REVISTA ENERGÍA MECÁNICA INNOVACIÓN Y FUTURO
Vol. 13 Núm. 1 / 2024
CHÁVEZ A., GARZOZI H., ESTRELLA M., Desarrollo de una aplicación de apoyo en el diagnóstico del sistema de inyección electrónica a gasolina utilizando programación Python
Edición No.13/2024 (10) ISSN 1390- 7395 (3/10)
----------Artículo Científico / Scientific Paper

Desarrollo de una aplicación de apoyo en el diagnóstico del sistema de inyección electrónica a gasolina utilizando programación Python
Development of an application to support the diagnosis of the electronic gasoline injection system using Python programming

Anthony Alexander Chávez Alvear [1], Havit Amin Garzozi Calderón [1], Marcelo Xavier Estrella Guayasamín[1]
[1] Universidad Politécnica Salesiana /Facultad de Ingeniería y Ciencias Aplicadas/ GMovInt, Guayaquil, Ecuador

Correspondencia Autores: achaveza3@est.ups.edu.ec, hgarzozi@est.ups.edu.ec,mestrellag@ups.edu.ec,

Recibido: 10 de septiembre 2024, **Publicado:** 18 de diciembre de 2024

Resumen— En el presente estudio ha dado lugar al desarrollo de una aplicación diseñada para servir como guía en las reparaciones automotrices relacionadas con el sistema de inyección a gasolina. La programación de esta aplicación se llevó a cabo utilizando el método ADDIE (Analizar, Diseñar, Desarrollar, Implementar y Evaluar), lo que implicó el uso del entorno de desarrollo integrado (IDE) "Kivy" y la librería "KivyMD". Esto permitió crear una interfaz gráfica compatible con sistemas operativos Android, asegurando un proceso de desarrollo ordenado, respaldado por la elaboración de un diagrama de flujo que sirvió de guía durante la creación de la aplicación. Además, para desarrollar los comandos, se optó por el lenguaje de programación Python, que facilitó la conexión entre el código y la interfaz de usuario. Para la migración del código a Java se utilizó "Buildozer", con el fin de crear la aplicación Android. En cuanto a la adquisición de datos, se utilizaron equipos de medición normalizados como el multímetro ANENG A3005 y el osciloscopio Micsig tBook mini, fundamentales para medir los parámetros normales de funcionamiento de los sensores, información que será utilizada en la aplicación. Como resultado, se obtuvo una aplicación con una interfaz interactiva que ofrece una guía para la localización y reparación de averías en el sistema de inyección.

Palabras clave— *ADDIE, IDE, Python, Buildozer, Java, Android*

Abstract— The present study has led to the development of an application designed to serve as a guide in automotive repairs related to the gasoline injection system. The programming of this application was carried out using the ADDIE method (Analysis, Design, Development, Implementation and Evaluation), which involved the use of the integrated development environment (IDE) "Kivy" and the "KivyMD" library. This allowed for the creation of a graphical interface compatible with Android operating systems, ensuring an orderly development process, supported by the elaboration of a flowchart that served as a guide during the creation of the application. Additionally, to develop the commands, the Python programming language was chosen, which facilitated the connection between the code and the user interface. For the migration of the code to Java, "Buildozer" was used, in order to create the Android application. Regarding the acquisition of data, standardized measuring equipment such as the ANENG A3005 multimeter and the Micsig tBook mini oscilloscope were used, essential for measuring the normal operating parameters of the sensors. As a result, an application with an interactive interface was obtained, offering a guide for locating and repairing faults in the injection system.

Keywords— *ADDIE, IDE, Python, Buildozer, Java, Android*

I INTRODUCCIÓN

En la actualidad la Industria 4.0, ha generado una significativa transformación en la forma en la que se producen, gestionan y diseñan los productos y servicios; la implementación de tecnologías como la inteligencia artificial, la robótica y la automatización de procesos puede resultar en la sustitución de ciertos trabajos manuales por máquinas, ya que representa un impacto significativo en productividad, economía y competitividad de la sociedad [1,2]. Además, ha demostrado ser relevante por su innovación, transformación de las cadenas de suministro y tecnologías avanzadas ya que cuenta con algunos aspectos claves como la inteligencia artificial, la automatización avanzada, el internet de las cosas, entre otros [3].

Su alcance se extiende afectando a diferentes sectores y a las industrias e incluso a la sociedad en general. Es así como la industria automotriz actualmente experimenta una transformación en la implementación de nuevas tecnologías que influyen en la fabricación, diseño, distribución y mantenimiento de los vehículos [4]. Una de esas tecnologías es la de conectividad, que es esencial para facilitar los servicios de conexión y comunicación a los conductores y pasajeros. Otra es el internet de las cosas ya que los vehículos modernos vienen equipados con dispositivos y sensores que recopilan datos a tiempo real y que permiten un monitoreo constante del rendimiento y detección temprana de problemas en el vehículo [5,6].

La industria automotriz ha creado diferentes alternativas para monitorear el motor de un vehículo, incluyendo sensores de lectura que constantemente se comunican con la Unidad de Control Electrónica (UCE) [7]. Estas alternativas contemplan el uso de programas de computadora que se comunican directamente con el vehículo para monitorear el funcionamiento del motor, o bien, utilizan un escáner automotriz para verificar parámetros de funcionamiento. Sin embargo, muchos de estos métodos de detección de códigos de falla son complejos y necesitan de personal capacitado para su correcto uso.

Entre las opciones disponibles se hallan programas informáticos que permiten el monitoreo de sensores y actuadores, facilitando así el diagnóstico de fallas electrónicas automotrices. No obstante, estas herramientas presentan desventajas, como la falta de compatibilidad con todas las marcas de automóviles y su potencial costo elevado tal como ocurre con Scantool de Autoenginuity, ProScan, PCMSCAN, EOBD Facile, OBD2 Auto Doctor, entre otros [8].

En [9], los autores desarrollaron una aplicación móvil denominada "CarAnalyzer", esta aplicación se creó con el fin de diagnosticar los vehículos a través de la interfaz OBD-II y funciona con sistema operativo Android, utiliza conectividad bluetooth e interfaz de transmisión de voz para recopilar información que la aplicación obtuvo en la lectura de datos. La aplicación tiene la capacidad de almacenar datos del usuario, proporcionando información detallada del vehículo y su historial de códigos de falla. Su desarrollo se fundamenta en el estudio [10], está utilizó Android Studio como entorno de desarrollo integrado (IDE) y el lenguaje de programación Java.

Sánchez et al [11], desarrollaron una aplicación móvil para el proceso de gestión automatizada de soporte técnico en un taller de servicio. La aplicación fue creada para uso exclusivo de smartphones con sistema operativo Android. Su función es ingresar datos del vehículo e indicar cuales son los problemas que presenta previo a la cita de mantenimiento, la aplicación está programada con lenguaje TypeScript, el diseño del entorno de la aplicación fue utilizado mediante en el estudio [12], donde se aplicó el Framework Ionic y se codificó con Visual Studio Code utilizando un flujograma de cliente a servidor e implementación de un modelo NoSQL a través de los servicios de Firestore y autentificación de Firebase de Google. Este trabajo se llevó a cabo bajo la guía de [13,14], que aplicaron herramientas de desarrollo eficientes y manejo intuitivo de interfaces para garantizar su seguridad y capacidad de adaptación a las demandas cambiantes de la industria automotriz.

En [15], se desarrolló una aplicación como método de aprendizaje para el mantenimiento de vehículos livianos con sistema de inyección a gasolina. El desarrollo de la aplicación utiliza medios de aprendizaje basados en Android, aplicando el método Analizar, Diseñar, Desarrollar, Implementar y Evaluar (ADDIE). La aplicación contiene un menú de introducción de los sistemas de control electrónico, diagnósticos, inspección y mantenimiento; además, presenta un medio de aprendizaje equipado con texto, imágenes y videos.

En [16], se desarrolló una aplicación para concientizar a la población sobre el significado de las señales de tránsito. Esta aplicación utilizó el método ADDIE para desarrollar un entorno de aprendizaje fácil de usar, lo que la convirtió en un entorno interactivo mediante el uso de Android Studio y el lenguaje de programación Python. La aplicación ofrece un menú de opciones múltiples donde puede el usuario seleccionar cualquier señal de tránsito para informarse sobre los riesgos. Además, cuenta con un medio de aprendizaje equipado con texto, imágenes y un video simulador en 3D.

Debido al incremento de tecnologías, muchos de los técnicos mecánicos sienten el temor de experimentar en estos nuevos sistemas debido a la complejidad y al desconocimiento [17]. Por lo tanto, el desarrollo de nuestra aplicación busca darles una mayor seguridad a los técnicos al tener una guía que les brinde indicaciones paso a paso de lo que tienen que hacer para la localización de la avería.

Este proyecto cobra relevancia al enfocarse en la creación de una aplicación destinada a orientar reparaciones automotrices, especialmente en el ámbito del sistema de inyección. La programación se realizó utilizando las metodologías previamente mencionadas como el entorno desarrollo integrado (IDE) Kivy y la librería KivyMD, garantizando así una interfaz gráfica compatible con dispositivos Android. A diferencia de otras aplicaciones, esta se centra en la identificación de códigos de avería y en la presentación de guías de reparación, ofreciendo opciones de retroalimentación para evaluar problemas y brindar alternativas.

II MÉTODOS Y MATERIALES

Software de programación

Se empleó el programa VSCodium para plasmar la codificación del entorno y programar la aplicación. Este programa es de libre uso y permite trabajar con varios lenguajes de programación [18]. Por otro lado, se utilizó la biblioteca de código abierto Kivy para desarrollar aplicaciones con interfaces nuevas de usuario. Además, se utilizó la librería KivyMD para la interfaz gráfica de usuario [19].

Para las funcionalidades del programa, se aplicó el lenguaje de programación Python con el fin de obtener un flujo de funcionamiento de la aplicación. Maneja un lenguaje de programación multipropósito que permite realizar funciones desde análisis de datos, desarrollo web hasta aprendizaje automático [20].

Los protocolos de diagnóstico se obtuvieron a partir de los manuales de servicio, accediendo al sitio web del National Automotive Service Task Force (NASTF), organización que facilita la resolución de problemas y la comunicación entre técnicos automotrices, fabricantes y propietarios de vehículos [21].

Equipos de medición

Para las mediciones, se emplearon herramientas específicas como: el osciloscopio Micsig tBook mini, con dos canales de conexión para obtener oscilogramas de cada uno de los sensores, el multímetro ANENG A3005, tipo bolígrafo con un rango de medición AC/DC de 0.8V a 600V y un rango de medición de resistencia de 0.1Ω a 40MΩ. Estas herramientas se utilizaron para medir parámetros de funcionamiento y validar que se cumplan con lo especificado en el manual de servicio.

Metodología

La metodología para la creación de la aplicación base se muestra enumerados los procesos del desarrollo de la aplicación en la Figura 1, el primer paso es la creación de interfaz de la aplicación utilizando VSCodium, un programa de código abierto que permite la creación de programas en varios lenguajes de programación (1). Luego, mediante el uso de Python se enlazo fácilmente varias librerías y dependencias, mejorando la conexión entre el código y la interfaz de usuario [22].

A continuación, se utilizó la información de la librería KivyMD para obtener los comandos necesarios y crear las clases individuales para cada pantalla con el propósito de organizar los componentes y códigos por funcionalidad (2). Cada pantalla se definió para asegurar que la aplicación tuviera acceso a una biblioteca completa de todos los segmentos registrados (3 y 4). Dentro de Kivy, se hizo referencia a las pantallas previamente definidas en las clases y se comenzó a construir el contenido de acuerdo con los requisitos del programa (5 al 7) [23].

Luego, para la migración del lenguaje de programación Python a Android se necesitó un traductor intermedio, que se conoce como "Buildozer" este migrador de código y creador de APK se utilizó por medio de Ubuntu, el cual se instaló desde la tienda de Windows (8) [24]. Este script de Ubuntu permitió traducir el código de Python y Kivy enlazando las funciones e interfaz gráfica, realizando una recopilación de cada una de las imágenes y documentos utilizados traduciendo de Python a Java, lenguaje el cual es compatible con sistemas Android y permite la lectura de archivos APK [25].

Por último, para desarrollar la aplicación, se implementó el método ADDIE con el fin de crear una aplicación interactiva para dispositivos Android. En primer lugar, se recopilaron datos, incluyendo parámetros normales de funcionamiento, características del sensor y señales eléctricas. Para la toma de datos de los sensores, se utilizaron herramientas como el multímetro ANENG A3005 para verificar que estén trabajando dentro de los rangos de voltaje y resistencia especificados en el manual de servicio, y un osciloscopio Micsig tBook mini para captar las señales eléctricas de los sensores y representarlas como ondas.

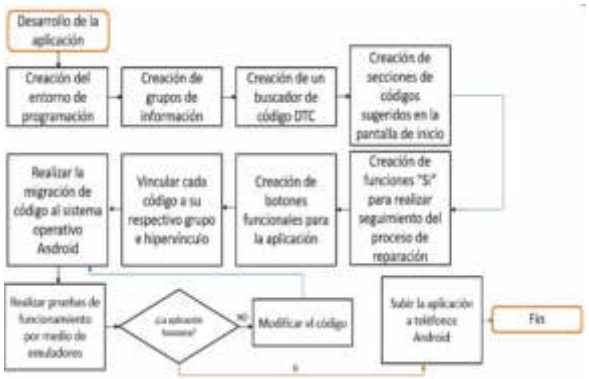

Fig. 1. Flujograma del proceso del desarrollo de la aplicación

Migración de Python a Java

El sistema operativo Android se basa en Java y sus Interfaces de Programación de Aplicaciones (APIs), lo que permite la ejecución de aplicaciones en formato APK. Debido a esta estructura, Android no es compatible directamente con Python. Por lo tanto, para ejecutar programas Python en dispositivos Android, dependen de la descarga de aplicaciones de terceros que interpretan el código Python a la migración del código a un lenguaje compatible con Android.

La metodología empleada para migrar de Python a Java se basó en el uso de Kivy, una plataforma especializada en la creación de aplicaciones Android. Buildozer, una extensión de Kivy, facilitó el proceso de migración y compilación de toda la información, imágenes y archivos utilizados en Python. Este procedimiento estableció una conexión funcional entre los archivos Python y Kivy, lo que permitió la generación de una interfaz gráfica operativa. Posteriormente, Buildozer tradujo este código junto con el de Kivy a Java, e implementó las dependencias necesarias para garantizar

su funcionamiento en Android y optimizar el tamaño del archivo APK resultante [26].

Vehículos de estudio

Esta aplicación fue diseñada para el modelo Kia Sportage R del 2016 con un motor 2.0 EX DOHC, transmisión automática de 4 y 16 válvulas, una potencia máxima de 6200 rpm y con un torque máximo de 19.5 kg/m. Un Kia Picanto del 2023 con un motor 1.2 DOHC CVVT dual de 16 válvulas 4 cilindros en línea, transmisión manual de 5 velocidades, con una potencia máxima de 83 hp y un torque máximo de 122 Nm y Kia Soluto del 2020 con un motor 1.3 DOHC CVVT dual de 16 válvulas 4 cilindros en línea, transmisión manual de 5 velocidades, con una potencia máxima de 94 hp y un torque máximo de 133 Nm [27].

La información sobre las especificaciones del fabricante, los parámetros de funcionamiento y los protocolos de diagnóstico se obtuvieron a partir de los manuales de servicio adquiridos a través de NASTF [28]. Se recopilaron los valores característicos y oscilogramas de los sensores y actuadores del motor en funcionamiento normal para alimentar la aplicación. Los sensores y actuadores incluidos fueron: Sensor CKP, Sensor CMP, Sensor MAP, Sensor IAT, Sensor de Oxígeno, Sensor Knock, Sensor ECT, Bomba de combustible, Sensor TPS, Sensor APS e Inyector.

III. PRUEBAS Y RESULTADOS

Entorno de la aplicación

A continuación, se explicará detalladamente la fase de creación del entorno, la sección de búsqueda dentro de grupos, la organización de los grupos y la incorporación de gráficas.

Fase de creación

RadLab es una compañía ecuatoriana especializada en Se empleó VSCodium para generar el ambiente visual, siguiendo el Algoritmo 1, que define la clase y tipos de variables dentro de la interfaz de usuario. Además, este algoritmo incluye la función de visualización de grupos, como se muestra en la Figura 2.

Fig. 2. Ambiente visual

```
#Se definen la selección de
vehículos class BaseSportageScreen:
    def pagsen(self):
        app = App.get_running_app()
        pagina_sensor = app.pagina_sensor
        print(f"Previous Screen:
        {pagina_sensor}") if pagina_sensor:
            app.root.transition.direction = 'right'
            app.root.current = pagina_sensor
    def go_back(self):
        app = App.get_running_app()
        pagina_anterior = app.pagina_anterior
        print(f"Previous Screen:
        {pagina_anterior}") if pagina_anterior:
            app.root.transition.direction = 'right'
            app.root.current = pagina_anterior
class Seleccion(Screen):
    def on_group_button_press(self,
        group): app =
        App.get_running_app()
        app.selected_group = group
        app.root.current = 'Intro'
```

Algoritmo 1. Creación del entorno

Para la creación de la función de búsqueda dentro del grupo seleccionado y la opción de regresar, se utiliza el Algoritmo 2. En la Figura 3 se muestra el entorno de búsqueda dentro de un grupo específico, lo que permite que el programa vincule únicamente la información relacionada con el grupo previamente elegido.

```
class Intro(Screen):
    def
    Buscador(self):
        app = App.get_running_app()
        screen_name =
        self.ids.input_text.text group_name
        = app.selected_group
#Se utiliza para ver que grupos se están
        seleccionando if hasattr(app,
        'screen_manager'):
                                    screens_in_group

= app.search_by_group(group_name)
            print("Codigos en el grupo:",
screens_in_group) #Esta función revisa que se está
buscando y en qué grupo se encuentra
            if screen_name in screens_in_group:
                app.screen_manager.transition.directi
                on =
'left'

                app.screen_manager.current =
            screen_name else:
                print("Código", screen_name)
                    self.display_error_message(f'El
                    código
'{screen_name}'
```

Algoritmo 2. Sección de búsqueda dentro del grupo

Fig. 3. Búsqueda dentro del grupo

Funciones de organización

Por otra parte, el Algoritmo 3 permite establecer los dígitos organizados por códigos de avería y establece un formato de búsqueda dentro del entorno. Esta codificación organiza los grupos en dígitos como se muestra en la Figura 4 que contiene la información del código de falla obtenido del manual de servicio.

```
class Inspeccion_P0030(Screen,
   BaseSportageScreen): def pagsen(self):
       app = App.get_running_app()
       pagina_sensor = app.pagina_sensor
          print(f"Previous Screen,
          BaseSportageScreen:

{pagina_sensor}")

       if pagina_sensor:
          app.root.transition.direction =
          'right' app.root.current =
          pagina_sensor
screen_groups = {

        'Sportage':                ['Intro','0030sportage',
'0031sportage',    '0032sportage',    '0036sportage',
'0037sportage',    '0038sportage',    '0106sportage',
'0107sportage', …]
```

Algoritmo 3. Sección de organización por códigos de falla

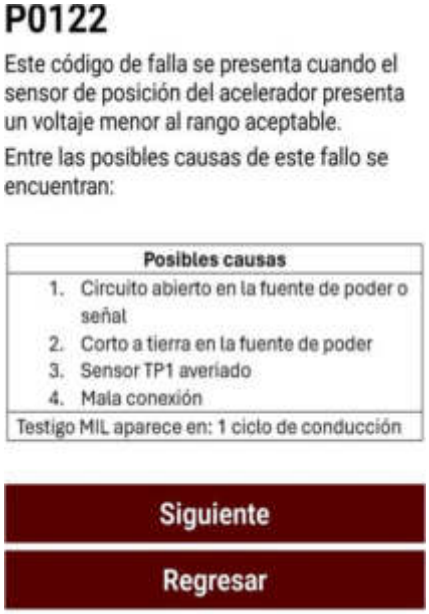

Fig. 4. Organización por códigos de falla

Función de colocación de diagramas

Por último, en el Algoritmo 4 se muestra la función de añadir imágenes, esta tiene el fin de insertar imágenes en formato gráfico y establecer su resolución en dimensión estándar y en aumento. La Figura 5 permite observar el circuito eléctrico, así como la información sobre los pines del sensor y la