1. INTRODUCTION
2. LITERATURE REVIEW
3. PROJECT MANAGEMENT
4. REQUIREMENT COLLECTION AND ANALYSIS
   1. Interview
   2. Persona
   3. USU
5. SYSTEM DESIGN
   1. Signal Image and live stream
   2. Sequence diagram
   3. Use case diagram
   4. Class diagram
6. SYSTEM IMPLEMENTATION
7. TESTING
8. CONCLUSION AND FUTURE WORK

This design project attempts to create a mobile phone application that seamlessly integrates the digital world with the real world by allowing users to measure the distance between them and people around them. This application requires the integration of many different components to produce an optimal user experience. The main components of the proposed design include geolocation, image processing, user interface design, software design and implementation on the Android and a website.

Applications:

1. Live Stream, measure the distance based on 2D understanding
   1. Android
   2. Web Site (Flask)
2. Single Image, Measure distance based on 3D understanding
   1. Android

CHAPTER 4 REQUIREMENT COLLECTION AND ANALYSIS

4.1 Foundational Research:

Foundational research is done before start designing. Within the product development life cycle, foundational research happens during the first stage to help empathize with users, understand their needs, and inspire new design directions. During this stage, personas and user stories are made. An Interview with Dr. Ali Alsfar been done to collect in-depth information on opinions, thoughts, experiences, and feelings about the application.

<https://www.coursera.org/learn/start-ux-design-process/supplement/IMKD6/learn-more-about-ux-research>

4.2 Design Research:

Within the product development lifecycle, design research happens during the design stage to help inform the designs, to fit the needs of users, and to reduce risk. One of the common methods used to conduct design research is a usability study, which is a technique to evaluate a product by testing it on users. The goal of usability studies is to identify pain points that the user experiences with your prototypes, so the issues can be fixed before the product launches.

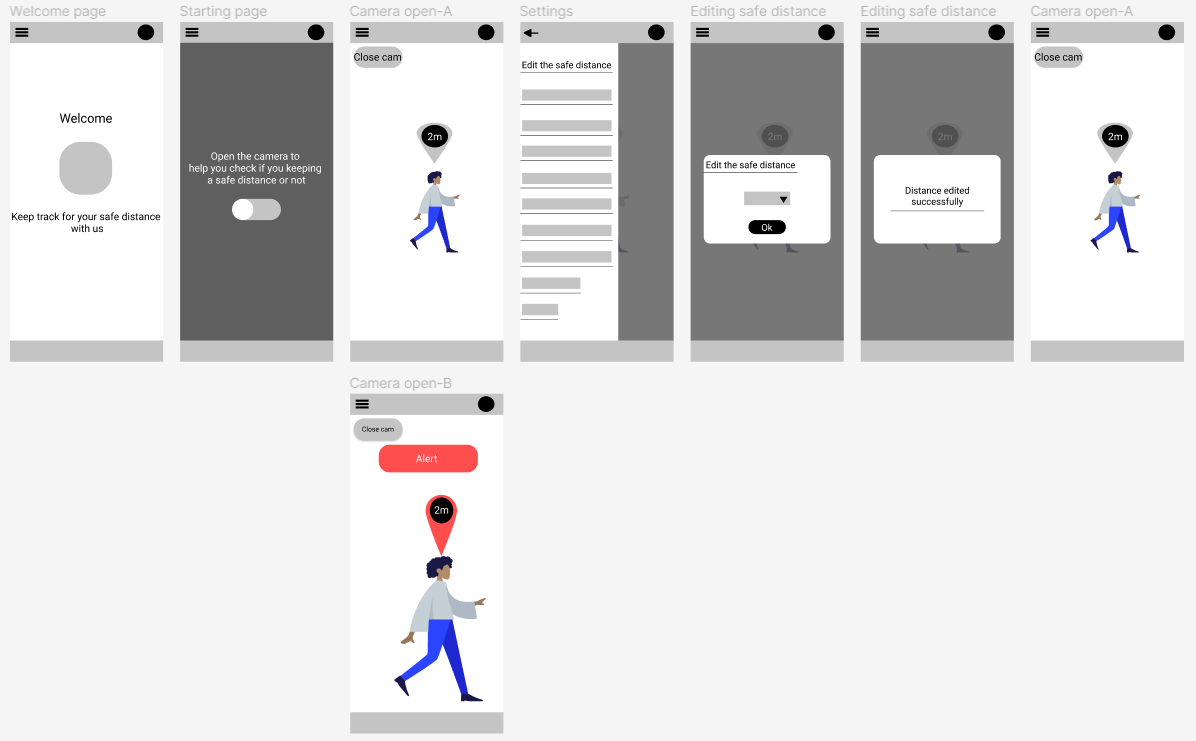
<https://www.coursera.org/learn/start-ux-design-process/supplement/IMKD6/learn-more-about-ux-research>

Two rounds of usability studies were conducted, the first was moving from digital wireframes to mockups, the designs were inspired by the results of the low-fidelity prototype. The second research showed which elements of the mockups needed to be improved through the usage of a high-fidelity prototype.

|  |  |
| --- | --- |
| **Introduction** | * **Title:** Safe distance tracker * **Date**: 6/20/2022 * **Project background**: We’re creating a new app to help people keep a safe distance during COVID or in any situation might need to track their safe distance. * **Research goals**: Track the safe distance of the user, how will the results of the research affect your design decisions? Help to provide a more enjoyable and interactive experience for the users |
| **Research**  **questions** | * How long does it take for a user to check if he/she keeps a safe distance * What can we learn from the steps that users take to check the safe distance? * Are there any parts where users are getting stuck? |
| **Key Performance Indicators**  **(KPIs)** | * User error rate: if there any issues with the AR application * System usability scale: if app perform as expected |
| **Methodology** | * Unmoderated usability study * **Location**: Bahrain, remote (each participant will complete the study in their own home) * **Date**: Sessions will take place on March 12 (normal business hours) and March 13 (after hours) * **Length**: Each session will last 5 to 10 minutes, based on a list of prompts * 6 participants will use the app. Each participant will then complete a SUS on their experience. |
| **Participants** | * Participants should include children between 6 to 16 years old * Participants need to be interested in keeping a safe distance during COVID or any situation similar to that. * Participants should include:   + 2 girls   + 2 boys   + 2 adults (male and female) |
| **Script** | **During the unmoderated usability study**  A list of prompts appears on the device screen   * **Prompt 1:** Prepare your device to start the app   + Prompt 1 Follow-Up: How easy or difficult was it to do the task? Is there anything you would change about the process? * **Prompt 2:** Set your safe distance that you want to keep.   + Prompt 2 Follow-Up: How easy or difficult was it to do the task? Is there anything you would change about the process? * **Prompt 3:** Follow the guides to help you to be in safe distance.   + Prompt 3 Follow-Up: How easy or difficult was it to do the task? Is there anything you would change about the process? What do you think about the alert? * **Prompt 4:** Close the app.   + Prompt 4 Follow-Up: How easy or difficult was it to do the task? Is there anything you would change about the process?   **After the unmoderated usability study**  Participants will complete the System Usability Scale   * Participants will score the following ten statements by selecting one of five responses that range from “Strongly Disagree” to “Strongly Agree.”   + The border of my safe distance was clear (only for those does NOT have a vision disability)   + The sound of the guider was clear   + I think that I would use this app frequently.   + I find the app unnecessarily complex.   + I think the app is easy to use.   + I need the support of a technical person to be able to use this app.   + I find the app easy to navigate.   + There is inconsistency within the app.   + I imagine that most people would learn to use this app quickly.   + I feel confident using the app.   + I feel happy using the app.   + I need to learn a lot of things before I can start using this app. |

4.2.1 Low-Fidelity Prototype Usability Study

Low-fidelity (lo-fi) prototyping is a quick and easy way to translate high-level design concepts into tangible and testable artifacts. It helps in checking and test functionality rather than the visual appearance of the product



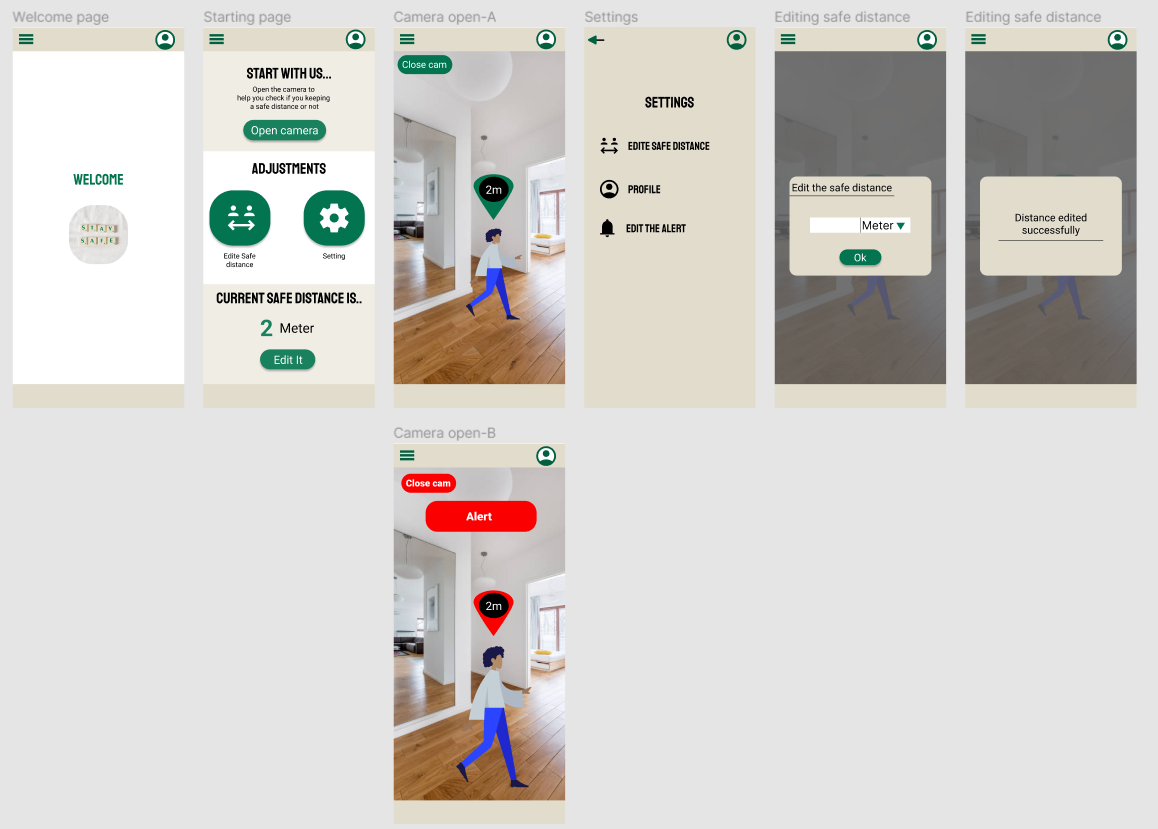
Click path is a record of what path the participant took to complete the task. Observations are note about behaviors, opinions, and attitudes along with any errors, issues, or areas of confusion. Quotes are any significant quotes (positive and negative). In task completion: **1** stands for easy to complete, **2** stands for completed but with difficulty and **3** stands for not completed.

Example of the low-fidelity usability study notes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task** | **Click Path** | **Observations** | **Quotes** | **Task Completion** |
| Prompt 1: open the camera from the app | click on the button | the participant finished the task fast and smoothly | "It is the only button I can see" | 1 |
| Prompt 2: trach your safe distance | moving around, while the camera is open | The participant got confused since the prototype does not provide a live camera scene | N/A | 2 |
| Prompt 3: change the safe distance | the options icon> change the safe distance | the participant got confused at the beginning, but tried to change the safe distance more than one time | "if I can specify the unit would be better" | 2 |
| prompt4: violate your safe distance to active the alert | letting a person enter in the indicated safe zone | the participant understood the task clearly and implemented it smoothly | N/A | 1 |

4.2.2 High-Fidelity Prototype Usability Study

high-fidelity prototypes are highly functional and interactive. They are very close to the final product, with most of the necessary design assets and components developed and integrated. It helps in test usability and identify issues in the workflow.



Example of the high-fidelity usability study notes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task** | **Click Path** | **Observations** | **Quotes** | **Task Completion** |
| Prompt 1: open the camera from the app | click on the button | the participant hovered the button then clicked on it | "It is not clear do I hover or I click" | 2 |
| Prompt 2: trach your safe distance | moving around, while the camera is open | The participant got confused since the prototype does not provide a live camera scene | N/A | 2 |
| Prompt 3: change the safe distance | N/A | the participant did not understand the meaning of the icons the participant was pressing on the line and the number that shows the current safe distance | "I can not see any setting or options" | 3 |
| prompt4: violate your safe distance to active the alert | letting a person enter in the indicated safe zone | the participant understood the task clearly and implemented it smoothly | N/A | 1 |

4.2.3 Results of Usability Studies

After conducting two round of usability studies, three main insights were derived and needed changes was applied on the prototype. The insights are:

1. Users need to be clear what unit measurement we are using, and changeable.
2. Users need more interactive and noticeable ways to be alerted when safe distance violation happens
3. Users need easier ways to access setting/change the safe distance.

4.3 Application Requirements

The interview and usability studies set clear expectations that were then converted into concrete functional and non-functional requirements.

4.3.1 Functional Requirements

Functional requirements are described as the way the system will behave when certain input criteria are met (ReQtest, 2012).

1. Measure the distance using phone camera
2. Detect human using phone camera
3. A noticeable alert when the distance between a detected person and user are less than certain value
4. Ability to change the safe distance
5. Ability to change the measuring unit

4.3.2 Non-functional Requirements

Non-functional requirements address all requirements that haven’t been included in the functional requirements. They refer to the criteria for judging a system's functionality rather than behaviors (ReQtest, 2012).

Non-functional requirements

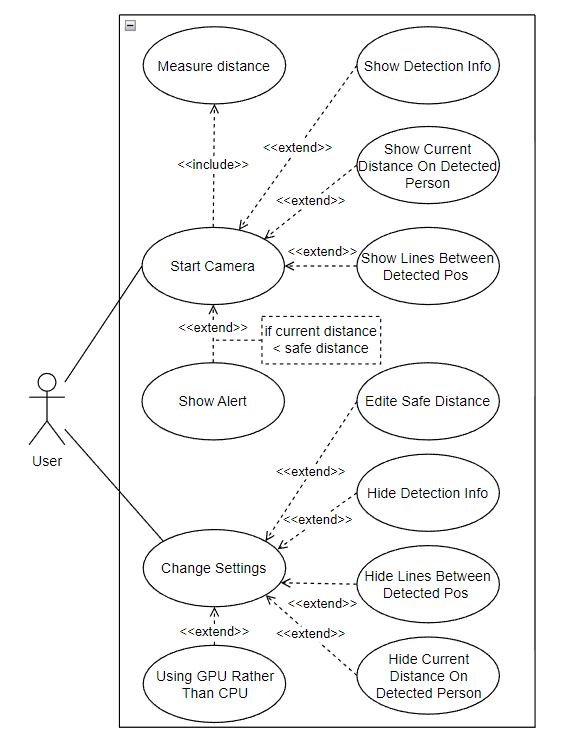
|  |  |
| --- | --- |
| **Non-functional requirement** | **Description** |
| Performance | The system responds quickly and efficiently to user’s inputs |
| Usability | The system is easy to use and navigate for first-time users |
| Scalability | The system is easy to add more functions to for enhancement |

CHAPTER 5 SYSTEM DESIGN

Unified Modeling Language (UML) is a standardized modeling language enabling developers to specify, visualize, construct, and document artifacts of a software system.

**5.1 Use case diagram**

Use case diagrams are used to show the usage requirements for a system. UML Use Cases are used for actual development to provide significantly more value because they describe the core of the requirements.



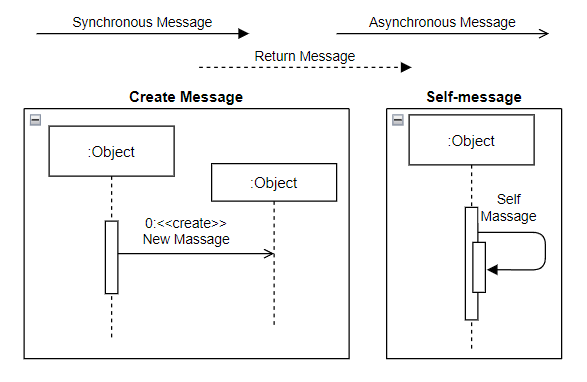
Start Camera: The user starts the camera device to measure the distance between the detected object and the camera. The user can show lines between person poses and the current distance between the detected person on this person. The user can show the detection information, like frame rate, image size, and detection latency. If the current distance between the detected person and the camera is less than the safe distance, an alert will be shown.

Change Settings**:** The user can hide detection information, lines between person poses, and the current distance that is rendered on the detected person. The user can use the device's GPU rather than CPU.

5.2 Sequence diagram

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time. They're also called event diagrams. A sequence diagram is a good way to visualize and validate various runtime scenarios.

5.2.1 Sequence diagram components



**Synchronous Message**  
A synchronous message requires a response before the interaction can continue. It's usually drawn using a line with a solid arrowhead pointing from one object to another.

**Asynchronous Message**  
Asynchronous messages don't need a reply for interaction to continue. Like synchronous messages, they are drawn with an arrow connecting two lifelines; however, the arrowhead is usually open and there's no return message depicted.

**Return Message**  
A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

**Self-Message**  
A message an object sends to itself, usually shown as a U-shaped arrow pointing back to itself.

**Create Message**  
This is a message that creates a new object. Similar to a return message, it's depicted with a dashed line and an open arrowhead that points to the rectangle representing the object created.

<https://www.smartdraw.com/sequence-diagram/>

<https://www.researchgate.net/figure/Sequence-diagram-of-face-recognition-system_fig4_321166919>

5.2.2 Application Sequence diagram

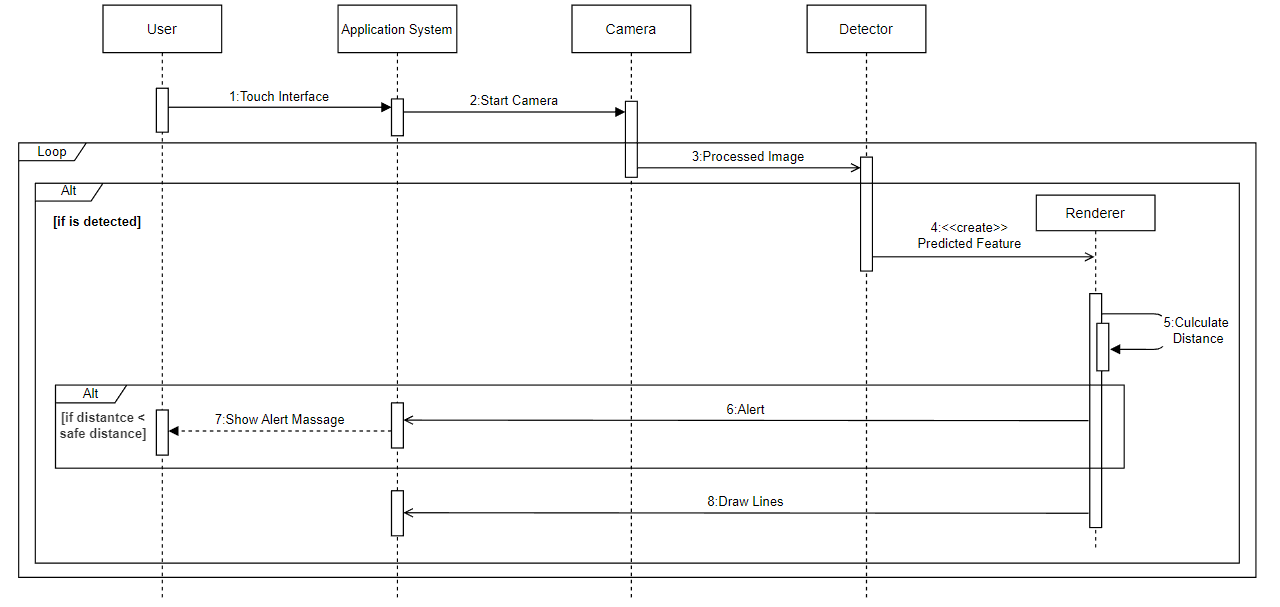


Figure X shows the four main objects used in the Measuring distance application. All four objects depend on user events. The application system the main logic that controls the application settings, permissions, and what the user sees. The camera object is responsible for accessing and communicating with the device camera. The detector object is responsible for detecting extracted features and getting their orientation in the image. The renderer object is responsible for calculating the distance between the past orientations and drawing lines between detected features.

Step 1: The user touches the screen to start the application.

Step 2: The application system opens the device camera.

Step 3: The camera gets live streaming data and processes them to match the detector input.

Step 4: If there is a detected object, the renderer gets created and receives the orientations of the detected object.

Step 5: The renderer calculates the distance between the detected object and the camera device.

Step 6: In case the calculated distance (in step 5) is less than the safe distance, the renderer will send an alert to the application system.

Step 7: the application system shows the user an alert message.

Step 8: If there is a detected object, The renderer draws lines on the detected person.

CHAPTER 6 SYSTEM IMPLEMENTATION AND TESTING

System implementation is the process of specifying how the information system should be developed, verifying that the information system is operational and being utilized, and ensuring that the information system fulfills quality standards. Testing occurs after the final integrations between the front-end and back-end have been made. It ensures that the system is properly running and displaying the necessary output.

6.1 Software Tools

1. Visual Studio Code

Visual Studio Code, also known as VS Code, is a free and open-source editor. It is compatible on many platforms and operating systems including Linux, macOS, and Windows. It also supports numerous programming languages such as Python, HTML, PHP, GO and many more. VS Code allows you to add extensions such as debuggers which improve user experience (Mustafeez).

1. Figma

Figma is a free, online user interface design application. It helps developers better visualize the system and its flow by creating an interactive design. It runs entirely on a browser which means that projects will be easily accessible anytime, anywhere (themejunkie).

1. GitHub

GitHub is a version control and collaboration tool for programming. It allows people to easily collaborate on projects from any location (GithubDocs).

1. Android Studio

Android Studio is the official integrated development environment for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for 48 Android development. It is available for download on Windows, macOS, and Linux based operating systems. We used Android Studio to build our application into Android devices.

1. CameraX

CameraX is an Android library makes complex camera functionality available in an easy-to-use API. It emphasizes use cases, which allow you to focus on the task you need to get done instead of managing device-specific nuances. Most common camera use cases are supported, Image analysis to access a buffer seamlessly for use in your algorithms, such as to pass to ML Kit, Image capture to save images and Video capture to save video and audio. In our project, we used two of the supported use cases, preview and image analysis. Preview use case allows view an image on the display. Image analysis use case provides the access a buffer seamlessly for use in machine learning detector.

<https://android-developers.googleblog.com/2022/08/camerax-12-is-now-in-beta.html>

1. ML Kit

ML Kit is a mobile SDK that enables you to add powerful machine learning features to a mobile application. It supports both Android and iOS and offers the same features for both platforms. The SDK is part of Firebase and bundles together various machine learning technologies from Google. This SDK comes with a set of ready-to-use APIs for common mobile use cases such as face detection, text recognition and pose detection.

* Pose Detector

The ML Kit Pose Detection API is a solution to detect the pose of a subject's body in real time from a continuous video or static image. A pose describes the body's position at one moment in time with a set of skeletal landmark points. The landmarks correspond to different body parts such as the shoulders and hips. We used the distance between shoulders and hips to get the distance between the device camera and detected person. Pose detection can only detect one person in an image. If two people are in the image, the model will assign landmarks to the person detected with the highest confidence, this will cause an issue will be discussed in …

<https://auth0.com/blog/a-look-at-android-ml-kit-the-machine-learning-sdk/>

<https://developers.google.com/ml-kit/vision/pose-detection>

1. TensorFlow Hub
2. OpenCV
3. Flask

6.2 Programing Languages

1. Java

Java is a programming language and computing platform first released by Sun Microsystems in 1995. It has evolved from humble beginnings to power a large share of today’s digital world, by providing the reliable platform upon which many services and applications are built. New, innovative products and digital services designed for the future continue to rely on Java, as well.

<https://www.java.com/en/download/help/whatis_java.html>

1. Kotlin

Kotlin is a general purpose, free, open source, statically typed “pragmatic” programming language initially designed for the JVM (Java Virtual Machine) and Android. It combines object-oriented and functional programming features. Kotlin originated at JetBrains, the company behind IntelliJ IDEA, in 2010, and has been open source since 2012. It is focused on interoperability, safety, clarity, and tooling support.

<https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html>

1. Python

Python programming language was released in 1991. It is amongst the most common languages in programming field. Python is considered as a powerful high-level language. It is used to develop many applications and specially software and web development. Python utilizes many operating systems such as UNIX, MAC OS, windows and others. Python is an easy-to-use language. Moreover, it is a beginner friendly language. It contains many features such as it is an object-oriented, opensource, and fully supported programming language. Currently, Python is one of the mostly used programming language in the world as it has numerous numbers of libraries and module. Python is being utilized in so many technologies starting from web development and all the way to machine learning (TechTarget, 2021).

6.3 Development

6.3.1 CameraX Use Cases

First, we create a CameraX instance, `cameraProvider`, and bind it to the lifecycle of the application. The `cameraProvider` will be used to bind the CameraX use cases to the application lifecycle. We use ViewModelProvider which provides ViewModels. ViewModel is a class that is responsible for preparing and managing the data for an Activity. Then, we use CameraXViewModel class which is responsible for interacting with CameraX to access processCameraProvider. The processCameraProvider is a method of the CameraXViewModel class that returns the LiveData that can be observed over the application lifecycle.

```java

private var cameraProvider: ProcessCameraProvider? = null

override fun onCreate(savedInstanceState: Bundle?) {

    ViewModelProvider(this, ViewModelProvider.AndroidViewModelFactory.getInstance(application))

        .get(CameraXViewModel::class.java)

        .processCameraProvider

        .observe(

            this,

            Observer { provider: ProcessCameraProvider? ->

            cameraProvider = provider

            bindAllCameraUseCases()

            }

        )

}

private fun bindAllCameraUseCases() {

    if (cameraProvider != null) {

        // As required by CameraX API, unbinds all use cases before trying to re-bind any of them.

        cameraProvider!!.unbindAll()

        bindPreviewUseCase()

        bindAnalysisUseCase()

    }

}

```

We create the builder that will be used to create the preview use case. We can attach some settings to the builder like the image resolution, that can be changed by the user. The previewUseCase is of type Preview which is a camera preview stream for displaying on-screen. Then we attach to the use case a Surface Provider. We are using previewView which is a custom View that displays the camera feed for CameraX's Preview use case. The previewView class manages the Surface lifecycle, as well as the preview aspect ratio and orientation. Internally, it uses either a TextureView or SurfaceView, classes in Open Graphics Library (OpenGL) renderer, to display the camera feed. Finally, we bind the use case to CameraX instance, cameraProvider.

```java

private var previewView: PreviewView? = null

private var previewUseCase: Preview? = null

private var cameraSelector: CameraSelector? = null

private fun bindPreviewUseCase() {

    val builder = Preview.Builder()

    // attach the settings of images resolution

    previewUseCase = builder.build()

    previewUseCase!!.setSurfaceProvider(previewView!!.getSurfaceProvider())

    cameraProvider!!.bindToLifecycle(/\* lifecycleOwner= \*/ this,

      cameraSelector!! /\* determines the camera lens to use.\*/,

      previewUseCase)

}

```

First, we create the analyzing use case and setts the executor, main thread, and its analyzer. The analyzer is where the pose detection, tracking, and rendering happens using ML Kit. The analyzer provides the input for the image processor as imageProxy. An image Proxy is a single image buffer.

```java

private var analysisUseCase: ImageAnalysis? = null

private var imageProcessor: VisionImageProcessor? = null

private fun bindAnalysisUseCase() {

    imageProcessor = PoseDetectorProcessor()

    val builder = ImageAnalysis.Builder()

    analysisUseCase = builder.build()

    analysisUseCase?.setAnalyzer(

      ContextCompat.getMainExecutor(this),

      ImageAnalysis.Analyzer {imageProxy: ImageProxy ->

        imageProcessor!!.processImageProxy(imageProxy, graphicOverlay)

      }

    )

    cameraProvider!!.bindToLifecycle(/\* lifecycleOwner= \*/ this, cameraSelector!!, analysisUseCase)

}

```

6.3.2 ML Kit Object detection, tracking and rendering

We set the options of the poseDetectorProcessor in the CameraXLivePreviewActivity, then pass them to PoseDetectorProcessor. PoseDetectorProcessor is an instance of VisionProcessorBase that extracts the result of the detected poses and classifies them. The pose detection and tracking process are done in PoseDetectorProcessor and its patent VisionProcessorBase. The result is of a type Pose and it is passed to the PoseGraph file. The Pose class has a method called getAllLandmark that returns a list of all detected landmarks. The getAllLandmark method will be used to draw some landmarks between the poses that are used to measure the distance. All those tasks are done asynchronously, and their results are handled using listeners.

class CameraXLivePreviewActivity :

  AppCompatActivity(), CompoundButton.OnCheckedChangeListener {

    // to give the user the choice of using the CPU or GPU

    val poseDetectorOptions =

        PreferenceUtils.getPoseDetectorOptionsForLivePreview(this)

    val shouldShowInFrameLikelihood =

        PreferenceUtils.shouldShowPoseDetectionInFrameLikelihoodLivePreview(this)

    val visualizeZ = PreferenceUtils.shouldPoseDetectionVisualizeZ(this)

    val rescaleZ =

        PreferenceUtils.shouldPoseDetectionRescaleZForVisualization(this)

    val runClassification =

        PreferenceUtils.shouldPoseDetectionRunClassification(this)

    imageProcessor = PoseDetectorProcessor(

            this,

            poseDetectorOptions,

            shouldShowInFrameLikelihood,

            visualizeZ,

            rescaleZ,

            runClassification,

            /\* isStreamMode = \*/ true

            )

  }

```java

class PoseDetectorProcessor(

    private val context: Context,

    options: PoseDetectorOptionsBase,

    private val showDistance: Boolean,

    private val visualizeZ: Boolean,

    private val rescaleZForVisualization: Boolean,

    private val runClassification: Boolean,

    private val isStreamMode: Boolean,

) : VisionProcessorBase<PoseDetectorProcessor.PoseWithClassification>(context) {

    private val detector: PoseDetector

    private val classificationExecutor: Executor

    private var poseClassifierProcessor: PoseClassifierProcessor? = null

    /\*\* Internal class to hold Pose and classification results. \*/

    class PoseWithClassification(val pose: Pose, val classificationResult: List<String>)

    init {

        detector = PoseDetection.getClient(options)

        classificationExecutor = Executors.newSingleThreadExecutor()

    }

    override fun stop() {

        super.stop()

        detector.close()

    }

    /\*

    \* continueWith():

    \* @param  Executor

    \* @param  Continuation: A function that is called to continue execution after completion of a Task.

    \* \*/

    override fun detectInImage(image: MlImage): Task<PoseWithClassification> {

        return detector

        .process(image)

        .continueWith(

            classificationExecutor

        ) { task ->

            val pose = task.getResult()

            var classificationResult: List<String> = ArrayList()

            if (runClassification) {

            if (poseClassifierProcessor == null) {

                poseClassifierProcessor = PoseClassifierProcessor(context, isStreamMode)

            }

            classificationResult = poseClassifierProcessor!!.getPoseResult(pose)

            }

            PoseWithClassification(pose, classificationResult)

        }

    }

    override fun onSuccess(

        results: PoseWithClassification,

        graphicOverlay: GraphicOverlay,

    ) {

        graphicOverlay.add(

        PoseGraphic(

            graphicOverlay,

            results.pose,

            showDistance,

            visualizeZ,

            rescaleZForVisualization,

            results.classificationResult

        )

        )

    //    Log.i("test", (results.pose.getPoseLandmark(0)?.position3D).toString());

    }

    override fun onFailure(e: Exception) {

        Log.e(TAG, "Pose detection failed!", e)

    }

    override fun isMlImageEnabled(context: Context?): Boolean {

        // Use MlImage in Pose Detection by default, change it to OFF to switch to InputImage.

        return true

    }

    companion object {

        private val TAG = "PoseDetectorProcessor"

    }

    }

```

```java

class PoseGraphic

internal constructor(

  overlay: GraphicOverlay,

  private val pose: Pose,

  private val showDistance: Boolean,

  private val visualizeZ: Boolean,

  private val rescaleZForVisualization: Boolean,

  private val poseClassification: List<String>,

) : Graphic(overlay) {

    private var zMin = java.lang.Float.MAX\_VALUE

    private var zMax = java.lang.Float.MIN\_VALUE

    private val classificationTextPaint: Paint

    private val leftPaint: Paint

    private val rightPaint: Paint

    private val whitePaint: Paint

    fun draw(canvas: Canvas) {

        val landmarks = pose.allPoseLandmarks

        if (landmarks.isEmpty()) {

        return

        }

        val nose = pose.getPoseLandmark(PoseLandmark.NOSE)

        val rightShoulder = pose.getPoseLandmark(PoseLandmark.RIGHT\_SHOULDER)

        val leftShoulder = pose.getPoseLandmark(PoseLandmark.LEFT\_SHOULDER)

        val rightHip = pose.getPoseLandmark(PoseLandmark.RIGHT\_HIP)

        val leftHip = pose.getPoseLandmark(PoseLandmark.LEFT\_HIP)

        val landmarksSub: List<PoseLandmark?> =

        listOf(nose, rightShoulder, leftShoulder, rightHip, leftHip)

        // Draw all the points

        for (landmark in landmarksSub) {

        drawPoint(canvas, landmark!!, whitePaint)

        if (visualizeZ && rescaleZForVisualization) {

            zMin = min(zMin, landmark.position3D.z)

            zMax = max(zMax, landmark.position3D.z)

        }

        // Log.i("test", (landmark.position3D).toString());

        }

        drawLine(canvas, leftShoulder, rightShoulder, whitePaint)

        drawLine(canvas, leftHip, rightHip, whitePaint)

        // Left body

        drawLine(canvas, leftShoulder, leftHip, leftPaint)

        val leftSide = calculateDistance(leftShoulder!!, leftHip!!)

        // Right body

        drawLine(canvas, rightShoulder, rightHip, rightPaint)

        val rightSide = calculateDistance(rightShoulder!!, rightHip!!)

        val avgDistance = (rightSide + leftSide) / 2

        // ! ------------------------------------- Show Distance -------------------------------------

        if (showDistance) {

        canvas.drawText(

            avgDistance.toString(),

            translateX(nose!!.position.x),

            translateY(nose.position.y),

            whitePaint

        )

        Log.d("Distance: ", avgDistance.toString())

        }

    }

}

```

6.3.3 Distance measurement

The railroad ties appear to get smaller as they get further away from us. If we measured the apparent size of each railroad tie, their measured size would decrease in proportion to their distance from our eyes. The same thing happens to the distance between shoulders and hips.

A picture containing text

Description automatically generated

In order to calculate the distance between the camera and the detected person, we calculate the distance between the shoulders and hips. We could use the pupillary distance (PD), the distance between the centers of your pupils since it is fixed as the detected person moves, but it will be affected if the detected person did not face the camera. We used the distance between the shoulders and hips, considering if the detected person rotated.

We get the average of the distance right side and the left one, then using a polynomial equation we get how far the detected person is from the camera. The coefficients of the polynomial equation calculate by measuring the size of landmarks, the distance between shoulders and hips, to how far a person is from the device camera.