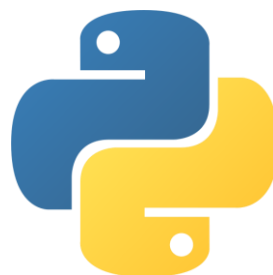




# APLICACIÓN MATRÍCULAS CIMUBB



Estudiante: Diego Ignacio Jiménez Muñoz

Fecha: 09-08-2023

## Contenido

<b>Introducción</b>	3
<b>Contexto de la necesidad a resolver</b>	3
<b>Recomendaciones previas</b>	3
<b>Instalación</b>	4
<b>Instalación de Ubuntu 20.04</b>	4
<b>Instalación de Python</b>	5
<b>Instalar OpenCV</b>	6
<b>Instalar Kivy</b>	6
<b>Instalar Buildozer</b>	6
<b>Código</b>	7
<b>main.py</b>	7
<b>homelayout.py</b>	8
<b>applayout.py</b>	9
<b>edgedetect.py</b>	9
<b>swipescreeen.py</b>	10
<b>buildozer.spec</b>	10
<b>Generar APK y debugear con dispositivo Android</b>	11
<b>Trabajo Futuro</b>	11
<b>Referencias</b>	12

## **Introducción**

Este documento ilustra la implementación de un prototipo de aplicación para Android, empleando el lenguaje de programación Python junto con el Framework Kivy para la creación de la interfaz. Adicionalmente, se detallan los pasos requeridos para generar un archivo APK mediante el uso de python-for.android y Android SDK. Asimismo, se proporcionan los prerequisites y recomendaciones esenciales para la instalación adecuada del software necesario en este proceso.

## **Contexto de la necesidad a resolver**

Los guardias de la Universidad del Bio-Bío tienen como necesidad de tener un registro y control sobre los vehículos que ingresan a la universidad, para ello se propone crear un sistema que consta de dos aplicaciones, la primera que debe funcionar en una cámara fija puesta en un lugar por donde ingresan los vehículos a la universidad para que pueda ir recopilando las matrículas y hora de ingreso de los vehículos, y la segunda aplicación que pueda ser usada por los guardias de la universidad para que puedan registrar de manera semi automatizada las matrículas de los vehículos con la cámara de sus dispositivos Android.

## **Recomendaciones previas**

En este proyecto, se empleó el lenguaje de programación Python para crear la aplicación Android. A pesar de la potencia y versatilidad de Python, es importante tener en cuenta que al desarrollar una APK pueden surgir desafíos de compatibilidad al implementar diversas funcionalidades y permisos en dispositivos Android.

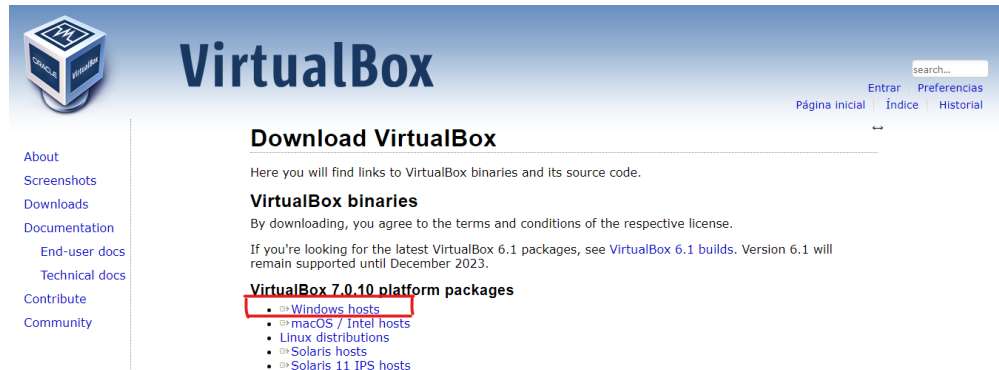
Por esta razón, al considerar la integración de APIs u otros servicios en la aplicación, sería recomendable evaluar la posibilidad de optar por un lenguaje de programación más ampliamente utilizado en el desarrollo de aplicaciones Android, como Java o Kotlin. Estos lenguajes tienen una presencia sólida en la comunidad de desarrollo Android y están respaldados por herramientas como Android Studio, que proporciona un entorno de desarrollo optimizado para la plataforma.

# Instalación

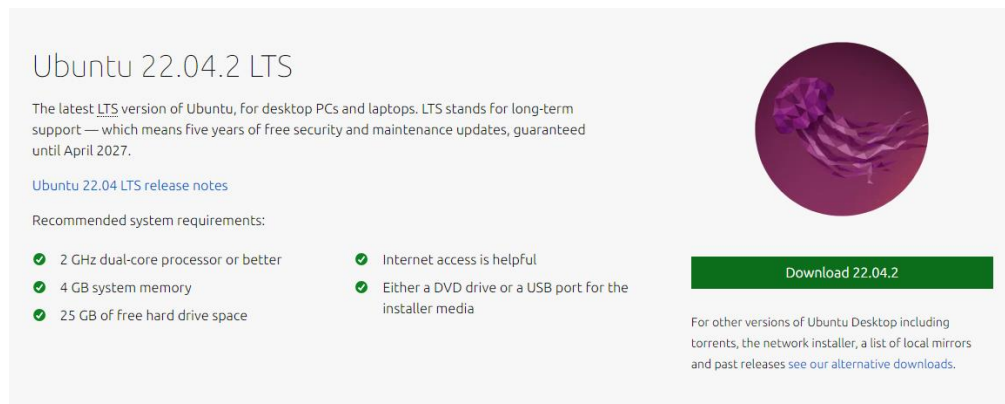
El primer requisito para el desarrollo de este proyecto es la instalación del sistema operativo GNU/Linux. Si bien es posible utilizar un subsistema de Windows para Linux (WSL), es ampliamente recomendable instalar el sistema operativo Linux en una máquina virtual. En este documento se mostrará la instalación de Ubuntu 20.04.

## Instalación de Ubuntu 20.04

1. Descargar e instalar Virtual Box en [Downloads – Oracle VM VirtualBox](#)

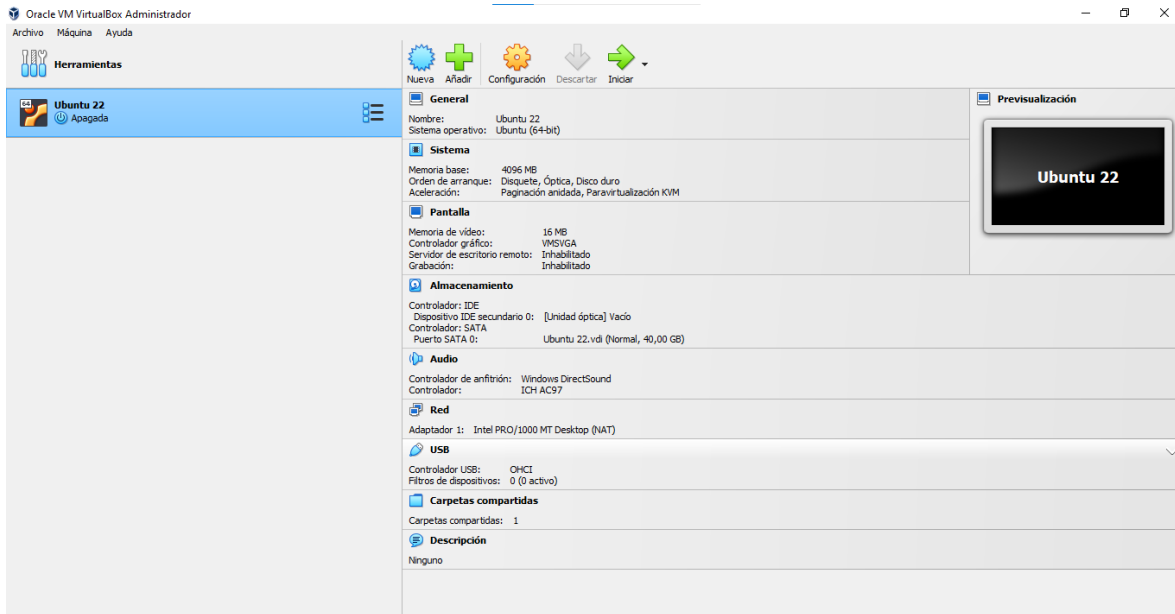


2. Descargar imagen de Ubuntu 20.04 en [Download Ubuntu Desktop | Download | Ubuntu](#)



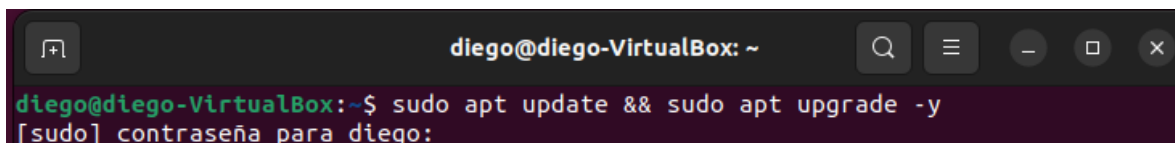
### 3. Instalar máquina virtual

En este punto se recomienda seguir los pasos del siguiente [vídeo](#), recordar asignar mínimo 4 GB de memoria RAM y al menos 20 GB de almacenamiento, además de configurar la resolución según a la de tu pantalla, una vez instalado podrás iniciar la máquina virtual.



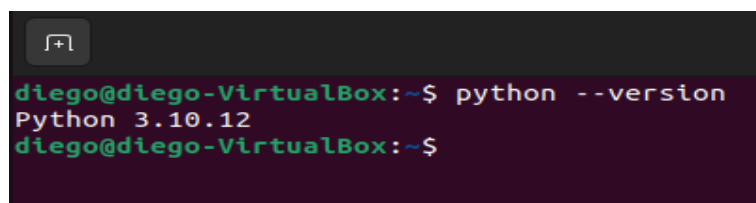
## Instalación de Python

1. Abrir la terminal (CTRL +ALT + T) y utilizar el comando: `sudo apt update && sudo apt upgrade -y`

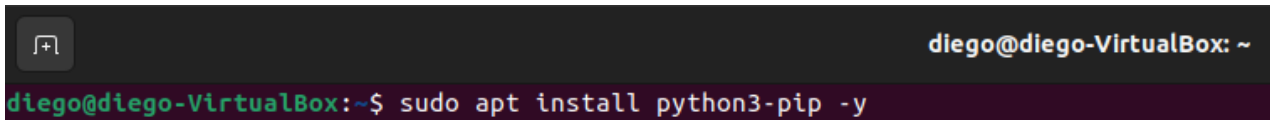


2. Utilizar el comando: `python3 --versión`

Dependiendo de la versión actual de Python debería mostrar algo similar a esto:



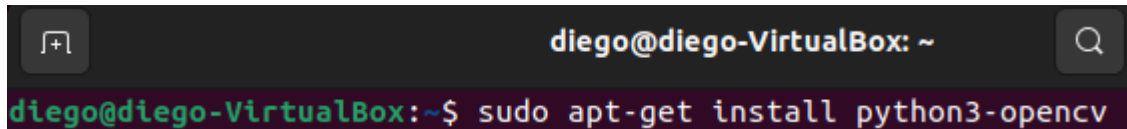
3. Utilizar el comando: `sudo apt install python3-pip -y`



```
diego@diego-VirtualBox:~$ sudo apt install python3-pip -y
```

## Instalar OpenCV

1. Utilizar el comando `sudo apt-get install python3-opencv`



```
diego@diego-VirtualBox:~$ sudo apt-get install python3-opencv
```

## Instalar Kivy

Kivy es el Framework que ayudara a crear el backend de la aplicación otorgando herramientas para generar interfaces de usuario.

Luego se debe instalar Kivy, tener en cuenta que en este proceso pueden ocurrir errores de dependencias por lo cual hay que apoyarse en la documentación para poder solucionarlos.

`python -m pip install "kivy[full]"`

Documentación: [Installing Kivy — Kivy 2.2.1 documentation](#)

## Instalar Buildozer

Buildozer es una herramienta que permitirá facilitar el proceso de construir una aplicación con Python.

Utilizar los siguientes comandos

`pip3 install --user --upgrade buildozer`

`sudo apt update`

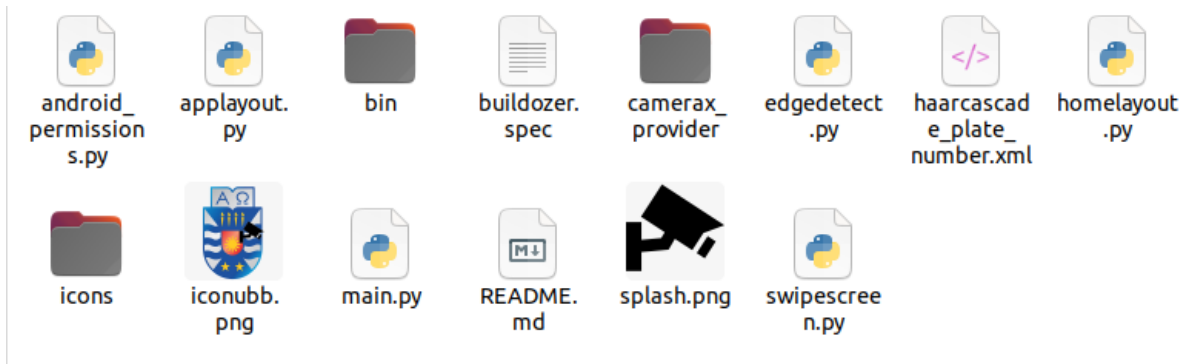
`sudo apt install -y git zip unzip openjdk-17-jdk python3-pip autoconf libtool pkg-config  
zlib1g-dev libncurses5-dev libncursesw5-dev libtinfo5 cmake libffi-dev libssl-dev`

`pip3 install --user --upgrade Cython==0.29.33`

Considerar revisar [documentación de buildozer](#) en caso de tener dificultades.

## Código

Se ha creado un [repositorio Github](#) con el proyecto, a continuación, se explicará cada archivo relevante del proyecto.



### main.py

En este archivo se manejan las ventanas que hay en la aplicación por medio de la función `ScreenManager()`, se utiliza `home` como ventana predeterminada al iniciar la aplicación.

```
37 class MyApp(App):
38
39     def build(self):
40         self.enable_swipe = False
41         self.sm = ScreenManager()
42         self.screens = [
43             HomeScreen0(name='home'),
44             AppLayout(name='applayout')]
45
46         if platform == 'android':
47             Window.bind(on_resize=hide_landscape_status_bar)
48
49         for screen in self.screens:
50             self.sm.add_widget(screen)
51
52         self.sm.current = 'home'
53
54         return self.sm
```

Por otro lado, se utilizan funciones que manejan los permisos de la cámara, además de su inicio y apagado.

```
def on_start(self):
    self.dont_gc = AndroidPermissions(self.start_app)

def start_app(self):
    self.dont_gc = None
    self.enable_swipe = True
    # Can't connect camera till after on_start()
    Clock.schedule_once(self.connect_camera)

def swipe_screen(self, right):
    if self.enable_swipe:
        if right:
            self.sm.transition.direction = 'right'
            self.sm.current = 'home'
        else:
            self.sm.transition.direction = 'left'
            self.sm.current = 'applayout'

def connect_camera(self, dt):
    opencv_screen = self.sm.get_screen('applayout')
    opencv_screen.connect_camera(analyze_pixels_resolution=720,
                                enable_analyze_pixels=True,
                                enable_video=False)

def on_stop(self):
    opencv_screen = self.sm.get_screen('applayout')
    opencv_screen.disconnect_camera()
```

### homelayout.py

Este archivo es una plantilla de una ventana, se propone en trabajo futuro que posea un botón para actualizar datos haciendo una llamada a una base de datos y muestre las matrículas junto con sus fechas.

```
6 class HomeScreen0(SwipeScreen):
7
8     def __init__(self, **args):
9         super().__init__(**args)
10        self.label = Label()
11        self.add_widget(self.label)
12
13    def on_size(self, *args):
14
15        if platform == 'android': # Android
16            text='Aplicación detección de matriculas.\n\n' +\
17                fill('Deslizar a la derecha para usar la cámara')
18
19        self.label.text = text
20
```



## applayout.py

Este archivo maneja la interfaz y visualización de la cámara encendiéndola y manejando la captura de imágenes.

```
class AppLayout(SwipeScreen):
    edge_detect = ObjectProperty()

    def connect_camera(self, analyze_pixels_resolution, enable_analyze_pixels, enable_video):
        self.edge_detect.connect_camera(analyze_pixels_resolution=analyze_pixels_resolution,
                                       enable_analyze_pixels=enable_analyze_pixels,
                                       enable_video=enable_video)
```

```
def photo(self):
    self.parent.edge_detect.capture_photo()
    toast('Capturando matricula')

Builder.load_string("""
<AppLayout>:
    edge_detect: self.ids.preview
    EdgeDetect:
        aspect_ratio: '16:9'
        id:preview
    ButtonsLayout:
        id:buttons

<ButtonsLayout>:

    Button:
        id:screen
        on_press: root.photo()
        height: self.width
        width: self.height
        background_normal: root.normal_camera
        background_down: root.down_camera
""")
```

## edgedetect.py

En este archivo se utiliza OpenCV para detectar la matrícula por medio de un clasificador Haar Cascade, permite dibujar un rectángulo en la vista previa de la cámara.

```

23 def analyze_pixels_callback(self, pixels, image_size, image_pos, scale, mirror):
24     rgba = np.fromstring(pixels, np.uint8).reshape(image_size[1], image_size[0], 4)
25
26     if self.camera_preview_enabled:
27         gray = cv2.cvtColor(rgba, cv2.COLOR_RGBA2GRAY)
28         # Detectar placas
29         plates = self.detect_plates(gray)
30         # Dibujar rectángulos alrededor de las placas detectadas
31         annotated_image = self.annotate_plates(rgba, plates)
32     else:
33         annotated_image = rgba
34
35     pixels = annotated_image.tostring()
36     self.make_thread_safe(pixels, image_size)
37
38 def detect_plates(self, gray):
39     plates = self.plate_cascade.detectMultiScale(gray, scaleFactor=1.1)
40     return plates
41
42 def annotate_plates(self, image, plates):
43     annotated_image = image.copy()
44     for (x, y, w, h) in plates:
45         # Ajustar el grosor y el color del borde del rectángulo
46         cv2.rectangle(annotated_image, (x, y), (x + w, y + h), (0, 255, 0), 3)
47     return annotated_image
48
49 @mainthread
50 def make_thread_safe(self, pixels, size):
51     if not self.analyzed_texture or self.analyzed_texture.size[0] != size[0] or self.analyzed_texture.size[1] != size[1]:
52         self.analyzed_texture = Texture.create(size=size, colorfmt='rgba')
53         self.analyzed_texture.flip_vertical()
54
55     if self.camera_connected:
56         self.analyzed_texture.blit_buffer(pixels, colorfmt='rgba')
57     else:
58         self.analyzed_texture = None
59

```

## swipescreeen.py

En este archivo se encarga de manejar la funcionalidad de deslizar de una ventana a otra.

```

### Para deslizar desde home a la cámara
class SwipeScreen(Screen, CommonGestures):
    def cgb_horizontal_page(self, touch, right):
        App.get_running_app().swipe_screen(right)

```

## buildozer.spec

En se manejan las dependencias que posee la aplicación, el nombre de la aplicación, splasher, icono y varias características de compatibilidad, además de permisos de la aplicación, los cuales también pueden ser manejados en el archivo android\_permissions.py

```

# (list) Application requirements
# comma separated e.g. requirements = sqlite3,kivy
requirements = python3,kivy,camera4kivy,gestures4kivy,opencv,kivymd

```

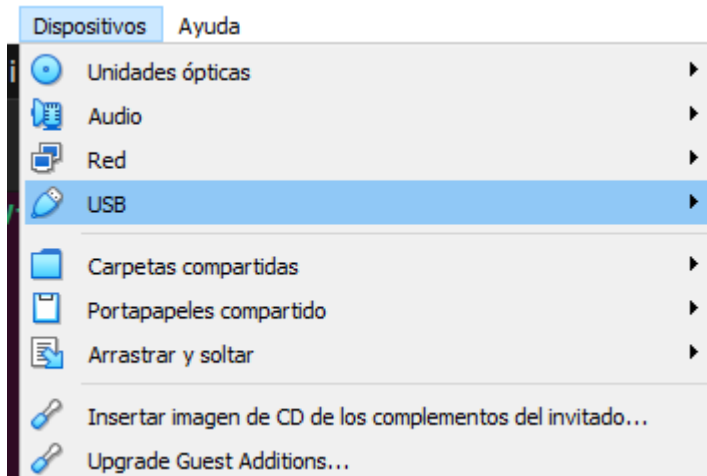
## Generar APK y debugear con dispositivo Android

Para poder debugear con un dispositivo Android es necesario activar el modo desarrollador del dispositivo, para ello debes buscar como activarlo dependiendo del modelo de tu dispositivo.

En consola utiliza el siguiente comando: `sudo apt install adb fastboot`

Ahora puedes conectar el dispositivo vía USB, también puedes revisar el dispositivo en la siguiente pestaña de Virtual Box, o utilizando el siguiente comando en consola:

`adb devices`



Una vez en el directorio del proyecto, teniendo buildozer instalado y con el dispositivo Android conectado puedes utilizar el siguiente comando para generar una APK e instalarla en el dispositivo, una vez generada la APK hay que permitir la instalación en el celular.

`sudo buildozer android debug deploy run logcat`

El archivo .apk se guardará en la carpeta bin.

## Trabajo Futuro

Para la continuación del proyecto se recomienda la implementación de ocr en las imágenes capturadas con el dispositivo Android, como se mencionó al principio la implementación de Android con Python lleva a diversos problemas de compatibilidad, por ejemplo, el ocr implementado con pytesseract en la aplicación de análisis de matrículas automático no pudo realizarse con Android, por lo que se recomendaría buscar alternativas a en la creación de esta aplicación. Por otro lado, la implementación de una base de datos que contenga las matrículas y fechas también es recomendado.

## Referencias

[Diegoj95/Matriculas-CIMUBB: Android app that detects car license plates \(github.com\)](#)

[Nachi3/CIM\\_license\\_plate\\_recognition: Análisis automático de placas vehiculares, realizado en el laboratorio CIM de la Universidad del Bío-Bío. \(github.com\)](#)

[Android-for-Python/Camera4Kivy: Yet Another Camera for Kivy \(github.com\)](#)

[Downloads – Oracle VM VirtualBox](#)

[Download Ubuntu Desktop | Download | Ubuntu](#)

[Installing Kivy — Kivy 2.2.1 documentation](#)

[Installation — Buildozer 0.11 documentation](#)