture Recognition subsystem is responsible for producing a gesture ID representing different mouse actions such as left-click, right-click, double left-click and scroll. This subsystem will use a Machine Learning(ML) model to identify a hand and discern between 4 gestures, push, pull, shine, and shake. The AI model will be sensitive enough to have less than 1 false positive per minute. The Machine Learning Algorithm used is Convolutional Neural Network. The CNN has been used for the Computer Vision implementation, which will make the output of the mmWave Radar implementation more predictable.

Input Mapping to Computer
The Input Mapping to Computer subsection is responsible for taking the vector from the Mouse positioning subsection and converting that to corresponding mouse movements on the computer. As well as taking the gestures recognized by the Gesture Recognition and converting those to mouse actions on the computer. On top of that, this subsystem should have a mouse smoothing function that is able to make the movement of the mouse look smoother.

3.4. Modes of Operations

Active
The mouse is in active mode when a hand is recognized by the Gesture Recognition system. When the mouse is in active mode it will operate normally with full functionality. Full functionality represents hand tracking and gesture recognition.

Idle
The mouse is in idle mode when no hand is recognized. In this mode, the mouse is on standby and will produce no mouse input.

3.5. Users
The targeted users of the mmWave Radar mouse are people who prefer to have a hands free option to controlling their computer. Basic computer literacy is all that is required to operate the mouse. The mouse will contain a manual for operation, and the gestures will reflect actions done in a physical mouse.

3.6. Support
To help the customers install and use the radar mouse, two instruction manuals will include standard gestures and provide information on installation and self-optimization settings.

4. Scenario(s)

4.1. Presentation

The virtual computer mouse will be an excellent resource when presenting. Navigating slides with the mouse will be easy and convenient. It would be comparable to a presentation clicker/pointer in functionality.

4.2. Medical

The hands free operation of the mouse is very useful for maintaining a clean and sterile environment in a hospital. Doctors and nurses could use the virtual mouse to control a monitor during surgery, eliminating the need for physical interaction.

5. Analysis

5.1. Summary of Proposed Improvements

- The mmWave radar is more power efficient than a camera based virtual mouse.
- The mmWave radar will be less processing intensive due to the computer processing hex-streamed frame data from the radar, instead of a video or photo data format.
- The mmWave radar can successfully operate in dark and hazy environments without impacting performance.
- The mmWave radar is hands-free making it more hygienic than a regular mouse.

5.2. Disadvantages and Limitations

- The mmWave radar has a lower ability to discern finger gestures due to a lower resolution when compared to a camera.
- The mmWave radar is bigger than a conventional computer camera.
- The mmWave radar has a low polling rate ~ 30Hz (max).
- The mmWave radar is more expensive compared to a camera based solution.

5.3. Alternatives

- **Virtual Mice -** Virtual mice would be a camera based solution. They would be more power and processing-intensive, and also would not function in the dark. However, they would be more accurate in determining gestures, have a smaller physical footprint, and would be significantly cheaper than our solution.
- **Physical Mouse -** A standard mouse would be far more accurate for recognizing inputs, tracking position, and running at a higher polling rate. On the other hand, it requires physical contact, making it less hygienic.
- **Touch Screens -** Physical finger inputs are able to interact directly with the screen, and would be more accurate for movement and for recognizing inputs. Nevertheless, they would be unhygienic as the user must physically touch the screen.

5.4. Impact

This product will likely have very little impact on the environment other than the environmental impact of manufacturing it, though all devices of this type would have a similar impact. There is not likely to be any major societal impact as this will replace a device that is commonly used by society.

Virtual Computer Mouse using mmWave Radar (TI)

Oscar Chavez Araiza
Greyson Heath
Daniel Lu
Zane Meikle

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – 1.3
28 April 2025

Functional System Requirements
FOR
Virtual Computer Mouse using mmWave Radar (TI)

Team <71>

Approved by:

_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	9/22/2024	Zane Meikle		Revision 1.0
1.1	11/11/2024	Zane Meikle		Revision 1.1
1.2	12/3/2024	Greyson Heath		Revision 1.2
1.3	4/28/2025	Zane Meikle		Revision 1.3

Table of Contents

Table of Contents **20**
List of Tables **21**
List of Figures **22**
1. Introduction **23**
1.1. Purpose and Scope..... 23
1.2. Responsibility and Change Authority..... 23
2. Applicable and Reference Documents 24
2.1. Applicable Documents..... 24
2.2. Reference Documents..... 24
2.3. Order of Precedence..... 24
3. Requirements **25**
3.1. System Definition..... 25
3.2. Characteristics..... 26
3.2.1. Functional / Performance Requirements..... 26
3.2.2. Physical Characteristics..... 27
3.2.3. Electrical Characteristics..... 27
3.2.4. Environmental Requirements..... 28
3.2.5. Failure Propagation..... 28
4. Support Requirements **30**
Appendix A Acronyms and Abbreviations **31**
Appendix B Definition of Terms **32**
Appendix C Interface Control Documents **33**

List of Tables

Table 1: Subsystem Responsibility..... 24

Table 2: Applicable Documents.....24

List of Figures

Figure 1: Concept Image..... 23

Figure 2: System Block Diagram.....25

1. Introduction

1.1. Purpose and Scope

Operating a computer with a virtual mouse has the potential to improve a variety of computing tasks. Rather than using a camera with computer vision to enable gesture recognition, our goal is to utilize Texas Instruments' mmWave radar to build upon and improve flaws associated with computer vision.

Users will be able to directly interact with their computer using their hand as a virtual mouse. With our system, the mmWave radar will actively interpret user gestures and appropriately translate them to a meaningful format for the computer to use. Machine learning algorithms will be incorporated to ensure accurate signals are detected, and to further reduce external noise. Our system shall be able to control the mouse cursor, and mouse actions such as left-click, right-click, double left-click, and scroll.

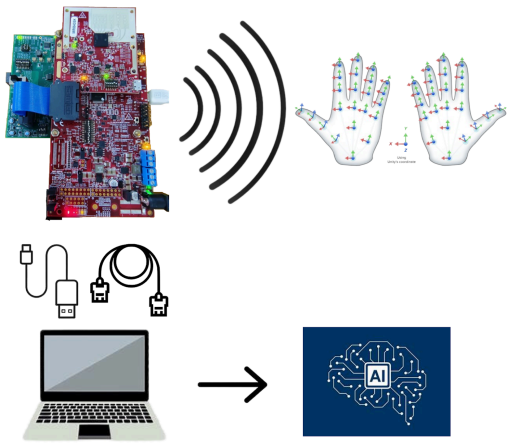


Figure 1: Concept Image

1.2. Responsibility and Change Authority

The team leader, Zane Meikle, will be responsible for verifying that all project requirements are met. If these requirements are to be altered, only the team leader, Professor John Lusher, and TI can approve said changes.

Subsystem	Responsibility
Chirp Optimization and Data Collection	Greyson Heath
Mouse Positioning	Daniel Lu
Gesture Recognition	Oscar Chavez Araiza
Input Mapping to Computer	Zane Meikle

Table 1: Subsystem Responsibility

2. Applicable and Reference Documents

2.1. Applicable Documents

Document Number	Revision/Release Date	Document Title
SPRUIJ4A	Revision A - May 2018	DCA1000EVM Data Capture Card User's Guide
SWRU546E	Revision E - November 2020	60GHz mmWave Sensor EVMs

Table 2: Applicable Documents

2.2. Reference Documents

Due to all documents being immediately applicable to the virtual mouse mmWave radar system, there are no referenced documents as of now.

2.3. Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings, or other documents that are invoked as "applicable" in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

3.1. System Definition

The virtual mouse mmWave radar is a reliable way to operate a computer without physically interacting with a traditional computer mouse. It enables users to actively use gestures to simulate the use of a computer mouse. The virtual mouse mmWave radar consists of four subsystems: Chirp Optimization and Data Collection, Mouse Positioning, Gesture Recognition, and System to Mouse Input Mapping.

The mmWave radar will collect data as a frame which will be transmitted to the computer via microUSB to USB. The data will then be analyzed by an ML algorithm to determine if a hand/hand gesture was detected. A separate algorithm will track the hand to produce a location for the mouse pointer. The gesture ID produced by the ML algorithm and the position vector will be passed to another program which will produce the mouse inputs.

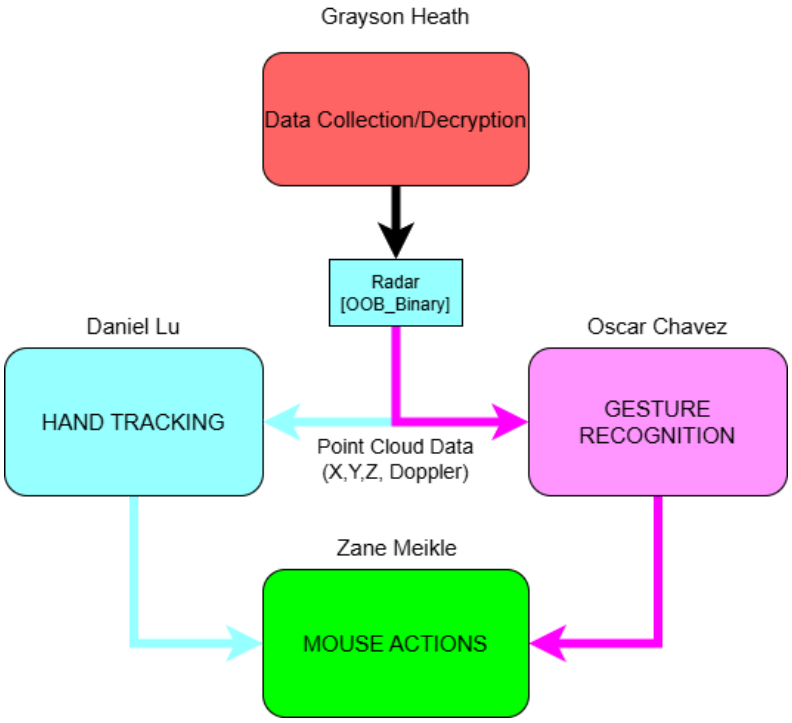


Figure 2: Block Diagram of System

Using computer USB ports, and a standard 120V outlet, the mmWave radar will be connected to the user's computer and powered by a power brick attached to a standard 120V outlet. Software development kits from Texas Instruments will be used to communicate and properly configure the mmWave radar, and ultimately interpret the data seen from the radar via the computer USB port. The data will be used by an ML program to identify hand gestures used to represent right and left clicks. Once a hand gesture is recognized the program will output a gesture ID which will be passed to the mapping program.

Mouse positioning is converted from the data of radar signals. With the application of the gesture recognition system and the SDK from Texas Instruments, position, velocity, and

acceleration of the gesture will be plotted as a 3D graph. All the position vectors will be sent to the Input Mapping Program. The gestures and positioning vectors from the ML program and Mouse Positioning Program will then go to the Input Mapping Program where they will be translated into actual mouse movements and actions.

3.2. Characteristics

3.2.1. Functional / Performance Requirements

3.2.1.1. False Positive

The maximum number of non-gestures triggering a mouse input shall be less than 1 per minute.

Rationale: The system's gesture recognition depends on an AI model, thus mischaracterization by the AI is inevitable.

3.2.1.2. Recognition of Position

The position data is the reflected signal received by the radar. Multiple points per frame is provided for gesture recognition demo and the tracking program for the generation of moving path and AI training of gesture recognition.

Rationale: The recognition and positioning system must detect the right gesture and position from hundreds of example tests with accuracy close to 100%.

3.2.1.3. Angle of Detection

The mmWave radar should be able to detect within a range of 60 degrees from where it is positioned.

Rationale: The user will be free to control the virtual mouse within a range of angles, meaning the user does not have to be directly in front of the antenna.

3.2.1.4. Distance of Detection

The mmWave radar should be able to detect within a range of ~1.5 feet from where it is positioned.

Rationale: The user will have an adequate area to move their hand to represent a smooth cursor on the monitor..

3.2.1.5. Gesture Recognition Accuracy

The mmWave radar should recognize the hand gesture 90% regardless of hand-size.

Rationale: The user should have reliable ability to control the mouse regardless of hand shape and size.

3.2.1.6. Input Mapping Requirements

The Input Mapping Subsystem must be able to move the mouse up, down, left, and right. It must also be able to left and right click.

Rationale: The mouse needs to move in all the directions a normal mouse would

3.2.1.7. mmWave Radar and Computer Functionality

The mmWave radar must be able to communicate with the user's computer and transmit data.

Rationale: The radars must communicate with the computer for further data processing to occur.

3.2.1.8. Data Decoding

The respective data received from the mmWave radar must be properly decoded and used for mouse inputs and actions.

Rationale: The virtual mouse requires meaningful data in order to properly function.

3.2.1.9. Mouse Smoothing

The mouse smoothing will generate interpolated frames of movements based on the variable that is set in the GUI. This function will not produce errors or leave a zombie process.

Rationale: Since the radar is operating at a max of 30hz, this function is useful for making the movement look more natural.

3.2.1.10. Graphical User Interface

The graphical user interface must adjust corresponding settings appropriately, and throw error codes if invalid options are entered.

Rationale: The graphical user interface serves to assist the user in changing settings in a convenient manner.

3.2.1.11. Tracking Algorithms

The tracking program must track the target object' position with high accuracy according to the reflected signal in the 3D plane from the radar.

Rationale: The virtual mouse requires meaningful data in order to properly function.

3.2.2. Physical Characteristics

3.2.2.1. System Area

The system can operate over any area up to ~1.5 feet so long as the mmWave radar is within direct line of sight of the user's gestures.

Rationale: As long as the antennas are oriented towards the user and their gestures, and the mmWave radar remains flat on its surface, the assembled package will have no area restriction.

3.2.2.2. Volume Envelope

The volume envelope of the assembled mmWave radar excluding the power, ethernet, and microUSB cables shall not exceed a height of 1 inches, a width of 2 inches, and a length of 5.5 inches.

Rationale: The assembled boards will lie upon one another, and will not occupy a large amount of space.

3.2.2.3. Assemble Location

The assemble location information for the virtual mouse mmWave radar system shall be captured in the Search and Rescue System ICD.

Rationale: As the assembled mmWave radar will lie upon the user's desk, the mounted location requires a specified orientation such that the antenna is facing towards the user and their gestures.

3.2.3. Electrical Characteristics

3.2.3.1. Inputs

a. The presence or absence of any combination of the input signals in accordance with ICD specifications applied in any sequence shall not damage the user's computer nor the mmWave radar, reduce its life expectancy, or cause any malfunction, either when the unit is powered or when it is not.

b. No sequence of command shall damage the user's computer nor the mmWave radar, reduce its life expectancy, or cause any malfunction.

Rationale: By design, should limit the chance of damage or malfunction by user/technician error.

3.2.3.1.1 Power Consumption

The maximum peak power of the system shall not exceed 12.5 watts.

Rationale: This is a requirement that ensures proper and continuous operation of the mmWave radar.

3.2.3.1.2 Input Voltage Level

The input voltage level for the virtual mouse mmWave radar shall be 5V.

Rationale: The mmWave radar is designed to operate off of a standard 5V USB connection to the user's computer.

3.2.3.1.3 External Commands

The virtual mouse mmWave radar shall document all external commands in the appropriate ICD.

Rationale: The ICD will capture all interface details from the low level electrical to the high-level packet format.

3.2.3.2. Outputs

3.2.3.2.1 Data Output

The system shall use the input data from mmWave radar to do target tracking, which is done by the positioning subsystem. The positioning will generate output data of the coordinates, velocities in the axis of radar.

Rationale: Being able to see active data output will allow the user to ensure the mmWave radar is properly connected to the computer and functioning.

3.2.4. Environmental Requirements

The mmWave radar shall be designed to withstand and operate in the environments and laboratory tests specified in the following section.

Rationale: The mmWave radar may be placed in a variety of environments, thus the requirements listed below describe the limitations

3.2.4.1. Thermal

The virtual mouse mmWave radar shall be able to function properly in an environment with temperatures ranging from -4°F to 140°F.

Rationale: This is a requirement specified by Texas Instruments' for proper functioning of the circuit boards.

3.2.4.2. Rain

The mmWave radar shall be distanced from the water.

3.2.4.3. Ambient Brightness

The mmWave radar shall be able to operate in any environment regardless of ambient brightness.

Rationale: Due to the method of collecting data using electromagnetic fields, ambient brightness will not impact the resolution of the radar.

3.2.5. Failure Propagation

All errors in the subsystems will be passed through to the Input Mapping Program's terminal where the program is running. If there is not output from a subsystem or if the subsystem is not responding then whichever subsystem detects the

3.2.5.1. Gesture Recognition Program and Mouse Positioning Program

If these subsystems are unable to connect or receive data from the mmWave radar, then they are to send an error code to the Input Mapping Program.

3.2.5.2. Input Mapping Program

If this program receives an invalid input it must be able to handle it and give a corresponding error message and then end the program.

4. Support Requirements

The mmWave radar requires a computer using Windows 10 or later. Users must provide a microUSB cable to allow the mmWave radar to communicate with the computer. Associated software that enables the functionality of the device will be required to communicate with the mmWave radar. For the code to operate, Python 3.8.1 must be used.

Appendix A: Acronyms and Abbreviations

3D	Three Dimensional
AI	Artificial Intelligence
BIT	Built-In Test
CCA	Circuit Card Assembly
CNN	Convolutional Neural Network
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
EO/IR	Electro-optical Infrared
FOR	Field of Regard
FOV	Field of View
FSR	Functional System Requirements
GPS	Global Positioning System
GUI	Graphical User Interface
Hz	Hertz
ICD	Interface Control Document
ID	Identification
kHz	Kilohertz (1,000 Hz)
LCD	Liquid Crystal Display
LED	Light-emitting Diode
mA	Milliamp
mmWave	Millimeter Wave
MHz	Megahertz (1,000,000 Hz)
ML	Machine Learning
MTBF	Mean Time Between Failure
MTTR	Mean Time To Repair
mW	Milliwatt
PCB	Printed Circuit Board
RF	Radar Frequency
RMS	Root Mean Square
SDK	Software Development Kit
TBD	To Be Determined
TI	Texas Instruments
TTL	Transistor-Transistor Logic
USB	Universal Serial Bus
VME	VERSA-Module Europe

Appendix B: Definition of Terms

Appendix C: Interface Control Documents

Virtual Computer Mouse using mmWave Radar (TI)

Oscar Chavez Araiza
Greyson Heath
Daniel Lu
Zane Meikle

INTERFACE CONTROL DOCUMENT

REVISION – 1.2
28 April 2025

INTERFACE CONTROL DOCUMENT
FOR
Virtual Computer Mouse using mmWave Radar (TI)

TEAM <71>

APPROVED BY:

_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	9/22/2024	Zane Meikle		Revision 1.0
1.1	12/3/2024	Greyson Heath		Revision 1.1
1.2	4/28/2025	Zane Meikle		Revision 1.2

Table of Contents

Table of Contents **37**
List of Tables **38**
List of Figures **39**
1. Overview **40**
2. References and Definitions **41**
2.1. References....41
2.2. Definitions....41
3. Physical Interface **42**
3.1. Weight....42
3.2. Dimensions....42
3.3. Assembly Location....42
4. Thermal Interface **43**
5. Electrical Interface **44**
5.1. Input Power....44
5.2. Voltage and Current Levels....44
6. Communications / Device Interface Protocols **45**
6.1. Host Device....45

List of Tables

Table 1: mmWave Radar Weight.....42
Table 2: mmWave Radar Dimensions.....42
Table 3: Assembled mmWave Radar Dimensions.....42
Table 4: Maximum Voltage and Current Levels..... 46
Table 5: Expected Average Voltage and Current Levels.....46

List of Figures

No table of figures entries found.

1. Overview

The Interface Control Document will provide users with information about the operating status of the product in different environments, such as thermal, physical, and electrical conditions.

2. References and Definitions

2.1. References

Refer to section 2.2 of the Functional System Requirements document.

2.2. Definitions

A	Amp
TBD	To Be Determined
USB	Universal Serial Bus
W	Watt
V	Volt

3. Physical Interface

3.1. Weight

Dimensions are in inches.

3.1.1. mmWave Radar

Component	Weight	Number of Items	Total Weight
IWR6843AOPEVM	TBD	1	TBD

Table 1: mmWave Radar Weight

3.2. Dimensions

Dimensions are in inches.

3.2.1. mmWave Radar

Component	Length (in)	Width (in)	Height (in)
IWR6843AOPEVM	4.75	1.22	0.18

Table 2: mmWave Radar Dimensions

3.2.1. Assembled mmWave Radar (with Case)

Component	Length (in)	Width (in)	Height (in)
Assembled mmWave Radar	5	2	1

Table 3: Assembled mmWave Radar Dimensions

3.3. Assemble Location

The virtual mouse mmWave radar will be assembled and placed on top of the user's computer. The assembled mmWave radar will need to be oriented such that the antenna on the IWR6843AOPEVM modules are facing towards the area the user will be located. This will ensure that the accurate capturing of gestures is performed.

4. Thermal Interface

The mmWave radar contains a heat sink for the radar module, that being the IWR6843AOPEVM. Due to the consistent output and active monitoring for the radar signals, the heat sink will keep the module from overheating and prevent decreased efficiency. This is the only thermal interface for the mmWave radar, as all other components are capable of dissipating their own heat.

5. Electrical Interface

5.1. Input Power

5.1.1. mmWave Radar

The mmWave radar will receive power from the user's computer via microUSB to USB. User's will have to provide their own power to their computer.

5.2. Voltage and Current Levels

5.2.1. Maximum Values

Component	Voltage (V)	Current (A)	Power (W)
mmWave Radar	5	2.5	12.5

Table 4: Maximum Voltage and Current Levels

These values are based on the specifications provided by Texas Instruments.

5.2.2. Expected Average Values

Component	Voltage (V)	Current (A)	Power (W)
mmWave Radar	5	0.4	2

Table 5: Expected Average Voltage and Current Levels

6. Communications / Device Interface Protocols

6.1. Host Device

The host device and the mmWave radar communicate via a single USB 2.0 micro usb cable.

Virtual Computer Mouse using mmWave Radar (TI)

Oscar Chavez Araiza
Greyson Heath
Daniel Lu
Zane Meikle

SCHEDULE AND VALIDATION

REVISION – 1.2
28 April 2025

List of Tables

Table 1: Schedule....49
Table 2: Validation Plan....52

List of Figures

Figure 1: Fall Semester Gantt Chart..... 51
Figure 2: Spring Semester Gantt Chart..... 51

Schedule:

Work	End Date	Owner	Status	Date Completed
Concept of Operations	9/15/2024	All	Complete	9/14/2024
Functions System Requirements	9/26/2024	All	Complete	9/23/2024
Interface Control Document	9/26/2024	All	Complete	9/23/2024
Midterm Presentation	9/25/2024	All	Complete	9/25/2024
Radar Out-of-box Demo (mmWave and 6843)	9/30/2024	Greyson	Complete	9/30/2024
Gesture Resolution R&D (Chirp Optimization)	10/12/2024	Greyson	Complete	10/10/2024
Input Mapping Intro Project	10/14/2024	Zane	Complete	9/30/2024
Machine Learning Intro Project	10/14/2024	Oscar	Complete	9/30/2024
Chirp Optimization Intro Project	10/14/2024	Greyson	Complete	9/30/2024
Data Processing Intro Project	10/14/2024	Daniel	Complete	9/30/2024
Find a fitting ML model to use a base for our ML model	10/14/2024	Oscar	Complete	10/14/2024
Output Real-time Data using DCA1000EVM	10/16/2024	Greyson	Complete	10/28/2024
Mouse Positioning Research	10/20/2024	Daniel	Complete	10/14/2024
Input Mapping get initial code (mouse movement and actions) working	10/20/2024	Zane	Complete	10/14/2024
Train ML Model	10/21/2024	Oscar	Complete	10/28/2024
Status Update Slides	10/22/2024	All	Complete	10/12/2024
Status Update Presentation	10/23/2024	All	Complete	10/13/2024
Test/Validate ML model	11/01/2024	Oscar	Complete	10/28/2024
Refine and stress-test Chirp Optimization	11/01/2024	Greyson	Complete	11/4/2024
Validate code and make a	11/14/2024	Zane	Complete	11/11/2024

configurable tuning system to ensure proper scaling				
Test the hand position tracker	11/14/2024	Daniel	Complete	11/4/2024
Make Final Presentation	11/19/2024	All	Complete	11/10/2024
Final Presentation (403)	11/20/2024	All	Complete	11/11/2024
Make sure all subsystems are ready for Final Demo	11/25/2024	All	Complete	11/23/2024
Final Demo (403)	11/26/2024	All	Complete	11/23/2024
Make Final Report	12/5/2024	All	Complete	12/3/2024
Final Report (403)	12/5/2024	All	Complete	12/3/2024
Update Presentation 1	01/27/2025	All	Complete	01/27/2025
System enclosure	02/03/2025	Greyson	Complete	02/03/2025
Subsystem Integration	02/03/2025	All	Complete	02/10/2025
Mouse Movement Optimization	02/10/2025	Daniel	Complete	02/17/2025
Update Presentation 2	02/10/2025	All	Complete	02/10/2025
New Gesture Integration	02/10/2025	Oscar	Complete	03/03/2025
Interactive GUI	03/10/2025	Greyson	Complete	03/24/2025
Remove IC Boost From System	02/10/2025	Greyson	Complete	02/10/2025
Further Mouse Movement Optimization	02/24/2025	Daniel	Complete	03/24/2025
Create Case for Radar	02/24/2025	Greyson	Complete	02/24/2025
Check Viability of 1 Radar	02/24/2025	Oscar	Complete	02/24/2025
Update Presentation 3	02/24/2025	All	Complete	02/24/2025
Get one board to track and recognize gestures	03/03/2025	Oscar	Complete	03/03/2025
Improve execution time (parallelization)	03/17/2025	All	Complete	03/17/2025
Integrate new ML model to system	03/17/2025	Oscar	Complete	03/17/2025

Update Presentation 4	03/17/2025	All	Complete	03/17/2025
Test Final Product	03/24/2025	All	Complete	04/31/2025
Update Presentation 5	03/31/2025	All	Complete	03/31/2025
Final Presentation	04/14/2025	All	Complete	04/14/2025
Final Demo	04/21/2025	All	Complete	04/21/2025
Project Showcase	04/25/2025	All	Complete	04/25/2025
Final Report	04/28/2025	All	Complete	04/26/2025

Table 1: Schedule

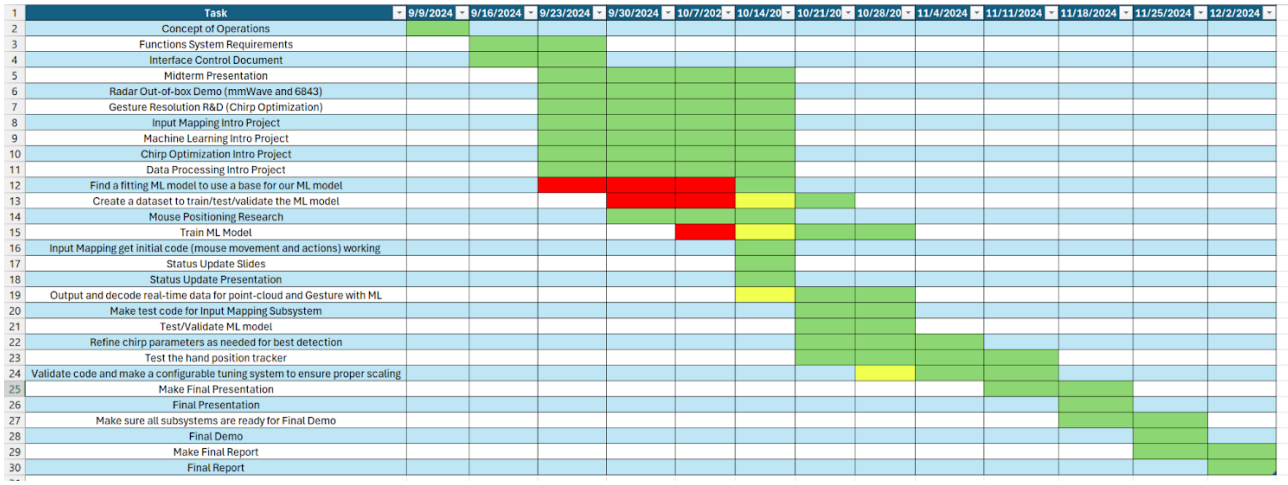


Figure 1: Fall Semester Gantt Chart

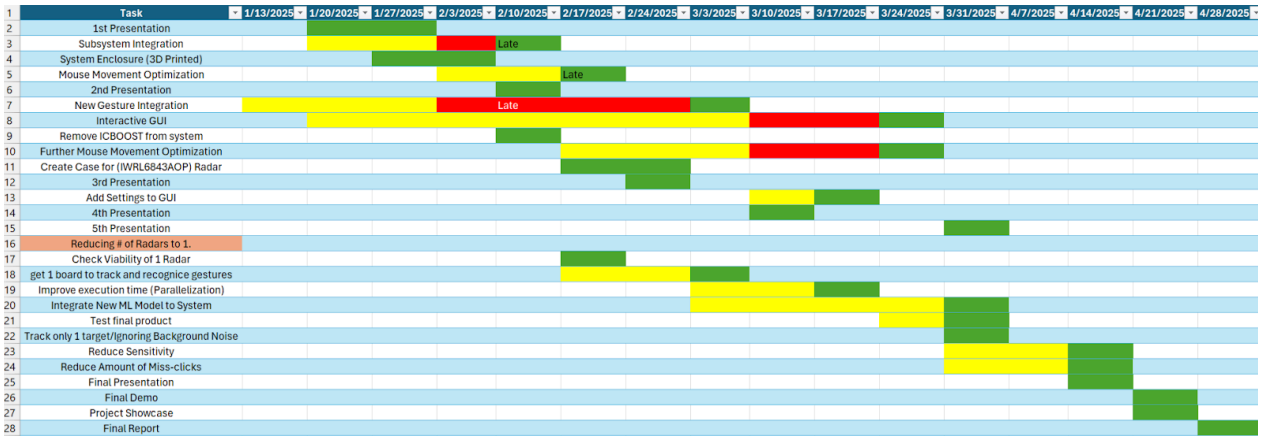


Figure 2: Spring Semester Gantt Chart

Validation Plan:

Section	Task	Specification	Test	Status	Owner
3.2.1.6	Mouse Movement	Mouse can be moved to all corners of the screen by the program.	While running the main program use hand movements to reach all corners.	Passed	Zane
3.2.1.6	Mouse Actions	The program must be able to right click, left click, scroll, and double left click.	While running the main program use hand gestures to initiate all configurable mouse actions.	Passed	Zane
3.2.5.2	Mouse Input Error Handling	The program must properly handle all invalid inputs.	Intentionally create errors that are passed to the input mapping subsection.	Passed	Zane
3.2.1.9	Mouse Smoothing Function	Mouse smoothing when turned on generates the frames as specified in the GUI (to the limitations of the hardware)	Using the GUI activates mouse smoothing and checks different amounts of smoothing.	Passed	Zane
3.2.1.9	Mouse Smoothing Error Handling	Mouse smoothing program does not throw an error or leave the terminal running (the process did not end properly).	When initializing the mouse smoothing thread, check for its existence as a zombie process after the main program terminates.	Passed	Zane
3.2.1.4 3.2.1.6	Gesture positioning	The coordinates of the hand should be calculated when deleting the error points.	Check position calculations in terminal and compare them to actual hand position data	Passed	Daniel
3.2.1.6 3.2.1.8 3.2.1.11 3.2.3.2.1	Velocity accuracy(redacted)	The output velocity of the moving hand should be close to a multiple of its true velocity.	Using a phone to record hand movements, measure the hands true speed and the velocity calculated by the algorithm	Passed	Daniel
3.2.1.4 3.2.1.6	Position tracking program accuracy	In the position tracking program, the mouse should move to the relative position of the hand according to the scale of the computer screen.	Perform horizontal movement, vertical movement, and circle movement using your hand. Compare the moving path of the mouse with the	Passed	Daniel

			movement of the hand.		
3.2.1.6 3.2.1.8 3.2.3.2.1	Average Velocity Program accuracy	In the average velocity program, the mouse should move according to the average weight velocity of the user's hand.	Perform horizontal movement, vertical movement, and circle movement using your hand. Compare the moving path of the mouse with the movement of the hand.	Passed	Daniel
3.2.1.5 3.2.1.3	Hand Gesture Recognition	ML model can recognize the user's hand gestures.	Ask someone with no prior experience with the program to try to imitate the gestures. The results will be recorded.	Passed	Oscar
3.2.1.3	Gesture Recognition Accuracy	ML model can correctly identify and categorize hand gestures at least 90% of the time.	Using the results from the testing with an outside party, calculate the accuracy of the model.	92%, Passed	Oscar
3.2.1.7	Functional Radar System	The IWR6843AOPEVM properly communicates with the computer and transmits data.	Check frame data for correct output in the terminal.	Passed	Greyson
3.2.1.8	Data Decoding(redacted)	The python script accurately decodes the TLV data seen from the radar.	Use the parser algorithm to obtain relevant data.	Passed	Greyson
3.2.1.3 3.2.1.4	Radar Angle and Distance(redacted)	The radars can capture gestures and pointer positions within a 60 degrees from the horizontal, and range up to 2 feet	Test radar tracking and gesture recognition at different angles and distances.	Passed	Greyson
3.2.1.5	Hand Size	Radar will recognize gestures regardless of hand size.	Asked multiple people with varying hand sizes to test the gesture recognition.	Passed	Oscar
3.2.1.10	GUI	Available settings change the relevant functionality of the program.	Changed every setting individually, and in combination with other settings.	Passed	Greyson
3.2.1.10	GUI	If a setting input is invalid the relevant variables will not be	Intentionally input invalid settings into GUI.	Passed	Greyson

		changed.			

Table 2: Validation Plan

Virtual Computer Mouse using mmWave Radar (TI)

Oscar Chavez Araiza
Greyson Heath
Daniel Lu
Zane Meikle

SUBSYSTEM REPORTS

REVISION – 1.1
28 April 2025

SUBSYSTEM REPORT

FOR

Virtual Computer Mouse using mmWave Radar (TI)

TEAM <71>

APPROVED BY:

_____

Project Leader Date

_____

Prof. Kalafatis Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	12/03/2024	Zane Meikle		Revision 1.0
1.1	04/28/2025	Zane Meikle		Revision 1.1

Table of Contents

Table of Contents **58**
List of Tables **59**
List of Figures **60**
1. Introduction **61**
2. Chirp Configuration and Data Collection Subsystem Report (Greyson Heath) **62**
2.1. Subsystem Introduction..... 62
2.2. Subsystem Details..... 62
2.3. Subsystem Validation.....63
2.4. Subsystem Conclusion..... 66
3. Mouse Positioning Subsystem Report (Daniel Lu) **67**
3.1. Subsystem Introduction..... 67
3.2. Subsystem Details..... 67
3.3. Subsystem Validation.....68
3.4. Subsystem Conclusion..... 70
4. Gesture Recognition Subsystem Report (Oscar Chavez Araiza) **71**
4.1. Subsystem Introduction..... 71
4.2. Subsystem Changes.....71
4.3. Subsystem Details..... 71
4.4. Subsystem Validation.....72
4.5. Subsystem Conclusion..... 74
5. Input Mapping Subsystem Report (Zane Meikle) **75**
5.1. Subsystem Introduction..... 75
5.2. Subsystem Details..... 75
5.3. Subsystem Validation.....75
5.4. Subsystem Conclusion..... 79

List of Tables

Table 1: Validation For Position Tracking System.....69
Table 2: Validation For Gesture Recognition Subsystem.....72
Table 3: Detection Statistics..... 73

List of Figures

Figure 1: Functional visual block diagram of the mmWave radar.....62
Figure 2: Windows Device Manager Displaying Radar Device.....63
Figure 3: Raw Radar UART Hex-stream Data.....64
Figure 4: Decoded Pointer Position Data.....64
Figure 5: GUI Terminal and Terminal Response.....65
Figure 6: Downsizing to single radar device.....65
Figure 7: Radar Case and Mount.....66
Figure 8: Data points selecting process.....67
Figure 9: Sample data in x-y-z axis.....68
Figure 10: Sample moving velocity in x-y plane in 5 frames.....69
Figure 11: Sample moving velocity in x-y plane per frame.....70
Figure 12: CNN accuracy & layout.....71
Figure 13: Gestures.....72
Figure 14: Test Confusion Matrix.....73
Figure 15: Real Time Confusion Matrix.....73
Figure 16: Mouse Moving.....76
Figure 17: Mouse Left Clicking Opening Windows Start Button.....76
Figure 18: Mouse Right Clicking Opening Options.....77
Figure 19: Invalid Values.....77
Figure 20: Invalid Left-Click.....77
Figure 21: Invalid Right-Click.....77
Figure 22: Mouse Smoothing Function Moving the Mouse.....78
Figure 23: Terminal Ending the Program Properly Without a Zombie Process.....78

1. Introduction

The virtual computer mouse using mmWave radar will track a human hand, moving the computer mouse and also recognize certain gestures to left and right click on the computer. This system was broken up into four subsystems, that being the Chirp Configuration and Data Collection Subsystem, Mouse Positioning Subsystem, Gesture Recognition Subsystem, and the Input Mapping Subsystems. All of these subsystems were validated as working so they will be able to be integrated to create the final product as outlined in the Conops, FSR, and ICD.

2. Chirp Configuration and Data Collection Subsystem Report (Greyson Heath)

2.1. Subsystem Introduction

The chirp configuration and data collection subsystem is designed to oversee the successful functionality of the radar, properly configure the radar chirp to accurately detect user hand inputs, and accurately decode and pass forward the processed radar data. This subsystem is powered and communicates with the computer via a single microUSB to USB ports. Python is used to further decode the processed radar data before passing it forward to the next subsystems.

2.2. Subsystem Details

A visual block diagram of the subsystem is displayed below.

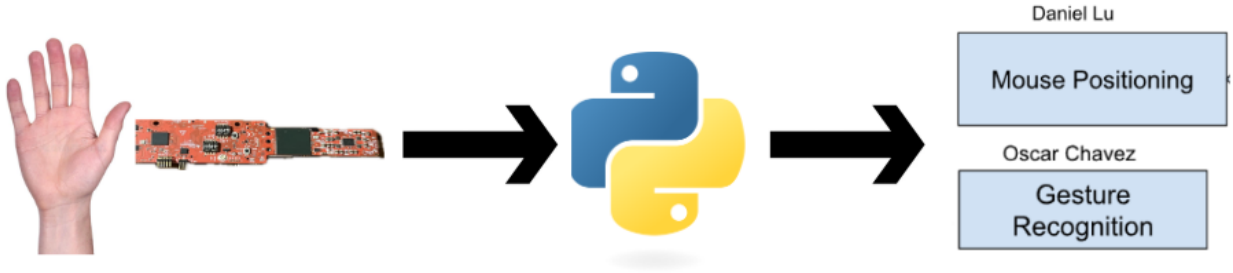


Figure 1: Functional visual block diagram of the mmWave radar

The radar device operates with a microUSB to USB cables for power and communication with the user's computer. The radar device operates on a variable ‘frame-by-frame’ basis, such as a camera recording 30 frames per second (FPS). Per frame, the radar device will detect a variable number of points depending on the position and location of the user's hand. Each point contains data that will be further processed by the python program, and the data fields are dependent on which binary the radar is flashed with.

Using Texas Instrument provided radar binaries, the radar device is flashed with their point cloud binary. This binary contains location-based information per point, and is used to track the movement of the user's hand. Additionally, it enables custom chirp configurations, and has since been refined to best detect and track a human’s hand. This binary is used for both gesture detection and hand movement detection.

The radar device communicates with the computer with a hex-stream via UART (universal asynchronous receiver/transmitter) through the microUSB to USB cable. Per frame, a packet of data is sent to the computer via UART. Each packet contains a fixed size frame header which always begins with the Texas Instrument ‘magic word’ denoting the start of the frame header. Additional information regarding the radar device and binary used is provided in the frame header. Following the frame header, a variable number of TLV (type-length-value) items are present depending on the number of points detected. One point detected represents one TLV item, and the TLV contains the point-data that is extracted and decoded using the python program. After the data is extracted and decoded in real-time, it is passed forward to the next subsystems.

2.3. Subsystem Validation

The chirp configuration and data collection subsystem was validated for the following three fields:

- **Functional radar system performance**
  - Does the radar properly communicate with the computer and transmit its respective data?
- **Data extraction and decoding**
  - Does the python program successfully and accurately decode the hex-stream seen from the radar?
- **Adequate radar angle and distance**
  - Does the radar capture the user's hand within a ~60 degree range up to ~1.5 feet?
- **GUI performance and error checking**
  - Does the GUI properly change settings and throw errors if an invalid setting is entered?

After connecting the radar device to the computer using a microUSB cable, there is a simple test to see if the computer initially detects the device. Using Windows device manager under ‘Ports (COM & LPT)’, the radar will display as an XDS110 accessory and output data via the ‘Class Auxiliary Data Port’, assuming a proper connection to the computer was made.

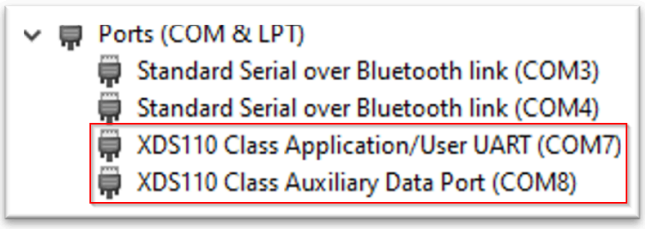


Figure 2: Windows Device Manager Displaying Radar Device

Furthermore, after flashing the radar with the proper binary, we are able to view the raw UART hex-stream using python. Displayed below is a screenshot of the raw radar data, and the blue highlighted portion denotes TI’s ‘magic word’, which denotes the beginning of each frame header. This confirms and validates the proper functionality of the radar devices.

Figure 3: Raw Radar UART Hex-stream Data

Using the python data parser script, the hex-streamed data is successfully decoded and the important fields are pulled from each TLV. Seen below is an example of the data extracted and decoded from the pointer position radar. For each frame in the pointer positioning radar, the python script will denote the number of points in the frame and the frame number. Each point outputs an X, Y, and Z position that is passed forward to the mouse positioning subsystem. Additionally, this validates the radar's angle and distance detection, as the hand is successfully detected when it is within ~60 degrees and ~1.5 feet of the radar's antennas, and not detected otherwise.

Figure 4: Decoded Pointer Position Data

It is worth noting that a handful of changes were implemented to this subsystem. Initially being provided with 5 TI hardware boards, the project was successfully downsized to operate on a single radar device and is visualized in Figure 6. Additionally, a 3D printed case and mount were implemented to secure the device on the user's screen and can be seen in Figure 7. At last a GUI (graphical user interface) was implemented to enable the user to easily modify virtual mouse settings and can be seen in Figure 5.

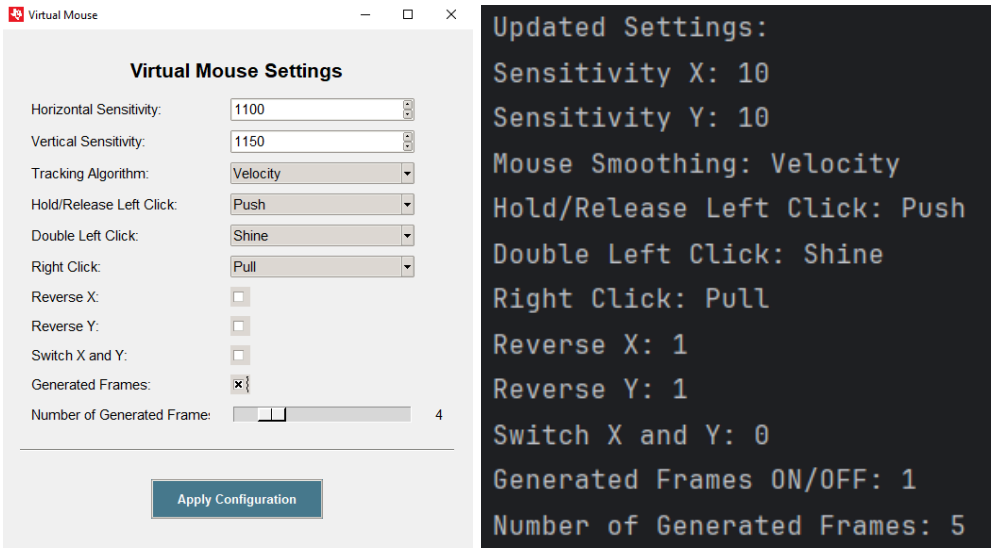


Figure 5: GUI Terminal and Terminal Response

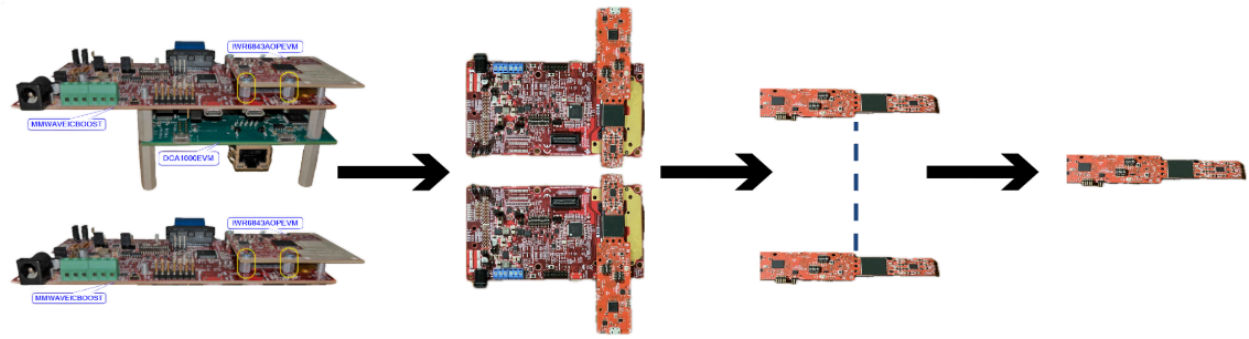


Figure 6: Downsizing to single radar device

Figure 7: Radar Case and Mount

2.4. Subsystem Conclusion

The chirp configuration and data collection subsystem performs as expected and passes all validation conditions. The radar device successfully runs in parallel and communicates with the computer, only detecting the user's hands in a close proximity to the devices, and the data seen from the device is properly decoded and passed onwards. The data is readily available to use for enabling the functionality of the virtual mouse.

3. Mouse Positioning Subsystem Report (Daniel Lu)

3.1. Subsystem Introduction

The chirp configuration demo will generate the coordinates of the target's points in the 3D plane relative to the radar as output data. The mouse positioning subsystem is designed to analyze the output data of the chirp configuration demo and generate a digitalized moving path for the virtual mouse with two algorithms, position tracking and average weight velocity.

3.2. Subsystem Details

A visual block diagram of the subsystem is displayed below.

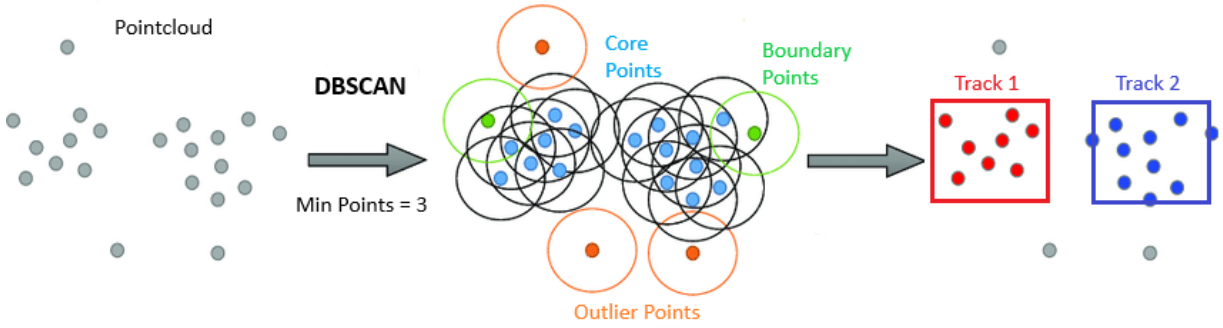


Figure 8: Data points selecting process

The function of the mouse positioning program is to give a moving command to the mouse on the computer to move according to the relative position of the target object. The chirp configuration program collects the point cloud, which is the reflected signal received by the radar in the 3D plane. In the chirp configuration program, the point cloud consists of five to ten points, representing the relative position of the target object in the 3D plane. The chirp configuration program generates point clouds for every frame for the moving object. The time difference between two continuous frames is modified by the chirp configuration program.

The position tracking algorithm generates a moving path for the mouse according to the target object's position relative to the center axis of the radar instead of the moving path of the target. The position tracking program generates a limit region, usually a square area above the radar, where the program converts the position of the target object inside the limited region and converts the relative position of the object according to the center axis of the radar in the 3D plane to the actual position on the computer screen.

The average weight velocity program gives a moving command to the mouse on the computer according to the relative movement of the target object. The program performs geometric average calculation to determine the center of the point cloud provided by the chirp configuration program. Each frame of the radar generates a point cloud with the

calculated geometric average center. The moving vector from the last frame to the next frame is the relative movement of the mouse according to the moving target object.

When the target object is moving on the horizontal axis, the moving vector calculated by using the geometric center of the point cloud has an unremovable error component in the vertical axis, similar to a sine waveform, which is caused by the imperfect performance of the radar, not the tracking program. The function of average weight velocity is to remove the error component from the moving component, calculated by taking the average of five past frame's moving vectors according to their "time weight." The closer the past frame is to the current frame, the larger the weight it takes in the average weight velocity calculation. Mathematically, the error component in the target axis is removed, where the positive error component is canceled with the negative error component. A sample horizontal movement of the target object detected by the radar is shown below.

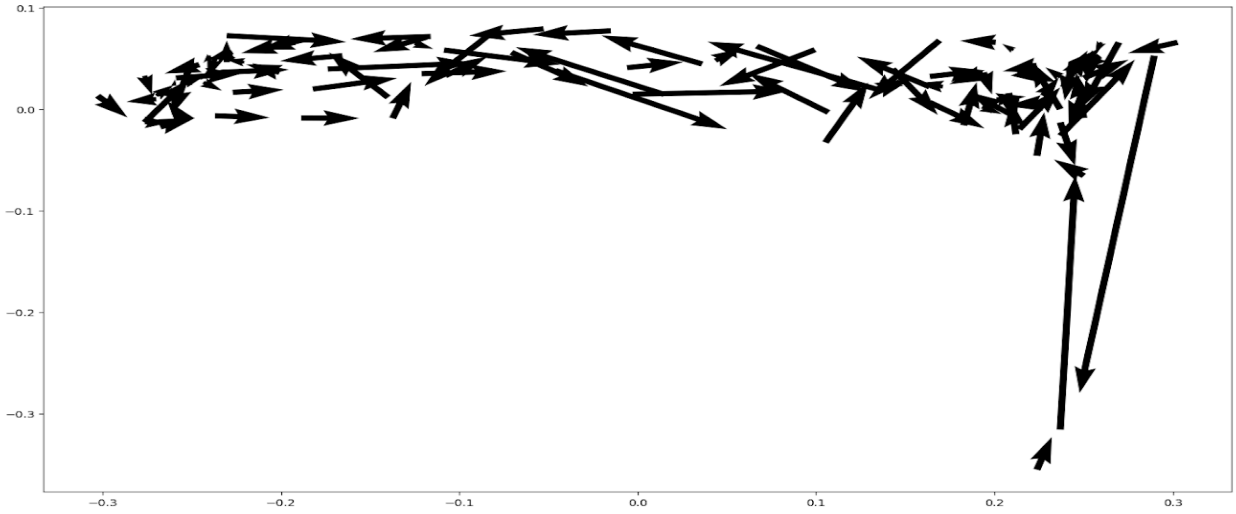


Figure 9: Sample moving vector in x-y axis

3.3. Subsystem Validation

- **Functional point selection**

  Does the subsystem select all the points representing the target(hand), and deselect all the data points representing other objects?

- **Moving path validation**

  Does the program generate a moving path with high accuracy of magnitude and direction according to the axis of the radar?

Table 1: Validation For Position Tracking System

3.2.1.4 3.2.1.6	Gesture positioning	The coordinates of the hand should be calculated when deleting the error points.	Check position calculations in terminal and compare them to actual hand position data	Passed	Daniel
3.2.1.6 3.2.1.8 3.2.1.11 3.2.3.2.1	Velocity accuracy(redacted)	The output velocity of the moving hand should be close to a multiple of its true velocity.	Using a phone to record hand movements, measure the hands true speed and the velocity calculated by the algorithm	Passed	Daniel
3.2.1.4 3.2.1.6	Position tracking program accuracy	In the position tracking program, the mouse should move to the relative position of the hand according to the scale of the computer screen.	Perform horizontal movement, vertical movement, and circle movement using your hand. Compare the moving path of the mouse with the movement of the hand.	Passed	Daniel

The validation accuracy is directly tested by executing the position tracking system by performing multiple direction and multiple speed movements. The performance of the movement can be modified in the program by changing the setting of sensitivity, which is how much the cursor on your screen moves in response to a physical movement of the mouse and the target object.

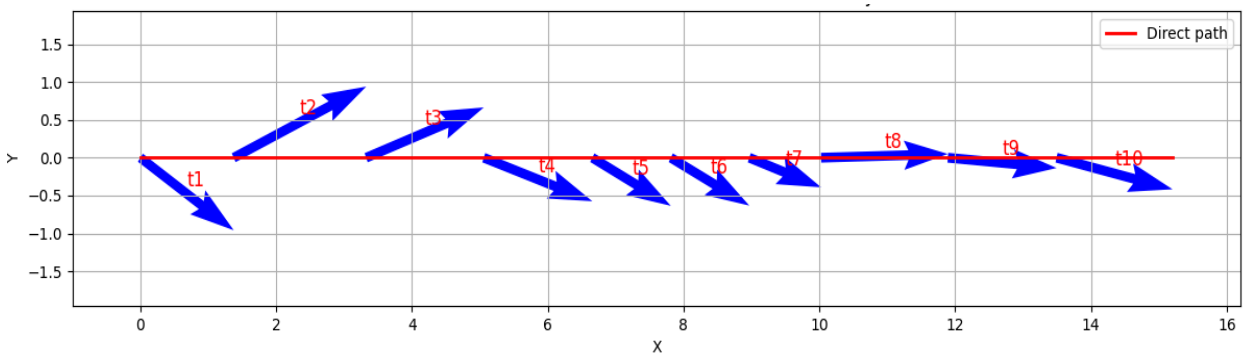


Figure 10: Sample moving velocity vector of Average Weight Velocity

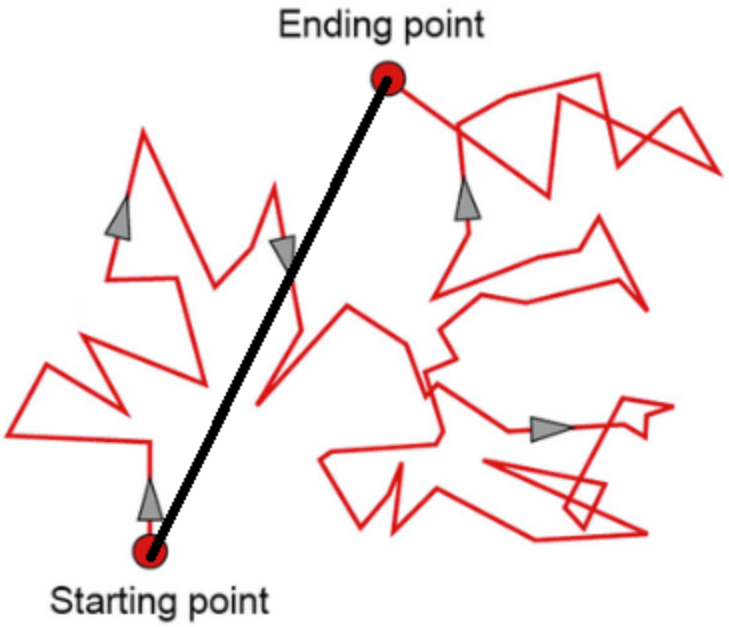


Figure 11: Sample moving velocity vector of Position Tracking Program

In the average weight velocity tracking program, the movement depends on calculating the average velocities in multi-frames. In the position tracking program, the movement of the mouse depends on the position instead of the moving path of the target object.

3.4. Subsystem Conclusion

The Positioning Subsystem demo works with high accuracy in the validation plan. The final demo can be directly used to generate a moving path of the mouse, working with Chirp Configuration Demo, and the Input Mapping Subsystem Demo. However, the movement error movement of the mouse due to the imperfect performance of the radar cannot be fixed by the positioning tracking program. The average weight velocity program removes the movement error by taking an average velocity of multi-frames to cancel the error. The position-tracking program ignored the error movement by only tracking the relative location of the target object. In conclusion, both programs generate mouse movement with high accuracy.

4. Gesture Recognition Subsystem Report (Oscar Chavez Araiza)

4.1. Subsystem Introduction

The Gesture Recognition Subsystem is in charge of detecting hand gestures captured by the radar. The subsystem was responsible for creating an ML model to take in feature data extracted from the radar, and predict the gesture based on this data. The subsystem was required to create its own training set, and select the gestures that would be used for the Virtual Mouse. The subsystem was also tasked to create a script to obtain real time predictions using the model.

4.2. Subsystem Changes

There were multiple changes made to the gesture recognition subsystem, but there were 2 main changes. The main changes were using one radar instead of two and the number of gestures identified.

The first was that during the Fall semester the system used two radars flashed with different programs, now the system only uses one radar. The radar was flashed with the program used for tracking the hand, so I had to design the features I wanted to use to train the new model. This also meant that the model would need to be retrained every time the configuration file was changed. This was done to maintain a high accuracy rate, but reduced the amount of data the model would be trained on.

The second main change was the introduction of two new gestures, pull and shake. This change was minimal and only meant that more data was going to be collected for the model.

4.3. Subsystem Details

The radar is able to output frames containing 5 features that can be used to predict the hand gesture. Since 1 frame does not give enough information to make a prediction, the subsystem used a CNN that would take as input 30 frames of data from the radar. The CNN was trained using 3 convolutional layers and 3 max-pooling layers. The kernel size used was 3x3 to detect meaningful features in the 40 frames. Over 4000 gesture samples were used to train and validate the model.

```
Epoch 1/10
105/105 [==============================] - 2s 7ms/step - loss: 0.5560 - accuracy: 0.8048 - val_loss: 0.1788 - val_accuracy: 0.9441
Epoch 2/10
105/105 [==============================] - 0s 5ms/step - loss: 0.1368 - accuracy: 0.9595 - val_loss: 0.0826 - val_accuracy: 0.9738
Epoch 3/10
105/105 [==============================] - 1s 5ms/step - loss: 0.0703 - accuracy: 0.9792 - val_loss: 0.0445 - val_accuracy: 0.9869
Epoch 4/10
105/105 [==============================] - 0s 5ms/step - loss: 0.0555 - accuracy: 0.9812 - val_loss: 0.1014 - val_accuracy: 0.9655
Epoch 5/10
105/105 [==============================] - 1s 5ms/step - loss: 0.0323 - accuracy: 0.9905 - val_loss: 0.0405 - val_accuracy: 0.9845
Epoch 6/10
105/105 [==============================] - 1s 5ms/step - loss: 0.0163 - accuracy: 0.9961 - val_loss: 0.0524 - val_accuracy: 0.9845
Epoch 7/10
105/105 [==============================] - 1s 5ms/step - loss: 0.0185 - accuracy: 0.9955 - val_loss: 0.0214 - val_accuracy: 0.9917
Epoch 8/10
105/105 [==============================] - 0s 5ms/step - loss: 0.0153 - accuracy: 0.9958 - val_loss: 0.0193 - val_accuracy: 0.9941
Epoch 9/10
105/105 [==============================] - 0s 4ms/step - loss: 0.0093 - accuracy: 0.9976 - val_loss: 0.0212 - val_accuracy: 0.9881
Epoch 10/10
105/105 [==============================] - 0s 5ms/step - loss: 0.0089 - accuracy: 0.9985 - val_loss: 0.0113 - val_accuracy: 0.9952
```

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 28, 32)	512
max_pooling1d (MaxPooling1D)	(None, 14, 32)	0
conv1d_1 (Conv1D)	(None, 12, 64)	6208
max_pooling1d_1 (MaxPooling1D)	(None, 6, 64)	0
conv1d_2 (Conv1D)	(None, 4, 128)	24704
max_pooling1d_2 (MaxPooling1D)	(None, 4, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 64)	32832
dense_1 (Dense)	(None, 5)	325

Total params: 64581 (252.27 KB)
Trainable params: 64581 (252.27 KB)
Non-trainable params: 0 (0.00 Byte)

Figure 12: CNN accuracy & layout

The model was trained to recognize 4 gestures, “push”, “pull”, “shine”, “shake”. These four gestures were the best choice for the subsystem since they require minimal movement and won’t interfere with the tracking subsystem. The gestures are also very simple which makes training a model for them easier.

Figure 13: Gestures

Training the model was done mostly from data captured using my hand. The model was also trained using data from other team members and some family members. It would be ideal to capture more data to make the model more reliable. The model was trained using Jupyter Notebook, while the script to use the model to produce an output was made using a python IDE, Visual Studio Code.

4.4. Subsystem Validation

Table 2: Validation For Gesture Recognition Subsystem

Section	Task	Specification	Test	Result	Owner
3.2.1.1	False Positives	ML model should have less than 1 false positive per minute	Ran the model and recorded its performance using my hand as a target.	Passed	Oscar
3.2.1.5	Hand Gesture Recognition	ML model can recognize the user's hand gestures.	Ask someone with no prior experience with the program to try to imitate the gestures. The results will be recorded.	Passed	Oscar
3.2.1.5	Gesture Recognition Accuracy	ML model can correctly identify and categorize hand gestures at least 90% of the time.	Using the results from the testing with an outside party, calculate the accuracy of the model.	Passed	Oscar

3.2.1.5	Hand Size	Radar will recognize gestures regardless of hand size.	Asked multiple people with varying hand sizes to test the gesture recognition.	Passed	Oscar

Table 3: Detection Statistics

Test Accuracy	~99%
Real Time Accuracy	~92%
Time Between False Positives	~70sec
Max Detection Distance	~1½ ft
Area of Detection at 1ft	~1½ ft x 2 ft
Hand-Size Sensitivity	~None

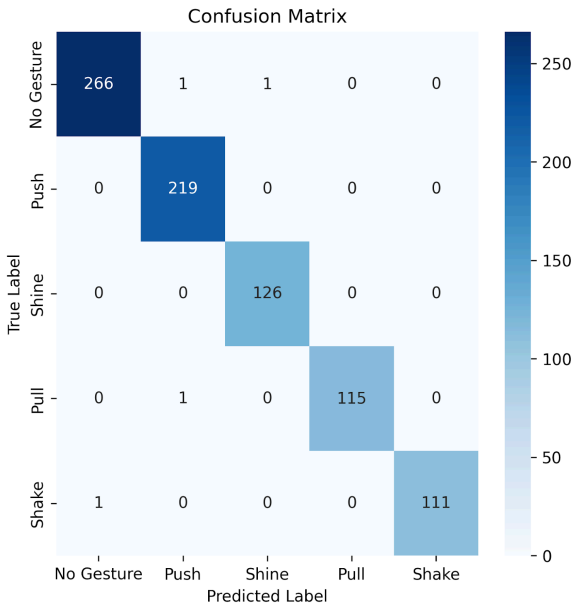
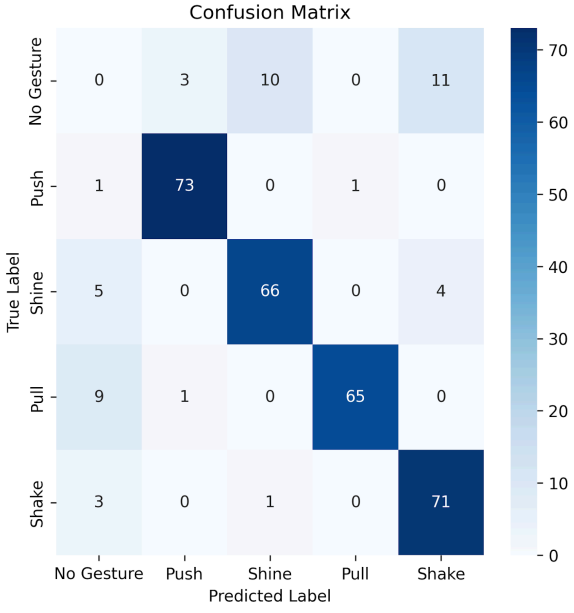


Figure 14: Test Confusion Matrix



**top row was used to calculate false positive rate

Figure 15: Real Time Confusion Matrix

Hand Size

To show that the model works no matter what the size of the user's hand is, I asked people with varying hand sizes to test the model while the full program was running.

4.5. Subsystem Conclusion

The subsystem performed as expected and the final product achieved all expectations set by the FSR. Although the testing showed that the subsystem worked well above expectations, it should be noted that the false positive rate could be improved drastically. The main gestures that are being falsely detected are shine, and shake. These two gestures have very specific data patterns that the model could learn better from an increased data pool. In conclusion, the gesture recognition model can be reliably used to identify the four hand gestures it was trained to detect.

5. Input Mapping Subsystem Report (Zane Meikle)

5.1. Subsystem Introduction

The input mapping subsystem is designed to take the data from the gesture recognition subsystem and mouse positioning subsystem and convert that into actual mouse movements and mouse actions.

5.2. Subsystem Details

To move the mouse, the subsystem will take a movement vector from the mouse positioning subsystem and then will feed that into the pynput move command to move the mouse. For mouse actions, the subsystem will take a value of 0, 1, or 2 from the gesture recognition subsystem for right and left click. 0 being a no action state, 1 being for a mouse release, and 2 being for a mouse press. There are also several variables added for convenience in debugging, such as Redresh_Rate, Reverse_X, Reverse_Y, Switch_XY, and Scale. Refresh_Rate sets the refresh rate for the system, Reverse_X and Reverse_Y flip the direction of either X or Y, Switch_XY will make X equal to the Y data and vice versa, and Scale will be multiplied to the X and Y vectors. Along with this, this semester a mouse smoothing function was added that will take a variable from the GUI and will interpolate that many times between frames to give a smoother appearance to the mouse movement.

5.3. Subsystem Validation

To test the subsystem, a test program was run that moved the mouse to each corner of the screen and then left clicked, and right clicked. As well, to validate error handling, incorrect values were put into the variables to confirm an error message would appear for each setting, as well two test programs would run that would throw an error for an invalid left or right click input (anything that isn’t 0, 1, or 2).

As seen below, the subsystem passed the first validation with the mouse moving to every corner of the screen (only one screenshot of the mouse moving to save space) and then left and right clicking.

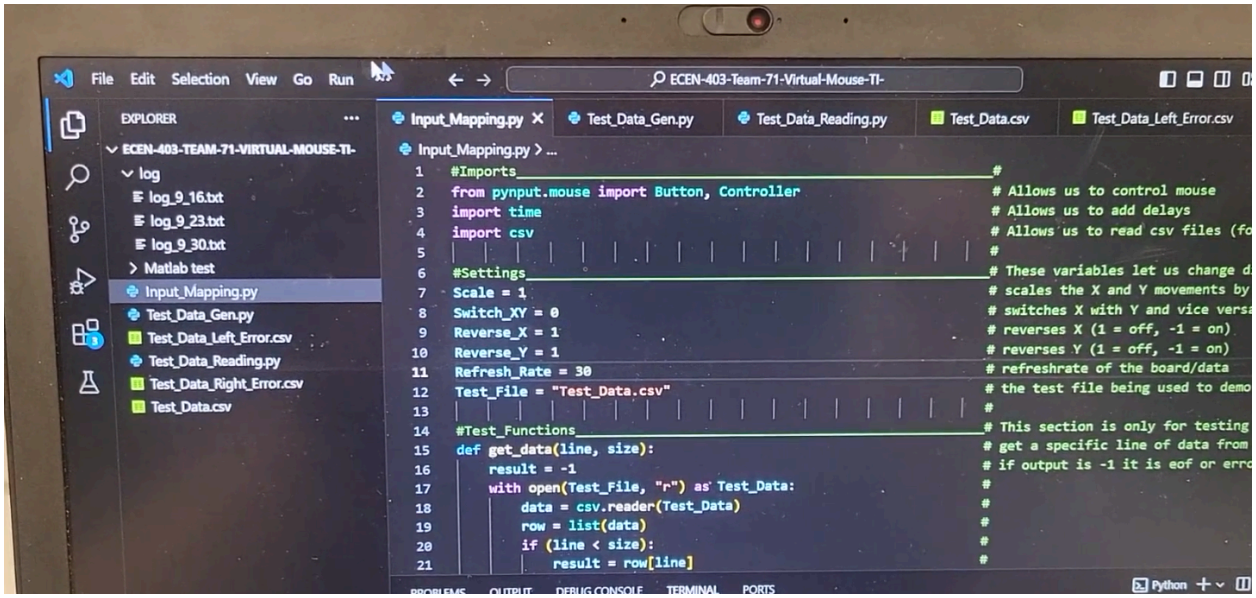


Figure 16: Mouse Moving

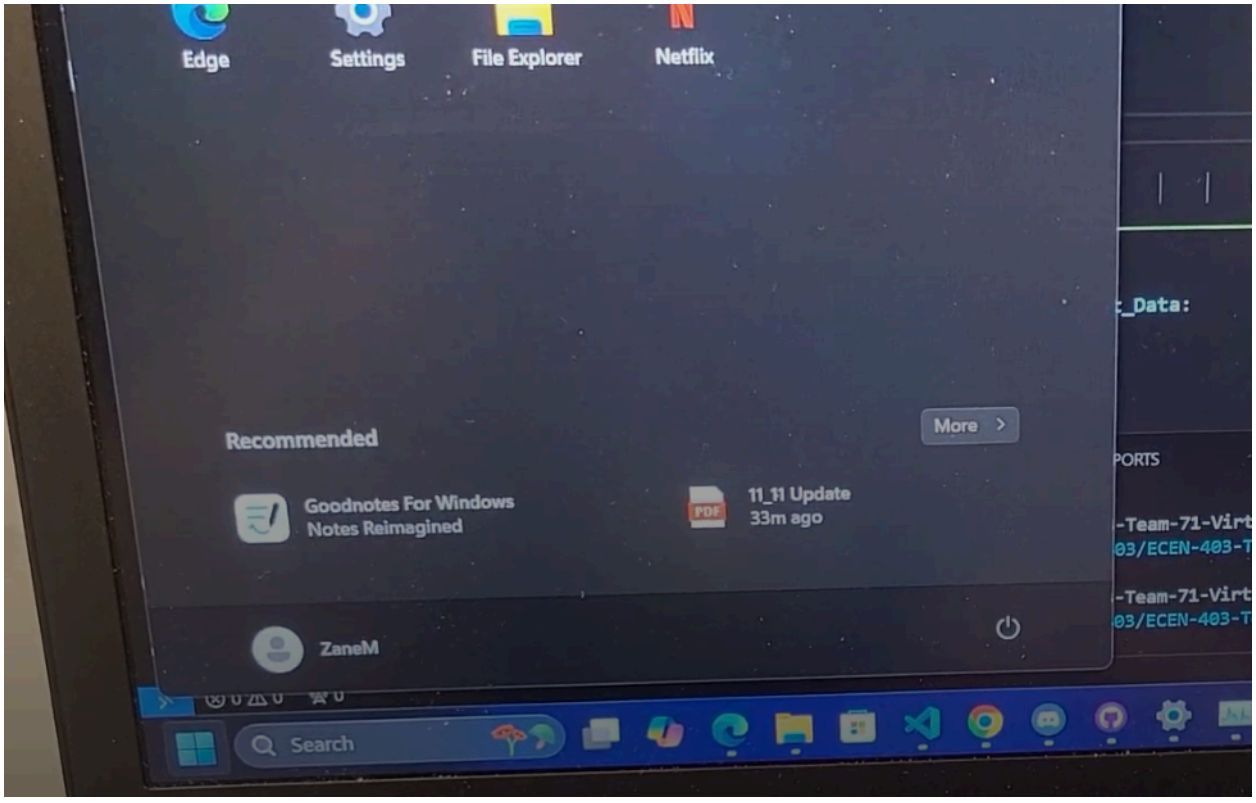


Figure 17: Mouse Left Clicking Opening Windows Start Button

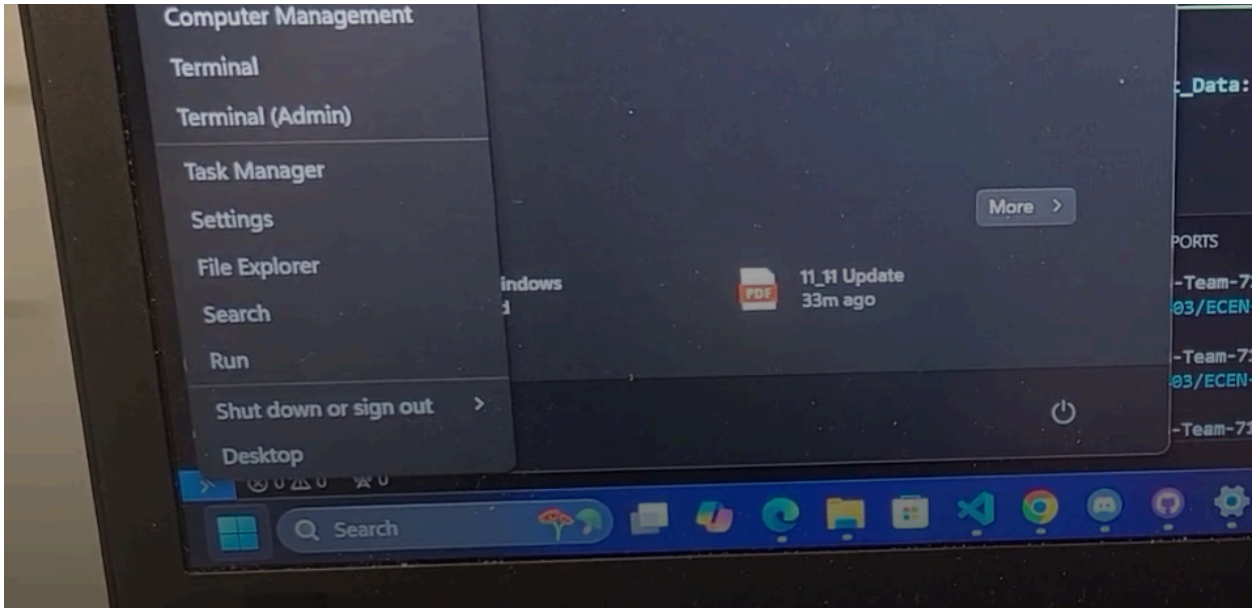


Figure 18: Mouse Right Clicking Opening Options

The error handling also passed with the program throwing an error whenever there was an invalid input.

```
2/python.exe "c:/Users/ZaneM/Documents/ECEN
ERROR: Refresh_Rate must be greater than 0.
ERROR: Reverse_X must be 1 or -1.
ERROR: Reverse_Y must be 1 or -1.
ERROR: Switch_XY must be 0 or 1.
ERROR: Scale must be an int.
PS C:\Users\ZaneM\Documents\ECEN 403\ECEN-4
```

Figure 19: Invalid Variables

```
2/python.exe "c:/Users/ZaneM/Documents/ECEN 4
ERROR: invalid left click input: -12312
PS C:\Users\ZaneM\Documents\ECEN 403\ECEN-403
```

Figure 20: Invalid Left Click

```
2/python.exe "c:/Users/ZaneM/Documents/ECE
ERROR: invalid right click input: 3123
PS C:\Users\ZaneM\Documents\ECEN 403\ECEN-
```

Figure 21: Invalid Right Click

As for testing the mouse smoothing function, the function was tested with the existing mouse movement test code. Below is an image from that test.

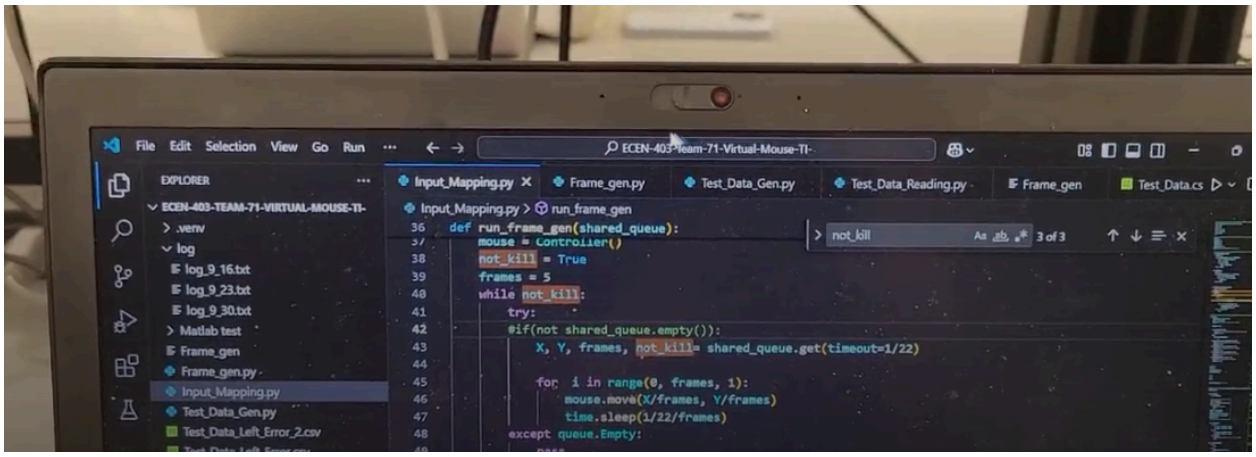


Figure 22: Mouse Smoothing Function Moving the Mouse

As well, to ensure that it wouldn't create a zombie process, the terminal was checked to confirm that the process was over. The terminal would continue to run if the process didn't die properly.

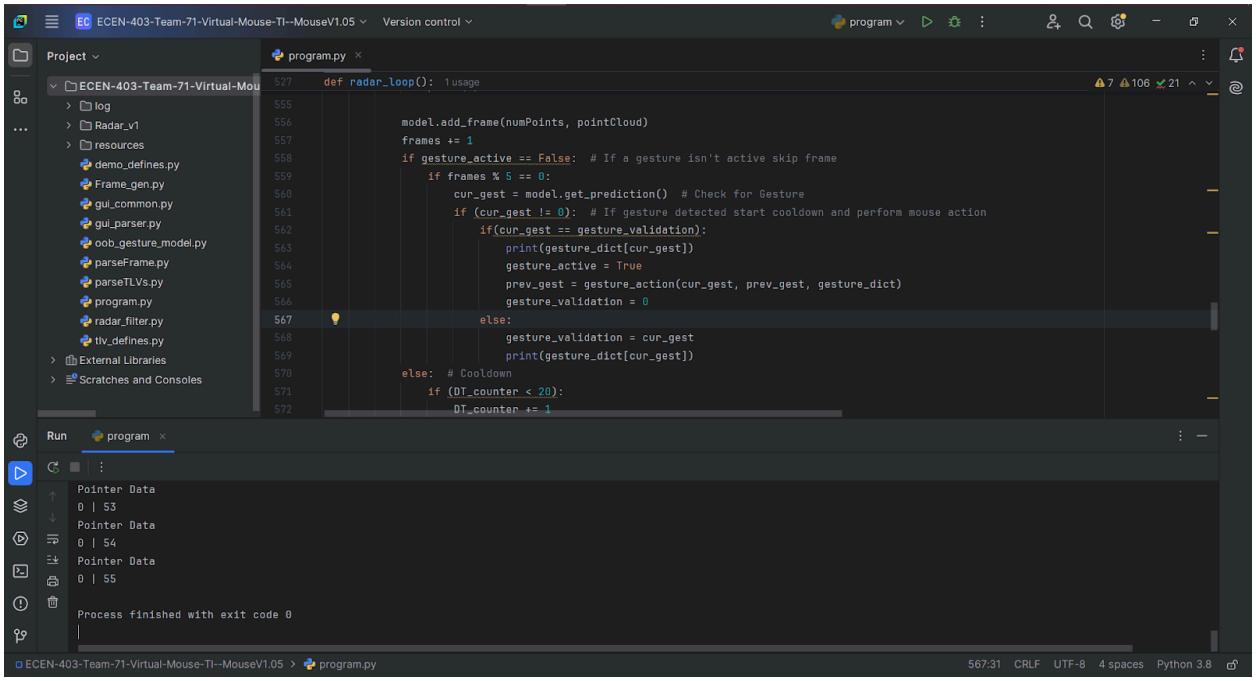


Figure 23: Terminal Ending the Program Properly Without a Zombie Process

5.4. Subsystem Conclusion

As seen in the validation, the subsystem is functioning properly, with it being able to move the mouse, left/right click, and generate frames for smoothing. Since last semester, with parallelization the process runs much smoother without issues due to the optimizations that were down by parallelizing everything.

Virtual Computer Mouse using mmWave Radar (TI)

Oscar Chavez Araiza
Greyson Heath
Daniel Lu
Zane Meikle

System Report

REVISION – 1.0
28 April 2025

SYSTEM REPORT
FOR
Virtual Computer Mouse using mmWave Radar (TI)

TEAM <71>

APPROVED BY:

_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	04/28/2025	Zane Meikle		Revision 1.0

Table of Contents

Table of Contents **83**
List of Tables **84**
List of Figures **85**
1. Overview **86**
2. Development Plan & Execution **87**
2.1. Design Plan....87
2.2. Execution....88
2.3. Validation Plan....89
3. Conclusion **91**
3.1. Sponsor Feedback....91
3.2. Future Work....91

List of Tables

List of Figures

Figure 1: Virtual Mouse System User Experience.....86
Figure 2: Original design for the system.....87
Figure 3: New system design.....87
Figure 4: Execution Plan Diagram.....88
Figure 5: GUI Window.....89
Figure 6: Invalid settings message.....90

1. Overview

The company sponsor, Nick Farrel with Texas Instruments, requested for a virtual mouse to be incorporated using TI's mmWave radar products with machine learning models. This conveyed building a system where a user can control a computer mouse movement and actions using a radar, as compared to the traditional method of using computer vision via a camera. The culmination of this system is represented in Figure 1 where the mmWave radar device is positioned on the top of the computer screen, and the user is playing a game by controlling the movement of the mouse to further manipulate the movement of the snake. By moving your hand in a viewable region for the sensor (located at the top of the computer screen), one is able to successfully move the computer mouse and perform mouse actions using various gestures.

Figure 1: Virtual Mouse System User Experience

The creation of this system involved four core subsystems: chirp optimization and data collection, mouse positioning, gesture recognition, and input mapping to the computer. The chirp optimization and data collection properly configures the radar and decodes the data for future subsystems. The mouse positioning and gesture recognition perform a multitude of calculations and utilize machine learning models to calculate the precise movements and actions. The input mapping takes the movements and maps them to the respective computer screen. The purpose of this system report is to review the key decisions and milestones of the entire development process.

2. Development Plan & Execution

2.1. Design Plan

The original plan for this system was to have 2 radars that feed separate streams of data to the mouse positioning and gesture recognition subsystems. This original design can be seen in the figure below.

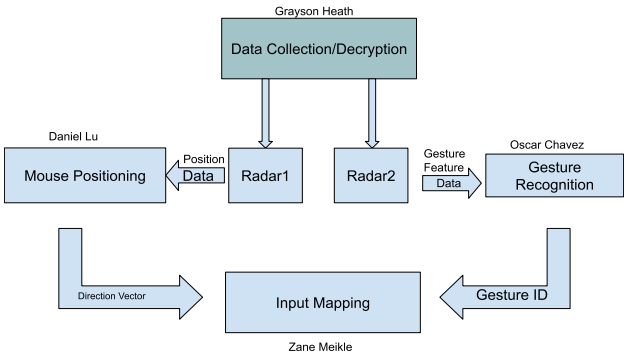


Figure 2: Original design for the system

Integrating the subsystems using 2 radar devices proved to be very difficult due to a few factors: the sensors on both radar devices would collectively interfere with one another due to being positioned similarly, and collecting and decoding data from both radars consecutively caused the computer to run slower as compared to collecting data from a single radar. These difficulties motivated our team to reduce operations to a single radar device, primarily to preserve data and reduce overall processing speed. As well, our team developed more functions to the software to improve the function of the virtual mouse. We added a mouse smoothing function that would generate interpolated frames of movement, more algorithms for processing the data, and more gestures for mouse actions.

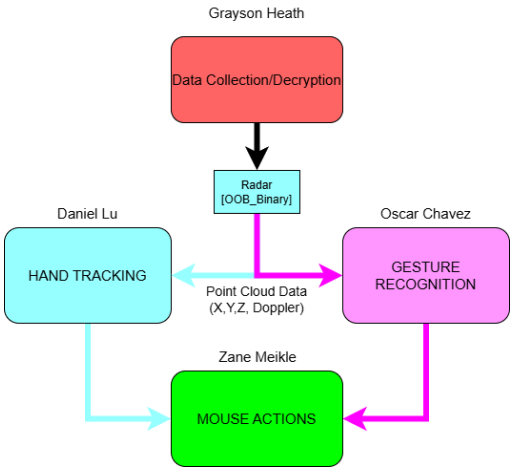


Figure 3: New system design

2.2. Execution

The initial part of the development process began with integrating the individual subsystems that were completed in the Fall Semester. It was at this point we made our revelations as described in the Design Plan. We put together the execution plan that is shown in the figure below. The first step was to make sure that the subsystems from the Fall Semester could be integrated correctly into an initial test product. Once we were able to successfully implement the initial product, each subsystem moved on to making adjustments and improvements.

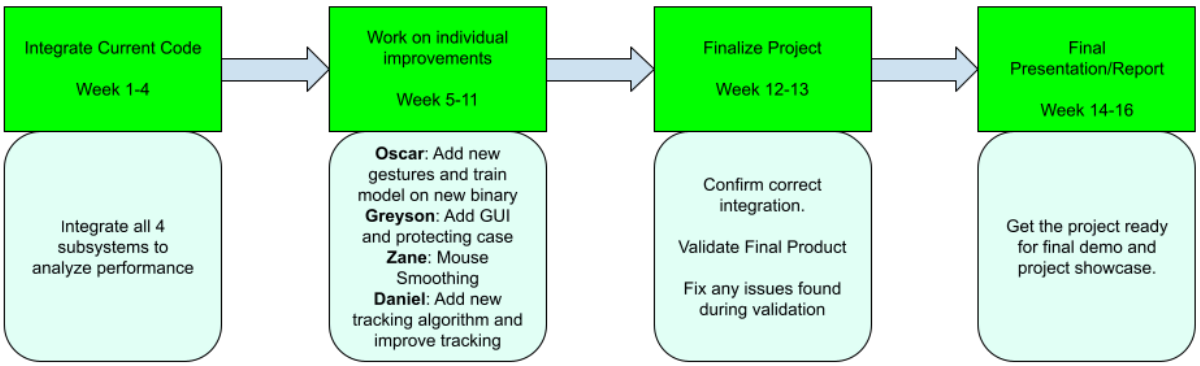


Figure 4: Execution Plan Diagram

For the new gestures and retraining the model on the new binary, the gesture recognition model had to be retrained on the binary used to track the hand movement. Two additional gestures were added once the model was redesigned and retrained for the new binary. There is usually a way to change multiple settings when using a mouse, and to replicate that feature a GUI was added to the product. The GUI would allow users to change the sensitivity of the mouse, change what each gesture represents which mouse action, and it could also invert the X-Y axis. Additionally the GUI would implement a way to use frame generation which would be used to smooth out the movement of the mouse. The mouse movement was incredibly tricky to implement, so two different algorithms were used for different settings. The project implemented a position tracker for a faster and less precise use of the radar. It also implemented a velocity tracker for a slower, more precise use. All the features previously mentioned were implemented and improved during weeks 5-11.

Once the improvements to each subsystem were finalized, the full implementation of the program was tested. Once the subsystems were implemented, tests were made to ensure correct implementation. Although each subsystem was tested in standalone mode, tests were made to validate the correct functionality of each system requirement while using the full program. A few errors were found on the way the subsystems interacted with each other and promptly fixed each issue as it arose.

After the final product was finalized, the team focused on making the project work best for the final demonstration and project showcase. The team made sure to improve the project to make it more intuitive. This means that default GUI settings were chosen for intuitive use.

2.3. Validation Plan

This section will focus on the validation of the full operational system, validation of subsystem specific functionality can be accessed in the *subsystem report* above. Validation of our system began by proving it operates as it should. To start, when the python script starts, a GUI window will pop up, as seen in Figure 5. From that GUI window, all of the functions that need to be validated can be selected and tested.

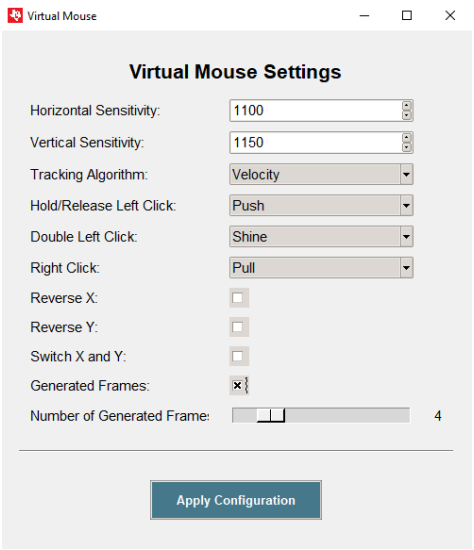


Figure 5: GUI Window

The first test performed was to see if the GUI program called upon the radar program. This was tested by starting the GUI and checking if the cursor responded to the movement of the hand when the GUI was open. This test also demonstrated that the radar was correctly communicating with the program.

The second test performed was the use of each tracking algorithm, since the default algorithm is “velocity” we tested it first by checking the mouse response to hand movement. We saw that the hand moved slower which is unique to the velocity tracking algorithm. We moved on and selected the “Position” algorithm using the GUI. Once the configuration was applied, more erratic movements emulating the movements of the hand were observed. This was descriptive of the position tracking algorithm.

The third test performed was to see the functionality of the sensitivity. The sensitivity only affects the velocity tracker. Once we changed the sensitivities to double it was observed that the cursor movements were larger which was expected of a higher sensitivity.

The fourth test performed was to see if the GUI could control what each gesture did. There are four different gestures, but only 3 can be changed. This was done because shake interferes with mouse movement, so shake will only scroll down the screen. You can also set each mouse action to “NONE” which means that that action will not be performed. The test was made in the following manner. Only one mouse action is mapped to a gesture. The gesture is verified to do the action and this is repeated for each gesture.

The fifth test performed was to test “Reverse X” and “Reverse Y”. To do this we turned on the setting “Reverse X” and began moving using a virtual mouse showing that the movement in the horizontal direction had been reversed. Then the “Reverse Y” setting was applied showing that movement in the vertical direction was reversed.

The sixth test performed was to test the “Switch X and Y” setting. To do so, the “Switch X and Y” setting was applied, this made moving right move the mouse up, moving left move the mouse down, moving up moves the mouse right, and moving down moves the mouse left.

The seventh test was to test the “Generate Frames” setting. To do this the “Generate Frames” setting was apple and the “Number of Generated Frames” is set to 1. The mouse is moved with the user's hand to show the mouse moves normally. Then the “Number of Generated Frames” is set to 2 and the user moves their hand again to show that the movement is now a lot smoother.

The final test was to check if the program handled the input of incorrect settings correctly. Since the only incorrect input for the settings would be to set two mouse actions to the same gesture, the test was performed by having multiple combinations of gestures in which at least 2 actions were mapped to the same gesture. The result of one of the tests can be seen in **Figure 6.**

**Invalid settings message on line 1 followed by input settings. The last line shows the actual state of the gesture dictionary.
0 = no gesture, 1 = push, 2 = shine, 3 = pull, 4 = shake

Figure 6: Invalid settings message

3. Conclusion

The final product is a functional and feature rich proof of concept showing the capabilities of Texas Instruments IWR6843AOPEVM. This project ran into many unexpected challenges that resulted in a change of course. Some of the biggest being the initial integration where we realized having two radars will cause issues where they would interfere with each other. This caused massive changes to the gesture recognition subsystem where the AI model had to be retrained. Though despite the hardship, the final product has met or exceeded the sponsors expectations.

3.1. Sponsor Feedback

After Nick Farrel, the project sponsor, was able to test the radar mouse locally. He was impressed with the functionality of the radar. He mentioned that it was a good implementation given the binary that was used to obtain data from the radar. He suggested using different binaries to potentially increase the amount of data obtained from the radar.

> “[I am] happy with your work and impressed with the final state of the project” - *Nick Farrel, TI*

3.2. Future Work

Although the gesture recognition was able to reach an accuracy of 92%, tracking proved to be challenging. The non constant output of the radar made the tracking of the hand challenging. In order to improve upon our design, the following suggestions are made:

- Custom\Dedicated Binary: A custom binary would exponentially increase the accuracy and value of the data obtained from the radar.
- Larger Data Set: Using a larger data set to train the model would increase the overall accuracy. **overfitting the model could become an issue.