Diseño e implementación de un generador de código fuente a partir de diagramas de flujo manuscritos

Alumno: Juan Antonio Lagos Carrera

Nombre del programa: Sidekick-S

Área de trabajo final: Aplicaciones multimedia de nueva generación

Nombre del Tutor de TF: David García Solórzano

Índice

Introducción

- Generar código fuente a través de diagramas de flujo manuscritos.
- Ayuda a los estudiantes STEM en sus inicios en la programación.
- Aplicación standalone pero con posibilidad futura de implementación en una API para diversos usos.

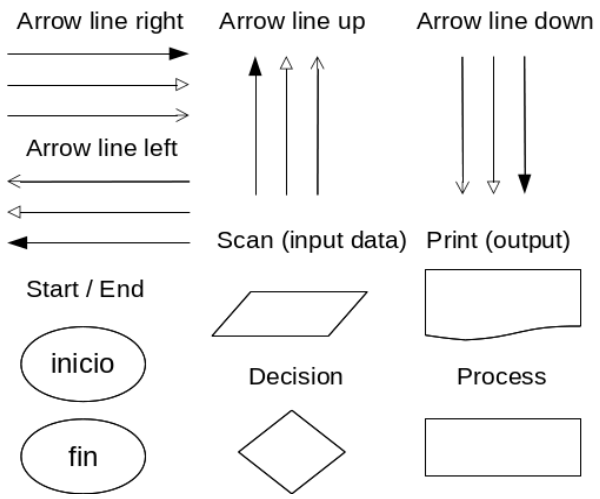
Objetivos del trabajo

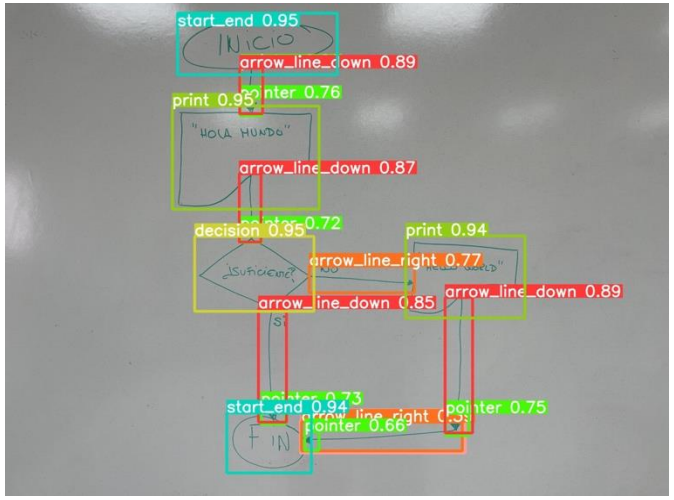
- Detección de objetos en imágenes.
- Sistema OCR.
- Ordenación de los elementos del diagrama.
- Generar el pseudocódigo del diagrama.
- Empleo de ANTLR4 para generar código fuente.

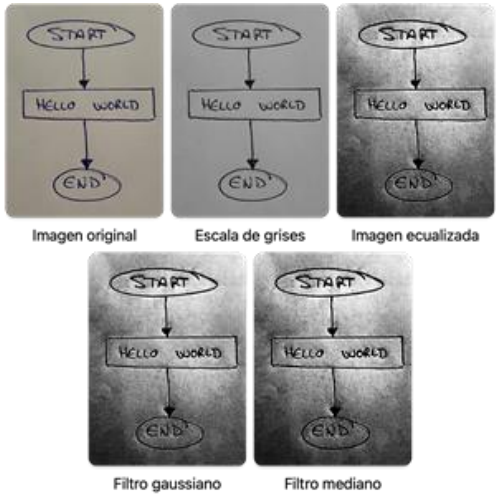
Reconocimiento de objetos en imágenes

- TensorFlow Lite → YOLOv5
- Precisión: 80% → 99%

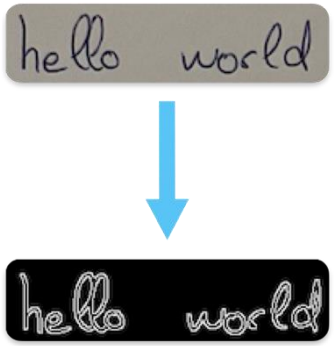
- Modificaciones en las imágenes
  - Paso a escala de grises
  - Ecualización
  - Filtros de distorsión gaussiano y mediano







OCR

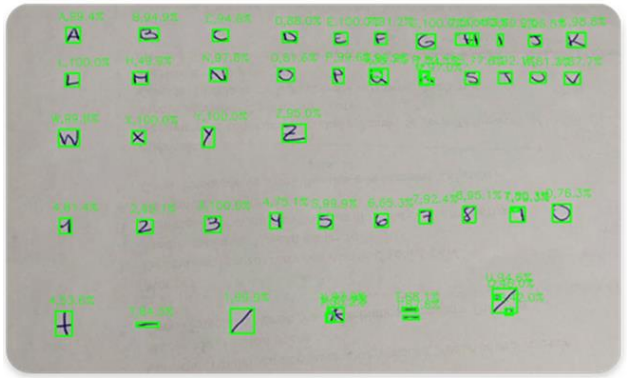


- Tesseract + EasyOCR

Malos resultados

- Sistema OCR propio → Keras + TensorFlow + OpenCV
  - Datasets MNIST + AZ symbols + math symbols

Malos resultados

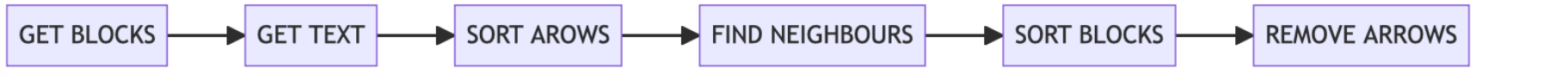


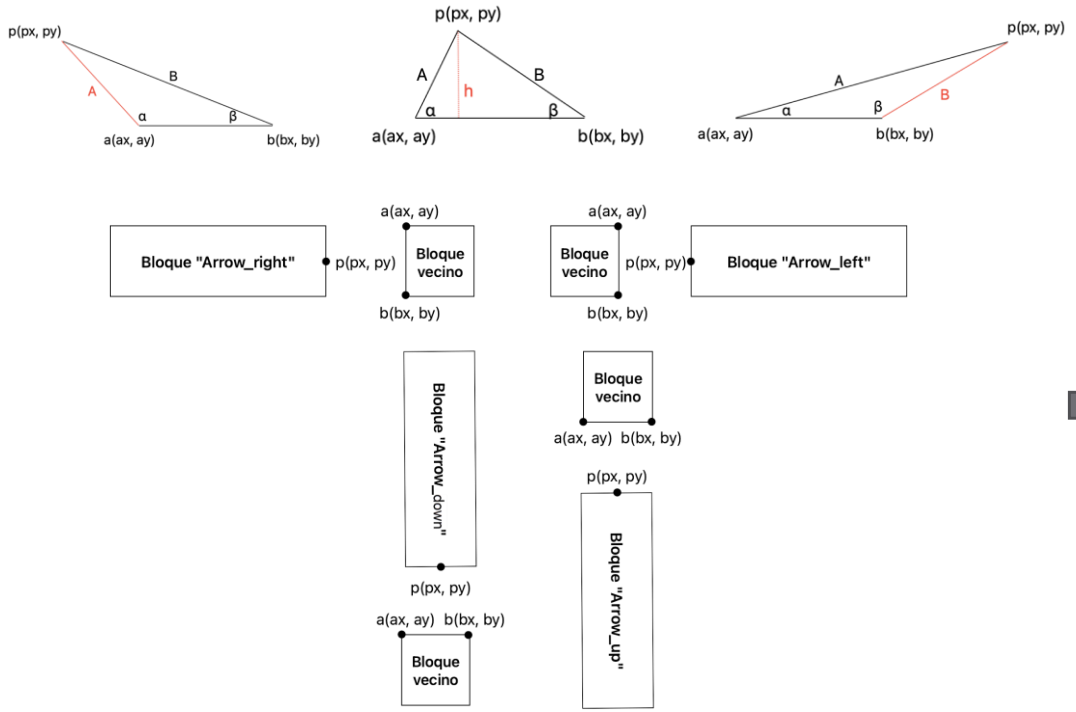
- Algoritmo de corrección de detecciones

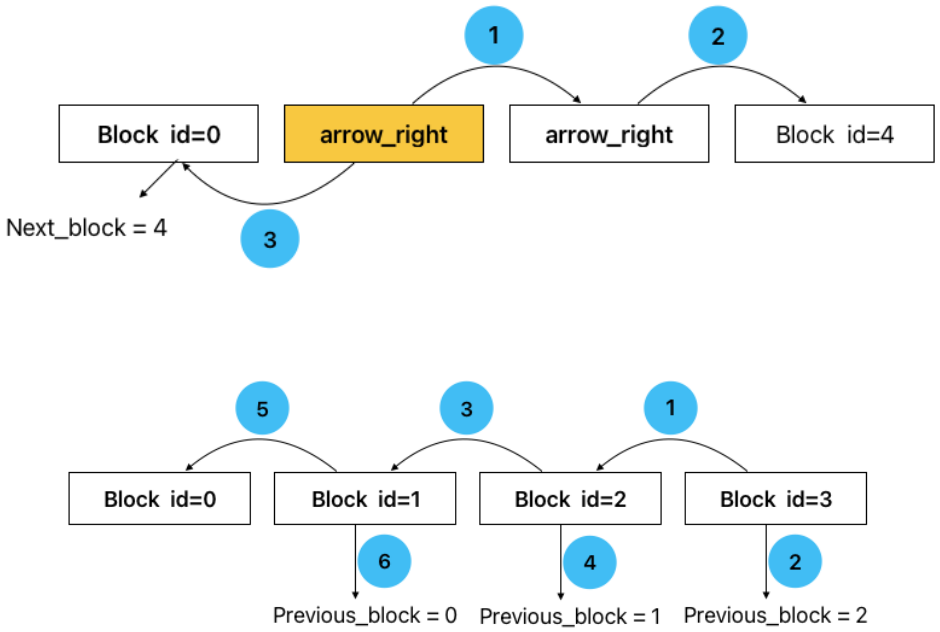
Malos resultados

- Azure OCR

Algoritmo de ordenación

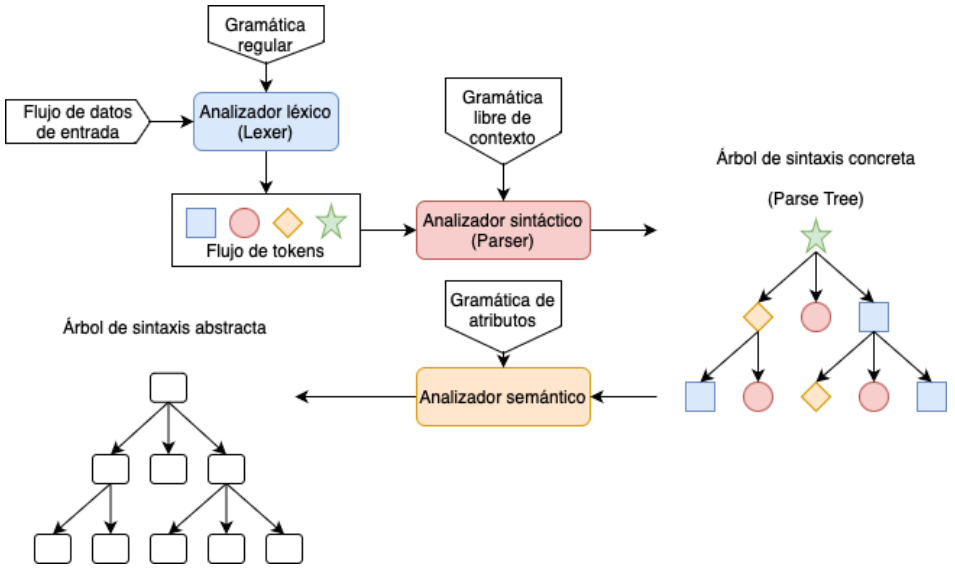






Parser generator (ANTLR4)

- Pseudocódigo en función del tipo de bloque:
  - <PRINT> <CONDITIONAL>
- Fichero de gramática genera el flujo de tokens y el árbol de sintaxis concreta.
- Implementando el patrón de diseño Visitor (analizador semántico) se efectúa el paso a código fuente




```
def __handle_scan__(self, expression: Scan) -> str:
    if self.variables[expression.get_var()] == Constants.INTEGER:
        return "{0} = {1}.nextInt();".format(expression.get_var(), self.scanner_var)
    elif self.variables[expression.get_var()] == Constants.STRING:
        return "{0} = {1}.nextLine();".format(expression.get_var(), self.scanner_var)
```

Java

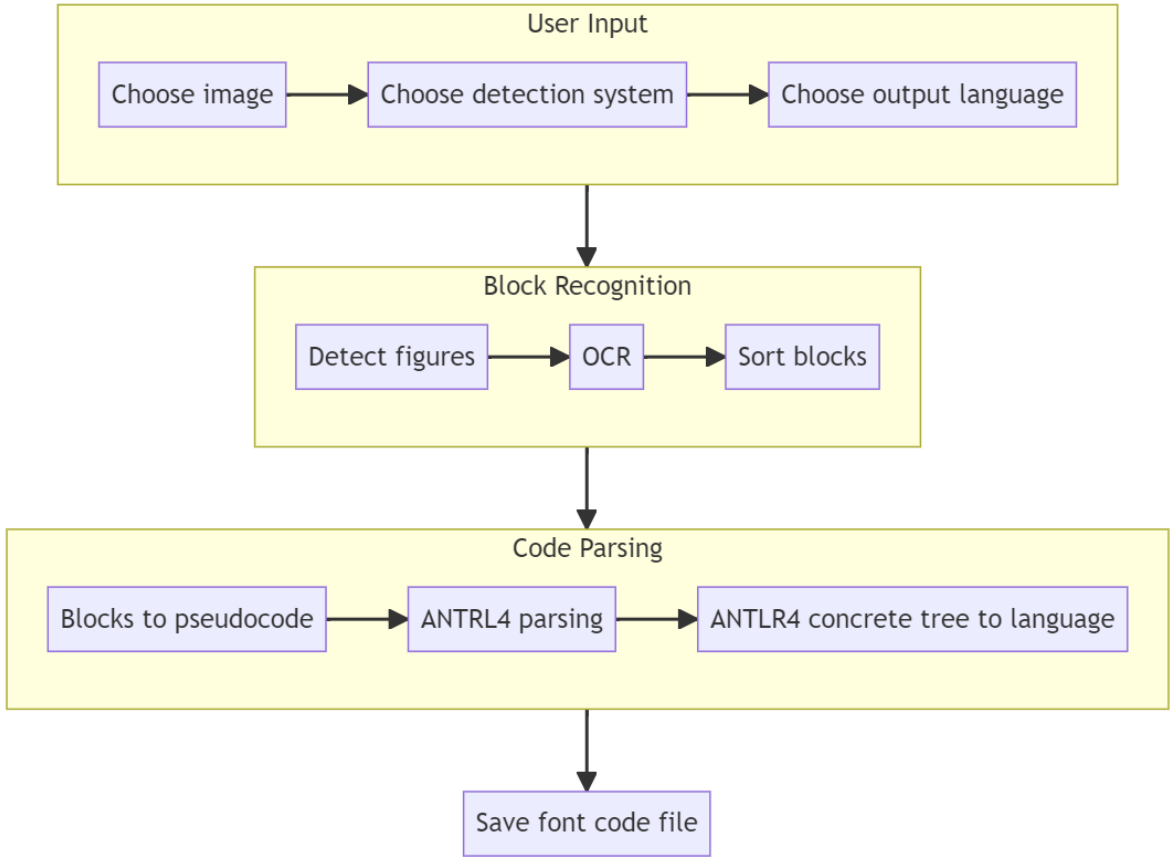
```
@abstractmethod
def __handle_switch_case__(self, expression: Conditional) -> [str]:
    pass
```

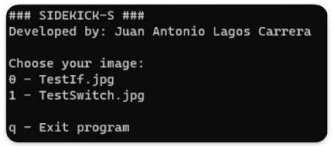
Clase abstracta

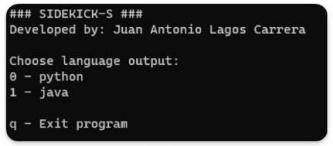
```
def __handle_scan__(self, expression: Scan) -> str:
    return "{0} = input()".format(expression.get_var())
```

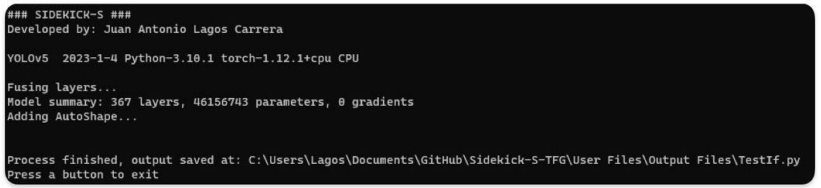
Python

Flujo del programa

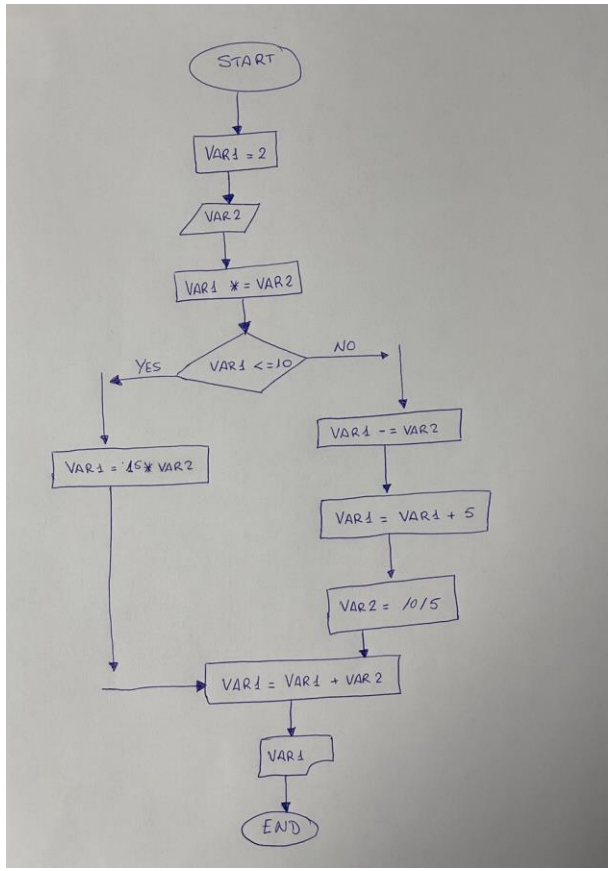








Ejemplo 1: condicional If

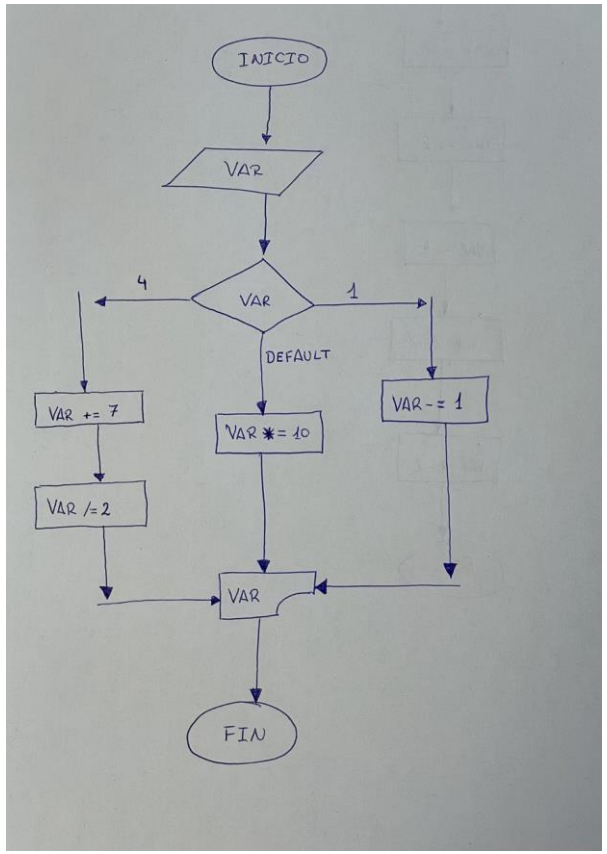



```
<BASE_FUNC>
<VAR_DECLARATION>VAR1<INT>
<VAR_DECLARATION>VAR2<INT>
VAR1=2
<SCAN>VAR2
VAR1*=VAR2
<CONDITIONAL> VAR1 <= 10
        <CONDITION> YES
                VAR1=15*VAR2
        </CONDITION>
        <CONDITION> NO
                VAR1-=VAR2
                VAR1=VAR1+5
                VAR2=10/5
        </CONDITION>
</CONDITIONAL>
VAR1=VAR1+VAR2
<PRINT>VAR1
<END>
```

```python
def main():
        VAR1: int = None
        VAR2: int = None
        VAR1 = 2
        VAR2 = input()
        VAR1 *= VAR2
        if VAR1 <= 10:
                VAR1 = 15 * VAR2
        else:
                VAR1 -= VAR2
                VAR1 = VAR1 + 5
                VAR2 = 10 / 5
        VAR1 = VAR1 + VAR2
        print(VAR1)
```

```java
import java.util.Scanner;
public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int VAR1 = null;
        int VAR2 = null;
        VAR1 = 2;
        VAR2 = scan.nextInt();
        VAR1 *= VAR2;
        if (VAR1 <= 10) {
                VAR1 = 15 * VAR2;
        } else {
                VAR1 -= VAR2;
                VAR1 = VAR1 + 5;
                VAR2 = 10 / 5;
        }
        VAR1 = VAR1 + VAR2;
        System.out.println(VAR1);
}
```

Ejemplo 2: condicional Switch-Case




```
<BASE_FUNC>
<VAR_DECLARATION>VAR<INT>
<SCAN>VAR
<CONDITIONAL> VAR
        <CONDITION> DEFAULT
                VAR*=10
        </CONDITION>
        <CONDITION> 1
                VAR-=1
        </CONDITION>
        <CONDITION> 4
                VAR+=7
                VAR/=2
        </CONDITION>
</CONDITIONAL>
<PRINT>VAR
<END>
```

```
def main():
        VAR: int = None
        VAR = input()
        if VAR == 1:
                VAR -= 1
        elif VAR == 4:
                VAR += 7
                VAR /= 2
        else:
                VAR *= 10
        print(VAR)
```

```
import java.util.Scanner;
public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int VAR = null;
        VAR = scan.nextInt();
        switch (VAR) {
                case 1:
                        VAR -= 1;
                        break;
                case 4:
                        VAR += 7;
                        VAR /= 2;
                        break;
                default:
                        VAR *= 10;
        }
        System.out.println(VAR);
}
```

Conclusiones

- Necesario un procesado de la imagen antes de ser inferenciada (paso a escala de grises)
- No es necesaria la generación de una red neuronal para la detección de objetos, existen alternativas con un gran rendimiento (YOLO)
- Los mecanismos OCR no funcionan correctamente con texto manuscrito
- Gran potencial de ANTLR4, no solo flujos de texto

Trabajos futuros

- Mejora del sistema OCR generado
- Mejorar la arquitectura de software
- Generar una API
- Reducir las limitaciones existentes para el dibujo del diagrama de flujo
- Agregar soporte para la detección de diagramas UML

DEMO

Muchas gracias por su atención