IMAGine

Francesco Paolo Castiglione, Davide Iraci, Andrea Montemaggiore

Settembre 2020

Indice

1 Introduzione **2**

2 Stato dell'arte **3**

3 Descrizione del progetto **4**
3.1 Scelte progettuali . . . 4
3.2 LibVips . . . 4
3.3 Guida all'uso . . . 8

4 Caratteristiche del linguaggio **9**
4.1 Grammatica . . . 9
4.1.1 Tipi . . . 9
4.1.2 Operazioni . . . 9
4.1.3 Strutture di controllo . . . 10
4.1.4 Funzioni . . . 10
4.2 Parser . . . 11
4.3 Lexer . . . 13
4.4 Funzionalità del linguaggio . . . 15
4.4.1 Metodo print . . . 15
4.4.2 Metodi per le liste . . . 15
4.4.3 Metodi per le immagini . . . 15

5 Casi d'uso **18**
5.1 Primo caso . . . 19
5.2 Secondo caso . . . 21
5.3 Terzo caso . . . 23

6 Risultati **25**
6.1 Fattoriale . . . 25
6.2 Pi greco . . . 27
6.3 Restauro . . . 29

7 Conclusioni **31**

Appendice **33**

1 Introduzione

IMAGine (IMAge-enGINE) è un linguaggio ideato per l'elaborazione delle immagini a trecentosessanta gradi. Il linguaggio consente le elaborazioni più comuni attraverso un processo immediato ed intuitivo. E' stato pensato e sviluppato tenendo conto di alcuni fattori chiavi quali semplicità d'uso e rapidità di apprendimento, basandosi anche su altri linguaggi già esistenti quali *Python*, *MATLAB* o software per la modifica di immagini.
Il resto della relazione è strutturato come segue: la `sezione 2` illustra il contesto in cui si colloca il linguaggio proposto, nella `sezione 3` vengono descritte le scelte progettuali, nella `sezione 4` si illustrano le caratteristiche di *IMAGine*, nella `sezione 5` vengono esposti i possibili casi d'uso, nella `sezione 6` sono mostrati i risultati dei confronti fra `IMAGine`, *Python* e *MATLAB* nell'esecuzione di classici calcoli per effettuare benchmark quali il calcolo del pi greco, del fattoriale ed inoltre il caso del restauro, fondamentale nel dominio applicativo in esame. Nella `sezione 7` vengono riepilogati i punti principali del lavoro svolto e tratte le conclusioni.

2 Stato dell'arte

Attualmente l'Image Processing avviene principalmente per mezzo di software suites già pronte quali Photoshop, Gimp, etc. Questi software sono stati pensati per un utilizzo da parte di amanti delle immagini e per professionisti in campo fotografico. Alcuni di questi, come il già citato Adobe Photoshop, vengono proposti in una suite software chiamata "Adobe Creative 2020", la quale, nella sua ultima edizione, comprende altri applicativi per l'Image Processing di livello sempre più alto. Per lo sviluppo del progetto ci siamo principalmente basati sullo stato dell'arte dei linguaggi che permettono operazioni su immagini e sulle operazioni che permettono da semplici rotazioni, zooming e shrinking alle più complicate operazioni di edge detection, cambio dello spazio di colori, etc;
Come linguaggio da adoperare per produrre il progetto sono stati individuati linguaggi dalla sintassi concisa ed efficace quali C, C++, Java e Python. Ognuno di detti linguaggi ha pro e contro ma tutti richiedono l'utilizzo di librerie esterne per l'implementazione delle modifiche e operazioni su immagini. C e Python sono attualmente i più veloci. C presenta una sintassi più complessa per l'utilizzo delle relative librerie mentre Python possiede numerose librerie con una sintassi di facile utilizzo ma alcune di esse, o meglio quasi tutte, non ottengono un buon risultato performando alcune operazioni più complicate come edge detection. Va sottolineato come il package "skimage" per Python sia uno dei migliori in termini di prestazioni e risultati. Per C e C++ la libreria migliore in termini di prestazioni è "LibVips" adoperata anche in questo progetto, nonostante possieda una sintassi di difficile utilizzo per via dei lunghi nomi dei metodi. L'ultima analisi si basa su Matlab, ampiamente usato, specialmente in ambito accademico. Si tratta del linguaggio per immagini per eccellenza. Alcune operazioni, come media adattiva, mediano adattivo, etc, richiedono che il programmatore abbia una conoscenza approfondita del linguaggio ed ambiente di sviluppo MATLAB, mentre per operazioni più semplici, quale applicazione di filtri di media e mediano, è sufficiente utilizzare funzioni built-in. Dallo stato dell'arte vengono dunque individuati i passaggi chiave per la creazione del linguaggio: sintassi semplici e prestazioni d'alto livello.

3 Descrizione del progetto

In questa sezione vengono illustrate le scelte progettuali e le motivazioni che hanno portato alla scelta della libreria LibVips.
IMAGine, volendo emulare Python, ha un funzionamento da interprete interattivo, quindi lanciando l'eseguibile è possibile "programmare live". E' viceversa possibile indicare come secondo argomento un file con estensione `.ig` da mandare all'interprete il quale leggerà e valuterà man mano le espressioni inserite. E' stato creato un helper per aiutare gli utenti nell'utilizzo del linguaggio. Esso è utilizzabile dal terminale, richiamando l'eseguibile inserendo come secondo argomento la stringa `help`, o durante l'esecuzione con attraverso il comando `#help?`.

3.1 Scelte progettuali

Dal punto di vista sintattico il team si è ispirato a linguaggi dalla sintassi intuitiva quali *Javascript* , *Python* e *C#* . Le variabili vengono dichiarate senza specificare un tipo esplicito. Il valore viene semplicemente assegnato alla variabile in fase di inizializzazione della stessa o successivamente. Le immagini vengono dichiarate fornendo il path, rendendo cosi' più manegevole l'elaborazione delle immagini. Il tipo ricorsivo lista può contenere qualunque elemento, compreso un tipo lista.
Particolare attenzione è stata prestata alla *leggibilità* del codice IMAGine ed alla *semplicità d'uso* nel dominio scelto, ovvero l'elaborazione delle immagini. Il linguaggio si presta all'utilizzo da parte di utenti occasionali e presenta una veloce curva di apprendimento.

3.2 LibVips

l team in fase di analisi ricercato al fine di individuare una libreria che fosse veloce e che allo stesso tempo non usasse una alta quantità di memoria per le operazioni effettuate sulle immagini. A tal proposito il team ha valutato i risultati dei benchmarks sulle librerie. Da tale analisi è apparso chiaro che LibVips si classifica per 0.04 secondi come seconda libreria più veloce utilizzando in media ben 100 mb di memoria in meno rispetto alla prima classificata per la stessa operazione. In figura 2 viene riportata una tabella contenente delle informazioni sui benchmarks risultato dell'operazione di copia di un' immagine di 5000x5000 pixel 8-bit RGB in formato TIFF non compressa.
Si può facilmente notare che LibVips lanciato in modalità one thread o in modalità command-line rimane comunque una delle librerie più affidabili. Inoltre nella lunga lista di alternative non sono presenti esclusivamente librerie di image processing per C/C++ ma anche librerie per Python come la famosa SciPy basata su NumPy. In figura 1 vengono riportati i relativi dati.
Analizzando meglio le varie implementazioni ci si è resi conto che VipLips in modalità command-line genera un grosso traffico di dati sul disco. In figura 3 è possibile notare come LibVips (ultimo nella legenda) sia quello che utilizza meno memoria e meno tempo e quindi la miglior libreria utilizzabile.
Un'altra serie di benchmarks, illustrati in figura 4 è stata effettuata prendendo un'immagine di 10000x10000 pixel scattata da una macchina digitale sulla quale sono successivamente state applicate le seguenti operazioni : sottoposta a cambio di spazio di colori, ridimensionata, tagliata (cropping) e sottoposta ad operazione di sharpening. Tale operazione è stata effettuata su diversi processi, alcuni di livello server ad altri di uso domestico.
Si può notare che perfino su dispositivi completamente diversi come Raspberry Pi 2 il tempo per l'operazione non sia altissimo, considerando che l'utilizzo di questa libreria rimane una delle poche opzioni a disposizione dell'utente in quanto sicuramente le suite di software con interfaccia grafica, ad esempio Adobe Creative, non possono essere utilizzate agevolmente su alcuni degli ultimi processori/dispositivi in figura 4 .

scipy 1.2, numpy 1.15.4, pillow	4.33	361	29

Figura 1: Risultato benchmark SciPy

Software	Run time (secs real)	Memory (peak RSS MB)	Times slower
tiffcp -s	0.11	148	
libvips C 8.8	0.15	40	1.0
lua-vips 1.1-9	0.15	49	1.0
pyvips 2.1.6	0.18	49	1.2
ruby-vips 2.0.14	0.21	49	1.4
libvips C 8.8, JPEG images	0.26	48	1.7
libvips command-line 8.8	0.33	38	2.2
Pillow-SIMD 5.3 *see 1*	0.36	230	2.4
ymagine git master 15/12/15 *see 2*	1.06	2.9	3.2 *compared to vips-c JPEG*
libvips C 8.8, one thread	0.52	33	3.5
libvips nip2 8.8	0.57	77	3.8
NetPBM 10.0-15.3	0.60	75	4
sips 10.4.4 *see 6*	0.7 *est.*	268	4.1
GraphicsMagick 1.3.28	0.64	493	4.3
ImageMagick 6.9.7-4	0.82	463	5.5
RMagick 2.16.0 *ImageMagick 6.9.7-4*	0.83	720	5.5
libgd 2.2.5 *see 2*	1.46	193	5.6 *compared to vips-c JPEG*
OpenCV 3.2	0.93	222	6.2
Imlib2 1.4.10 *see 9*	1.14	242	7.6
ExactImage 1.0.1 *see 3*	1.15	122	7.7
FreeImage 3.17 *see 7*	1.28	185	8.5

Figura 2: Risultato benchmark

Figura 3: Risultato benchmark uso memoria

Processor	Clock (GHz)	Cores	Real Time (s)	Speedup
Ryzen Threadripper 3970x	3.7	32 (64ht)	0.2	10 x
E5-2630V3 dual	2.4	2x8 (32ht)	0.44	8.4 x
Ryzen 7 3700X	3.6	8 (16ht)	0.44	5.6
AMD EPYC 7401P	2	24	0.48	10.2 x
I9-9900K	5	8 (16ht)	0.55	4.4 x
E5-2695V3 dual	2.3	28 (56 ht)	0.56	10.4 x
I7-6700K	4.4	4 (8 ht)	0.7	4 x
I7-3930K	3.2	6 (12 ht)	0.79	5.9 x
dual E5649 6core	2.5	12 (24 ht)	0.80	10.8 x
E5-1650	3.2	6 (12 ht)	0.87	5.94 x
I7-6700	3.2	4 (8 ht)	0.89	3.3 x
I7-8550U (XPS laptop)	1.8	4 (8 ht)	0.91	3.2 x
Xeon X5560	2.8	8 (16 ht)	1.08	13 x
Itanium2	?	64	1.1 (est.)	39.4 x
Intel I7 MacBook Pro	2.6	4 (8HT)	1.31	3.7 x
I5-3470S (IMac 27")	2.9	4	1.47	3.1 x
Xeon E5402 (64 bit)	2.0	8	1.88	7.3 x
Opteron 8220 (64 bit)	3.0	8	1.96	7.6 x
Phenom II X6	3.2	6	2.39	4 x
I7-3540m laptop (64b)	3.0	2 (4ht)	2.58	2.2 x
Dual quad-core Intel (64 bit)	3.0	8	2.8	7 x
I5-3210M	2.5	2 (4 ht)	3.51	1.8 x
Core 2 Extreme Quad (32 bit)	2.66	4	3.69	3.8 x
I5-5200U	2.20	2 (4 ht)	3.75	1.9 x
Opteron 850 (HP server)	2.4	4	4.25	3.7 x
Core 2 Duo (MacBook)	2.26	2	5.81	1.9 x
Opteron 254 (HP workstation)	2.7	2	6.14	1.9 x
P4 Xeon (64 bit)	3.6	2 (4 ht)	7	2.4 x
Core Duo (IMac)	2.0	2	11.5	1.85 x
ARM A15 Exynos 5 Chromebook	1.7	2	12.6	1.7 x
P4 Xeon (32 bit)	3.0	2 (4 ht)	19.7	1.6 x
ARM Exynos 5420 (-O0)	1.8-9 or 1.3	8 (4 big + 4 little)	21.31	2.2x
ARM A7 quad core Raspberry PI 2 B (32 bit)	1	4	21.6	3.9 x
PM (HP laptop)	1.8	1	31.8	--
P4 (Dell desktop)	2.4	1	36.6	--
EeePC atom/ssd	1.6	1 (2 ht)	41.5	1.6 x

Figura 4: Risultato benchmark

3.3 Guida all'uso

Per poter utilizzare il linguaggio in esame è necessario installare LibVips [1], a tal proposito è consigliato scaricare il .tar dal seguente link : `https://github.com/libvips/libvips/releases`
Prima di installare la libreria è necessario installare le seguenti dependencies:
`build-essential, pkg-config, glib2.0-dev, libexpat1-dev, expat`
Per ognuna di esse scrivere su terminale Unix "`sudo apt-get install XXX`". Dopo aver fatto ciò si può installare Libvips tramite:

```
tar xf vips-x.y.z.tar.gz
cd vips-x.y.z
./configure
```

Sul terminale, dopo l'ultimo comando verranno specificate eventuali dependencies da installare. E' consigliato l'utilizzo del comando `sudo apt-get update`. Per concludere, quando il comando `./configure` non crea problemi:

```
make
sudo make install
sudo ldconfig
```

4 Caratteristiche del linguaggio

4.1 Grammatica

4.1.1 Tipi

In seguito ad analisi relative ai possibili usi del linguaggio proposto, sono stati individuati i seguenti tipi primitivi:

- `integer`, utilizzato per rappresentare i numeri interi
- `doublePrecision`, utilizzato per rappresentare i numeri in virgola mobile
- `str`, utilizzato per rappresentare le stringhe di caratteri
- `img`, utilizzato per rappresentare le immagini

E' stato inoltre individuato il tipo composto:

- `list`, tipo ricorsivo

Le variabili di tipo *int*, *doublePrecision* e *str* vengono dichiarate senza specificare un tipo esplicito.

Esempio di dichiarazione ed assegnazione:

stringVariable="this is a string";
intVariable=5;
doubleVariable=5.5;
img imageVariable="/path/img.png";
list li={1,2,"element"};

4.1.2 Operazioni

Il linguaggio consente le seguenti operazioni : somma, sottrazione, moltiplicazione, divisione, and e or logici, e paragoni (==, !=, >, >=, <, <=).

- Il risultato di divisioni, prodotti, somme e differenze fra *integer* e *doublePrecision* è sempre un *doublePrecision*.
- Il risultato di divisioni fra *integer* è un *doublePrecision*
- Il risultato di somme, prodotti e differenze fra tipi uguali mantiene il tipo degli elementi iniziali

String1 == String2 valuta il contenuto delle stringhe
String1 != String2 valuta il contenuto delle stringhe

Le combinazioni di tipi consentite per ogni operatore sono riassunte nella tabella 1. I paragoni e gli operatori logici restituiscono sempre degli int, tutti gli altri operatori restituiscono un valore del tipo della colonna, tranne la moltiplicazione `int * string` equivalente a `string * int`. Le operazioni illustrate non sono ammesse per il tipo immagine e per il tipo lista.

V1\V2	int	double	string
int	+ - * / or and CMP	+ - * / or and CMP	*
double	+ - * / or and CMP	+ - / * CMP	+
string	+ *	+	+ CMP

Tabella 1: Operazioni consentite dal linguaggio, sono da intendersi come V1 <operatore> V2. Con CMP si indicano le operazioni di comparazione

4.1.3 Strutture di controllo

Il linguaggio fornisce la possibilità di usare sia comandi condizionali che comandi iterativi. I comandi condizionali sono i seguenti:

- `if ( condition ) then { instructions };`
- `if ( condition ) then { instructions } else { instructions }`

I comandi iterativi sono qui elencati :

- `while ( condition ) do { instructions }`
- `foreach ( tempElement : list ) { instructions }`

Nell'espressione `while` si itera indefinitivamente finché la condizione non è più verificata. Nle costrutto `foreach` si itera invece per ogni elemento contenuto nella lista. E' possibile effettuare operazioni sull'elemento della lista tempElement, anche nel caso in cui si tratti di un'immagine. Questo rende possibile l'elaborazione di una lista di immagini.

4.1.4 Funzioni

Il linguaggio presenta una ricca selezione di funzioni built-in. All'utente viene inoltre fornita la possibilità di definire la propria funzione custom, con sintassi:

"`def nome_funzione ( parametri_formali ) { istruzioni }`".

4.2 Parser

Il linguaggio è stato progettato con i tools Bison e Flex [2], due strumenti per costruire programmi che gestiscano input strutturati. Flex si occupa dell'analisi lessicale (lexing) mentre Bison si occupa dell'analisi sintattica (parsing). L'output dei due tools è un parser di tipo LALR(1) che effettua un parsing bottom-up, con cui è possibile gestire produzioni left-recursive, utilizzando Bison per generare il parser e Flex per riconoscere i token nella fase di lexing.

Riportiamo di seguito le regole di produzione in formato BNF del linguaggio proposto:

```
<program>::=
| HELP
| <program> <stmt>
| <program> DEF NAME '(' <symlist> ')' '{' <list> '}'
| <program> LIST NAME '='''{' elements '}' ';'
| <program> error '\n'

<stmt>::= IF '(' <exp> ')' THEN '{' <list> '}' ';'
| IF '(' <exp> ')' THEN '{' list '}' ELSE '{' <list> '}'
| WHILE '('<exp> ')' DO '{' <list> '}'
| FOREACH '(' <foreach> ')' '{' <list> '}'
| exp ';'

<list >::=
| <stmt> <list >

<exp>::= <exp> AND <exp>
|<exp> OR <exp>
| <exp> CMP <exp>
| <exp> '+' <exp>
| <exp> '-' <exp>
| <exp> '*' <exp>
| <exp> '/' <exp>
| '|' <exp> '|'
| '(' <exp> ')'
| FUNC '(' <explist> ')'
| NAME '=' <exp>
| IMG NAME '=' <img>
| NAME '(' <explist> ')'
| <value>

<foreach >::=     <name> ':' <name>

<name>::= NAME

<value>::=  '-' INT %prec UMINUS
| INT
| '-' DOUBLE %prec UMINUS
| DOUBLE
| STRING
| NAME

<img>::=  STRING
| FUNC '(' <explist> ')'

<explist >::= <exp>
```

```
| <exp> ',' <explist>

<symlist>::= NAME
| NAME ',' <symlist>

<elements>::=
| <value>
| <value> ',' <elements>
```

Si possono eliminare le ambiguità della grammatica attraverso strumenti che permettono di specificare la precedenza degli operatori eliminando le possibili ambiguità e scartando automaticamente gli alberi di parsing non validi, consentendo l'utilizzo di una grammatica più semplice e compatta. A tal proposito sono utilizzati gli operatori di precedenza

4.3 Lexer

Nel lexer vengono adoperate delle regex per matchare i nomi delle variabili,le stringe,i numeri interi,i numeri in virgola mobile, l'inserimento di commenti, i simboli per i confronti, operazioni consentite sui tipi, i simboli consentiti, le funzioni built-in del linguaggio e le strutture di controllo.

Help method

```
/* shows helper */
"#help?"
```

Valori

```
/* string */
\"{1}.[^"]*\"{1}

/* int */
[0-9]+

/* double */
[0-9]+\.{1}[0-9]+
```

Tipi espliciti

```
"img"
"list"
"def"
```

Nome variabili

```
/* names */
[a-zA-Z][a-zA-Z0-9]*
```

Simboli

```
"+"
"-"
"*"
"/"
"="
"|"
","
";"
"("
"{"
"}"
":"
")"
```

Paragoni

```
">"
"<"
"!="
"=="
">="
"<="
```

Operatori logici

```
"&&"
"||"
```

Strutture di controllo ed iterative

```
 "if"
 "then"
 "else"
 "while"
 "do"
 "foreach"
```

Funzioni built-in

```
"print"
```

Funzioni built-in per elaborazione immagini

```
"width"
"height"
"bands"
"min"
"max"
"average"
"invert"
"histEq"
"normalise"
"canny"
"sobel"
"sharpen"
"convert"
"convertColorSpace"
"add"
"subtract"
"rotate"
"flip"
"gaussianBlur"
"smartCrop"
"extractBand"
"zoom"
"crop"
"showImage"
```

Funzioni built-in per liste

```
"length"
"push"
"insert"
"remove"
"pop"
"get"
```

Commenti

```
"//".*
[ \t]    /* ignore white space */
\\\n     /* ignore line continuation */
"\n"
```

4.4 Funzionalità del linguaggio

4.4.1 Metodo print

Il metodo **print(var)** stampa su terminale il contenuto di **var**. Accetta qualunque tipo come argomento. **In generale**, *print(var + string)* stamperà su schermo la concatenazione delle due variabili, indipendentemente dal tipo di *var* .
E' anche possibile ripetere una stringa **n** volte utilizzando all'interno del metodo l'operatore *****

print(3*string) stamperà per tre volte il contenuto di string
print(3+string) stamperà la concatenazione fra la 3 e il contenuto di string
print(string1+string2) stamperà la concatenazione fra le due stringhe

4.4.2 Metodi per le liste

Vengono qui elencati i metodi per le liste. *Element* può essere un tipo qualunque, perfino una lista, in quanto le liste di liste sono permesse.

- `int length(list list_name)` restituisce la lunghezza della lista ;
- `void push(list list_name, element)` inserisce un elemento nella lista(all'ultimo posto) ;
- `void insert(list list_name, element, int position)` inserisce un elemento nella posizione desiderata ;
- `void remove(list list_name, int position);` rimuove l'elemento dalla posizione desiderata ;
- `element pop(list list_name)` rimuove e restituisce l'ultimo elemento ;
- `element get(list list_name, int position)` restituisce l'elemento nella posizione desiderata ;

4.4.3 Metodi per le immagini

Di seguito vi sono tutte le operazioni delle immagini con una breve descrizione delle loro operazioni.

- `int width(img name_image)` restituisce la larghezza dell'immagine in pixel ;
- `int height(img name_image)` restituisce l'altezza dell'immagine in pixel ;
- `int bands(img name_image)` restituisce il numero di canali presenti nell'immagine ;
- `double min(img name_image)` restituisce il minimo valore dei pixel nell'immagine ;
- `double max(img name_image)` restituisce il massimo valore dei pixel dell'immagine ;
- `double average(img name_image)` restituisce il valore medio dei pixel dell'immagine ;
- `img invert(img name_image, char* output_path)` restituisce e salva l'immagine invertita ;
- `img histEq (img name_image, char* output_path)` restituisce e salva un'immagine sottoposta a equalizzazione dell'istogramma ;
- `img normalise(img name_image, char* output_path)` restituisce e salva un'immagine sottoposta a normalizzazione ;
- `img canny(img name_image, char* output_path)` individua gli edge con il metodo di Canny, salva e restituisce l'immagine ;

- `img sobel(img name_image, char* output_path)` individua gli edge con il metodo di Sobel, salva e restituisce l'immagine ;
- `img sharpen(img name_image, char* output_path)` esegue un blur gaussiano e lo sottrae all'immagine originale per generare un segnale d'alta frequenza.
  Ritorna e salva l'immagine risultante ;
- `img convert(img name_image, char* output_path)` ritorna e salva l'immagine con un output specifico* ;
- `img convertColorSpace(img name_image, char* output_path, char* space)` sposta l'immagine nello spazio dei colori indicato, salva e restituisce l'immagine** ;
- `img add (img name_image, img name_image2, char* output_path)` restituisce e salva la somma bit a bit di due immagini ;
- `img subtract (img name_image, img name_image2, char* output_path)` restituisce e salva la differenza bit a bit di due immagini ;
- `img rotate(img name_image, char* output_path, double rotate_angle)` restituisce e salva l'immagine di input rotata di un certo angolo ;
- `img flip(img name_image, char* output_path, char* direction)` restituisce e salva l'immagine di input capovolta in una certa direzione (verticale/orizzontale) ;
- `img gaussianBlur(img name_image, char* output_path, double mask_dim)` restituisce e salva l'immagine di input sottoposta a blur gaussiano con maschera di dimensioni specificate in input ;
- `img extractBand(strimg name_image, str output_path, integer band_n);` estrae la band_n banda dall'immagine di input e la salva in output_path ;
- `img smartCrop (img name_image, char* output_path, double width, double height)` ritaglia un'immagine,la salva e la restituisce. La selezione delle parti da ritagliare avviene con un algoritmo che cerca di individuare le caratteristiche di maggiore interesse per gli esseri umani ;
- `img zoom(img name_image, char* output_path, int xfactor, int yfactor)` restituisce e salva l'immagine ingrandita in base a due fattori di zoom x e y ;
- `img crop(img name_image, char* output_path, double left, double top, double width, double height)` ritaglia un'immagine,la salva e la restituisce. Le dimensioni della porzione da ritagliare sono fornite in input (rispettivamente distanza in pixel dal margine sinistro,distanza in pixel dal margine in alto, larghezza ed altezza);
- `void showImage(img name_image);` mostra l'immagine fornita su schermo ;

*Formati ammessi:

- tif
- jpeg or jpg
- hdr
- raw

**Spazi dei colori ammessi:

- grey16
- hsv
- yxy

- lch
- cmc
- lab

5 Casi d'uso

Vengono qui mostrati dei possibili casi d'uso del linguaggio. L'obbiettivo è di mostrare le funzionalità e la semplicità d'uso di *IMAGine*.
Le immagini in 5 sono utilizzate per effettuare le elaborazioni presentate nel tre casi d'uso.

Figura 5: Immagini di input

5.1 Primo caso

Dall'esempio in esame si evince la semplicità ed efficacia del linguaggio. Viene inizialmente definito un metodo per trovare le dimensioni da ritagliare dell'immagine. Viene successivamente convertita l'immagine da formato `tif` a `jpg`. Dopo ciò viene generata iterativamente una lista che contiene i path di uscita che verranno successivamente utilizzati. In seguito si procede all'estrazione di una banda dalle due immagini a colori (3 canali). Utilizzando il costrutto `foreach` si applicano gli operatori di *Sobel* e *Canny* ad entrambe le immagini ad una banda. Successivamente si esegue uno `smartcrop`, che ritaglia le immagini con le dimensioni appropriate(fornite attraverso l'appropriata funzione definita dall'utente `getDim()`). Il risultato viene mostrato a schermo attraverso il metodo `showImage()`. Dalle immagini ottenute si noti come gli edge ottenuti con il filtro di *Sobel* forniscano ottimi risultati.

```
def getDim(a){
        // Finds the dimensions of the area to crop
        wi=width(a);
        hi=height(a);
        if(wi>200 && hi>200)then{
                100;
        }else{
                wi*hi;
        }
}
def pageLine(a){
        print("---------"*a);
}

img lena="../program/Images/lena.tif";
img pencils="../program/Images/pencils.jpg";

convert(lena,"../program/Images/lena.jpg");
pageLine(1);

list outLi={};
i=1;
while(i<9)do{
        push(outLi,"../program/Images/Case1/outImg"+i+".jpg");
        i=i+1;
}

list imgLi={lena,pencils};
push(imgLi, extractBand(get(imgLi, 1), "../program/Images/Case1/band"+1+".jpg
    ↪ ", 0));
remove(imgLi, 1);
push(imgLi, extractBand(get(imgLi, 1), "../program/Images/Case1/band"+2+".jpg
    ↪ ", 0));
remove(imgLi, 1);
pageLine(3);

list imgOut={};
counterLi=1;
counter=1;
foreach(tempLi:imgLi){
                push(imgOut, sobel(tempLi,get(outLi,counterLi)));
                push(imgOut, canny(tempLi,get(outLi,counterLi+1)));
```

```
    smartCrop(get(imgOut, counter), get(outLi,counterLi+2), getDim(tempLi),
        ↪ getDim(tempLi));
    smartCrop(get(imgOut, counter+1), get(outLi,counterLi+3), getDim(tempLi),
        ↪ getDim(tempLi));
                counter=counter+1;
                counterLi=counterLi+4;
}

foreach(elaborate:imgOut){
                showImage(elaborate);
}
```

Listing 1: Sorgente firstCase.ig

Figura 6: Immagini di output, i risultati di Canny con queste dimensioni sono difficilmente apprezzabili

5.2 Secondo caso

Viene inizialmente definita una funzione per ottenere le dimensioni appropriate della maschera utilizzata per effettuare un `blur gaussiano`.Vengono successivamente caricate due immagini in una lista,sulle quali viene effettuato il `blur gaussiano` e vengono mostrate a video. Si possono evincere dalle immagini le differenze fra maschere di dimensioni diverse.

```
def getBlur(a){
        blur=1;
        if(width(a)>100 && height(a)>288)then{
                blur=3;
                print("Blur di dimensioni: " + blur);
                blur;
        }else{
                print("Blur di dimensioni: " + blur);
                blur;
        }
}
def pageLine(a){
        print("---------"*a);
}

img lena="../program/Images/lena.tif";
img pencils="../program/Images/pencils.jpg";

out1="../program/Images/Case2/blurredLena.tif";
out2="../program/Images/Case2/blurredPencils.jpg";

list imgLi={lena,pencils};
list outLi={out1,out2};

index=1;
while(index<=length(imgLi))do{
        img tempIm=get(imgLi,index);
        showImage(gaussianBlur(tempIm,get(outLi,index),getBlur(tempIm)));
        pageLine(6);
        index=index+1;
}
```

Listing 2: Sorgente secondCase.ig

Figura 7: Immagini di output secondo caso

5.3 Terzo caso

Il terzo caso mostra tutte le operazioni che si possono effettuare sulle immagini. Vengono inizialmente mostrate tutte le caratteristiche di un'immagine su schermo. Successivamente viene creata iterativamente, grazie alle comode caratteristiche che permettono di concatenare con facilità stringhe e variabili, la lista di path di uscita delle immagini. Vengono infine effettuate le operazioni sulle immagini.

```
def pageLine(a){
        print("---------"*a);
}
img im="../program/Images/pencils.jpg";

wi=width(im);
he=height(im);
ba=bands(im);
mi=min(im);
ma=max(im);
avg=average(im);
print("Image 'im' details:");
print("Width: " + wi + " Height: " + he + " Bands: " + ba + " Min: " + mi + "
    ↪ Max: " + ma + " Avg: " + avg);
pageLine(2);
list outNames=
{"inverted","hsv","equalized","normalised",
"canny","sobel","sharpen","converted","added",
"subtracted","rotated","flipped","gaussianBlu",
"smartCrop","zoomed","cropped"};
i=1;
while(i<=length(outNames))do{
        insert(outNames, "../program/Images/Case3/"+get(outNames, i)+".jpg", i
            ↪ +1);
        remove(outNames, i);
        i=i+1;
}
pageLine(3);
img inverted=invert(im,get(outNames,1));
img differentColor=convertColorSpace(im,get(outNames,2),"hsv");
img equalized=histEq(im,get(outNames,3));
img normalised=normalise(im,get(outNames,4));
img cannyImg=canny(im,get(outNames,5));
img sobelImg=sobel(im,get(outNames,6));
img sharpened=sharpen(im,get(outNames,7));
img converted=convert(im,get(outNames,8));
img added=add(im,equalized,get(outNames,9));
img subtracted=subtract(im,sobelImg,get(outNames,10));
img rotated=rotate(im,get(outNames,11),45.7);
img flipped=flip(im,get(outNames,12),"hor");
img gaussianBlurred=gaussianBlur(im,get(outNames,13),1);
img smartCropped=smartCrop(im,get(outNames,14),200,200);
img zommed=zoom(im,get(outNames,15),5,5);
img cropped=crop(im,get(outNames,16),50,50,50,50);
```

Listing 3: Sorgente thirdCase.ig

Figura 8: Immagini di output terzo caso

6 Risultati

Al fine di testare la performance di IMAGine sono stati effettuati dei test su alcuni fra i più noti problemi calcolo quali il fattoriale e il pi greco. Infine viene effettuato un paragone con MATLAB al fine di testare operazioni nel dominio applicativo del linguaggio, ovvero l'elaborazione delle immagini.

6.1 Fattoriale

```
n=150;
f=1.0;
i=1;

while(i<=n)do{
   f=f*i;
   i=i+1;
}
```

Listing 4: Sorgente factorial.ig

```
n=100
f=1

for i in range(1,n+1):
    f*=i
```

Listing 5: Sorgente factorial.py

Fattorial Calc N=150		
Attempt	**IMAGine**	**Python**
1	0,005	0,009
2	0,005	0,007
3	0,005	0,008
4	0,007	0,009
5	0,006	0,007
6	0,006	0,008
7	0,006	0,007
8	0,006	0,008
9	0,006	0,007
10	0,006	0,009
AVG	**0,0058**	**0,0079**

Figura 9: Risultato benchmark

Figura 10: Risultato benchmark

6.2 Pi greco

```
pi=1.0;
n=1;
while (n < 100) do{
    app=(2*n-1)*(2*n+1);
    app2=10*pi*(2*n)*(2*n);
    pi = app2/app;
    n = n+1;
}
```

Listing 6: Sorgente pigreco.ig

```
# -*- coding: utf-8 -*-
pi=1;
n = 1;
while n < 100:
    pi = 10*pi*(2*n)*(2*n)/((2*n-1)*(2*n+1));
    n = n+1;
```

Listing 7: Sorgente pigreco.py

Pi Calc		
Attempt	***IMAGine***	***Python***
1	0,006	0,011
2	0,006	0,009
3	0,006	0,008
4	0,006	0,008
5	0,005	0,007
6	0,006	0,008
7	0,007	0,008
8	0,006	0,008
9	0,007	0,008
10	0,008	0,007
AVG	**0,0063**	**0,0082**

Figura 11: Risultato benchmark

Figura 12: Risultato benchmark

6.3 Restauro

```
img im="/home/kali/Desktop/Ingegneria/Linguaggi/IMAGine/test/Benchmark/
    ↪ GaussianNoise.jpg";
out="/home/kali/Desktop/Ingegneria/Linguaggi/IMAGine/test/Benchmark/
    ↪ GaussianRestored.jpg";

img restored=gaussianBlur(im, out, 5);
```

Listing 8: Sorgente gaussianrestore.ig

```

tic
    im=im2double(imread("GaussianNoise.jpg"));
    %dimensione del filtro
    filter_size=5;
    %media
    filtro_media=fspecial('average', filter_size);
    im_avg=imfilter(im, filtro_media, 'conv', 'circular');
    imwrite(im_avg,"GaussianRestoredMat.jpg");
time10 = toc;
```

Listing 9: Sorgente gaussianrestore.m

Gaussian Restore Mask=5		
Attempt	***IMAGine***	***Matlab***
1	0,0400	0,0721
2	0,0380	0,0392
3	0,0350	0,0479
4	0,0470	0,0443
5	0,0310	0,0402
6	0,0400	0,0599
7	0,0310	0,0380
8	0,0420	0,0378
9	0,0410	0,0381
10	0,0380	0,0369
AVG	**0,0383**	**0,0454**

Figura 13: Risultato benchmark

Figura 14: Risultato benchmark

7 Conclusioni

Durante lo sviluppo del linguaggio sono state affrontate numerose tematiche trattate durante il corso di *Linguaggi e traduttori* . L'esperienza di sviluppo ha permesso al team di apprezzare l'importanza di solide fondamenta teoriche unite a una significativa esperienza pratica. Nonostante le tempistiche, le funzionalità ottenute risultano soddisfacenti . Le caratteristiche del linguaggio lo rendono fruibile come general-purpose senza compromettere la sua utilità nel dominio applicativo scelto, ovvero l'elaborazione delle immagini.

Riferimenti bibliografici

[1] Libvips API documentation `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm`

[2] John Levine. *Flex & Bison: Text Processing Tools.* " O'Reilly Media, Inc.", 2009

Appendice

Grafico del FSA generato utilizzando GNU Bison 3.5.4 XML Automaton Report. L'immagine è in formato vettoriale, quindi è possibile ingrandire a piacere.