

Rapport de projet - Où est Charlie ?

Lina Mezghani - Sarah Perrin

1 Introduction

Ce rapport présente les différentes approches que nous avons eu face au problème de résoudre le célèbre jeu pour enfants « Où est Charlie », qui consiste à localiser un petit personnage caractéristique parmi une foule d'objets et de personnages colorés.



FIGURE 1 – Exemple d'enigme "Où est Charlie ?"

Voici un exemple d'image « Où est Charlie » (FIGURE 1), ainsi qu'une partie de l'image coupée (FIGURE 2) où l'on peut discerner Charlie. On peut voir que de prime abord, il n'est pas évident de le trouver, puisque même l'oeil humain peut mettre plusieurs minutes à résoudre le problème.



FIGURE 2 – Charlie

Ce rapport présente donc nos différentes approches face à cette énigme, de la plus simple à la plus complexe.

2 Approche Naïve

2.1 Fonction MatchTemplate

Lors d'une première étape nous avons tout d'abord cherché à simplifier notre problème. Pour cela, nous avons tout simplement cherché le maximum de corrélation de l'image entière avec le Charlie coupé de la photo, à l'aide de la fonction d'OpenCV `matchTemplate`.

Fort heureusement, cette méthode a abouti. Cependant, le problème était loin d'être résolu puisque cet algorithme est incapable de trouver Charlie dans de nouvelles images à partir de ce Charlie précis, notamment à cause du fond vert et des variations d'une image à l'autre de la tête de Charlie, étant donné que toutes les images sont réalisées à la main par un dessinateur. Notre tâche était donc de faire en sorte que pour une nouvelle image sans connaissance a priori de Charlie, l'ordinateur puisse le localiser.

2.2 Une première amélioration

Afin d'améliorer ce premier algorithme, nous avons cherché à localiser Charlie en rendant transparents les pixels du fond. En fait, nous avons d'abord essayé d'appliquer `matchTemplate` avec une image rectangulaire du visage de Charlie, mais la couleur de fond de cette image rectangulaire fausait le résultat. Nous avons donc pensé à rendre transparent le fond de l'image modèle (FIGURE 3) en ajoutant un quatrième canal à notre image BGR, indiquant la présence ou non du pixel dans l'image. L'algorithme détecte bien toujours le bon Charlie.



FIGURE 3 – Charlie de la FIGURE 2 sans fond

Puis, nous avons cette fois substitué à la tête originale de Charlie une autre image ressemblante (FIGURE 4) que nous avons redimensionnée pour qu'elle fasse approximativement la même taille en

terme de pixels.



FIGURE 4 – Autre image de Charlie sans fond

Nous obtenons alors trois résultats positifs, dont Charlie (FIGURE 5).



FIGURE 5 – Résultat obtenu avec l'image de Charlie de la FIGURE 4

Cependant, cette méthode a plusieurs inconvénients :

- La fonction `matchTemplate` ne permet pas de faire des variations d'échelle lors de la recherche du maximum de corrélation. Ainsi, la recherche dans une nouvelle image où la tête de Charlie serait plus grande ou plus petite ne fonctionnerait pas. Une solution pourrait éventuellement de faire une boucle et de tester `matchTemplate` avec différentes tailles de modèles en la redimensionnant.

- Sur certaines énigmes, la tête de Charlie varie significativement par rapport aux deux modèles ci-dessus. La fonction `matchTemplate` n'est alors pas capable de détecter correctement Charlie.

3 Haar cascade

3.1 Choix de cette méthode

Afin de pallier aux problèmes décrits précédemment, nous avons décidé de changer totalement d'approche et d'opter pour une méthode plus générale et robuste. Pour cela, nous avons généré un Haar classifier à l'aide d'OpenCV à partir de deux bases de données : les positifs et les négatifs. Plus précisément, nous avons généré un fichier .xml qui est notre classifier, puis nous l'avons utilisé dans le main. Nous avons opté pour cette méthode car ce type de classifier est souvent utilisé pour détecter des visages, nous avions donc l'intuition que cela pouvait également fonctionner dans le cas de Charlie.

En effet, étant donné que le Haar classifier applique des milliers de features (FIGURE 6) en les faisant convoluer avec l'image et en retranchant la somme des pixels en noir à celle des pixels en blanc, il arrive à trouver les lignes ou points principaux plus ou moins sombres dans l'image. Par exemple pour un visage, une ligne horizontale plus sombre va être détectée au niveau des yeux, et une ligne verticale plus claire entre les yeux correspondant au nez. Charlie étant en plus dessiné, ces variations sont très codifiées et nous avions bon espoir que cette méthode fonctionne.

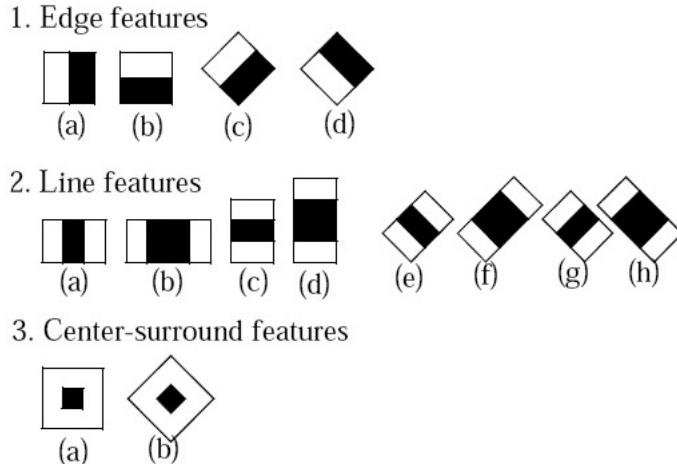


FIGURE 6 – Features utilisés dans le Haar classifier

3.2 Générer les positifs et les négatifs

Pour construire notre base de donnée de positifs, nous avons pris 41 images différentes de Charlie (FIGURE 7) extraites des livres officiels. Nous avons découpé sa tête et le haut de son buste, et retiré

le fond de l'image.



FIGURE 7 – Visages de Charlie utilisés pour générer la base de donnée des positifs

Ces images servent ensuite à générer 1500 positifs grâce à la fonction createsamples, en collant les têtes de Charlie sur des images de fonds aléatoires tirées des négatifs et en appliquant de petites déformations sur la tête de Charlie (rotation autour de l'axe vertical d'une trentaine de degrés maximum).

Pour générer la base de donnée des négatifs, nous avons découpé sept images complètes de « Où est Charlie » en rectangles de 160*200 pixels, en prenant soin de retirer les positifs (images contenant Charlie). Ceci nous a donné environ 1000 images (FIGURE 8).



FIGURE 8 – Aperçu de la base de donnée des négatifs

3.3 Paramètres pour construire la Haar Cascade

Pour créer le fichier .xml, nous avons suivi un tutoriel sur internet (lien), ainsi que consulté de nombreuses sources différentes pour comprendre l'influence des paramètres ainsi que leur réglage pour optimiser nos résultats. Après plusieurs tentatives infructueuses, nous avons finalement réussi à générer une cascade qui fonctionne sur nos exemples. Voici une liste non exhaustive des paramètres qui sont apparus comme importants :

3.3.1 La base de donnée

En tant que support du Haar classifier, c'est certainement la chose à laquelle il faut apporter le plus de soin. Lors de nos premières tentatives, nous n'avions par exemple pas retiré le fond des positifs, ce qui faisait que les positifs créés avaient tous le même fond derrière la tête de Charlie. Notre cascade manquait donc de robustesse et trouvait beaucoup de faux positifs et de faux négatifs (Charlie n'était pas trouvé).

De plus, nous avons fixé la taille des négatifs un peu plus grande que la taille des Charlie car lorsque nous faisons tourner notre algorithme, nous analysons dans des petites fenêtres de taille 160*200 (même taille que les négatifs) si un Charlie se trouve dedans ou non.

3.3.2 La taille des positifs générés dans createsamples

En consultant des forums où des personnes semblaient rencontrer le même type de problèmes que nous, nous avons trouvé qu'il était important que les positifs ne soient pas trop grands, car sinon le Haar classifier aurait du mal à trouver la présence de Charlie plus petits. Nous avons également lu que le ratio (hauteur/largeur) des images était important et devait être à peu près semblable dans les positifs. Nous avons donc fait attention à ces deux points et ceci s'est révélé décisif pour la réussite de notre projet. Les positifs générés sont tous de taille 24*30.

3.3.3 Autres paramètres

Pour lancer l'entraînement de notre classifier, nous avons exécuté la commande suivante dans la console :

```
opencv_traincascade data classifier -vec samples.vec -bg negatives.txt -numStages 20  
-minHitRate 0.999 -maxFalseAlarmRate 0.25 -numPos 1200 -numNeg 1000 -w 24 -h 30  
-mode ALL -precalcValBufSize 1024 -precalcIdxBufSize 1024
```

Dans notre cas, où beaucoup de faux positifs peuvent être détectés a priori étant donné la complexité des images, le nombre d'étape est fixé à 20, mais en pratique, notre classifier ne tournait plus après la treizième étape et nous l'avons donc arrêté. minHitRate représente la performance minimum de notre classifier à chaque étape.

3.4 Résultats

Les résultats sont encourageants malgré qu'il y ait encore beaucoup de faux positifs. En effet, lorsque l'on teste notre algorithme sur les images qui ont servi à générer les positifs, on obtient

environ 90% de réussite.

Parmi les faux positifs détectés, on obtient souvent Wenda (FIGURE 9), l'homologue féminin de Charlie, ce qui est encourageant puisqu'elle lui ressemble beaucoup.



FIGURE 9 – Wenda

Nous avons également remarqué qu'une partie des faux positifs détectés consistent en un fond uni, souvent clair, avec une barre horizontale plus sombre vers la partie supérieure (FIGURE 10), ce qui pourrait éventuellement correspondre au feature des cheveux de Charlie.

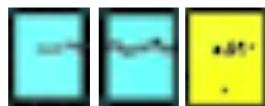


FIGURE 10 – Quelques faux positifs

4 Dernières améliorations

Au vu de nos précédents résultats, le principal problème reste les faux positifs. Nous avons donc réfléchi à deux méthodes pour faire diminuer ce nombre de faux positifs.

4.1 Déetecter les lunettes rondes de Charlie

S'il y a bien une chose facilement reconnaissable chez Charlie, ce sont ses lunettes rondes. Celles-ci sont présentes à chaque fois, et nous semblaient être un bon critère pour affiner notre recherche. En effet, si nous arrivions à détecter deux cercles approximativement de même taille et proche l'un de l'autre, alors nous saurions avec beaucoup plus de certitude que le positif détecté n'est pas un faux positif.

Nous avons donc tenté d'implémenter cette méthode avec la fonction `HoughCircles` d'OpenCV. Cependant, même en essayant de jouer au maximum avec les paramètres, les résultats étaient très mauvais (FIGURE 11). En effet, les paramètres optimaux étaient très variables selon les images, et nous n'avons pas réussi à obtenir de résultat satisfaisant. Nous avons donc décidé de nous concentrer sur la deuxième approche.

4.2 Déetecter les couleurs dans les faux positifs

4.2.1 Approche naïve

Trouver s'il y a au moins un pixel de couleur rouge parmi les positifs détectés. Cette approche étant un peu naïve, nous sommes vites passé à la suivante décrite juste après.



FIGURE 11 – Détection de cercles dans un visage de Charlie

4.2.2 k-moyennes

On fait les k-moyennes dans les positifs trouvés pour calculer les k couleurs les plus représentées. Comme les positifs contiennent également une partie de fond dont on ne connaît pas a priori la couleur, on pose k relativement grand (égal à 20). Les bornes de la couleur rouge, exprimées en HSV pour la simplicité, sont également prises relativement grandes pour ne pas éliminer le vrai Charlie dans le cas où l'image serait en mauvaise qualité. En effet, on a par exemple observé que parfois, le bonnet rouge de Charlie était en réalité composé de seulement quelques pixels oranges et noires qui donnaient de loin une impression de rouge. On cherche ensuite à vérifier que le rouge fait bien parti des 20 couleurs les plus représentées parmi les positifs. Si c'est le cas, ce positif est conservé, sinon il est éliminé.

Cette méthode nous a permis d'éliminer une grande partie des faux positifs, notamment ceux qui ne ressemblaient pas du tout à Charlie. Cependant, cela ne permet pas de faire la différence dans des cas plus fins, notamment lorsque l'ordinateur classifie Wenda comme étant Charlie.

5 Conclusion

Nous avons bien réussi à trouver où est Charlie dans la plupart des cas, même lorsque l'image ne faisait partie ni des positifs, ni des négatifs. De plus, une partie des faux positifs peut être éliminée dans le cas où ils ne contiennent pas de rouge.

Nous aurions aimé tester notre programme sur davantage de données encore, mais une de nos principales difficultés lors de ce projet a été de trouvé un nombre suffisant de ressources. En effet, le nombre d'images est limité et celles-ci sont de plus protégées ce qui rend la tâche de les trouver en qualité suffisante difficile voire impossible pour certaines images. Ceci a été une de nos limitations principales à l'agrandissement de notre base de donnée et aux tests que l'on a effectués ensuite.

Bien sûr, notre projet peut encore être amélioré. Nous pourrions sûrement diminuer le nombre de faux positifs en rajoutant davantage d'images dans les négatifs de notre classifier, et tenter d'éliminer Wenda en ajoutant beaucoup d'images d'elle également.