Graduation Project
2023-2024

The Egyptian E-Learning University (EELU)

Faculty of Computer & Information Technology

EELU- Aswan2024

Smart stick for visually impaired people.

Team members:

NAME	ID:
TAHA MOHAMED ALI	2000999
AHMED MOHAMED ALI	2001001
ABDELRAHMAN MAHMOUD FATHY	2001306
MOHAMED ABDEL BASET	2001028
MOHAMED TALAAT BAKHIT	2002049
MOHAMED AHMED ALI	2001721
BASSAM ABBAS SHARIF	2000477

Under Supervision of:

Dr: Ahmed Hamza	Eng: Hams Khaled
Assistant Prof of Computer and Information Technology Egyptian E-Learning University	Teaching Assistant of Computer and Information Technology Egyptian E-Learning University

"This documentation is provided as a partial fulfillment of the requirements for the degree of Bachelor of Science in Computer & Information Technology."

Outline:

- **Abstract.**
- **Chapter 1**

1. Idea name
2. Introduction.
3. Problem definition.
4. solutions definition.
5. Project objectives.
6. Related Work.

- **Chapter 2**

1. METHODOLOGIES and APPROACH.

- **Chapter 3**

1. Implementation and Coding.
2. Software Techniques.
3. Analyzing the system.
4. Application.
5. Coding.
6. Running.

- **Chapter 4**

1. Summary.
2. Conclusion.
3. Future Work.

- **Chapter 5**

1. References.

Abstract

Technologies are rapidly evolving, allowing people to live healthier and simpler lives. Sightless people are unable to carry out their everyday activities, such as walking down the street, visiting friends or relatives, or doing some other mundane tasks. As a result, the smart stick is a stick that can assist a person in walking safely without fear of colliding with another person or solid objects is proposed as a solution to this major issue. It is a development of the traditional blind stick as it acts as a companion for the blind when walking by sending audio alerts to the blind via a headphone connected to the phone with obstacles (water/walls/stairs / muddy ground) and enables him to make a phone call to ask for help. After statistics, we found that there are many blind people all over the world who suffer from difficulty dealing with and moving in the outside world, so we decided to design this stick in order to help these people and make life easier for them.

Chapter 1

1.Idea name

Braille a stick, smart stick for blind people.

After seeing the blind people in the street pleasing people to help them walk on street. I felt sad for that, and then.

started working on this project.

2.Introduction.

Smart Blind Stick is an interactive device which mainly aims at helping the blind to navigate easily and in a safer manner. In a normal day to day situation a blind person wave the

blind stick ahead of them in order to check for any objects or obstacles.

The smart stick helps them in this by detecting if any.

obstacle is blocking the path being taken by the subject.

The device detects the obstacle with the help of a camera attached to the front of the stick.

On detection of the obstacle, it is identified and appropriate instructions.

Navigating the world can be a challenging task for visually impaired people. The conventional blind walking stick serves the purpose but has limitations. Our Smart Blind Stick Project aims to revolutionize the way blind and visually impaired individuals navigate their surroundings.

Blindness is a very common disability among the people throughout the globe. About 90%

of the world's population, which are visually impaired, live in developing countries. They need help to walk and do the essential work of daily life. Smart Blind Stick is a fully automated as well as manually operated, easy to maintain, cheap and comfortable to use device. It is an innovative device designed for visually disabled people for refined navigation and advanced obstacle detection. In this device, we propose an advanced blind stick that allows visually challenged people to navigate with relieve using advanced technology. The blind stick is integrated with three ultrasonic sensors, panic switch, navigation switch, and Bluetooth and soil moisture detector along with Arduino UNO. The three ultrasonic sensors are used to detect obstacles ahead using ultrasonic waves. On sensing obstacles, the sensor passes this data to the person through the microphone device. The navigation process is implemented by smart stick with Global Positioning

System (GPS) for the blind, which will detect the obstacle and hurdle in the path and determine the position and location through GPS coordinates. The technologies used for the device include embedded C language for programming and coding,

Atmega328 microcontroller is used which is a low power CMOS microcontroller, GSM (Global System for Mobile Communication) for voice communication, Bluetooth to connect microphone with the device and GPS (Global Positioning System) to interface with device for navigation. It is not an effortless task for a blind person to use this device with complete accuracy as it requires necessary training to help the user understand the information and to react to them in real time.

3. Problem Definition

1. "Mobility and Navigation" The primary issue faced by visually impaired individuals is navigating through unfamiliar or crowded environments safely. A smart blind stick should assist in detecting obstacles, drop-offs, and other hazards in the user's path to prevent accidents.

2. "Obstacle Detection" Traditional canes can detect obstacles on the ground, but a smart blind stick should also be able to identify obstacles at waist height and above, such as low-hanging branches or protruding objects.

3. "Localization and Wayfinding" Blind individuals often struggle with orientation and knowing their exact location. A smart blind stick could incorporate GPS or indoor positioning systems to provide auditory or tactile cues for navigation, helping users to reach their destinations efficiently and independently.

4. "User Interface Accessibility" The interface of the smart blind stick needs to be intuitive and accessible for individuals with visual impairments. This involves designing tactile or auditory

feedback systems that convey information effectively without relying on visual displays.

5. "Integration with Other Assistive Technologies" Many visually impaired individuals use other assistive devices, such as smartphones with screen readers or braille displays. The smart blind stick should seamlessly integrate with these technologies to provide a comprehensive support system for the user.

6. Blind people depend on several methods to move around safely and independently. These are traditional solutions.

7. Training and Support: Effective training and ongoing support are essential for users to maximize the benefits of a smart blind stick and overcome any challenges or barriers they may encounter. Manufacturers should provide comprehensive training materials, tutorials, and customer support services to empower users and promote independent living.

4. Solutions Definition.

1. need, creating a smart blind stick equipped with obstacle detection and hazard alert systems could greatly enhance the mobility and safety of visually impaired individuals. Using technologies like ultrasonic sensors, cameras

2. Absolutely, expanding obstacle detection capabilities to include hazards at waist height and above is crucial for ensuring the safety and convenience of visually impaired individuals. This can be achieved through the integration of additional sensors such as cameras or LiDAR modules positioned strategically on the smart blind stick to scan the surrounding environment at different heights.

3. Object Detection: The code utilizes the YOLO algorithm to detect and identify objects in real-time from the camera feed. This helps blind people gain awareness of their surroundings, including the presence of objects.

4. Auditory Feedback: The code converts the object count information into speech using the GTTS library. By providing auditory announcements of the detected objects, blind individuals can receive real-time feedback about the number and type of objects around them.

5. Customizable Speech Delay: The code includes a speech delay feature (controlled by the speech delay variable) that introduces a pause between each object count announcement. This feature may be useful for blind individuals to have sufficient time to process the auditory information before the next announcement occurs.

5. Project objectives

1. Obstacle Detection: Develop a system capable of detecting obstacles on the ground, at waist height, and above to ensure comprehensive hazard awareness for the user.

2. Feedback Mechanisms: Design customizable feedback mechanisms, including vibrations, sound cues, or haptic feedback, to alert users to detected obstacles and hazards in real-time.

3. User Interface: Develop an intuitive user interface that allows users to customize settings, adjust feedback preferences, and easily interact with the smart blind stick.

4. Facilitating the Self-education process.

5. Facilitating the interaction process.

6. Increasing self-confidence among blind people.

7. Enabling an equal life for the blind.

8. Facilitating the communication process.

6.Related Work

- Ultrasonic Blind Walking Stick

- Smart Blind Stick

- Obstacle-Detecting

- Arduino Blind Stick

Chapter 2

1.METHODOLOGIES and APPROACH

A. Importing Libraries:

- The code starts by importing various Python libraries.

- required for different tasks, such as
- Random.
- Time.
- Play sound.
- Gtts (Google Text-to-Speech) ,

- Os
- cv2 (OpenCV)
- and ultralytics (YOLO implementation).

- **Prototype**
- The prototype was finalized in Python

1. Tested and tried

2. Appropriate improvements have been made

3. The present work has been started

B. Reading Class List

The code reads a text file containing a list of classes.

(labels) for object detection. It opens the file, reads its contents, and splits it into a list of classes.

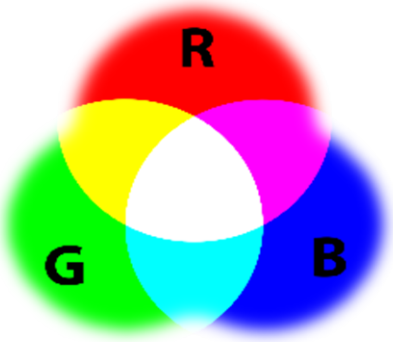
```
my_file = open("utils/coco.txt", "r")
data = my_file.read()
class_list = data.split("₩n")
my_file.close()
```

C. Generating Random Colors:

The code generates random colors for each class to be used for drawing bounding boxes around detected.

objects. It creates a list of random RGB color tuples.

```
detection_colors = []
for i in range(len(class_list)):
r = random.randint(0, 255)
g = random.randint(0, 255)
b = random.randint(0, 255)
detection_colors.append((b, g, r))
```

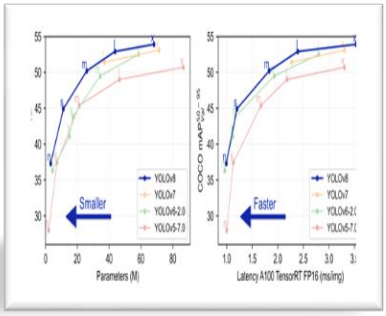


D. Loading the YOLO Model:

YOLOv8

The code initializes an instance of the YOLO model by providing the paths to the model weights file and the configuration file.

```
model = YOLO("weights/yolov8n.pt", "v8")
```



E. Setting up Video Capture:

The code opens a video capture object to read frames from the camera. If the camera cannot be opened, it.

prints an error message and exits.

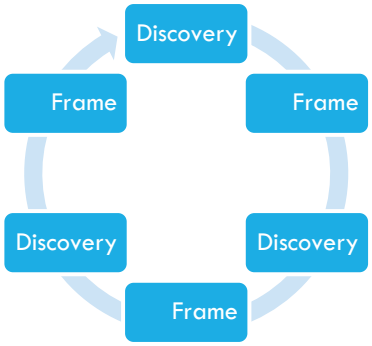
```
cap = cv2.VideoCapture(0)

cap = cv2.VideoCapture("inference/videos/colage.mp4")
if not cap.isOpened():
print("Cannot open camera")
exit()
```

F. Object Detection Loop:

The code enters a while loop to continuously read.

frames from the camera and perform object detection on each frame.

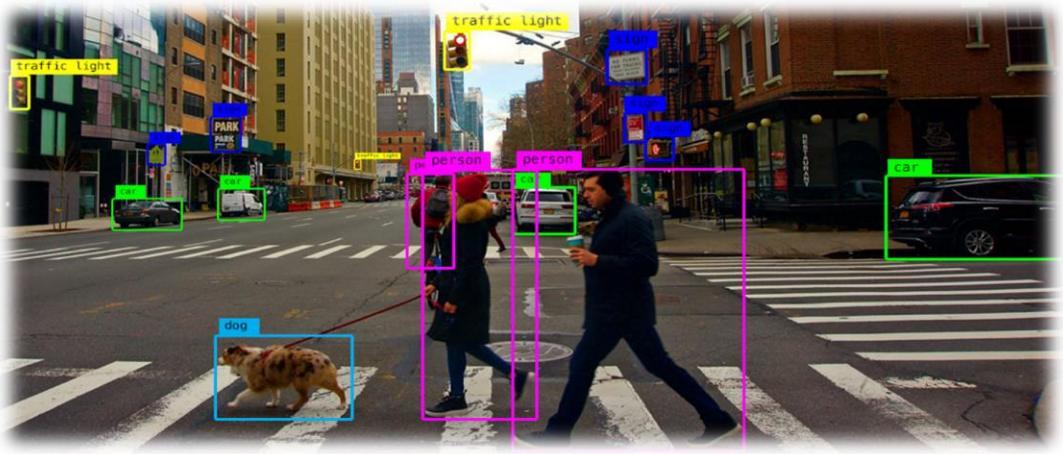
```
while True:
# Cap frame-by-frame
reta, frame = cap.read()
```

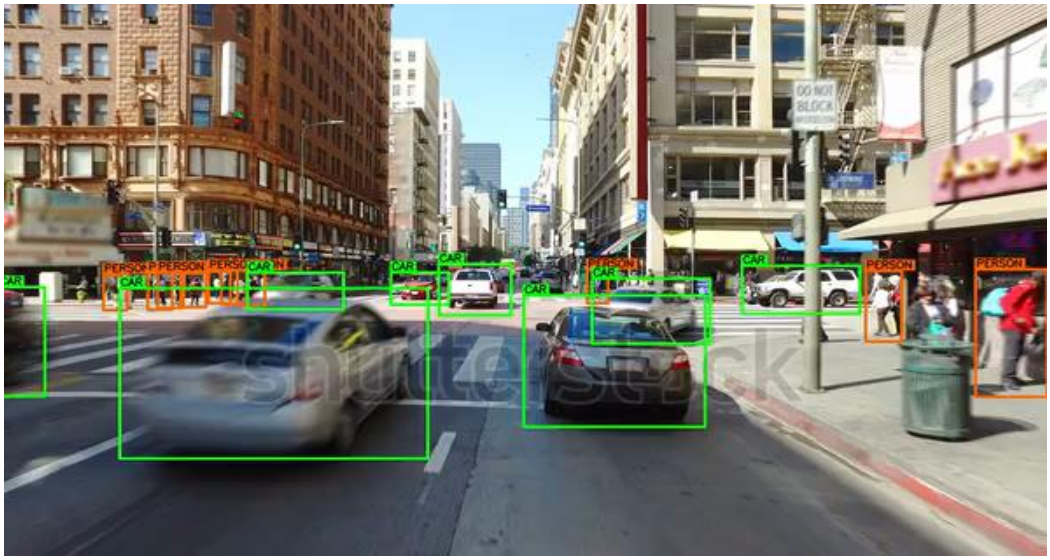


G. Object Detection using YOLO:

Inside the loop, the code uses the YOLO model to perform object detection on the current frame. It sets a confidence threshold of 0.6 and retrieves the detection results.

```
detect_params = model.predict(source=[frame], conf=0.6, save=False)
```

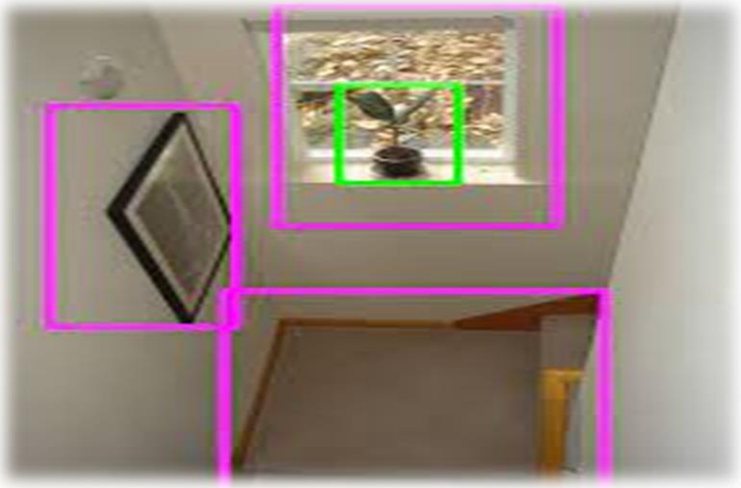




H. Drawing Bounding Boxes:

For each detected object, the code retrieves the bounding box coordinates, class ID, and confidence. It draws a rectangle around the object on the frame and displays the class label and confidence percentage.

```
boxes = detect_params[0].boxes
        box = boxes[i]  # returns one box
        clsID = box.cls.numpy()[0]
        conf = box.conf.numpy()[0]
        bb = box.xyxy.numpy()[0]

        cv2.rectangle(
            frame,
            (int(bb[0]), int(bb[1])),
            (int(bb[2]), int(bb[3])),
            detection_colors[int(clsID)],3,)
```



I. Counting Objects:

The code keeps track of the count of each detected. object class in the current frame. It compares the current count to the previous count to determine if there is a change.

```
for i in range(len(class_list)):
    if current_object_count[i] != prev_object_count[i]:
        prev_object_count[i] = current_object_count[i]
```

10.Text-to-Speech Announcement:

If there is a change in the object count, the code.

generates a text-to-speech audio file announcing the

count of the detected objects. It uses GTTS to convert.

the text to speech, saves the audio file, plays it using

play sound and removes the audio file.

```
if current_object_count[i] != 0:
        object_count_text = str(current_object_count[i])
        + " " + class_list[i] + " detected"

        obj = gTTS(text=object_count_text, lang='en', slow=False)
        obj.save("audio/object_count_{}.mp3".format(i))
```

11.Displaying the Frame and Handling User Quit:

Displaying a frame and handling user quit in the context of programming typically involves creating a graphical user interface (GUI) window and providing functionality for the user to close or quit the application gracefully.

The code shows the frame with the bounding boxes. and text overlays using OpenCV's Imshow function, waits for the 'q' key to be pressed to exit the program.

```
cv2.imshow("ObjectDetection", frame)

if cv2.waitKey(1) == ord("q"):
    break
```

12. **Kotlin**

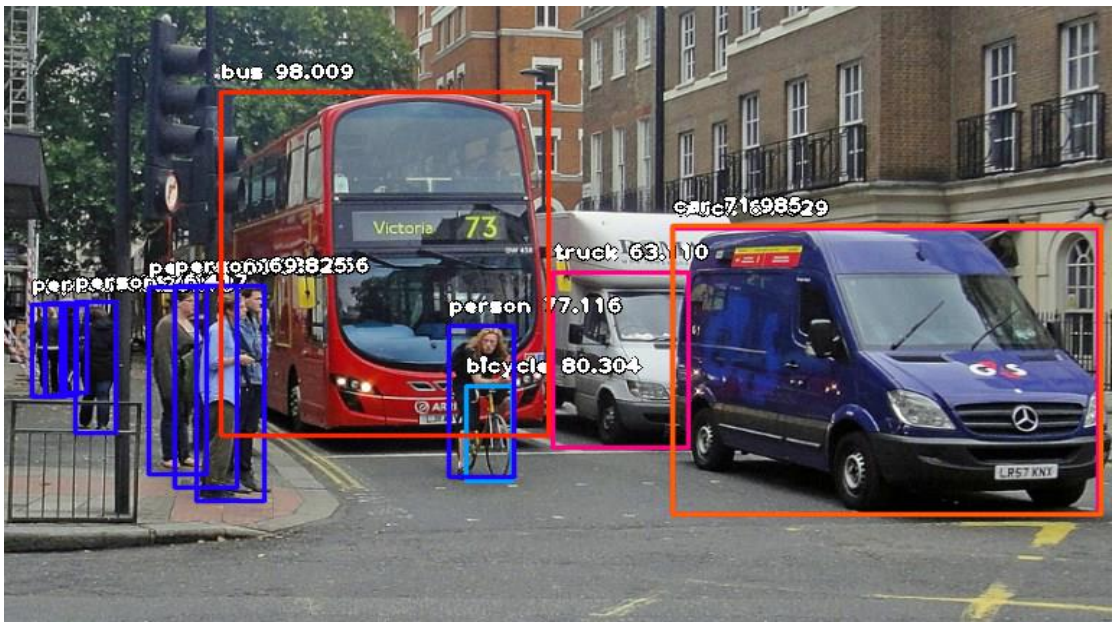
- ❑Why Use Kotlin?
- ❑Kotlin is fully compatible with Java.
- ❑Kotlin works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- ❑Kotlin is concise and safe.
- ❑Kotlin is easy to learn, especially if you already know Java.
- ❑Kotlin is free to use.
- ❑Big community/support

13. distance Measurement

- we researched the topic of distance calculation in depth and found some common and widely used methods:
- Using a three-dimensional (3D) camera:
- Using Depth Sensors:
- Optical information:
- Training AI models to estimate distance:

Calculate distance based on object sizes and boundaries

- Voice reading of objects with larger boundaries will be done according to the given condition



Chapter 3

1.Implementation and Coding.

2.Software Techniques:

- **Machine Learning Object Detection:**

Blind Stick employs machine learning algorithms to detect obstacles and objects in the user's environment. This technique allows the app to analyze images captured by the device's camera in real-time and identify potential hazards such as walls, furniture, or other obstacles.

- **Text-to-Speech (TTS) Technology:**

The application utilizes text-to-speech technology to convert textual information, such as alerts and notifications, into spoken words. This allows visually impaired users to receive auditory feedback about their surroundings, including detected obstacles and navigation instructions.

- **Sensor Integration:**

Blind Stick integrates with various sensors available on smartphones, such as accelerometers and gyroscopes, to gather data about the device's orientation and

movement. By analyzing sensor data, the app can provide context-aware alerts and guidance to users as they navigate their environment.

- **Customizable Settings:**

The app offers customizable settings that allow users to adjust parameters such as alert volume, speech rate, and detection sensitivity. These settings enable users to personalize their experience according to their preferences and specific needs, enhancing usability and user satisfaction.

- **Real-time Feedback and Alerts:**

Blind Stick provides real-time auditory alerts and feedback to users based on the analysis of sensor data and image processing. This immediate feedback allows users to react promptly to detected obstacles and navigate their surroundings more safely and effectively.

3.Analyzing the system:

Designing a system for a blind stick involves integrating various sensors and technologies to help visually impaired individuals navigate their surroundings safely. Here's an analysis of the components and considerations typically involved:

Ultrasonic Sensors: These sensors emit ultrasonic waves and measure the time it takes for them to bounce back after hitting an obstacle. This information helps in detecting obstacles in the path of the user.

Infrared Sensors: Infrared sensors work similarly to ultrasonic sensors but use infrared light instead of sound waves. They are effective for detecting objects within a shorter range.

Vibration Motors: These are used to provide haptic feedback to the user. Different vibration patterns can convey information about the proximity and direction of obstacles detected by the sensors.

4.Application:

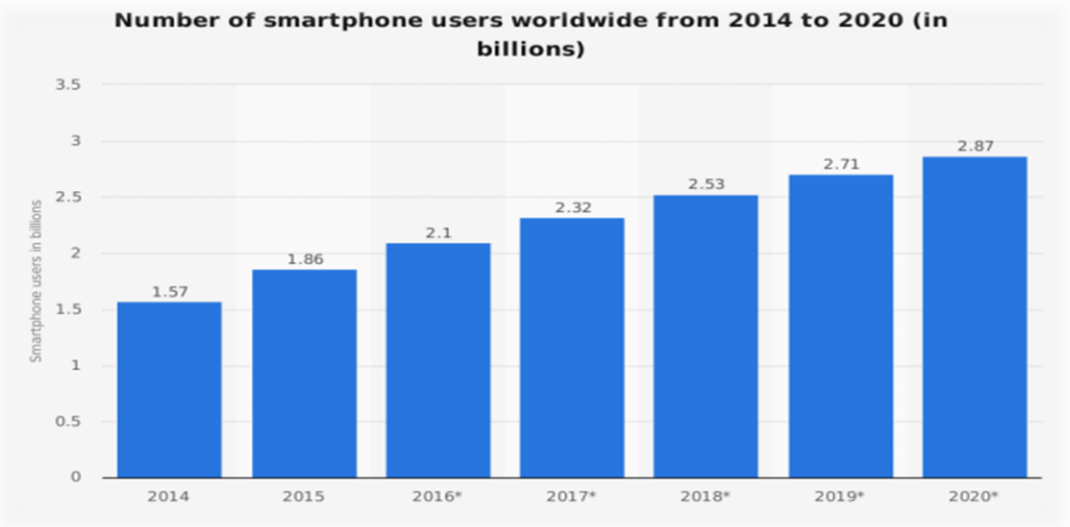
- The best solution for blind people

1. No blind man is without a smart phone

2.The smartphone is easy to carry

3. It contains many features to support the blind

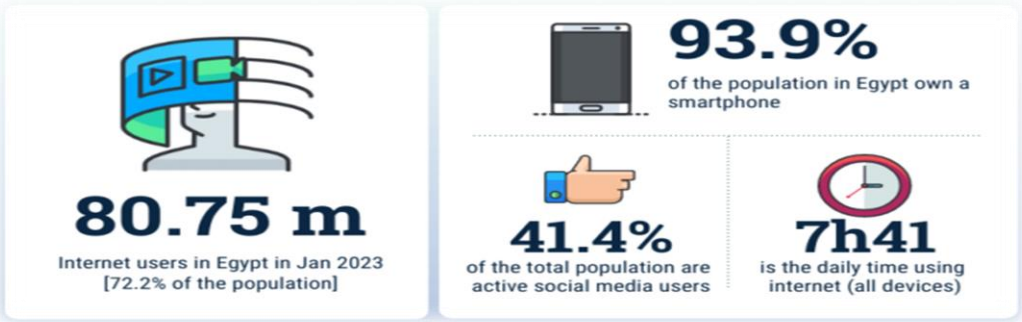
- Global statistics



- According to the Ministry of Communications and Information Technology

about 98.8% of Egyptian families own a mobile phone, and about 93.9% of individuals use mobile phones. In

January 2023, there were 105 million mobiles connections in Egypt

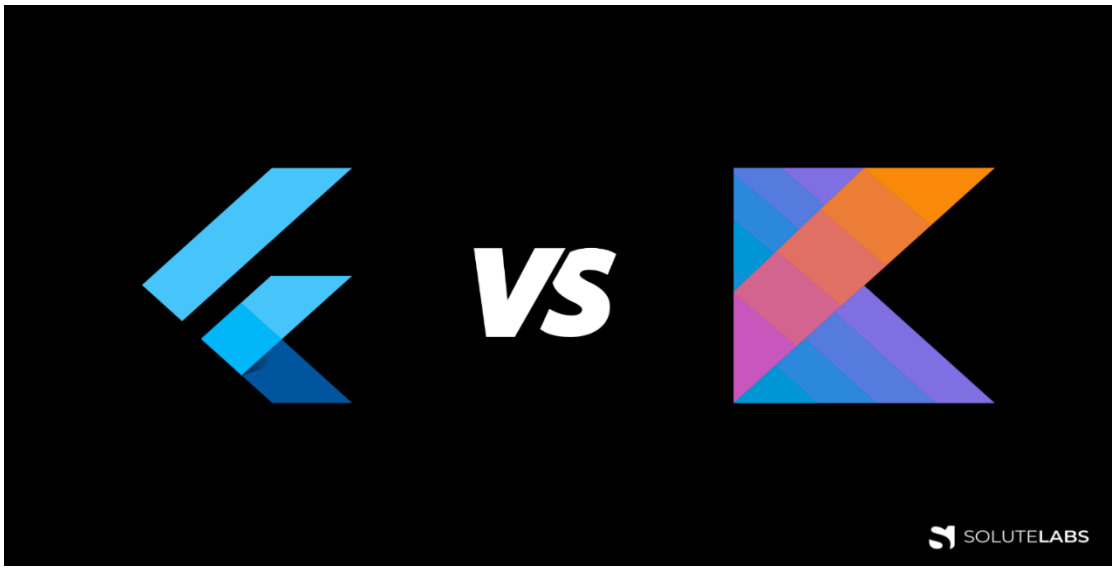


- No home is without a smartphone these days

- Kotlin programming language was used to build the application

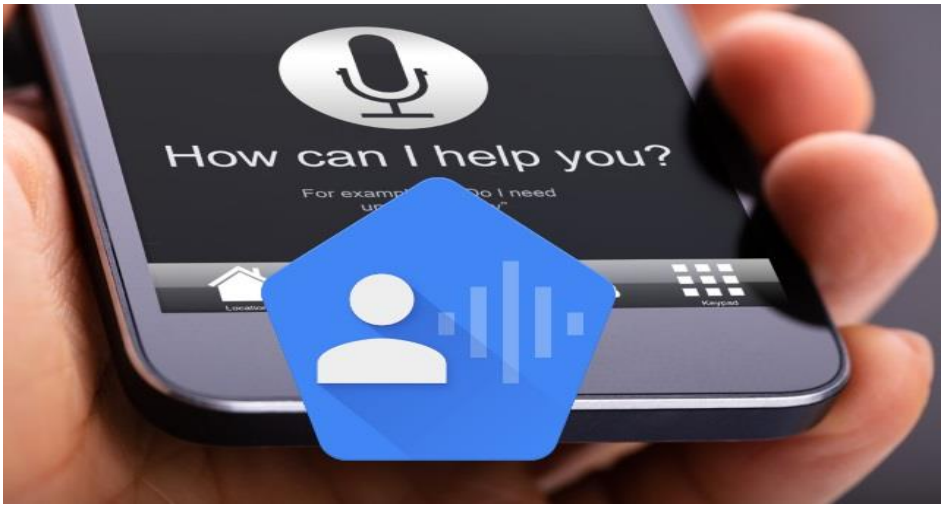
- Kotlin requires fewer lines of code as compared to java

- Kotlin for Android offer native performance, as they are specifically designed for their respective platforms. This can result in faster and more efficient apps compared to Flutter

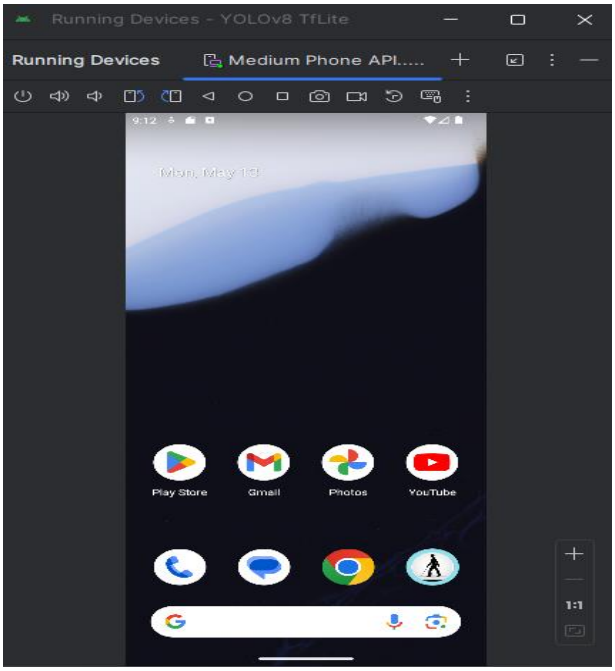


- **Voice interaction**

  The Voice Access app for Android lets you control your device with spoken commands. Use your voice to open apps, navigate, and edit text hands-free

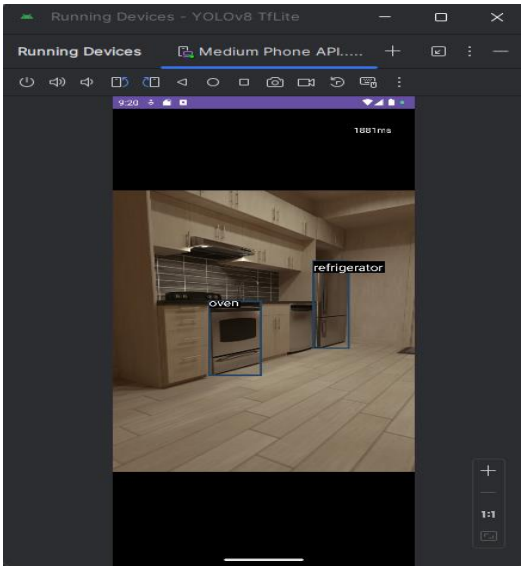


- First, using a voice command, the application is opened



➢ Practical experience

The object detection process takes place directly



➢ The blind person will have several options

1. Press the volume down button twice to turn on voice
2. reading for larger objects
3. Press twice more to stop the sound
4. Press the volume up button twice to close the
5. application

Object Detection

1. **Object Detection vs. Object Recognition vs. Image Classification:**
    - **Image Classification:** Determines what objects are in an image but doesn't provide their locations. For example, it might classify an image as containing a "cat" or "dog."
    - **Object Recognition:** Identifies and labels objects within an image.
    - **Object Detection:** Combines both recognition and localization by identifying objects and their positions using bounding boxes.
2. **Key Components:**
    - **Bounding Boxes:** Rectangles drawn around detected objects to show their locations.
    - **Confidence Scores:** Probability estimates that indicate how likely it is that a detected object belongs to a particular class.
    - **Class Labels:** Names or categories assigned to detected objects (e.g., "person," "car," "dog").

Steps in Object Detection

1. **Data Collection:**
    - **Image Acquisition:** Gather a diverse set of images that include the objects of interest in various scenarios and backgrounds.
    - **Annotation:** Manually or semi-automatically label the images by drawing bounding boxes around objects and assigning class labels. Tools like LabelImg or services like Rob flow can assist in this process.
2. **Data Preprocessing:**
    - **Normalization:** Scale image pixel values to a standard range, usually [0, 1] or [-1, 1].
    - **Augmentation:** Apply transformations (e.g., rotation, flipping, scaling) to increase the diversity of the training dataset and improve the model's robustness.
3. **Model Selection:**
    - **YOLO (You Only Look Once):** A real-time object detection system known for its speed and accuracy. It divides the image into a grid and predicts bounding boxes and class probabilities directly.

- **SSD (Single Shot MultiBox Detector):** Another efficient object detection model that produces bounding boxes and class predictions in a single forward pass.
- **Faster R-CNN:** A more accurate but slower model that uses region proposal networks to generate candidate object bounding boxes.

4. **Training the Model:**
   - **Loss Function:** Combine classification loss (for class labels) and localization loss (for bounding box coordinates) to train the model.
   - **Hyperparameters:** Set parameters like learning rate, batch size, number of epochs, etc.
   - **Training Process:** Feed annotated images into the model, calculate loss, and update model weights using backpropagation.
5. **Evaluation:**
   - **Validation Set:** Use a separate set of images not seen during training to evaluate the model's performance.
   - **Metrics:** Common metrics include Precision, Recall, and mAP (mean Average Precision), which provide insights into the model's

accuracy and ability to detect objects correctly.

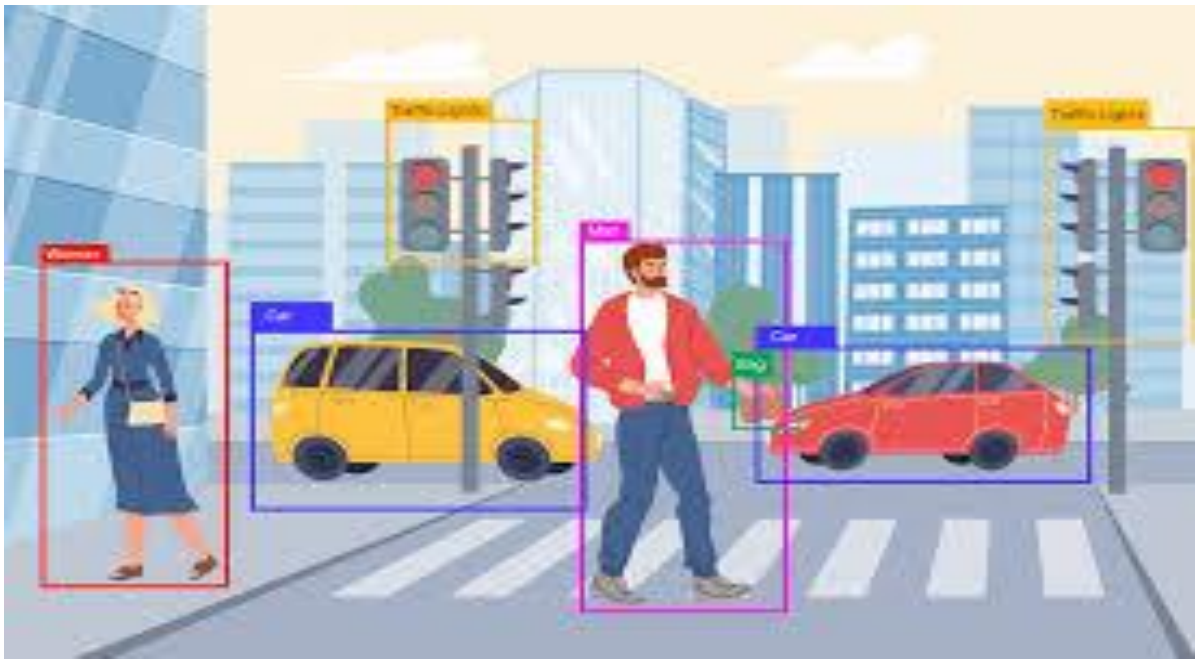
6. **Inference:**
    - **Object Detection:** Apply the trained model to new images to detect objects. The model outputs bounding boxes, class labels, and confidence scores.
    - **Visualization:** Draw bounding boxes on the images to visualize detected objects.
7. **Deployment:**
    - **Integration:** Incorporate the trained model into applications, such as mobile apps, web services, or embedded systems, to perform real-time object detection.
    - **Optimization:** Optimize the model for speed and efficiency, especially for deployment on resource-constrained devices.
8. **Continuous Improvement:**
    - **Monitoring:** Track the model's performance in the real world and gather new data to identify cases where the model might fail.
    - **Retraining:** Regularly update the model with new data to improve its accuracy and adaptability.

Practical Example Using YOLOv8

1. **Preparation:**
    - Collect a dataset of images containing objects you want to detect (e.g., cups, cars, people).
    - Annotate the images by drawing bounding boxes around the objects and labeling them.
2. **Data Handling with Roboflow:**
    - Upload the annotated dataset to Roboflow.
    - Use Roboflow to preprocess and augment the dataset.
    - Download the dataset in a format compatible with YOLOv8.
3. **Model Training:**
    - Set up YOLOv8 in your environment.
    - Configure the model with appropriate parameters and train it using the downloaded dataset from Roboflow.
    - Monitor the training process to ensure the model is learning effectively.
4. **Evaluation and Testing:**
    - Evaluate the trained model on a validation set to check its performance.

- Adjust hyperparameters and retrain if necessary.

5. **Deployment:**
   - Deploy the trained model in your application.
   - Use the model to detect objects in new images or live video streams.
6. **Inference and Monitoring:**
   - Continuously monitor the model's performance.
   - Update the model with new data as needed to maintain or improve accuracy.



5.Coding.

```
class MainActivity : AppCompatActivity(), Detector.DetectorListener {
    private lateinit var binding: ActivityMainBinding
    private lateinit var detector: Detector
    private lateinit var textToSpeech: TextToSpeech
    private lateinit var cameraExecutor: ExecutorService
    private lateinit var handler: Handler
    private var lastSpeakTime: Long = 0
    private val speakInterval: Long = 10000
    private var lastVolumeUpTime: Long = 0
    private var lastVolumeDownTime: Long = 0
    private val doublePressInterval: Long = 500
    private var isSpeaking: Boolean = false
    private val REQUIRED_PERMISSIONS = arrayOf(Manifest.permission.CAMERA)
```

- This **Main Activity** class is an Android activity that extends **App Compat Activity** and implements **Detector. Detector Listener.** Let's go through its attributes and explain their purposes:
- **Binding:**
- kotlin
- private lateinit var binding: Activity Main Binding
    - This variable holds the binding object for the activity's layout. It is used to access views defined in the layout XML file.
- **Detector:**
- kotlin

- private lateinit var detector: Detector
    - An instance of the **Detector** class, which presumably handles object detection functionality in the app.
- **TextToSpeech:**
- kotlin
- private lateinit var textToSpeech: TextToSpeech
    - An instance of **TextToSpeech** class, used for converting text into spoken words.
- **Camera Executor:**
- kotlin
- private lateinit var camera Executor: Executor Service
    - An executor service used for managing tasks related to the camera, such as starting and stopping camera operations.
- **Handler:**
- kotlin
- private lateinit var handler: Handler

- An instance of **Handler** class, which allows communication between the worker threads and the UI thread.

- **Timers:**
- kotlin
- private var lastSpeakTime: Long = 0
  private Val speak Interval: Long = 10000
  private var last Volume Up Time: Long = 0
  private var lastVolumeDownTime: Long = 0
  private Val doublePressInterval: Long = 500
  - These variables are used to manage timing-related operations, such as controlling the interval between spoken alerts and detecting double presses of volume buttons.
- **Speech State:**
- kotlin
- private var is Speaking: Boolean = false
  - This variable track whether the text-to-speech functionality is currently active.
- **Required Permissions:**
- kotlin

- private Val REQUIRED_PERMISSIONS = array Of(Manifest.permission.CAMERA)
    - An array containing the permissions required by the activity. In this case, it only contains the camera permission.

AndroidManifest.xml.

```
      <?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-feature android:name="android.hardware.camera" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />


    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher_round"
        android:label="Blind_App"
        android:roundIcon="@mipmap/ic1_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.YOLOv8TfLite"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest
```

- This AndroidManifest.xml file defines essential configurations and permissions for an Android application. Here's an explanation of the key elements:
- Permissions and Features
- Camera Feature (<uses-feature android:name="android.hardware.camera" />):
- Specifies that the application requires access to a camera.
- Permissions:
- <uses-permission android:name="android.permission.CAMERA" />: Grants the application permission to access the device's camera.
- <uses-permission android:name="android.permission.RECORD_AUDIO" />: Grants the application permission to record audio, which might be necessary for certain features like voice commands or audio feedback.
- <uses-permission android:name="android.permission.INTERNET" />: Grants the application permission to access the internet, allowing it to make network requests.

- <uses-permission android:name="android.permission.WAKE_LOCK" />: Grants the application permission to prevent the device from going to sleep while the app is active, which might be necessary to keep the camera running continuously.
- Application Configuration
- Application Attributes:
- android:allowBackup="true": Specifies whether the application data should be backed up.
- android:dataExtractionRules="@xml/data_extraction_rules": Specifies rules for data extraction.
- android:fullBackupContent="@xml/backup_rules": Specifies rules for full backup content.
- android:icon="@mipmap/ic_launcher_round": Sets the application icon.
- android:label="Blind_App": Sets the label for the application.
- android:roundIcon="@mipmap/ic1_launcher_round": Sets the round icon for the application.

- Android supportsRtl="true": Indicates whether the application supports right-to-left (RTL) layout direction.
- Android theme="@style/Theme.YOLOv8TfLite": Specifies the theme to be used for the application.
- Activity Configuration
- Main Activity:
- Defines the main activity of the application.
- Android name="Main Activity": Specifies the activity class.
- Android exported="true": Indicates whether other applications are allowed to interact with this activity.
- Intent Filter:
- Specifies that the Main Activity is the launcher activity, meaning it is the entry point of the application.
- <action android name="android. Intent action MAIN" />: Specifies the main action for the intent filter.
- <category android:name="android intent category LAUNCHER" />: Specifies that the activity should appear in the launcher.
- Explanation

The AndroidManifest.xml file provides essential metadata about the application to the Android system. It declares permissions, features, activities, and other application components required for the proper functioning of the application. Each element serves a specific purpose in defining the behaviour and capabilities of the application.

```
textToSpeech = TextToSpeech( context: this) { status ->
    if (status == TextToSpeech.SUCCESS) {
        val result = textToSpeech.setLanguage(Locale.getDefault())
        if (result == TextToSpeech.LANG_MISSING_DATA || result == TextToSpeech.LANG_NOT_SUPPORTED) {
            Log.e( tag: "Camera",  msg: "اللغة غير مدعومة.")
        }
    } else {
        Log.e( tag: "Camera",  msg: "فشل في تهيئة TextToSpeech.")
    }
}
```

Certainly! This code is responsible for initializing the `Textspeak` engine within an Android activity. Let's break down each part of the code and explain its purpose:

1. Textspeak Initialization:

```
kotlin

textspeak = Textspeak(this) {status ->

    // Initialization callback code

}
```

- Here, a new instance of the `Textspeak` class is created using the activity context (`this`). The constructor also takes a callback function as a parameter. This function will be invoked once the initialization process is complete.

2. Initialization Callback:

kotlin

```
{status ->
    // Code to handle initialization status
}
```

- The callback function receives a single parameter `status`, which indicates whether the initialization was successful. It's executed once the initialization process finishes.

3. Checking Initialization Status:

kotlin

```
if (status == TextToSpeech. SUCCESS) {
    // Initialization was successful, proceed to set language
} else {
    // Initialization failed, log an error
```

}

- Inside the callback function, the `status` parameter is checked to determine if the initialization was successful (`TextToSpeech. SUCCESS`). If it's successful, further actions are taken to set the language. Otherwise, an error message is logged.

4. Setting Language:

kotlin

```
Val result = textToSpeech.setLanguage(Locale.getDefault())
```

- If the initialization is successful, the language for the `TextToSpeech` engine is set using the `set Language ()` method. In this case, it attempts to set the language to the default locale of the device (`Locale.getDefault() `).

5. Handling Language Setting Result:

kotlin

```
if (result == TextToSpeech.LANG_MISSING_DATA || result == TextToSpeech.LANG_NOT_SUPPORTED) {
    Log.e("Camera", "اللغة غير مدعومة.")
```

}

- After attempting to set the language, the `result` variable is checked to see if the language data is missing or not supported. If it's missing or not supported, an error message is logged.

6. Handling Initialization Failure:

kotlin

else {

Log.e("Camera", "فشل في تهيئة TextToSpeech.")

}

If the initialization of the `TextToSpeech` engine fails for any other reason, an error message is logged.

```kotlin
override fun onKeyDown(keyCode: Int, event: KeyEvent?): Boolean {
    val currentTime = SystemClock.uptimeMillis()

    if (keyCode == KeyEvent.KEYCODE_VOLUME_UP) {
        if (currentTime - lastVolumeUpTime <= doublePressInterval) {
            finish()
            return true
        }
        lastVolumeUpTime = currentTime
    }

    if (keyCode == KeyEvent.KEYCODE_VOLUME_DOWN) {
        if (currentTime - lastVolumeDownTime <= doublePressInterval) {
            isSpeaking = !isSpeaking
            if (isSpeaking) {
                textToSpeech.speak( text: "تفعيل القراءة الصوتية", TextToSpeech.QUEUE_FLUSH, params: null, utteranceId: null)
            } else {
                textToSpeech.speak( text: "ايقاف القراءة الصوتية", TextToSpeech.QUEUE_FLUSH, params: null, utteranceId: null)
            }
            return true
        }
        lastVolumeDownTime = currentTime
    }

    return super.onKeyDown(keyCode, event)
```

This `on Detect` function is a callback method called when objects are detected in the camera feed. Let's break down what it does:

.It updates the UI with the inference time, which represents how long it took for the object detection algorithm to process the current frame.

Kotlin

binding. Inference Time. Text = "${inference Time}MS"

.It updates the overlay with the bounding boxes around the detected objects and invalidates the overlay to trigger a redraw.

Kotlin

binding.overlay.apply}

set Results(bounding Boxes)

invalidate()

{

.It checks if speech output is enabled (`is Speaking`) and whether enough time has passed since the last spoken alert (`speak Interval`). If both conditions are met, it iterates over the list of detected bounding boxes.

.For each bounding box, it constructs a speech text containing the class name of the detected object) e.g$ ،احترس" ,.label.("

.It checks if the width (`w`) and height (`h`) of the bounding box are greater than 0.1 (which is likely a threshold to filter out very small detections). If the condition is met, it attempts to speak the speech text using the `textToSpeech` engine.

It updates the `lastSpeakTime` to the current time after speaking to ensure that alerts are not spoken too frequently.

Overall, this function orchestrates the updating of the UI with inference time and bounding box

overlays, and triggers spoken alerts for detected objects based on certain conditions.

```
package com.surendramaran.yolov8tflite

import android.Manifest
import android.content.pm.PackageManager
import android.graphics.Bitmap
import android.graphics.Matrix
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.os.SystemClock
import android.speech.tts.TextToSpeech
import android.util.Log
import android.view.KeyEvent
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.camera.core.*
import androidx.camera.lifecycle.ProcessCameraProvider
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import com.surendramaran.yolov8tflite.Constants.LABELS_PATH
import com.surendramaran.yolov8tflite.Constants.MODEL_PATH
import com.surendramaran.yolov8tflite.databinding.ActivityMainBinding
import java.util.Locale
import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors
```

- The code
- The application is designed to capture live camera input, process the frames using an object detection model (YOLO), and provide audio feedback on the detected objects through TTS.

- **Key Components**
- Permissions Handling:

- Manifest Permissions: The app requests camera permissions to access the device's camera.
- Checking Permissions: It checks whether the necessary permissions have been granted. If not, it requests them from the user.
- Handling Permission Result: Once the user responds to the permission request, the app handles the result to either start the camera if permissions are granted or log an error and close the app if not.
- Camera Setup:

- Camera X Library: The app uses the Camera X library for camera functionality, which simplifies the integration process.
- Camera Provider: It binds the camera lifecycle to the activity's lifecycle, ensuring that camera resources are managed appropriately.

- Preview Use Case: A preview is set up to display the camera feed on the screen.
- Text-to-Speech (TTS):

- Initialization: TTS is initialized to convert text to speech, providing auditory feedback to the user.
- Language Setting: The app sets the language for TTS, ensuring that it matches the user's locale (in this case, US English).
- Lifecycle Management: The app ensures that TTS resources are released when the activity is destroyed.
- Thread Management:

- Executor Service: A background thread is managed by an Executor Service to handle camera operations and object detection, ensuring that these tasks do not block the main UI thread.
- How It Works Together
- On Activity Creation:

- The app inflates the layout using Activity Main Binding.

- It initializes TTS and checks for camera permissions.
- Permissions Granted:

- If permissions are granted, the camera is started using Camera X.
- A Preview object is created and linked to the viewfinder in the UI to display the live camera feed.
- Text-to-Speech Initialization:

- TTS is set up and configured to use the appropriate language.
- If TTS initialization fails, an error is logged.
- Lifecycle Management:

- The app ensures proper cleanup by shutting down the ExecutorService and TTS when the activity is destroyed.
- Next Steps for Full Implementation
- Object Detection Integration:

- Load the YOLO model and label file in the app.

- Capture frames from the camera feed for processing.
- Run the object detection model on each frame to identify objects and their bounding boxes.
- Drawing Results:

- Implement a custom view or overlay to draw bounding boxes around detected objects on the camera preview.
- Annotate the objects with labels and confidence scores.
- Audio Feedback:

- Use TTS to announce detected objects, providing real-time auditory feedback to the user.

```
package com.surendramaran.yolov8tflite

data class BoundingBox(
    val x1: Float,
    val y1: Float,
    val x2: Float,
    val y2: Float,
    val cx: Float,
    val cy: Float,
    val w: Float,
    val h: Float,
    val cnf: Float,
    val cls: Int,
    val clsName: String,

)
```

- The Bounding Box data class is designed to represent the details of a detected object in an image or video frame, specifically for object detection applications. Here's an explanation of each property in the Bounding Box class and how they are used:
- Properties

x1 (Float):

- Description: The x-coordinate of the top-left corner of the bounding box.
- Purpose: Marks the horizontal starting point of the bounding box, indicating where the detected object begins on the x-axis.

y1 (Float):

- Description: The y-coordinate of the top-left corner of the bounding box.
- Purpose: Marks the vertical starting point of the bounding box, indicating where the detected object begins on the y-axis.

x2 (Float):

- Description: The x-coordinate of the bottom-right corner of the bounding box.
- Purpose: Marks the horizontal endpoint of the bounding box, showing where the detected object ends on the x-axis.

y2 (Float):

- Description: The y-coordinate of the bottom-right corner of the bounding box.

Purpose: Marks the vertical endpoint of the bounding box, showing where the detected object ends on the y-axis.

cx (Float):

- Description: The x-coordinate of the center of the bounding box.
- Purpose: Indicates the horizontal midpoint of the bounding box, useful for calculating the object's central position.

cy (Float):

- Description: The y-coordinate of the center of the bounding box.
- Purpose: Indicates the vertical midpoint of the bounding box, useful for determining the object's central position.

  w (Float):
- Description: The width of the bounding box.
- Purpose: Represents how wide the bounding box is, providing the horizontal extent of the detected object.

  h (Float):
- Description: The height of the bounding box.
- Purpose: Represents how tall the bounding box is, providing the vertical extent of the detected object.

  Cnf (Float):
- Description: The confidence score of the detection.
- Purpose: Indicates the likelihood that the bounding box correctly contains an object, with higher values meaning higher confidence.
- cls (Int):

- Description: The class ID of the detected object.
- Purpose: Provides an identifier for the type of object detected, based on a predefined set of classes (e.g., 1 for "person", 2 for "car").
- Cls Name (String):

- Description: The class name of the detected object.
- Purpose: Offers a human-readable name for the detected object's class, making the detection results more understandable.
- Usage
- Bounding Box Coordinates (x1, y1, x2, y2): These coordinates define the rectangle surrounding the detected object. The top-left corner is defined by (x1, y1) and the bottom-right corner by (x2, y2).
- Center Coordinates (cx, cy): The center of the bounding box provides a reference point for the detected object, which can be useful in various calculations such as tracking the movement of the object.

- Dimensions (w, h): The width and height of the bounding box describe the size of the detected object.
- Confidence Score (cnf): The confidence score helps in filtering out less certain detections, allowing only the most probable objects to be considered.
- Class Information (cls, cls Name): The class ID and class name provide both a numeric identifier and a descriptive label for the type of object detected, aiding in categorizing and interpreting detection results.

```
private fun startCamera() {
    val cameraProviderFuture = ProcessCameraProvider.getInstance( context: this)
    cameraProviderFuture.addListener({
        val cameraProvider = cameraProviderFuture.get()
        bindCameraUseCases(cameraProvider)
    }, ContextCompat.getMainExecutor( context: this))
}
```

- Obtaining the Camera Provider

- The function starts by getting an instance of the Process Camera Provider
- This provider is a key component that manages the lifecycle and state of the camera.

- Asynchronous Setup

- The process of obtaining the camera provider is asynchronous. Therefore, a listener is added to the future result.

  This listener will be notified once the camera provider is ready.

- Main Thread Execution:

- The listener's operations are executed on the main thread. This is important because most camera operations, especially those interacting with the UI, need to be run on the main thread.
- Binding Camera Use Cases

- Once the camera provider is available, the function calls another method (not detailed here) to bind specific camera use cases, such as preview, image capture, or video capture, to the camera lifecycle. This step ensures that the camera starts and stops in sync with the

application's lifecycle, providing a seamless user experience and proper resource management.

```
override fun onDetect(boundingBoxes: List<BoundingBox>, inferenceTime: Long) {
    runOnUiThread {
        binding.inferenceTime.text = "${inferenceTime}ms"
        binding.overlay.apply { this: OverlayView
            setResults(boundingBoxes)
            invalidate()
        }
    }

    val currentTime = SystemClock.uptimeMillis()
    if (isSpeaking && currentTime - lastSpeakTime >= speakInterval) {
        for (boundingBox in boundingBoxes) {
            val label = boundingBox.clsName
            val speechText = " ،احترس$label"
            if (boundingBox.w > 0.1 && boundingBox.h > 0.1) {
                try {
                    textToSpeech.speak(speechText, TextToSpeech.QUEUE_ADD, params: null, utteranceId: null)
                    lastSpeakTime = currentTime
                } catch (e: Exception) {
                    Log.e(TAG, msg: " :TextToSpeech تشغيل أثناء خطأ${e.message}")
                }
            }
        }
    }
}
```

- This code:
  Conceptual Breakdown
- Handling Detection Results:

- The function on Detect is called when bounding boxes (detected objects) are available.
- It receives a list of Bounding Box objects and the time taken for inference (detection).
- Updating the UI on the Main Thread:

- UI updates must be executed on the main thread to ensure thread safety and avoid crashes.
- The run on UI Thread method is used to update the UI elements safely.
- Displaying Inference Time:

- The inference time (time taken to detect objects) is displayed on the screen. This provides real-time feedback on the detection performance.
- Updating the Overlay with Bounding Boxes:

- An overlay element (likely a custom view) is updated to show the detected bounding boxes. This visual feedback helps users see what the system has detected.
- Checking for Audio Alerts:

- The function checks if the system is in a state where it should provide audio alerts (indicated by is Speaking).
- It ensures that alerts are not given too frequently by comparing the current time with the last time an alert was given (lastSpeakTime) and a defined interval (speak Interval).
- Generating Speech Alerts for Detected Objects:

- For each detected bounding box that meets certain size criteria, an audio alert is generated using Text-to-Speech (TTS).
- The alert message includes a warning and the class name of the detected object.
- The TTS system reads out the alert, adding it to the queue to be spoken.
- Error Handling for TTS:

- Any issues with the TTS system are caught and logged. This ensures that the app does not crash due to TTS errors and provides useful error messages for debugging.

- Handling No Detection:

- The function on Empty Detect is called when no objects are detected.
- It updates the overlay to reflect the absence of detected objects.
- Roles of Key Components
- Bounding Boxes: Represent detected objects, including their position and size on the screen.
- Inference Time: The duration taken by the model to detect objects, providing a performance metric.
- UI Thread Updates: Ensures all UI changes are made on the main thread to maintain app stability.
- Overlay: A visual layer that displays the bounding boxes over the camera feed or image.
- Text-to-Speech (TTS): Provides audio alerts to notify users about detected objects.
- Error Logging: Captures and logs any exceptions to help with troubleshooting.
- Example Workflow

- Detection Occurs:

- The system detects objects and calls the on Detect method with the results.
- UI Updates:

- The UI is updated with the inference time and the visual overlay showing the bounding boxes.
- Audio Alerts:

- If the conditions for speaking are met, audio alerts are generated and spoken using TTS.
- No Detection:

- If no objects are detected, the on Empty Detect method is called to update the overlay accordingly.

```
[ ]  from ultralytics import YOLO
     import os
     from IPython.display import display, Image
     from IPython import display
     display.clear_output()
     !yolo mode=checks
```

- The code

1. **Import the Necessary Modules**:

   python
   Copy code
   from ultralytics import YOLO
   import os
   from IPython.display import display, Image
   from I Python import display
   display. clear output ()

   - from ultralytics import YOLO: Imports the YOLO class from the ultralytics package.
   - import os: Imports the os module for interacting with the operating system.
   - from IPython.display import display, Image: Imports functions to display images and control display output in Jupyter notebooks.
   - from I Python import display: Imports the display module for clearing the output.
   - display. clear output (): Clears the current output in a Jupyter notebook to keep it tidy.

2. **Run YOLO Environment Checks**:

   python

Copy code
! yolo mode=checks

The! character at the beginning of the line indicates a shell command in a Jupyter notebook. This command runs the YOLO environment checks to ensure that your setup is correct, and all dependencies are properly installed.

Full Script

Here is the full script put together for clarity:

python
Copy code
# Import necessary modules
from ultralytics import YOLO
import os
from IPython.display import display, Image
from I Python import display

Clear current output in Jupyter notebook
display. clear output ()

Run YOLO environment checks
! yolo mode=checks

yolo mode=checks:

Purpose:

The yolo mode=checks command is designed to perform a series of checks on your system to verify that everything is in place for running YOLO models. This includes checking software dependencies, hardware availability, and configuration settings.

Environment Verification:

The command checks for the presence of necessary packages and their versions. It verifies that you have the correct versions of Python, PyTorch, and other dependencies required by the YOLO model.

Hardware Checks:

It verifies the availability and compatibility of hardware components such as GPUs. This is crucial for performance since YOLO models can significantly benefit from GPU acceleration.

Configuration Settings:

The command may also check for proper configuration settings that are necessary for the YOLO model to function correctly. This includes checking environment variables, paths, and other settings.

- Importing **Modules**:

- from ultralytics import YOLO: Imports the YOLO class from the Ultralytics package.
- import os: Imports the OS module for operating system interactions.
- from IPython.display import display, Image: Imports display functions for Jupyter notebooks.
- from I Python import display: Imports the display module to clear output.

- Clearing **Output**:

- display. clear output (): Clears the current output in the Jupyter notebook to keep the output clean.

- Running **the Check Command**:

- ! yolo mode=checks: Executes the yolo mode=checks command in the shell. This runs the built-in environment checks provided by the YOLO package.

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="YOUR-API-KEY")
project = rf.workspace("nicolai-hoirup-nielsen").project("cup-detection-v2")
dataset = project.version(3).download("yolov8")
```

- The code

Step 1: Installing the Roboflow Python Package

To use Roboflow's functionalities in your Python environment, you need to install a specific package. Think of this package as a toolkit that provides all the necessary functions and tools to interact with Roboflow's services. Installing this package ensures that your environment is set up correctly and that you have everything you need to communicate with Roboflow's servers.

Step 2: Importing the Roboflow Library

After installation, you need to bring this toolkit into your current working environment. Importing the library is like unpacking a toolbox. It allows you to access all the functions and classes provided by Roboflow, making them available for you to use in your script. This step is crucial because it sets up the bridge between your code and Roboflow's functionalities.

Step 3: Authenticating with the Roboflow API

Authentication is a security measure to ensure that only authorized users can access their data. When you use an API key, it's like showing your ID card to access a restricted

area. This key is unique to you and verifies that you have the right permissions to access your projects and datasets. Once authenticated, your session is recognized by Roboflow, and you can proceed to interact with your data securely.

Step 4: Accessing Your Workspace and Project

Roboflow organizes data in a hierarchical structure:

- **Workspace:** This is like a folder that contains multiple projects. Each workspace is dedicated to a user or team.
- **Project:** Inside a workspace, you can have several projects. Each project might focus on different tasks or datasets.

To work with a specific dataset, you need to specify which project (and workspace it belongs to) you are targeting. Think of it as navigating through a file system where you select the right folder (workspace) and file (project) you need to work with.

Step 5: Selecting and Downloading a Specific Version of the Dataset

Each project can have multiple versions of datasets. Versions might exist due to updates, improvements, or changes in the data. For example, you might have:

- Version 1 with basic data,
- Version 2 with more samples,
- Version 3 with cleaned and annotated data.

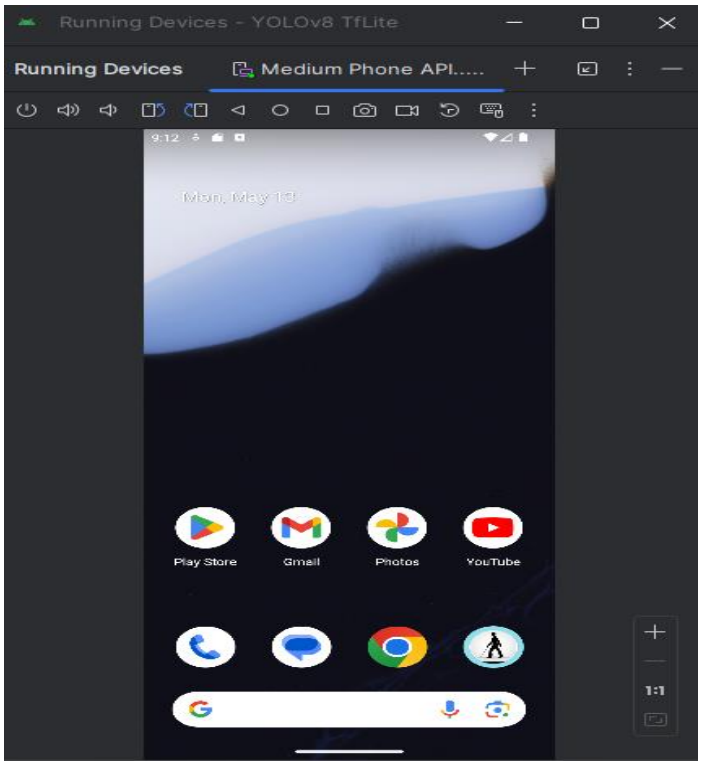
You need to specify which version you want to use. After selecting the version, you download it in a format compatible with your machine learning model—in this case, YOLOv8. Downloading the dataset involves fetching the data from Roboflow's servers to your local environment, formatted and ready to be used for training or testing your model.

Summary

In summary, the process involves setting up your environment with the necessary tools, authenticating yourself, navigating to the correct data, and finally obtaining the specific version of the data in the format you need. This workflow ensures you have access to the right data, properly formatted, to efficiently train or deploy your machine learning models.

6.Running.

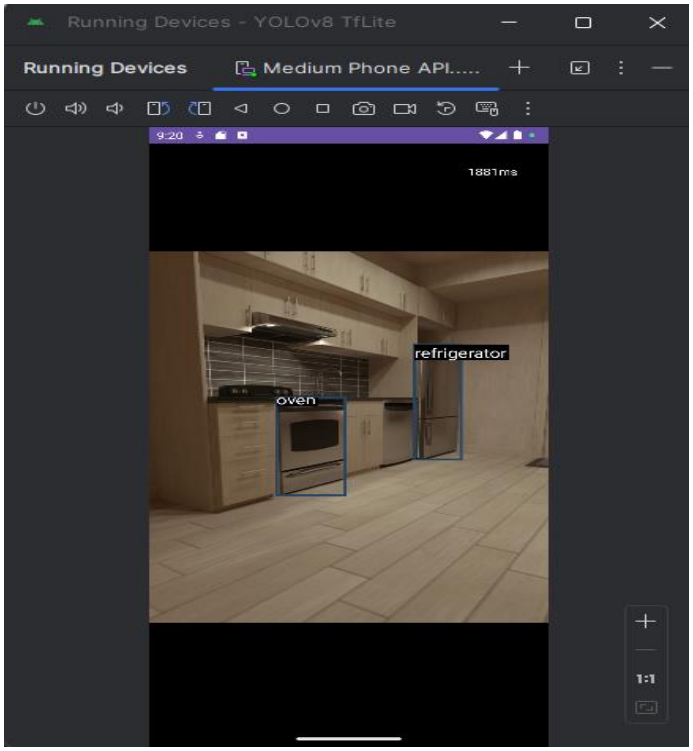
- First, using a voice command, the application is opened.



- The object detection process takes place directly

Inside the loop, the code uses the YOLO model to perform object detection on the current frame. It sets a confidence threshold of 0.6 and retrieves the

detection results.



➢ The blind person will have several options.

1. Press the volume down button twice to turn on voice
2. reading for larger objects
3. Press twice more to stop the sound
4. Press the volume up button twice to close the
5. application

Chapter 4

1. Summary.

- Blind Stick is an Android application designed to assist visually impaired individuals in navigating their surroundings more safely and independently. The app utilizes smartphone sensors and machine learning algorithms to detect obstacles and provide auditory alerts to the user.

- Key Features:

- Object Detection: The app uses a machine learning model to detect obstacles, such as walls, furniture, and other objects, in the user's vicinity.
- Real-time Alerts: Upon detecting obstacles, the app provides real-time auditory alerts to the user, informing them of the presence and approximate location of the detected objects.
- Speech Output: The app utilizes text-to-speech technology to convert textual information into spoken words, allowing the user to receive audible alerts about their surroundings.
- Customizable Settings: Users can customize various settings, such as alert volume, speech rate, and detection sensitivity, to suit their preferences and needs.

- Accessibility Features: Blind Stick incorporates accessibility features, such as voice-guided setup and intuitive user interface design, to ensure ease of use for visually impaired individuals.

- Benefits:

- Enhanced Safety: By alerting users to obstacles in their path, Blind Stick helps prevent accidents and injuries, allowing visually impaired individuals to navigate their environment more safely.
- Increased Independence: With the assistance of Blind Stick, visually impaired individuals can navigate unfamiliar environments with greater confidence and independence, reducing their reliance on assistance from others.
- User-Friendly Design: Blind Stick's intuitive interface and customizable settings make it easy for users to tailor the app to their specific needs and preferences, ensuring a seamless and personalized user experience.

2. Conclusion.

The Blind Walking Stick has been finally made into prototype that can be used to guide the blind. It aims to solve the problems faced by the blind people in their daily life. The system also takes measures to ensure their safety. This project will help all the blind people in the world and will make it easier for them to walk. It was done to help the blind move ahead very well. It helps to facilitate the movement ensuring safety.

Blind Stick is a powerful and innovative Android application that serves as a vital tool for visually impaired individuals. By harnessing the capabilities of smartphone sensors and machine learning algorithms, Blind Stick provides real-time assistance in navigating the environment, enhancing safety, independence, and confidence for users.

The app's ability to detect obstacles and provide auditory alerts in real-time

enables users to navigate unfamiliar environments with greater ease and assurance. Additionally, the customizable settings allow users to tailor the app to their specific needs, ensuring a personalized and user-friendly experience.

Blind Stick's emphasis on accessibility features and intuitive design further enhances its usability for visually impaired individuals, making it a valuable companion in their daily lives. By empowering users to navigate their surroundings independently and safely, Blind Stick promotes inclusivity and equality .for individuals with visual impairments

In essence, Blind Stick stands as a testament to the potential of technology to improve the lives of individuals with disabilities, reaffirming the importance of innovation in creating a more accessible and inclusive world.

Chapter 4

References.

- **W3school** https://www.w3schools.com/
- **Udemy** https://www.udemy.com/
- **Coursera** https://www.coursera.org/
- **El zero** https://elzero.org/tracks/front-end
- **Hassouna academy** https://www.hassouna-academy.com/
- **Desouki** https://www.youtube.com/user/DesoukiEgypt
- **Codepen** https://codepen.io/
- **GitHub** https://github.com/
- **Stack Overflow** https://stackoverflow.com/
- **Kaggle** https://www.kaggle.com/
- **United Nations website** https://www.un.org/ar/observances/braille-da

- Psychology website https://kenanaonline.com/users/alenshasy/posts/
- the seventh day https://www.youm7.com/story/2023/
- viso https://viso.ai/deep-learning/yolov8-guide/
- The ministry of communications https://mcit.gov.eg/ar
- Gtts https://pypi.org/project/gTTS/
- Kotlin https://kotlinlang.org/
- Voice Access https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.voiceaccess&hl=en
- Ultralytics Model https://github.com/ultralytics

THANKS