UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Paradigmas de la programación

Informe de laboratorio N°2

Software de edición o manipulación de imágenes

Andres Zelaya

Profesor Gonzalo Martínez

2 de noviembre 2022

El presente informe corresponde al laboratorio N°2 del curso; donde en cada experiencia se trabajará utilizado un paradigma de la programación distinto.

Siendo utilizado en esta ocasión el Paradigma lógico, donde es ocupado el lenguaje de programación Prolog a través del compilador Swish-Prolog.

En el informe sigue se cubrirán los siguientes tópicos:

- Descripción breve del problema
- Introducción al paradigma
- Objetivos del proyecto
- Análisis del problema,
- Diseño de solución del problema
- Aspectos de implementación
- Instrucciones y ejemplos de uso
- Resultados y autoevaluación
- Conclusión

Descripción del problema

El proyecto de este semestre es la creación de un software de edición y/o manipulación de imágenes digitales, es decir, que permita a un usuario realizar distintas operaciones sobre estas: rotar una imagen, invertirla, retocarla, transformarla y redimensionarla, entre otras. Algunos softwares más conocidos, capaces de realizar esta tarea son GIMP y Adobe Photoshop.

Este proyecto se concentrará en trabajar en imágenes RGB-D o RGB-D, esto es imágenes que además de tener información en el espacio de colores (R)ed, (G)reen, (B)lue, contiene información de la profundidad (D)epth en un espacio tridimensional, por lo que el desafío de esta experiencia es implementar lo descrito anteriormente de forma que se acople al paradigma lógico, de modo que se puedan realizar las operaciones mencionadas.

De esta forma, en la imagen bidimensional de la Figura 1 (mostrada en el apéndice), donde es posible distinguir los colores en el espectro RGB; al incorporar la dimensión (D)epth capturada a través de una cámara especializada, sería posible saber más sobre los detalles del rostro, proyección de la nariz, sombrero, distancia del espejo en la parte posterior, etc. Incluso sería posible construir una representación tridimensional del rostro, como se ilustra en la Figura 2.

Descripción del paradigma

Una forma de razonar para resolver problemas en matemáticas se fundamenta en la lógica de primer orden: se puede representar en forma de axiomas, a los que se añaden distintas reglas para deducir cosas verdaderas (teoremas) a partir de estos mismos.

Gracias al trabajo de algunos matemáticos de finales del siglo pasado y principios de este, se encontró la manera de automatizar computacionalmente la lógica matemática, en particular para un conjunto significativo de la lógica de primer orden, lo que permitió dar origen a otros tipos de lenguajes de programación, conocidos como *lenguajes lógicos*.

En el Paradigma Lógico se tienen algunos elementos bastante importantes a la hora de construir un código utilizándolo:

Átomo: aquellas cosas sobre las que se basa el conocimiento que queremos expresar (Se escriben en minúsculas).

Predicado: lo que queremos decir. Los resultados o variables van en mayúsculas.

Consultas: preguntas que se hacen mediante consola y donde Prolog busca en la base de conocimientos para entregar *True* o *False.* También puede entregar un elemento que satisfaga una consulta para que sea verdadera, en el caso de que se haya introducido una variable.

Hechos: cada una de las sentencias se “iguala a” unidades de información de una base de conocimiento. Los hechos y reglas deben terminar con un punto.

Objetivos

Aprender sobre el Paradigma y la programación lógica, para así obtener la habilidad de programar de forma distinta a la clásica, y que muchas personas están acostumbradas.

Programar correctamente en Prolog y aprender a utilizar las herramientas de este para completar el proyecto de laboratorio y tener una base para otros próximos y proyectos que en un futuro se desarrollen con la programación lógica.

Análisis del problema

Primero que todo, debido a la forma en que se ocupa el paradigma y al no existir el uso de funciones, es necesario trabajar mediante reglas y hechos, ya que en este caso Prolog hace todo el trabajo de forma interna a la hora de realizar una consulta.

A grandes rasgos todo lo implementado son transformaciones hipotéticas, que se realizan a un dato ingresado, estos cambios siguen reglas dadas que ya fueron previamente definidas en el proceso de programación.

A su vez, a la hora de hacer funcionar el programa y hacer transformaciones a una imagen, esta tendrá que haber sido determinada con antelación en la base de conocimientos o incluirse como parte de la consulta.

También es necesario implementar distintos tipos de datos abstractos (TDA) para cada tipo de píxel que compone una imagen, ya que las imágenes pueden contener de estas 3 variedades.

Diseño de solución

Para diseñar la solución, no solo es necesario utilizar los tipos de datos nativos de *Prolog*, sino que también se deben crear tipos de datos específicos, a través de lo que se conoce como TDA.

Los implementados en este laboratorio fueron:

TDA pixbit-d

- TDA pixrgb-d
- TDA pixhex-d
- TDA image

Sin embargo, cabe destacar que en *Prolog*, un predicado puede actuar como muchas cosas a la vez. Los TDAs creados son utilizados para la construcción de las operaciones de otros TDAs y para las operaciones obligatorias y opcionales.

Cada TDA tiene la siguiente estructura:

- Representación
- Constructores
- Funciones de Pertenencia
- Selectores
- Modificadores
- Otras funciones asociadas al TDA.

Vale mencionar que como la mayoría de las funciones requeridas son para una imagen, estas estarán en el TDA imagen.

Primero hay que hablar de pixeles, ya que son la unidad menor de una imagen, cada uno tendrá:

- **Posición:** Ya que la representación de imagen utilizada es una matriz con pixeles, cada uno debería tener una posición X e Y, que será representada mediante 2 valores que hacen el rol de posición X e Y respectivamente.

- **Contenido:** El contenido de cada píxel puede variar dependiendo de su tipo, si es bit tendrá solo un valor que podría ser 0 o 1, si es tipo RGB tendrá tres valores que

oscilarán entre el 0 y 255, y finalmente si es tipo hexadecimal tendrá un respectivo código indicado un color en este formato.

- **Profundidad:** Como se menciona anteriormente, esta implementación contara con posibilidad de manejo de imágenes en 3D, por lo que este argumento está pensado para representar la profundidad en cada píxel de una imagen.

- **ID o Píxel completo:** El ultimo parámetro se ocupa para almacenar ya sea la posible ID del píxel o una lista con todos los parámetros del píxel.

Aspectos de implementación

Para este proyecto es necesario el IDE *Swi-Prolog*, específicamente de versión 8.x.x o superior. Como alternativas, también es posible utilizar *Visual Studio Code* con la extensión de *Swi-Prolog* o bien, se puede hacer uso de SWISH (*Prolog* online).

La implementación se baja en el trabajo con listas, sin embargo, se espera que algunas operaciones de listas como por ejemplo member o cualquiera que no sea *append*, *reverse* o *índex* sean implementadas.

Instrucciones de uso

Primero, se debe cargar la base de conocimientos. Simplemente se debe ir a la opción *File->Consult* y luego seleccionar el archivo “main_20095832_ZelayaDrogett.pl”. Si todo sale bien, *Prolog* dará como respuesta “*true*.”, lo cual significa que la base de conocimiento fue cargada con éxito.

Una vez hecho esto, se pueden hacer por consola las distintas consultas, ya sean transformaciones de imágenes u otros. Lo que la consola mostrará serán las distintas posibilidades de respuestas a las consultas del usuario.

Resultados esperados

Se espera que todas las consultas en el *main.pl* funcionen sin problemas.

Se espera poder trabajar exitosamente con imágenes de distintos tipos, donde la implementación del programa sin errores. Todo esto logrado mediante un buen uso de listas y recursiones, que son fundamentales para la programación lógica.

Finalmente se espera una implementación correcta de los TDA.

Posibles errores

El error más probable es que al ejecutar el archivo principal, el usuario se equivoque en escribir el comando para consultar la base de conocimiento.

Resultados obtenidos

Los resultados obtenidos fueron los esperados, ya que se lograron definir las reglas correctas para poder realizar consultas donde los resultados fueron los esperados.

Se hicieron múltiples pruebas con distintos ejemplos para probar de que no hubiera fallos en la ejecución del código y que el código hiciera lo correcto, lográndose la implementación de 16/18 definiciones.

Autoevaluación

Las funciones que si fueron implementadas funcionan en la totalidad de las veces probadas.

La Autoevaluación se realiza de la siguiente forma: 0: No realizado – 0.25: Funciona 25% de las veces – 0.5: Funciona 50% de las veces 0.75: Funciona 75% de las veces – 1: Funciona 100% de las veces.

Se adjunta una taba de autoevaluación individual por funciones como anexo en la tabla 1.

Conclusiones

Luego de realizar y casi completar el proyecto, se puede concluir que se cumplieron los objetivos principales, ya que fue posible aprender a utilizar correctamente *Prolog* para poder completar el proyecto de laboratorio correspondiente.

Si bien la implementación en términos generales tuvo sus dificultades, lo más difícil de esta experiencia fue poder separar los TDA en distintos archivos, ya que la exportación e importación de modelos en este lenguaje de programación es relativamente compleja.

Por otro lado, no se experimentaron complicaciones por el lado de la instalación o uso del compilador ni por el aspecto del versionamiento con *Git*.

Bibliografía y referencias

1. Flores, V. (2021). “Proyecto Semestral de Laboratorio”. Paradigmas de Programación. Enunciado de Proyecto Online. Recuperado de: https://docs.google.com/document/d/1pORdJwp5WJ7V3DIAQik9B58llpdxpurQRVTKPZHOpS8/edit
2. Flores, V. (2021). “4 - P. Lógico”. Paradigmas de Programación. Material de clases Online. Recuperado de: https://uvirtual.usach.cl/moodle/course/view.php?id=10036&section=16
3. Martinez G. (2021) “Logic Paradigm” https://github.com/USACH-A1-Programming-Paradigms/Logic-Programming
4. Chacon, S. y Straub, B. (2020). “Pro-Git – Todo lo que necesitas saber sobre Git’’. Libro Online. Recuperado de: https://drive.google.com/file/d/1mHJsfvGCYcIhdrmK lBI6a1WS-U1AAPi/view
5. Multiples Autores. (Septiembre 2013). “El lenguaje de programación PROLOG”. Libro online. Recuperado de:

https://drive.google.com/file/d/1DgxI64oAB5do7A ajCXCTphnGWHTCEmk/view?usp=sharing

6. Autores Desconocidos. (2021). “Swi-Prolog Documentation’’. Swi-Prolog. Documentación Online. Recuperado de: https://www.swi prolog.org/pldoc/index.html

Anexos

Figura 1. Lenna, una de las imágenes más empleadas (desde 1973) como imágenes estándar de prueba de algoritmos de compresión
Fuentes:
https://taglang.io/blog/post/Understanding_Kernel_Convolution_Part_2/

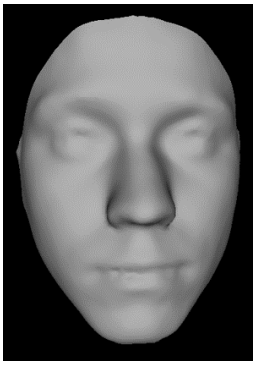

Figura 2. Reconstrucción sintética de un rostro a partir de una imagen RGBD.
Fuente:
https://www.researchgate.net/publication/316442804_Facial_depth_map_enhancement_via_neighbor_embedding/figures?lo=1

Tabla 1.

Autoevaluación de requerimientos funcionales.

TDAs	0.5
TDA image - constructor	1
TDA image - is bitmap	1
TDA image – is pixmap	1
TDA image - is hexmap	1
TDA image - is compressed	1
TDA image - flipH	1
TDA image - flipV	1
TDA image - crop	1
TDA image - imgRGB->imgHex	1
TDA image - histogram	1
TDA image - rotate90	1
TDA image - compress	1
TDA image - changePixel	0
TDA image - invertColorRGB	1
TDA image - image->string	0.25
TDA image - depthLayers	0
TDA image - decompress	0