Comparaison entre le RANSAC et un Algorithme Focalisant pour la Reconnaissance d'Objets *

C. Bernay-Angeletti[1] R. Aufrère[1,2] R. Chapuis[1]

[1]Institut Pascal UMR 6602 UBP / CNRS
Campus des Cézeaux 63177 AUBIERE Cedex

[2] LIMOS UMR 6158 UBP/ CNRS
Campus des Cézeaux 63177 AUBIERE Cedex

Coralie.Bernay-Angeletti@etudiant.univ-bpclermont.fr,
(romual.aufrere, roland.chapuis)@univ-bpclermont.fr

Résumé

La reconnaissance d'objets est un problème complexe. Nous proposons ici une méthode originale reposant sur une approche focalisante de type top-down. Cette approche utilise au mieux toutes les informations disponibles sur l'objet à détecter tout en réduisant l'espace de recherche au fur et à mesure de l'évolution de l'algorithme et supprime ainsi rapidement les hypothèses erronnées. D'un autre côté, les algorithmes "forces brutes" tels que le RANSAC ou ses variantes sont très rapides pour traiter une hypothèse mais doivent en contrepartie en examiner de nombreuses avant d'arriver à la solution. L'objectif de cet article est de savoir quand vaut-il mieux utiliser notre méthode par rapport à un algorithme classique de type RANSAC.

Mots Clef

Reconnaissance d'objets, focalisation, RANSAC

Abstract

Object recognition is a complex problem. We propose here an original method based on a top-down approach with focalisation. This approach makes best use of all available information on the object to be detected while reducing the space of search according to the evolution of the algorithm and it eliminates quickly bad hypotheses. On the other hand, the "brute-force" algorithms such as the RANSAC or its variants are very fast to handle an hypothesis but have to examine it in return of numerous before arriving at the solution. The objective of this article is to know when it is better to use our method or an classical algorithme like RANSAC.

Keywords

object recognition, focalisation, RANSAC

*Ce travail a bénéficié d'une aide de l'État gérée par l'Agence Nationale de la Recherche au titre du programme Investissements d'avenir dans le cadre des projets EquipEx Robotex (ANR-10-EQPX-44) et LabEx IMobS3 (ANR-10-LABX-16-01), d'une aide de l'Union Européenne au titre du Programme Compétitivité Régionale et Emploi 2007-2013 (FEDER – Région Auvergne), d'une aide de la Région Auvergne et de l'Institut Français de Mécanique Avancée

1 Introduction

La reconnaissance d'objets est un enjeu important en imagerie. Elle sert à de multiples tâches telles que : la détection automatique d'anomalies dans un environnement donné ; la réalisation de statistiques en dénombrant la population présente dans une série d'images (d'animaux, de piétons, de véhicules ou autres) ; une meilleure interprétation de l'environnement à des fins de localisation [1]. Cependant, ce n'est pas une tâche aisée [8] : les algorithmes doivent être robustes pour pouvoir travailler selon différentes illuminations, malgré les occultations et la grande variabilité d'objet au sein d'une même classe. Ils doivent également être suffisamment rapides pour être utilisables dans l'application souhaitée.

Les approches génériques de type grammaire permettent une grande robustesse et une grande variation intraclasse. Elles utilisent une décomposition de l'objet en sous parties qui sont reliées entre elles par des relations de positions les unes par rapport aux autres. Certains groupements vont être priviligiés : ainsi dans [7], les auteurs indiquent qu'il vaut mieux commencer par construire les roues avant de construire le cadre d'un vélo. Le défaut principal de ces approches demeure le temps nécessaire à la détection (une dizaine de secondes pour le vélo). Dans [10], la détection nécessite environ 30s en raison de l'apprentissage automatique de ces méthodes.

Notre algorithme, appelé Algorithme Focalisant, abrégé AF pour la suite de l'article, a pour objectif d'utiliser de manière intelligente les informations dont il dispose. Pour ce faire, l'objet est représenté comme un vecteur d'état $\underline{X}$ et une covariance associée C_x. La reconnaissance consiste à mettre à jour ce vecteur d'état et sa covariance tout en gérant la confiance p (probabilité d'une bonne reconnaissance) à chaque itération. Différentes parties forment cet objet. Ces parties peuvent extraire du vecteur d'état, caractérisant l'ensemble de l'objet à détecter, des informations sur leurs positions et leurs formes possibles. Par ailleurs, une fois détectées, elles permettent une remise à jour du vecteur d'état global. Par exemple, si l'objet à détecter est une tête humaine, une fois le contour du visage détecté,

les positions et tailles possibles des yeux sont grandement restreintes, la détection d'un oeil permettra d'affiner à nouveau la position des autres parties. Pour détecter l'objet, il va donc suffire de détecter les différentes parties. L'AF sera plus détaillé dans la section suivante.
Les détecteurs renvoient une liste de primitives (des contours de visages, des cercles...). Détecter l'objet complet (la tête dans l'exemple) revient alors à associer les bonnes primitives aux différentes parties. La détection devient alors un problème d'associations. C'est la raison pour laquelle nous confrontons l'AF à des approches plus traditionnelles d'association de données de type RANSAC[6].
Le RANSAC est un algorithme très connu. Il est notamment utilisé en vision pour le calcul de poses. C'est un algorithme qui peut être très rapide et relativement robuste au bruit. Il est également très générique ce qui permet de l'adapter à une multitude de problèmes. Cependant, il devient rapidement gourmand en temps de calcul lorsque le problème devient plus complexe. C'est pourquoi, il existe plusieurs extensions du RANSAC qui permettent de le rendre plus fiable comme le DEGENSAC [5]. Le LO-RANSAC [4] fait une optimisation locale sur le set d'inliers afin de respecter la convergence vers la solution. Mais les approches dérivées de la version initiale cherchent surtout à optimiser les temps de calcul pour atteindre la solution finale. Le PROSAC[3] sélectionne prioritairement les candidats les plus prometteurs car plus ressemblants aux points d'origine. Le SCRAMSAC [9] va exiger une certaine consistance des données dans un voisinage.
Il est alors intéressant de savoir à partir de quand il vaut mieux utiliser l'AF par rapport à un RANSAC (ses variantes étant pour la plupart clairement teintées "calcul de pose" et souvent les heuristiques dépendent des données).
Dans une première partie, nous présenterons brièvement l'AF proposé. Dans une deuxième, nous ferons une comparaison théorique entre le RANSAC et l'AF. Dans une troisième partie, une comparaison sur un exemple sera faite.

2 Brève présentation de l'AF

Un objet sera caractérisé par trois paramètres : un vecteur d'état $\underline{X}$, une covariance C_x et une probabilité p ; voir [2]. Le vecteur d'état $\underline{X}$ et sa covariance C_x permettent de connaitre l'estimation de la position de l'objet avec son incertitude associée. La probabilité p représente la confiance en l'estimation donnée par $\underline{X}$ et C_x de l'objet.
L'objet à détecter est composé de plusieurs sous parties considérées comme des agents. Par exemple, une tête est constituée d'un oeil droit, un oeil gauche, une bouche, un nez... La définition des différentes parties n'est pas l'objectif de cet article, il est admis que celles choisies sont pertinentes et suffisantes pour permettre la détection de l'objet complet. A chaque partie est associée une primitive. Par exemple, la partie $OeilDroit$ et la partie $OeilGauche$ nécessiteront la même primitive : $cercle$.
Le fonctionnement de l'AF est alors représenté par la figure 1. Après une phase d'initialisation, les étapes qui se

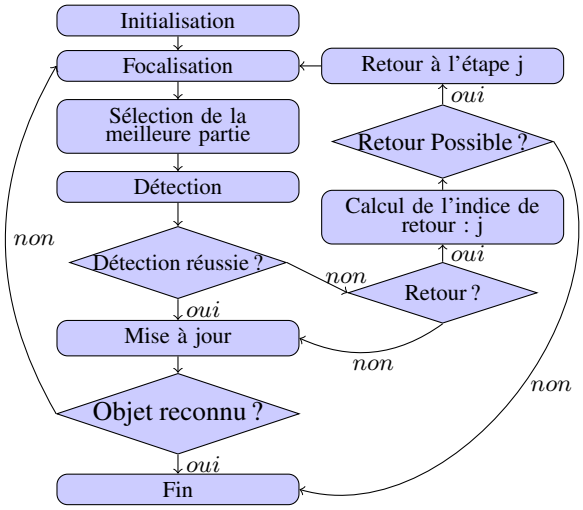

FIGURE 1 – Fonctionnement de l'AF

répètent sont les suivantes :

- la focalisation : l'état actuel de l'objet permet de focaliser les parties, c'est à dire calculer à la fois la zone de l'image dans laquelle elles peuvent se trouver mais aussi quelles valeurs pour leurs primitives sont acceptables.
- la sélection de la meilleure partie : pour chaque partie un critère est calculé avant toute détection. Il prend en compte le temps que nécessitera la détection, la probabilité de trouver une primitive correspondant à nos attentes et s'il en existe plusieurs les chances de choisir celle correspondant réellement à la partie recherchée. La partie ayant le critère le plus élevé sera choisie.
- la détection : la meilleure partie lance son détecteur. Celui-ci renvoie une liste de primitives. Pour toutes les primitives pouvant correspondre à la partie recherchée, une distance de Mahalanobis entre ce qui était attendu et ce qui a été trouvé est calculée. S'il y en a au moins une pour laquelle cette distance est inférieure à un seuil fixé, la détection est considérée réussie et c'est la primitive pour laquelle la distance est la plus petite qui est choisie.
- l'éventuel retour en arrière : il est possible qu' une mauvaise association primitive/ partie ait été faite entrainant par la suite la non détection d'une ou plusieurs parties. Par conséquent, lorsque la confiance devient trop faible et que la détection a échoué, un retour en arrière est effectué jusqu'à la dernière détection réussie. Celle-ci est annulée (ainsi que toutes les non détections qui l'avaient suivie s'il y en a) et la primitive est marquée : elle ne pourra plus être utilisée pour cette partie. Si aucune détection n'a réussie, l'algorithme se termine par un échec : l'objet n'a pas pu être trouvé.
- mise à jour : la confiance est toujours réestimée mais la position ne le sera que si la détection a réussi.
- test d'arrêt : si les objectifs de précision et de confiance sont atteints, l'AF cesse et l'objet a été trouvé.

Pour plus de détails sur le fonctionnement de l'AF voir [2].

3 Comparaison théorique

3.1 Scénario

L'objectif est de détecter un objet O qui peut être décomposé en M parties, chacune d'elles nécessitant la détection d'exactement 1 primitive.

Hypothèses. Les hypothèses sont les suivantes :
- l'objet est présent dans la scène en un seul exemplaire.
- toutes les primitives constituant l'objet ont été détectées et éventuellement d'autres (considérées comme bruit).
- une primitive correspond à une seule et unique partie.
- la détection de l'objet nécessite la détection de toutes ses parties (donc de toutes les primitives associées).
- les algorithmes testent toutes les associations parties/primitives admissibles. Un modèle de l'objet est construit à l'aide de ses associations (c'est à dire que son vecteur d'état et sa covariance sont calculés). Le modèle peut être éliminé s'il n'est pas cohérent. Le meilleur modèle parmi tous ceux testés est conservé.
- l'ordre dans lequel les parties seront détectées sera connu à l'avance et toujours le même indépendamment des résultats des détections.

Par exemple, il faut détecter un carré parfait. Les quatres parties qui le constituent vont être la $barreDroite$ la $barreGauche$, la $barreBasse$ et la $barreHaute$. Il n'y a ici qu'un seul type de primitives utile : le $segment$. L'ordre de détection établi peut être par exemple $barreDroite$, $barreBasse$, $barreGauche$ et $barreHaute$. Le vecteur d'état de l'objet contient : la position du centre du carré et la taille d'un côté. Une fois un bord détecté, la position des autres est fixée et donc les primitives sont connues. Pour vérifier que le modèle construit est viable, il suffit alors de vérifier si les autres primitives existent effectivement.

Notations.
- T : le nombre de primitives différentes nécessaires pour la détection de l'objet (plusieurs parties peuvent avoir besoin d'une primitive de même type). Donc $T \leq M$.
- nP_i : le nombre de parties nécessitant une primitive de type $i, 1 \leq i \leq T$. D'où $\sum_{i=1}^{T} nP_i = M$
- p_i : le nombre de primitives de type i. $p_i \geq nP_i$ (car l'objet est présent et toutes les primitives détectées).
- k_i : le nombre de choix possibles pour chaque partie (c'est à dire le nombre de primitives acceptables pour cette partie).
- C_i : le nombre de combinaisons différentes de taille i qu'il est possible de construire en respectant les contraintes. C_M représente alors soit le nombre d'objets dans l'image soit le nombre de façons différentes de parvenir à un même objet. C_0 représente le nombre de combinaison vide vaut donc 1.

3.2 Cas du RANSAC

Rappel rapide du principe du RANSAC. En pseudo code, il est possible d'écrire le RANSAC comme suit :

Initialisation
meilleur-modele = aucun ; meilleure-erreur = ∞ ;
while *!(critere de fin)* **do**
  tirer aléatoirement des données
  créer un modèle à partir de ce sous ensemble de données
  déterminer les inliers et les outliers
  calculer l'erreur
  if *erreur* < *meilleure-erreur* **then**
    meilleure-erreur = erreur ; meilleur-modele = modele ;
  end
end

Algorithme 1 : Pseudo code pour le RANSAC

En sortie, le meilleur modèle et son erreur associée sont renvoyés. Le *critère de fin* est le plus souvent du style $i < nbIterMax$ où i décompte le nombre d'itérations déjà exécutées. $nbIterMax$ est généralement fixé afin d'être sûr, avec une probabilité donnée, d'avoir bien trouvé le modèle cherché lorsque l'algorithme se termine. Il peut aussi servir à assurer un temps de calcul maximum mais sans garantie de convergence vers la solution.

Nombre de Combinaisons ($NCRansac$). C_i est le nombre de combinaisons différentes de taille i qu'il est possible de construire en respectant les contraintes. Pour le RANSAC, la seule contrainte est de ne pas jamais prendre plus d'une fois une primitive (une primitive ne peut servir que pour une seule partie). Pour trouver l'objet, il faut avoir trouvé toutes ses parties d'où $C_M = NCRansac$. Notons $\phi_1^i \quad ... \quad \phi_{nP_i}^i$ les nP_i parties nécessitant une primitive de type i. D'où :

$$k_{\phi_1^i} = p_i, \quad k_{\phi_2^i} = p_i - 1, \quad ..., \quad k_{\phi_{nP_i}^i} = p_i - (nP_i - 1) \tag{1}$$

Pour chaque partie l, il y a k_l choix possibles.

$$\text{Donc} \qquad NCRansac = k_1 k_2 ... k_M \tag{2}$$

$$\text{Or} \qquad \forall l, \exists \text{un unique couple} i, j \text{ tels que } l = \phi_j^i \tag{3}$$

En utilisant (3) dans (2) et en reordonnançant :

$$NCRansac = k_{\phi_1^1} ... k_{\phi_{nP_1}^1} k_{\phi_1^2} ... k_{\phi_{nP_2}^2} ... k_{\phi_1^T} ... k_{\phi_{nP_T}^T} \tag{4}$$

Avec (1), (4) devient :

$$\begin{aligned} NCRansac = \quad & p_1(p_1-1)...(p_1-(nP_1-1)) \\ & p_2(p_2-1)...(p_2-(nP_2-1)) \\ & ... \\ & p_T(p_T-1)...(p_T-(nP_T-1)) \end{aligned} \tag{5}$$

Ce qui peut aussi s'écrire :

$$NCRansac = \frac{p_1!}{(p_1-nP_1)!}\frac{p_2!}{(p_2-nP_2)!}...\frac{p_T!}{(p_T-nP_T)!} \tag{6}$$

Nombre d'Iterations ($nbIterRansac$). Une itération se déroulera comme suit :
- pour chaque partie, tirage aléatoire d'une primitive du bon type parmi les primitives non utilisées
- calcul du modèle d'objet correspondant
- évaluation du modèle

Puisque toutes les combinaisons doivent être testées : $nbIterRansac = NCRansac$.

La complexité pour le RANSAC va donc dépendre à la fois du nombre de parties composant l'objet et du nombre de primitives de type intéressant dans l'image.

3.3 Cas de l'Algorithme Focalisant

Remarques et hypothèses. Un des grands principes de l'AF est de focaliser. Ainsi, au lieu d'examiner toutes les combinaisons possibles, seules celles respectant les contraintes exigées par l'état actuel le seront. L'ordre de détection est donc un facteur essentiel. Ici, cet ordre est supposé fixe, bien qu'en pratique, il soit recalculé au fur et à mesure.
Il sera de plus admis que le moindre échec de détection sera suffisant pour remettre en cause le modèle et déclencher un retour en arrière. C'est vrai ici car le détecteur est supposé avoir renvoyé toutes les primitives sans exception. Sinon, la faute pourrait éventuellement être rejetée sur le détecteur et l'algorithme se poursuivrait quand même jusqu'à une autre non détection. Cette hypothèse est acceptable ici puisque le RANSAC ne gère pas les non détections.

Combinaisons sous contraintes. Sur un exemple, il est facile de donner "à la main" la valeur des C_i (le nombre de combinaisons de taille i respectant les contraintes qui peuvent être construites). Le décompte manuel peut paraître très contraignant et peu réaliste dans le fonctionnement. Cependant il est en fait possible d'en faire une approximation raisonnable. En effet, imaginons connaître la densité de primitives dans un ellipsoide initial (donc dans un volume multidimentionnel initial), par exemple la densité de primitives de type point dans une aire donnée. Rechercher une primitive respectant des contraintes revient à diminuer les paramètres acceptables pour celle-ci et donc à définir un nouvel ellipsoide acceptable (donc un nouveau volume). Il suffit alors de faire le ratio du nouveau volume sur le volume initial multiplié par la densité initiale de primitives pour obtenir une approximation du nombre de primitives vérifiant les contraintes. Les contraintes dépendant des combinaisons précédentes, cela nous donne en fait le nombre probable de choix possibles pour poursuivre une combinaison (noté k_i^c). Pour obtenir le nombre de combinaisons de taille supérieure, il suffit alors de multiplier le nombre probable de choix par combinaisons par le nombre de combinaisons de taille inferieure : $C_{i+1} \approx C_i k_i^c$.

Définition d'une itération. Une itération comprend :
- choix d'une partie à détecter
- détection des primitives
- (si possible) choix d'une primitive
- mise à jour du modèle
- (si nécessaire) un retour en arrière

Calcul du nombre d'itérations($nbIterAF$). Pour construire toutes les combinaisons de taille 1, il va falloir C_1 itérations. En outre, si l'objet n'est pas présent dans l'image, il faudra une itération supplémentaire qui lancera une détection pour vérifier qu'il n'y a aucun candidat potentiel. Par contre, si l'objet est présent, il y aura au moins une des combinaisons C_1 qui pourra se poursuivre et parvenir à la détection.
Posons maintenant C_i^f : le nombre de combinaisons de taille i pour lesquelles il existe au moins une combinaison finale (c'est à dire une combinaison de taille n) qui débutent par elles. En particulier : $C_0^f = 1$ s'il y a au moins un objet dans la scène et 0 sinon ; $C_n^f = C_n$ par définition. Ces coefficients seront calculées dans la section suivante.
Notons I_i : le nombre d'itérations nécessaires pour traiter toutes les combinaisons de taille inférieure ou égale à i.
Ainsi $I_1 = C_1 + C_0^f$.
De manière plus générale, pour étudier toutes les combinaisons de taille $i+1$, il faut déjà avoir étudié toutes les combinaisons de taille i. Parmi ces combinaisons, une partie va donner naissance à une ou plusieurs combinaisons de taille supérieure, chaque combinaison restante nécessite en outre une itération pour être invalidée. Il faut donc C_{i+1} itérations pour construire toutes les combinaisons de taille $i+1$ et $C_{i-1} - C_{i-1}^f$ itérations pour supprimer les mauvaises pistes. Ainsi $I_{i+1} = I_i + (C_i - C_i^f) + C_{i+1}$.
Le nombre totale d'itération est alors $nbIterAF = I_n$.

Calcul des C_i^f. $C_n^f = C_n$, en effet, toute combinaison de taille n est finale par définition. Pour $i = n$, le nombre de combinaisons finales est connu. L'objectif est donc de calculer par récurrence le nombre de combinaisons finales de taille $i \quad \forall i$. Supposons connaitre le nombre de combinaisons finales de taille $i+1$, c'est à dire C_{i+1}^f. C_i est le nombre de combinaison de taille i qu'il est possible de créer. De plus, la répartition des combinaisons finales sera supposée uniforme : s'il y a 8 combinaisons de taille $i+1$ qui sont finales et qu'il existe 4 combinaisons de taille i, c'est que chaque combinaison de taille i donne naissance à 2 combinaisons finales de taille supérieure. Avec ces hypothèses, si $C_i <= C_{i+1}^f$, cela signifie que chaque combinaison de taille i donnera naissance à au moins une combinaison finale de taille $i+1$ et donc chaque combinaison de taille i est finale : $C_i^f = C_i$. Si $C_i > C_{i+1}^f$, seule une partie des combinaisons de taille i est finale et comme la répartition est uniforme, il y a C_{i+1}^f parmi les C_i qui sont finales. En regroupant les deux, $C_i^f = min(C_{i+1}^f, C_i)$.

3.4 Conclusion de la comparaison théorique

Cette comparaison permet de mettre en avant les critères qui vont jouer sur le temps de calcul. Le nombre de parties va déterminer la taille maximale des combinaisons. Le bruit (outliers) va également avoir une grande influence. Pour le RANSAC, de manière évidente, plus il y a de bruit, plus il y a de chances de tirer aléatoirement une mauvaise primitive. Pour l'AF, plus il y a de bruit, plus la densité de primitives sera grande et donc plus il y aura de mauvaises combinaisons. Globalement, l'AF sera quand même moins sensible au bruit car seule une partie du bruit entrainera la création de mauvaises combinaisons. En effet une grande partie des mauvaises primitives ne respectera pas les contraintes et ne pourra donc pas poursuivre des combinaisons. Enfin, les temps de calculs de l'AF dépendent des contraintes qu'il fixe. Par exemple, entre détecter un carré parfait et détecter un polygone à 4 sommets, le premier sera davantage contraint : en effet une fois la première partie détectée, les autres parties n'ont guère de choix pour

les primitives acceptables alors que ce n'est pas le cas pour le second. Il y a donc trois critères qui influencent vraiment les temps de calculs : le nombre de parties, le bruit et uniquement pour l'AF les contraintes. Les expressions littérales obtenues sont pour le reste difficilement comparables.

4 Comparaison sur un exemple

4.1 Objectifs

L'objectif est de comparer le RANSAC et l'algorithme focalisant sur un exemple donné. Les deux critères communs seront testés : le bruit et le nombre de parties dans l'image.

4.2 Conditions d'expérimentation

L'objet à détecter sera ici un polygone à n sommets. Les parties de l'objet sont les côtés qui constituent le polygone. La figure 2 montre des exemples de polygones à 4 sommets (en vert) avec d'autres segments (en magenta) présents dans l'image.

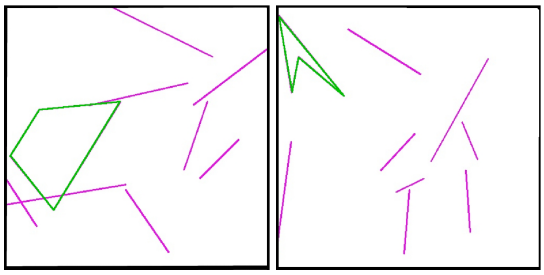

FIGURE 2 – Deux exemples de polygones à 4 sommets

D'un point de vue technique, les segments sont créés aléatoirement et librement dans l'image sauf pour les n segments du polygone qui doivent être reliés entre eux. Les algorithmes ont accès aux coordonnées de tous les segments. Le temps de détection des segments ne sera donc pas comptabilisé dans cette comparaison.

Les deux algorithmes disposeront de trois secondes maximum pour arriver à trouver le modèle. Celui-ci étant connu, chaque algorithme se terminera dès lors qu'il l'aura trouvé. Etant donné l'aléatoire des deux algorithmes, cent tirages seront effectués pour chaque couple de valeurs nombre de sommets/ bruit ; cela permettra à la fois d'estimer un temps moyen et d'obtenir un taux de réussite.

Pour le RANSAC, le tirage aléatoire devra être ordonné : c'est à dire que si l_1, ..., l_n sont les lignes qui constituent le polygone, il faudra qu'il tire aléatoirement l_i l_{i+1} ... l_n l_1 l_2 l_{i-1} dans cet ordre là, la valeur i de départ est par contre sans importance. L'ordre de tirage est l'ordre d'affectation des primitives aux parties.

Pour l'algorithme focalisant, cette question se pose moins puisqu'il sait à chaque instant pour quelle partie il est en train d'effectuer sa recherche.

Afin de certifier que les deux algorithmes ont bien les mêmes données, la même liste initiale de primitives leur est fournie. A chaque itération, l'AF cherchera pour la partie choisie, la primitive qui correspond le mieux à ses attentes dans cette liste de primitives.

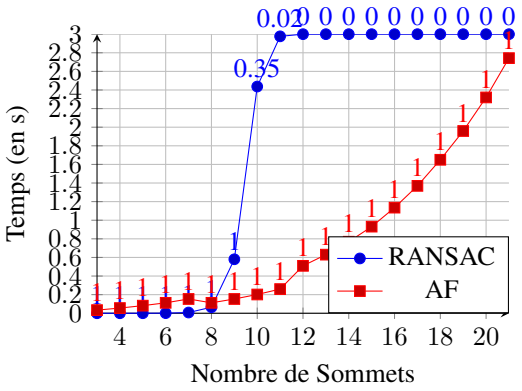

FIGURE 3 – Comparaison en fonction du nombre de sommets

4.3 Situation sans bruit (sans outliers)

Ici, le nombre de sommets du polygone va varier de 3 jusqu'à 22. Il n'y a aucun bruit donc seuls les segments constituant le polygone sont disponibles (segments verts sur la figure 2).

Sur les figures 3 jusqu'à 6, les nombres au dessus des points indiquent le taux de réussite : 1 signifie 100% de réussite et 0.35 correspond à 35 résultats positifs sur les 100 essais. Nous constatons que le RANSAC est bien plus rapide au début, par contre, il arrive rapidement un seuil critique où son temps augmente brutalement. En deçà de 8 sommets, l'AF est moins rapide mais au delà, il devient le plus intéressant.

C'est maintenant l'influence du bruit qui va être étudiée pour un nombre de sommets allant de 3 à 8.

4.4 Avec bruit

Le bruit (exprimé en %) est défini comme le rapport du nombre d'outliers (c'est à dire le nombres de segments n'appartenant pas au polygone) sur le nombre total de segments (inliers + outliers).

Pour des raisons de place, les figures pour 4, 6 et 7 sommets ne seront pas présentées mais les tests ont été réalisés similairement et les résultats seront donnés dans le tableau récapitulatif.

Les figures 4 à 6 montrent qu'au fur et à mesure que le nombre de sommets augmente, le RANSAC devient de plus en plus sensible au bruit et perd de son efficacité. Dans ces situations l'AF est plus pertinent.

4.5 Tableau Récapitulatif

Bruit	Nombres de Sommets						
	3	4	5	6	7	8	≥9
0%	R	R	R	R	R	R	AF
≥5%	R	R	R	R	R	AF	AF
≥23%	R	R	R	R	AF	AF	AF
≥40%	R	R	R	AF	AF	AF	AF
≥53%	R	R	AF	AF	AF	AF	AF
≥80%	R	AF	AF	AF	AF	AF	AF

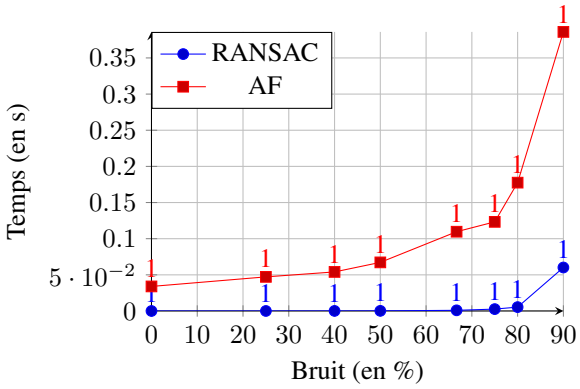

FIGURE 4 – Comparaison pour 3 Sommets

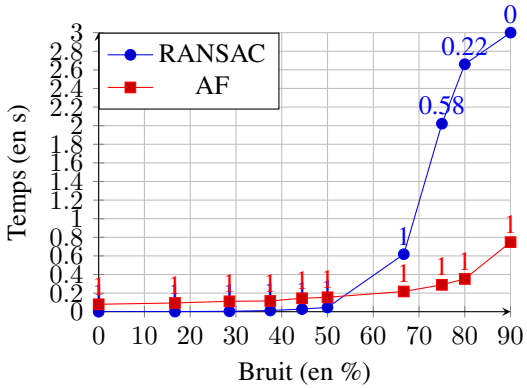

FIGURE 5 – Comparaison pour 5 Sommets

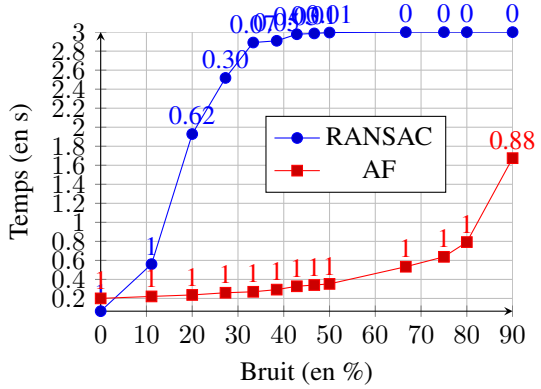

FIGURE 6 – Comparaison pour 8 Sommets

Le tableau comparatif permet de voir quelles sont les situations plus propices à chaque algorithme.

5 Conclusion et Perspectives

Les résultats observés correspondent aux résultats attendus : le RANSAC est bien meilleur dès qu'il s'agit de petits problèmes peu bruités. Par contre, l'AF est prometteur dès lors que le bruit et la complexité augmentent. Il faut en outre noter que les conditions d'expérimentations étaient favorables au RANSAC : en effet, l'AF espère aussi gagner du temps en ne lançant pas nécessairement des détections dans toute l'image. De plus, sous certaines hypothèses a priori, l'AF peut également tolérer la non-détection d'une ou plusieurs parties (par exemple si dans le modèle, la partie est optionnelle comme des lunettes sur un visage) et trouver malgré tout l'objet. Cela permet donc une plus grande souplesse. Cet article a permis de comparer notre approche qui utilise au mieux les informations disponibles avec une approche générique sans connaissance aucune mais très rapide pour traiter une hypothèse. Les résultats montrent que l'intelligence a un coût qui ne devient rentable que lorsque le problème est difficile soit en raison de sa complexité (un grand nombre de parties) soit en raison d'un trop fort bruit. Ceci nous permet de savoir quand utiliser l'AF par rapport au RANSAC selon les situations. Il reste plusieurs améliorations à faire au niveau de l'AF afin de permettre de gérer les occultations en particulier.Une autre étape va être de le comparer à des algorithmes spécialisés tel que la détection et le suivi de piétons. Ces comparaisons permettront de prouver l'adaptabilité et la robustesse de l'AF.

Références

[1] Claude Aynaud, Coralie Bernay-Angeletti, Romuald Aufrère, Christophe Debain, and Roland Chapuis. Sélection des données et regard critique sur le résultat dans le cadre de la localisation de véhicules. In *GRETSI'2013*, Brest, France, September 2013.

[2] Coralie Bernay-Angeletti, Claude Aynaud, Romuald Aufrère, and Roland Chapuis. Stratégie de perception active pour l'interprétation de scènes : Application à une scène routière. In *ORASIS*, Cluny, France, 2013.

[3] Ondrej Chum and Jiri Matas. Matching with PROSAC-progressive sample consensus. In *IEEE Computer Society Conference on, Computer Vision and Pattern Recognition*, volume 1, page 220–226, San Diego, CA, USA, 2005.

[4] Ondrej Chum, Jirí Matas, and Josef Kittler. Locally optimized RANSAC. In *Pattern Recognition*, page 236–243. Springer Berlin Heidelberg, 2003.

[5] Ondrej Chum, Tomas Werner, and Jiri Matas. Two-view geometry estimation unaffected by a dominant plane. In *IEEE Computer Society Conference on, Computer Vision and Pattern Recognition*, volume 1, page 772–779, San Diego, CA, USA, 2005.

[6] Martin A. Fischler and Robert C. Bolles. Random sample consensus : a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6) :381–395, 1981.

[7] Liang Lin, Tianfu Wu, Jake Porway, and Zijian Xu. A stochastic graph grammar for compositional object representation and recognition. *Pattern Recognition*, 42(7) :1297–1307, July 2009.

[8] Dilip K. Prasad. Survey of the problem of object detection in real images. *International Journal of Image Processing (IJIP)*, 6(6) :441, 2012.

[9] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. SCRAMSAC : improving RANSAC's efficiency with a spatial consistency filter. In *IEEE 12th International Conference on, Computer Vision*, page 2090–2097, Kyoto, Japan, 2009.

[10] Z. Si and S. C. Zhu. Learning and-or templates for object recognition and detection. Technical report, Statistics Dept., UCLA, 2011. 4, October 2012.