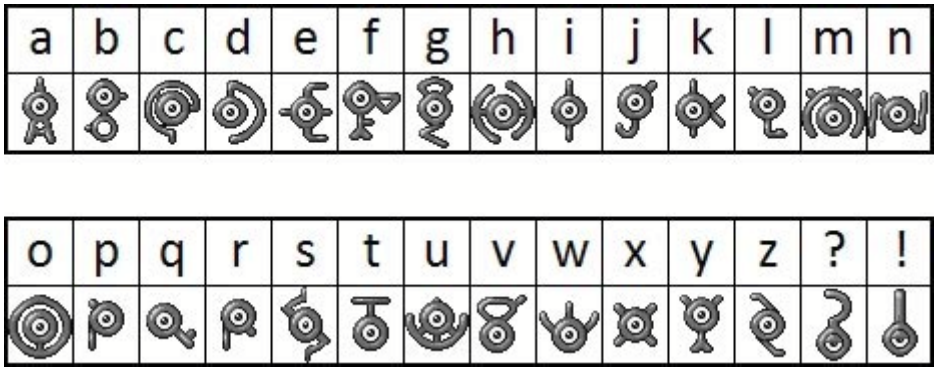
Projet de semestre
Traitement d'image

Github : https://github.com/LoisonYo/ZarbiDecriptor
Nom du projet : Zarbi Decryptor
Classe : INF3dlm-b
Étudiants :

- Ugo Crucy
- Yohann Loison

1. Introduction

Dans le monde de Pokémon, les zarbis sont des créatures qui ont l'apparence des lettres de l'alphabet. Comme vous pouvez le voir dans la *figure 1*, il existe, pour chaque lettre de l'alphabet, une forme de Zarbi équivalente.



(*Figure 1*) Comparaison alphabet - zarbis

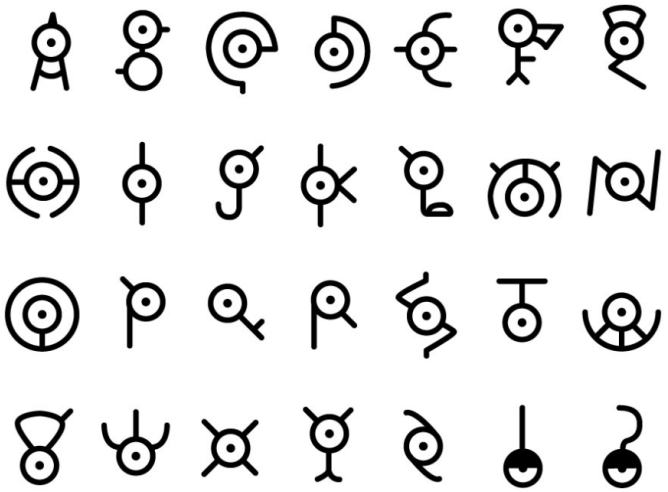
Pour notre projet de traitement d'image, nous sommes partis sur un traducteur de Zarbi. En effet, l'utilisateur a la possibilité d'importer une image et le traducteur va traiter l'image puis afficher le texte équivalent.

2. Approche

2.1. Choix de l'image

Avant de commencer quoi que ce soit, il est impératif de trouver un alphabet Zarbi sans trop d'effet spéciaux (ombre, reflet de lumière, etc...). Ainsi, il est plus simple de faire du traitement d'image sur une image "propre" que sur un alphabet avec des effets.

Après quelques recherches, nous avons trouvé un alphabet Zarbi adapté pour notre traitement d'image (voire *Figure 2*). En effet, les Zarbis n'ont pas d'ombre, pas de reflet de lumière et la couleur est la même pour tous.



(*Figure 2*) Zarbis utilisés pour le traitement d'image

2.2. Détection des lettres

Pour la détection des Zarbis, nous sommes partis sur la fusion de deux méthodes : les rectangles englobants et le template matching. Les rectangles englobant vont permettre de localiser chaque lettre de l'alphabet sur l'image. Ensuite, il suffit de faire du template matching sur chaque rectangle pour trouver les lettres correspondantes.

Cette méthode marche très bien sur une image sans bruit. En revanche, si après traitement, il y a toujours du bruit sur l'image, ce dernier sera considéré comme une lettre par le programme et le résultat se retrouvera biaisé. Il est donc nécessaire de correctement traiter l'image avant de faire du template matching.

2.3. Image redimensionnée

Lorsqu'on prend du texte en photo, il se peut que la taille des lettres se retrouve modifiée. Heureusement pour nous, ce problème se résout facilement.

En effet, lors du template matching, il nous suffit de redimensionner chaque template pour qu'il soit de la même taille que le rectangle englobant.

3. Développement

3.1. Déroulement

Le traitement effectué par notre programme peut être divisé en 5 parties distinctes.

3.1.1. Gaussian Blur

Le bruit gaussien va permettre "d'adoucir" l'image et ainsi éliminé la majorité des très petits bruits.

3.1.2. Seuillage

La deuxième étapes consiste à seuiller l'image pour avoir une image en noir et blanc en sortie. Pour le seuillage, nous avons appliqué le seuillage binaire avec Otsu à l'aide de OpenCV.

3.1.3. Erosion et dilatation

La troisième étape consiste à supprimer les petites zones de bruit. Pour ce faire, nous avons appliqué une érosion puis une dilatation avec une matrice de 5x5. Comme expliqué précédemment, cela va éliminer le petite zone de bruit sans modifier les lettres Zarbis.

3.1.4. Rectangle englobant

Les rectangles englobants vont permettre de situer les Zarbis sur l'image pour ensuite appliquer le template matching.

3.1.5. Template matching

Pour chaque rectangle englobant trouvé, le programme va superposer tous les templates (Zarbis) et prendre celui qui correspond le mieux. On arrive ainsi à reconstituer le texte écrit en Zarbis.

3.2. Rectangles englobants

Comme expliqué plus haut, la détection des rectangles englobants de OpenCV va nous permettre d'isoler les Zarbis de l'image.

Cette méthode, sur une image sans bruit, fonctionne parfaitement. En revanche, sur une image bruitée, chaque pixel noir qui n'ont pas disparu lors du prétraitement vont être considéré comme une une lettre.

Pour régler ce problème nous avons mis une aire minimum pour les rectangles de 100 pixels. Ainsi, les petits bruits ne sont pas considérés comme une lettre. De plus, pour régler les problèmes des gros bruits (ombres, etc...), nous avons utilisé la moyenne de l'aire des rectangles pour prendre les rectangles qui sont proches de la moyenne. Cette méthode fonctionne sur la majorité de nos images.

3.3. Template Matching

Comme expliqué précédemment, notre application utilise la technique du template matching pour convertir les zarbis en lettres de l'alphabet. Afin de tester le fonctionnement de cette méthode, nous avons appliqué le template matching sur chaque zarbis pour comparer les résultats :

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	1	0,2	0,016	0,086	0,181	0,304	0,192	0,083	0,187	0,316	0,187	0,443	0,211	0,309	0,032	0,45	0,095	0,215	0,365	0,295	0,316	0,307	0,168	0,287	0,454	0,175
b	0,2	1	0,201	0,193	0,098	0,128	0,26	0,171	0,152	0,175	0,206	0,114	0,061	0,036	0,179	0,196	0,162	0,366	0,168	0,147	0,11	0,41	0,09	0,193	0,134	0,093
c	0,016	0,201	1	0,521	0,393	0,198	0,467	0,695	0,251	0,267	0,247	0,118	0,212	0,202	0,757	0,302	0,19	0,254	-0,002	0,188	0,078	0,098	0,209	0,235	0,2	0,35
d	0,086	0,193	0,521	1	0,294	0,26	0,22	0,447	0,238	0,225	0,174	0,058	0,3	0,29	0,53	0,286	0,173	0,291	0,258	0,289	0,316	0,066	0,272	0,101	0,154	0,239
e	0,181	0,098	0,393	0,294	1	0,062	0,29	0,462	0,454	0,194	0,403	0,176	0,149	0,025	0,376	0,196	0,124	0,199	0,171	0,194	0,307	0,065	0,223	0,175	0,226	0,665
f	0,304	0,128	0,198	0,26	0,062	1	0,352	0,075	0,134	0,18	0,109	0,137	0,219	0,212	0,145	0,332	0,306	0,337	0,203	0,279	0,124	0,127	0,081	0,217	0,209	0,181
g	0,192	0,26	0,467	0,22	0,29	0,352	1	0,262	0,074	0,082	0,101	0,133	0,261	0,177	0,48	0,127	0,149	0,219	0,089	0,151	0,157	0,046	0,131	0,443	0,247	0,312
h	0,083	0,171	0,695	0,447	0,462	0,075	0,262	1	0,314	0,239	0,246	0,182	0,384	0,203	0,748	0,24	0,22	0,2	0,08	0,186	0,172	0,114	0,18	0,169	0,23	0,436
i	0,187	0,152	0,251	0,238	0,454	0,134	0,074	0,314	1	0,0146	0,941	0,19	0,614	0,244	0,236	0,158	0,163	0,205	0,514	0,363	0,275	0,108	0,201	0,188	0,151	0,586
j	0,316	0,175	0,267	0,225	0,194	0,18	0,082	0,239	0,0146	1	0,146	0,402	0,221	0,075	0,174	0,591	0,056	0,396	0,129	0,154	0,555	0,365	0,342	0,098	0,652	0,121
k	0,187	0,206	0,247	0,174	0,403	0,109	0,101	0,246	0,941	0,146	1	0,095	0,502	0,165	0,207	0,221	0,185	0,1	0,388	0,081	0,264	0,132	0,228	0,2	0,181	0,477
l	0,443	0,114	0,118	0,058	0,176	0,137	0,133	0,182	0,19	0,402	0,095	1	0,223	0,24	0,112	0,474	0,043	0,105	0,295	0,126	0,304	0,296	0,171	0,205	0,688	0,227
m	0,211	0,061	0,212	0,3	0,149	0,219	0,261	0,384	0,614	0,221	0,502	0,223	1	0,242	0,242	0,254	0,227	0,248	0,445	0,375	0,313	0,267	0,151	0,146	0,259	0,342
n	0,309	0,036	0,202	0,29	0,025	0,212	0,177	0,203	0,244	0,075	0,165	0,24	0,242	1	0,168	0,229	0,269	0,328	0,107	0,134	0,196	0,109	0,123	0,434	0,277	0,244
o	0,032	0,179	0,797	0,53	0,376	0,145	0,48	0,748	0,236	0,174	0,207	0,112	0,242	0,168	1	0,261	0,165	0,17	0,053	0,164	0,118	0,064	0,147	0,216	0,107	0,421
p	0,45	0,196	0,302	0,286	0,196	0,332	0,127	0,24	0,158	0,591	0,221	0,474	0,254	0,229	0,261	1	0,179	0,273	0,22	0,209	0,517	0,396	0,395	0,195	0,566	0,196
q	0,095	0,162	0,19	0,173	0,124	0,306	0,149	0,22	0,163	0,056	0,185	0,043	0,227	0,269	0,165	0,179	1	0,229	0,147	0,251	0,373	0,052	0,284	0,245	0,113	0,193
r	0,215	0,366	0,254	0,291	0,199	0,337	0,219	0,2	0,205	0,396	0,1	0,105	0,248	0,328	0,17	0,273	0,229	1	0,159	0,276	0,455	0,22	0,215	0,202	0,15	0,227
s	0,365	0,168	-0,002	0,258	0,171	0,203	0,089	0,08	0,514	0,129	0,388	0,295	0,445	0,107	0,053	0,22	0,147	0,159	1	0,391	0,264	0,245	0,095	0,134	0,205	0,276
t	0,295	0,147	0,188	0,289	0,194	0,279	0,151	0,186	0,363	0,154	0,081	0,126	0,375	0,134	0,164	0,209	0,251	0,276	0,391	1	0,278	0,11	0,239	0,116	0,178	0,236
u	0,316	0,11	0,078	0,316	0,307	0,124	0,157	0,172	0,275	0,555	0,264	0,304	0,313	0,196	0,118	0,517	0,373	0,455	0,264	0,278	1	0,229	0,22	0,161	0,359	0,408
v	0,307	0,41	0,098	0,066	0,065	0,127	0,046	0,114	0,108	0,365	0,132	0,296	0,267	0,109	0,064	0,396	0,052	0,22	0,245	0,11	0,229	1	0,103	0,178	0,312	0,096
w	0,168	0,09	0,209	0,272	0,223	0,081	0,131	0,18	0,201	0,342	0,228	0,171	0,151	0,123	0,147	0,395	0,284	0,215	0,095	0,239	0,22	0,103	1	0,115	0,321	0,276
x	0,287	0,193	0,235	0,101	0,175	0,217	0,443	0,169	0,188	0,098	0,2	0,205	0,146	0,434	0,216	0,195	0,245	0,202	0,134	0,116	0,161	0,178	0,115	1	0,221	0,155
y	0,454	0,134	0,2	0,154	0,226	0,209	0,247	0,23	0,151	0,652	0,181	0,688	0,259	0,277	0,107	0,566	0,113	0,15	0,205	0,178	0,359	0,312	0,321	0,221	1	0,227
z	0,175	0,093	0,35	0,239	0,665	0,181	0,312	0,436	0,586	0,121	0,477	0,227	0,342	0,244	0,421	0,196	0,193	0,227	0,276	0,236	0,408	0,096	0,276	0,155	0,227	1

(Figure 3) Correspondance entre les lettres : Disponible dans ./others/ratio.xlsx

On peut voir que le template matching est une technique assez fiable. Il y a tout de même deux lettres qui peuvent poser quelques problèmes. En effet, la lettre "C" et "O" ont une correspondance de 94,1%, ce qui est extrêmement proche. Si les images à traiter ont beaucoup de bruit, le programme risque de confondre ces deux lettres.

4. Résultats

4.1. Images non-bruitée

Pour les images non bruitées (prises directement dans le programme), le résultat trouvé par le programme est 99.9% du temps correct.

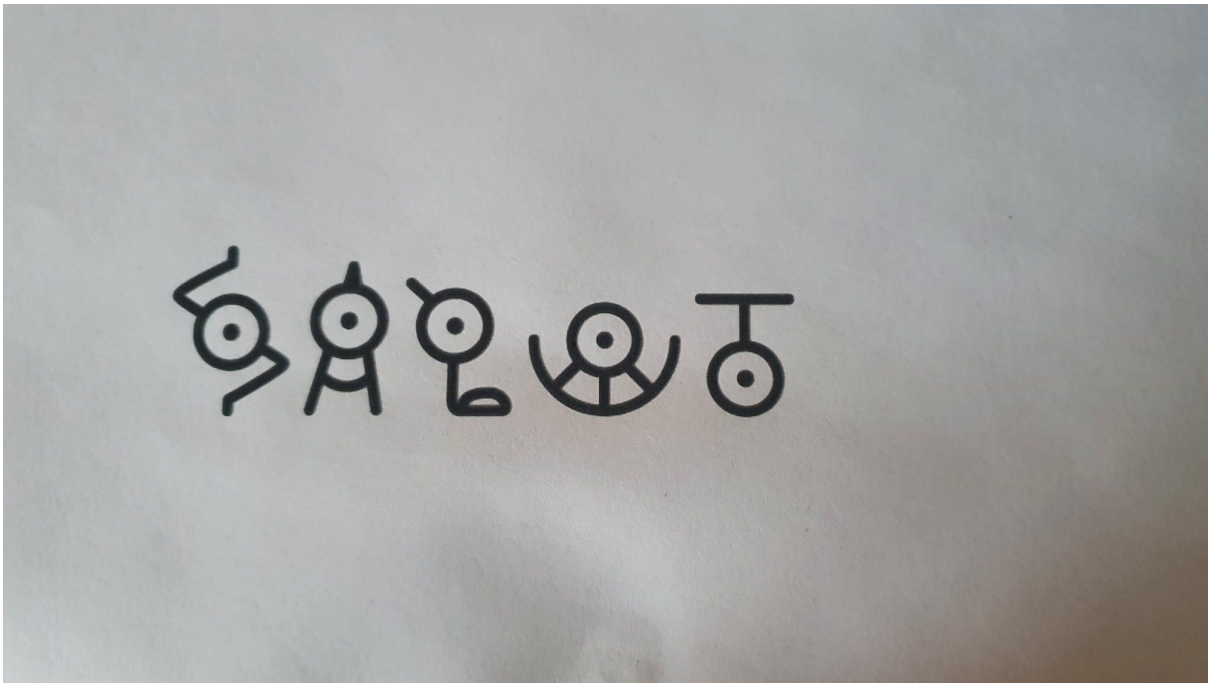
(Figure 4) Image non bruitée après rectangles englobant

Le résultat de l'image en *Figure 4* est "salut". Le programme a donc correctement traduit le message.

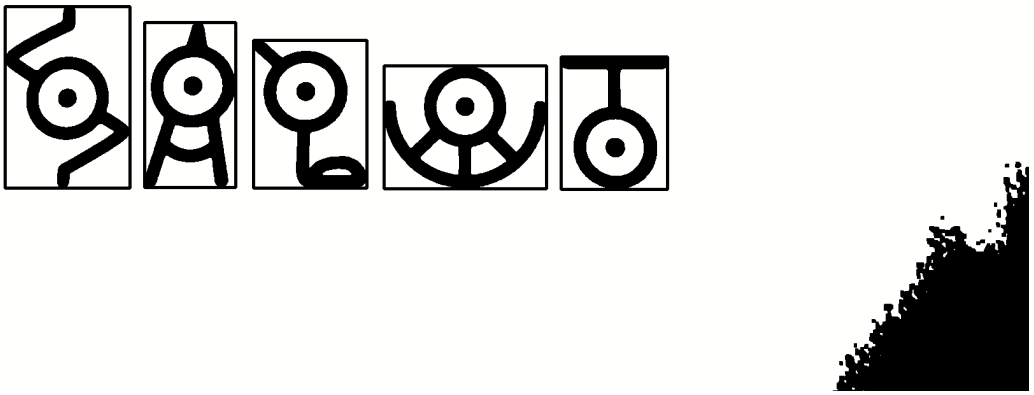
4.2. Image bruitée fonctionnelle

La partie intéressante de ce projet est de pouvoir tester avec des images prises directement par un appareil photo (Samsung Galaxy S10+). La *Figure 5* est une photo d'une page A4 sur

laquelle est imprimé le mot “salut” en zarbi. La particularité de cette image est l'ombre dans le coin en bas à droite ainsi que le pli de la page A4 en bas de l'image.



(Figure 5) Texte zarbi imprimé sur une feuille A4

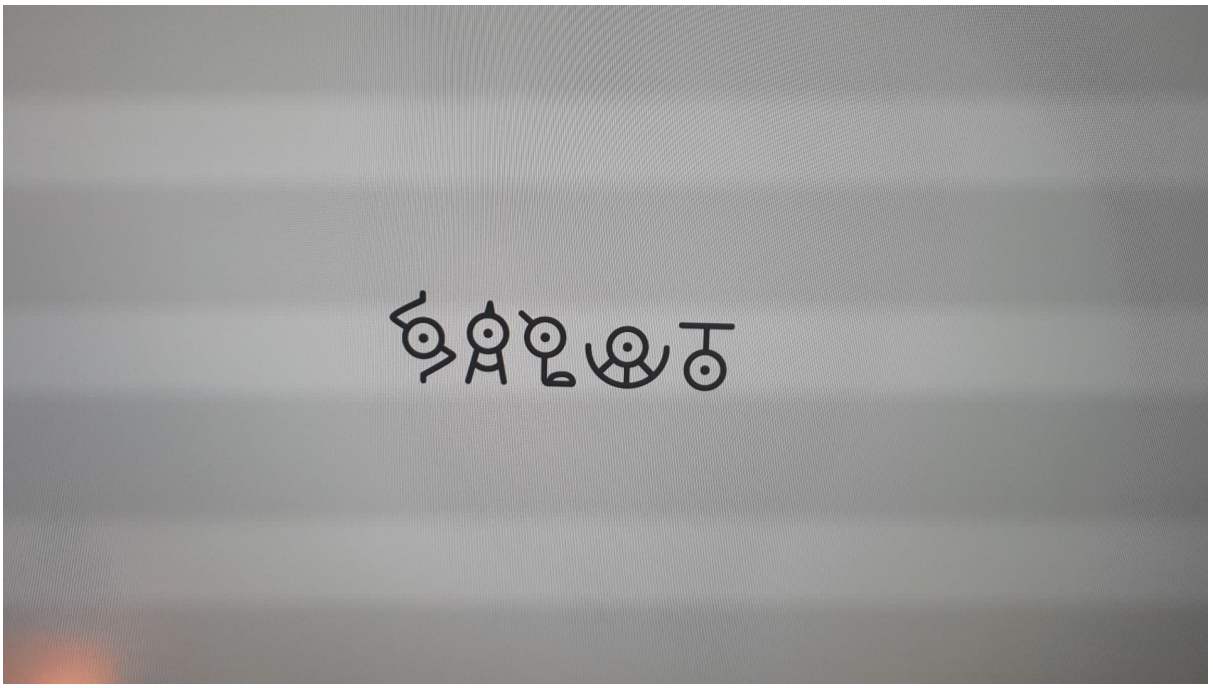


(Figure 6) Image bruitée après traitement

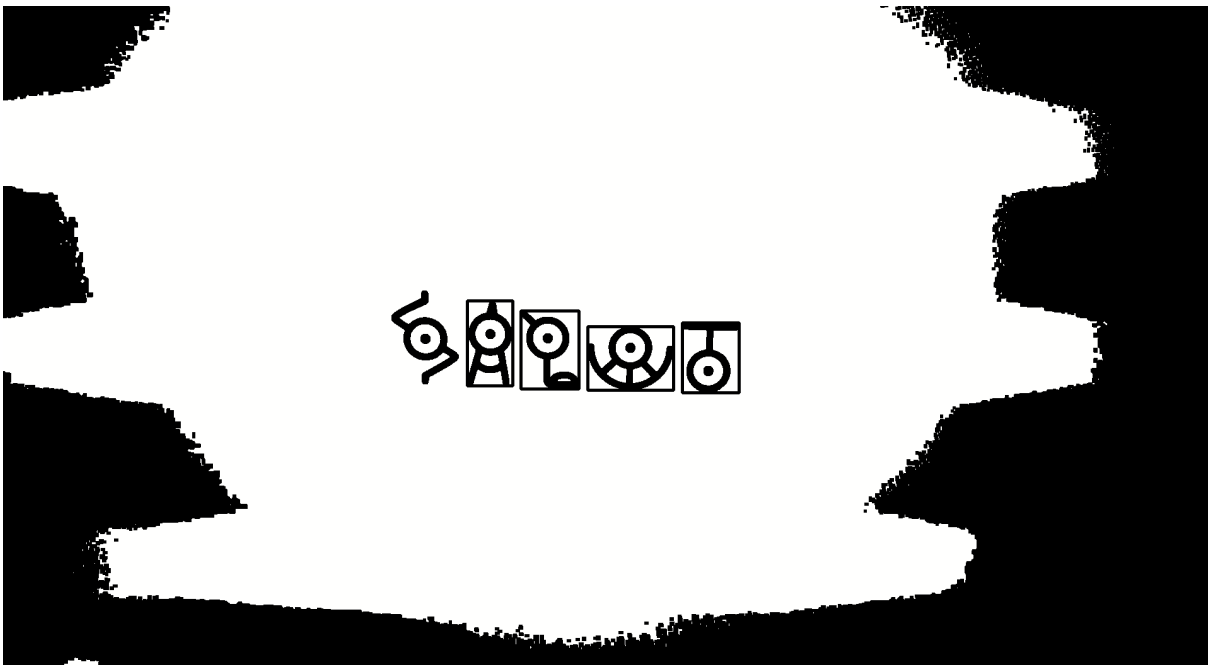
Comme le bruit de cette image n'est pas trop important, le programme a pu correctement séparer les lettres et l'ombre et ainsi traduire le texte.

4.3. Image bruitée non-fonctionnelle

Nous avons également utilisé une image prise sur une écran d'ordinateur. Nous avons directement remarqué deux grands problèmes avec cette image. Le premier concerne les lignes horizontales causées par l'écran. Le deuxième concerne les ombres dans tout les bord de l'image.



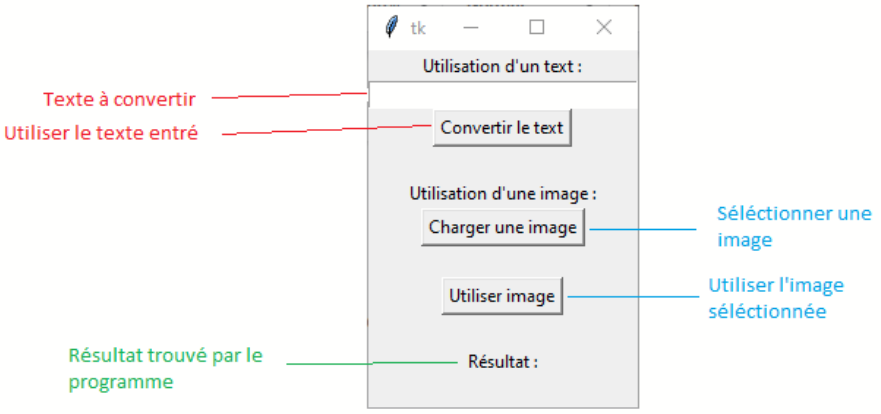
(Figure 7) Texte Zarbi pris sur un écran d'ordinateur



(Figure 8) Image bruitée après traitement

Comme on peut le voir sur la *Figure 8*, le "s" du mot "salut" n'a pas été pris en compte par le programme. Cela est causé par le tri des rectangles. Comme expliqué lors de la section "rectangles englobants", nous utilisons la moyenne pour trier les rectangles. Le "s" étant à peine trop grand, il n'est pas compté comme lettre par le programme.

5. Utilisation



Il existe deux moyens d'utiliser notre programme. Le premier consiste à entrer un texte (sans espace et que des lettres de l'alphabet) puis à appuyer sur le bouton "Convertir le texte".

La deuxième méthode consiste à sélectionner une image (des images sont à votre disposition dans le dossier ./images) puis à appuyer sur le bouton "Utiliser image".

Dans les deux cas, le résultat sera affiché dans le bas de la fenêtre.

6. Améliorations potentielles

6.1. Espace

Dans l'état actuel, le programme ne permet pas d'entrer des espaces entre les mots. Une des améliorations possibles pourrait être l'implémentation de cette fonctionnalité pour permettre à l'utilisateur d'entrer des textes et pas seulement des mots.

6.2. Images penchées

A cause de l'approche prise pour ce projet, il faudrait refaire le programme depuis la fondation (p.ex. avec les rectangles englobants d'air minimale). Cela est causé par les rectangles encombrants et le template matching. En effet, si l'image se retrouve penchée (et

ainsi les lettres), le template matching ne va pas pouvoir reconnaître les lettres et ainsi faire une fausse traduction du texte.

7. Conclusion

Pour conclure, le programme implémente tous les objectifs que nous nous sommes fixés au début du projet. Nous arrivons à traiter correctement certaines images prises par un appareil photo et à traduire le texte correctement.

Il reste, tout de même, des améliorations à apporter au projet pour correctement finaliser le projet. Par exemple, pour la détection des rectangles, il est important de trouver une autre technique pour filtrer les rectangles et ainsi corriger le problème de la *section 4.3*.