Programming Assignment
CS 708, Term 2 2024-2025 [20 Marks]
Due: **April 25,** 2025 11.59pm SGT

A. Question

Spatial awareness in multimodal agents, deployed on mobile or wearable devices, enables a wide range of innovative applications. For example, in immersive furniture shopping, an agent capable of spatial reasoning can assist customers in virtually refurnishing their homes by interacting with real objects in their surroundings. Similarly, in collaborative assistive scenarios, such a smartglass-embedded agent can help users identify and interact with physical objects using verbal and gestural cues. For instance, a user might point to a nut and ask for a wrench, prompting the agent to resolve the nut's spatial dimensions to determine the appropriately sized wrench required for the task.

These types of applications are now feasible due to the development of wearables equipped with advanced sensors (e.g., microphones, 2D cameras, and LiDAR), as well as edge servers with significantly higher processing capabilities. Furthermore, advances in artificial intelligence (AI) and deep neural networks (DNNs) have enabled such devices to possess “machine intelligence”—the ability to recognize objects, interpret verbal commands, and perceive human intentions in a manner similar to actual human cognition.

This programming assignment aims to demonstrate some of these capabilities and provide a lightweight, simplified preview on how advanced sensing technologies, combined with novel AI techniques, can pave the way for innovative applications. The central idea revolves around leveraging a mobile application as an AI-based agent capable of interpreting moderately complex natural human instructions with spatial awareness. To create a basic prototype of such an assistant application, you are tasked with implementing an app that integrates (or emulates the integration of) scene images from a camera, depth data from a LiDAR camera, the user’s verbal/text commands, , and the user’s right-hand pointing gesture as captured by both types of cameras. The pipeline should calculate spatial dimensions (e.g., height, width) and render a bounding box, for an appropriately identified object, along with a label indicating the dimension on a scene image that represents a typical workplace or home environment.

At a high level, the idea is for the user to first browse and choose an environment “instance” (e.g., living area, lab setting) from a given database, which consists of multiple environments and sensory inputs, including **a 2D image, depth image, and a Homogeneous Coordinate Projection matrix**. A vision-based DNN is then applied to detect/segment and classify various objects within this physical space. Beyond simply capturing the environment , the instance is also likely to include a **pointing gesture** made with the user’s hand, which is captured by the smartphone camera and lidar sensor. Additionally, the user may enter a concise explicit instruction into a text input field on the smartphone to accomplish a specific task (e.g. “calculate width of that chair”) within the given environment.

For example, in the environment instance depicted in **Figure 1**, the task is to determine the width of a chair labeled as the **target Furniture** and display its bounding box along with width on the image of the environment on the phone display. As a first step, the code written for this novel app will need to process **three inputs**—the **visual scene**, the **pointing gesture**, and the **verbal command**—to identify the specific object being referenced (which we call the referent object) within the camera-captured image and appropriately draw a bounding box around it.

The entire application comprises a Java (or Kotlin) portion that runs on an Android mobile device, as well as a Python API back end. (Because of issues related to variability across devices (and because not all of you may have an Android mobile device), the application will be developed and tested using an Android emulator.) While this may sound complicated, the programming effort required by you will be considerably simplified by the following steps:

(a) Typically, NLP models allow the processing of long and semantically complex sentences, which can be used to handle the input text (the instruction). However, for the purposes of this project, we simplify the task to a word-tagging problem to identify the target furniture class. An example input text instruction style is defined later in the write-up

(b) Most of the machine intelligence steps (specifically, those related to identifying objects and calculating their dimensions in the camera image, as well as deciphering the pointing location associated with the gesture) will be performed on the server, with the phone merely acting as a relay to transfer the corresponding sensor data to the server. This will minimize your need to write complex Android code.

(c) For several of the more complicated parts (uploading the multi-modal, instance-specific, sensor data to the server and receiving the bounding box and width of the target object from the server), you will use standard Android or python libraries, with well-defined APIs. We will provide you these libraries and APIs.

(d) The type of scene and the range of verbal commands to be analyzed will be limited to a very small set. This restriction will help manage the workload and allow for easier testing of the code with relatively little effort.

Your extensions to the base application (that you'll provide) will consist of several distinct tasks/components:

Detection and dimension resolution of the selected object: The extension should identify objects in the scene according to the task described for each environment instance by integrating **instance segmentation vision capabilities** and incorporating cues from an accompanying verbal command (input as text). If a **pointing gesture** is available, the module must combine **visual information, textual clues from the instruction, and the pointing direction** derived from the depth image (where the pointing direction is calculated) to determine the specific object in the scene that the user is selecting. For example, given a scene containing **two similar chairs located near a table**, as depicted in **Figure 1**, if the user points towards the **left-hand side chair**, the extension should consider the chair on the left side as the target or referent object. Even though **Figure 1** depicts identical chairs, the **chair type** may vary in other environments. Finally, the extension should compute the dimension of the referent object.

Rendering of bounding box/dimension on 2D image. The extension should use the results of the computation output of previous step to overlay (i.e., render) a **2D bounding box and dimension** on the original scene's **2D image**.

The envisioned usage of the final app is as follows:

- The user toggles the arrow buttons (right arrow or left arrow) to select an environment from a given database on the phone for further processing. The database is stored on device itself, but the App will have a mechanism to select a different database as needed (e.g., for testing)
- The selected environment's 2D image is displayed on the display. Each environment instance is given a predefined target (i.e, a referent object), with the objective being to identify this furniture object, compute its width and display the width (together with a bounding box) on the display. The image may or may not contain a person’s hand (right hand) pointing to the target (e.g., a table).
- The user enters an instruction as text, such as "width of that table," based on the selected environment and presses the "Send" button to send the data over the network to a server. This action transmits the 2D image, depth image, coordinate projection matrix (for 2d to 3D conversion), and command text to the server for further processing.
- After the server API returns the resolved target furniture's width along with the bounding box coordinates (on the 2D image) for that object, the Android app displays the furniture's width as text and visually draws the bounding box on the image around the furniture.
- Finally, the user can click one of two buttons (left arrow or right arrow) to browse through the database and repeat the task for different test environments.
- The user can press the "Reset" button to remove the bounding box from the object and restart the process

Screenshot of the envisioned app is as shown below.

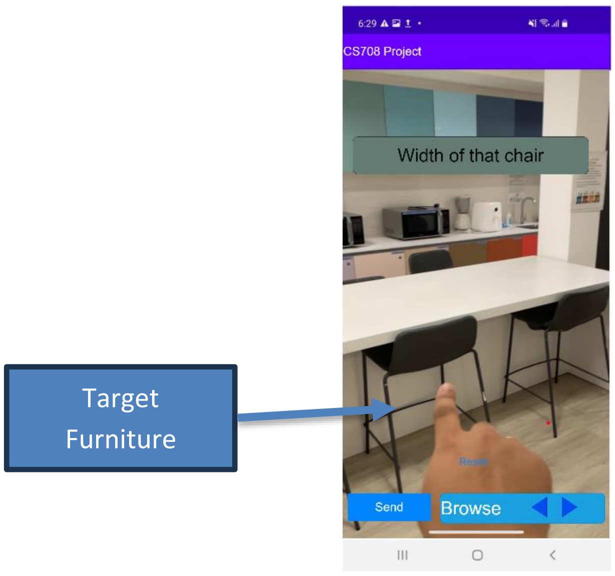


Figure 1: Example UI of the app showing the Three buttons for browsing through test environments, the additional button to send data to the server and the input text field to enter the user command.

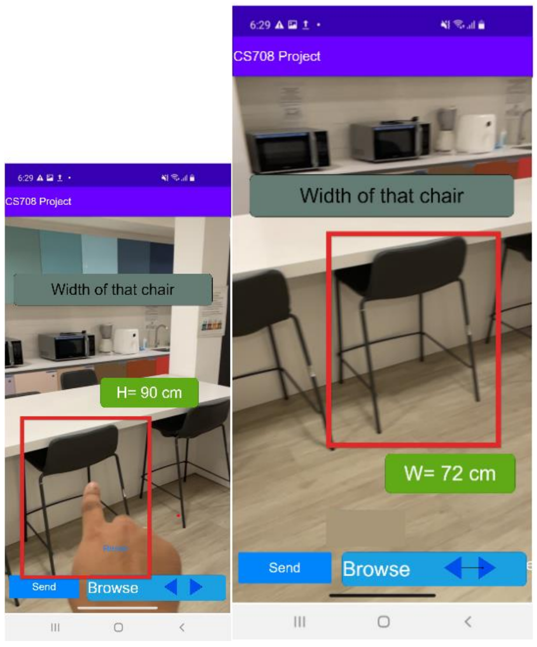


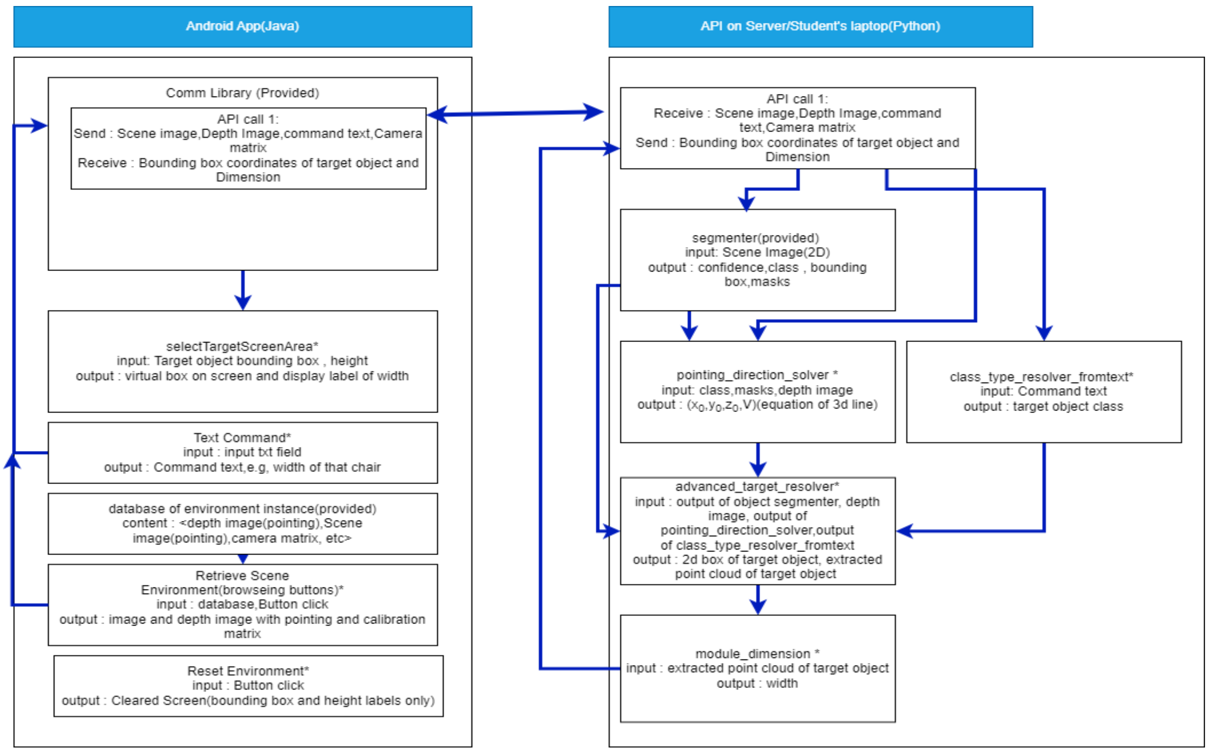
Figure 2: Example UI of the app showing after visualizing the dimensions of the preferred object.

Based on the above ideas, the overall app functionality includes the following components:

1. **(C1)** Developing the base Android application to browse and select a test environment, and to send data (image, depth image, text, projection matrix) to the server.
2. **(C2)** Developing a DNN-based object segmentation system using a camera image (the model will be provided by us)
3. **(C3)** Building a logic-based module to identify the referred object class from the command text. Be sure to avoid hardcoding when extracting the object class, as your

logic will eventually be evaluated using a separate “test” dataset. The command length may vary, and the object type could differ from the text examples provided for building the application.

4. **(C4)** Developing a 3D pointing direction resolver using 3D points of the hand by extracting the hand from the point cloud (point cloud can be constructed from depth image). This involves taking a set of key points corresponding to the hand and mapping them to a pointing location(along the 3D line) in the point cloud. (The inputs are the outputs of the segmentation network and the depth image.) (The process of 3D pointing direction can comprise the following steps: first, extract the hand (Hand segmentation); second, identify and extract the midpoints of the index finger; and finally, fit a 3D line along the selected midpoints of the index finger. In addition to the above, or as an alternative, you can also use various improvised heuristics and different methods or libraries available for keypoint detection on 2D images, then map those to the 3D domain to construct the 3D pointing direction. Here are links to some such libraries: https://paperswithcode.com/task/3d-hand-pose-estimation/latest , GitHub - google-ai-edge/mediapipe: Cross-platform, customizable ML solutions for live and streaming media.)
5. **(C5)** Building a target object resolver that uses inputs from C2, C3, and C4.
6. **(C6)** Building a target dimension calculation module for the object's width calculation that uses inputs from C5 together with the depth image mentioned previously.
7. **(C7)** Rendering an object bounding box on the 2D image (to draw bounding boxes around objects on the image and display width) and displaying the final scene image on the Android device.



Android App(Java)
API on Server/Student's laptop(Python)
Comm Library (Provided)
API call 1:
Send : Scene image,Depth Image,command text,Camera matrix
Receive : Bounding box coordinates of target object and Dimension
selectTargetScreenArea*
input: Target object bounding box , height
output : virtual box on screen and display label of width
Text Command*
input : input txt field
output : Command text,e.g, width of that chair
database of environment instance(provided)
content : <depth image(pointing),Scene image(pointing),camera matrix, etc>
Retrieve Scene Environment(browseing buttons)*
input : database,Button click
output : image and depth image with pointing and calibration matrix
Reset Environment*
input : Button click
output : Cleared Screen(bounding box and height labels only)
API call 1:
Receive : Scene image,Depth Image,command text,Camera matrix
Send : Bounding box coordinates of target object and Dimension
segmenter(provided)
input: Scene Image(2D)
output : confidence,class , bounding box,masks
pointing_direction_solver *
input: class,masks,depth image
output : (x0,y0,z0,V)(equation of 3d line)
class_type_resolver_fromtext*
input: Command text
output : target object class
advanced_target_resolver*
input : output of object segmenter, depth image, output of pointing_direction_solver,output of class_type_resolver_fromtext
output : 2d box of target object, extracted point cloud of target object
module_dimension *
input : extracted point cloud of target object
output : width

Figure 3: High-level block diagram of the final system. Parts marked with asterisk indicates components to be implemented

You could choose to either run the python backend code on the Android phone itself using a library like chaquopy or use your laptop running python as a server. For the second scenario, we have rovided the code as an API for sending and receiving input modality data over a websocket.

B. Detailed Description of Individual Components

The following section provides more details on the various components:

(C1) Base Android Application: The base application, as shown in Figure 1, will be developed using Android Studio and Java, although users may opt to use the more recent Kotlin language if preferred. The application consists of three buttons and an input field. The two arrow buttons, named **"Browse,"** allows users to navigate left or right through a dataset stored on the phone to select an environment instance. Each environment is associated with a single target piece of furniture (The target object, marked on a separate image, will be included as part of the dataset, but not be part of the images viewable in the phone, along with other input modalities in the dataset.). Once an instance is selected, users can manually complete the input field by entering an appropriate instruction to select the target furniture based on the image, without relying on any audio-to-text conversion process. For example, as shown in Figure 1, if the user is pointing at the left chair, a possible command text would be: **"Width of that chair."** After entering the instruction in the input field, users can tap the third button, labeled **"Send,"** to initiate the dimension estimation process. The application streams the relevant data (image, depth image, and homogeneous coordinate projection matrix) to a Python backend server, where target object identification and dimension estimation are performed. Once the computations are complete, the resulting information (object bounding box coordinates and width value) is sent back to the Android application and displayed, as illustrated in Figure 2.

(C2) Object Segmenter: This module takes in the scene image (2D image) and detects objects within it, as well as their instance-segmented masks, confidence scores, and bounding boxes. The goal is not to detect all possible object types but rather to accurately identify pertinent objects. For this assignment, the focus will be on the following objects: **table, people, chair and sofa**. The environment may or may not include people. You are not expected to train your own model from scratch. Instead, we are providing a pre-trained deep neural network (DNN) model, YOLACT (Yolact: Real-time instance segmentation), which you can use directly for detecting/segmenting objects in the scenes

(C3) Target Class type resolver: This module takes in the text instruction from the user and outputs the target object class (e.g., chair or table). As mentioned previously, the object class will be one of 3 different values (table, chair, sofa). The input instructions follow the format below: (i) Furniture type (Instruction: chair), (ii) Furniture type's height (Instruction: chair's height), (iii) Demonstrative adjectives followed by the furniture type (e.g., "that chair's height"). The furniture type can be a table, chair or sofa. (Note: We will use furniture classes different from mentioned classes [table,chair,sofa] only to evaluate this **Target Class type resolver.** Thus, you should exercise care not to hardcode the identification of the 3 classes mentioned here, but to extract the class type by parsing the command according to the prescribed syntax.)

(C4) Pointing Direction Resolver using Point Cloud: This module takes in the point cloud derived from the depth image using the homogeneous coordinate projection matrix, along with output from C2. Any suitable hand detection based logic (hint:you learned about one approach, albeit using other sensing modalities, during the class on Continuous Vision) can be used (note: the hand can be classified as the “person” class in segmentor’s output (which assumes that the hand is associated with a partially visible person object); also, the scene can be include multiple people), to develop a point cloud based pointing direction resolver. This is achieved by extracting the hand from the point cloud and using its 3D points. The process involves identifying a set of key points corresponding to the hand and mapping them to a pointing location in the point cloud. The output of the module will be a 3D line equation in the format **(x,y,z)=(x_0,y_0,z_0)+t(a,b,c)**. where **(x_0,y_0,z_0)** is a point on the line, V=(a,b,c) is the direction vector, and t is a parameter that varies over all real numbers

(C5) Advanced Target Resolver: This module takes in the output of C2, C3, and C4, along with the point cloud of the environment, to determine/infer the final user-preferred target furniture. The module's logic should identify the target object by combining the class clue extracted from the text instruction (C3) and the pointing clue derived from the point cloud (C4).

Figure 4 depicts an exemplar 3D line constructed using the 3D point cloud of the hand. It is important to construct the pointing direction resolver in the 3D domain, as it provides a more accurate directional clue compared to an image-based 2D pointing direction-solving method. The correctness of the pointing direction is crucial for precise object selection in a dense environment, where there may be multiple distractor objects (i.e., objects of the same class referred to by the user instruction) in close proximity. The output of this module is the target furniture’s point cloud, which will be used to calculate the width.

(C6) Width Compute module: Subsequently, after processing the module described in **C5**, the selected furniture’s **3D point cloud** is passed to the module as input to calculate the width of the furniture. For simplicity, you can assume that the furniture is always on the ground. (**Note:** Different pieces of furniture in different scenes have varying orientations with respect to the user’s egocentric view) (Note: Use scales as **cm**)

(C7) Bounding Box and Width Display in the Final Scene Image: This module takes in information about the target furniture, including its **bounding box coordinates in 2D space** and **width**, to overlay this information on the **Android app’s image preview**. The bounding box and label of width dimension are rendered entirely within the Android app based on the return values of server API calls. The target furniture's bounding box, as returned by the server, determines its placement on the screen.

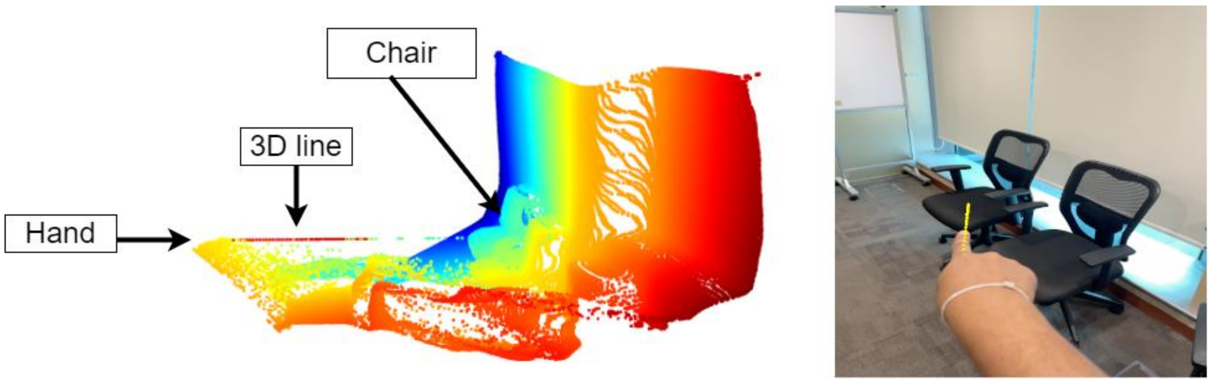


Figure 4: Visualization of the Pointing Direction Overlaid on the Point Cloud and Its Projection onto a 2D Image

C. Datasets and Their Description

As part of the project, which involves developing suitable modules, we will provide you with a dataset containing five different environment instances with varying scene complexities. These complexities are expressed as a tuple of two orthogonal attributes: (a) the number of furniture items in the physical space (sparse vs. dense) and (b) instruction complexity (unambiguous vs. ambiguous), indicating whether the environment contains a single instance or multiple instances of the furniture class that is referred to by the user instruction. You can use this dataset to effectively test the end-to-end pipeline. Eventually, for grading your project, we will utilize a separate TEST dataset, which will **NOT** be made available to you. However, this test dataset has been verified to have the same properties as the dataset provided to you

Dataset for Implementation/Validation

We will provide the following dataset, including a 2D image, a 2D depth image, and camera matrices for converting the depth image (2D) to a 3D point cloud (PCD) to be stored on the phone. Additionally, the dataset will include the target object marked on a separate 2D image, the ground truth width dimension of the target object as text, and the point on the line, directional vector [$(\boldsymbol{x_0}, \boldsymbol{y_0}, \boldsymbol{z_0})$, V] of the 3D line separately, to validate the co-modules you will implement during the server-side pipeline..

Environment1 <2D image,2D depth image, camera matrices, target object, width value, ground truth 3D pointing line

...

Environment5 <2D image,2D depth image, camera matrices, target object, width value, ground truth 3D pointing line

D. Sample Code

To reduce the time needed to complete the *less-interesting* parts of the assignment, we are providing you with the following projects:

• Android project **CS708_Android**: The Android app to be enhanced, with built-in implementation of several components:

- UI components that include
  - Two arrow buttons to browse through the image database
  - Single button to initiate the dimension calculation process and receive outputs from the server
  - An area showing Image View. The bound box and dimension label will also be rendered on this area.
- Helper components, including the Commn. Library that allows the mobile phone-based components to exchange data with the server-based components

We tested the provided base code in Android Studio (**2024.2.2**). It is recommended to use the same version for development to avoid any unexpected errors. Python project **CS708_server**: Python Flask project, which can be used as scaffolding for server-side API code. This Python-based code will include built-in implementation of several components:

- Processing components, including stubs for the **Target Class type resolver, Pointing Direction Resolver using Point Cloud, Advanced Target Resolver** (functions are included in pre_processing.py)
- DNN components, including the Yolact instance segmentor
- Helper components, including the Commn. Library that allows the mobile phone server communication and data exchange

How to Run the Project

1. Run the server
2. Run the android application (you can run it either in the emulator or a real device)
3. Set up the communication between the emulator and the server (Do this ONLY if you run the android app in the emulator)

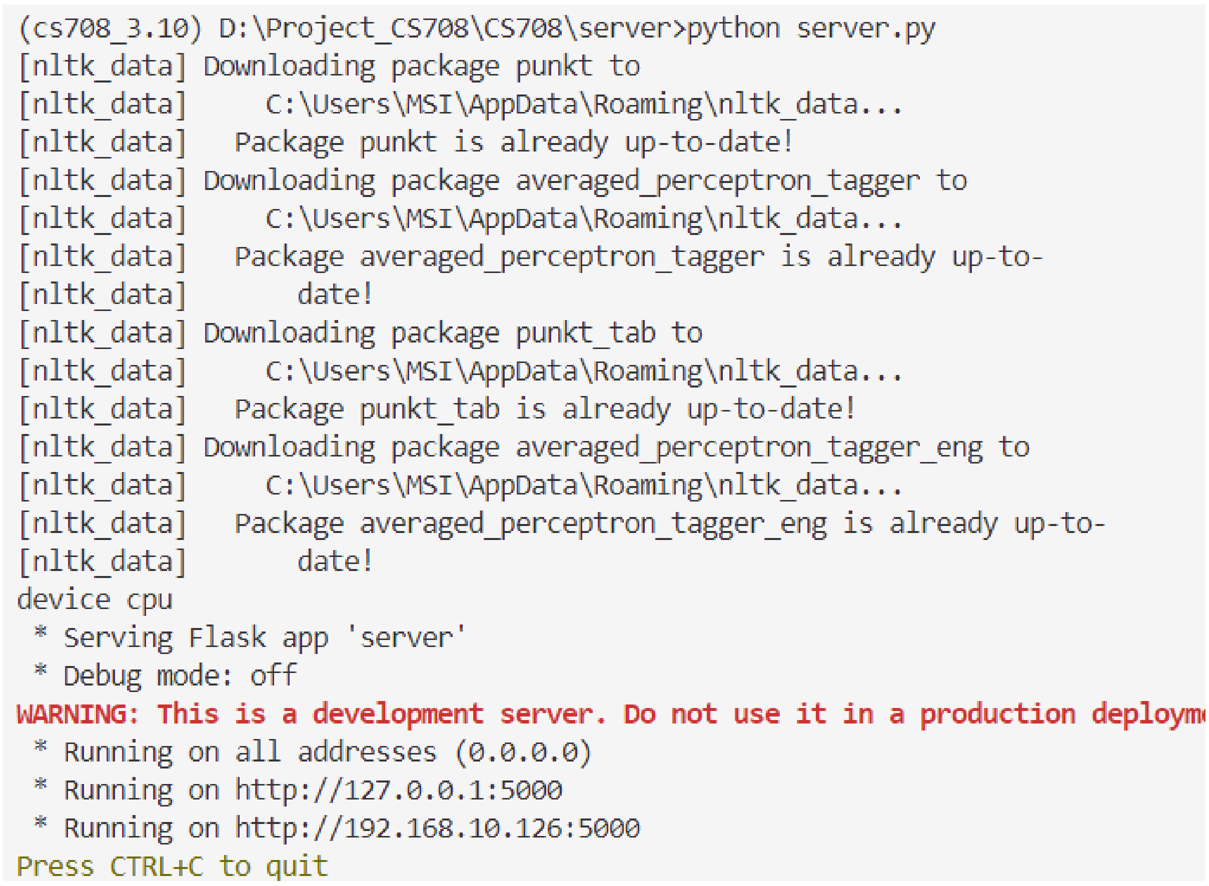
How to run the server:

1. Go to the server directory
2. Install all dependencies
   pip install -r requirements.txt

3. Set the IP address of the server inside the server.py. The default value is 0.0.0.0 and the port is 5000

```
app.run(host='0.0.0.0', port=5000)
```

4. The message below is shown if the server is successfully run



```
(cs708_3.10) D:\Project_CS708\CS708\server>python server.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\MSI\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\MSI\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:\Users\MSI\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     C:\Users\MSI\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
device cpu
 * Serving Flask app 'server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deploym
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.10.126:5000
Press CTRL+C to quit
```

5. Note down the last IP address shown in the terminal (192.168.10.126 in picture above). We will use this IP address in the client code (Android app)

How to run the android App

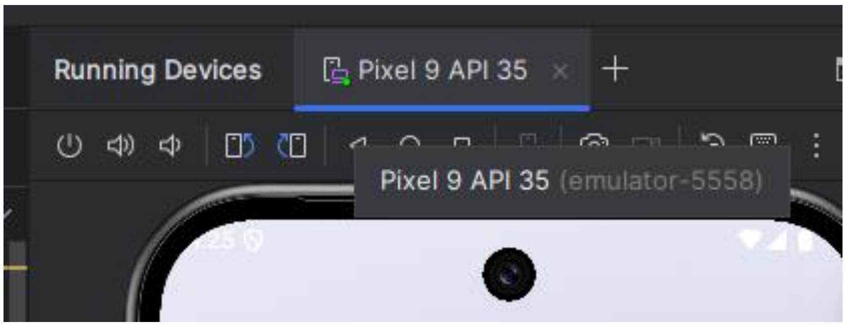
1. Go to android_app/app/src/main/java/com/example/cs708_android/CommModule.java
2. Set the IP address (line 17) to the one that we obtain previously in the last step when running the server

   private final static String SERVER_IP = "http://192.168.10.126:5000/";

3. Run App (Shift + F10 or Run > Run App)

Set Up the Communication between the Emulator and Server App

1. Check the emulator port. You can do it by pointing your cursor to the Running Device Tab



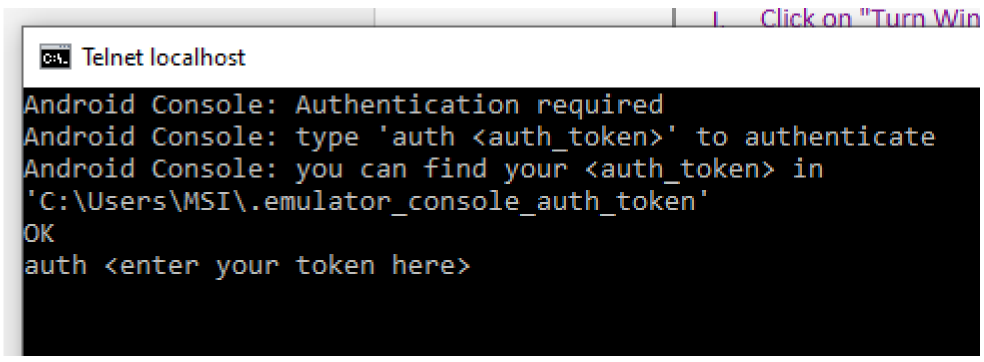
Our emulator port is 5558 in the above picture

2. Make sure you have installed telnet. If not yet, then follow these steps:
   a. Open the Control Panel (search for it in the Start menu)
   b. Go to "Programs and Features"
   c. Click on "Turn Windows features on or off" in the left sidebar
   d. In the Windows Features dialog box, scroll down and check the box next to "Telnet Client"
   e. Click "OK" and wait for Windows to apply the changes
   f. Once installation is complete, you can use Telnet from the Command Prompt
   g. Open the Control Panel (search for it in the Start menu)
   h. Go to "Programs and Features"
   i. Click on "Turn Windows features on or off" in the left sidebar
   j. In the Windows Features dialog box, scroll down and check the box next to "Telnet Client"
   k. Click "OK" and wait for Windows to apply the changes
   l. Once installation is complete, you can use Telnet from the Command Prompt
3. Open cmd, run the telnet

   telnet localhost 5558

   Here, 5558 is the port of our android emulator. We use localhost because we run the server in the same desktop/laptop where the emulator run

4. Perform authentication by following these steps



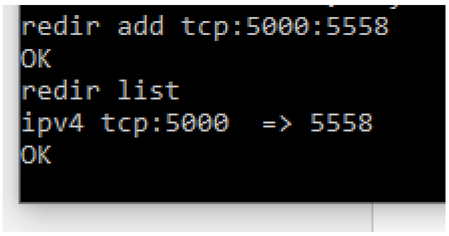
```
Android Console: Authentication required
Android Console: type 'auth <auth_token>' to authenticate
Android Console: you can find your <auth_token> in
'C:\Users\MSI\.emulator_console_auth_token'
OK
auth <enter your token here>
```

5. Add redirection

redir add tcp:5000:5558

5000 is the port of our server and 5558 is the port of our emulator

6. Check if the redirection is added by executin redir list



```
redir add tcp:5000:5558
OK
redir list
ipv4 tcp:5000  => 5558
OK
```

The communication has been successfully initiated

Running the Server

The provided base project (both the server and Android project) can run completely on a Windows laptop. The Android application can be emulated on an Android emulator, and the Python server can be initialized on VS Code on a regular laptop (no GPU required).

If you are developing the code using only the emulator (without an actual Android mobile phone), there are a few extra steps required to connect the emulator device to the internal server running on the same laptop.

Steps to Follow on Android Studio:

1. **Run the virtual device on the Android Studio emulator**
   Networking between the emulator and the host computer can be tricky. More details can be found in the official Android Emulator Networking documentation (https://developer.android.com/studio/run/emulator-networking). Below are simplified steps to set up the connection.

2. Open the terminal on Windows and install telnet: **(If not already installed, you may need to enable it from Windows Features or install it via a package manager.)**

3. Execute the following command in the terminal. it is important to run the device on the emulator to execute this step:
   telnet localhost 5554(execute on windows terminal)

4. After above step,Set up port forwarding between the host computer and the emulator:

   redir add tcp:5000:5000. (General format: redir add tcp:<host computer port>:<emulator port>). To verify the port forwarding, run: redir list

5. Open VS Code (latest version recommended) and install WSL (Windows Subsystem for Linux) with an Ubuntu distribution. This is required to initialize the server in a Linux-like environment.

6. Retrieve the IP address of the WSL instance**.**
    - **Open the WSL terminal and execute: ifconfig**
    - **Look for the IP address under the eth0 Ethernet interface.**
    - **Set this IP in SERVER_IP in CommModule.java of the Android project.**
      Example: SERVER_IP = "http://172.17.161.118:5000/";

7. Following these steps will ensure that the emulator can communicate with the internal Python server running on the same laptop

Reference: If interested, you may note that the App functionality described in this project is similar to, but not identical (actually, it's a much more restricted and simplified version) of the AIrFurn App described in a recent paper titled "*MIMIC: AI and AR-enhanced Multi-Modal, Immersive, Relative Instruction Comprehension*" that has been accepted for publication in ACM IMWUT. A preprint of the paper is available (purely for your personal consumption) at: https://drive.google.com/file/d/1jzyG9YX2lHEvqIHNHZxqPYhyMC226eSJ/view?usp=sharing

You are welcome (although under no obligation) to review the paper and adopt (or use a baseline to modify) the logic described there for various components of your project—e.g., for the Pointing Director Resolver or Advanced Target Resolver components. You will not be penalized for replicating any of the logic mentioned there, although you are completely welcome to construct your own logic as needed.

E. Implementation Specifics & Testing

Note: To avoid issues with grading, which will be done at least in part via automated test scripts, please ensure that the format of return values of your function implementations conform to the specification.

E1. (3 points) Implementation of the **Target Class type resolver** Module: This will be tested by scripting execution of your Python implementation of the function on several test inputs and comparing the output against expected furniture category. Grades will be awarded according to accuracy.

E2. (4 points) Implementation of **Pointing Direction Resolver using Point Cloud**. This will be tested by scripting execution of your Python implementation of the function on several test inputs and comparing the output against *cosine similarity* (https://en.wikipedia.org/wiki/Cosine_similarity)with expected 3d line drawn from index finger. Grades will be awarded according to accuracy.

E4. (3 points) Implementation of the **Advanced Target Resolver** module. This will be tested by scripting execution of your Python implementation of the function on hold-out set of test segment files and observing how well it predicts the target furniture predefined per environment instance. Grades will be awarded according to accuracy.

E5. (3 points) Implementation of the **Width Compute** module. This will be tested by scripting execution of your Python implementation of the function on hold-out set of test point cloud set of chair and table objects and observing how well it calculate the target furniture's width in {CM}. Grades will be awarded according to accuracy.

E6. (**2 points**) Implementation of the **bounding box and width overlaying** module. This will be tested by having your sample Android app communicate with a mock API that returns test values (target bounding box and width) to your Android app, and observing the rendered virtual object type, its placement, and bounding box location. Grades will be awarded according to correctness of width and placing the bounding box around the object.

E7. (**2 points**) **Overall End-to-End App Integration**: This will be tested by running your server-side implementation (on a laptop or a server) and your Android app (on a phone running Android 15 or newer) and evaluating its operation and output on test dataset.

E8. (**3 points**) **General Code Design, Documentation and Explanation**: As always, it is important that you document your code clearly and that you use good coding hygiene (e.g., naming variables to clearly reflect what they denote).

F. Details of Things to be Submitted

Please submit your materials via the corresponding assignment folder that has been created in eLearn. Please submit a single zipped file (named <First Name>_<Last Name>.zip) that should contain the following:

- Your API source code (.py) of your algorithm implementation and your model file. Please keep the same structure as the scaffolding code, i.e. script files in the same directory.
- Your Android app .apk and Android Studio project. Please also keep the same structure as the provided Android Studio project.
- A *readme* file which includes detailed instructions on how to run your code, including required libraries. In other words, I should be able to "blindly" follow your instructions and create a running instance of your executable for testing.
- A *design* file (called either **design.docx** or **design.pdf**) that explains the logic that you've implemented in your approach. For example, it should describe, as examples, (a) the logic that you've used to identify target object class from text, (for the part C3); (b) a description of logic used, and the final model choice, for the Pointing Direction Resolver (part C4), (c) a description of logic used, and the final model choice for integrate multimodalities to select target object(part C5). [Please keep this file to a maximum of 4 pages—there's no need to ramble on.]

G. Contacts for Clarifying Doubts

As you implement the project, you may desire clarifications for issues or need assistance with some programming issues. Here is the protocol to follow:

(a) Email the course Teaching Assistant, Dhanuja, at: dhanujaw.2020@phdcs.smu.edu.sg , Imam, at : imamy.2020@phdcs.smu.edu.sg or contact them via Microsoft Teams.

(b) In case the issue is not resolved and you need to escalate further, please email me at archanm@smu.edu.sg.