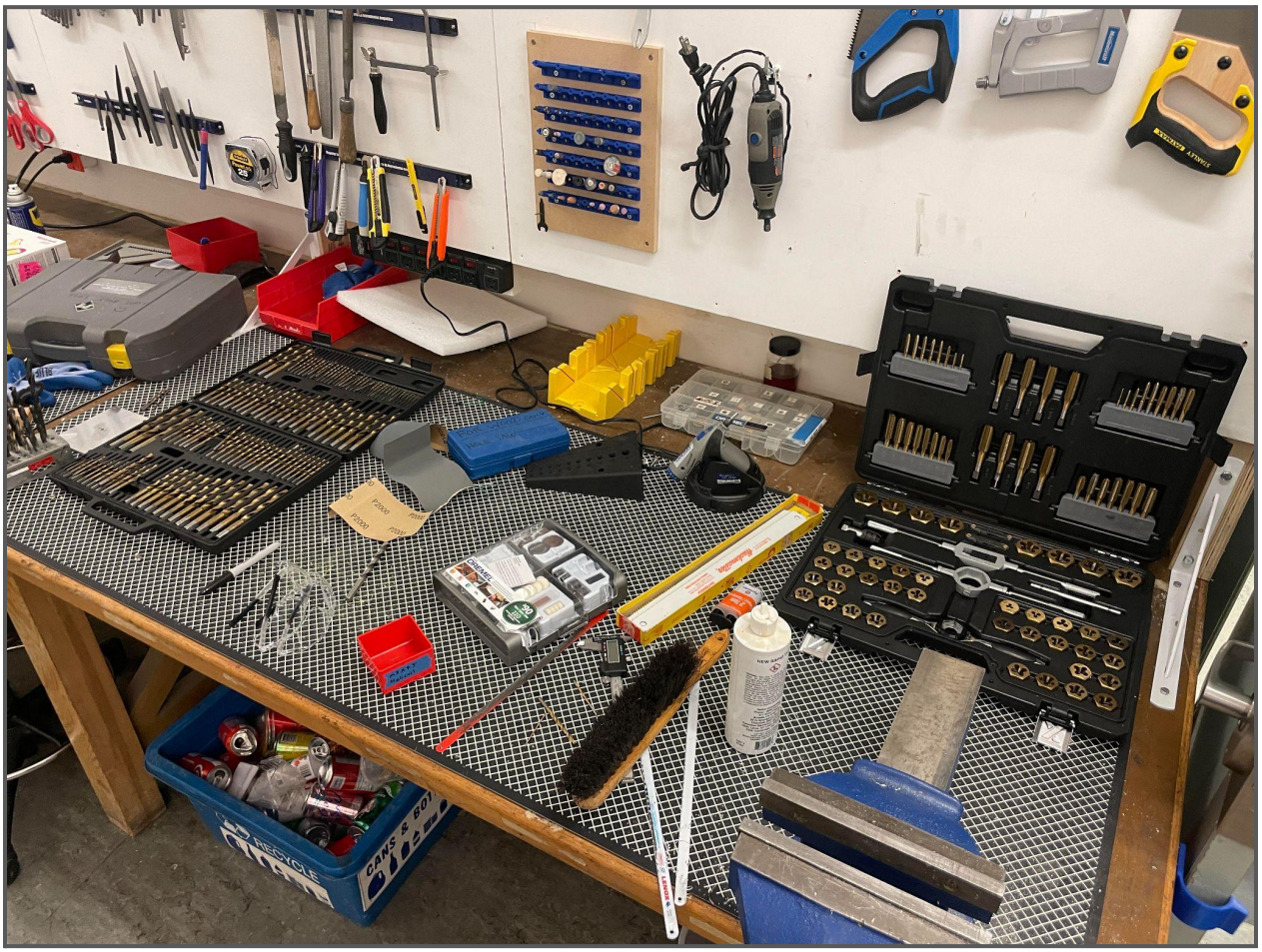
Software Tools for Organizing Physical Workspaces



Emily Love
Seth Hinz
Davis Johnson
Sean Lan

Project Sponsor:
Engineering Physics Project Lab

Project 2209
ENPH 459
The University of British Columbia
April 8th, 2022

Executive Summary

This project's goal is to simplify organization in shared workspaces by building an app to help users identify and put away tools and equipment. Furthermore, this project aims to explore the technologies available to solve this problem, and to further understand their capabilities and limitations.

This project is motivated by the need to improve organization in the Engineering Physics Project Lab. The space is used by hundreds of students for a variety of projects, and the Project Lab team does not have the resources to continually clean up themselves, nor to teach every student where each tool in the workspace goes.

Additionally, augmented reality (AR) is now available on most mobile devices. Thus, an AR app that helps workspace users keep their space clean is a viable solution to this problem.

We developed a prototype Android app that can identify objects from a set of simple items, determine where the user is in their workspace and guide them to where the object should be put away. In addition, we implemented a basic administrator workflow, which can add additional objects to the database and set their desired locations. We demonstrated that we can use computer vision and AR to identify objects, localize a user in their workspace, and navigate them to where an object should be put away. Through this process, the app simplifies the organization of complicated workspaces with many users.

The main remaining risk factor is integrating a more advanced neural network to improve the reliability of object detection. Many current solutions struggle to add new objects to the set of detectable items without time and resource-intensive computations. However, we expect this problem to be solved in the near future due to advancements in few-shot machine learning.

Other recommendations for development include moving the app to a cross-platform framework, improving the user experience, and adding a login and security system to more seamlessly combine the admin and user workflows.

Table of Contents

Table of Figures

Introduction

In many workplaces, organization and tidiness are significant challenges. When tools and equipment are not put away properly, they build up in shared areas, which interferes with the productivity of everyone using the workspace. Our project aims to solve the issue of messy workspaces through a phone that can identify objects and guide users through the workspace.

Sponsor

Our sponsor is the UBC Engineering Physics Project Lab, a space at UBC used by hundreds of Engineering Physics students as well as engineering student design teams to work on a variety of technical projects. Due to the variety in projects, students typically are not familiar with all the different equipment in the space, as only some of it is relevant to them. Additionally, teaching each student about all the tools in the space and where they go is usually unnecessary, and would waste the time of students and faculty.

Because of this, it is difficult to maintain a tidy environment (Fig. 1). Even if someone were assigned to clean up left-out tools, given the variety of different projects in the space which may each use niche or unique tools, it can be difficult to identify tools or know where they belong.

Since the Project Lab team is involved with managing so many student projects, they do not have the resources to dedicate to keeping the lab clean themselves. We have been tasked with developing an app that can better enable the users of the lab to keep things tidy ourselves, without requiring time-consuming training. This app should also explore using augmented reality to enhance guiding the user through the workspace.

Figure 1. Image attachment taken from an email sent out last month about the messy state of the Project Lab.

Background

Computer vision and augmented reality have evolved significantly in the recent past, and have a growing presence in the software space.

Augmented reality (AR) is already all around us, from Snapchat filters, to Google Street View, and the popularity of Pokemon Go (Fig. 2). Due to the rapid advancement of mobile devices, it is now simple to track a device's three-dimensional (3D) location and display live 3D visuals alongside a device's camera feed.

Figure 2. Pokemon Go using AR to render a pokemon (left) and Google Street View using AR to display directions (right).

In addition, object recognition techniques are rapidly improving. Initially, SIFT (Scale Invariant Feature Transform) [5], an algorithm that tracks key points in an image to find matches, was a breakthrough in this technology. Today, modern neural networks trained on millions of sample images enable reliable object recognition in many applications. There are even pre-trained models such as YOLO [9] available, which makes object recognition more accessible than ever.

However, a significant drawback remains: adding new labels to a neural network requires substantial time and computation for retraining. Given the nature of projects in the lab, we expect to frequently need to add custom tools that the app must identify.

In spite of this drawback, due to advances in few-shot learning [10] we expect this problem to be solved in the near future. In the meantime, we are confident that with the technologies in object detection and AR currently available to us, we can build a proof-of-concept solution to address the unmet need of keeping busy workspaces tidy.

Project Objectives and Scope

The objective of this project is to improve organization in workspaces through an app that helps people put things away. In developing this app, we also aim to explore, compare, and understand the broader capabilities of the available technologies.

During the app's development, our main focus is on the AR component of the application, with a secondary focus on a placeholder object detection system. We aim to include a set of simple objects to be identified by the app, and to be capable of navigating a user through the workspace using AR to put them away.

Given our focus on the user-facing features, we will support a simplified administrative workflow for this app, which includes setting up the object locations in the workspace as well as adding new objects.

Future features could be added to the administrative side, but we have constrained the scope in order to maximize our exploration into the user-facing AR technology and to prove the fundamental capabilities of our app.

Discussion

Problem Breakdown

The following process outlines a human approach to organization (Fig. 3):

- Identify the object to be sorted
- Figure out where the object should go, either by knowing beforehand or asking someone
- Figure out where you currently are
- Determine the path from the current location to the target storage location
- Put away the object

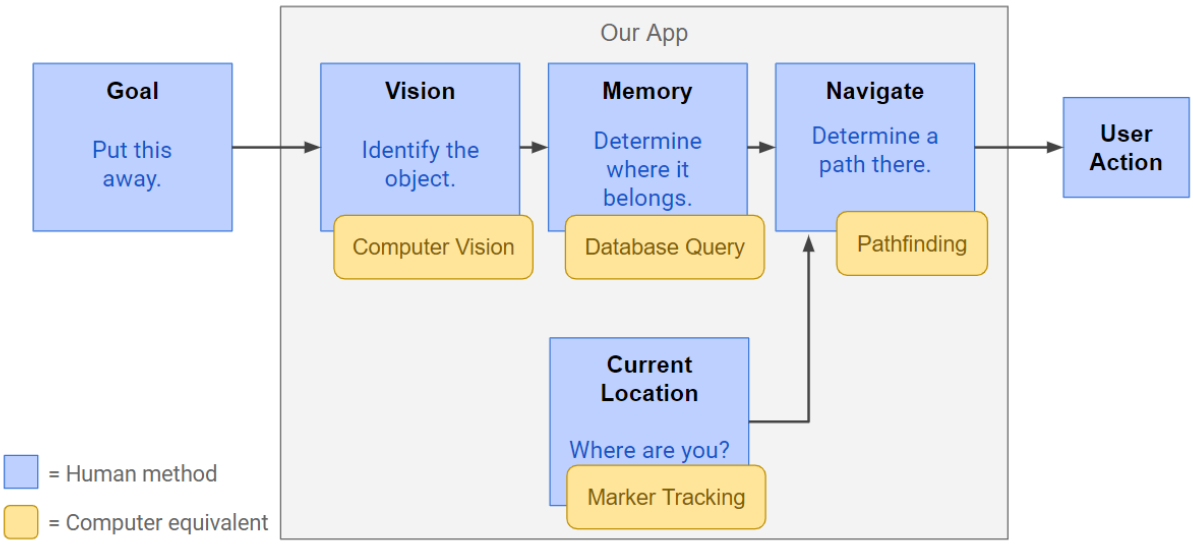


Figure 3. The typical workflow for returning an object to its location (blue), as well as the technologies that our application uses to simplify the process (yellow).

Using software, we can replace the most tedious parts of the organization process. In Figure 3, the portions in yellow indicate the alternatives that we have developed and integrated into our application, which streamlines the organization workflow.

As an analogue to the process described above, we separated the functionality of our app into three major portions: to Identify, Localize, and then Navigate, as seen in Figure 4. We *identify* an object, taking advantage of computer vision and database technology to determine what an object is from a picture and where in the workspace it should go. We *localize* the user, by calibrating their position with predetermined markers placed around the workspace then tracking their location using AR as they move to put the object back. Finally, we *navigate* the user to the object's storage location using AR visuals.

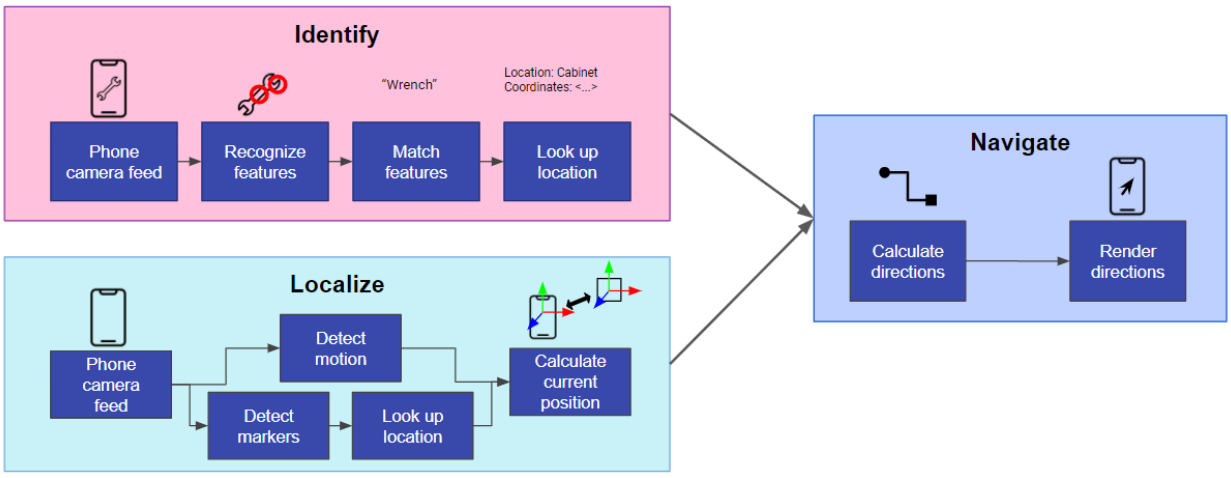


Figure 4. The functionality of the app can be broken down into 3 main portions which reflect the process normally taken by humans when organizing.

Technology investigation and selection:

Identify

The first step in our process is to identify the object that is to be organized. We considered two main technologies: Scale-invariant Feature Transform (SIFT), and neural networks.

SIFT is a computer vision algorithm designed to detect features and keypoints in images, which we can compare with a database to detect an object [5]. SIFT is publicly available in OpenCV and is thus easy to integrate. The algorithm is also well understood, meaning it is easy to troubleshoot and test. However, its performance suffers under inconsistent lighting and angle conditions.

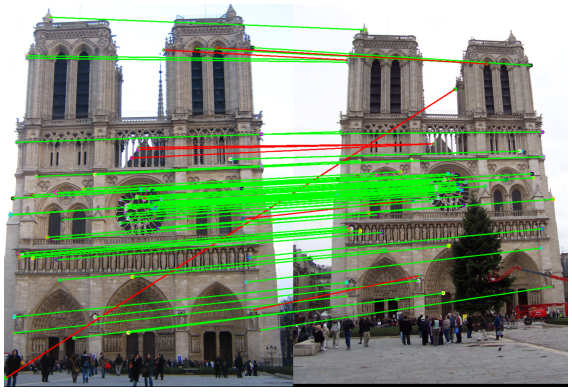


Figure 5. A demonstration of SIFT detecting and matching keypoints in similar images

In comparison, a neural network should be more accurate and resilient than SIFT, but requires users to take images from a variety of angles for each object they want to be able to identify, which increases the barrier to entry for the application. Furthermore, a neural network necessarily requires re-training of the entire identification network if only a single new image is added to the database, which can take hours or more for larger databases. This is a major inconvenience from a developer and user perspective and convinced us to rely on SIFT for the current iteration of the app.

We are looking into an emerging neural network technology in the form of few-shot learning. Few-shot learning creates a network that can determine similarities and differences based only on a few training images [10], as opposed to traditional neural networks which require as much training data as possible. This technology could both reduce the amount of training images needed for a given object and the time required for training. However, because there is no library readily available for our use at this time, we have defaulted to SIFT as a more traditional method for identification.

Our particular identification process is as follows: we store in a database several images of each object. SIFT calculates key features in each image, along with descriptors of each key point. If we do the same on a newly scanned image of an unknown object, we can apply a matching

algorithm between the two sets of descriptors, and get as a result a match score counting the number of similar features between two images. We then select the “best match” by finding the image in the database with the highest match score.

After the object is identified, we can simply perform a database lookup to determine where the object belongs in the workspace. Any database technology would work for our application, as our needs are straightforward. Thus, we prioritized familiarity and implementation speed by choosing to use MongoDB Realm, a serverless platform that is tailored for mobile apps. As our prototype is only for internal use and stores no sensitive data, our database does not currently employ any authentication. In a real deployment, user permissions would be segmented both by workspace (an employee at company A should not have access to company B’s data) and employee clearance (such that company administrators can control private information).

Localize

The next step is to figure out where the user currently is in the workspace and keep track of their position as they move around the space.

We briefly considered GPS and IR beacons located around the workspace to provide tracking. GPS has poor accuracy (particularly indoors), while beacons are both expensive and slow to install. [6]

Fortunately, motion tracking is a solved problem in existing AR frameworks, and is generally easier to use than these alternatives. There are 3 primary frameworks that provide this feature: ARCore, ARKit, and ARToolKit. We have opted to use ARCore due to the features available, richness of documentation, and device compatibility (see Appendix C).

ARCore has a feature known as simultaneous localization and mapping (SLAM) [1], which combines accelerometer and gyroscope readings from the phone’s sensors with feature detection from the camera feed to track the user’s position in the environment. This provides us with the user’s position relative to where they opened the application.

However, when using motion tracking as a localization solution, we must also determine the user’s absolute position. We researched and tested two primary technologies for localization: Aruco and ARCore’s Augmented Images API. Both technologies identify markers in the workspace from the camera feed and return the user’s position and orientation (together known as pose) relative to the marker. Aruco is meant for efficient detection of specific fiducials, whereas ARCore’s library can detect more general planar images, provided they contain enough unique features. This flexibility potentially incurs more computational cost and may have more stringent requirements for image visibility [4].

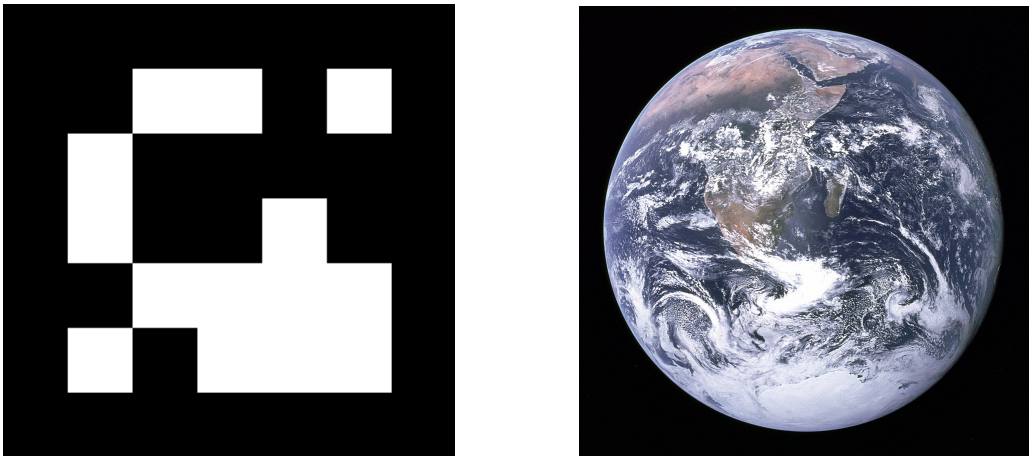


Figure 6. An Aruco fiducial (left; simply colored, straight lines) compared with an image that performs well with the Augmented Images API (right; high resolution with many unique features)

During our development, we had difficulties integrating the Aruco library, and found that ARCore's performance was adequate for our system at this time. We decided that if the absolute localization process was significantly impeding the user experience, then we could replace the marker detection with Aruco at a later time.

Navigate

Finally, we considered two options for providing navigation guidance to the user. We could either provide a simple compass arrow pointing the user to the storage destination, or we could go further and provide pathfinding that goes around obstacles and walls, possibly being updated dynamically as the workspace changes (Fig. 7).

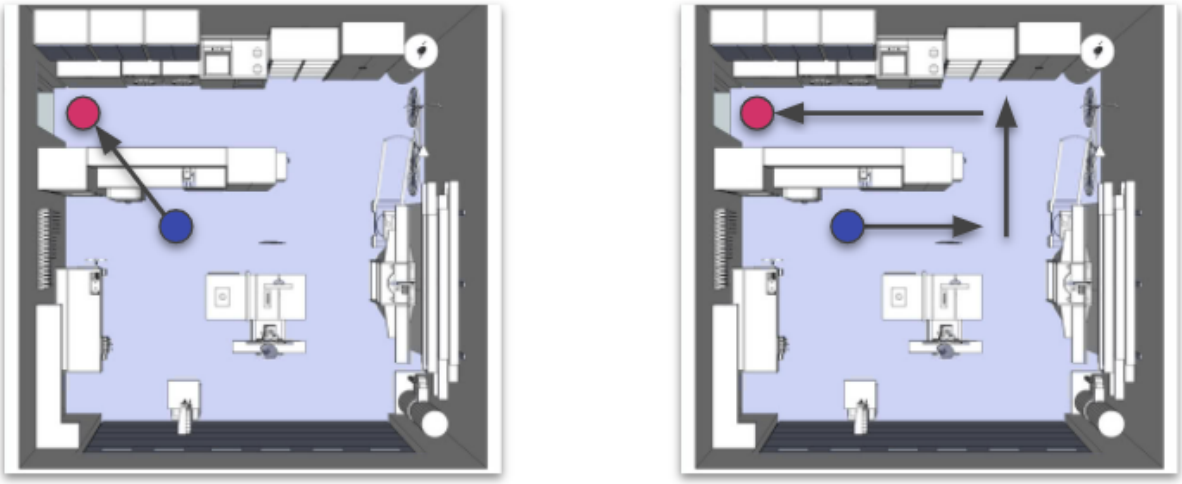


Figure 7. Comparison between displacement vector and computed path directions for navigation when encountering an obstacle.

We settled on the compass arrow displacement vector, as we believe it provides sufficient guidance for a proof of concept application. Sophisticated pathfinding may improve the user experience, but would require significant software infrastructure to be developed for the app, and more initial set-up by an administrator to map out the workspace.

System Integration and Data Flow

Having established the tools and technologies we wanted to use for this project, we began integrating them into an Android app. Because we had not previously worked with ARCore, we built our application off of augmented_image_java, a sample project provided in the ARCore SDK [1]. Using Gradle (Android Studio's default build system) we then installed opencv-contrib [8] and MongoDB Realm.

The main functionality of our app is to route data between these packages, and present a user interface built on top of them. When the user opens the app, and is prompted to scan an object, our app routes data as seen in Figure 8.

In order to classify an object, SIFT requires an image (from the camera) and a set of reference images to compare against (from MongoDB). Once complete, the classification and a reference image are displayed to the user. This allows the user to be sure the classifier identified the target item correctly.

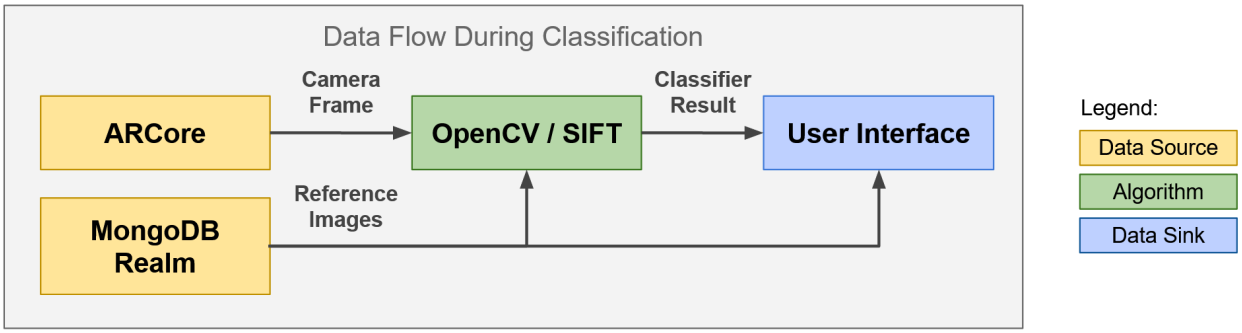


Figure 8. The flow of data through our app during the classification process.

The data flow is similar during the navigation process, as seen in Figure 9. In this process, the classifier result (from SIFT), the item location (from MongoDB), and the user's current location (which is determined from ARCore's localization) are used to calculate a path from the user to the target.

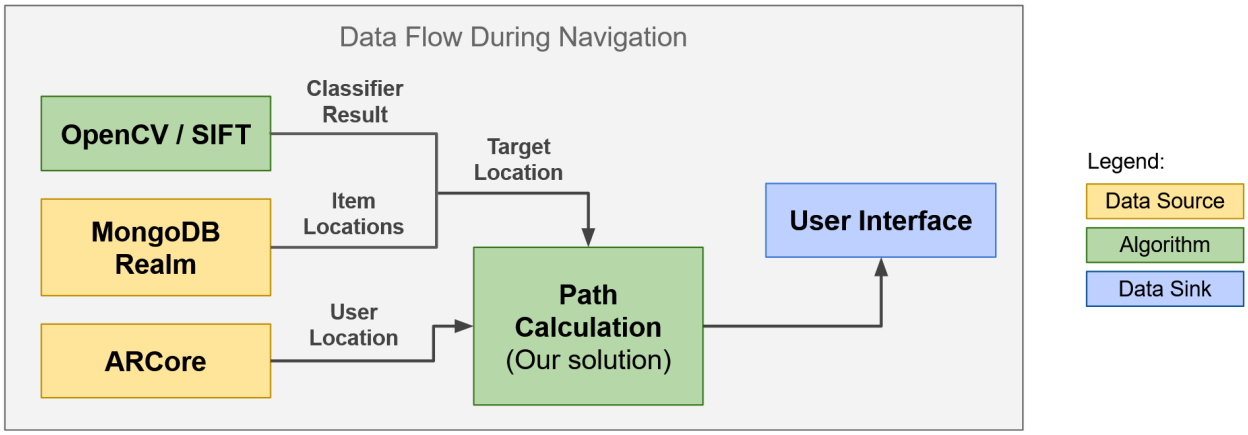
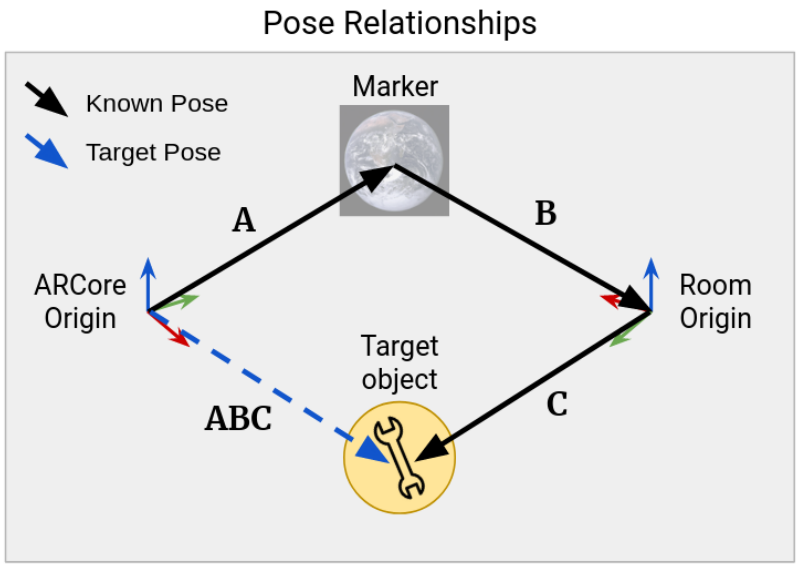


Figure 9. The flow of data through our app during the navigation process.

As discussed earlier, we chose to compute a displacement vector as a simple form of navigation. However, this requires bridging the absolute coordinate system (relative to the room) to the ARCore coordinate system (relative to the user).



A : Pose relating user to marker (via ARCore SLAM)
B : Pose relating maker to room origin (via database)
C : Pose relating room origin to target item (via database)

Figure 10. The relationships between known and unknown object positions in a workplace. In this diagram, capital letters represent ARCore "pose" objects, and adjacency represents the composition operation.

This is further complicated by the relative rotation between different coordinate systems; the x-axis of the room is not coincident with the x-axis that ARCore establishes. Fortunately,

ARCore's API includes a "Pose" class [3] which describes a composable, invertible transformation between two coordinate systems. In the diagram above, we show how the pose relating the user and the target object is computed. Our database stores the poses of the target objects and markers relative to the room, and ARCore computes the pose between the user and a marker. Using pose composition, we are therefore able to find the pose of a target object relative to the user.

User Interface

Although the backend described constituted most of the work in this project, it alone is not a complete prototype. As user action drives this workflow, a convenient and intuitive interface is necessary to demonstrate the function of the app.

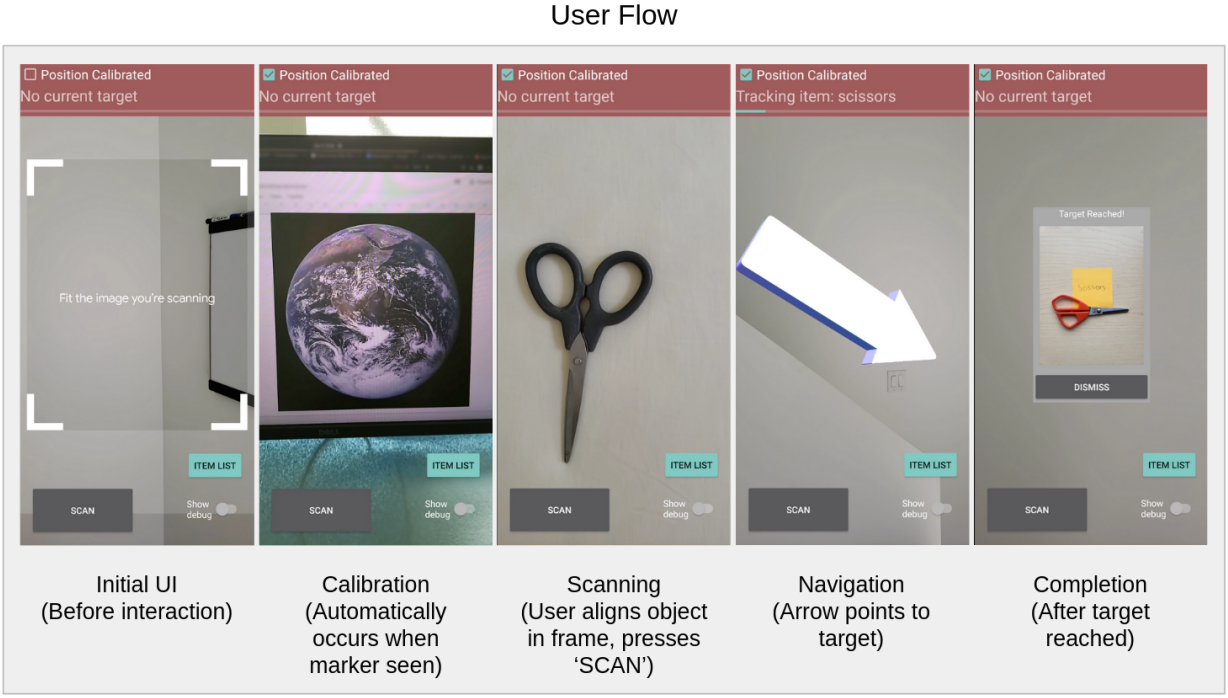


Figure 11. Several screenshots from our app during the classification and navigation workflow.

Figure 11 details the main workflow in our prototype. When the app is first opened, the user is prompted to scan a marker image by holding it in the frame. Once this is done, a calibration indicator (top left of the screen) is toggled, and the user can then scan an object (as shown in the central panel). The classification is then displayed at the top of the screen, and a 3D navigation arrow directs the user to their destination.

Finally, when the user reaches their destination, the arrow disappears, and an image is displayed that shows where the object should be placed. In the diagram, a labelled spot on the table was used, but this image could be used to identify the specific cabinet, drawer, or hanger where an item belongs.

We also have included menus that allow users to directly set a target without needing to rely on the classifier. Basic administrative capabilities are included as well, such as the ability to add new items to the database, and automatically set the position of an item using the localization component already active in the app.

Details of app usage are available in Appendix A: App Usage Manual.

Testing and Results:

In order to characterize the performance of our application, we performed tests on each of our core components: identification, localization and navigation.

Identification

The key performance parameter of our identification system is the ability to consistently identify the correct object. We took 5-6 images of each of three objects (a tape measure, whiteboard marker, and lock) and loaded these into the application; then, we performed ten scans of each object and recorded the SIFT algorithm's identification.

		Classifier Result		
		Lock	Tape	Marker
Ground Truth	Lock	2	4	4
	Tape	0	5	5
	Marker	1	0	9

Figure 12. Table of classifier testing results between a lock, a tape measure and a marker.

As expected, the SIFT algorithm is not very reliable; the identification is biased towards certain objects, in this case, the marker.

We later tested with two objects (scissors and fork) with 16 reference scans each, and found identification was correct with approximately 90% accuracy. The quality of detection does appear to improve as more images are included; however, computation time also increases linearly, so this may not be scalable to databases with large numbers of objects.

Since the focus of our project was on the AR components of the application, we are unconcerned by these results. We recommend eventually replacing this component with a neural network as few-shot learning improves.

By replacing the identification process with a neural network, we expect that the system will identify objects more reliably.

Localization

The performance of the localization can be quantified through the range of distances and angles from which a user can scan a marker. We thus determined the range within which users could calibrate their position by scanning a fiducial.

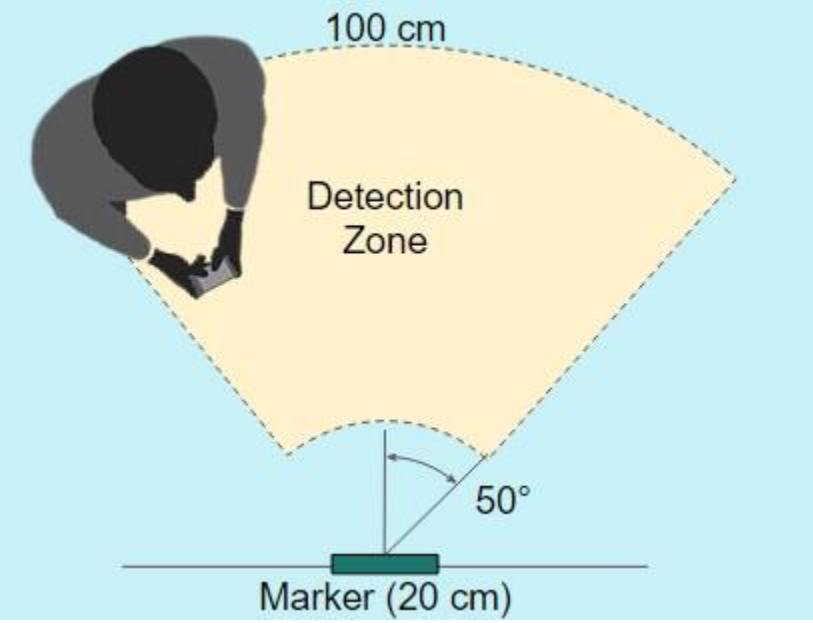


Figure 13. Range where fiducials detected encompasses a 100° range from 30 cm to 100 cm away from the marker.

We determined that for a 20 cm marker, the user must be within a distance of 1m for the app to detect it and must be far enough from the marker that it fits within the camera's frame (30 cm away, for a 20 cm marker). It also can detect the fiducial within a 100° range as long as the image is fully within the camera's frame.

Navigation

Finally, to evaluate the app's navigation, we calibrated the app, then moved along a ruler and recorded the in-app coordinates and real-world coordinates simultaneously. We determined that when moving 5m in real world coordinates, the app's coordinates drifted by less than 3cm, which indicates a high degree of accuracy. As such, we recommend placing fiducials every 10m in the workspace to keep drift under 10cm at maximum.

Of note is that results vary to an extent depending on conditions. For example, in a room with many features and little movement, like the Project Lab, the app's coordinates retain near-perfect accuracy. However, in environments with many people or particularly blank walls, the app loses tracking much more easily.

User Interface

We had the chance to provide a live demonstration of our app at the Engineering Physics Project Fair, and thus also were able to gain some insight into the effectiveness of the user interface of the app. Some observations we made were as follows:

- The rendered 3D arrow seems to be an intuitive guide for the most part, but some users found it harder to use near the target (where small movements start to have significant effect on the arrow's direction). Additionally, if the arrow appeared pointing axially toward or away from the user, such that the shape of the arrow was less evident, the user often took longer to understand and respond.
- In the demonstration, we had to explain the need to calibrate position verbally. Ideally, this should be obvious from the design of the app; especially since a user may not know what markers to look for in a given workspace. Perhaps on startup, the app could prompt the user to calibrate their position by showing images of the fiducials to look for in the workspace.

Conclusions

In this project we aimed to identify and study augmented reality tools that can assist the navigation and organization of a physical workspace and to integrate such tools into a proof-of-concept mobile phone application. We have been successful in both of these pursuits.

Our app is built on ARCore, and can be deployed on most modern Android devices. ARCore's Augmented Images feature calibrates position by detecting an image of known location in frame. This detection is generally accurate and consistent enough to be satisfactory, though it preferably could be done from greater distances to put less burden on the user. Motion is then tracked using ARCore's SLAM algorithms, with precision exceeding our expectations, though it relies on having a good environment for tracking.

The app also provides a framework for integrating an object identification tool. The SIFT-based classifier currently in use is reasonably accurate for small sets of objects with many reference images. However, it is not currently reliable enough for actual deployment, and should be further developed or replaced.

A sample administrative workflow is in place, to demonstrate how AR functionality could streamline the process of setting up the app. However, the workflow is currently neither convenient nor secure.

All in all, the core technology of the application is in place, and we conclude that our app is able to address the sponsor's goal of improving organization in workspaces.

Recommendations

A few key risks remain in taking the project from its current prototype to a deployable state, which we recommend be investigated further.

1. **Quality and performance of object recognition.** As discussed in our testing, our current app is not able to consistently identify items. This may be improved by experimenting with image resolution, number of images, SIFT parameters, and the schema used to choose the best match. However, the necessity for SIFT to compare an image against every image in the database presents a fundamental limitation on the performance of the app as more items are added. We highly recommend looking into a few-shot neural network as that technology advances.
2. **Convenience of administrative workflow.** We currently have two options for loading objects (see Appendix A), but either of these has the potential to be extremely arduous for a large number of items, so we recommend researching a better approach.
3. **Security.** As the app is deployed, there may be a need to differentiate some users as administrators, and prevent non-admins from directly accessing and tampering with the database. Additionally, if the app has multiple clients, then admins for one database must not be able to view or interfere with another database. Some form of login system will need to be implemented, and the app would need to be rebuilt with security in mind or very carefully reviewed for security leaks.
4. **Moving to cross-platform.** If the app is to be deployed on workspace members' personal phones, we must also target the app toward Apple's iOS phones. This may involve rebuilding the app in ARKit (Apple's AR framework) or in a third-party framework that can be deployed to both platforms.
5. **Multiple marker detection.** We have designed our app under the assumption that in a large workspace, there will be multiple markers set up for position calibration. However, we have yet to specifically implement and test how the app responds to detecting a second marker. This would also require a set of images, each with high detectability (see Appendix D: 4.a for information on scoring images) but with mutually distinct features.
6. **User Experience.** This project has been a proof-of-concept, and was conducted with little consultation or market research. Some minor improvements we have thought of include prompting users with images of the markers in a workspace, to make localization more intuitive; and making adjustments to how the 3D arrow behaves near the target; but consultation with potential users or people experienced in user experience design would be much more useful.
7. **App Architecture.** As we had a focus on rapid development and were learning the tools as we developed the app, there are certainly ways the app should be structured differently. Work should be put into unifying the API, reducing exposure between classes, and improving performance of the app. See Appendix B for details on the current implementation.

Deliverables

The project can be found at the following (private) repository:
https://github.com/Combobulator-5000/combobulator

Appendices

Appendix A: App Usage Manual

Installation

---

1. Install Android Studio.
2. Clone the repo to your machine
    - Via the command line:
    - git clone https://github.com/Combobulator-5000/combobulator
    - Via Android Studio: File > New > Project from Version Control...
3. Open the project in Android Studio
4. Press the Gradle Sync button near the top right of the IDE.

To deploy this app on a physical android device, enable developer options and USB debugging. When you run the app via USB debugging, your phone will automatically install ARCore, which is a dependency of the project.

To run the app in an emulator, you must install the ARCore SDK on your emulated device.

User Workflow

On opening the app, you will notice a few key features:

- Main “Scan” button
- Top status bar
- “Show Debug” slider
- “Item List” button

Procedure

1. **Calibrate position.** Center the earth marker (see Appendix D 4.b), or some other image loaded into the app by an admin, in the camera frame. You may need to adjust the phone to reach an appropriate angle for detection. When the image is automatically detected, a frame being rendered around the image, and the “Position Calibrated” indicator in the top status bar will be checked.
2. **Set a target.** There are two approaches for this:
    - *By Classifier:* Center the object to be put away on screen, ideally on a blank background. Press “Scan” to classify the object and set the target.
    - *From List:* Press the “Item List” button to bring up a scrollable list of all objects in the database. Press “Locate” on the row of the item you want to find; this will bring you back to the main screen with the target set.

3. **Navigate to the target.** An arrow will render on screen, pointing in the direction of the target. Move to the indicated location; you will notice the progress bar in the top status bar increasing as you get closer.

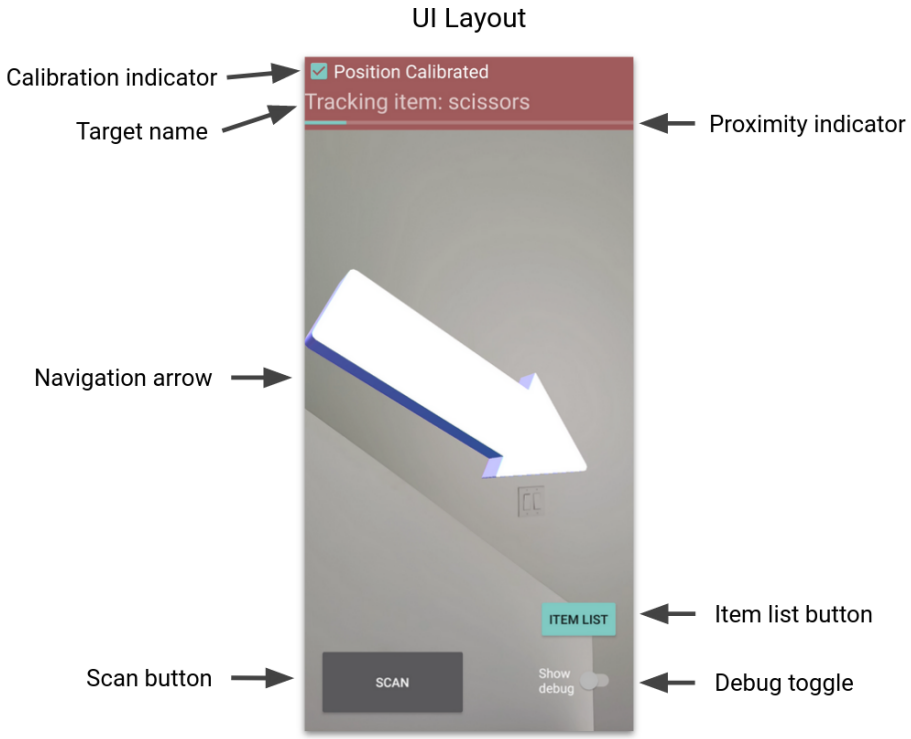


Figure 14. App UI Layout.

4. **Arrive at the target.** Within a set threshold of the target location, the arrow will disappear and the target will be unset.
    - If a “location reference image” has been set, then a dialog box will appear in the center of the screen with the image. This image may provide additional context on how to put away the object. The “Dismiss” button will close the dialog.

The debug screen contains the following information:

- The phone’s current absolute position, if calibrated
- The current item being tracked, and its absolute position
- The last image scanned
- Match scores between the last scanned image, and each image stored by each item in the database

Administrative Workflow

Some useful app parameters have been summarized in a file:
app/src/main/java/com/enph/plab/java/combobulator/Parameters.kt

Taking reference scans

For best performance:

- Scan objects against a plain background
- Take many (15+) images of each object from various angles
- If images are taken externally, scale down image resolution (e.g. to 640x480 pixels). Otherwise, computing SIFT keypoint descriptors takes much too long on startup, such that the app effectively crashes.

Setting up workspace

Currently, the app must be provided with marker absolute location data in a JSON workspace file. Items can be pre-loaded with items in two different methods: either from the same workspace file, or from a pre-existing MongoDB Realm database. The appropriate method can be selected by changing the *DATASOURCE* parameter in the Parameters.kt file.

While defining the workspace via JSON is convenient for testing purposes, a final deployable app should probably not use this method.

JSON Workspace

A sample workspace JSON file can be seen at app/src/main/assets/workspaces/default.json.

The file must contain two top-level JSON Arrays, “markers” and “objects.”
Each entry in “markers” must contain fields:

- “x”, “y”, “z”: coordinates of position of the marker in meters
- “qx”, “qy”, “qz”, “qw”: orientation in quaternion format
- “image”: path of the marker image relative to the assets folder

Each entry in “objects” must contain fields:

- “name”: a human-readable and uniquely identifiable tag for the item
- “x”, “y”, “z”: position where the item belongs in meters
- “Images_path”: path, relative to the assets folder, to a directory where reference scans of the item are stored

The path to the workspace file (relative to the assets folder) should be specified as the constant WORKSPACE_FILE in Parameters.kt.

The JSON file is currently packaged into the APK file as an asset, which means the app must be rebuilt and reinstalled each time the workspace is updated.

MongoDB Realm Database

The database is hosted on MongoDB Realm. We created a free tier MongoDB Atlas account. From the web UI, we created a Realm using the blank project setup dialogue. Of note:

- We allowed anonymous login (Realm > Data Access > Authentication > Allow users to log in anonymously)
- We enabled access from any IP address (Atlas > Network Access > Add IP Address “0.0.0.0/0”)

- We enabled Sync (Realm > Build > Sync)
    - Permissions: full read/write permission for all users (this is undesirable in production, but for our tests, it allowed us to get work done quickly). (Enter “true” and nothing else in both the “Read” and “Write” text fields)
    - Partition key: “deployment”. This is a string present in each entry in all collections that specifies where the data “belongs”. For example, the “deployment” value for objects in the project lab is “plab”. This can later be used for access control, so that users can only read or write to objects in their partition.
    - We enabled development mode. This allows the app to push data that does not match the remote schema, and in fact will cause MongoDB to reconfigure the schema to match the data we push. This is useful for development, as you can add new data without having to fuss with the schema.
- Finally, to connect the app to your Realm instance, you’ll need to update the App ID. This data does not need to be secured, as long as you have proper access control set up for the database. As such, we’ve left the old key in our GitHub repo. It can be found here: https://github.com/Combobulator-5000/combobulator/blob/1e0a3d131ce376d856c66b359cd6aefa123805e2/app/src/main/java/com/enph/plab/java/combobulator/Parameters.kt#L11
    - You can find your Realm’s App ID from Realm > (project name - first thing in the list below “realm apps”) > App ID

Editing Objects In-App

We have prototyped a workflow for adding or editing item profiles in-app, to make use of the localization component as a measuring tool and streamline the process of adding photos.

Note that in this mode, there is currently no way to delete objects or images; and there is not yet any way to permanently save changes made in the app to disk or database, so closing the app discards all changes.

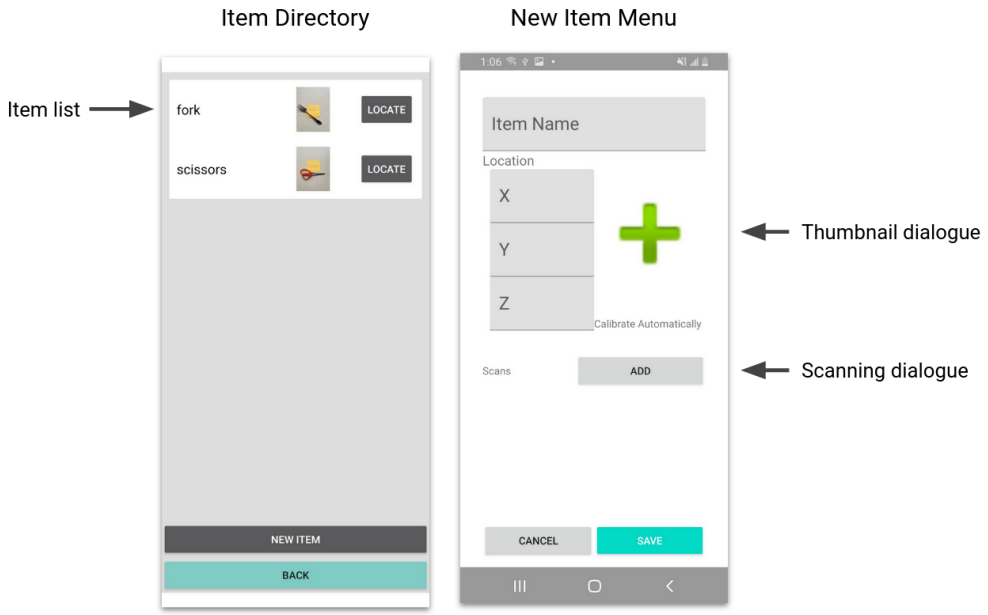


Figure 15. App administrative menu layout.

1. Open Item List
2. Click “New Item”, or click on a row in the item list (outside the “Locate” button)
3. Click on Item Name, or X, Y, Z fields to edit
4. Alternatively, automatically detect the item’s location as follows:
    a. Click on the image (or green plus button) to the right of the location fields. This will bring up the camera feed.
    b. If the location is not already calibrated, locate a calibration marker and ensure it has been detected.
    c. Go to the location where the item in question belongs. Set up a shot of the item appearing as it should when put away (e.g., if the item should be put away in a drawer, show the item in a
    d. Click “Scan.” This will automatically return you to the item editor screen. The location reference image will be set to the image just taken, and the X,Y,Z fields will be automatically populated with the location of the camera at that instant.
    e. Alternatively, press the Android back button to cancel and return to the editor screen without recording a new location.
5. Add scans to the item.
    a. Click “Add”. This will bring up the camera feed.
    b. Click “Scan” repeatedly to take pictures of the item from various angles.
    c. Press “Done” or android back button to exit; the new scans will be added to the list of images immediately below the “Add” button.
6. Click “Save” to push changes made in the editor screen to the item list. Clicking “Cancel” or the Android back button discard changes.

Appendix B: App Architecture and Classes

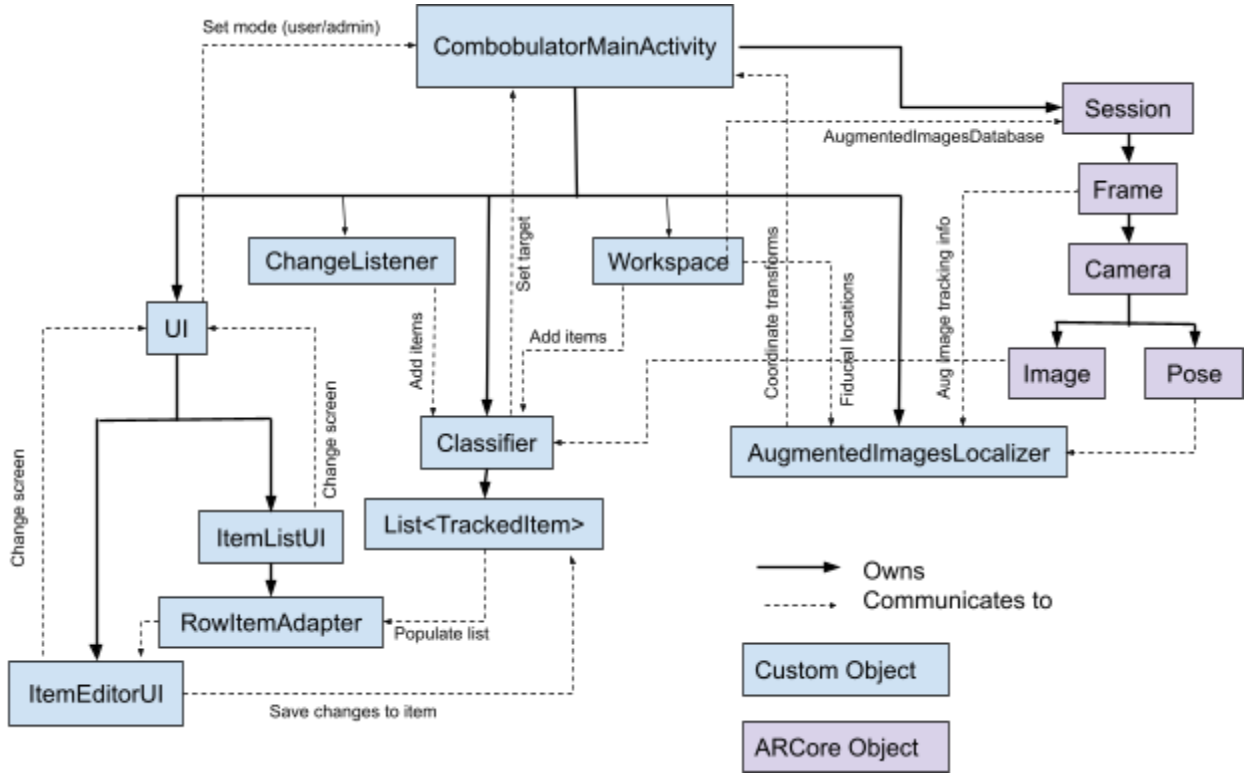


Figure 16. Map of main app classes.

CombobulatorMainActivity

The main object, that directs app functionality and interfaces between the ARCore session and our own code. Most notable methods are onCreate, which is called on app launch and is where we perform all setup operations; and onDrawFrame, which gets called at each frame of the program and handles the main information flow of the app.

Workspace

Handles parsing JSON files to read information about a workspace. This includes locations and image files for position calibration markers; as well as the names, locations, and images associated with each item.

AugmentedImagesLocalizer

Stores and updates a relation between the coordinate frames used by ARCore and the Workspace. It must be updated every frame with information about the camera's current position and new augmented images being tracked. It supports methods (convertToAbsPose and convertToFramePose) that support transforming points in space from ARCore's instantaneous coordinate system to the absolute (workspace) coordinate system, and vice versa, that may be safely called once the position has been calibrated.

TrackedItem
Represents a single object (e.g. tool) that can be located with the app. Stores its name, absolute location, and a list of scanned images of the object for use by SIFT. Optionally, it may also include a location reference image; this is an image demonstrating how the object should be put away, to be shown to the user upon arriving at the target location.

Classifier
Stores a list of TrackedItems associated with the active workspace, and computes SIFT descriptors for each image of each TrackedItem. When the evaluate method is called on an image, it computes descriptors for the incoming image, and predicts what kind of item it is by performing a Flann-based match against the stored descriptors. It currently selects the best match by

UI
A layer of abstraction between the activity and user interface. Some tasks handled by the UI object include:

- Recording “Scan” button presses to a queue, which may be polled and handled from other threads
- Communicating information about current app state from the OpenGL thread and UI thread
- Displaying “Fragments” (Appendix D 1.a) like the item list and item editor

Linked to the layout file app/src/main/res/layout/activity_main.xml.

ItemListUI
A UI Fragment (Appendix D 1.a) that displays a list (in the form of a RecyclerView) of TrackedItems stored by the Classifier.
The ItemListUI currently implements a SwipeRefreshLayout.onRefreshListener; this means that swiping down at the top of the list will call the method onRefresh. This is not currently implemented but could be used to request the cloud database for updates. The list could also be made searchable using an Android SearchView.
Linked to the layout file app/src/main/res/layout/item_list.xml.

RowItemAdapter
Populates each entry of the ItemListUI RecyclerView using the list of TrackedItems stored by the Classifier, and sets up basic operations that can be performed on entries (like setting the object as the main target by pressing “Locate”, or opening an editor screen for the given item by pressing elsewhere).
Linked to the layout file app/src/main/res/layout/row_item.xml.

ItemEditorUI
A UI Fragment (Appendix D 1.a) that provides an interface to display, edit and save fields of a TrackedItem in-app.

ChangeListener
Subscribes to changes on the database, and updates the classifier entries accordingly.

RealmTrackedItem
A clone of TrackedItem that also contains internal attributes needed for serialization and partitioning. It is likely possible to merge TrackedItem and RealmTrackedItem, however Realm demands that RealmObject instances (which RealmTrackedItem is) only be accessed from the thread on which they are created. This would mean that the ChangeListener would need to be moved to a background thread which would also be used for classification tasks.

Note on threading
The ARCore app we are building off of uses multiple threads: the UI thread for general UI operations (button presses, updating text, etc), and the OpenGL thread for the primary ARCore loop (i.e., the onDrawFrame method in CombobulatorMainActivity

Ideally, computationally heavy CPU operations should not be done on the OpenGL thread as this may affect the frame rate of the app. For example, we are currently running the Classifier's evaluate operation directly from the onDrawFrame method, which means that if matching against a large number of images, the app may freeze for a few seconds. This should be mitigated.

Some operations may only be performed on a specific thread, and will throw fatal errors if attempted on the wrong thread. For example, objects tracked by a MongoDB Realm database may only be accessed or edited from the thread on which they were created. (As a workaround, we have created a RealmTrackedItem class to receive or send objects to Realm, and a TrackedItem wrapper class that may be used elsewhere in the app.) Similarly, View objects (such as those defined in layout files) may only be updated on the UI thread.

Appendix C: Comparison of AR Platforms

	ARCore	ARKit	Unity AR Foundation	ARToolKit
Cross-platform			✔	✔
Plenty of available support	✔	✔	✔	
Convenience of development	✔		✔	
Ease of UI design	✔	✔		
App size	✔	✔		✔
Open Source	✔			✔

Primary decision-making factors:

- ARKit apps can only be developed on MacOS.
- ARToolKit does not have the same degree of community and documentation support, and is not ideal for first-time AR developers (though is gaining momentum).
- Unity is excellent for rendering 3D options but has poor UI design, and apps tend to be less lightweight; for this application it does not seem suitable.
- ARCore/Android development offers fine (possibly daunting) control over app design, but the primary development software (Android Studio)

Appendix D: Useful resources

1. Android Development Principles
    a. Guide to app architecture https://developer.android.com/jetpack/guide
    b. Fragments https://developer.android.com/guide/fragments
2. Recommended reading before getting started on AR
    a. https://developers.google.com/ar/develop/fundamentals
    b. https://developers.google.com/ar/develop/anchors
    c. https://developers.google.com/ar/develop/augmented-images
    d. https://developers.google.com/ar/develop/machine-learning
        - We were not able to get this working, but may be useful in the future as more focus is placed on object detection
3. ARCore sample projects
    a. https://github.com/google-ar/arcore-android-sdk/tree/master/samples
4. Augmented Images
    a. Database generator and scoring tool: https://developers.google.com/ar/develop/augmented-images/arcoreimg
    b. Source of earth image: https://developers.google.com/ar/images/augmented-images-earth.jpg
5. Rotation converter tool: https://www.andre-gaschler.com/rotationconverter/
    a. May be useful for determining marker orientation in a human-understandable format (e.g. axis-angle) and then converting to quaternions for use in app
6. AR Frameworks
    a. ARToolKit5 Source Code: https://github.com/artoolkit/ARToolKit5
    b. ARCore Source Code: https://github.com/google-ar/arcore-android-sdk
    c. ARKit Documentation: https://developer.apple.com/documentation/arkit
7. ArUco
    a. Aruco marker generator: https://chev.me/arucogen/
    b. Standard charuco calibration board: https://docs.opencv.org/3.4/charucoboard.jpg
    c. Charuco board generator: https://calib.io/pages/camera-calibration-pattern-generator

Appendix E: Discussion of Aruco

We had considered using Aruco for absolute localization, and it may still be an effective approach. However, we encountered a number of challenges in the process.

- Aruco is currently part of the OpenCV contrib [7] repository. While it is possible to compile a custom build for Android, we found this process to be complicated and difficult to debug, and most tutorials online are outdated. We eventually found the quickbirdstudios pre-compiled OpenCV contrib builds for Android [8], which could be included in the app by adding the following line to the module build.gradle file:
    - `implementation \`com.quickbirdstudios:opencv:VERSION\``
- Unlike ARCore, OpenCV does not have innate access to internal camera parameters, which affect how the position of a fiducial is estimated based on the location of the fiducial in the image. These must be retrieved from the device or manually calibrated. We found calibration to be erratic (though this may be resolved now).
- Not much documentation is provided for the Java version of OpenCV, particularly the contrib modules, and it is not immediately obvious how types and arguments translate. Some methods worked as we expected (for example, calibrateCameraCharuco) while some threw incomprehensible errors from the native C++ code when running in Java while the Python counterpart worked smoothly (for example, calibrateCameraAruco).
- When estimating the pose of a fiducial, Aruco appears to use Rodriguez vectors for orientation, which are not particularly common. Converting the raw Aruco output from estimatePoseSingleMarkers to a format usable by ARCore may not be simple. We avoided this task in our initial investigation of Aruco by making the assumption that the camera would be approximately square to the fiducial whenever close enough to scan it.

It may now be easier to implement an Aruco-based solution:

- We discovered near the end of the project that the images retrieved from an ARCore Frame are in YUV format. Attempting to interpret a YUV image as RGB/BGR format results in curious behaviour, including that the image appears to have four smaller grayscale, partially transparent copies of the desired image lined across the top. This may have been a major contributor to positional error in our earlier trials of Aruco, since this would cause four fiducials to be detected in the image per single real fiducial in frame, and the positions within the camera frame would not be accurate. We have since worked out a conversion from YUV to RGB.
- We have implemented a method (displayImage) in our main repository that is able to display an OpenCV Mat to an Android ImageView object. This may simplify debugging.
- This file contains our previous attempts at an Aruco solution, and may prove a useful reference: (note, this repository is also private to the Combobulator-5000 organization) https://github.com/Combobulator-5000/ARCore-Starter/blob/6bb8b9389b5d9966b27878c8784465ba60cdb1dc/app/src/main/java/com/google/ar/core/examples/kotlin/helloar/Localizer.kt

Appendix F: Glossary

Android	An open-source operating system used for smartphones and tablet computers.
APK	The Android Package with the file extension apk is the file format used by the Android operating system for distribution and installation of mobile apps.
AR (Augmented Reality)	A technology that superimposes a computer-generated image on a user's view of the real world, thus providing a composite view.
ARCore	An AR development framework developed by Google for making Android apps with augmented reality features.
ArUco	A computer vision library specializing in detecting and estimating the relative position of simple QR-code-like fiducials.
Asset	A file in an android app that is packaged into the APK at compile time, and is accessed in-app via the AssetManager object.
Augmented Image	An ARCore feature that can recognize pre-registered 2D images in the real world and anchor virtual content on top of them
Calibration	The process of determining a relation between room space and AR space by detecting a known point,
Classification	Associating a scan of an object with a label of what it is through computer vision.
Descriptors	A SIFT descriptor is a computed characterization of the appearance of a keypoint.
Few-shot learning	Few-short learning is a machine learning training strategy that aims to build accurate machine learning models with less training data than traditional methods in order to minimize training time.
Fiducial	A standard reference image used to calibrate the user’s position.
FLANN-based matching	FLANN (Fast Library for Approximate Nearest Neighbors) is an algorithm for quickly determining the closest potential matches between keypoints in different images.
Gradle	A compiler tool integrated with Android Studio that helps manage app configurations and dependencies.
Identification	The process of classify an unknown object is and where it should be located in the workspace.
iOS	An operating system used for mobile devices manufactured by Apple.
Item	Something in the workspace that is registered to a designated location and that can be identified by the app.
JSON (JavaScript Object Notation)	JSON is a standard file format that uses readable text to store and transmit data.
Keypoints	Keypoints are the prominent corners of an object from which we can uniquely identify the object.
Localization	The process of determining the user’s location in the workspace.
Marker	A standard reference image used to calibrate the user’s position.

MongoDB Atlas	A cloud database service.
MongoDB Realm	A serverless data retrieval framework to assist integrating a database like MongoDB Atlas into a mobile application.
Navigation	The process of guiding the user through the workspace to put away an object.
Neural Network	A computer system modeled on the human brain. Used to perform tasks that humans excel with - especially based on pattern recognition.
Object	Something in the workspace that is registered to a designated location and that can be identified by the app.
OpenCV	OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.
OpenGL	OpenGL (Open Graphics Library) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics.
Project Lab	The UBC Engineering Physics Project Lab. Operates a variety of engineering student projects in a lab space.
RGB	The RGB color model is an additive color model in which the red, green, and blue primary colors of light are added together in various ways to reproduce a broad array of colors.
SIFT (Scale Invariant Feature Transform)	Scale invariant feature transform is a computer vision algorithm to detect and identify features and keypoints in images.
SLAM (Simultaneous Localization and Mapping)	Simultaneous localization and mapping algorithms are a class of algorithms used to track an agent’s position given live sensor data.
Training	The process by which a machine learning model is fed data for it to learn from.
Workspace	The environment the app should keep organized.
YOLO (You only look once)	YOLO is a pre-trained neural network that can identify a wide class of common objects.
YUV	YUV is a color model comprised of one luminance component (Y), and two chrominance components, called U (blue projection) and V (red projection) respectively.

References

[1] Google (2021). ARCore Android SDK v1.29.0 (Source Code). Retrieved April 2022, from https://github.com/google-ar/arcore-android-sdk/tree/6b01498cbd

[2] Google (2022). *Fundamental concepts*. ARCore. Retrieved April 2022, from https://developers.google.com/ar/develop/fundamentals

[3] Google (2022). *Pose | ARCore Reference*. Retrieved April 2022, from https://developers.google.com/ar/reference/java/com/google/ar/core/Pose

[4] Kulich, T. (2019). Indoor navigation using vision-based localization and augmented reality (Dissertation). Retrieved April 2022, from http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-391930

[5] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision* (Vol. 2, pp. 1150–1157).

[6] Makarov, A. (2021, December 24). *Augmented reality navigation & indoor positioning guide*. MobiDev. Retrieved April 2022, from https://mobidev.biz/blog/augmented-reality-indoor-navigation-app-developement-arkit

[7] OpenCV (2022). OpenCV Contrib (Source Code). Retrieved April 2022, from https://github.com/opencv/opencv_contrib

[8] QuickBird Studios (2021). OpenCV Android (Source Code). Retrieved April 2022, from https://github.com/quickbirdstudios/opencv-android

[9] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). *You Only Look Once: Unified, Real-Time Object Detection*. doi:10.48550/ARXIV.1506.02640

[10] Wang, Y., Yao, Q., Kwok, J., & Ni, L. M. (2019). *Generalizing from a Few Examples: A Survey on Few-Shot Learning.* doi:10.48550/ARXIV.1904.05046

Images

- Fig. 2 (Pokemon Go): https://www.ign.com/articles/2017/12/20/pokemon-go-introduces-apple-exclusive-ar-mode-with-proximity-sensing-new-catch-bonuses
- Fig. 2 (Google Street View): https://www.digitaltrends.com/mobile/how-to-use-ar-mode-google-maps/
- Fig. 5 (SIFT keypoints): https://www.cc.gatech.edu/~hays/compvision2018/proj2/

- Fig. 6 (Earth): https://developers.google.com/ar/images/augmented-images-earth.jpg
- Fig. 10 (Wrench icon): https://www.flaticon.com/authors/good-ware