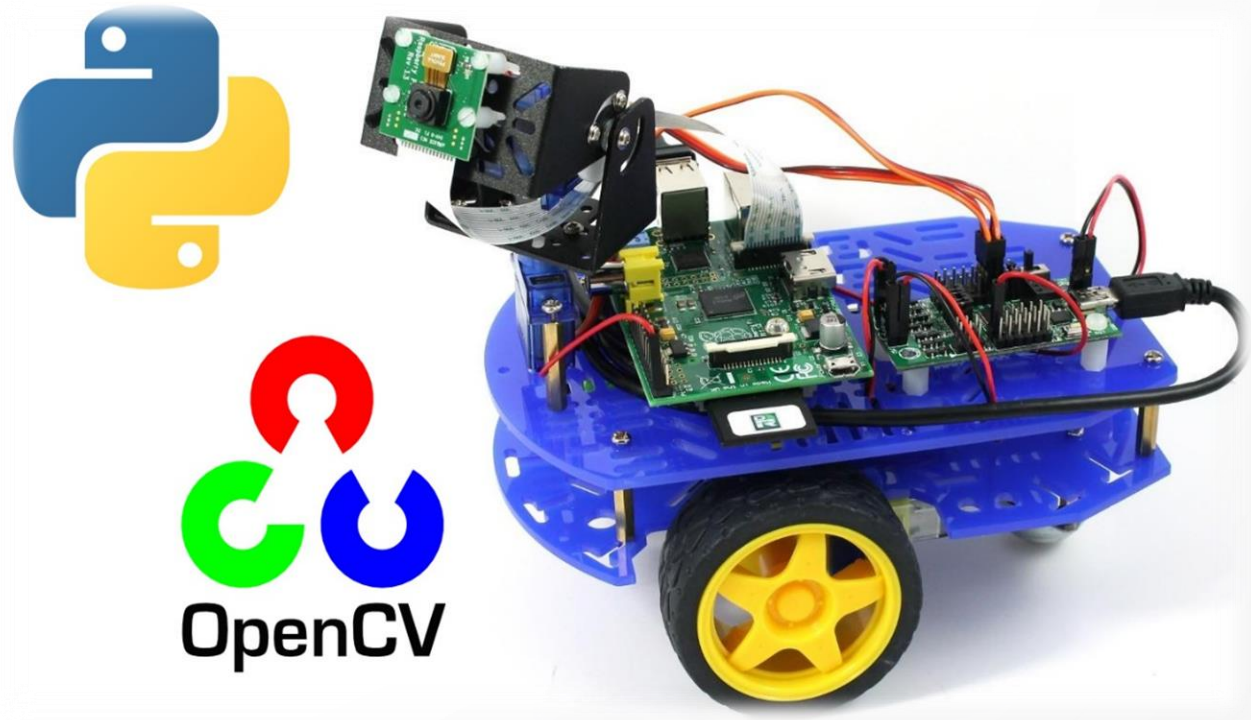
تعلم OpenCv ببساطة

Simply OpenCv



طريقك نحو تعلم الرؤية الحاسوبية

ومعالجة الصورة الرقمية

Computer Vision

المهندس خالد الدبش

الإصدار الأول 2017


بسم الله الرحمن الرحيم

رُخصة الكتاب

شكراً للمساهمين في مراجعة الكتاب

م. عبدالله علي، م. عبدالله عادل، م.نورهان علاء

للتواصل مع المؤلف

Khaleddobosh544@gmail.com

حسابLinkedIn

http://www.linkedin.com/in/khaled-aldobosh

خالد الدبش

١٤٣٨ هـ الموافق ٢٠١٧ م

قال الله ﷻ في القران الكريم:

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

ٱقْرَأْ بِٱسْمِ رَبِّكَ ٱلَّذِى خَلَقَ ﴿١﴾ خَلَقَ ٱلْإِنسَـٰنَ مِنْ عَلَقٍ ﴿٢﴾
ٱقْرَأْ وَرَبُّكَ ٱلْأَكْرَمُ ﴿٣﴾ ٱلَّذِى عَلَّمَ بِٱلْقَلَمِ ﴿٤﴾ عَلَّمَ ٱلْإِنسَـٰنَ
مَا لَمْ يَعْلَمْ ﴿٥﴾

مقدمة الكاتب

بسم الله الرحمن الرحيم

الحمد لله رب العالمين، بعد مرور أكثر من ٨ أشهر من العمل المستمر لإعداد هذا الكتاب انتهيت من كتابته وظهر كما تراه بهذا التنسيق الذي استغرق من وقتي وجهدي الكثير. الكتاب ليس ترجمة حرفية لكتاب آخر وإنما من تأليفي، حيث اعتمدت على مرجع رئيسي في الكتابة وهو الدعم الذي تقدمه مكتبة opencv والموجود على موقعها، والعديد من المراجع والمواقع الآخرى.

جميع الأكواد التي في الكتاب تم تطبيقها وتجريبها بشكل عملي

أغلب المراجع الأجنبية التي قرأتها لتعلم الرؤية الحاسوبية باستخدام مكتبة opencv لا تهتم بشرح الكود إلا جزء يسير منه أو قد لا يتم التطرق للشرح أبدا لذا استثمرت الكثير من الوقت لفهم الكود وشرحه للقارئ فقد كانت خبرتي بلغة بايثون صفر أما الآن ولله الحمد فأتقنت لغة بايثون.

في المراجع الأجنبية يكون هناك افتراض مسبق بخبرتك في لغة البرمجة بايثون، أما في هذا الكتاب فاعتبرت أن القارئ لا يعلم شيئ عن اللغة لذا بدأت معه من الصفر باستثناء أن يكون لديه معرفة ولو صغيرة بالبرمجة كتعليمة if وelse if وfor وwhile.

كل فقرة فيها مقدمة نظرية وجانب عملي فإذا استصعب عليك فهم المقدمة النظرية للفقرة فلا تقلق فأغلب المقدمات النظرية يمكنك تخطيها دون فهمها حيث يمكنك الانتقال للجانب العملي وتطبيق الكود بشكل مباشر.

في البداية سأشرح أساسيات معالجة الصورة التي ستساعدنا في بداية الانطلاق بمجال الرؤية الحاسوبية، وبعد ذلك سأشرح أساسيات لغة البايثون، بعد ذلك يمكنك البدء بتعلم استخدام مكتبة OpenCv (المكتبة البرمجية المفتوحة للرؤية الحاسوبية).

إهداء

إلى الرّحمة المهداةِ للعالمين..
إلى الصادقِ الوعدِ الأمين..
إلى الذي تشتاقُ لرؤيتهِ العيون..
إلى من بالجنة معه أتمنى أن أكون..

سيدنا محمد صلى الله عليه وسلم

إلى عاصمةِ الصبرِ وأيوبِ هذا العصر..
إلى مُلهمي وناصحي في كل أمر..
إلى من علّمني الثبات في مَرِّ المحن..

أبي الغالي

إلى صاحبةِ القلبِ الدافئِ..
إلى نبعِ الحنانِ ومفتاحِ الجنان..
إلى من أنهلُ منها الصبر والأمان..
إلى من أرجو بها من ربيّ الغفران..

والدتي الغالية

إلى النجوم التي أنارت ظلمة وحدتي..
إلى من بالتبسم أطلقوا ضحكتي..
إلى الذين حين أرى ابتسامتهم تكونُ سعادتي..
إلى من بهم يشتد أزري وبهم تنجلي وحدتي..

إخوتي وأخواتي

إلى من أمضيتُ معهم أحلى لحظاتِ الحياة..
إلى من لي معهم أجمل الذكريات..
إلى من ذكرهم في القلبِ موجود..
إلى من قدرهم عندي ليس له حدود..

أصدقائي

الفهرس

الفهرس المُوسَع

مقدمة

" العلم مغرس كل فخر فافتخر ... واحذر يفوتك فخر ذاك المغرس
واعلم بأن العلم ليس يناله ... من همه في مطعم أو ملبس"

الإمام الشافعي

الرؤية الحاسوبية Computer Vision

ما هي الرؤية الحاسوبية

هي إحدى مجالات علم الحاسوب التي تتضمن طرق تحصيل ومعالجة وتحليل وفهم الصور، وتهدف الرؤية الحاسوبية إلى بناء تطبيقات قادرة على فهم محتوى الصور كما يفهمها الإنسان بهدف الاستفادة منها في تطبيقات مختلفة وتعتبر أيضاً إحدى المجالات التي أحدثت ثورة في التطبيقات الهندسية الحديثة الروبوتية منها والصناعية. يمكن للصور الرقمية أن تأخذ عدة أشكال مثل صور متتابعة (فيديو) ، مشاهد من عدة كاميرات أو صورة متعددة الابعاد من الماسحات الطبية.

تطبيقات الرؤية الحاسوبية

١. تطبيقات الواقع الافتراضي
٢. الاستخدامات الطبية وتحليل الصور الطبية وتشخيص بعض الأمراض .
٣. عمليات التحكم الصناعية والروبوتات الصناعية.
٤. عمليات الملاحة كما في العربات ذاتية الحركة والروبوت النقال(mobile robot)
٥. .كشف الأحداث والاشخاص مثل المراقبة المرئية والتعرف على الاشخاص.
٦. تصنيف المعلومات والمعطيات وفهرسة قواعد البيانات للصور والفيديو.

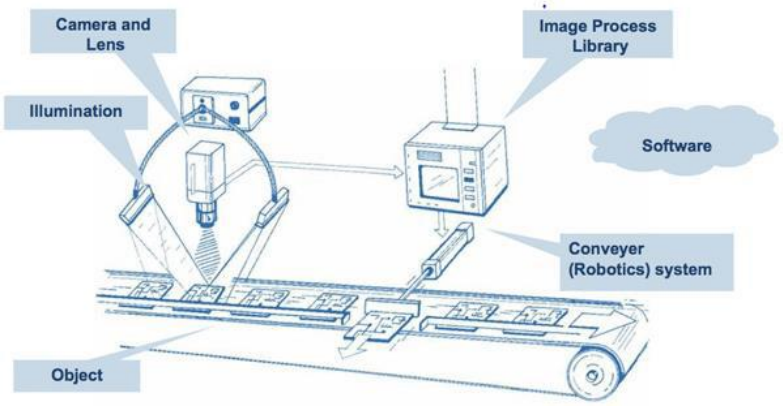
واحدة من أهم تطبيقات الرؤية الحاسوبية هي الرؤية الحاسوبية الطبية أو معالجة الصور الطبية وهذا المجال يعتمد على استخلاص المعلومات من الصور الرقمية بغية الوصول إلى تشخيص طبي للمريض .بشكل عام تكون المعلومات على شكل صور مايكروسكوبية أو صور شعاعية(x-rays) ، أو صور شعاعية للأوعية (angiography images) أو صور فوق صوتية .

كمثال عن المعلومات الي يمكن استخلاصها من الصور الرقمية الطبية هي الأورام أو تصلب الشرايين أو التغيرات الخبيثة الأخرى .ويمكن أيضاً أن تستخدم الرؤية الحاسوبية لقياس الأبعاد

للأعضاء، وتدفق الدم .هذا المجال أيضاً يدعم الأبحاث الطبية بتوفير معلومات جديدة مثلاً عن بنية الدماغ وأيضاً عن مستوى العلاج الطبي.

التطبيق الثاني للرؤية الحاسوبية هو في الصناعة .أحياناً تدعى برؤية الآلة (machine vision) حيث تكون المعلومات المستخلصة من الصورة تهدف إلى تطوير العملية الإنتاجية مثلا تحديد مكان واحداثيات واتجاه القطع لكي تقوم ذراع روبوت بإلتقاطها.

كتطبيق آخر للرؤية الحاسيوبية في الصناعة، عملية التحكم في الجودة حيث يتم فحص المنتج النهائي بشكل آلي من قبل الآلة للتأكد من عدم وجود أية أخطاء أو عيوب إنتاجية.



نظام صناعي مؤتمت يعتمد على الرؤية الحاسوبية

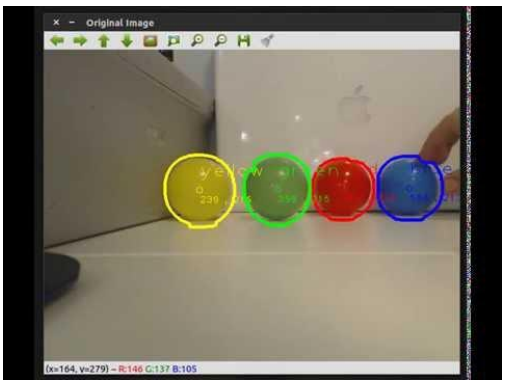
واحدة من أحدث التطبيقات للرؤية الحاسوبية هي المركبات ذاتية الحركة مثل الغواصات أو العربات الصغيرة المزودة بالعجلات(wheeled mobile robot) ، السيارات أو الشاحنات، والطائرات بدون طيار.

عربة ناسا Curiosity على كوكب المريخ تعتمد بشكل أساسي على الرؤية الحاسوبية

المركبات الذاتية الحركية تستخدم الرؤية الحاسوبية في الملاحة، وبناء خرائط للبيئة المحيطة(SLAM)، ومعرفة المكان الحالي للمركبة وتجأوز العقبات (obstacle avoiding).ويمكن أن تستخدم أيضاً لإكتشاف أحداث معينة مثل حرائق في الغابات أو أرض تحتوي نفط أو غاز .إكتشاف الفضاء يتم الآن عن طريق المركبات ذاتية الحركة المدعمة بأنظمة الرؤية الحاسوبية.

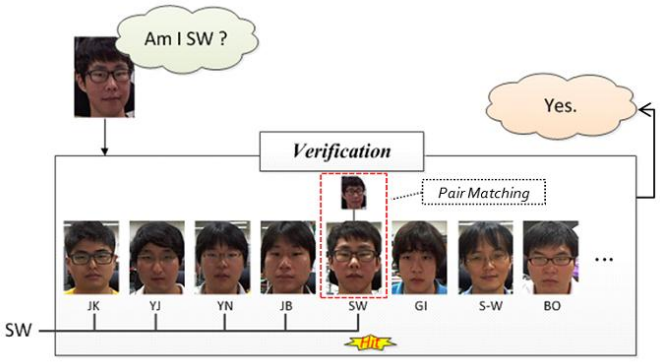
المهام الرئيسية للرؤية الحاسوبية

- **التعرف Recognition**



هي المهمة التقليدية في الرؤية الحاسوبية، وهي القيام بتحديد ما إذا كانت الصورة تحتوي أو لا تحتوي جسماً، معلماً، أو نشاطاً معيناً .هذه المهمة من الممكن حلها ببساطة وبدون أي جهد يذكر بواسطة الإنسان، لكن لا تزال هذه المسألة غير محلولة بشكل فعال ونهائي من قبل الحاسوب في شكلها العام .جميع الطرق الموجودة لحل هذه المسألة تقوم بإيجاد أفضل الحلول من أجل إيجاد أشكال معينة كالأشكال الهندسية، وجوه الأشخاص، الأحرف المطبوعة أو المكتوبة، أو السيارات، وفي حالات معينة فقط محددة على الغالب بظروف إضاءة محددة، خلفية ووضعية معينة للجسم بالنسبة للكاميرا .

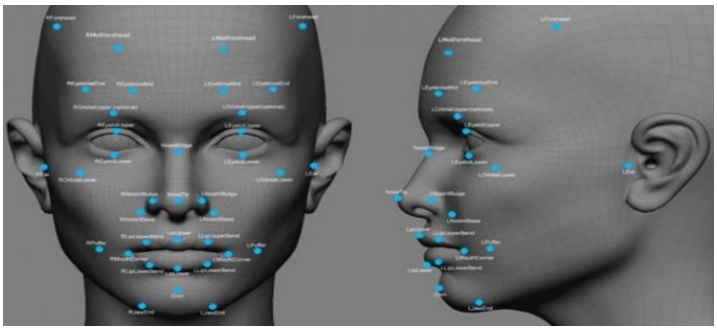
- **التحديد Identification**



تحديد مطابق وحيد للجسم المعرف . مثلاً :تحديد وجه شخص معين أو التعرف على بصمة شخص معين أو سيارة من نوع معين.

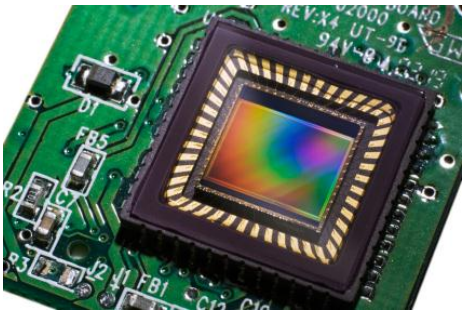
- **التحري Detection**

يتم البحث في بيانات الصورة لإيجاد جسم معين .مثال :تحري وجود خلايا مريضة في صورة طبية، التحري عن وجود سيارة على طريق سريع. كما يوضح الشكل التالي لتحديد ملامح الوجه.



الكاميرا الرقمية Digital Camera

لقد ظهرت الكاميرات الرقمية من حوالي عشر سنوات، وهي تعد الآن من أكثر الأجهزة الالكترونية استخداما في التطبيقات التجارية والصناعية وتعتبر الكاميرا الرقمية جهاز الرؤية لأي نظام .تشبه الكاميرا في هذه الحالات العين البشرية في نظام الرؤية الصناعي، والتي يمكن أن تكون نوع من أنواع الكاميرات ذات الخرج التشابهي أو كاميرات الرقمية، حيث أنه في الكاميرات ذات الخرج التشابهي الضوء يصطدم بشاشة حساسة للضوء وهذه الشاشة تحلل بوساطة حزمة الإلكترون وإن الإشارة الناتجة تستخدم للمعالجة.



في الكاميرات الرقمية إن العناصر المقاومة للضوء المقطعة والموجودة في دارة متكاملة يتم إستخدامها، والإشارة الناتجة من هذه العناصر تستعمل للمعالجة.

مقارنة بين أداتي التصوير ذات الخرج التشابهي والخرج الرقمي

- **مميزات الكاميرا التشابهية**

الكلفة القليلة نسبياً، الإيضاح (التحليل) الأعلى، مستوى تمييز رمادي أفضل.

- **مميزات الكاميرا الرقمية**

تتألف هذه الكاميرا من عدسة وحساس (دارة متكاملة) والذي يتضمن: مكبر الفيديو، مولد النبضات المستخدم من أجل التزامن ودارات مسح منطقية، كلها ضمن نفس الشريحة .هذا يعني أن الكاميرات تزن فقط بضع غرامات ويمكن أن تركب بسهولة على الروبوت .ولكن الكاميرا ذات الخرج التشابهي يمكن ان تزن مئات الغرامات.

معالجة الصورة الرقمية Digital Image Processing

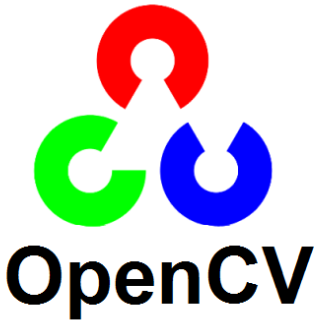
هو أحد فروع علم الرؤية الحاسوبية، تهتم بإجراء عمليات على الصور بهدف تحسنيها طبقاً لمعايير محددة أو استخلاص بعض المعلومات منها .نظام معالجة الصور التقليدي يتألف من ستة مراحل متتالية وهي على الترتيب:

١. **استحصال الصورة (Image acquisition)** بوساطة حساس ضوئي (كاميرا).
٢. **المعالجة المبدئية(pre-processing)** تصفية الصورة من التشويش وتحويلها إلى صورة ثنائية.
٣. **تقطيع الصورة (segmentation)** لفصل المعلومات المهمة في الصورة عن الخلفية.

٤. **استخلاص المميزات (features extraction)** تحديد الصفات الهامة والمميزة في الصورة.
٥. **تصنيف المميزات (classification)** وربطها بالنمط الذي تعود إليه والتعرف على الأنماط.
٦. **فهم الصورة (image understanding)**

وتستخدم نظم معالجة الصورة في الكثير من التطبيقات ولاسيما تطبيقات الرؤية الحاسوبية في التحكم الآلي،والروبوتات.

مكتبة OpenCV



المكتبة البرمجية المفتوحة للرؤية الحاسوبية (OpenCv)هي مكتبة اقترانات برمجية تهدف بشكل أساسي إلى تطوير الرؤية الحاسوبية، قامت بتطويرها شركة إنتل .(Intel) وه مكتبة برمجية مجانية تحت رخصة المصدر المفتوحة(open source BSD license) . يمكن إستخدامها على جميع الأنظمة الحاسوبية، وهي تركز بشكل أساسي على معالجة الصورة بالزمن الحقيق(real time) .

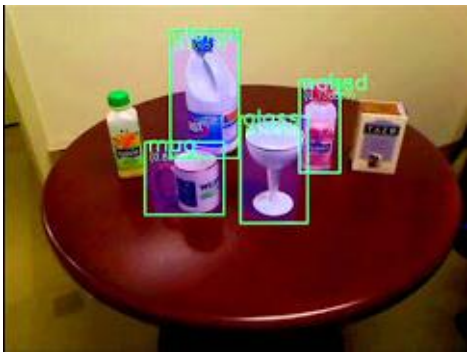
محتوبات مكتبة OpenCv

- **Core:** وحدة نمطية مدمجة لتحديد هياكل البيانات الأساسية، المستخدمة من قبل جميع وحدات أخرى
- **Imgproc:** وحدة معالجة الصور الي تُعنى بعمليات تحسين الصورة كعمليات الترشيح، والتحويلات الهندسية على الصور مثل تغيير حجم الصورة، تغيير تباين وسيطوع الصورة، تعديل الهيستوغرام للصورة والعديد من العمليات.
- **Video:** وحدة تحليل الفيديو الي تتضمن تقديرات الحركة للهدف، الخلفية والبيئة المحيطة، خوارزميات تتبع الكائن أو الهدف.
- **Calib3d:** وحدة تحليل المعلومات للصورة ثنائية الأبعاد والتي هي مأخوذة من الأساس من بيئة ثلاثية الأبعاد وتحتوي على خوارزميات تحديد الخواص المهمة للبيئة ثلاثية الأبعاد.

- **Features2d:** وهي المسؤولة عن كشف الخصائص المعروفة للهدف وفصلها عن باق البيئة المحيطة مثل الأشكال الهندسية مثلاً (مربع، دوائر، مستطيلات.)
- **Objectdetect:** كشف عن وجوه وحالات من الفئات المحددة مسبقا (على سبيل المثال، وجوه، عيون، أكواب، أشخاص، السيارات ..إلخ.)
- **Highgui:** وهي واجهة سيهله الاستخدام لالتقاط الفيديو والصورة ، بالإضافة إلى قدرات أخرى لواجهة المستخدم.
- **CPU module:** تستخدم كامل القدرات الحسابية للنظام عن طريق استخدام قوة بطاقة معالجة الفيديو من أجل تطبيق خوارزميات OpenCV .

تطبيقات OpenCv

- أنظمة التعرف على الوجوه Facial recognition system
- أنظمة التعرف على الإيماءات (Gesture recognition الأوامر عن طريق الإيماءات مثل تحريك كرسي لشخص معاق عن طريق حركة العين.)



- التفاعل بين الكمبيوتر والإنسان HCI.
- الروبوت النقال Mobile Robot.
- التعرف على الهدف Object Identification.
- ملاحقة الحركة Motion Tracking.

لغات البرمجة التي تدعم OpenCv

مكتبة OpenCV مكتوبة أساساً بلغة C++ ، لكنها تدعم معظم لغات البرمجة بشكل كامل مثل :

- **Python**
- **Java**
- **Matlab/Octave**
- **C#**
- **Perl**
- **Ch**
- **Ruby**

تعلم **OpenCv** بلغة البايثون (**Python**) أسهل بكثير مقارنة باللغات الأخرى. لذلك سأشرح هذا الكتاب اعتمادا على هذه اللغة. لا تخف من هذه اللغة فهي لغة بسيطة جدا، وأبسط كثيرا من لغة السي فقد تعلمتها أثناء تعلم لغة **OpenCv**. سأقوم بشرح أساسيات هذه اللغة حتى تنطلق لتعلم **OpenCv**.

أساسيات معالجة الصورة Image Processing

ما هي الصورة من وجهه نظر الحاسب الآلي؟

الصورة هي عباره عن مصفوفة ابعادها تمثل ابعاد الصورة الحقيقية (بالبيكسل) فإذا كانت ابعاد الصورة ٦٠٠*٤٨٠ بيكسل إذا ابعاد المصفوفة هي ٦٠٠*٤٨٠.

ما هو البيكسل؟

البيكسل هو جزء من الصورة ويمثل مربع من مربعات الصورة ويحتوي على قيمة معينه تبعا للون الذي يحتويه هذا البيكسل أو هذا المربع من الصورة.

ماذا يعني ان الكاميرا ٨ ميجا بيكسل؟

أي أن مساحة الصورة التي تلتقطها = ٨ ميجا بيكسل = ٨٠٠٠٠٠٠ بيكسل، مثال: إذا كان هنالك كاميرا تلتقط صورة بحجم ٢٠٤٨×١٥٣٦ إذا الناتج سيكون ٣.٢ مليون بيكسل اي ٣.٢ ميجا .إذا كلما زاد عدد البيكسل في الصورة كلما كانت الصورة أوضح وأنقى.

كيف يقرأ الحاسب الصورة؟

ذكرنا سابقا ان الصورة في الحاسب تمثل بمصفوفة ابعادها تساوي ابعاد الصورة.. وعناصرها = عدد البيكسل ... وكل عنصر يحتوي قيمة اللون الذي تحتويه البيكسل...

قيمة اللون في البيكسل

الصورة عندما تلتقط لها أكثر من شكل.. فهنالك صور ابيض واسود فقط.. وهنالك صور Gray اي ابيض واسود ورصاصي.. وهنالك صور ملونه.. إلخ .هنا نأتي لتعريف مهم في ال Image processingوهو number of bits per pixel : اي عدد ال Bit التي تمثل فيها ال بيكسل الواحدة واختصارهاBpp

فإذا كانت الصورة ١ Bpp يعني bit واحده فقط.. فهي تحتمل اما صفر أو واحد.. وهي الصورة الأبيض والأسود فقط.. الأبيض هو الواحد والاسود هو الصفر وإذا كانت ٢ Bpp معناها ان كل بيكسل تمثل في ٢ bit ومعناها ان range الألوان قد زاد وأصبح $4 = 2^2$ ألوان

وهكذا حتى نصل الى الصور الملونة وهي تلتقط في الكاميرا ٣ مرات للصورة الواحدة.. مره تقيس اللون الأحمر ومره تقيس اللون الاخضر ومره تقيس اللون الازرق ... وتسمى ال RGB اي ان الصورة الملونة لها ٣ مصفوفات مصفوفة للأخضر ومصفوفه للأحمر ومصفوفه للأزرق. وهنا البيكسل الواحدة تمثل في ٢٤ bit. ثمانية للأحمر وثمانية للأزرق وثمانية للأخضر.. وهنا يكون عدد احتمالات درجات الاحمر = $8^2 = 255$ وكذلك الاخضر وكذلك الأزرق. إذاً عدد احتمالات الألوان للبيكسل ككل = 24^2= ١٦٧٧٧٢١٦ لون

ملخص

إذا قمت باستدعاء البيكسل رقم (٠,٠) اي أول مربع في الصورة ستجد الناتج ممثل في ٣ قيم (٢٥٥,٢٥٥,٢٥٥) وهي قيمة الR,G,B . إذا كانت ال ٣ قيم قيمتها ٢٥٥ فهذا يمثل اللون الأبيض وإذا كانوا جميعا ٠ إذا هذا هو اللون الأسود

ملاحظة: إذا كان هنالك لون أصفر مثلا فإنه يمكن ان يكون له أكثر من قيم في ال **RGB** بمعنى انه يمكن استنتاجه بأكثر من طريقة (تقليل الازرق وزيادة الاحمر والاخضر، زيادة الاخضر والازرق والاحمر، تقليل الاحمر والازرق وزيادة الاخضر .... إلخ)

ما هو الفيديو أو البث الحي ؟؟

البث الحي أو الفيديو ما هو إلا عباره عن التقاط عدد كبير من الصور في الثانية الواحدة وعرضها بالترتيب.. وتسمى number of frames per second وهنا تسمى الصورة فريم ..Frameوكلما زاد عدد ال frames في الثانية الواحدة كلما كان الفيديو مزامن للحقيقة..

مثال: إذا كانت الكاميرا ١٠ frame per second وكانت تصور مبارة كرة قدم ستجد فرق كبير جدا في الوقت ستجد الكره تمر في الحقيقية ثم بعدها بثواني ستجدها في الكاميرا.. وستلاحظ هذا في الموبايلات القديمة أنك تحرك يدك بسرعه والكاميرا تبثها بعدها بفرق وقتي. لذلك تجد في الكاميرات وضع يسمى Sport الذي يزيد من عدد ال fbs حتى تجاري الواقع.

ما هي معالجة الصور؟؟ وما هي العمليات الرياضية التي تجرى عليها؟

بما ان الصور تمثل في مصفوفات إذا يمكننا اجراء عليها كل العمليات الرياضية التي تجرى على المصفوفات: بداية من الجمع والطرح حتى المعادلات والتحويلات (transformations) والمقلوب وغيرها

ماذا نستفيد من ذلك؟

- **الجمع:** إذا جمعت على صورة ما ٣ مصفوفات كلها تحتوي على عنصر = ١ فستجمع على مصفوفة ال red 1 وكذلك الاخضر وكذلك الازرق ... فإذا زادت قيمة الاحمر والاخضر والازرق فإنك تقترب من الابيض قليلا فستجد ان الصورة اصبحت أكثر إضاءة

- **الطرح:** إذا التقطت صورتين لنفس الجسم في نفس الظروف وطرحت الصورتين من بعضهم ستحصل على صورة جديده تحتوي الفروقات بينهم (بمعنى أنك إذا التقطت صورة

لجسم مره ثم وضعت عليه جسم اخر والتقط مره اخرى وطرحت الصورتين ستجد الناتج عباره عن صورة سوداء لا تحتوي إلا الجسم الذي وضعته مؤخرا

- **التحويلات:** كثيرا ما نسمع عن التشويش أو الاهتزاز (البكسلة) وهي عباره عن صورة تم التقاطها بسرعه فأصبحت الصورة غير واضحة وهنا يتم اجراء عليها transformations تقوم بعمل Filter للتشويش واعادة الصورة واضحة مثلFourier transformation ، geometric transformation.

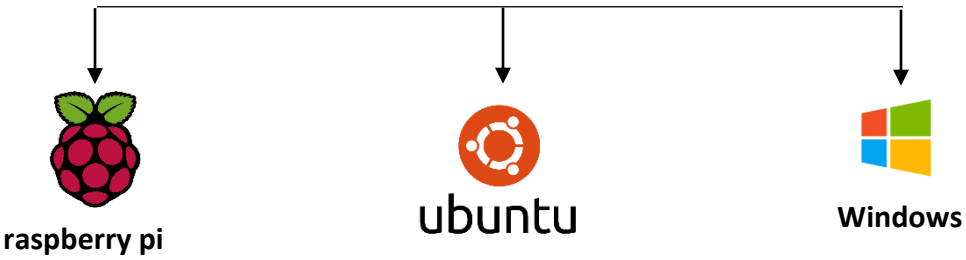
ما هي مكتبة الOpenCv ؟

كما وضحنا سابقا ان كل العمليات على الصور هي مجرد عمليات رياضية بحته.. ولكن ليس من المعقول ان المبرمجين سيقومون بعمل كل هذه العمليات في كل مره يستخدمون فيها الصور ولذلك تم انشاء مكتبه الOpenCv التي هي عباره عن مكتبه تحتوي على دوال تقوم بعمل معظم عمليات الصور التي يحتاجها المبرمجين.

كما تحتوي على دوال تقوم بتحديد اجسام معينه في الصورة مثل الخطوط والاشكال الهندسية ودوال اخرى لتحديد الألوان، كما يمكن استدعاء ملفات XML خاصه بتحديد وجه الانسان وعينيه وشعره ويده وجسمه ككل.

ويمكن استخدامها في التعرف على الشخص كما يمكن عمل مصنف classifier خاص بأي جسم (التقاط صور عديده لنفس الجسم من كل الاتجاهات حتى تتعرف عليه الكاميرا من كل الاتجاهات) ... وليس هذا مقتصرا على الانسان فقط بل أي جسم (مثال مباريات كرة القدم).

طريقة تثبيت مكتبة OpenCv



١. طريقة تثبيت مكتبة OpenCv على نظام Windows

✔ طريقة ١: (مستحسنة)

مشروحة في الفيديو التالية (انقر على الفيديو وستشاهد الشرح بعد الانتقال لموقع youtube)

https://www.youtube.com/watch?v=BAFSi_xTGiI

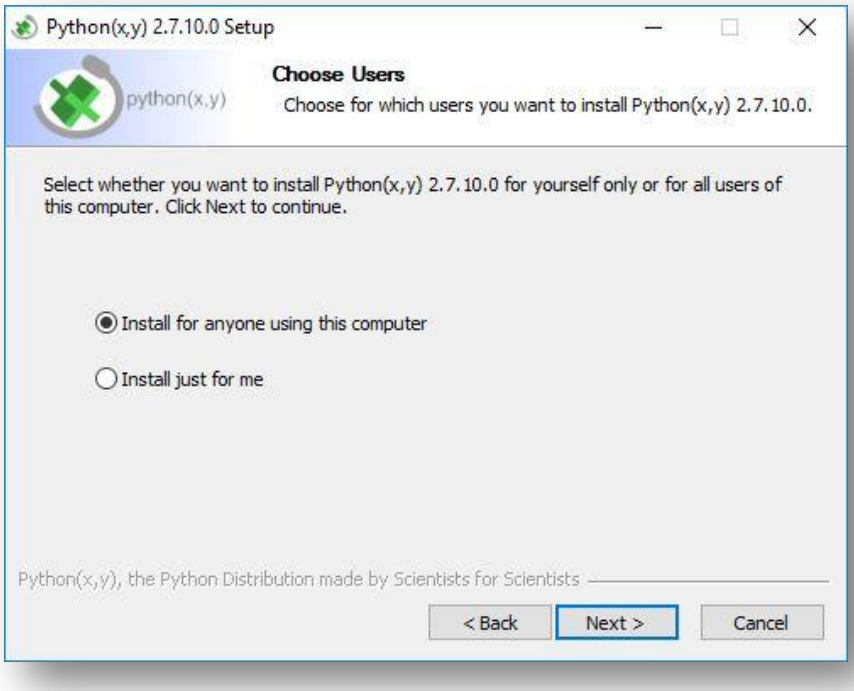
✔ طريقة ٢: (يدوياً)

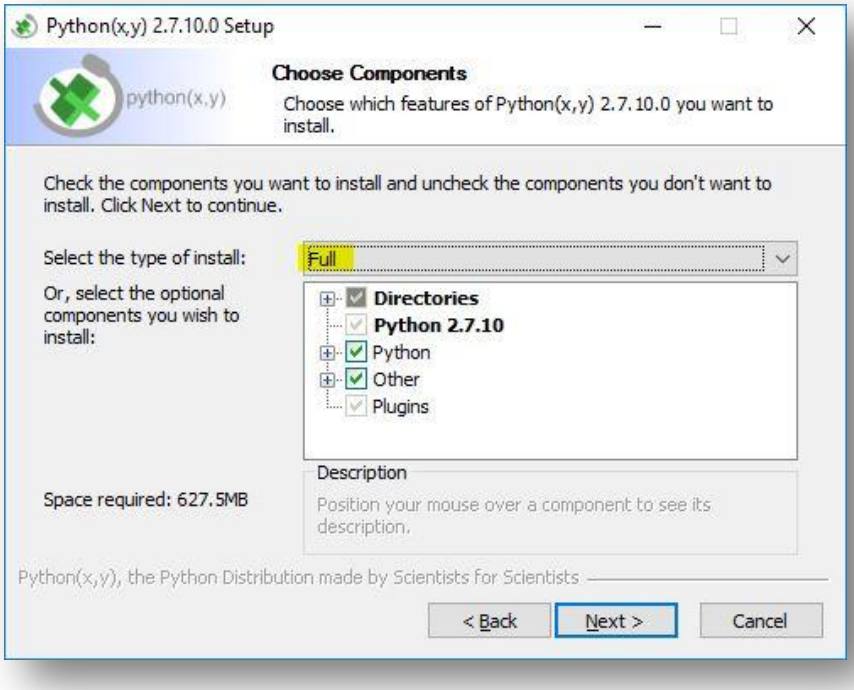
يمكنك تحميل كل ملف على حدى وتنصيبهم واحد تلو الأخر

- لتحميل برنامج python(x,y) https://python-xy.github.io/downloads.html
- حمل الملف التالي الذي يحوي مجموعة إصدارات لمكتبة OpenCv

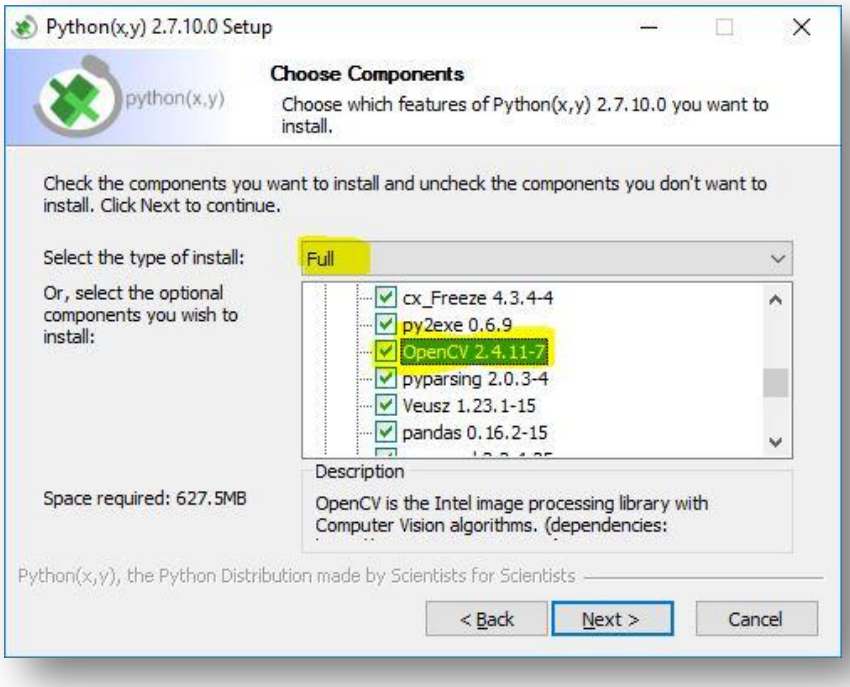
https://drive.google.com/file/d/0B0kFf-FN5r9tS1EzSlppUGItNUU/view?usp=sharing

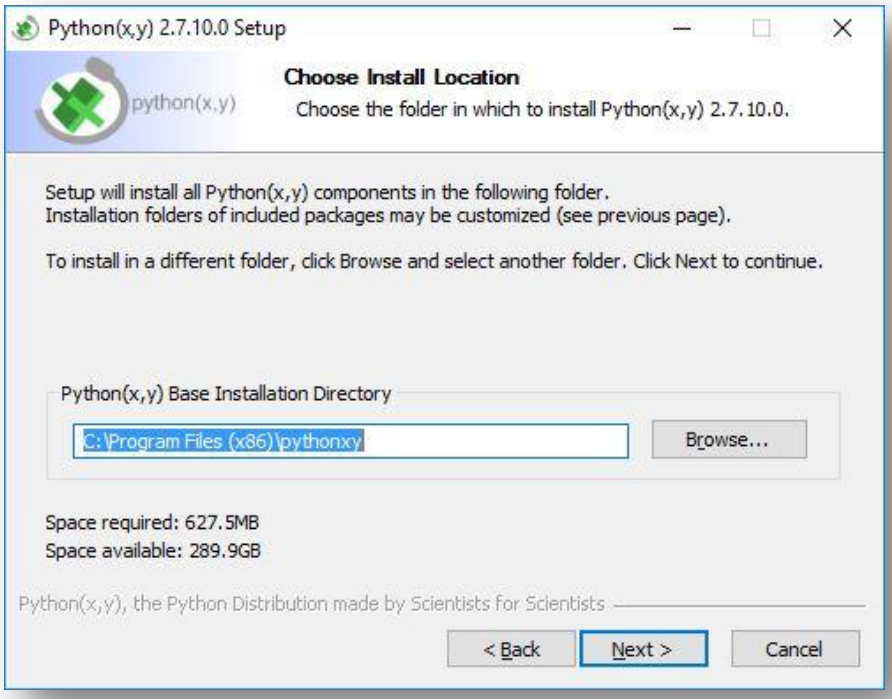
ثم قم بتنصيب برنامج python(x,y) كما هو واضح في الصورة التالية:





تأكد من وجود علامة الصح عند مكتبة **OpenCv** ضمن قائمة **python**


Python(x,y) 2.7.10.0 Setup
python(x,y)
Choose Components
Choose which features of Python(x,y) 2.7.10.0 you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select the type of install:
Full
Or, select the optional components you wish to install:
cx_Freeze 4.3.4-4
py2exe 0.6.9
OpenCV 2.4.11-7
pyparsing 2.0.3-4
Veusz 1.23.1-15
pandas 0.16.2-15
Description
OpenCV is the Intel image processing library with Computer Vision algorithms. (dependencies:
Space required: 627.5MB
Python(x,y), the Python Distribution made by Scientists for Scientists
< Back
Next >
Cancel


Python(x,y) 2.7.10.0 Setup
python(x,y)
Choose Install Location
Choose the folder in which to install Python(x,y) 2.7.10.0.
Setup will install all Python(x,y) components in the following folder.
Installation folders of included packages may be customized (see previous page).
To install in a different folder, click Browse and select another folder. Click Next to continue.
Python(x,y) Base Installation Directory
C:\Program Files (x86)\pythonxy
Browse...
Space required: 627.5MB
Space available: 289.9GB
Python(x,y), the Python Distribution made by Scientists for Scientists
< Back
Next >
Cancel

٢. طريقة تثبيت مكتبة OpenCv على نظام التشغيل Ubuntu

يمكنك تنصيب openCV على أنظمة لينكس مثل Debian – Ubuntu – mint بطريقتين مختلفتين، الأولى من خلال تنصيب الملفات الجاهزة pre-compiled من مستودعات البرامج مثلاً: في حالة نظام أوبنتو يكفي أن تقوم بتنفيذ الأمر التالي:

```
$ sudo apt-get install libopencv-dev python-opencv
```

الطريقة الثانية هي عن طريق عمل Compile للملفات المصدرية من البداية (وتضمن هذه الطريقة الحصول على أحدث نسخة من المكتبة.

افتح سطر الأوامر Terminal وقم بكتابة التعليمات التالية:

تحديث حزم (package) النظام.

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

تثبيت أدوات المطور

```
$ sudo apt-get install build-essential cmake git pkg-config
```

مكتبة OpenCv تحتاج الحزم التالية لتستطيع قراءة أنواع الصور المختلفة كـ JPEG, PNG, TIFF

```
$ sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev libpng12-dev
```

تنصيب المكتبات التي سنستخدمها لتحسن أداء مكتبة OpenCv.

```
$ sudo apt-get install libatlas-base-dev gfortran
```

تنصيب pip المسؤول عن إدارة حزم python وتنصيب المكتبات التي نريدها.

```
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python get-pip.py
```

```
$ sudo pip install virtualenv virtualenvwrapper
$ sudo rm -rf ~/.cache/pip
```

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
```

```
$ source ~/.bashrc
```

```
$ mkvirtualenv cv
```

تنصيب python 2.7

```
$ sudo apt-get install python2.7-dev
```

تنصيب مكتبة Numpy داخل python

```
$ pip install numpy
```

تنزيل مكتبة OpenCv

```
$ cd ~
$ git clone https://github.com/Itseez/opencv.git
$ cd opencv
$ git checkout 3.0.0
```

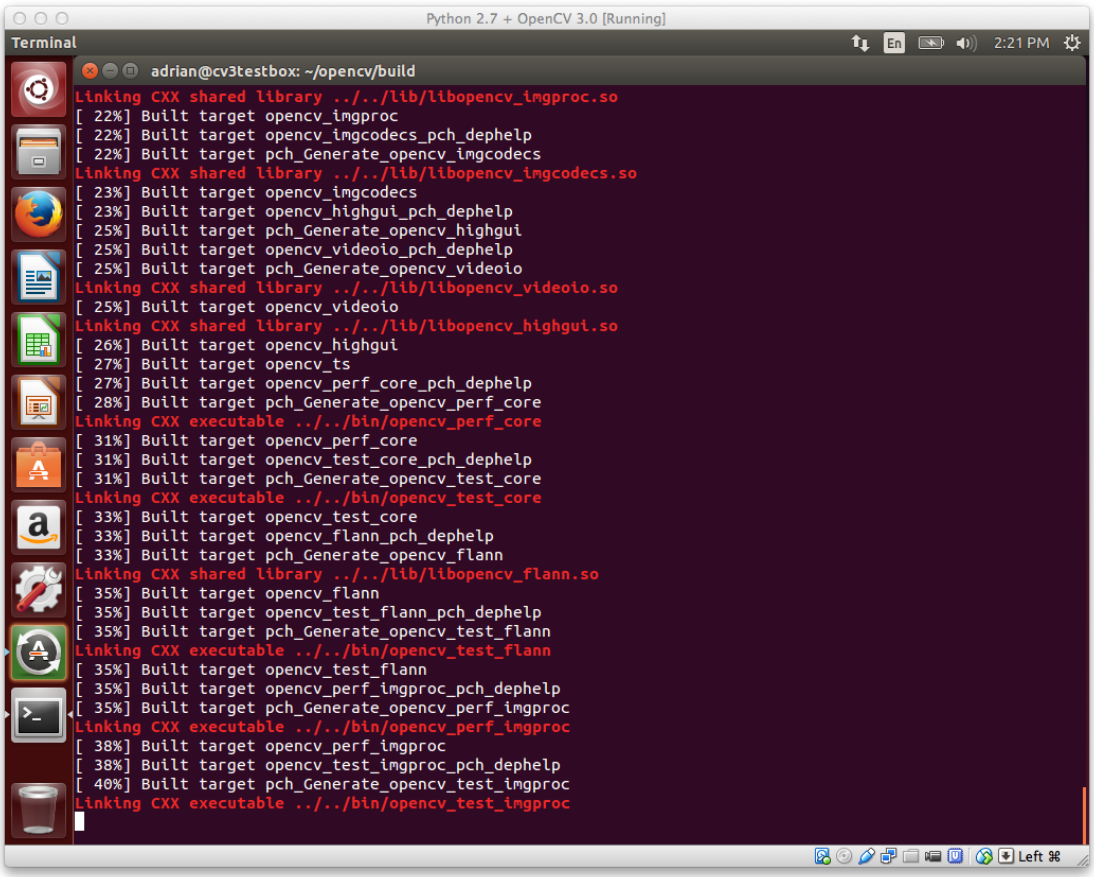
تنزيل opencv_contrib

سنستخدمه حتى نستطيع استخدام بعض التوابع مثل (SIFT, SURF, etc.) التي كانت موجودة في الإصدار 2.4.X لمكتبة OpenCv ثم تم حذفها في الإصدار الجديد OpenCV 3.0.

```
$ cd ~
$ git clone https://github.com/Itseez/opencv_contrib.git
$ cd opencv_contrib
$ git checkout 3.0.0
```

```
$ cd ~/opencv
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
        -D CMAKE_INSTALL_PREFIX=/usr/local \
        -D INSTALL_C_EXAMPLES=ON \
        -D INSTALL_PYTHON_EXAMPLES=ON \
        -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
        -D BUILD_EXAMPLES=ON ..
```

```
$ make
```

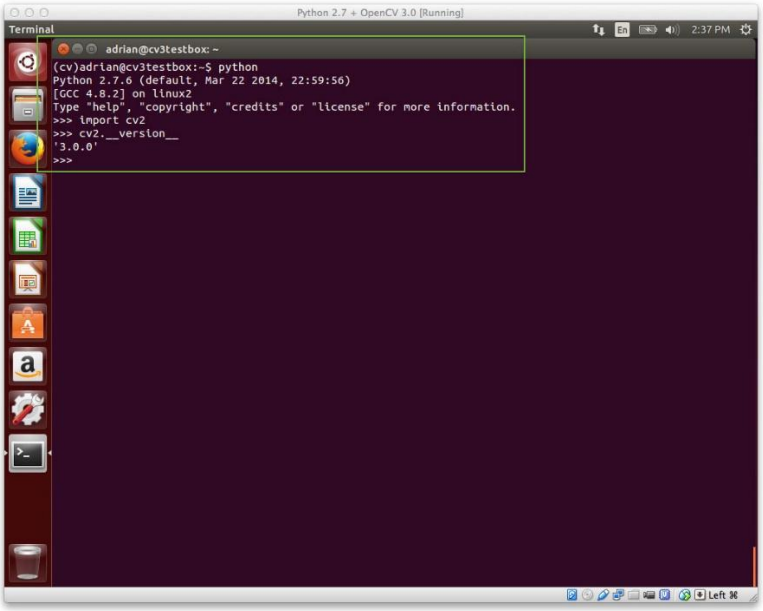


```
$ sudo make install
$ sudo ldconfig
```

```
$ cd ~/.virtualenvs/cv/lib/python2.7/site-packages/
$ ln -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so
```

الآن لنختبر هل قمنا بإعداد مكتبة OpenCv بشكل صحيح.

```
$ workon cv
$ python
>>> import cv2
>>> cv2.__version__
'3.0.0'
```

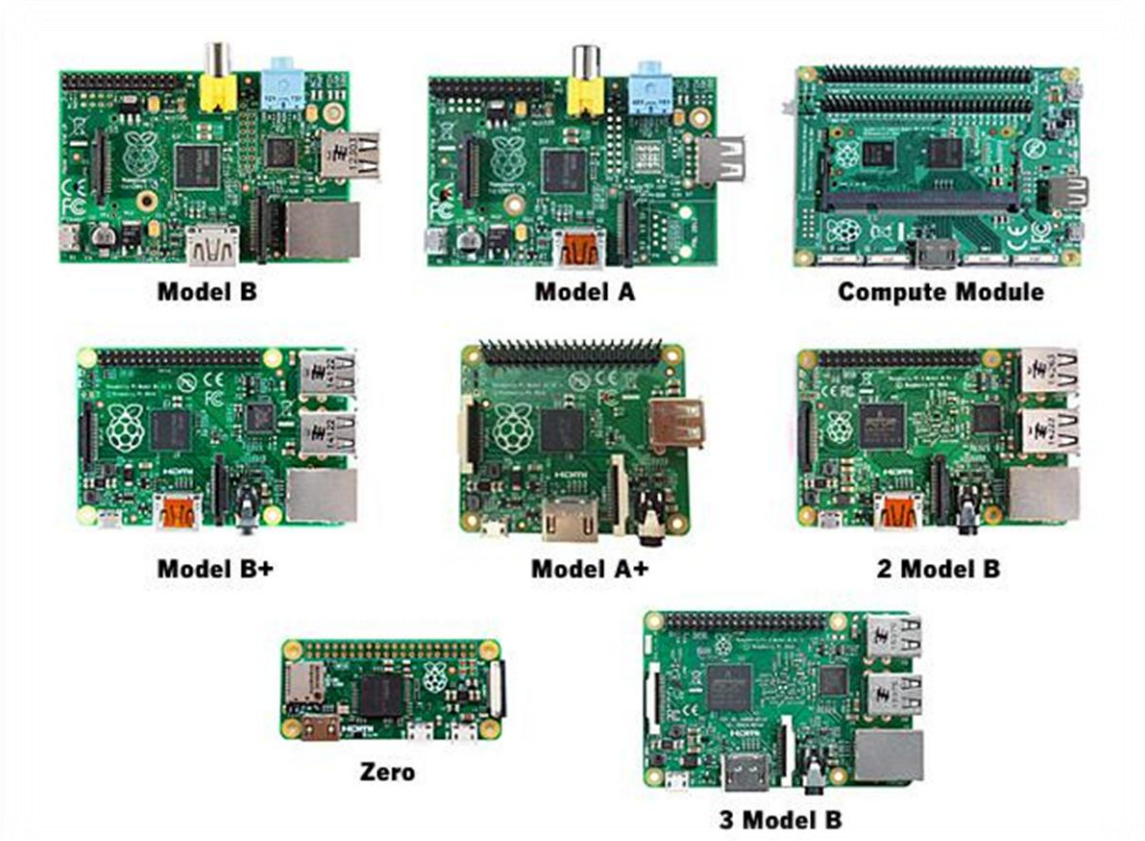


المصدر:

/http://www.pyimagesearch.com/2015/06/22/install-opencv-3-0-and-python-2-7-on-ubuntu

٣. طريقة تثبيت مكتبة OpenCv على الراسبيري باي:

يمكنك باستخدام الحاسب المصغر الراسبيري باي لمعالجة الصور دون الحاجة للكمبيوتر فسرعة معالجتها للصور سريعة بما يكفي (بل ويمكنها معالجة الفيديو أيضاً). لمن لا يعرف ما هي الراسبيري باي فانصحك بقراءة الكتاب المميز راسبيري باي ببساطة للمهندس عبد الله علي والذي يشرح فيه فائدتها وطريقة استعمالها بشكل مفصل ومبسط.



أيضاً يمكنك استعمال أي لوحة تعمل بنظام لينكس مثل **BegalBone** أو **OrangePi** أو **NanoPi**

لتنزيل مكتبة OpenCv على الراسبيري باي اتبع الخطوات الموجودة على الرابط التالي:

www.pyimagesearch.com

http://www.pyimagesearch.com/2015/10/26/how-to-install-opencv-3-on-raspbian-jessie/

مقدمة عن لغة بايثون

ظلت كلمة البايثون "الأصلة" تعبر عن ثعبان ضخم يعيش في انهار الأمازون وبعض مستنقعات افريقيا لكن ماإن اتى عام ١٩٩١ حتى اكتسب هذا الاسم شهره جديدة بين المبرمجين فأصبحت كلمة البايثون تعبر عن أشهر لغات البرمجة مفتوحة المصدر في العالم والتي تعتبر من لغات المستوى العالي وتتميز ببساطة كتابتها وقراءتها مقارنة بباقي اللغات.

تعتبر لغة بايثون لغة تفسيرية، متعددة الأغراض وتستخدم بشكل واسع في العديد من المجالات، كبناء البرامج المستقلة باستخدام الواجهات الرسومية GUI وفي عمل برامج الويب، بالإضافة إلى استخدامها كلغة برمجة نصية للتحكم في أداء بعض من أشهر البرامج المعروفة أو في بناء برامج ملحقة لها، كما تدعم البرمجة المتوازية وبرمجة الحواسيب الخارقة Supercmputers (cluster parrallel programming) –كما تدعم برمجة منافذ التحكم GPIO في لوحة الراسبيري باي.

بشكل عام يمكن استخدام بايثون لبرمجة البرامج البسيطة للمبتدئين، ولإنجاز المشاريع الضخمة كأي لغة برمجية أخرى في نفس الوقت. غالباً ما يُنصح المبتدئين في ميدان البرمجة بتعلم هذه اللغة لسهولتها وقوتها في ذات الوقت، ومع ذلك نجد مؤسسات عملاقة تستخدم هذه اللغة داخل برامجها مثل جوجل و مؤسسة الفضاء الدولي "ناساNASA".

غالباً ما تحصل هذه اللغة على الترتيب الرابع أو الخامس في قائمة أشهر لغات البرمجة في العالم (تبعاً لأحصائيات موقع GitHUB الشهير لمشاركة الأكواد البرمجية)، نشأت بايثون في مركز CWIمركز العلوم والحاسب الآلي بأمستردام على يد "جويدو فان رُزوم" Guido van Rossum في أواخر الثمانينات، وكان أول إعلان عنها في عام ١٩٩١، تم كتابة نواة اللغة بلغة السي C أطلق فان رُزوم اسم "بايثون" على لغته تعبيرًا عن إعجابه بفِرقَة مسرحية هزلية شهيرة من بريطانيا، كانت تطلق على نفسها اسم مونتي بايثون Monty Python

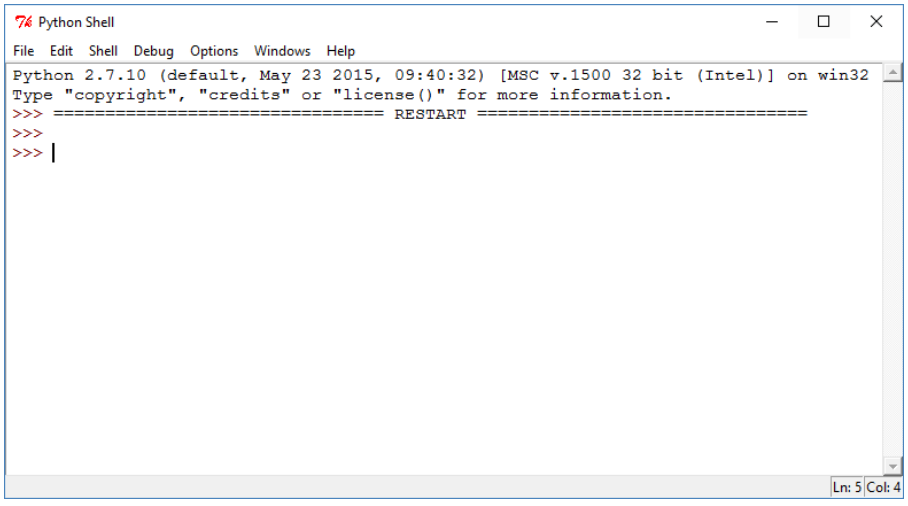
مميزات لغة بايثون

- ✔ تعمل على جميع أنظمة التشغيل واصدارتها المختلفة: ويندوز – لينكس (ومشتقاته) – يونكس (ومشتقاته) – أنظمة الهواتف المحمولة مثل Symbian و Android
- ✔ وجود أغلب المكتبات الإضافية معها فتسطيع في بايثون إيجاد مكتبة لكل شيء وأغلب هذه المكتبات تأتي مرفقة مع اللغة، لكن هناك قليل من المكتبات التي تحتاج الى تحميلها من مصادر خارجية ومن الأمثلة على هذه المكتبات: البلوتوث، منافذ التحكم الإلكتروني، واجهات الويب، التعامل مع الشبكة و الإنترنت، برمجة الحواسيب الفائقة، تطبيقات سطح المكتب، مكتبات لتصميم الألعاب ثنائية وثلاثية الأبعاد .... إلخ.
- ✔ التكامل مع C++ و Java
- ✔ تعمل ضمن بيئة تفاعلية أو عبر سكربتات (ملفات) مكتوبة
- ✔ التعامل مع قواعد البيانات التالية
- ✔ Oracle, sybase , PostGres, mSQL , persistence , dbm

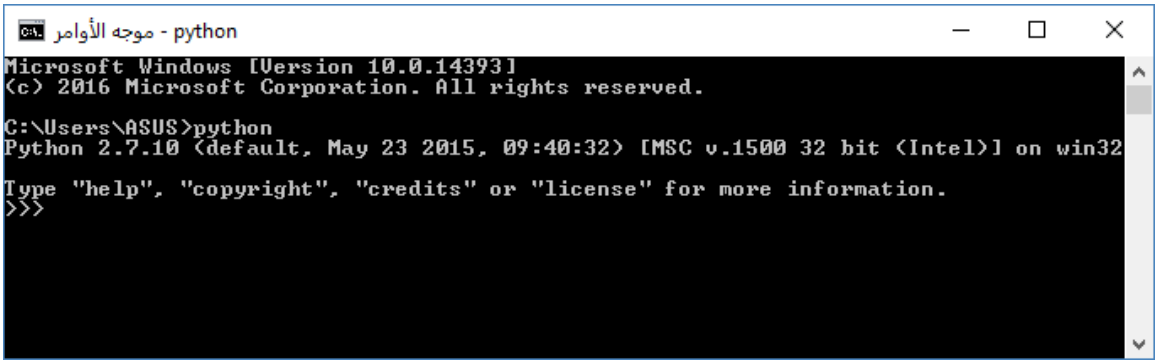
أساسيات لغة python

مفسر بايثون التفاعلي

تتماز لغة بايثون بإمكانية عمل برامج عن طريق كتابتها في ملف (سكربت) أو تشغيلها مباشرة ومشاهده النتائج فور كتابتها عن طريق مفسر بايثون التفاعلي والذي يمكنك تشغيله من سطر الأوامر مباشرة عبر كتابة python أو يمكنك تشغيله بالضغط مرتين على أيقونة IDLE الموجودة في مكان تنصيب python.

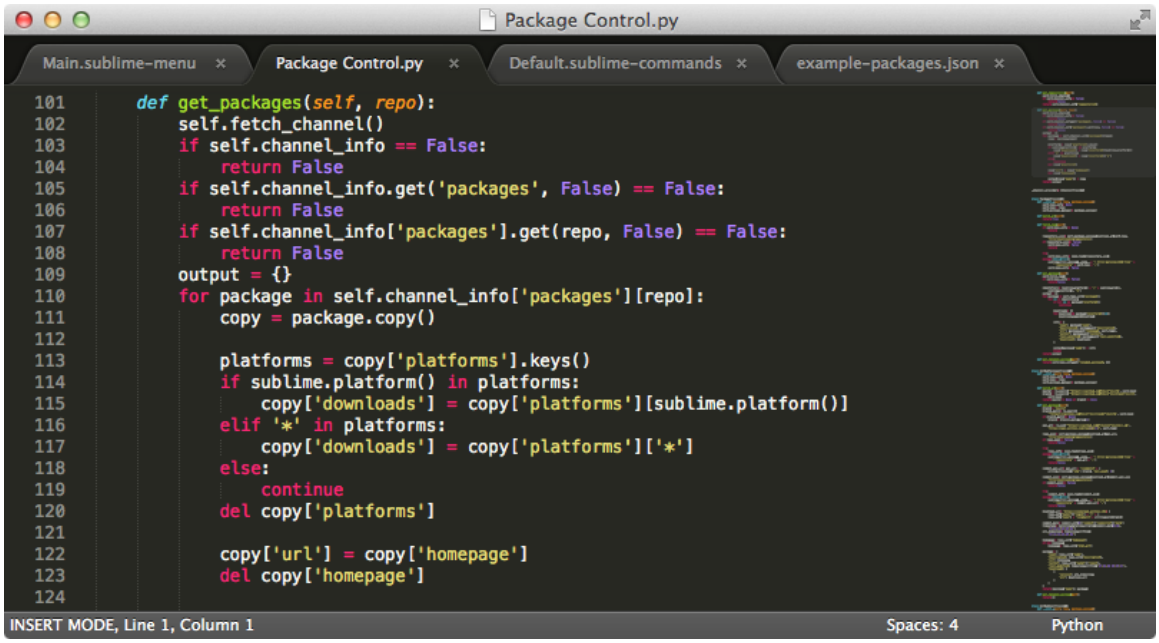


شخصياً أفضل التعامل مع مفسر البايثون من سطر الأوامر . فقط افتح سطر الأوامر واكتب كلمة python كما في الصورة التالية.



ايضاً يمكنك استخدام محرر نصوص متطور مثل Sublimeأو ATOM ليساعدك على كتابة برامج بايثون بسهولة واحترافية

https://www.sublimetext.com/



العمليات الأساسية

سنكتب أول برنامج لعرض جملة "مرحباً أيها العالم" Hello World وذلك عن طريق كتابة الأمر print ثم الجملة المراد عرضها مثل الصورة التالية:

```
print " Hello World "
```

سنجد أن مفسر بايثون التفاعلي قد اظهر النتيجة فوراً بمجرد الضغط على زر Enter

```
>>> print "Hello World"
Hello World
>>>
```

العمليات الحسابية:

تستطيع البايثون القيام بالعمليات الحسابية مباشرة مثل الجمع، الطرح، القسمة، الضرب.

اكتب 1+1 ثم اضغط Enter

```
>>> 1+1
2
>>>
```

اكتب 2*2-1 (ضرب ٢ في ٢ ثم طرح ١)

```
>>> 2*2-1
3
>>>
```

تعريف المتغيرات:

لتعريف أي متغير رقمي في لغة البايثون كل ما عليك فعلة هو كتابة اسم (المتغير = قيمته)

ولعرض قيمة المتغير نكتب print ثم اسم المتغير

```
x = 2
print x
```

```
>>> x = 2
>>> print x
2
>>>
```

لتغير المتغيرات النصية (كلمة أو جملة) نكتب اسم المتغير ثم = "الكلام الذي يحتويه المتغير" – لا تنسى علامات " " بين قيمة المتغير مثل:

```
myName = "Khaled"
print myName
```

```
>>> myName = "Khaled"
>>> print myName
Khaled
>>>
```

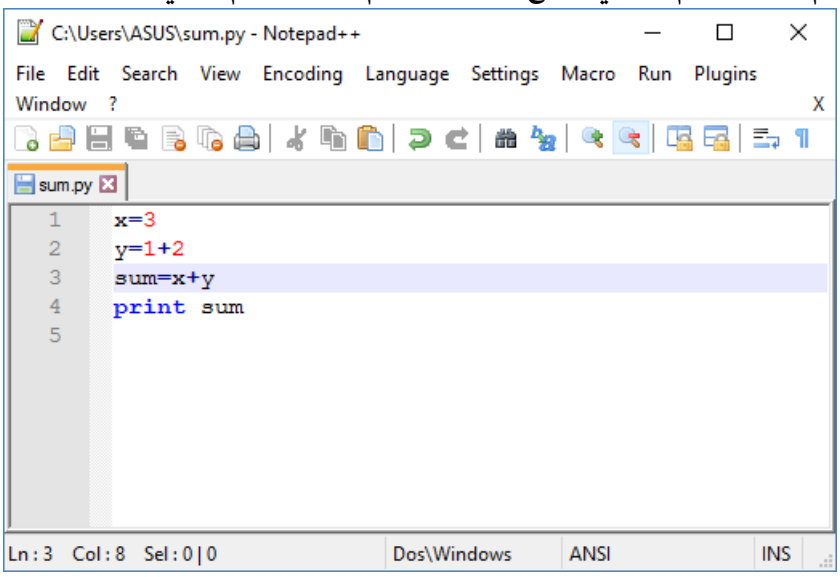
لإغلاق المفسر التفاعلي من سطر الأوامر نضغط على زر **Ctrl+D**

حفظ البرنامج في سكربت

في الأمثلة السابقة استخدمنا البايثون في تنفيذ الأوامر مباشرة عبر المفسر التفاعلي، لكن بالتأكيد عندما نبني مشاريع حقيقة فسنحتاج لكتابة برامج ثابتة لا تضيع منا بمجرد غلق المفسر ولعمل هذا يمكننا استخدام أي محرر نصوص وكتابة نفس الأوامر السابقة وحفظها على صورة سكربت (ملف نصي) بامتداد py وهو امتداد جميع برامج البايثون. (يمكنك استخدام أي محرر نصوص مثل Sublime – ATOM – Notepad++)

افتح برنامج محرر النصوص ، ثم اكتب النص و احفظ الملف باسم sum.py داخل المجلد C:\Users\ASUS.

(ASUS هو اسم المستخدم عندي ، ضع بدلا منه اسم المستخدم الذي لديك)



برنامج لجمع رقمين

```
x=3
y=1+2
sum=x+y
print sum
```

لتشغيل البرنامج افتح سطر الأوامر واكتب

```
python sum.py
```

```
C:\Users\ASUS>python sum.py
6

C:\Users\ASUS>
```

برنامج لتعريف تابع واستدعائه

لتعريف تابع نكتب الأمر def ثم نكتب اسم التابع ثم نفتح قوسين ونمرر لهما المتغيرات المطلوبة ثم نكتب نقطتين (:) ونترك فاصل ثم نكتب الأوامر التي سينفذها التابع.

```
def sum():
    x=3
    y=1+2
    sum=x+y
    print sum

sum()
```

برنامج لتعريف مصفوفة

سنحتاج للمصفوفات لأننا سنتعامل مع الصور والصورة عبارة عن مصفوفة

سنقوم بتعريف مصفوفة اسمها arrayوتحتوي على العناصر [2,4,7,9]. ثم قمنا باستدعاء العنصر الذي ترتيبه ٢ في المصفوفة.

```
>>> array = [2, 4, 7, 9]
>>> array[2]
7
```

برنامج لتعريف مصفوفة بالاستعانة بمكتبة Numpy.

أولا قمنا باستيراد مكتبة Numpy ثم عرفنا مصفوفة اسمها arrayوتحتوي على العناصر [2,4,7,9]. ثم قمنا باستدعاء العنصر الذي ترتيبه ٣ في المصفوفة.

```
>>> import numpy
>>> array = np.array([2,4,7,9])
>>> array[3]
9
>>>
```

الفصل الأول

" هؤلاء الذين يمتلكون الجنون الكافي ليؤمنوا أن
بإمكانهم تغيير العالم، هم من يغيرونه بالفعل "

ستيف جوبز – شريك مؤسس لشركة Apple

الفصل الأول: ميزات الواجهة الرسومية GUI في OpenCv

❖ **البدء في التعامل مع الصور**

تعلم تحميل الصورة وعرضها وحفظها.

❖ **البدء في التعامل مع الفيديو**

تعلم تشغيل فيديو، التقاط فيديو من الكاميرا، وحفظ الفيديو الملتقط.

❖ **ميزات الرسم في OpenCv**

تعلم رسم خط، مستطيل، دائرة والعديد من الأشكال مع توابع مكتبة OpenCv.

❖ **استخدام الفأرة كفرشاة للرسم**

رسم أي رسمة باستخدام الفأرة.

استخدام شريط التمرير الجانبي(Track bar) كلوحة ألوان

تعلم إنشاء شريط جانبي للتحكم ببعض البارامترات.

طريقة عرض صورة وحفظها

الهدف:

تعلم قراءة الصورة وعرضها وحفظها عن طريق التوابع التالية :

cv2.imread(), cv2.imshow(), cv2.imwrite()

تحسين طريقة عرض الصورة عن طريق استيراد مكتبةMatplotlib .

الصورة الأصلية

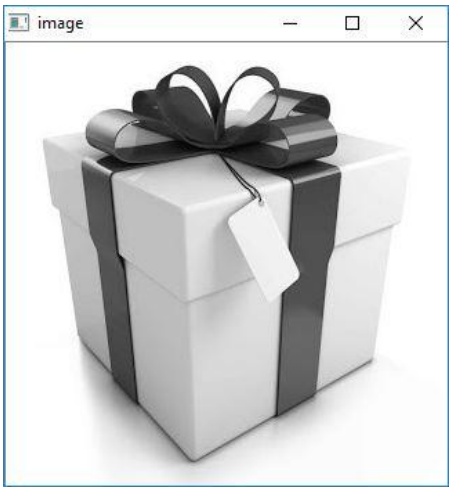
- **قراءة وعرض الصورة:**

البرنامج التالي سيقوم بتحميل الصورة السابقة وحفظها بالمستوى الرمادي ومن ثم سيقوم بعرضها

الكود:

```
import numpy as np
import cv2

img = cv2.imread('gift.jpg',0)

cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:



شرح الكود:

```
import numpy as np
import cv2
```

في البداية قمنا باستدعاء مكتبة OpenCv ومكتبة Numpy (دائما سنقوم باستدعائهما)

```
img = cv2.imread('gift.jpg',0)
```

قمنا بقراءة الصورة عن طريق الأمر cv2.imread() وخزناها في متغير سميناه img

شرح التابع imread():

المتغير الأول هو اسم الصورة (يجب وضع مسار الصورة كامل أو وضعها ضمن مجلد العمل)

والمتغير الثاني هو مؤشر يحدد كيفية قراءة الصورة ويأخذ ثلاث قيم:

- cv2.IMREAD_COLOR معناها أقرأ الصورة بالألوان الكاملة بدون شفافية (يكافئ رقم ١)
- cv2.IMREAD_GRAYSCALE معناها أقرأ الصورة باللون الرمادي (يكافئ رقم ٠)
- cv2.IMREAD_UNCHANGED معناها أقرأ الصورة بالألوان الكاملة مع شفافية (يكافئ رقم -١)

في حالة عدم وضع قيمة في المتغير الثاني تكون القيمة الافتراضية ١

```
cv2.imshow('image',img)
```

قمنا بعرض الصورة عن طريق الأمر cv2.imshow

شرح التابع imshow ():

المتغير الأول هو أسم النافذة التي سنعرض عليها الصورة

المتغير الثاني هو تابع قراءة الصورة

```
cv2.waitKey(0)
```

ثم أخبرنا البرنامج أن ينتظر مدة زمنية معينة وبعد انهاء هذه المدة يقوم البرنامج بتنفيذ الأمر الذي بعده.نضع قيمة الزمن المطلوب انتظاره ضمن الأقواس (الزمن يحسب بالميلي ثانية)

وإذا وضعنا رقم ٠ سينتظر البرنامج حتى يتم ضغط أي مفتاح من لوحة المفاتيح.

```
cv2.destroyAllWindows()
```

تابع إغلاق (تدمير) جميع النوافذ التي فتحناها عن طريق هذا البرنامج.

ملاحظة: إذا أردنا تحجيم حجم الصورة لما يناسب الشاشة فعلينا استخدام الأمر **cv2.namedWindow** مع العلم أن له خيارين:

- **cv2.WINDOW_AUTOSIZE**
- **cv2.WINDOW_NORMAL**

```
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

٢-حفظ الصورة:

لحفظ الصورة نستخدم الأمر cv2.imwrite

شاهد البرنامج التالي:

وظيفة البرنامج: يقوم هذا البرنامج بتحميل الصورة وقراءتها بالمستوي الرمادي ثم عرضها. وخصصنا مفتاح Esc للخروج من نافذة الصورة عند الضغط عليه وخصصنا المفتاح s لحفظ الصورة.

```
import numpy as np
import cv2

img = cv2.imread('gift.jpg',0)
cv2.imshow('image',img)

k = cv2.waitKey(0)
if k == 27:          # wait for Esc key to exit ([27 =Esc] in ASCII)
    cv2.destroyAllWindows()
elif k == ord('s'):      # wait for 's' key to save and exit
    cv2.imwrite('copygift.png',img)
    cv2.destroyAllWindows()
```

شرح الكود:

أول أربع أسطر تم شرحها سابقا

```
k = cv2.waitKey(0)
```

انتظر حتى يتم ضغط أي مفتاح من لوحة المفاتيح

ثم خزن قيم الحرف الذي ضغطنا عليه في متغير سميناه k

```
if k == 27:
    cv2.destroyAllWindows()
```

إذا كان الحرف المضغوط هو Esc (الزر Esc في الكيبورد يكافئ الرقم ٢٧ بشفرة الآسكي) أغلق جميع النوافذ

```
elif k == ord('s'):
```

إذا كان الحرف المضغوط هو s (تعليمة ord تعني حول المحرف الذي بداخلها للرقم الذي يقابله بشفرة الآسكي)

```
 cv2.imwrite('copygift.png',img)
```

احفظ الصورة img باسم جديد "copygift.png" (هنا الاسم الجديد للصورة الجديدة هو copygift)

شرح التابع imwrite():

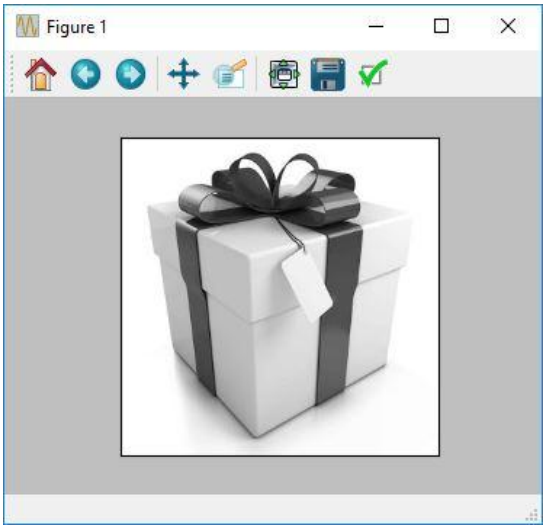
- المتغير الأول: أسم الصورة الجديدة (في حالة عدم وضع مسار محدد سيتم حفظ الصورة الجديدة في مجلد العمل)
- المتغير الثاني: الصورة المطلوب حفظها

٣- تحسين طريقة عرض الصورة عن طريق مكتبة Matplotlib:

الكود:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('gift.jpg',0)

plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')
plt.xticks([]), plt.yticks([])    # to hide tick values on X and Y axis
plt.show()
```

النتيجة:



شرح الكود:

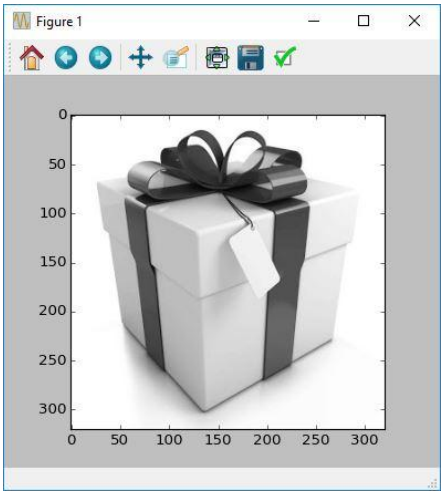
أول أربع أسطر تم شرحها مسبقا

```
plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')
```

plt.imshow تابع عرض الصورة

```
plt.xticks([]), plt.yticks([])
```

وظيفة هذا السطر إخفاء تدرجات محور x وy من نافذة الصورة
(قم بحذف هذا السطر ولاحظ الفرق)



plt.show()

تابع فتح الصورة ضمن نافذة.

هناك العديد من الخيارات متوفرة ضمن مكتبة matplotlib، للحصول على تعليمات أكثر ضمن هذه المكتبة يمكنك قراءة المرجع الخاص بها.

في الكود السابق لاحظ بأنه إذا عرضنا الصورة بغير المستوي الرمادي سنشاهد بأن لون الصورة أختلف فما هو السبب!

السبب:

عند تحميل الصور عبر OpenCv نحصل على الصورة بصيغة BGR ولكن عند عرض الصورة بـ Matplotlib يتم اعتبارها RGB (أي يكون الأحمر أزرق وبالعكس) لذلك يجب مراعاة هذا الوضع والانتباه لتبديل الألوان قبل إظهارها عبر Matplotlib

OpenCv => (B, G, R)
Matplotlib => (R, G, B)

التقاط الفيديو من الكاميرا

في المثال التالي سنقوم بالتقاط صور من الكاميرا وعرضها بسرعة حتى تظهر الصور على هيئة فيديو بث مباشر. (الكاميرا عندما تصور فيديو مباشر هي بالحقيقة تلتقط عدة إطارات للصور بسرعة عالية وتقوم بعرضها فتبدو لنا وكأن الكاميرا تصور بث مباشر).

ملاحظة: كل إطار من إطارات الصورة الملتقطة يدعى فريم **frame**.

(كلما زاد عدد إطارات الصور الملتقطة في الثانية سنحصل على زمن حقيقي أكبر لنقل الصورة).

الكود:

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while(True):
        # Capture frame-by-frame
        ret, frame = cap.read()
        # Our operations on the frame come here
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # Display the resulting frame
        cv2.imshow('frame',gray)
        if cv2.waitKey(1) == ord('q'):
                break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

شرح الكود:

```
cap = cv2.VideoCapture(0)
```

أي حدد الكاميرا التي سنلتقط بها الفيديو من الكاميرا وسمها **cap**

التابع VideoCapture (): تابع التقاط الصور من الكاميرا. قيمة المتغير الذي بداخل القوسين تمثل ترتيب الكاميرا (إذا كنا نستخدم كاميرا واحدة فستأخذ غالبا الترتيب ٠، وإذا كان لدينا ٣ كاميرات وأردنا تشغيل الكاميرا ٣ سنضع بدل صفر الرقم ٢)

```
while(True):
```

حلقة لا نهائية ما دام الشرط TRUE محقق فإذا تحول إلى FALSE سوف تنتهي حلقة while

```
ret, frame = cap.read()
```

التابع read() يعيد قيمتان هما:

المتغير frame: يخزن بداخله الصورة (الفريم) الملتقط من الكاميرا وذلك عن طريق التابع cap.read().

المتغير ret: يأخذ القيمة المعادة من التابع cap.rad() وتكون قيمته إما True أو False.

فإذا كانت القيمة True هذا يعني بأن الكاميرا تلتقط الصور، أما إذا كانت القيمة False فهذا يعني بأن الكاميرا لا تلتقط الصور .

باختصار التابع cap.read() يخزن الصورة الملتقطة من الكاميرا في المتغير frame ، ويعلمنا هل الكاميرا تلتقط الصور أم لا عن طريق تخزين قيمة True أو False في المتغير ret.

أحيانا لا تعمل معك التعليمة cap.read() عندها يمكنك أن تستعمل بدلا منها التعليمة cap.isOpened().

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

حول قيمة كل إطار من إطارات الصورة للون الرمادي وخزنها في متغير سميناه gray

التابع cvtColor():

المتغير الأول يمثل أسم الصورة والمتغير الثاني يمثل المستوى اللوني المطلوب (هنا اخترنا مستوى اللون الرمادي).

```
cv2.imshow('frame',gray)
```

أعرض الصورة frame باللون الرمادي

```
if cv2.waitKey(1) == ord('q'):
        break
```

إذا تم ضغط مفتاح q قم بالخروج من حلقة while وفي الأمر الذي بعده سيتم إغلاق جميع النوافذ المفتوحة.

```
cap.release()
```

هذا الأمر مسؤول عن تحرير الكاميرا (أي أيقاف تشغيلها).

تشغيل الفيديو من ملف

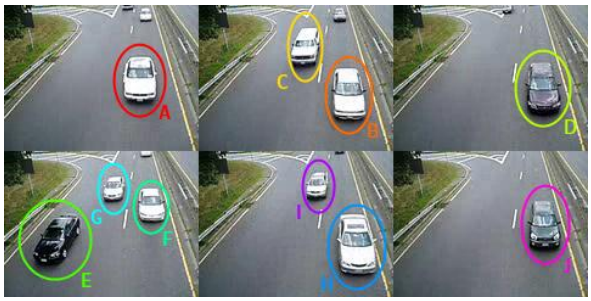
مماثل لما سبق. فقط نستبدل ترتيب الكاميرا في التابع cv2.VideoCapture() باسم الفيديو.

```
import numpy as np
import cv2

cap = cv2.VideoCapture('C:\Users\A\Documents\video.mp4')

while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()
    # Our operations on the frame come here
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Display the resulting frame
    cv2.imshow('frame',gray)
    if cv2.waitKey(10) == ord('q'):
        break

# When everything done, release the capture
cv2.destroyAllWindows()
```

ملاحظة: في إصدار **OpenCv 3** إن لم يعمل معك الكود السابق لعدم القدرة على قراءة الفيديو، فأنت بحاجة لأن تنصب **ffmpeg or gstreamer**.

Make sure proper versions of ffmpeg or gstreamer is installed. Sometimes, it is a headache to work with Video Capture mostly due to wrong nstallation of ffmpeg/gstreamer.

توابع الرسم



الهدف:

- تعلم رسم مختلف الأشكال الهندسية باستخدام مكتبة OpenCv.
- سنتعلم استخدام التوابع التالية:
- cv2.line(), cv2.circle(), cv2.rectangle(), cv2.ellipse(), cv2.putText()

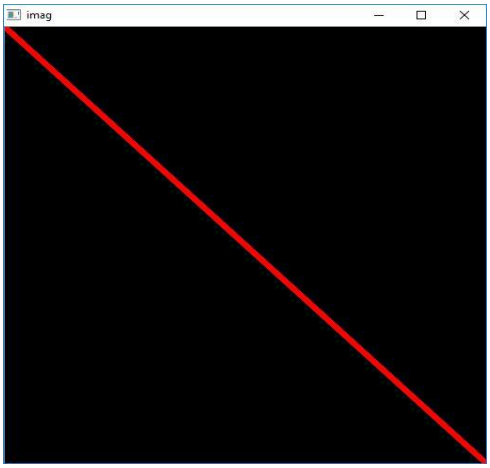
سنلاحظ عند استخدام هذه التوابع بأنها تحتوي على المتغيرات التالية:

١. الصورة التي سيتم الرسم عليها
٢. لون الشكل (لتحديد اللون نمرر ثلاث قيم هي R,G,B)
٣. سماكة الخط
٤. نوع الخط (متصل – متقطع..)

رسم خط

لرسم خط نحتاج إحداثية البداية والنهاية. في هذا المثال سنرسم خط قطري في الصورة

```
import numpy as np
import cv2
# Create a black image
img = np.zeros((512,512,3), np.uint8)

# Draw a diagonal blue line with thickness of 5 px
cv2.line(img,(0,0),(511,511),(0,0,255),5)

# lets see
cv2.imshow('imag',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



شرح الكود:

```
img = np.zeros((512,512,3), np.uint8)
```

التابع np.zeros():

لرسم صورة لونها أسود (مصفوفة صفرية: عناصرها تكون جميعها أصفار ٠ = اللون الأسود)

المتغير الأول: أول قيمتين هما أبعاد المصفوفة (حجم الصورة) وثالث متغير هو عمقها

المتغير الثاني: يمثل نوع عناصر المصفوفة التي تكون الصورة هنا النوع هو uint8

Uint8 هي اختصار لـ unsigned integer 8-bit عدد صحيح غير مقيد بالإشارة يتألف من ٨ بت

```
cv2.line(img,(0,0),(511,511),(0,0,255),5)
```

التابع line () يستخدم لرسم خط، يتضمن عدة متغيرات تساعدنا في عملية الرسم وهي:

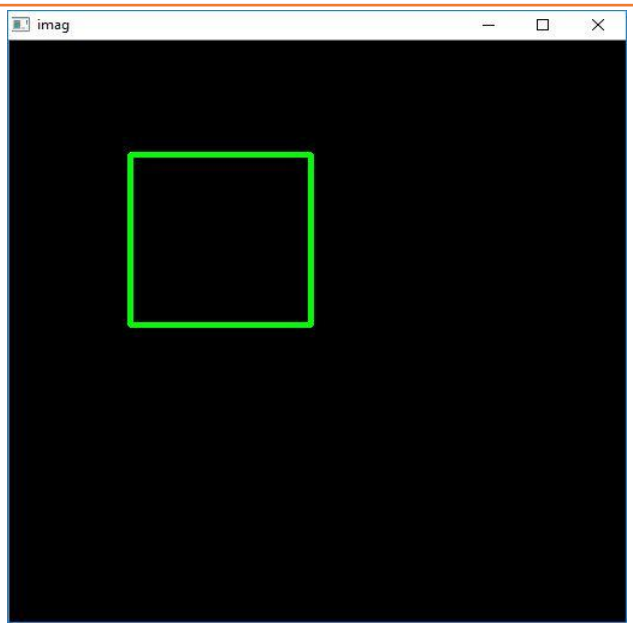
المتغير الأول هو أسم الصورة والمتغير الثاني هو إحداثيات بدية الخط والمتغير الثالث هو إحداثيات نهاية الخط والمتغير الرابع هو قيمة اللون (B,G,R) والمتغير الخامس هو سماكة الخط

رسم مستطيل

لرسم المستطيل يجب أن نحدد زأويته العليا اليسرى وزأويته السفلى اليمنى.

كود رسم مستطيل على صورة سوداء:

```
import numpy as np
import cv2
# Create a black image
img = np.zeros((512,512,3), np.uint8)

#Draw Rectangle
cv2.rectangle(img,(100,100),(250,250),(0,255,0),3)

# lets see
cv2.imshow('imag',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



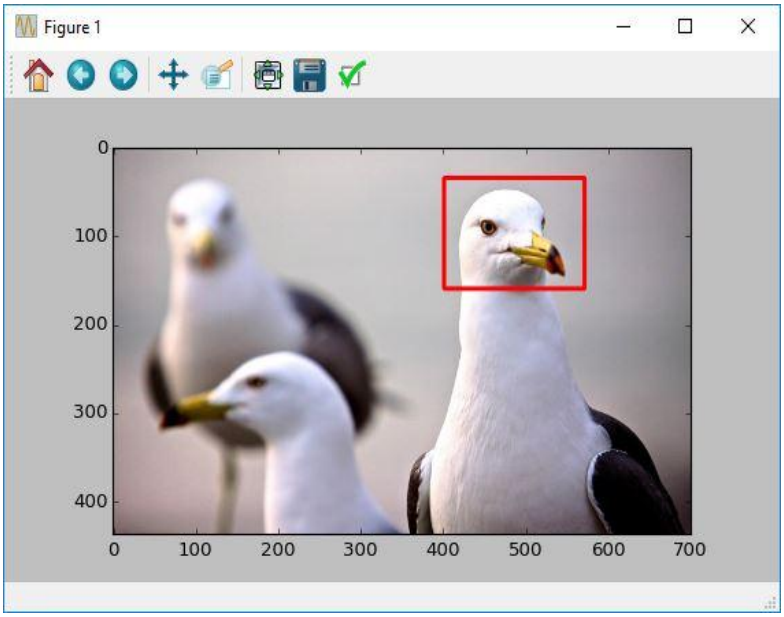
شرح الكود:

```
cv2.rectangle(img,(100,100),(250,250),(0,255,0),3)
```

التابع rectangle() وظيفته هي رسم مستطيل ويتم تحديد خصائص هذا المستطيل عن طريق المتغيرات التالية. المتغير الأول هو أسم الصورة والمتغير الثاني والثالث هما إحداثيات زأوية المستطيل العليا اليسرى والسفلى اليمنى.

كود رسم مستطيل على صورة موجودة مسبقا:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

# read image
img =cv2.imread('brid.jpg')

#invert color from B,G,R  to R,G,B
b,g,r = cv2.split(img)       # get b,g,r
img = cv2.merge([r,g,b])     # switch it to RGB

# Draw a diagonal blue line with thickness of 5 px
cv2.rectangle(img,(400,35),(570,160),(255,0,0),3)

# lets see
plt.imshow(img)
plt.show()
```



شرح الكود:

```
b,g,r = cv2.split(img)       # get b,g,r
img = cv2.merge([r,g,b])     # switch it to RGB
```

cv2.split() وظيفته الحصول على ألوان الصورة الـ B, G, R كل واحد لوحده.

cv2.merge() وظيفته دمج الألوان سنستخدمه لدمج الألوان التي حصلنا عليها من التابع السابق ولكن بصيغة معكوسة. (أقصد بدمج الألوان بصيغة معكوسة، بأن أبدل اللون الأحمر باللون الأزرق وبالعكس أما اللون الأخضر سنبقي على ترتيبه كم هو، فنحصل على الترتيب RGB بدل BGR) قمنا باستدعاء التابعين السابقين لتحويل الألوان من الترتيب BGR إلى RGB.

لقد ذكرنا سابقا بأن مكتبة cv2 تعرض الألوان بالترتيب BGR ولكن مكتبة matplotlib تأخذ الترتيب RGB لهذا السبب قد عكسنا الألوان لتظهر الصورة باللون الحقيقي. جرب حذف هذين السطرين وستعرف ماذا سيحدث. (سيصبح اللون الأحمر بدل الأزرق والأزرق بدل الأحمر).

ملاحظة: مكتبة cv2 تعرض الألوان بالترتيب BGR، بينما مكتبة matplotlib تعرض الألوان بالترتيب RGB.

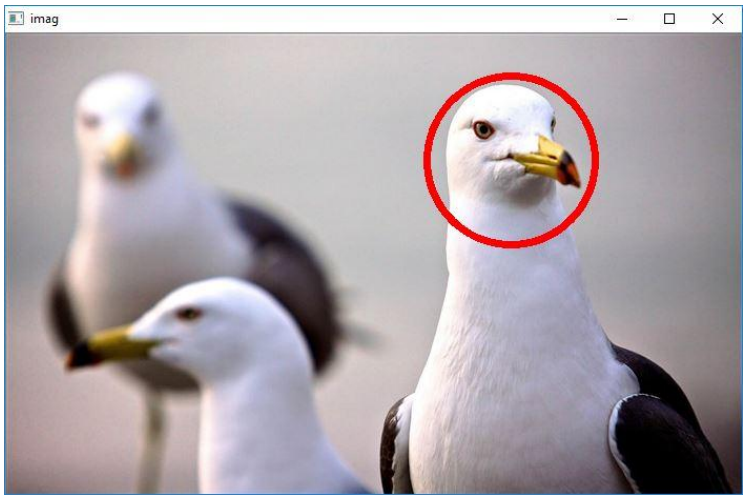
```
cv2.rectangle(img,(100,100),(250,250),(0,255,0),3)
```

التابع () rectangle وظيفته هي رسم مستطيل ويتم تحديد خصائص هذا المستطيل عن طريق المتغيرات التالية: المتغير الأول هو أسم الصورة والمتغير الثاني والثالث هما إحداثيات زأوية المستطيل العليا اليسرى والسفلى اليمنى.

رسم دائرة

لرسم دائرة نحتاج مركزها ونصف قطرها

```
import numpy as np
import cv2

img =cv2.imread('brid.jpg')

# Draw circle
cv2.circle(img,(480,120), 80, (0,0,255), 5)

cv2.imshow('imag',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



شرح الكود:

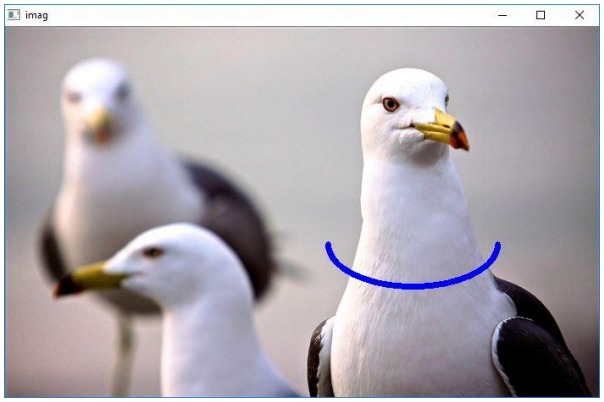
```
cv2.circle(img,(250,250), 63, (0,0,255), ٥)
```

التابع () circle وظيفته رسم دائرة المتغير الأول هو اسم الصورة التي سنرسم عليها والمتغير الثاني هو مركز الدائرة والمتغير الثالث هو قطر الدائرة والمتغير الرابع هو لون الدائرة والمتغير الخامس يمثل سماكة محيط الدائرة.

رسم قطع ناقص

لرسم قطع ناقص نحتاج تمرير عدة متغيرات (المركز + أطوال المحوريين +زأوية الدوران +زأوية البداية والنهاية) يتم الرسم عن طريق استدعاء التابع cv2.ellipse

```
cv2.ellipse(img,(480,256),(100,50),0,0,180,255,5)
```

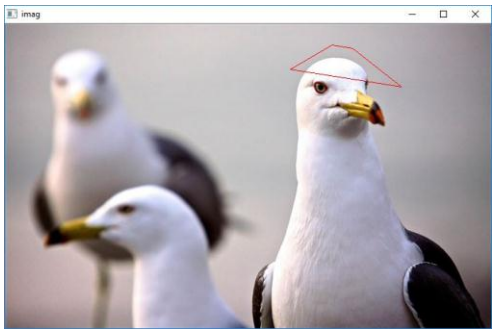


رسم مضلع

لرسم مضلع نحتاج رؤوس الإحداثيات ومن ثم نجعلها بشكل مصفوفة ويجب أن تكون هذا المصفوفة بالصيغة int32المثال التالي لرسم مضلع بأربع رؤوس

```
pts = np.array([[410,65],[470,30],[500,35],[570,90]], np.int32)
pts = pts.reshape((-1,1,2))
cv2.polylines(img,[pts],True,(0,0,255))
```

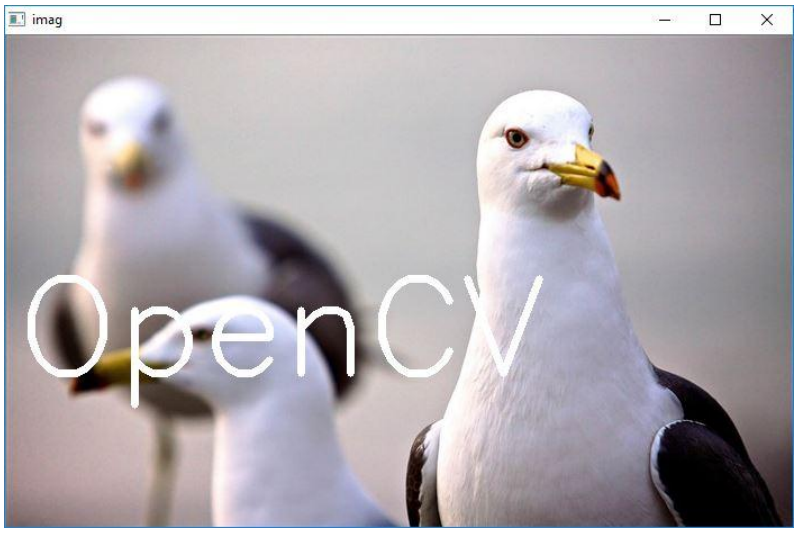
لو كانت قيمة المتغير الثالث للتابع cv2.polylines() هي False لحصلنا على مضلع مفتوح .
يمكن للتابع السابق رسم عدة مستقيمات معا، ما عليك إلا تمرير مصفوفات النقاط وهذا أفضل من رسم كل مستقيم على حدى.



إضافة نص للصورة

لإضافة نص للصورة عليك تحديد ما يلي:
النص الذي تريد إدراجه، موضع النص بالنسبة للصورة (من الزأوية السفلى اليسرى)، نوع الخط، حجم الخط بالبيكسل، السماكة، اللون، هل الخط متصل أم منفصل.

```
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'OpenCV',(10,300), font, 4,(255,255,255),5)
```



ملاحظة: يمكنك استدعاء أكثر من تابع رسم على نفس الصورة

أحداث الفأرة

- في هذه الدرس سنتعلم التعامل مع أحداث الفأرة في OpenCv.
- سنتعلم استخدم التابع التالي cv2.setMouseCallback()

أحداث الفأرة: هي أي شيء مرتبط بالفأرة مثل ضغط الزر الأيسر، أو رفع الضغط، أو الضغط مرتين.....الخ

يعطينا التابع cv2.setMouseCallback() إحداثيات (X,Y) المكان الذي يوجد به مؤشر الفأرة على الشاشة.لمعرفة أحداث الفأرة المتوفرة، يمكنك كتابة الأمر التالي:

```
import cv2
events = [i for i in dir(cv2) if 'EVENT' in i]
for a in events :
    print a
```

```
E  EVENT_FLAG_ALTKEY
EVENT_FLAG_CTRLKEY
EVENT_FLAG_LBUTTON
EVENT_FLAG_MBUTTON
EVENT_FLAG_RBUTTON
EVENT_FLAG_SHIFTKEY
EVENT_LBUTTONDBLCLK
EVENT_LBUTTONDOWN
EVENT_LBUTTONUP
EVENT_MBUTTONDBLCLK
EVENT_MBUTTONDOWN
EVENT_MBUTTONUP
EVENT_MOUSEHWHEEL
EVENT_MOUSEMOVE
EVENT_MOUSEWHEEL
EVENT_RBUTTONDBLCLK
EVENT_RBUTTONDOWN
EVENT_RBUTTONUP
```

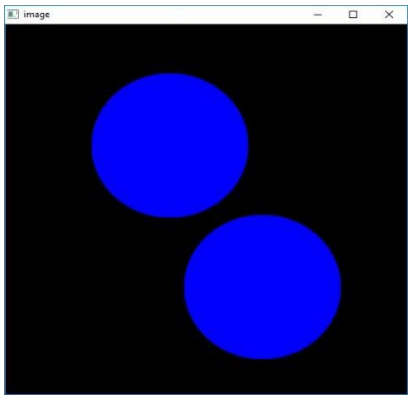
الرسم باستخدام الفأرة

ملاحظة: يمكنك تنفيذ أي أمر عند الضغط على زر الفأرة عن طريق التابع الذي ستكتبه ولكن هنا سنقتصر على استخدام حدث الضغط على زر الفأرة للرسم فقط.

مثال١:

سننشأ تطبيق بسيط لرسم دائرة عندما نضغط بالفأرة على الشاشة. وسننشأ تابع استدعاء مرتبط بحدث ضغط زر الفأرة الأيسر مرتين لرسم دائرة.

```
import cv2
import numpy as np

# mouse callback function
def draw_circle(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDBLCLK:
        cv2.circle(img,(x,y),100,(255,0,0),-1)
# Create a black image, a window and bind the function to window
img = np.zeros((512,512,3), np.uint8)
cv2.namedWindow('image')
cv2.setMouseCallback('image',draw_circle)

while(1):
    cv2.imshow('image',img)
    if cv2.waitKey(20) == 27:
        break
cv2.destroyAllWindows()
```



شرح الكود:

```
def draw_circle(event,x,y,flags,param):
```

تستخدم عبارة defلتعريف التوابع في لغة بايثون
عرفنا تابع باسم draw_circle ومررنا له ٥ قيم.

```
    if event == cv2.EVENT_LBUTTONDBLCLK:
        cv2.circle(img,(x,y),100,(255,0,0),-1)
```

إذا كان الحدث المضغوط هو الضغط على زر المأوس الأيسر مرتين نفذ ما بداخل جملة if الشرطية: ارسم دائرة مركزها هو إحداثيات (x,y) لموقع مؤشر الفأرة

```
img = np.zeros((512,512,3), np.uint8)
cv2.namedWindow('image')
```

قمنا بإنشاء صورة لونها أسود بحجم 512 x 512 ثم سمينا نافذة الصورة بالاسم image.

```
cv2.setMouseCallback('image',draw_circle)
```

التابع setMouseCallback() مسؤول عن تنفيذ حدث الفأرة وهنا سيقوم برسم دائرة عن طريق استدعاء التابع الذي أنشأناه في بداية الكود وهو draw_circle.

شرح التابع setMouseCallback()

المتغير الأول هو أسم النافذة المطلوب تنفيذ حدث الفأرة عليها.
المتغير الثاني هو الأمر المطلوب تنفيذه عند حدوث حدث الفأرة.

مثال٢:

سنقوم في هذا المثال برسم مستطيل أو دائرة بناءً على الحالة التي سنختارها، لذلك سيكون لدى تابع استدعاء الصورة لديه شقين، واحد لرسم مستطيل والأخر لرسم دائرة.

هذا المثال قد ينفعنا في بعض التطبيقات كتتبع الأجسام وتقطيع الصورة. سنقوم بتعديل الكود السابق بحيث سنختار بأن نرسم دائرة أو مستطيل حسب ما نختار، وللتنقل بين اختيار مستطيل أو دائرة نضغط على حرف m من لوحة المفاتيح.

```
import cv2
import numpy as np

drawing = False # true if mouse is pressed
mode = True # if True, draw rectangle. Press 'm' to toggle to curve
ix,iy = -1,-1

# mouse callback function
def draw_circle(event,x,y,flags,param):
    global ix,iy,drawing,mode

    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix,iy = x,y
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing == True:
            if mode == True:
                cv2.rectangle(img,(ix,iy),(x,y),(0,255,0),-1)
            else:
                cv2.circle(img,(x,y),5,(0,0,255),-1)
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        if mode == True:
            cv2.rectangle(img,(ix,iy),(x,y),(0,255,0),-1)
        else:
            cv2.circle(img,(x,y),5,(0,0,255),-1)


img = np.zeros((512,512,3), np.uint8)
cv2.namedWindow('image')
cv2.setMouseCallback('image',draw_circle)
```

```
while(1):
    cv2.imshow('image',img)
    k = cv2.waitKey(1)
    if k == ord('m'):
        mode = not mode
    elif k == 27:
        break

cv2.destroyAllWindows()
```

مثال٣:

في المثال السابق رسمنا مستطيل ممتلئ فهل بإمكانك تغيير ذلك لرسم مستطيل مفرغ.

الحل:

```
import cv2
import numpy as np

drawing = False
mode = False
ix,iy = -1,-1
# mouseCallback Function
def draw_circle(event,x,y,flags,param):
    global ix,iy,drawing,mode,spareimg,img

    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix,iy = x,y
```

```
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing == True:
            if mode == True:
                img = spareimg.copy()
                spareimg = img.copy()
                cv2.rectangle(img,(ix,iy),(x,y),(0,255,0),2)
            else:
                cv2.circle(img,(ix,iy),3,(0,0,255),-1)
                cv2.line(img,(ix,iy),(x,y),(0,0,255),3)
                ix,iy = x,y

    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        if mode == True :
            cv2.rectangle(img,(ix,iy),(x,y),(0,255,0),2)
            spareimg = img.copy()
        else:
            cv2.circle(img,(x,y),3,(0,0,255),-1)

img = cv2.imread('gift.jpg',1)
spareimg = img.copy()
# img = np.zeros((512,512,3),np.uint8)
cv2.namedWindow('image',cv2.WINDOW_NORMAL)
cv2.setMouseCallback('image',draw_circle)

while(1):
    cv2.imshow('image',img)
    k = cv2.waitKey(1)
    if k == ord('m'):
        mode = not mode
    elif k == 27:
        break

cv2.destroyAllWindows()
```

شريط التمرير الجانبي

- **استخدام شريط التمرير الجانبي(Track bar) كلوحة ألوان**

الهدف:

- تعلم إنشاء شريط سحب في نافذة windows ضمن مكتبة OpenCv.
- سنتعلم استخدام التوابع التالية:() cv2.createTrackbar , cv2.getTrackbarPos() .

كود التجربة:

سننشأ تطبيق بسيط يعرض لون محدد في نافذة، ويتم تغيير اللون المعروض عن طريق ثلاثة أشرطة سحب، كل أداة سحب تمثل لون **R,G,B**.

سنعين اللون الأسود كلون افتراضي.

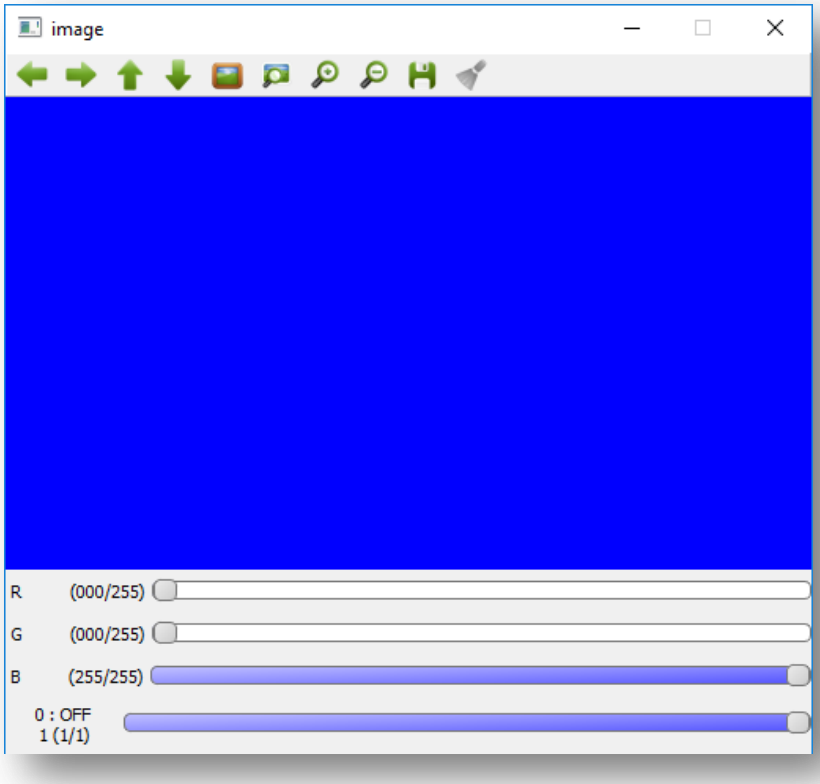
التابع cv2.getTrackbarPos() يأخذ المتغيرات التالية:

أول متغير يمثل أسم شريط السحب، وثاني متغير هو اسم النفذة التي سنضع عليها الأداة، وثالث متغير هو القيمة الافتراضية، ورابع متغير هو القيمة العظمى، وخامس متغير هو تابع الاستدعاء (أي الأمر المطلوب تنفيذه).في مثالنا هذا لن نستعمل تابع الاستدعاء لذلك سنمرر له تابع خُلَّبي.

وهناك تطبيق آخر لتابع الاستدعاء وهو استخدامه كزر تبديل (لأن مكتبة OpenCv لا تحوي خاصية زر)، لذلك سننشأ خاصية لتفعيل ظهور اللون أو لا كما هو موضح في الكود التالي:

```
import cv2
import numpy as np

def nothing(x):
  pass

# Create a black image, a window
img = np.zeros((300,512,3), np.uint8)
cv2.namedWindow('image')

# create trackbars for color change
cv2.createTrackbar('R','image',0,255,nothing)
cv2.createTrackbar('G','image',0,255,nothing)
cv2.createTrackbar('B','image',0,255,nothing)

# create switch for ON/OFF functionality
switch = '0 : OFF \n1 : ON'
cv2.createTrackbar(switch, 'image',0,1,nothing)

while(1):
  cv2.imshow('image',img)
  k = cv2.waitKey(1) & 0xFF
  if k == 27:
    break
  # get current positions of four trackbars
  r = cv2.getTrackbarPos('R','image')
  g = cv2.getTrackbarPos('G','image')
  b = cv2.getTrackbarPos('B','image')
  s = cv2.getTrackbarPos(switch,'image')

  if s == 0:
    img[:] = 0
  else:
    img[:] = [b,g,r]

cv2.destroyAllWindows()
```

النتيجة:

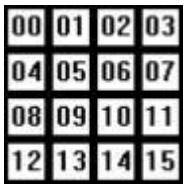


الفصل الثاني

”من سلك طريقا يلتمس فيه علما سهل الله له طريقا إلى الجنة “

سيد الرُسل – محمد صلى الله عليه وسلم

الفصل الثاني: العمليات الأساسية على الصور



❖ العمليات الأساسية على الصور

تعلم قراءة البيكسل وكيفية تغيير قيمته، وتعلم نسخ جزء محدد من الصورة، وعمليات أساسية أخرى

❖ العمليات الحسابية على الصور

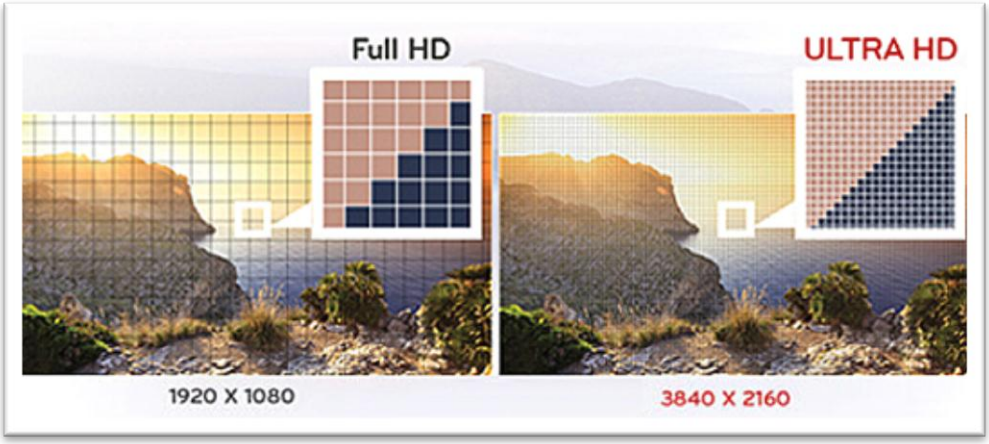
إجراء عمليات حسابية على الصور

❖ تقنيات قياس الأداء وتحسينه

الحصول على الحل الذي نريد ولكن بطريقة أسرع، وتعلم فحص سرعة الكود، وكيفية الحصول على الكود المثالي.

❖ الأدوات الرياضية في مكتبة OpenCv

العمليات الأساسية على الصور



الهدف:

- قراءة قيمة أي بيكسل في الصورة وطريقة تعديل قيمته.
- معرفة خصائص الصورة.
- تغير منطقة محددة من الصورة.
- دمج الصور وفصلها.

أغلب العمليات الاساسية على الصورة تتم عن طريق الاستعانة بمكتبة numpy، وتتم بشكل أفضل من استعمال مكتبة OpenCv. لذلك يجب أن تتعرف على مكتبة Numpy بشكل جيد.

قراءة وتعديل قيمة أي بيكسل في الصورة

دعنا نحمل الصورة الملونة التالية حتى نبدأ بالشرح عليها.

```
import cv2
import numpy as np
img = cv2.imread('robot_football.jpg')
```

يمكننا الوصول لقيمة أي بيكسل عن طريق إحداثيات السطر والعمود (X,Y).

بالنسبة للصورة الملونة عند قراءة قيمة البيكسل سيتم قراءة ثلاث ألوان وهي الأحمر والأخضر والأزرق. الكود التالي سيقرأ ألوان البيكسل الذي إحداثياته y=100 , x=100 وسيعيد لنا قيمة الألوان R G B.

```
px = img[100,100]
print px
```

[١٣٩ ٦٨ ٤١]

قراءة لون واحد: الكود التالي فقط سيقرأ اللون الأحمر

```
# accessing only blue pixel
red = img[100,100,2]
print red
```

٤١

يمكنك تغيير لون أي بيكسل في الصورة عن طريق تغيير قيمته.
في الكود التالي سنغير قيمة لون البيكسل الذي إحداثياته (100,100) إلى اللون الأحمر.

```
img[100,100] = [0, 0,255]
print img[100,100]
```

ملاحظة:

- مكتبة Numpy هي أفضل مكتبة لمعالجة المصفوفات بسرعة
- عند استعمال الطريقة السابقة للوصول إلى كل بيكسل سيؤدي هذا الأمر لبطئ في البرنامج.

لتعامل مع كل بيكسل لوحده يفضل استخدام التابعين item() و itemset() وذلك كما يلي:
الكود التالي سيقرأ قيمة اللون الأحمر في البيكسل ذو الإحداثية (10,10).
(اللون الأحمر يكافئ رقم ٢، واللون الأخضر يكافئ الرقم 1، واللون الأزرق يكافئ الرقم 0)

```
# accessing RED value
img.item(10,10,2)
```

الآن سنغير قيمة الـ R (اللون الأحمر) في البيكسل ذو الإحداثيات (10 , 10) ونستبدله بقيمة لونية قيمتها 100.

(قارن بين النتيجة في الكود السابق وهذا الكود في نفس البيكسل ولاحظ ماذا حدث)

```
# modifying RED value
img.itemset((10,10,2),100)
img.item(10,10,2)
```

خصائص الصورة

عدد القنوات (channels)، عدد الأسطر، عدد الأعمدة، عدد البيكسلات، .....

لمعرفة شكل الصورة shape(عدد الأسطر، عدد الأعمدة، عدد القنوات) نكتب الكود التالي:

```
print img.shape
```

(320 ،320 ،3)

لمعرفة عدد البيكسلات نكتب الكود التالي:

```
print img.size
```

١٢٦٨٥٦٨

لمعرفة نوع بيانات الصورة نكتب الكود التالي:

```
print img.dtype
```

uint8

تنبيه: يجب الانتباه لنوع بيانات الصورة لأنه يوجد العديد من الأخطاء تحدث في بيئة **Opencv_Python** سببه نوع بيانات الصورة **data type**

المنطقة المهمة في الصورة (Image ROI)

ما المقصود بـ ROI؟! (Region Of Interest)

أحيانا نحتاج لتحرير منطقة محددة من الصورة، فمثلا من أجل اكتشاف العين عليك أولا البحث عن الوجه، ومن ثم نضع حدود للوجه ونقوم بالبحث داخل هذه الحدود عن العينين. هذه العملية توفر علينا البحث ضمن الصورة كلها وتحسن سرعة ودقة الاكتشاف (لأن العينين دائما موجودة على الوجه :D) ويتحسن الأداء (لأننا بحثنا ضمن مساحة صغيرة).

في هذا الكود التالي سنقوم بتحديد منطقة من الصورة ومن ثم نقوم بنسخها ووضعها في مكان أخر من الصورة.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('football.jpg')

part = img[150:300, 400:550]
img[150:300, 50:200] = part

plt.imshow(img)
plt.xticks([])
plt.yticks([])
plt.show()
```

أنظر للنتيجة في الصور التالية:

قبل:

بعد:

شاهد كيف تم نسخ الكرة مرتين.

شرح الكود:

```
part = img[150:300, 400:550]
```

هذا السطر مسؤول عن القسم الذي نريد نسخه

لنفرض أن احداثيات الزأوية العليا اليسرى للقسم المراد نسخه هي (X1,Y1) وإحداثيات الزأوية السفلى اليمنى هي (X2,Y2) ، عندها سيكون الترتيب كما يلي: img[Y1:Y2, X1:X2]

```
img[150:300, 50:200] = part
```

هذا السطر مسؤول عن القسم الذي نريد وضع القسم المنسوخ فيه، ويأخذ أيضا نفس الترتيب السابق img[Y1:Y2, X1:X2].

ملاحظة: يجب مراعاة أن يكون الفرق بين **X1** و **X2** نفسه في القسم المنسوخ، وكذلك الأمر بالنسبة لـ**Y1** و**Y2**.

فصل ودمج قنوات الصورة

أحيانا نحتاج أن نقوم بفصل قنوات ألوان الصورة B,G,R كل قناة لونية على حدى. بحيث يمكننا التعامل مع كل قناة لونية بشكل منفصل، كما يمكننا أيضا دمج قنوات الألوان معا للحصول على صورة ملونة.

طريقة الفصل والدمج سهلة جدا ويمكنك القيام بذلك عن طريق الكود التالي:

```
b,g,r = cv2.split(img)          # تعليمة فصل قنوات الألوان عن بعضها
img = cv2.merge((b,g,r))        # تعليمة دمج  قنوات  الألوان مع بعضها
```

```
# Or
b = img[:,:,0]
```

لنفرض أنك تريد أن تجعل جميع البيكسلات في القناة R قيمتها صفر، ستقوم في البداية بفصل قنوات الألوان عن طريق تعليمة () split ثم ستجعل قيمة القناة Rتسأوي الصفر. (هذه العملية بطيئة)

يمكنك فعل الأمر السابق بشكل أبسط عن طريق الاستعانة بمكتبة المصفوفات Numpy، وسيتم الأمر السابق بشكل أسرع. شاهد لكود التالي:

```
img[:,:,2] = 0
```

تنبيه:

عملية **split** تأخذ وقت كبير نسبيا لتنفيذ الأمر، لذلك لا تستعملها إلا عند الضرورة، واستعمل بدلا عنها العمليات على المصفوفات فهي أفضل بكثير.

إنشاء حدود للصورة

إذا أردت إنشاء حدود للصورة استعمل التابع cv2.copyMakeBorder().
يأخذ هذا التابع المتغيرات التالية:
صورة الدخل، القمة، القاعدة، اليمين، اليسار، عرض الحدود، نوع الحدود، لون الحد.
متغير نوع الحدود يأخذ القيم التالية:

cv2.BORDER_CONSTANT: إضافة لون ثابت للحدود (يأخذ قيمة اللون من المتغير التالي)
cv2.BORDER_REFLECT: عكس عناصر الحدود مثل ...

FEDCBA | ABCDEFG | GFEDCBA

الملون بالرمادي نعتبره قيمة الحدود والعناصر التي حول الحدود هي الملونة بالأزرق. كل حرفي يمثل قيمة لونية لبيكسل.
cv2.BORDER_REFLECT_101 or cv2.BORDER_DEFAULT: نفس التعليمة السابقة ولكن مع القليل من التغيرات

GFEDCBA | ABCDEFG | GFEDCBA

cv2.BORDER_REPLICATE: يتم نسخ العنصر الأخير في جميع الأنحاء مثل:

AAAAA | ABCDEFGH | HHHHHHH

v2.BORDER_WRAP، لا يمكن تفسيره ولكن سيبدو هكذا: : التفاف الحدود

CDEFGH | ABCDEFGH | ABCDEFG

المتغير الأخير هو اللون (فقط نستعمله في حال أخترنا النوع الأول من الحدود)

حتى نفهم أنواع الحدود بشكل أوضح سنكتب البرنامج التالي الذي يحتوي على كل أنواع الحدود.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

BLUE = [255,0,0]

img1 = cv2.imread(' opencv_logo.jpg')

replicate = cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER_REPLICATE)
reflect = cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER_REFLECT)
reflect101 = cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER_REFLECT_101)
wrap = cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER_WRAP)
constant= cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER_CONSTANT,value=BLUE)

plt.subplot(231),plt.imshow(img1,'gray'),plt.title('ORIGINAL')
plt.subplot(232),plt.imshow(replicate,'gray'),plt.title('REPLICATE')
plt.subplot(233),plt.imshow(reflect,'gray'),plt.title('REFLECT')
plt.subplot(234),plt.imshow(reflect101,'gray'),plt.title('REFLECT_101')
plt.subplot(235),plt.imshow(wrap,'gray'),plt.title('WRAP')
plt.subplot(236),plt.imshow(constant,'gray'),plt.title('CONSTANT')

plt.show()
```

العمليات الجبرية على الصور

الهدف:

- تعلم العديد من العمليات الحسابية على الصورة مثل: الإضافةaddition ، الطرح subtraction، العمليات التي تجرى على مستوى البت (bitwise operations).
- تعلم استخدام التوابع التالية: () cv2.add،cv2.addWeighted()

جمع الصور:

يمكن جمع صورتين من خلال التابع () cv2.add أو ببساطة من خلال جمع المصفوفات res = img1 + img2. ، لكن يجب أن تملك المصفوفتين نفس العمق depthوالنوع type أو أن تكون المصفوفة الثانية قيمتها صحيحة.

ملاحظة: هناك فرق بين عمليات الجمع العددية بواسطة **Numpy**وعمليات الجمع بواسطة **OpenCv**، فالأولى هي عملية طرح أما الثانية فهي عملية جمع بحت.

شاهد المثال التالي:

سنقوم بإنشاء صورتين بنفس النوع وسنجري عليهما عملية الجمع مرة بالاستعانة بمكتبة OpenCvومرة أخرى بالاستعانة بمكتبة Numpy.

```
x = np.uint8([250])
y = np.uint8([10])
```

الجمع بالاستعانة بـمكتبة OpenCv

```
print cv2.add(x,y)          # 250+10 = 260 => 255
```

```
[[255]]
```

الجمع بالاستعانة بـ numpy

```
print x+y                   # 250+10 = 260 % 256 = 4
```

```
[[4]]
```

ستشاهد النتائج بشكل واضح عند جمع الصورتين، وبما أن عملية الجمع بالاستعانة بمكتبة OpenCv كانت نتائجها أفضل لذا سنستعملها دائما عند الجمع.

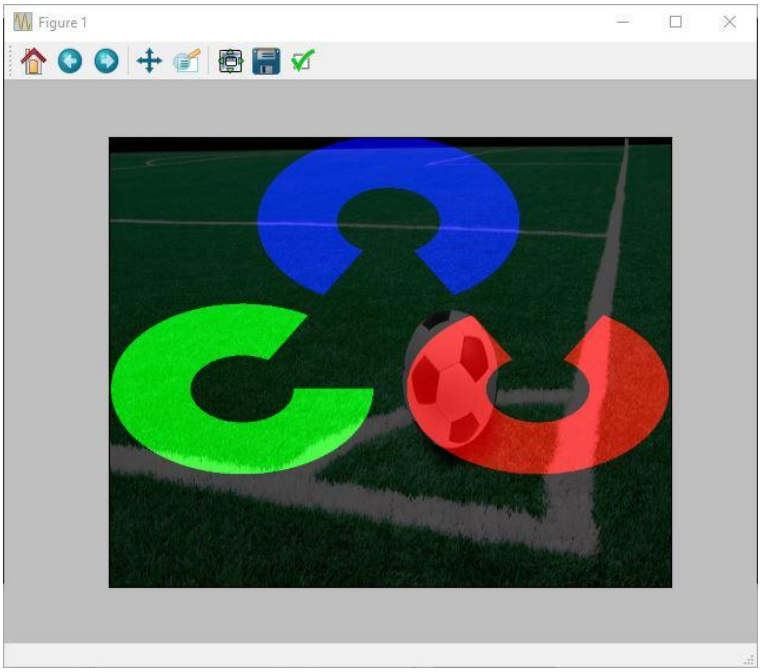
دمج الصور:

هنا أيضا نقوم بجمع الصور ولكن نعطي كل منها وزن مختلف أي سنعطي كل صورة شفافية مختلفة عن الأخرى، كما هو واضح في الصورة التالية.

هنا أخذنا صورتين لدمجهم معا، الصورة الأولى وزنها (أي مقدار شفافيتها) 0.7والصورة الثانية وزنها 0.3،وسنقوم بدمج هاتين الصورتين عن طريق التابع cv2.addWeighted

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img1 = cv2.imread('logo_opencv.png',1)
img2 = cv2.imread('football.jpg',1)

# 600X600 sound cool
# img2 = cv2.resize(img2,(img1.shape[1],img1.shape[0]))
img2 = cv2.resize(img2,(750,600))
img1 = cv2.resize(img1,(750,600))

res = cv2.addWeighted(img1,0.7,img2,0.3,0)

plt.imshow(res)
plt.xticks([]),plt.yticks([])
plt.show()
```

النتيجة:

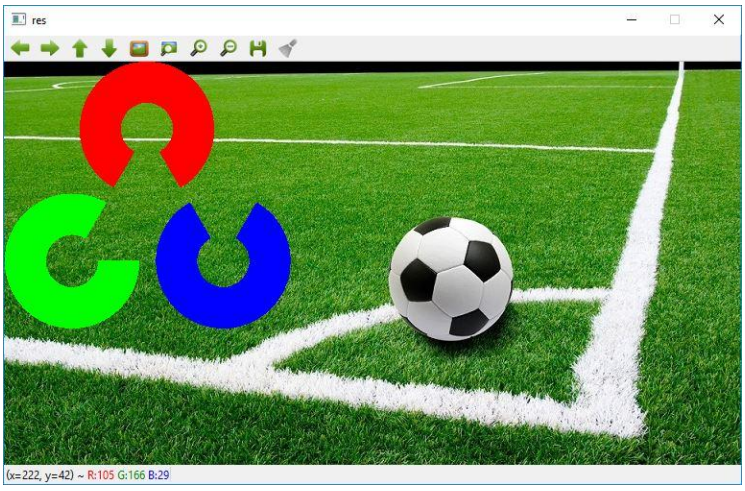


العمليات على مستوى البت: (bitwise operations)

تتضمن عملياتbitwise العمليات التالية AND، OR، XOR، NOT.

هذ العمليات لها فائدة كبيرة عند تحديد أي جزء من الصورة شكله ليس مستطيل. مثلاً في البرنامج التالي سنقوم بوضع شعار OpenCvعلى زأوية الصورة، وهذا الأمر لا يمكن فعله بعملية الجمع (Addition) لأن اللون سوف يتغير، ولا بالدمج (blending)لأن اللون سيصبح شفافا، لكن يمكننا أن نفعل ذلك بعملية الاسناد ROI كما قمنا بها في هذا الفصل ولكن المشكلة بأن شعار OpenCvشكله غير مستطيل لذلك سنقوم بما يلي:

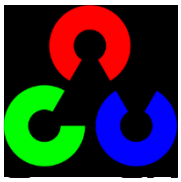
```
import cv2
import numpy as np

# Load two images
img1 = cv2.imread('football.jpg')
img2 = cv2.imread('opencv_logo.png')

# I want to put logo on top-left corner, So I create a ROI
rows,cols,channels = img2.shape
roi = img1[0:rows, 0:cols ]

# Now create a mask of logo and create its inverse mask also
img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
ret, mask = cv2.threshold(img2gray, 10, 255, cv2.THRESH_BINARY)
mask_inv = cv2.bitwise_not(mask)

# Now black-out the area of logo in ROI
img1_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)

# Take only region of logo from logo image.
img2_fg = cv2.bitwise_and(img2,img2,mask = mask)

# Put logo in ROI and modify the main image
dst = cv2.add(img1_bg,img2_fg)
img1[0:rows, 0:cols ] = dst

cv2.imshow('res',img1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



شرح الكود:

في البداية قمنا باستيراد مكتبة OpenCvومكتبة Numpy ثم قمنا بتحميل الصورتين وخزنا كل صورة في متغير.

```
# I want to put logo on top-left corner, So I create a ROI
rows,cols,channels = img2.shape
roi = img1[0:rows, 0:cols ]
```



ثم قمنا بقراءة ابعاد صورة شعارOpenCv (حتى نعرف الحجم الذي ستأخذه صورة الشعار لنقوم بحجزه في الصورة التي سنضع عليها الشعار (صورة الملعب))، وخزنا قيمة صفوف صورة الشعار في المتغير row، وقيمة اعمدتها في المتغير cols، وعدد قنواتها في المتغير channels.

ثم قمنا بتحديد موقع في الصورة الذي سنضع عليه شعار OpenCv، الموقع الذي سنضع فيه الشعار وهو الزأوية اليسرى العليا للصورة، وذلك عن طريق الكود التالي:

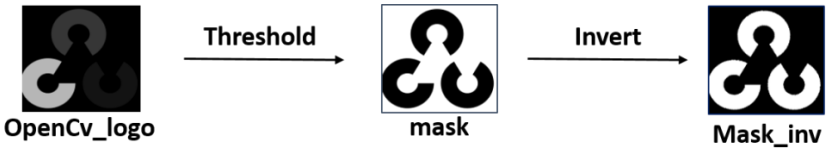
img1[0:rows, 0:cols] قراءة الحيز من الصورة الذي تبدأ إحداثياته من (0,0) إلى (row,col) وتخزينه في المتغير roi (حيث row و col هما أبعاد صورة شعار OpenCv).

```
# Now create a mask of logo and create its inverse mask also
img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
ret, mask = cv2.threshold(img2gray, 10, 255, cv2.THRESH_BINARY)
mask_inv = cv2.bitwise_not(mask)
```

الآن سننشأ قناع(mask) لصورة شعارOpenCv.

(الغاية هنا من إنشاء قناع هو تحديد المنطقة المكتوب فيها شعار OpenCvفقط دون تحديد الخلفية) لعمل قناع للشعار يجب علينا القيام بما يلي:

١. تحويل الصورة التي سننشأ لها القناع (هنا هي صورة الشعار) للمستوي الرمادي.
٢. ثم القيام باستدعاء تابع أسمه تابع التعتيب (threshold) الذي سنقوم من خلاله بقراءة لون معين فقط وذلك حسب مجال الألوان الذي نريد إظهاره، هنا سنختار مجال اللون الأسود (سيتم شرح تابع التعتيب في الفصول اللاحقة)
٣. في النهاية سنقوم بعكس التحديد لأننا لا نريد قناع اللون الأسود الذي حول الشعار وإنما نريد إنشاء قناع للشعار فقط لذلك قمنا بعملية العكس عن طريق التابع cv2.bitwise_not () (لو لم نقم بعملية العكس ستلاحظ أن الشعار أصبح مجوف وسيظهر اللون الأسود الذي حول الشعار)



تابع التعتيب cv2.threshold() وظيفته باختصار إظهار مجال لوني معين في الصورة، ويأخذ هذا التابع المتغيرات التالية:

المتغير الأول هو أسم الصورة المطلوبة، المتغير الثاني والثالث هما أعلى وأدنى قيمة للعتية اللونية المطلوبة (في مثالنا نحدد مجال عتبة اللون الأسود)، والمتغير الرابع هو نوع التعتيب.

هنا استعملنا تابع التعتيب لإظهار اللون الأسود فقط.

```
# Now black-out the area of logo in ROI
img1_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)
```

الآن نقوم بعملية تعتيم المنطقة التي نريد وضع الشعار فيها وذلك عن طريق عملية and.

```
# Take only region of logo from logo image.
img2_fg = cv2.bitwise_and(img2,img2,mask = mask)
```

هذا الكود يقوم باستخراج منطقة الشعار فقط دون المنطقة السوداء (الخلفية) وتخزينها في المتغير img2_fg.

```
# Put logo in ROI and modify the main image
dst = cv2.add(img1_bg, img2_fg)
img1[0:rows, 0:cols ] = dst
```

الآن سنقوم بالخطوة النهائية وهي إظهار الشعار في المكان الذي حددناه في صورة الملعب وذلك عن طريق عملية الجمع كما هو واضح في الكود، ثم قمنا بعرض نتيجة الجمع على الزأوية اليسرى العليا من الصورة.

وأخيرا نعرض النتيجة عن طريق التابع imshow().

تقنيات قياس الأداء وتحسينه

الهدف:

في معالجة الصورة نتعامل مع عدد كبير من العمليات في الثانية، ولذلك يجب عند كتابة الكود ألا نهتم فقط بكتابته بشكل صحيح فقط، وإنما يجب علينا الانتباه لكتابة الكود بحيث يتم تنفيذ البرنامج بالشكل الأسرع.

لذلك في هذ الفصل سنتعلم ما يلي:

- قياس أداء البرنامج
- بعض التحسينات لتحسين البرنامج

سنتعرف على التوابع التالية: cv2.getTickCount, cv2.getTickFrequency

بمعزل عن مكتبة OpenCv، بايثون يقدم لنا وحدة time التي تساعدنا على قياس الوقت اللازم لتنفيذ تعليمة أو مجموعة تعليمات، ووحدة profile التي تعطينا تقرير كامل عن البرنامج مثل: الزمن المستغرق لتنفيذ كل تابع – عدد المرات التي تم استدعاء فيها كل تابع .....الخ.

قياس الأداء عن طريق مكتبة OpenCv

التابع cv2.getTickCount يعطينا عدد دورات الساعة (clock-cycles) التي تم استغراقها لتنفيذ أمر أو عدة أوامر في البرنامج.

التابع cv2.getTickFrequency يعطينا تردد دورات الساعة(clock-cycle)، أو عدد دورات الساعة في الثانية.

لإيجاد زمن تنفيذ الكود بالثواني عليك أن تكتب كما يلي:

```
import cv2
import numpy as np
e1 = cv2.getTickCount()

# your code execution

e2 = cv2.getTickCount()
time = (e2 - e1)/ cv2.getTickFrequency()
```

مثال:

```
import cv2
import numpy as np

img1 = cv2.imread(' brid.jpg')

e1 = cv2.getTickCount()

for i in xrange(5,49,2):
   img1 = cv2.medianBlur(img1,i)

e2 = cv2.getTickCount()

t = (e2 - e1)/cv2.getTickFrequency()
print t
# Result I got is 0.521107655 seconds
```

ملاحظة: يمكنك قياس الوقت أيضا عن طريق الوحدة **time**، بحيث يحسب فرق الوقت بين استدعاء التابع في أول مرة وعند استدعائه في المرة الثانية **time.time()**.

١. الحل المثالي الافتراضي في OpenCv:

تتضمن OpenCvالعديد من التوابع التي يمكن تنفيذها بشكل مثالي، فلذلك إذا كان لدينا نظام يدعم خاصية جعل الكود مثالي أم لا عندها يجب استغلال هذه الخاصية (تقريبا معظم المعالجات الجديدة تدعم هذه الخاصية) وتكون مفعلة عند تنفيذ البرنامج بشكل افتراضي.

لمعرفة هل زمن تنفيذ الكود مثالي أم لا نقوم باستدعاء التابع cv2.useOptimized() الذي يعيد لنا قيمة TRUE إذا كانت خاصية الكود المثالي ممكنة وإلا سيعطينا FALS
كما يمكننا تفعيل وإلغاء تفعيل هذه الخاصية عن طريق التابع cv2.setUseOptimized().
شاهد الكود التالي:

```
# check if optimization is enabled

cv2.useOptimized()
```

TRUE

```
%timeit res = cv2.medianBlur(img,49)
```

10 loops, best of 3: 34.9 ms per loop

```
# Disable it
cv2.setUseOptimized(False)
```

```
cv2.useOptimized()
```

FALSE

```
%timeit res = cv2.medianBlur(img,49)
```

10 loops, best of 3: 64.1 ms per loop

كم شاهدنا في الأكواد السابقة بأنه عند تفعيل خاصية المثالية في الكود سيكون أسرع بحوالي الضعفين من الكود الذي نلغي تفعيل هذه الخاصية فيه.

٢. قياس الأداء في IPython:

أحيانا نحتاج بأن نقيس أداء عمليتين متشابهتين.
بيئة IPythonتزودنا بهذا الأمر السحري عن طريق التعليمة %timeit.
هذه التعليمة تشغل البرنامج عدة مرات لتعطينا أدق وأفضل نتيجة.
وهذه الطريقة مناسبة أيضا للتعليمات المفردة.
مثلا هل تعرف أي من عمليات الجمع الحسابية أفضل

؟ x = 5; y = x**2, x =5; y = x*x, x = np.uint8([5]); y = x*x or y = np.square(x)

هذا ما سنعرفه من خلال الأمر التالي:

```
x = 5
%timeit  y=x**2
```

10000000 loops, best of 3: 73 ns per loop

```
%timeit y=x*x
```

10000000 loops, best of 3: 58.3 ns per loop

```
z = np.uint8([5])
%timeit y=z*z
```

1000000 loops, best of 3: 1.25 us per loop

```
%timeit y=np.square(z)
```

1000000 loops, best of 3: 1.16 us per loop

يمكننا الاستنتاج بان عملية الضرب y=x*x أسرع بحوالي 20مرة مقارنة مع Numpy
وإذا أخذنا بعين الاعتبار إنشاء المصفوفات فقد يصل لحولي سرعة أعلى بـ 100 مرة، شيء غريب أليس كذلك؟ (هذه الثغرة في مكتبة Numpyيجري العمل على إصلاحها)

- ✓ عمليات الأرقام في بايثون أسرع من العمليات التي تتم في **Numpy** بالنسبة للعمليات التي تتضمن عنصر أو عنصرين.
- ✓ استعمال **Numpy** يكون له فائدة عند ازدياد حجم المصفوفة قليلا.

سنقوم الآن بتجريب مثال آخر. في هذه المرة سنقارن أداء التابعين cv2.countNonZero()، np.count_nonzero() لنفس الصورة.

```
%timeit z = cv2.countNonZero(img)
```

100000 loops, best of 3: 15.8 us per loop

```
%timeit z = np.count_nonzero(img)
```

1000 loops, best of 3: 370 us per loop

سنلاحظ أن توابع cv2أسرع بحوالي 25 مرة من توابع Numpy

ملاحظة: عادة زمن تنفيذ توابع **OpenCv**أسرع بكثير من الزمن المستغرق في **Numpy**وذلك لنفس العملية، لكن هناك استثناءات وذلك عند استعمال مكتبة **Numpy**للعرض بدل النسخ.

الأوامر السحرية لـ IPython:

يوجد هناك العديد من الأوامر السحرية لقياس الأداء وأوامر لإعطاء لمحة مختصرة عن الكود وأمر لإعطاء لمحة عن أي سطر من الكود، وأمر لقياس حجم الذاكرة التي استهلكها البرنامج.

٣. الخطوات التي يجب اتباعها للحصول على الأداء المثالي:

- ✓ عند كتابة البرنامج عليك أولا أن توجد الشكل الذي ستطبق به الخوارزمية بشكل بسيط ثم توضح عملها بشكل صحيح، ثم توجد المشاكل التي فيها، ثم توجد أمثل (أفضل) حل لتكتب به الخوارزمية.
- ✓ تجنب استخدام الحلقات بمايثون قدر الإمكان وخصوصا الحلقات الثنائية والثلاثية
- ✓ (double/triple loops)نظرا لبطئها
- ✓ استفد من المتغيرات المعرفة مسبقا
- ✓ لا تنسخ المصفوفات مالم تحتج لها، وحأول استخدام العرض بدلاً عن نسخها لأن نسخها عملية مكلفة للوقت
- ✓ حتى بعد تطبيق الخطوات السابقة إن بقي برنامجك بطيئاً أو لم يعد ممكن تعديله، عندها يمكنك التسريع باستخدام مكتبات مثل cythonلجعل البرنامج أسرع.

الفصل الثالث

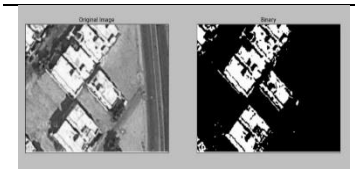
" هناك اختياران مبدئيان في الحياة: إما أن تتقبل ظروف الوضع الحالي كما هي، أو أن تتقبل مسؤولية تغيير هذا الوضع "

د. دينيس ويتلي - كاتب ومحاضر أمريكي

الفصل الثالث: معالجة الصورة في OpenCv

❖ تغيير الفضاءات اللونية Changing ColorSpaces

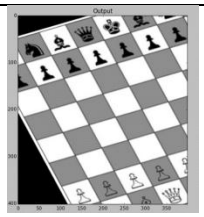
- ستتعلم طريقة تغيير فضاء الصورة لمختلف الفضاءات اللونية
- تعلم تعقب جسم ملون في الفيديو



❖ تعتيب الصورة (التعتيب: هو عملية إظهار عتبة لونية معينة)
Image Thresholding

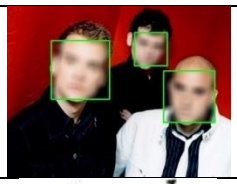
ستتعلم كيف تحول الصورة لصورة ثنائية (فقط أبيض وأسود) وذلك باستخدام إحدى الطرق:
التعتيب العام global thresholding، التعتيب المتكيف Adaptive thresholding، التحويل الثنائي لا وتسو Otsu's banalization

---



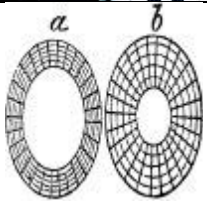
❖ التحويلات الهندسية على الصور Geometric Transformations of Images

ستتعلم كيف تطبيق مختلف التحويلات الهندسية على الصورة مثل الدوران وتحليل الصورة و .....الخ



❖ تهذيب الصورة Smoothing Images

تعلم كيفية طمس الصورة، فلترة الصورة



❖ التحويلات من الناحية الشكلية (Morphological)
Morphological Transformations:

التعرف على التحويلات الشكلية التي تتم على الشكل مثل التآكل، التمدد، الفتح، الإغلاق

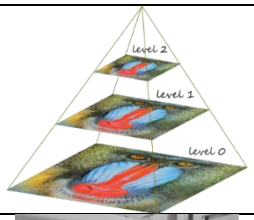
تدرجات الصورة Image Gradients:

تعلم إيجاد تدرجات الصورة وإيجاد حوافها.

❖ مكتشف حواف كاني Canny Edge Detection

- تعلم إيجاد الحواف (حدود الشكل) مع مكتشف حواف كاني

- أهرامات الصورة Image Pyramids:

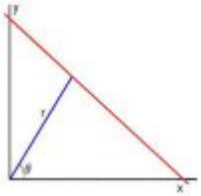


التعرف على أهرامات الصورة وطريقة استخدامها لمزج الصورة

- مقارنة القوالب Template Matching

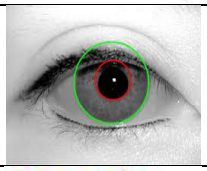
تعلم البحث عن أي كائن في الصورة باستخدام مطابقة القوالب

- تحويل هاف للخط Hough Line Transform

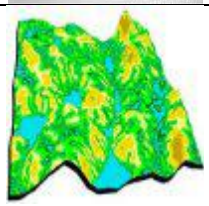


تعلم اكتشاف الخطوط التي في الصورة.

- تحويل هاف للدوائر Hough Circle Transform
- تعلم اكتشاف عن الدوائر في الصورة.



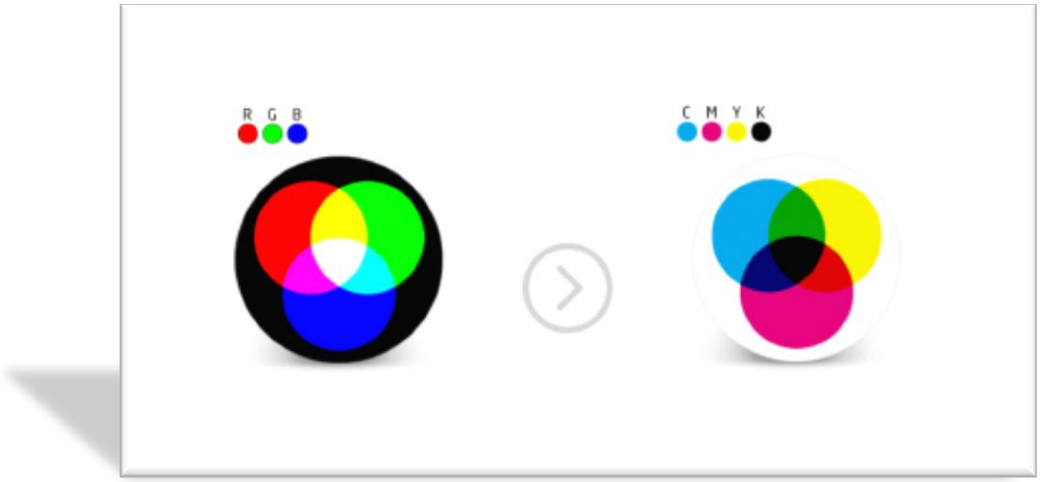
- تقطيع الصورة باستخدام خوارزمية Watershed:
- تعلم تقسيم الصورة بطريقة المجمعات المائية(Watershed)



الاشتقاق الأمامي التفاعلي باستخدام خوارزمية GrabCut

تعلم خوارزمية GrabCutلاستخراج الأجسام الأمامية في الصورة.

تغيير الفضاءات اللونية Changing Color spaces



الهدف:

- في هذا الفصل سنتعلم طريقة التحويل من فضاء لوني إلى آخر، مثلاً منBGR ↔ Gray ومن BGR↔HSV
- بالإضافة إلى ذلك سننشئ تطبيق يستخرج جسم معين في الفيديو
- تعلم استخدام التوابع التالية: cv2.cvtColor()،cv2.inRange()

١- تغيير الفضائي اللوني

هناك أكثر من ١٥٠ طريقة لتغيير الفضاءات اللونية متاحة في OpenCv. ولكن سندرس فقط التحويل من الفضائيين اللونيين الأكثر انتشاراً وهما:

BGR↔Gray وBGR↔HSV

لتغير الفضاء اللوني سنستخدم التابع cv2.cvtColor(input_image, flag) حيث قيمة المتغير flagهي التي سوف تحدد لنا نوع التحويل. مثلا للتحويل من BGR إلى GRAY تكون قيمة المتغير flagهي cv2.COLOR_BGR2GRAY وللتحويل من BGR إلى HSV نستعمل المتغير cv2.COLOR_BGR2HSV

لمعرفة قيم الفضاءات الأخرى التي يأخذها المتغير flag شغل الكود التالي:

```
import cv2
flags = [i for i in dir(cv2) if i.startswith('COLOR_')]
for x in flags:
    print x
```

```
COLOR_BAYER_BG2BGR
COLOR_BAYER_BG2BGR_VNG
COLOR_BAYER_BG2GRAY
COLOR_BAYER_BG2RGB
COLOR_BAYER_BG2RGB_VNG
COLOR_BAYER_GB2BGR
COLOR_BAYER_GB2BGR_VNG
COLOR_BAYER_GB2GRAY
COLOR_BAYER_GB2RGB
COLOR_BAYER_GB2RGB_VNG
COLOR_BAYER_GR2BGR
COLOR_BAYER_GR2BGR_VNG
COLOR_BAYER_GR2GRAY
.....

etc...
```

ملاحظة: في الفضاء **HSV** مجال السطوع هو **[0 , 179]**، مجال الإشباع **[0 , 255]**، مجال القيم **[0 , 255]**.
ومجالات القيم هذه قد تختلف من برنامج لآخر، لذلك إذا قارنت قيم**OpenCv**بغيرها ستحتاج للانتباه لطبيعية هذه المجالات.

٢- تعقب الأجسام

تعلمنا للتو كيف نحول الصورة من الفضاء BGR إلى الفضاء HSV، والآن يمكننا استعمال هذه الخاصية في استخراج جسم ملون. في الفضاء HSV يمكن تمثيل الألوان بسهولة أكبر من الفضاء BGRويساعدنا على تمييز الألوان في ظروف الإضاءة المختلفة لأنه يهتم بالسطوع. في تطبيقنا التالي سنقوم باستخراج جسم ملون أزرق من فيديو وذلك عن طريق عمل ما يلي:

١) نأخذ كل إطار من الفيديو
٢) نحول من BGR إلى HSV
٣) نقوم بتعتيب الصورة المحولة وذلك عن طريق أخذ عتبة اللون المطلوب (هنا سنأخذ عتبة اللون الأزرق)

٤) سنحصل بالنهاية على الجسم الملون بالأزرق لوحده، وعند ذلك يمكنك استخدام هذه النتيجة في التطبيق الذي تريده.

الكود التالي يوضح هذه الخطوات:

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while(1):
    # Take each frame
    _, frame = cap.read()

    # Convert BGR to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # define range of blue color in HSV
    lower_blue = np.array([110,50,50])
    upper_blue = np.array([130,255,255])

    # Threshold the HSV image to get only blue colors
    mask = cv2.inRange(hsv, lower_blue, upper_blue)

    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(frame,frame, mask= mask)

    cv2.imshow('frame',frame)
    cv2.imshow('mask',mask)
    cv2.imshow('res',res)
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break
cv2.destroyAllWindows()
```

شرح الكود

```
cap = cv2.VideoCapture(0)
```

استدعاء تابع التقاط الفيديو وتخزين إطارات الصور (الفريمات Frames) التي تكون الفيديو في المتغير cap

```
# Take each frame
   _, frame = cap.read()
```

- الخطوة الأولى: نحصل على كل إطار في الفيديو.

التابع cap.read() يعطينا إطارات الصور الملتقطة ، وضعنا رمز الشحطة (_) في المتغير الأول لأننا لا نريده ونريد فقط استعمال المتغير الثاني frame الذي ستخزن بداخله إطار الصورة الملتقط.

(تم شرح هذا الأمر مسبقاً في الفصل الثاني).

```
   # Convert BGR to HSV
   hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

- الخطوة الثانية: تحويل الصور الملتقطة من الفضاء اللوني BGR إلى HSV.
- الخطوة الثالثة: هي تعتيب الصورة (أي سنأخذ مجال لوني معين لإظهاره لوحده)

بمعنى أخر نقصد بالتعتيب هنا هو عمل قناع لمجال لوني معين في الصورة بحيث سيظهر هذا اللون بالأبيض وأي لون خارج المجال اللوني الذي اخترناه سيكون باللون الأسود. وذلك عن طريق التعليمات التالية:

```
   # define range of blue color in HSV
   lower_blue = np.array([110,50,50])
   upper_blue = np.array([130,255,255])
```

نقوم بتعريف حدي (عتبتي) المجال اللوني الذي نريده في الفضاء HSV (هنا اخترنا مجال اللون الأزرق)

قمنا بأخذ أقل عتبة لونية لمجال اللون الأزرق وخزناها في المتغير lower_blue

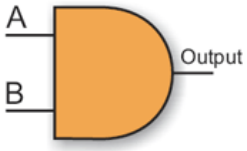
وأخذنا أكبر عتبة لونية لمجال اللون الأزرق وخزناها في المتغير upper_blue

```
mask = cv2.inRange(hsv, lower_blue, upper_blue)
```

قمنا بتخزين المجال اللوني (بين قيمتي العتبتين التي اخترناهما) الذي نريده في المتغير mask وذلك عن طريق التابع cv2.inRange().

```
# Bitwise-AND mask and original image
   res = cv2.bitwise_and(frame,frame, mask= mask)
```

نقوم بعمل عملية and ليظهر المجال اللوني الذي اخترناه فقط دون المجالات اللونية الأخرى.



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

في مثالنا هذا مثلنا اللون الأزرق بالواحد فإذا كان اللون الذي في الصورة بنفس قيمة المجال اللون الأزرق سيعطينا١

(أزرق AND أزرق = 1) (1 AND 1 = 1)

وأي لون ليس ضمن المجال اللوني الأزرق سيكون عندها ناتج عملية AND هو ٠ (اللون الأسود).

```
cv2.imshow('frame',frame)
cv2.imshow('mask',mask)
cv2.imshow('res',res)
```

قم بعرض الصور الملتقطة (سيتم عرضها على شكل فيديو بسبب سرعة تحديث التقاط الصور العالية)، وعرض القناع، وعرض الجسم الأزرق فقط دون غيره من الألوان. لاحظ بأن هناك بعض الضجيج في الصورة، سنرى في الفصول القادمة كيف سنزيل هذا الضجيج.

ملاحظة: هذا أبسط طريقة لتعقب جسم، وعندما تحترف توابع الإطارات **contours** سيكون إمكانك القيام بالعديد من الأشياء مثل إيجاد مركز الجسم، تعقب الأجسام، رسم رسومات بمجرد تحريك يدك أمام الكاميرا، وتعلم العديد من الأشياء الممتعة في الفصول القادمة.

كيف يمكننا إيجاد القيمة اللونية المطلوب تعقبها في الفضاء HSV؟

هذا السؤال يعد من الأسئلة الشائعة التي تُسأل على الموقع المهور stackoverflow.com.

بكل بساطة يمكنك معرفة القيمة اللونية المطلوبة ضمن HSVوذلك عن طريق التابع الذي استعملناه سابقاً cv2.cvtColor() لكن بدل تمرير قيمة الصورة للتابع نمرر له القيمة اللونية التي نريدها في BGR. على سبيل المثال لإيجاد قيمة اللون الأخضر في HSV نكتب الأوامر التالية:

```
green = np.uint8([[[0,255,0 ]]])
hsv_green = cv2.cvtColor(green,cv2.COLOR_BGR2HSV)
print hsv_green
```

```
[[[ 60 255 255]]]
```

الآن يمكنك أخذ أدنى قيمة وأعلى قيمة لمجال اللون الأخضر باتباع القاعدة التالية:

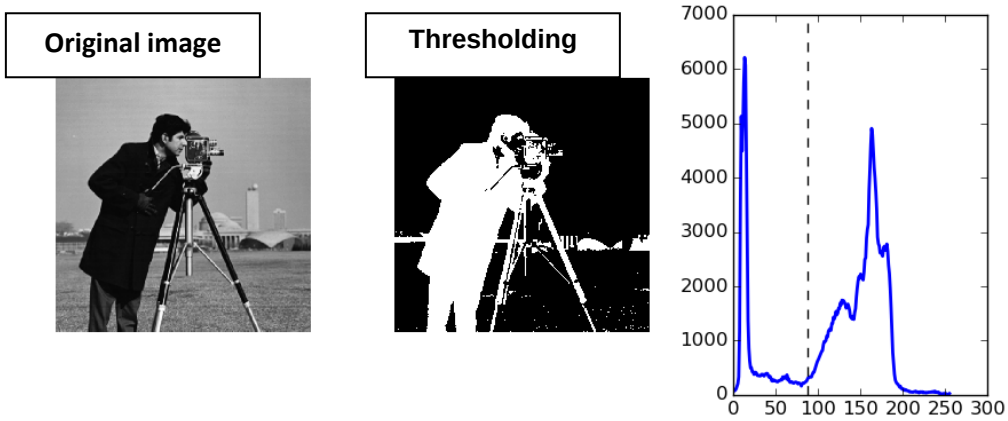
[H-10, 100,100] and [H+10, 255, 255]

القاعدة ([H-10]، [H+10]) لا تعمل دائماً معنا، لذلك يفضل أن تحدد يدويا بنفسك أعلى وأدنى قيمة عن طريق تمرير أدنى قيمة لعتبة للون الأخضر في RGB وأعلى قيمة لعتبة للون الأخضر في RGB.

تمرين: جرب بنفسك استخراج أكثر من جسم ملون بنفس الوقت مثلا الأجسام الزرقاء والخضراء والحمراء.

تعتيب الصورة Image Thresholding

التعتيب: هو إظهار عتبة لونية معينة عندما نقول عتبنا الصورة أي أظهرنا مستوى لوني محدد في الصورة



الهدف

- في هذا الدرس سندرس التعتيب البسيط، والتعتيب المتكيف، وتعتيب أوتسو.
- سنتعلم استخدام التوابع التالية: cv2.threshold, cv2.adaptiveThreshold

١- التعتيب البسيط (Simple Thresholding):

هذه الطريقة بسيطة، عندما تفوق قيمة البيكسل قيمة محددة، يسند له 1(ابيض غالبا)، وإلا يسند له 0(غالبا اللون الأسود) ، التابع المستخدم لهذه العملية هو(cv2.threshold.

شرح التابع cv2.threshold():

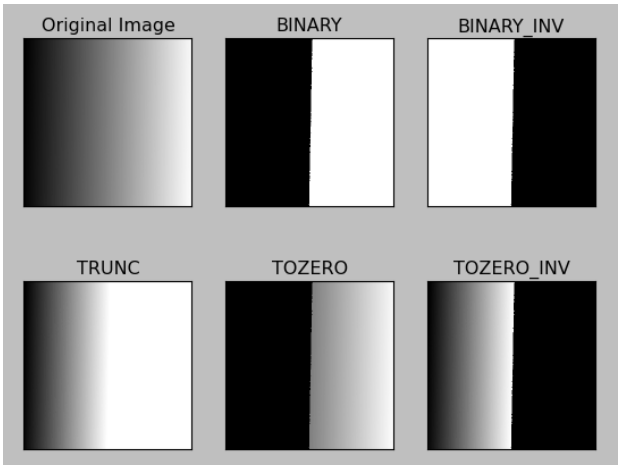
المتغير الأول: يمثل الصورة المطلوبة، ويجب أن تكون هذه الصورة بالمستوى الرمادي.

المتغير الثاني والثالث: هما أدنى وأعلى قيمة للعتبة التي ستصنف البيكسلات من خلالها.

المتغير الرابع: تقدم مكتبة OpenCvلنا عدة خيارات للتعتيب ويتم تحديد ذلك عن طريق هذا المتغير

cv2.THRESH_BINARY
cv2.THRESH_BINARY_INV
cv2.THRESH_TRUNC

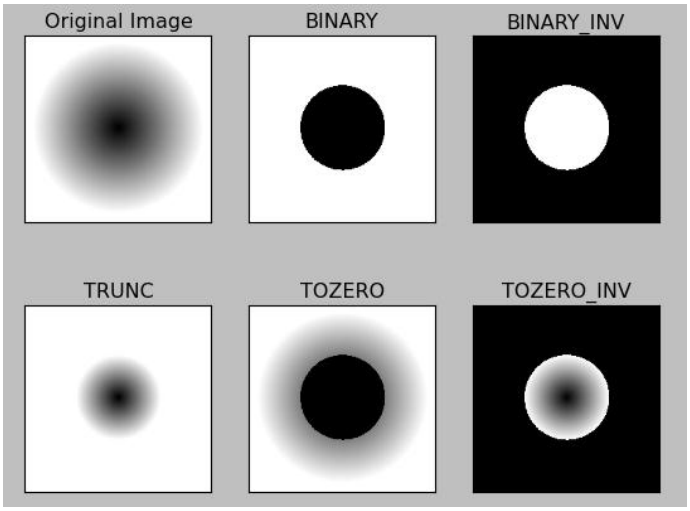
cv2.THRESH_TOZERO
cv2.THRESH_TOZERO_INV



سنقوم بتطبيق كل هذه الأنواع في الكود التالي:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('gradient.bmp',0)
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in xrange(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])

plt.show()
```

النتيجة:



شرح الكود:

```
img = cv2.imread('gradient.bmp',0)
```

قراءة الصورة وتخزينها في المتغير img

```
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)
```

قمنا بتعتيب الصورة باستخدام تابع التعتيب البسيط cv2.threshold()(تم شرحه سابقاً)

التابع cv2.threshold() يعيد نتيجتين:

النتيجة الأولى هي retval سيتم شرحها لاحقا في قسم (التحويل الثنائي لا وتسو).

والنتيجة الثانية هي threshold الصورة التي تم تعتيبها.

```
titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
```

إعطاء أسماء للنوافذ التي سيتم فتحها وخزنا هذه الأسماء في مصفوفة سميناها titles

```
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
```

تخزين قيمة الصورة ونتيجة التعتيب الأولى والثانية والثالثة والرابعة والخامسة في مصفوفة سميناها images

```
for i in xrange(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
```

- قمنا بإنشاء حلقة for حتى نظهر عن طريقها النوافذ التي ستعرض لنا نتائج التعتيب.
- **Plt.subplot** تابع في مكتبة matplotlib وظيفته إظهار أكثر من صورة في نافذة واحدة (أول متغير وثاني متغير هما حجم النافذة، وثالث متغير يحدد عدد النوافذ)
- **Plt.imshow()** تابع لتهيئة عرض الصور المطلوبة.
- **Plt.title()** لإعطاء أسماء الصور المطلوبة.
- **plt.xticks([]),plt.yticks([])** هذه التعليمة لإخفاء تدرجات محور الـ x و y .

```
plt.show()
```

Plt.show() تابع لعرض النوافذ التي تم تهيئتها من قبل التابع plt.imshow() .

٢- التعتيب المتكيف (Adaptive Thresholding):

في القسم السابق استعملنا قيمة عامة لقيمة العتبة، ولكن هذا الأمر غير جيد، فكل صورة تملك شروط إضاءة مختلفة في مناطق مختلفة منها، لذلك سنستعمل التعتيب المتكيف.

التعتيب المتكيف هو خوارزمية حساب عتبة منطقة صغيرة في الصورة، لذلك سنحصل على عدة عتبات لعدة مناطق في نفس الصورة، وهذا سيعطي نتائج أفضل للصور في مختلف شروط الإضاءة.
التعتيب المتكيف يمتلك ٣ متغيرات دخل ومتغير خرج وحيد.

- طريقة التكييف هي التي ستحدد طريقة حساب قيمة العتبة، هناك طريقتان للحساب:
  - cv2.ADAPTIVE_THRESH_MEAN_C: قيمة العتبة لمتوسط مساحات مناطق الجوار.
  - cv2.ADAPTIVE_THRESH_GAUSSIAN_C: قيمة العتبة لمتوسط أوزان مجموع القيم المتجأورة، حيث أن الأوزان هي النافذة الغأوسية.
- Block Size: يحدد حجم منطقة الجوار
- C: ثابت الطرح من المتوسط الحسابي أو المتوسط الموزون، قيمته تكون موجبة عادةً ولكن يمكن أن تكون قيمته أيضا صفر أو قيمة سالبة.

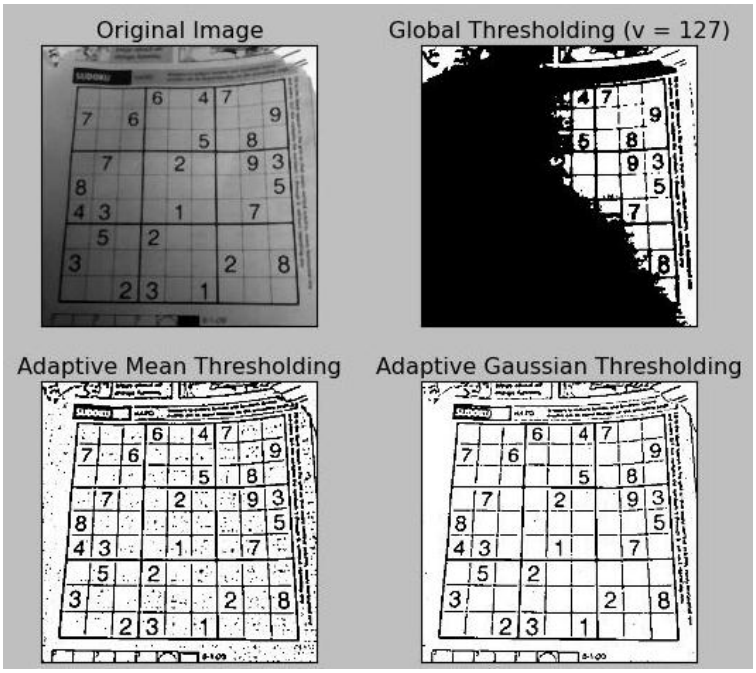
ما هو المتوسط الموزون؟ وما الفرق بين المتوسط الحسابي والمتوسط الموزون؟ (رياضيات)

لمعرفة ما هو المتوسط الموزون وما هو الفرق بينه وبين المتوسط الحساب انتقل لهذا الرابط:
http://www.mediafire.com/download/lztw213xna3n2xh

الكود التالي يوضح الفرق بين التعتيب العام والتعتيب المتكيف للصورة في شروط إضاءة مختلفة:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('dav.jpg',0)
img = cv2.medianBlur(img,5)

ret,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
        cv2.THRESH_BINARY,11,2)
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
        cv2.THRESH_BINARY,11,2)

titles = ['Original Image', 'Global Thresholding (v = 127)',
        'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]

for i in xrange(4):
   plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
   plt.title(titles[i])
   plt.xticks([]),plt.yticks([])
plt.show()
```

النتيجة:



كما هو واضح من النتائج بأن التعتيب المتكيف يعطي نتيجة أفضل وضجيج أقل.

شرح الكود:

```
img = cv2.medianBlur(img,5)
```

التابع medianBlur() يستخدم لطمس الصورة باستخدام الفلتر المتوسط.
المتغير الأول هو أسم الصورة، المتغير الثاني يمثل درجة الطمس.

```
ret,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
```

تعتيب الصورة باستخدام تابع التعتيب البسيط () threshold (تم شرحه مسبقاً).

```
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
        cv2.THRESH_BINARY,11,2)
```

تعتيب الصورة باستخدام تابع التعتيب المتكيف () adaptiveThreshold:

المتغير الأول هو الصورة المطلوب تعتيبها (يجب أن تكون الصورة في المستوي الرمادي)

المتغير الثاني يمثل أدنى قيمة للعتبة

المتغير الثالث أعلى قيمة للعتبة. في التعتيب المتكيف سنختار أحد التوابع التالية:

- cv2.ADAPTIVE_THRESH_MEAN_C
- cv2.ADAPTIVE_THRESH_GAUSSIAN_C

المتغير الرابع هو نوع التعتيب (تم شرحه مسبقا)

المتغير الخامس هو Block Size ويمثل حجم البيكسلات المتجأورة التي تستخدم لحساب قيمة عتبة البيكسل.

المتغير السادس هو الثابت C الذي يمثل عادةً ثابت الطرح من المتوسط أو المتوسط الموزون قيمته تكون عادة موجبة ولكن يمكن أن تكون قيمته أيضا صفر أو قيمة سالبة.

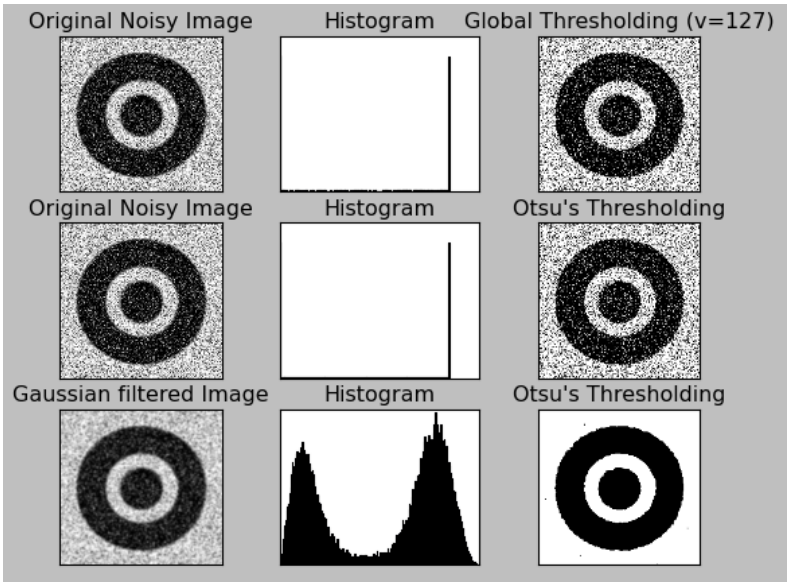
```
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
        cv2.THRESH_BINARY,11,2)
```

نفس التابع السابق مع اختلاف بسيط وهو باختيار نوع تابع المتغير الثالث.

٣- التحويل الثنائي لا وتسو (Otsu's Binarization):

لقد أوضحنا في القسم السابق بأن التابع cv2.threshold() يعيد قيمتين هما threshold وretVal، وشرحنا المتغير الثاني threshold، ولكن المتغير الأول retVal لم نشرحه، فما هو عمله؟؟

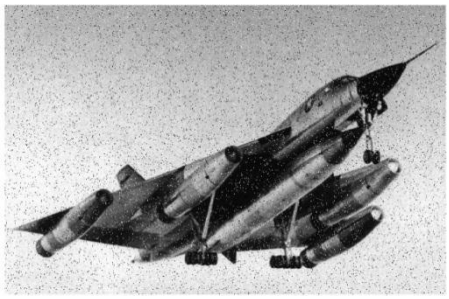
في التعتيب البسيط قمنا باختيار قيمة عشوائية للعتبة، ثم عن طريق التجريب نحدد القيم المثالية للتعتيب، ولكن إذا كان لدينا صورة ثنائية الصيغة (المقصود بالصورة ثنائية الصيغة هي الصورة التي تملك قمتين في المخطط البياني لها) ، سنأخذ قيمة عتبة وسطة بين هاتين القمتين وهذا ما يفعله تحويل أوتسو.



ملخص:

تحويل أوتسو وظيفته حساب قيمة العتبة للصور ثنائية الصيغة (الصورة التي تملك قمتين في المخطط البياني لها).بالنسبة للصور غير الثنائية تحويل أوتسو لن يكون دقيقاً.

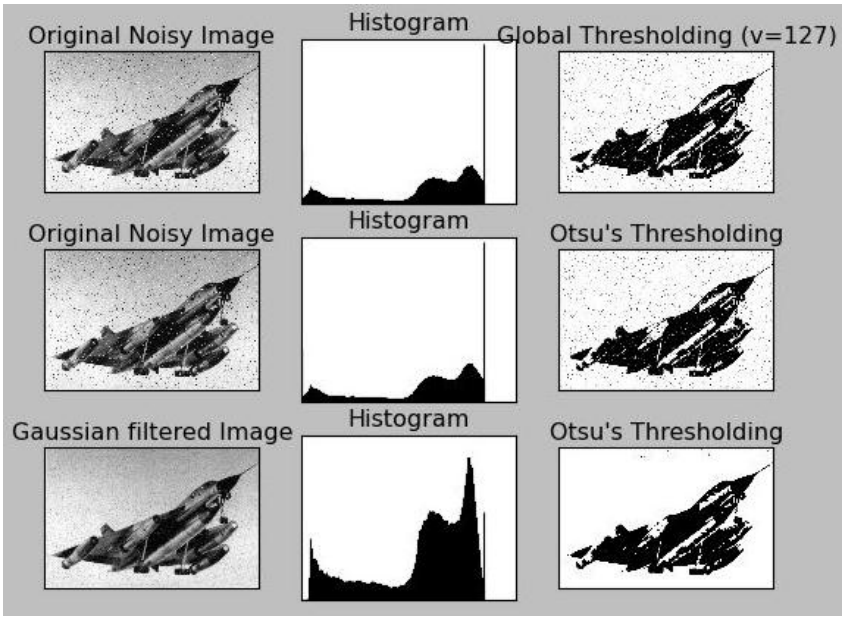
لتطبيق تحويل أوتسو نستخدم العلم cv2.THRESH_OTSU. وللحصول على قيمة العتبة ببساطة قم بتمرير القيمة صفر، عندها الخوارزمية ستحسب قيمة العتبة المثالية وتعيد قيمتها وتخزنها في المتغير retVal. تعتيب أوتسو إذا لم يُستَعمل، عندها ستكون قيمة retVal بنفس قيمة العتبة المستخدمة.



في المثال التالي صورة الدخل ذات ضجيج
في الحالة الأولى سنطبق على الصورة التعتيب العام (التعتيب البسيط)
في الحالة الثاني نطبق التعتيب حسب أوتسو
في الحالة الثالثة سنقوم أولا بإزالة الضجيج من الصورة عن طريق تطبيق مرشح غأوسي بقياس 5x5 ثم سنطبق ترشيح أوتسو.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('noisy.JPG',0)

# global thresholding
ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)

# Otsu's thresholding
ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

# Otsu's thresholding after Gaussian filtering
blur = cv2.GaussianBlur(img,(5,5),0)
ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

# plot all the images and their histograms
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
      'Original Noisy Image','Histogram',"Otsu's Thresholding",
      'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]

for i in xrange(3):
   plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
   plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
   plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
   plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
   plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
   plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```

النتيجة:



شرح الكود:

```
# global thresholding
ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
```

تعتيب الصورة وفق تابع التعتيب البسيط.

```
# Otsu's thresholding
ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

تعتيب الصورة وفق تعتيب أوتسو

```
blur = cv2.GaussianBlur(img,(5,5),0)
```

سنزيل الضجيج من الصورة عن طريق تطبيق مرشح غأوسي بحجم 5x5. (سيتم شرحه لاحقا)

```
ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

تطبيق تعتيب أوتسو

```
# plot all the images and their histograms
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
```

تخزين الصورة وتعتيباتها ومخططاتها البيانية في مصفوفة سميناها image.

تمرين:

اكتب تطبيق صغير لتغيير قيمة العتبة لصورة ما عن طريق شريط التمرير الجانبي.

الحل:

```
import cv2
import numpy as np
def nothing(x):
  pass
img = cv2.imread('shape2.jpg',0)
#Generate trackbar Window Name
TrackbarName='value threshold'

#Make Window and Trackbar
cv2.namedWindow('Window')
cv2.createTrackbar(TrackbarName,'Window',0,255,nothing)

# Allocate destination image
Threshold = np.zeros(img.shape, np.uint8)

# Loop for get trackbar pos and process it
while True:
  #Get position in trackbar
  TrackbarPos = cv2.getTrackbarPos(TrackbarName,'Window')

  #Apply Threshold
  cv2.threshold(img,TrackbarPos,255,cv2.THRESH_BINARY, Threshold)

  # Show in window
  cv2.imshow('Window',Threshold)

  # If you press "ESC", it will return value
  ch = cv2.waitKey(5)
  if ch == 27:
    break

cv2.destroyAllWindows()
```

التحويلات الهندسية على الصورة

الهدف:

- تعلم تطبيق مختلف التحويلات الهندسية على الصورة، مثل النقل، التدوير، تحويلات التقريبaffine.
- سنتعلم استخدام التابع cv2.getPerspectiveTransform

عمليات التحويل

OpenCvيعطينا تابعيين للتحويل هما cv2.warpAffine،cv2.warpPerspective .
عن طريق هذين التابعيين يمكنك إيجاد كل أنواع التحويل.

- التابع cv2.warpAffineيملك مصفوفة تحويل حجمها 2x3.
- التابع cv2.warpPerspective يملك مصفوفة تحويل بحجم 3x3.

- **التحجيم Scaling**

هي عملية تحديد حجم (أبعاد) الصورة، وذلك عن طريق التابع cv2.resize(). حيث يمكن تحديد حجم الصورة يدوياً ويمكن تحديده عن طريق عامل التحجيم. وتستخدم أساليب الاستيفاء المختلفة.

أفضل أساليب الاستيفاء هي cv2.INTER_AREA للتقلص، وcv2.INTER_CUBIC (slow) للتقريب، وcv2.INTER_LINEAR للتبعيد. بشكل افتراضي يكون أسلوب الاستيفاء هو cv2.INTER_LINEARأي التبعيد.

الكود التالي يوضح ما تحدثنا عنه:

```
import cv2
import numpy as np
img = cv2.imread('gift.jpg')

res = cv2.resize(img, None, fx=2, fy=2, interpolation = cv2.INTER_CUBIC)

print img.shape
print res.shape
```

(٣٢٠، ٣٢٠، 3)

(٦٤٠، ٦٤٠، 3)

يمكن كتابة الكود السابق بشكل آخر:

```
#OR
height, width = img.shape[:2]
res = cv2.resize(img,(2*width, 2*height), interpolation = cv2.INTER_CUBIC)
```

(٣٢٠، ٣٢٠، 3)

(٦٤٠، ٦٤٠، 3)

- **النقل Translation**

النقل هو عملية إزاحة جسم ما. للقيام بعملية النقل نتبع الخطوات التالية:

نحدد مقدار الانزياح بالإحداثيات (t_x, t_y) ثم نقوم بإنشاء مصفوفة M ونمرر لها مقدار الانزياح الذي نريده $M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$

ثم نقوم بتحويل هذه المصفوفة للصيغة np. float32 ونمررها للتابع cv2.warpAffine().

شاهد المثال التالي الذي يوضح انزياح الصورة (المصفوفة) بمقدار (100,50):

```
import cv2
import numpy as np

img = cv2.imread('gazal.jpg',0)

rows,cols = img.shape
M = np.float32([[1,0,100],[0,1,50]])
dst = cv2.warpAffine(img,M,(cols,rows))

cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:

شرح الكود:

```
rows,cols = img.shape
```

تخزين عدد صفوف وأعمدة الصورة img في المتغيرين rows وcols .

```
M = np.float32([[1,0,100],[0,1,50]])
```

تعريف مصفوفة $M = \begin{bmatrix} 1 & 0 & 100 \\ 0 & 1 & 50 \end{bmatrix}$ من النوع np. float32
سنستخدم هذه المصفوفة لتحديد انزياح الصورة بالمقدار (100,50).

```
dst = cv2.warpAffine(img,M,(cols,rows))
```

جعل الصورة تنزاح بمقدار (100,50) عن طريق تمرير المصفوفة M للتابع cv2.warpAffine().
المتغير الثالث للتابع cv2.warpAffine() يحدد قياس صورة الخرج.

- **التدوير Rotation**

للقيام بعملية تدوير الصورة بزأوية θ نقوم بما يلي:

ننشأ مصفوفة من الشكل $M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

مكتبة OpenCv توفر لنا مصفوفة لتدوير الصورة حول أي نقطة وبأي زأوية نريدها. وذلك من خلال التابع cv2.getRotationMatrix2D.

المثال التالي يدور لنا الصورة بزأوية 90° درجة دون تغيير حجم الصورة.

```
import cv2
import numpy as np

img = cv2.imread('gazal.jpg',0)

rows,cols = img.shape
M = cv2.getRotationMatrix2D((cols/2,rows/2),90,1)
dst = cv2.warpAffine(img,M,(cols,rows))

cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:

شرح الكود:

```
M = cv2.getRotationMatrix2D((cols/2,rows/2),90,1)
```

التابع cv2.getRotationMatrix2Dيحدد لنا زأوية الدوران التي نريد تدوير الصورة بها.

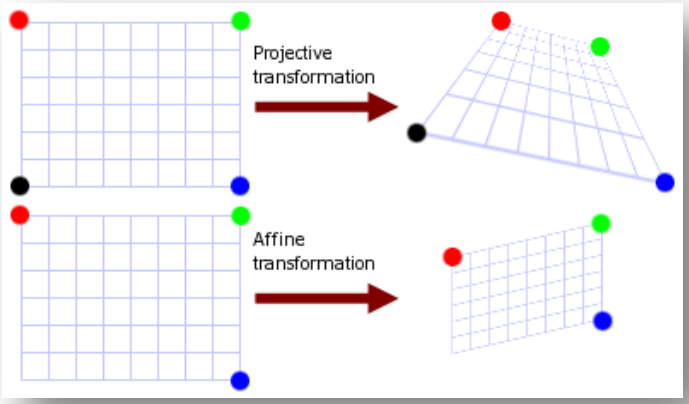
شرح التابع cv2.getRotationMatrix2D ():

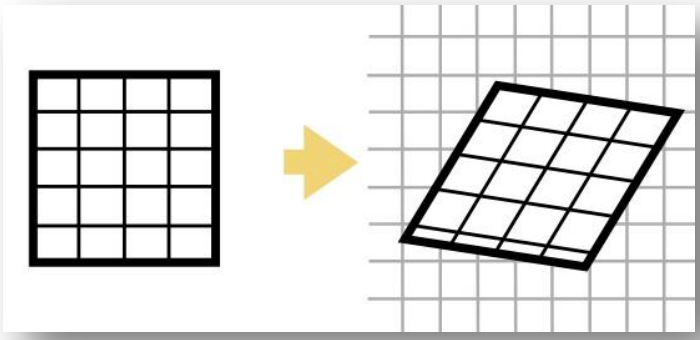
- المتغير الأول يحدد مركز دوران الصورة.
- المتغير الثاني يحدد زأوية الدوران.
- المتغير الثالث يحدد حجم صورة الخرج.

```
dst = cv2.warpAffine(img,M,(cols,rows))
```

نفس الشرح في الفقرة السابقة، ولكن هذه المرة بدل أن نمرر مصفوفة M لتغيير الحجم مررنا مصفوفة M لتحديد زأوية الدوران.

- **تحويل أفيني Affine Transformation**



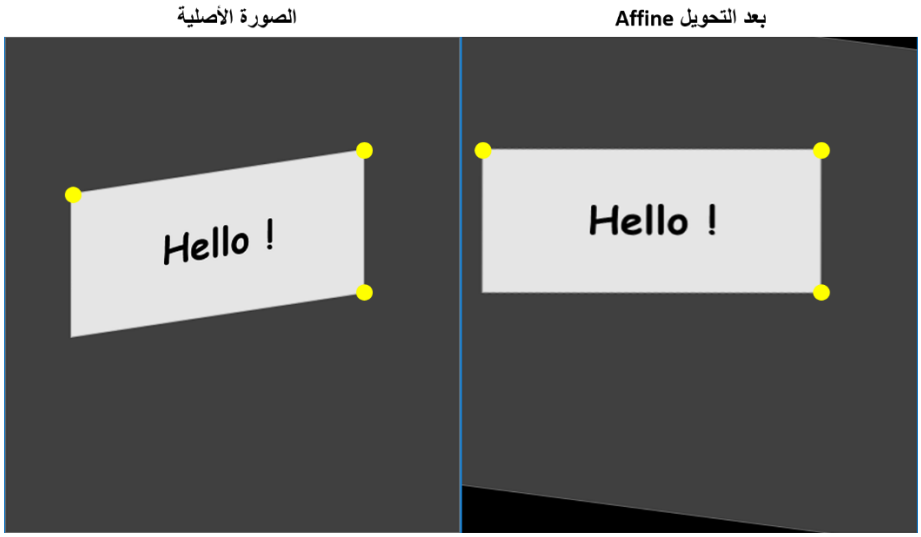


في هذا التحويل كل الخطوط المتوازية في الصورة الأصلية، ستبقى متوازية في صورة الخرج. لإيجاد مصفوفة التحويل نحتاج ثلاثة نقاط من الصورة الأصلية والنقاط المناظرة لها في صورة الخرج، ثم سيقوم التابع cv2.getAffineTransform بإنشاء مصفوفة 2x3 والتي سنمررها لتابع التحويل cv2.warpAffine.

انظر للمثال التالي، وشاهد النقاط المختارة (النقاط الملونة باللون الأصفر).

```
import cv2
import numpy as np

img = cv2.imread('hello.png',0)
rows,cols = img.shape
pts1 = np.float32([[67,160],[370,115],[371,263]])
pts2 = np.float32([[20,114],[370,115],[371,263]])
M = cv2.getAffineTransform(pts1,pts2)
dst = cv2.warpAffine(img,M,(cols,rows))

cv2.imshow('orginal',img)
cv2.imshow('res',dst)

k = cv2.waitKey(0)

cv2.destroyAllWindows()
```

النتيجة:



شرح الكود:

```
pts1 = np.float32([[50,50],[200,50],[50,200]])
```

الأرقام التي بين القوسين [ ] تمثل إحداثيات x وy للنقاط الصفراء للصورة الدخل.

```
pts2 = np.float32([[10,100],[200,50],[100,250]])
```

الأرقام التي بين القوسين [ ] تمثل إحداثيات x وy للنقاط الصفراء للصورة الخرج.

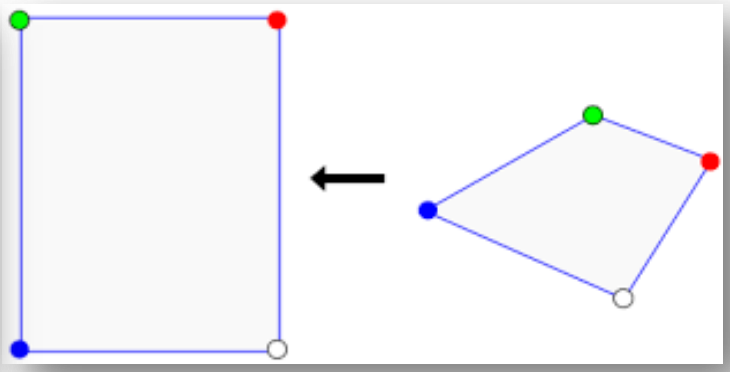
```
M = cv2.getAffineTransform(pts1,pts2)
```

الأن نقوم بحساب تحويل أفيني بين ثلاثة أزواج من النقاط الثلاثة الصفراء التي في صورة الدخل والخرج عن طريق التابع getAffineTransform ().

```
dst = cv2.warpAffine(img,M,(cols,rows))
```

وأخيراً نقوم بتطبيق تحويل أفيني الذي حسبناه على الصورة المطلوبة عن طريق التابع warpAffin (). يأخذ هذا التابع ثلاث متغيرات، الأول هو الصورة المطلوب إجراء تحويل أفيني عليها، والمتغير الثاني يمثل مقدار تحويل أفيني المطلوب، والمتغير الثالث يحدد قياس صورة الخرج.

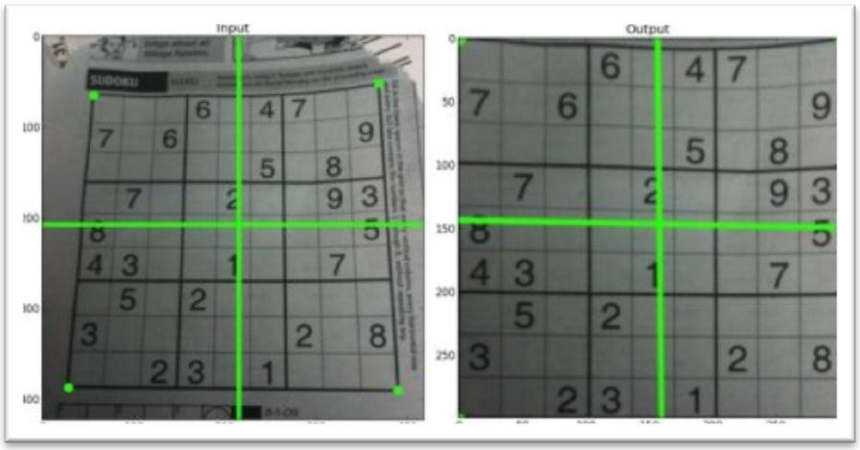
- **التحويل المنظوري (Perspective Transformation)**



للقيام بهذا التحويل سنحتاج مصفوفة 3X3، ستبقى الخطوط المتوازية كما هي بعد التحويل.
لإيجاد التحويل نحتاج أربع نقط في صورة الدخل وما يقابلها من النقاط في صورة الخرج، وثلاثة من هذه النقاط يجب ألا تكون على استقامة واحدة، عندها يمكن إيجاد مصفوفة التحويل، وذلك من خلال التابع cv2.getPerspectiveTransform، بعد ذلك نطبق التابع cv2.warpPerspective مع مصفوفة التحويل 3X3. الكود التالي يوضح ذلك:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread(' drawing.jpg',0)
rows,cols,ch = img.shape

pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])/2
pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])/2

M = cv2.getPerspectiveTransform(pts1,pts2)
dst = cv2.warpPerspective(img,M,(150,150))

plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```

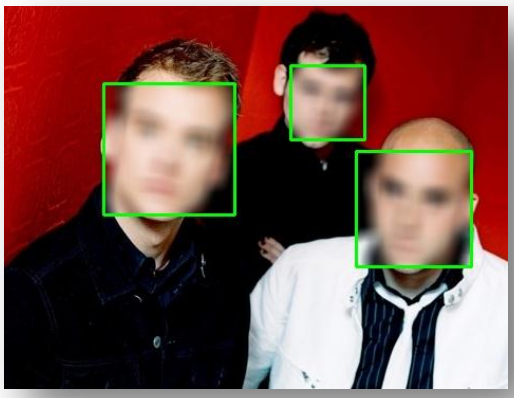
النتيجة



للمزيد من المعلومات عليك بقراءة المرجع:

Computer Vision: Algorithms and Applications " Richard Szeliski"

تهذيب الصورة (Smoothing Images)

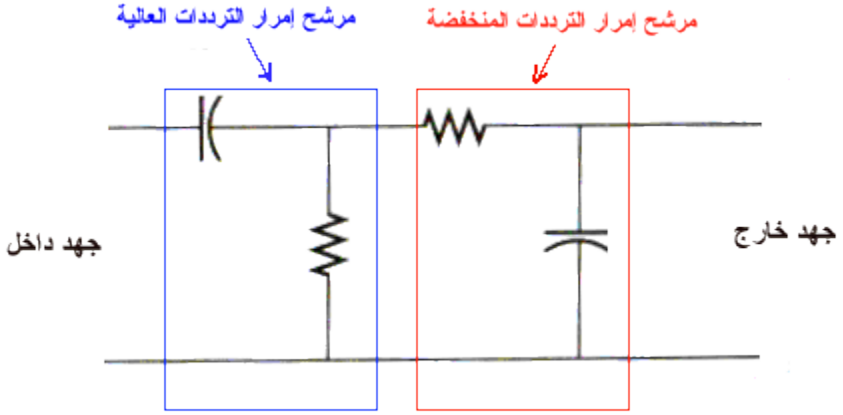


الهدف:

- تعلم كيفية تغبيش الصورة باستخدام عدة مرشحات للتردد منخفض(LPF).
- تعلم تطبيق مرشحات مصنوعة خصيصا للصور (الطي ثنائي البعد) (2D Convolution).

- **الطي ثنائي البعد _ ترشيح الصور**

كما إن الاشارات الوحيدة البعد (one-dimensional signals) يمكننا ترشيحها (فلترتها) بمرشحات التردد المنخفض والمرتفع LPF , H PF ، كذلك الأمر يمكننا تطبيقه على الصورة.



يمكننا ترشيح الصورة باستخدام مرشحات HPF،LPF .

- المرشح LPF يساعد على إزالة الضجيج أو تغبيش الصورة (blurring).
- المرشح HPF يساعدنا في إيجاد الحواف في الصورة.

مكتبة OpenCvتوفر لنا التابع () cv2.filter2D الذي يساعدنا في عملية طي (دمج) القناع مع الصورة لنحصل على صورة مرشحة، كمثال على ذلك سنجرب تطبيق المرشح المتوسط على الصورة.

سنعرف قناع بحجم 5x5 للمرشح المتوسط كما يلي:

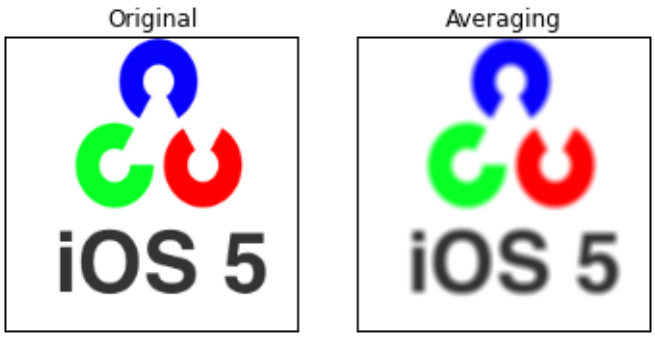
$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

هذه العملية تؤدي لحساب المتوسط لمجموعة البيكسلات الواقعة ضمن القناع.

طبق الكود التالي وشاهد النتيجة:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('opencv_logo.png')

kernel = np.ones((5,5),np.float32)/25
dst = cv2.filter2D(img,-1,kernel)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(dst),plt.title('Averaging')
plt.xticks([]), plt.yticks([])
plt.show()
```

النتيجة:



شرح الكود:

```
kernel = np.ones((5,5),np.float32)/25
```

إنشاء مصفوفة واحدية عن طريق الأمر np.ones()، حجمها هو 5x5 من النوع np.float32 ثم قسمنا المصفوفة على 25. سنستعمل هذه المصفوفة كقناع لترشيح الصورة.

```
dst = cv2.filter2D (img, -1, kernel)
```

نقوم بتمرير القناع Kernel للتابع cv2.filter2D الذي سيقوم بترشيح الصورة وفقا لهذا القناع.
المتغير الأول هو صورة الدخل، والثاني بحدد العمق اللوني لصورة الخرج فإذا كانت قيمته (-1) سنحصل على صورة بنفس العمق اللوني لصورة الدخل، والمتغير الثالث يمثل القناع الذي يجب أن يكون بقناة واحدة بمصفوفة من النوع float.

طمس الصورة (Image Blurring)

تتم عملية طمس الصورة باستعمال قناع مرشح التردد المنخفض. حيث يستخدم لإزالة الضجيج من الصورة، وخصوصا الترددات المرتفعة (مثل الضجيج والحواف..)، وتنعم الحواف (تطمسها بحيث تجعلها أخف)، (لكن هناك أنواع من تقنيات الطمس لا تقوم بطمس الحواف). المكتبة OpenCvتوفر لنا أربع تقنيات للطمس:

١- المرشح المتوسط (Averaging)

يتم ذلك عن طريق لف الصورة مع مرشح صندوقي (يمثل القناع) (عندما أقول لف الصورة مع قناع أي تمرير القناع على كل الصورة والذي سيعطينا صورة جديدة عبارة عن نتيجة محصلة القناع مع الصورة الأصلية)، فهو يأخذ ببساطة متوسط كل بيكسل في الصورة تحت منطقة القناع، ويستبدل العنصر المركزي في البيكسل بالمتوسط. تتم هذه العملية عن طريق التابع cv2.blur()أو cv2.boxFilter().

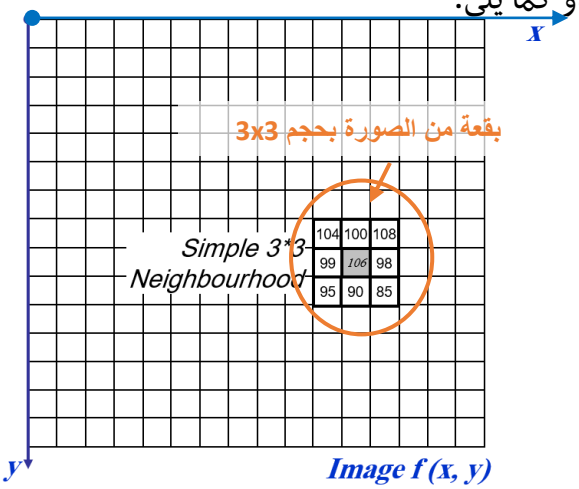
يجب علينا تحديد طول وعرض القناع المعدل.

لنفرض لدينا قناع مرشح صندوقي حجمه 3x3، عندها سوف يبدو كما يلي:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

الكسر $\frac{1}{9}$ يمثل المتوسط. هنا استخدمنا الجوار3x3.

(مصطلح الجوار يعني هنا أبعاد المصفوفة).

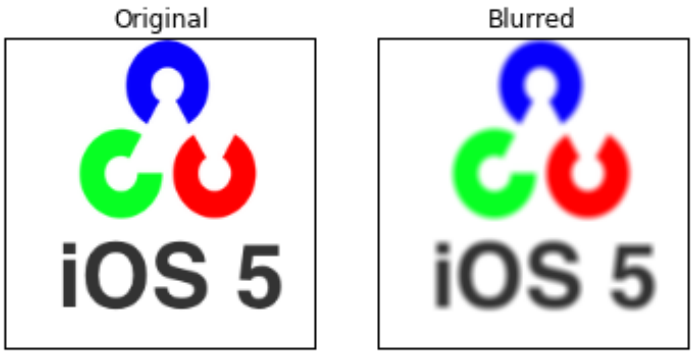


ملاحظة: إذا لم ترد استخدام المرشح الصندوقي، استخدم التابع **cv2.boxFilter()** ومرر له المتغير **normalize=False.**

ألقِ نظرة على البرنامج التالي ذو القناع 5x5:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('opencv_logo.png')

blur = cv2.blur(img,(5,5))

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

النتيجة:



شرح الكود:

```
blur = cv2.blur(img, (5,5))
```

ترشيح الصورة باستخدام تقنية الترشيح المتوسط وذلك عن طريق التابع cv2.blur().

شرح التابع cv2.blur():

المتغير الأول يأخذ الصورة المراد ترشيحا، والمتغير الثاني يمثل حجم القناع (حجم المرشح لصندوقي).

٢- المرشح الغأوسي (Gaussian Filtering):

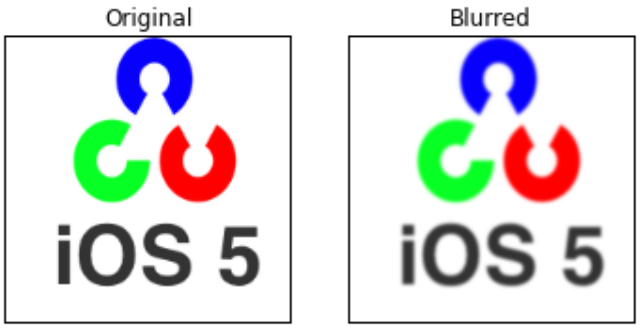
الآن بدلا من استعمال المرشح الصندوقي الذي يتألف من عناصر متسأوية، سنستخدم المرشح الغأوسي، وذلك عن طريق تطبيق التابع cv2.GaussianBlur(). في البداية يجب علينا تحديد طول وعرض قناع المرشح الغأوسي بأعداد مفردة وموجبة، ونحدد أيضاً الانحراف المعياري على المحورين X و Y. وفي حال كان أحد المحورين معطى فقط يكون الثاني مسأوياً له، وإذا كان كلا المحورين قيمتهم صفر يحسبان من حجم القناع.

المرشح الغأوسي مفيد جداً لإزالة الضجيج الغأوسي من الصورة. إذا أحببت يمكنك إنشاء قناع غأوسي باستخدام التابع cv2.getGaussianKernel().

الكود التالي نفس الكود السابق مع تغير سطر واحد فقط:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('opencv_logo.png')

blur = cv2.GaussianBlur(img,(5,5),0)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

النتيجة:



شرح الكود:

```
blur = cv2.GaussianBlur(img,(5,5),0)
```

ترشيح الصورة باستخدام تقنية الترشيح الغأوسي وذلك عن طريق التابع cv2.Gaussianblur().

شرح التابع cv2.GaussianBlur()

- المتغير الأول يأخذ الصورة المراد ترشيحا.
- المتغير الثاني يمثل حجم القناع الغأوسي.
- المتغير الثالث يمثل مقدار الانحراف المعياري.

٣- المرشح الأوسطي (Median Filtering):

التابع cv2.medianBlur يحسب متوسط مجموعة البيكسلات التي تحت القناع ويستبدل النتيجة بالعنصر المركزي. هذا المرشح فعال جدا في حالة ضجيج الملح والفلفل.

مثال: لدينا بقعة (مجموعة بيكسلات في الصورة) تأخذ بكسلاتها القيم التالية:

100,25,20,20,15,20,20,20,10

بعد تطبيق المرشح الأوسطي عليها تصبح النتيجة:

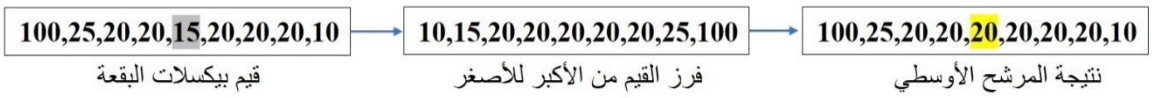
100,25,20,20,20,20,20,15,10

القيمة الأوسطية بدل ١٥

١٠٠	٢٥	٢٠
٢٠	٢٠	٢٠
٢٠	٢٠	١٠

١٠٠	٢٥	٢٠
٢٠	١٥	٢٠
٢٠	٢٠	١٠

آلية عمل المرشح الأوسطي



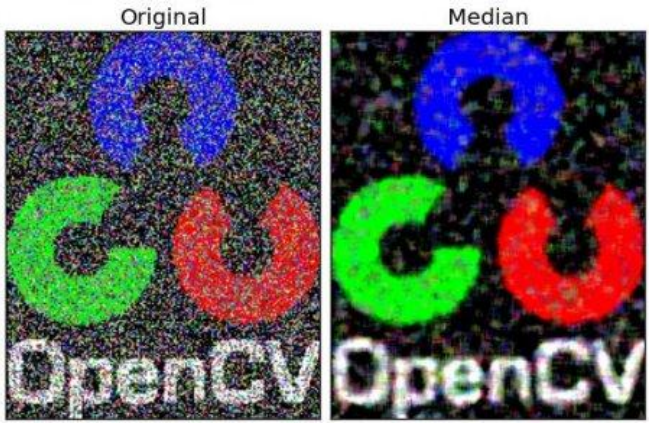
من الملاحظ بأنه دائما في المرشحات السابقة بأن العنصر المركزي في البقعة يستبدل بقيمة قد لا تكون موجودة في نفس البقعة التي مررنا فوقها النافذة، على العكس من المرشح الأوسطي الذي يستبدل العنصر المركزي ببيكسل من نفس البقعة، مما يؤدي لتقليل الضجيج بشكل فعال.

قياس القناع في المرشح الأوسطي يجب أن يكون صحيح ومفرد.

في المثال التالي سنقوم بتخفيض الضجيج من الصورة بنسبة 50% باستخدام المرشح الأوسطي.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('noisy_opencv.png')

median = cv2.medianBlur(img,5)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(median),plt.title('Median')
plt.xticks([]), plt.yticks([])
plt.show()
```

النتيجة:



شرح الكود:

```
median = cv2.medianBlur(img,5)
```

ترشيح الصورة باستخدام تقنية الترشيح الأوسطي وذلك عن طريق التابع cv2.medianblur().

شرح التابع cv2.medianBlur():

- المتغير الأول يأخذ الصورة المراد ترشيحا.
- المتغير الثاني يمثل حجم القناع، يجب أن يكون فرديا وأكبر من ١ مثل: ٣، ٥، ٧ ...

٤- المرشح الثنائي الجانبي (Bilateral Filtering):

في المرشحات السابقة نلاحظ أنها تميل لطمس الحواف، ولكن هذا الأمر لا يحدث في حالة استخدام المرشح الثنائي cv2.bilateralFilter()، وهذه هي وظيفته الأساسية، فهو يزيل الضجيج دون الحاق الضرر بالحواف، فالحواف تساعدنا في التعرف على الأشكال.

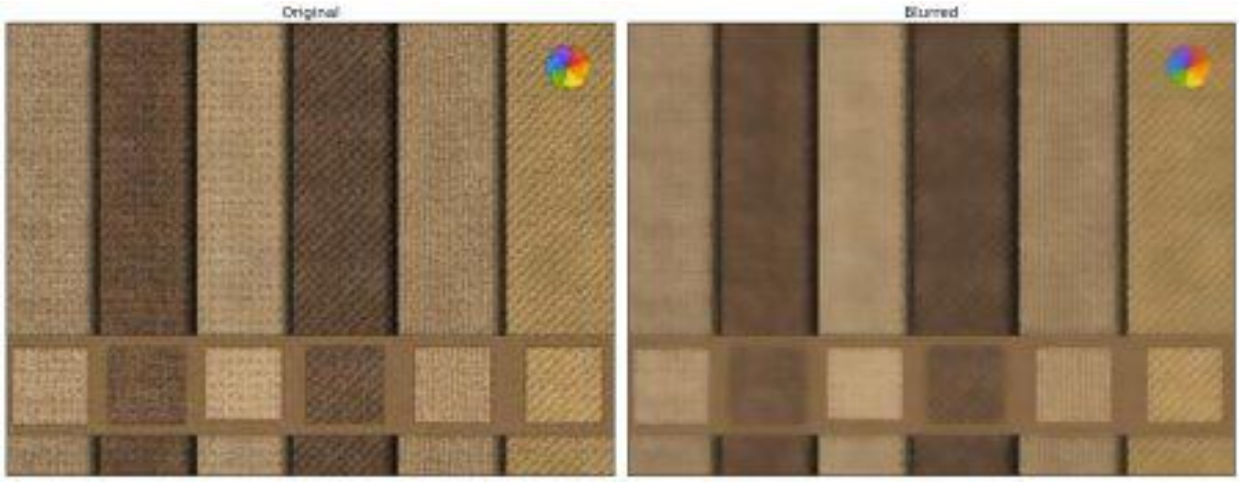
عملية الترشيح الثنائي بطيئة مقارنة مع المرشحات الأخرى. أيضاً شاهدنا سابقا كيف أن المرشح الغأوسي لا يهتم بشدة البيكسلات التي تحت القناع هل هي متقاربة بالشدة أم لا، مما يؤدي لطمس الحواف.

المرشح الثنائي الجانبي يأخذ أيضا قناع غأوسي ولكن مع مضروب إضافي تابع لفرق شدة البيكسلات، وبهذه العملية يكون حسب كل من الجوار المكاني، والجوار في الشدة، وبذلك يحافظ على الحواف.

الكود التالي يوضح ذلك:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('opencv_ios.png')

blur = cv2.bilateralFilter(img,9,75,75)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred ')
plt.xticks([]), plt.yticks([])
plt.show()
```

النتيجة:



شرح الكود:

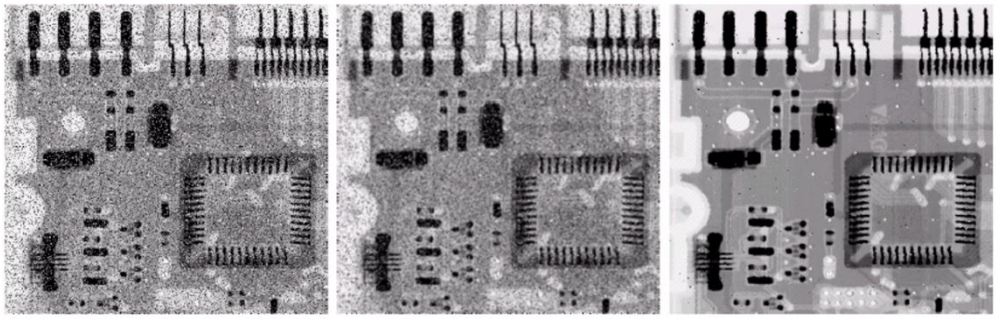
```
blur = cv2.bilateralFilter(img,9,75,75)
```

تابع طمس الصورة باستخدام المرشح الثنائي.

شرح التابع bilateralFilter ():

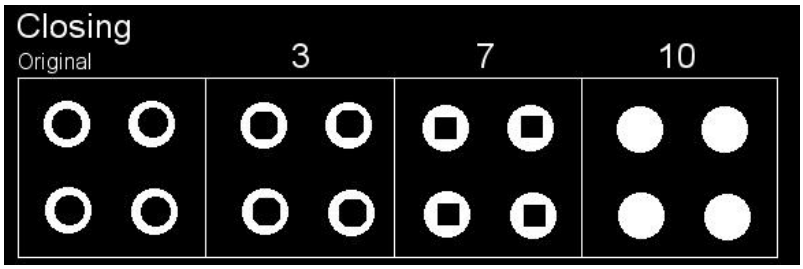
- المتغير الأول يمثل الصورة المطلوبة.
- المتغير الثاني والثالث والرابع نمررهم كما في المرشح الغأوسي، حيث المتغير الثاني والثالث يمثلان حجم القناع الغأوسي والمتغير الرابع يمثل مقدار الانحراف المعياري.

الصورة التالية لبورد الكتروني فيها ضجيج بالصورة فطبقنا عليها المرشح المتوسط والأوسطي. شاهد الفرق بين المرشحين.



Original Image With Noise | Image After Averaging Filter | Image After Median Filter

التحويلات من الناحية الشكلية (Morphological Transformations)



الهدف:

- تعلم مختلف العمليات على التحويلات التي تجري على الشكل مثل التآكل، التمدد، الفتح، الإغلاق .... الخ
- تعلم استخدام التوابع التالية: cv2.erode()،cv2.dilate()، cv2.morphologyEx()

معلومات نظرية:

التحويلات الشكلية هي بعض العمليات البسيطة التي تجرى على شكل الصورة، تنفذ عادة على الصور الثنائية. ونحتاج لتنفيذ هذه العملية لدخلين، الدخل الأول هو الصورة الأصلية، أما الدخل الثاني هو العنصر التركيبي (المصفوفة المسؤولة عن تحديد القناع) الذي يحدد طبيعة العملية.

العمليات الأساسية هي عملية التآكل وعملية التمدد Erosion and Dilation.
وعمليات الفتح والإغلاق والتدرج Opening, Closing, Gradient.

سنرى كيف سنطبق هذه العمليات على الصورة التالية:

١ – عملية التآكلErosion :

الفكرة الأساسية للتآكل هي تماما مثل تآكل التربة، عند تطبيق عملية التآكل على الشكل تؤدي لتنعيم حدود السطح الأمامي (حأول دائما أن تحرص أن يكون السطح الأمامي باللون الأبيض).

كيف تتم عملية التآكل؟!

- البيكسل في الصورة الأصلية يأخذ قيمتين إما 0 أو 1 (أسود أو أبيض).
- البيكسل الذي في الصورة الأصلية سيعتبر 1 فقط إذا كانت جميع البيكسلات تحت القناع 1، وإلا سيتم تجاهله أي يأخذ القيمة 0.
- الذي يحدث هو أن كل بيكسل قريب من الحدود سوف يتم تجاهله. سمك الحدود التي سيتم تآكلها يتوقف على حجم القناع.

هذه العملية تفيدنا في إزالة الضجيج كالنقاط البيضاء الصغيرة (كما شاهدنا في فصل الفضاء اللوني) وتفيدنا أيضاً في فصل جسمين متصلين خطيا.

الكود:

```
import cv2
import numpy as np
img = cv2.imread('j.png',0)
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 1)
cv2.imshow('orginal',img)
cv2.imshow('erosion',erosion)
```

النتيجة:

٢- عملية التمدد Dilation:

عملية التمدد عكس عملية الحت. هنا تأخذ عناصر البيكسل القيمة 1إذا كان عنصر واحد من عناصره تحت القناع 1 على الأقل، ونتيجة لذلك تزداد حجم المنطقة البيضاء.

بالعادة عند تطبيق عملية التآكل لإزالة ضجيج المنطقة البيضاء في الصورة نلاحظ بأن مساحة الجسم قد قلت، لهذا السبب نتبع عملية التآكل بعملية التمديد حتى نزيد من مساحة الجسم الذي تآكل. فعملية التمدد مفيدة لاسترجاع الأجزاء المتآكلة من الجسم.

الكود:

```
import cv2
import numpy as np
img = cv2.imread('j.png',0)

kernel = np.ones((5,5),np.uint8)
dilation = cv2.dilate(img,kernel,iterations = 1)

cv2.imshow('orginal',img)
cv2.imshow('dilation',dilation)
k = cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:

٣- عملية الفتح Opening:

عملية الفتح هي عبارة عن عملية التآكل + التمدد. هذه العملية مفيدة لإزالة الضجيج كما شرحنا سابقا. وتتم عبر التابع cv2.morphologyEx().

الكود:

```
import cv2
import numpy as np
img = cv2.imread('j.png',0)
kernel = np.ones((5,5),np.uint8)
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)

cv2.imshow('orginal',img)
cv2.imshow('opening',opening)
k = cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:

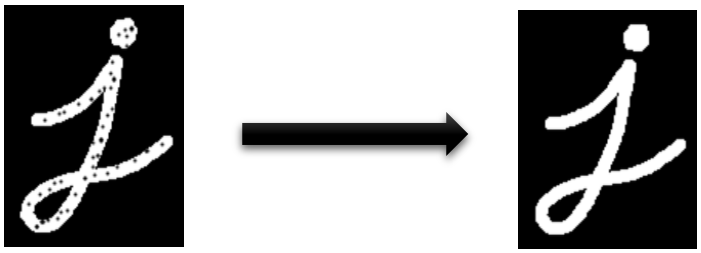
٤- عملية الإغلاق Closing:

عكس عملية الفتح، نقوم أولا بتنفيذ عملية التمدد ثم عملية التآكل.هذه العملية مفيدة لسد الثقوب في الجسم ومفيدة لسد النقاط السوداء الصغيرة على الجسم.

الكود:

```
import cv2
import numpy as np
img = cv2.imread('j.png',0)
kernel = np.ones((5,5),np.uint8)

closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

cv2.imshow('orginal',img)
cv2.imshow('closing',closing)
k = cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:

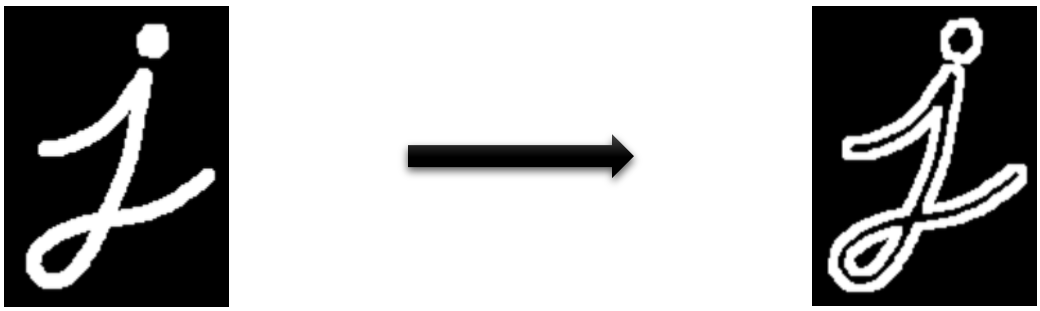


٥- عملية التدرج الشكلي Morphological Gradient:

هو الفرق بين عملية التآكل والتمدد، بحيث ستبدو كحدود للجسم.

```
import cv2
import numpy as np
img = cv2.imread('j.png',0)
kernel = np.ones((5,5),np.uint8)

gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)
cv2.imshow('orginal',img)
cv2.imshow('gradient',gradient)

k = cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:



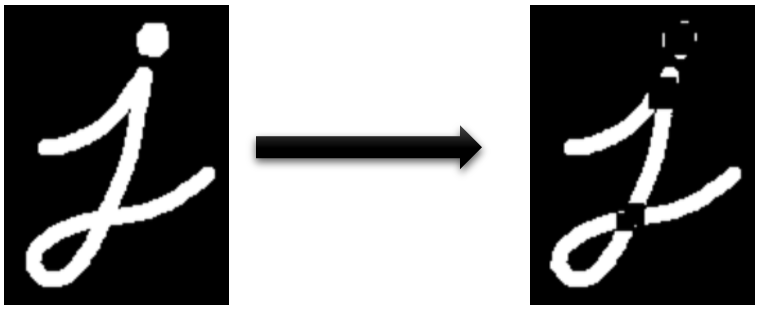
٦- عملية قبعة القمة Top Hat:

هو القرق بين عملية فتح الصورة والصورة نفسها. في الكود التالي سنطبق قناع بحجم 9x9.

الكود:

```
import cv2
import numpy as np
img = cv2.imread('j.png',0)
kernel = np.ones((9,9),np.uint8)

tophat = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)


cv2.imshow('orginal',img)
cv2.imshow(' tophat ', tophat)
k = cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:



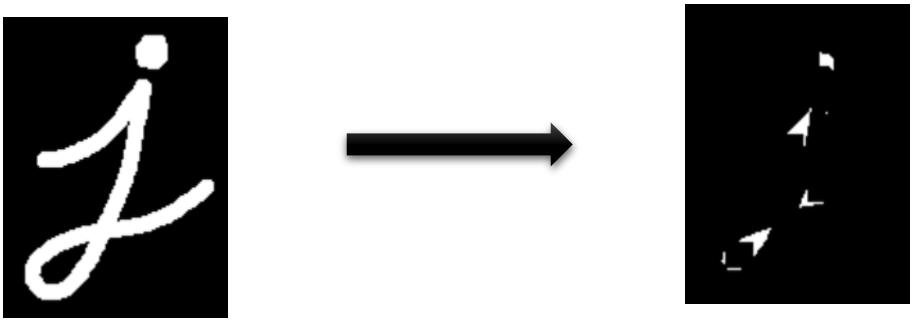
٧- عملية القبعة السوداء Black Hat:

هو القرق بين عملية إغلاق الصورة والصورة الأصلية.

الكود:

```
import cv2
import numpy as np
img = cv2.imread('j.png',0)
kernel = np.ones((9,9),np.uint8)

blackhat = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel)

cv2.imshow('orginal',img)
cv2.imshow(' blackhat ', blackhat)
k = cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:



- **العنصر التركيبي Structuring Element: (القناع mask)**

أنشأنا سابقا عنصرا تركيب يدوي بالاستعانة بمكتبة Numpy وكان شكله مستطيلي، ولكن أحيانا نحتاج لأشكال أخرى كالدائري والقطعي، ولهذا السبب وفرت لنا مكتبة OpenCvالتابع cv2.getStructuringElement(). فقط مرر شكل وحجم القناع، وستحصل على القناع الذي تريده.

```
# Rectangular Kernel
>>> cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
```

```
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)
```

```
# Elliptical Kernel
>>> cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
```

```
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

```
# Cross-shaped Kernel
>>> cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))
```

```
array([[0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

مرجع إضافي:

- **at HIPR2Morphological Operations**

تدرجات الصورة Image Gradients

الهدف:

- في هذا الفصل سنتعلم إيجاد تدرجات ألوان الصورة وتعلم ايجاد حواف الأجسام.
- سنتعلم استخدام التوابع التالية: cv2.Sobel()،cv2.Scharr()، cv2.Laplacian() .

نظري:

OpenCv توفر لنا ثلاث أنواع من مرشحات التدرج (مرشحات التردد المرتفع HPF) وهي:

سوبيل، شار، لابلاسيان **Laplacian، Scharr،Sobel** .

١- مشتقات سوبيل وشار Sobel and Scharr Derivatives:

عملية سوبيل عبارة عن دمج بين عمليتي الترشيح الغأوسي وعملية التفاضل(الاشتقاق)، لذلك فهي أكثر مقأومة للضجيج. ويمكنك تحديد اتجاه المشتق (أفقي أم عمودي) عن طريق التحكم بقيم متغيرات الدخل، ويمكنك أيضا التحكم بحجم القناع من خلال متغير في الدخل.

إذا مررت القيمة (-1) للمتغير المسؤول عن تحديد حجم القناع فسيعطي قناع بحجم 3x3 من النوع Scharr والذي سيعطينا نتائج أفضل.

٢- مشتق لابلاسيان Laplacian Derivatives:

يحسب لابلاسيان الصورة المعطى من خلال العلاقة $\Delta src = \frac{\partial^2 src}{\partial x^2} + \frac{\partial^2 src}{\partial y^2}$ حيث كل مشتق يُوجد باستخدام مشتق سوبيل.

لنفترض مثلاً أن حجم القناع كان 1، سيكون القناع المستخدم كما يلي:

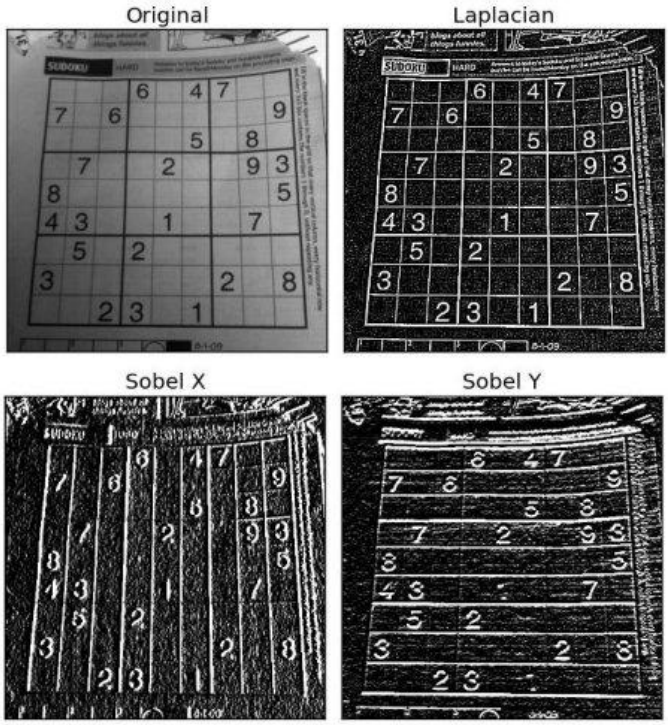
$$kernal = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

الكود:

الكود التالي سيوضح ما سبق في برنامج واحد. الأقنعة كلها ستكون بقياس 5x5، ومتغير عمق الصورة نمرر له القيمة 1- للحصول على النتيجة بصيغة np.unit8.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('dave.jpg',0)

laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()
```

النتيجة:



شرح الكود:

```
laplacian = cv2.Laplacian(img,cv2.CV_64F)
```

المتغير الأول هو صورة الدخل، والمتغير الثاني يمثل عمق صورة الخرج.

```
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0, ksize=5)
```

cv.Sobel (): المتغير الأول هو صورة الدخل، والمتغير الثاني يمثل عمق صورة الخرج، والمتغير الثالث يمثل الاشتقاق بالنسبة للمحور X والمتغير الرابع يمثل الاشتقاق بالنسبة للمحورY ،والمتغير الخامس يمثل قياس القناع (يجب أن يكون ١، ٣، ٥ أو ٧)

```
sobely = cv2.Sobel(img,cv2.CV_64F,0,1, ksize=5)
```

قضية مهمة:

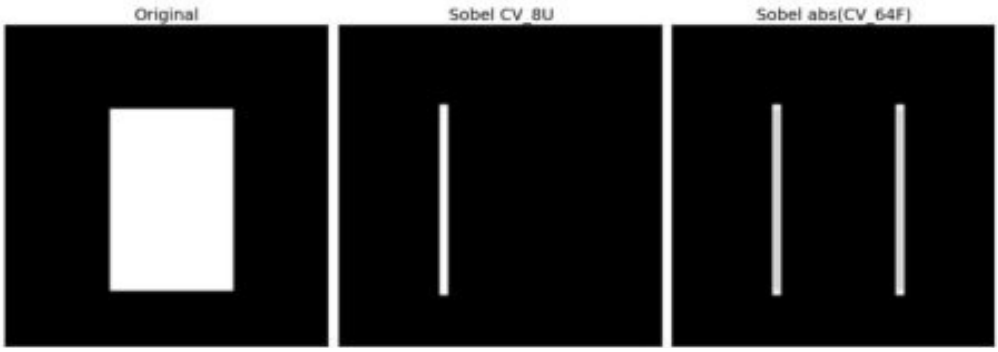
في المثال الأخير لدينا نوع بيانات الخرج هو np.uint8 أو cv2.CV_8U ولكن هناك مشكلة بسيطة، فعند التحويل من النوع الأول للثاني، تكون الحواف عند الانتقال من الاسود للأبيض بقيمة موجبة، اما عند التحويل بالعكس فستكون بقيمة سالبة، لذلك عند تحويل البيانات للنوع np.unit8 نفقد تلك الحواف لأنها تعتبر أصفاراً.

إذا أردت أن تكتشف الحواف، أفضل خيار هو أن تبقي على نوع بيانات الخرج بصيغة أعلى مثل cv2.CV_64F،cv2.CV_16S، ومن ثم نأخذ القيمة المطلقة لها، ثم نحولها للصيغة np.unit8 .

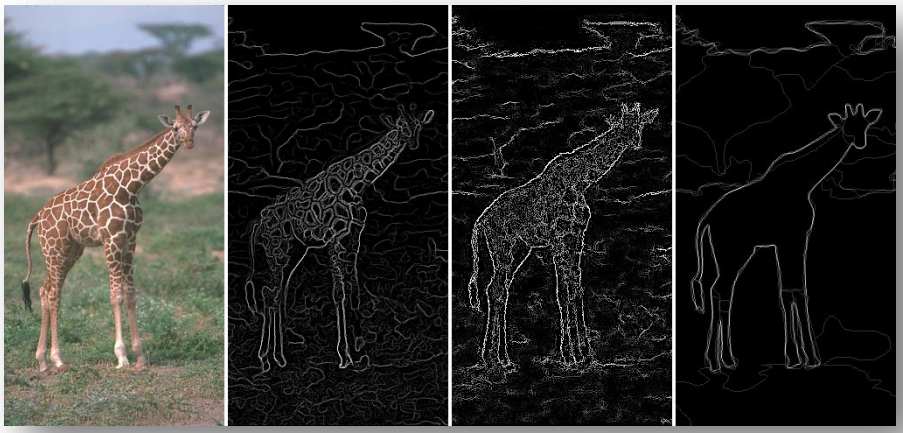
الكود التالي يوضح هذا الإجراء على مرشح سوبيل الأفقي، ويبين الخطأ الحاصل بالحالة الأولى.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('box.png',0)

# Output dtype = cv2.CV_8U
sobelx8u = cv2.Sobel(img,cv2.CV_8U,1,0,ksize=5)

# Output dtype = cv2.CV_64F. Then take its absolute and convert to cv2.CV_8U
sobelx64f = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
abs_sobel64f = np.absolute(sobelx64f)
sobel_8u = np.uint8(abs_sobel64f)

plt.subplot(1,3,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,2),plt.imshow(sobelx8u,cmap = 'gray')
plt.title('Sobel CV_8U'), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,3),plt.imshow(sobel_8u,cmap = 'gray')
plt.title('Sobel abs(CV_64F)'), plt.xticks([]), plt.yticks([])
plt.show()
```

النتيجة:



مكتشف الحواف كاني Canny Edge Detection



الهدف:

- إيجاد الحواف (حدود الأجسام) مع مكتشف حواف كاني.
- استخدام التابع cv2.Canny().

نظري:

مكتشف حواف كاني هو خوارزمية مشهورة لتحديد الحواف. حيث تتألف هذه الخوارزمية من عدة مراحل، وسوف نمر على هذه المراحل خطوة بخطوة:

أ. تخفيض الضجيج Noise Reduction

بما أن اكتشاف الحواف يتأثر بضجيج الصورة، لذلك أول خطوة سنقوم بها هي إزالة هذا الضجيج عن طريق استخدام مرشح غأوسي 5x5 ، كما تعلمنا سابقا.

ب. إيجاد شدة تدرج الصورة

الصورة المرشحة ترشح بعدها وفق قناع سوبيل في كلا الاتجاهين العمودي والأفقي لنحصل على التدرج (المشتق الأولي) لصورة في الاتجاه الأفقي G_x وصورة بالاتجاه العمودي G_y .

ومن هاتين الصورتين يمكننا ايجاد تدرج الحافة واتجاه كل بيكسل على النحو التالي:

$$EdgeGradient(G) = \sqrt{G_x^2 + G_y^2}$$

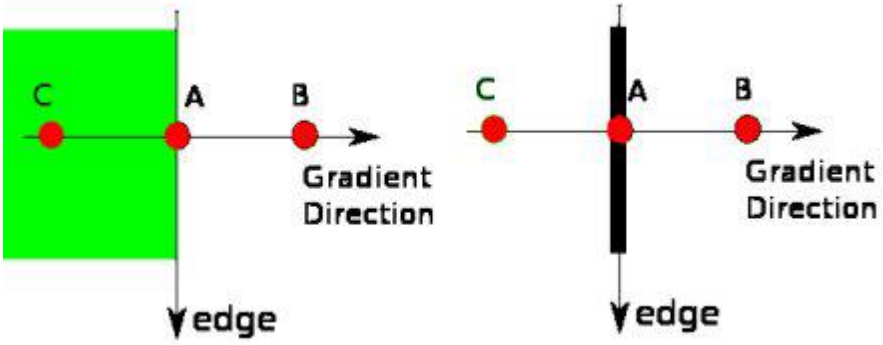
$$Angle\ (\theta) = tan^{-1}(\frac{G_y}{G_x})$$

اتجاه التدرج(الاشتقاق) دائما عمودي على الحافة، ويتم تقريبه إلى أحد من الأربع الزوايا المتمثلة بالاتجاه الافقي والعمودي واتجاه القطرين.

ت. إزالة التشويش الا أعظمي Non-maximum Suppression

بعد الحصول على التدرج والاتجاه، يتم إجراء مسح للصورة لحذف البيكسلات التي لا تشكل الحافة، لذلك فحص كل بيكسل، فيما إذا كانت قيمته أكبر (أعظم) من قيمة البيكسلات المجأورة له في الاتجاه والشدة.

انظر للصور لتالية

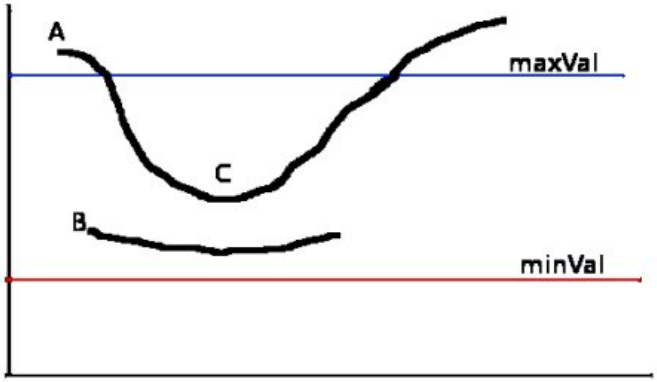


النقطة A على الحافة (في الاتجاه الأفقي). ومن الطبيعي أن يكون اتجاه الحافة باتجاه الميل. النقطة B وC في اتجاه التدرج. لذلك يتم فحص النقطة A مقارنة مع النقطة B وC لمعرفة ما إذا كانت هذه النقطة تشكل القيمة العظمى الحالية.

إذا كانت قيمة النقطة A فِعلاً هي أعظم قيمة يتابع للخطوة التي بعدها وإذا لم تكن عظمى يتم حذفها بوضع القيمة 0. وباختصار، فإن النتيجة التي نحصل عليها هي صورة ثنائية مع حواف أنعم.

ث. التعتيب المُتباطئ Hysteresis Thresholding:

في هذه المرحلة نحدد فيما إذا كانت الحواف فعلا حواف أم لا، لذلك نحتاج قيمتين للعتبة، عظمى وصغرى (maxVal، minVal)، وأي حافة شدة تدرجها أكبر من الحد الأعلى نعتبرها حافة أكيدية وإذا كانت قيمتها أقل من الحد الأدنى لا نعتبرها حافة ونصفرها. أما بالنسبة للقيمة الوسطى فنعتمد على اتصالها فإذا كانت متصلة مع حافة أكيدية فنأخذها أما إذا كانت غير متصلة لا نأخذها. شاهد الصورة التالية:



- الحافة A فوق قيمة العتبى العظمى maxValلذلك نعتبرها حافة أكيدية.
- الحافة Cأسفل maxValولكنها متصلة مع الحافة A، لذلك نعتبرها أيضاً حافة أكيدية.
- الحافة B على الرغم من أنها فوق القيمي الدنيا للعتبة minVal وفي نفس منطقة الحافة C ولكنها غير متصلة مع حافة أكيدية لذلك سنتخلص منها بتصفيرها.

نتيجة لما سبق يجب أن نهتم جدا باختيار قيمة صحيحة لـ minVal وmaxVal حتى نحصل على نتيجة صحيحة. هذه العملية تزيل أيضاً الضجيج الصغير للبيكسل بفرض أن الحواف خطوط طويلة (أي تزيل الحواف القصيرة). لذلك في النهاية سنحصل فقط على الحواف القوية.

- **اكتشاف حواف كاني باستخدام مكتبة OpenCv:**

OpenCv وضعت لنا جميع الخطوات السابقة في تابع واحد وهو cv2.Canny().

شرح التابع cv2.Canny():

أول متغير هو صورة المطلوب اكتشاف حوافها، والمتغير الثاني والثالث هما حدا العتبة، والمتغير الرابع هو قياس قناع سوبيل المُستعمل لإيجاد تدرج الصورة (بالحالة الافتراضية تكون قيمته 3)، والمتغير الخامس والأخير هو L2gradient وظيفته هي تحديد المعادلة التي توجد شدة التدرج، فإذا كانت قيمته صحيحة TRUE فإنه يستخدم المعادلة التي ذكرناها سابقا وهي:

$$\boldsymbol{EdgeGradient(G)} = \sqrt{\boldsymbol{G_x^2 + G_y^2}}$$

وهي أكثر دقة.

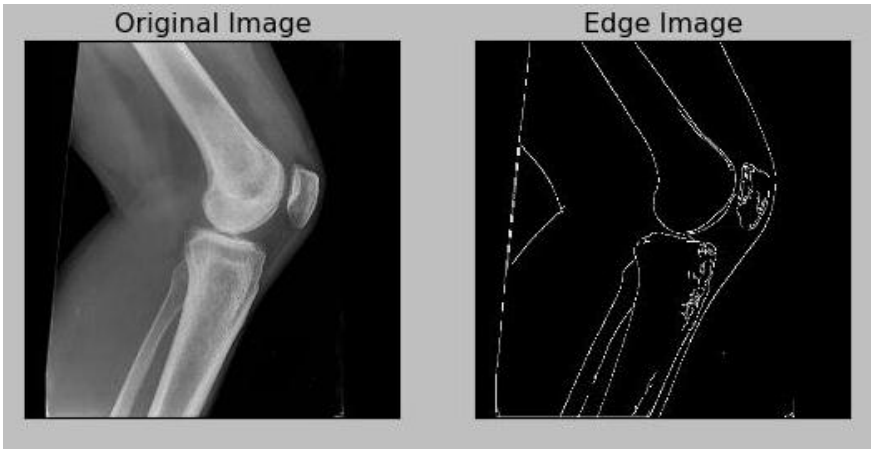
أما إذا كانت قيمته False فسيستخدم المعادلة التالية:

$$\boldsymbol{Edge\_Gradient(G)} = |\boldsymbol{G_x}| + |\boldsymbol{G_y}|$$

(الحالة الافتراضية للمتغير الخامس تكون FALSE).

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('patella.png',0)

edges = cv2.Canny(img,90,250)

plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

النتيجة:



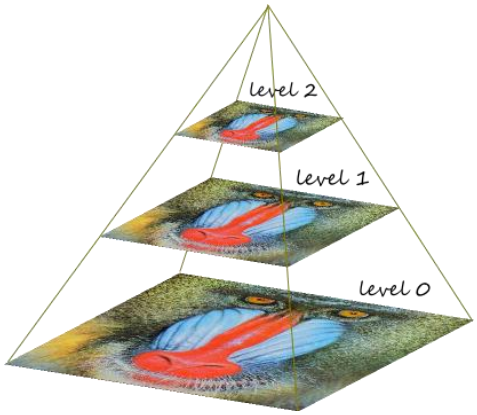
مراجع إضافية:

- **Canny edge detector at Wikipedia**
- **Canny Edge Detection Tutorial by Bill Green, 2002.**

تمرين:

اكتب تطبيق صغير لاكتشاف حواف كاني لصورة ما وقم بتحديد قيمة العتبتين عن طريق شريط الانزلاق side bar، وبهذه الطريقة ستفهم جيدا تأثير قيمة العتبات على نتيجة اكتشاف الحواف.

أهرامات الصورة Image Pyramids



الهدف:

- سنتعرف على أهرامات الصورة
- سنستعمل دمج الصور باستخدام الأهرامات، سننشئ فاكهة جديدة "برتفاح" (برتقال + تفاح).
- سنتعلم استخدام التوابع التالية:cv2.pyrUp()، cv2.pyrDown() .

نظري:

عادة نتعامل مع صور ذات قياس ثابت، ولكن قد نحتاج أن نتعامل مع قياسات مختلفة للصورة نفسها. فعلى سبيل المثال عند البحث عن شيء ما في الصورة مثل الوجه، نحنا لسنا متأكدين ما هو حجم الجسم الموجود في الصورة، لذلك سنحتاج لإنشاء مجموعة من الصور بقياسات مختلفة والبحث عن الوجوه في جميع الصور.

تسمى مجموعة الصور ذات القياسات المختلفة بهرم الصورة (لأنه عند الاحتفاظ بها ووضعها فوق بعضها البعض من أكبر صورة (أدق صورة) لأصغر صورة (أقل صورة دقة) ستبدو على شكل هرمي).
هناك نوعين لأهرامات الصورة: الهرم الغأوسي والهرم الابلاسي.

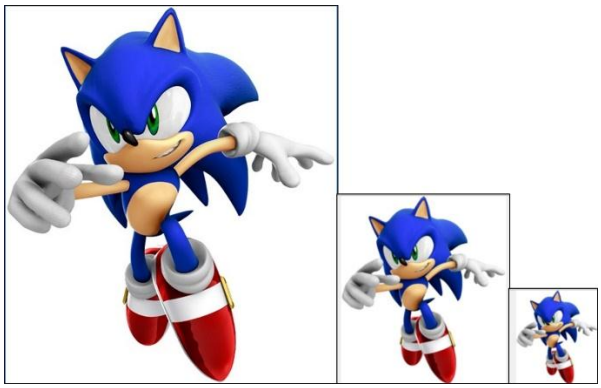
يتشكل أعلى مستوى (الأخف دقة) في الهرم الغأوسي عن طريق حذف الأسطر والاعمدة المتعاقبة في المستوي الأدنى للهرم (الأعلى دقة)

ثم يتشكل كل بيكسل في المستوي الأعلى للهرم بتطبيق القناع الغأوسي على 5بيكسلات من المستوى الأدنى للهرم. ونتيجة لذلك يتقلص حجم الصورة للربع. هذه العملية تسمى أوكتاف (Octave).

يستمر هذا الأمر كلما صعدنا لقمة الهرم (تنخفض الدقة ويقل حجم الصورة كلما صعدنا مستوي أعلى باتجاه القمة). أما عند التحرك من مستوي أعلى لمستوي أدنى يزيد حجم ووضوح الصورة.
يتم إنجاز عملية زيادة حجم ووضوح الصورة ونقصانهما عن طريق التابعين: cv2.pyrDown() و cv2.pyrUp().

- التابع cv2.pyrUp() يزيد دقة وحجم الصورة.
- التابع cv2.pyrDown() يخفف دقة وحجم الصورة.

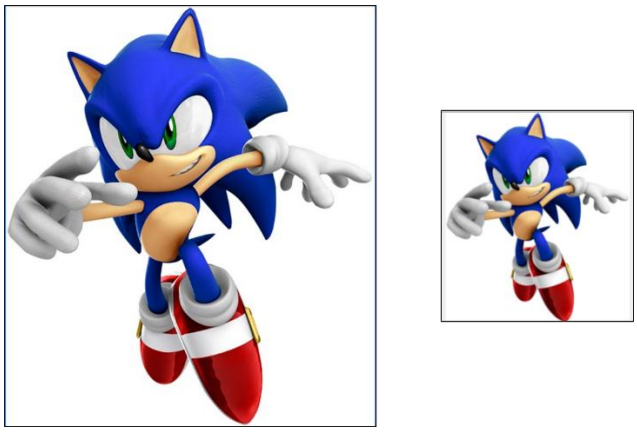
الصورة التالية تبين ثلاث مستويات من الهرم الغأوسي:



شاهد الكود التالي كيف سيخفض دقة الصورة:

```
import cv2
import numpy as np

img = cv2.imread('sonic.jpg',1)
lower_reso = cv2.pyrDown(img)

cv2.imshow('orginal',img)
cv2.imshow('lower_reso Image',lower_reso)
k = cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:



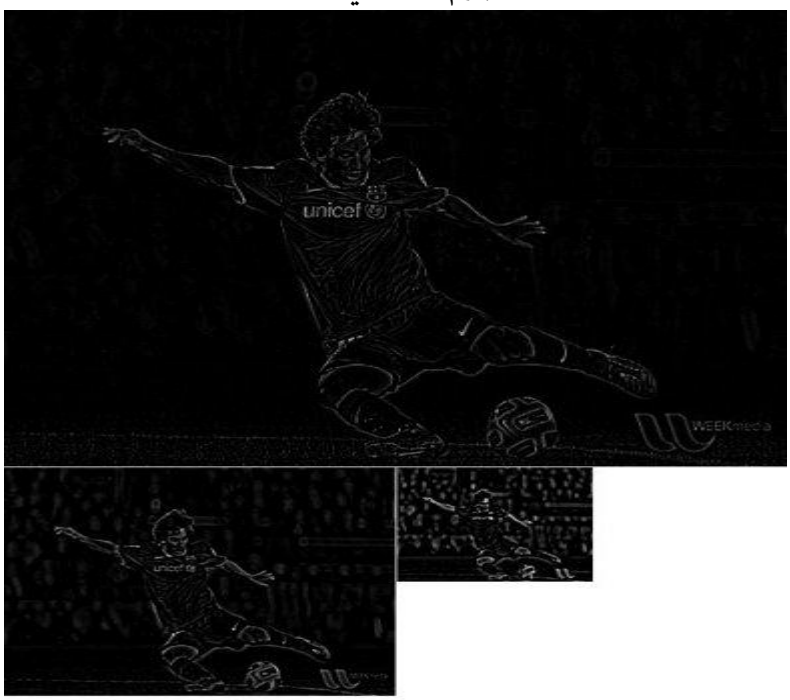
وشاهد الكود التالي كيف سيزيد وضوح الصورة الصغيرة، ولكن تذكر بأن الصورة الناتجة عند النزول في الهرم تختلف عن الصورة الأصلية لأن توليد صورة أقل دقة يحذف معلومات من الصورة، ولذلك ستكون مختلفة يمكنك ملاحظة الفرق في الصورة عند تطبيق الكود التالي:

```
higher_reso = cv2.pyrUp(lower_reso)
```

الأهرامات الابلاسية تتشكل من الأهرامات الغأوسية، ولا يوجد تابع مخصص لها.

صور الهرم الابلاسي معظم عناصرها أصفار. تستعمل طريقة الهرم الابلاسي لضغط الصور. حيث يتشكل المستوى في الهرم الابلاسي من خلال الفرق بين المستوي المناظر للهرم الغأوسي مع نسخة مكبرة في المستوي الأدنى للهرم.

الصورة التالية تبين ثلاث مستويات من الهرم الابلاسي:

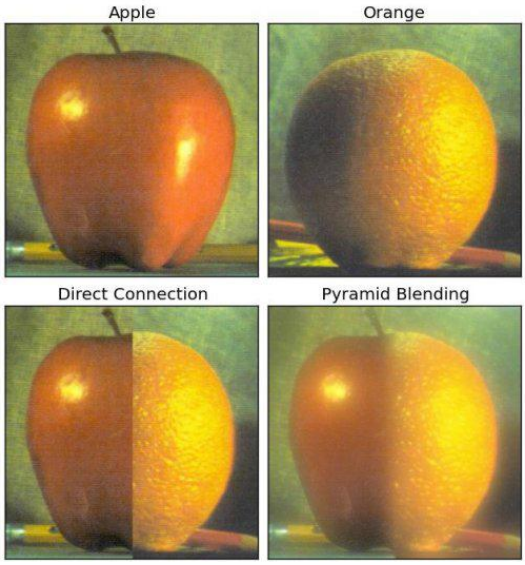


- **دمج الصور باستخدام الأهرامات:**

إحدى تطبيقات الاهرامات هي دمج الصورBlending .

عند دمج صور معا قد لا تبدو الصورة الناتجة جيدة بسبب وضوح الانقطاع بين الصور، لذلك سنحتاج لطريقة تساعدنا على مزج الصور معا بسهولة وبدون فقدان البيانات، وهذا ما توفره لنا أهرامات الصورة.

أحد الأمثلة التقليدية على ذلك هو عملية الدمج بين فكهتين هما التفاح والبرتقال، شاهد الصورة التالية لتفهم ما أقصد.



عملية الدمج بين التفاح والبرتقال تتم على النحو التالي:

١. نحمل صورة التفاحة والبرتقالة
٢. نوجد الهرم الغأوسي للتفاحة والبرتقالة (في هذا المثال أوجدنا 6مستويات للهرم)
٣. من الهرم الغأوسي نوجد الهرم الابلاسي
٤. الآن نجمع القسم اليساري للتفاحة مع القسم اليميني للبرتقالة في كل مستوي من الهرم الابلاسي.
٥. وأخيراً من صور الهرم المجموعة نبني الصورة الأصلية.

الكود التالي يشرح طريقة التنفيذ خطوة بخطوة وبكل بساطة (للتبسيط والإيضاح قمنا بكتابة الكود بهذه الطريقة. إن هذا الطريقة في كتابة الكود ستؤدي لاستهلاك قسم كبير من الذاكرة)

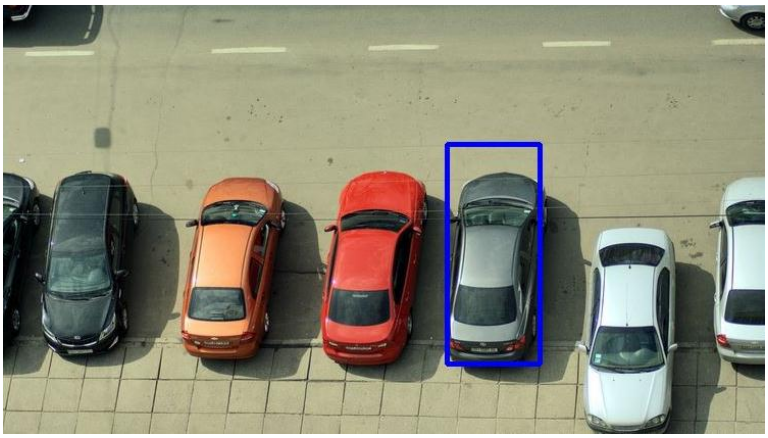
```
import cv2
import numpy as np,sys

A = cv2.imread('apple.png')
B = cv2.imread('orange.png')

# generate Gaussian pyramid for A
G = A.copy()
gpA = [G]
for i in xrange(6):
    G = cv2.pyrDown(G)
    gpA.append(G)

# generate Gaussian pyramid for B
G = B.copy()
gpB = [G]
for i in xrange(6):
    G = cv2.pyrDown(G)
    gpB.append(G)

# generate Laplacian Pyramid for A
lpA = [gpA[5]]
for i in xrange(5,0,-1):
    GE = cv2.pyrUp(gpA[i])
    L = cv2.subtract(gpA[i-1],GE)
    lpA.append(L)
```

```
# generate Laplacian Pyramid for B
lpB = [gpB[5]]
for i in xrange(5,0,-1):
    GE = cv2.pyrUp(gpB[i])
    L = cv2.subtract(gpB[i-1],GE)
    lpB.append(L)

# Now add left and right halves of images in each level
LS = []
for la,lb in zip(lpA,lpB):
    rows,cols,dpt = la.shape
    ls = np.hstack((la[:,0:cols/2], lb[:,cols/2:]))
    LS.append(ls)

# now reconstruct
ls_ = LS[0]
for i in xrange(1,6):
    ls_ = cv2.pyrUp(ls_)
    ls_ = cv2.add(ls_, LS[i])

# image with direct connecting each half
real = np.hstack((A[:,:cols/2],B[:,cols/2:]))

plt.imshow(ls_)
plt.show()

plt.imshow(real)
plt.show()
```

مرجع إضافي:

- Image Blending

مطابقة القوالب Template Matching



الهدف:

- إيجاد جسم داخل صورة باستخدام مطابقة القوالب
- سنتعلم استخدام التوابع: cv2.matchTemplate()، cv2.minMaxLoc()

معلومات نظرية:

مطابقة القوالب هي طريقة للبحث وإيجاد موضع قالب للصورة ضمن صورة أكبر. مكتبة OpenCv توفر لنا التابع cv2.matchTemplate() لتنفيذ عملية مطابقة القالب.

لنتفق في البداية على مصطلحين وهما صورة القالب وصورة الهدف. فإذا أردنا البحث مثلا عن صورة وجه الطفل في الصورة التالية، فإننا نسمي صورة الوجه بالقالب، والصورة التي بحثنا بداخلها عن صورة الوجه بالصورة الهدف.

عملية لمطابقة تتم ببساطة بحيث نمرر صورة القالب فوق صورة الهدف ونقارن بين القالب وكل بقعة في صورة الهدف حتى نجد صورة القالب المطلوبة. هناك عدة طرق لعملية المقارنة ضمن مكتبة OpenCV.

تتم عملية المطابقة عن طريق تمرير صورة القالب على كامل صورة الهدف، وتكون الصورة الناتجة عن المقارنة صورة رمادية، بحيث كل بيكسل ضمنها يدل على كمية البيكسلات المجأورة له التي تتطابق مع القالب.

إذا كانت صورة الدخل بحجم (W x H) وصورة القالب حجمها (w x h)، سيكون حجم صورة الخرج (W-w+1 x H-h+1)، عند الحصول على صورة نتيجة المطابقة يمكنك استعمال التابع cv2.minMaxLoc() لإيجاد أكبر وأقل قيمة من عناصر المصفوفة (للصورة الناتجة عن المطابقة) بحيث يعيد لنا هذا التابع قيمتين بالشكل (x,y) .

إذا اعتبرنا هاتين القيمتين هما إحداثيات الزأوية اليسرى العليا للمستطيل، واعتبرنا ان عرض هذا المستطيل وطوله هو (w x h) عندها سيكون هذا المستطيل هو الذي سيحدد منطقة القالب في الصورة الناتجة (للتوضيح: أخذنا الطول والعرض من صورة القالب وأخذنا إحداثيات زأوية المستطيل من التابع minMaxLoc()).

ملاحظة: إذا استخدمت **cv2.TM_SQDIFF** كطريقة للمقارنة فيفضل استخدام قيمة صغرى فهي تعطي نتيجة أفضل للمطابقة.

- **مطابقة القوالب في OpenCv:**

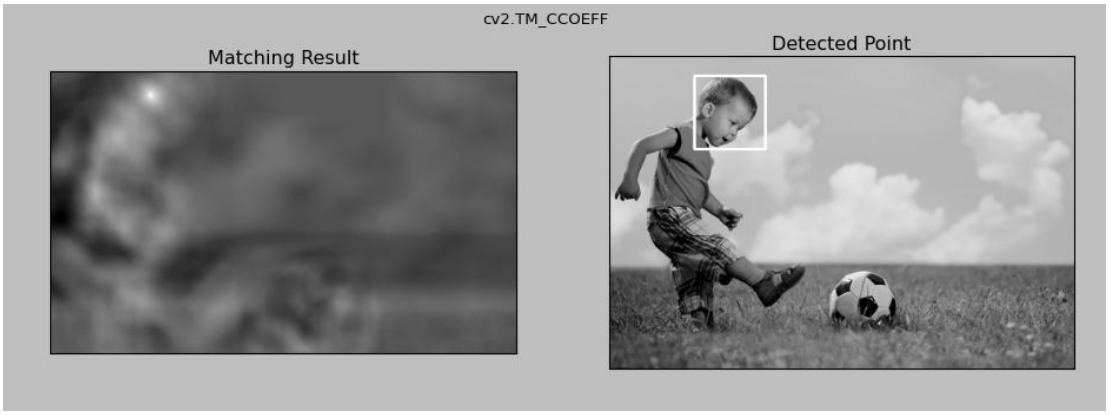
في هذا المثال سنقوم بالبحث عن وجه الطفل(القالب) في الصورة السابقة (الهدف)، لذلك سننشأ قالب كما يلي:

سنجرب كل طرق المقارنة وسنشاهد نتائجها:

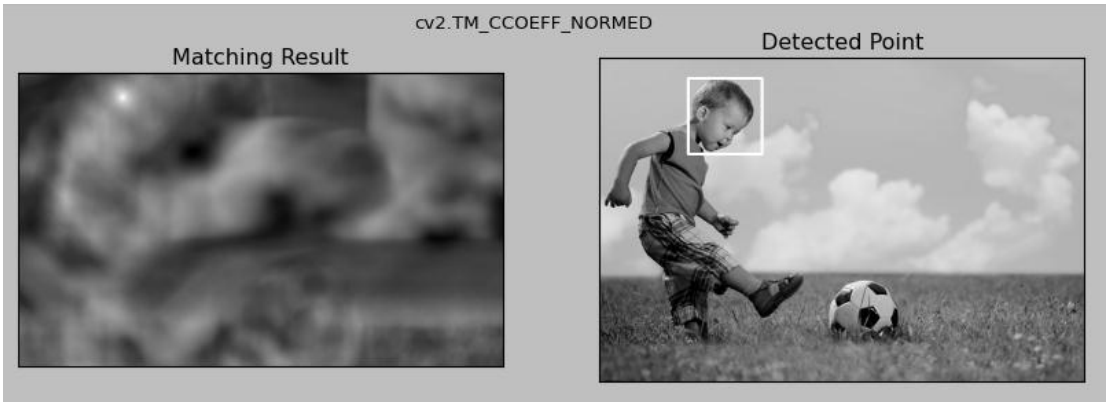
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('kid_football.jpg',0)
img2 = img.copy()
template = cv2.imread('template.jpg',0)
w, h = template.shape[::-1]

# All the 6 methods for comparison in a list
methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',
        'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF',
'cv2.TM_SQDIFF_NORMED']
for meth in methods:
   img = img2.copy()
   method = eval(meth)
   # Apply template Matching
   res = cv2.matchTemplate(img,template,method)
   min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

   # If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
   if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
      top_left = min_loc
   else:
      top_left = max_loc
   bottom_right = (top_left[0] + w, top_left[1] + h)

   cv2.rectangle(img,top_left, bottom_right, 255, 2)

   plt.subplot(121),plt.imshow(res,cmap = 'gray')
   plt.title('Matching Result'), plt.xticks([]), plt.yticks([])
   plt.subplot(122),plt.imshow(img,cmap = 'gray')
   plt.title('Detected Point'), plt.xticks([]), plt.yticks([])
   plt.suptitle(meth)
   plt.show()
```

النتيجة:

- cv2.TM_CCOEFF

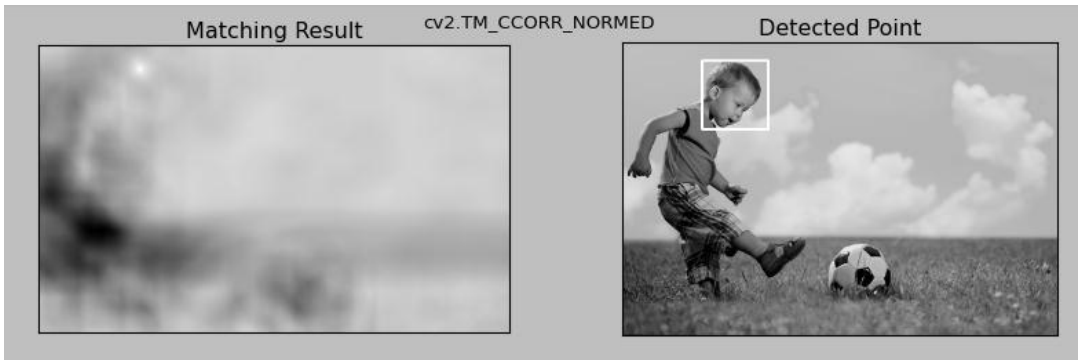


- cv2.TM_CCOEFF_NORMED



- cv2.TM_CCORR

- **cv2.TM_CCORR_NORMED**



- **cv2.TM_SQDIFF**

- **cv2.TM_SQDIFF_NORMED**

يمكنك الملاحظة بأن نتيجة استخدام cv2.TM_CCORR ليست جيدة.

شرح الكود:

```
img = cv2.imread('kid_football.jpg',0)
img2 = img.copy()
template = cv2.imread('template.jpg',0)
w, h = template.shape[::-1]
```

في البداية قمنا باستدعاء مكتبتي OpenCv وNumpy. ثم خزّنا الصورة الهدف (الصورة المطلوب إيجاد الجسم ضمنها) في المتغير img وقمنا بنسخ الصورة وتخزينها في المتغير imge2. ثم خزّنا صورة القالب في المتغير template. وبعد ذلك خزنا ارتفاع وعرض صورة القالب في المتغيرين w وh وذلك عن طريق التابع ()shape.

```
methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',
        'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORMED']
```

أنشأنا مصفوفة وخزنها بداخله الأسماء التي بين الأقواس. هذه الأسماء ستمثل أسماء النوافذ التي سنعرضها.

```
for meth in methods:
   img = img2.copy()
   method = eval(meth)
```

الغاية هنا من حلقة for هي عرض كل طرق المقارنة.

وظيفة التابع eval() هو إزالة علامة الاقتباس (لتوضيح أكثر شاهد الأمثلة من هنا)

```
  # Apply template Matching
   res = cv2.matchTemplate(img,template,method)
   min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
```

نقوم بعملية المطابقة عن طريق التابع ()cv2.matchTemplate .

المتغير الأول لهذا التابع يمثل الصورة الهدف والمتغير الثاني يمثل صورة القالب والمتغير الثالث يمثل طريقة المقارنة.

```
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
```

بشكل عام التابع minMaxLoc() يوجد قيمة أكبر وأقل عنصر في المصفوفة ويعطينا أيضا موقع هذين العنصرين ضمن المصفوفة بالإحداثيات (x,y).

في سطر الكود هذا سنحدد المنطقة التي تم إيجاد فيها صورة القالب ضمن الصورة الهدف بمستطيل وذلك بالاستعانة بالتابع **cv2.minMaxLoc()** الذي يعيد لنا أعلى وأدنى قيمة للبيكسل في الصورة الناتجة عن المطابقة مع موضع هذين البيكسلين ضمن هذه الصورة. نختار أحداثيات أحد هذين الموضعين لنحدد إحداثيات الزوية اليسرى العليا للمستطيل. (في مثالنا عند استعمال الطريقة cv2.TM_CCOEFF سنحصل على الحدين (١٣٣، ١٠) (٢٢، ٩١)). ثم نختار الحد المناسب الذي يمثل زأوية المستطيل.

سيتم تخزين الحد الأدنى في المتغير min_loc والحد الأعلى في المتغير max_loc. كما حصلنا حتى الآن على إحداثيات زأوية المستطيل، ونحتاج الآن لتحديد حدود منطقة المطابقة لذلك يجب أن نعرف طول وعرض هذا المستطيل

فكيف سنعرف الطول والعرض؟

طول وعرض المستطيل هما نفس طول وعرض صورة القالب.

```
# If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
    top_left = min_loc
else:
    top_left = max_loc
bottom_right = (top_left[0] + w, top_left[1] + h)

cv2.rectangle(img,top_left, bottom_right, 255, 2)
```

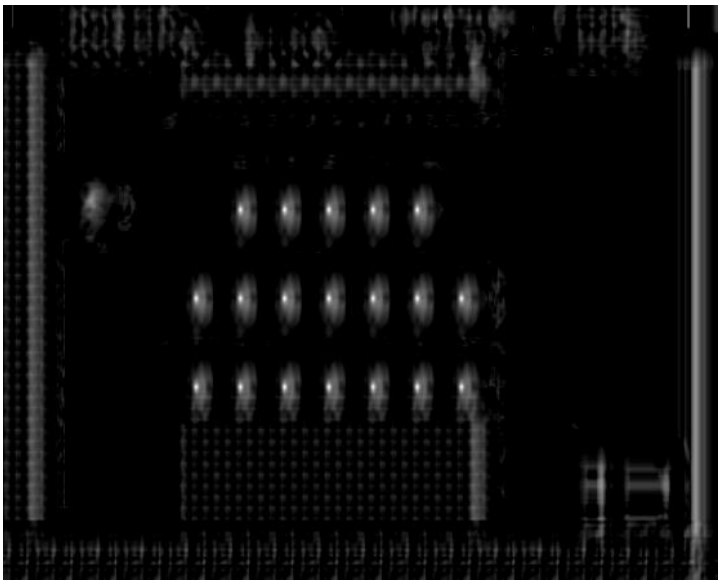
في هذا الكود اخترنا أحداثيات موضع أعلى أو أدنى قيمة للبيكسل وذلك حسب كل طريقة مقارنة، فبالنسبة لأخر طريقتي مقارنة اخترنا إحداثيات القيمة الأدنى فهو الحد المناسب لتحديد أحداثيات زأوية مستطيل منطقة المطابقة، أما بالنسبة لطرق المقارنة الباقية فقد اخترنا إحداثيات القيمة الأعلى فهو الأنسب لتحديد أحداثيات زأوية هذا المستطيل. وأخيرنا قمنا برسم مستطيل حول منطقة القالب الموجودة في صورة الهدف.

- **مطابقة القوالب لعدة أجسام معاً:**

في السابق بحثنا عن جسم واحد وهو وجه الطفل والذي يوجد مرة واحدة في الصورة، ولكن افترض بأن لدينا أكثر من جسم نريد إيجاده بالصورة، عندها لن ينفع استخدام التابع cv2.minMaxLoc() ويجب بدلاً عنه استخدام التعتيب. لنأخذ المثال التالي من لعبة ماريو المشهورة:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img_rgb = cv2.imread('mario.jpg')
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)
template = cv2.imread('mario_coin.png',0)
w, h = template.shape[::-1]

res = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)
threshold = 0.8
loc = np.where( res >= threshold)
for pt in zip(*loc[::-1]):
   cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,0,255), 2)

cv2.imshow('res',img_rgb)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:



شاهد من صورة نتيجة المطابقة بأن النقط البيضاء الأكثر سطوعا تمثل البيكسلات التي تأخذ أكبر قيمة، والتي رسمنا عندها مستطيل ملون بالأحمر كما هو واضح في الصورة السابقة.

مساعدة لشرح الكود:

لنعرف مصفوفة A ثنائية البعد 2D تأخذ القيم التالية:

A = [[2 3 4 5 6 78 97 100]

[8 3 1 5 7 8 9 10]]

```
A = np.array([[2,3,4,5,6,78,97,100],[8,3,1,5,7,8,9,10]])
print A
```

سنطبق عليها التابع np.where(A > threshold) وسنضع الشرط بداخل هذا التابع بأن يأخذ عناصر المصفوفة التي قيمتها تكون أكبر من ٥ (أي نضع الـ threshold = 5) وسيعيد لنا التابع موضع هذه العناصر ضمن المصفوفة. (في الكود مطابقة القوالب لعدة اجسام غير ال threshold وشاهد ماذا يحصل)

```
threshold = 5
loc = np.where( A > threshold)
```

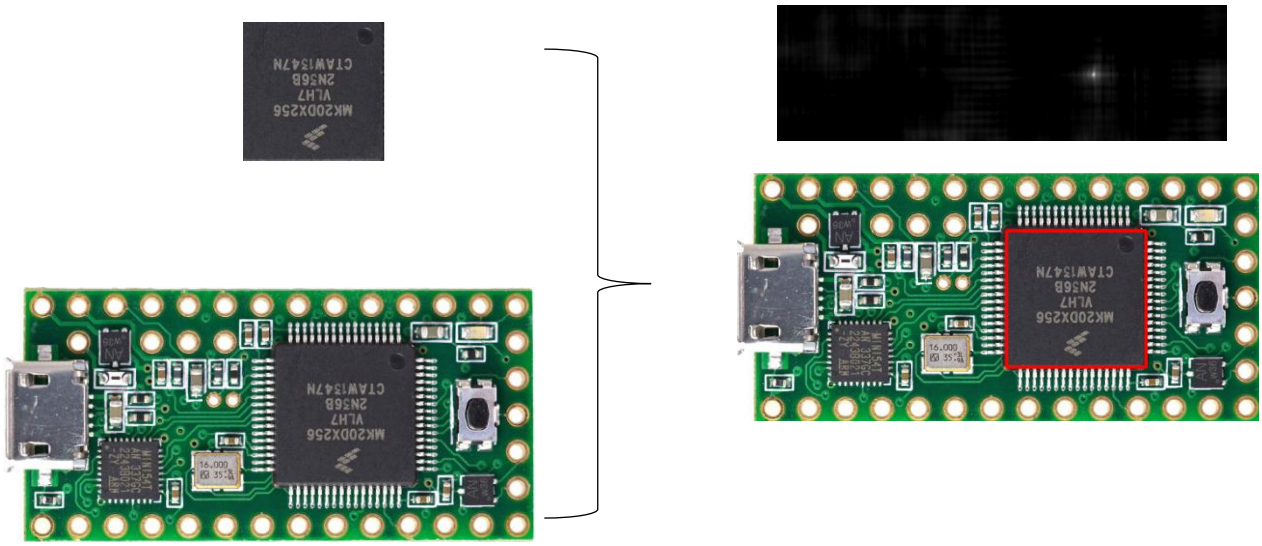
(array([0, 0, 0, 0, 1, 1, 1, 1, 1]), array([4, 5, 6, 7, 0, 4, 5, 6, 7]))

شاهد النتيجة السابقة سترى بأنه أعاد موضع كل عنصر في المصفوفة ضمن الشرط برقم السطر والعمود حيث الـarray الأولى تمثل السطر والـ array الثانية تمثل العمود.

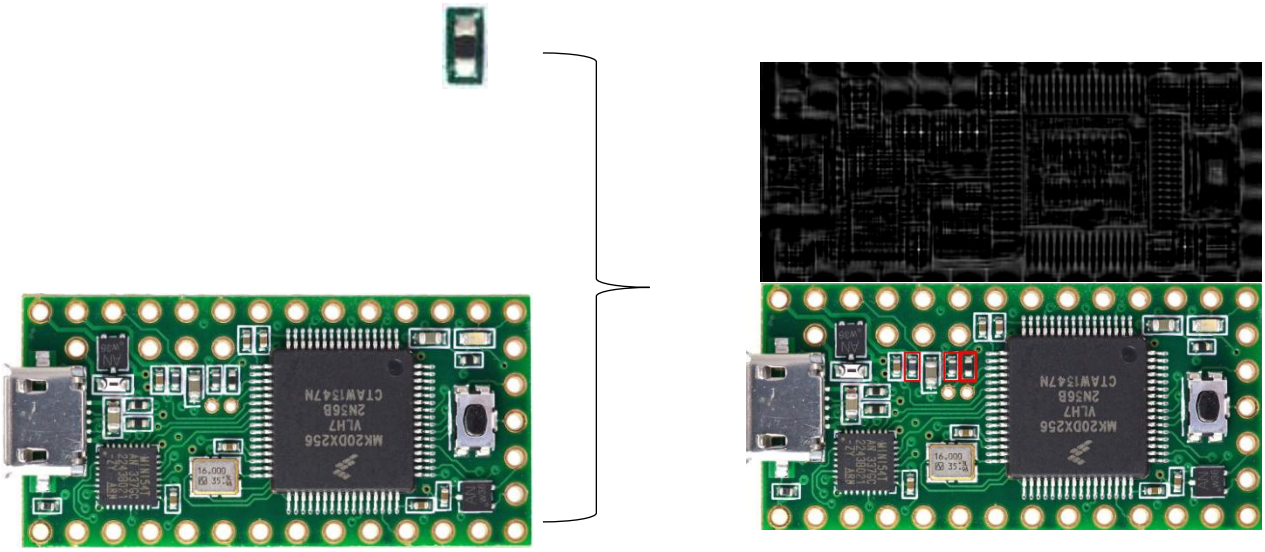
هذا الموقع يحوي على عدة أمثلة حتى تفهم العمليات التي تجري على المصفوفة والتي استخدمناها في الكود:

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

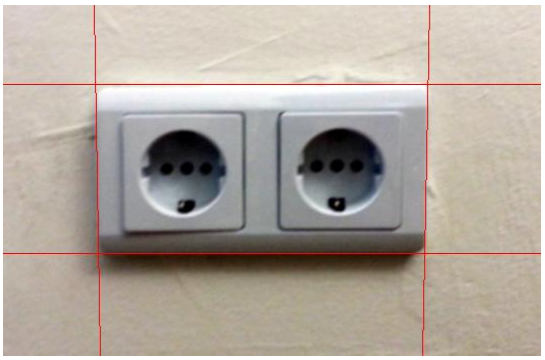
الصورة التالية تعرض نتيجة البحث عن الاي سي ضمن البورد.



الصورة التالية تعرض نتيجة البحث عن المقأومة في البورد الالكتروني.



تحويل هاف للخط Hough Line Transform



الهدف:

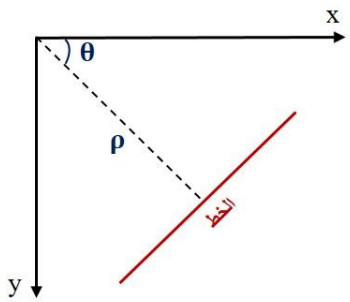
في هذا الفصل

- سنتعلم ما هو مفهوم تحويل هاف
- سنرى كيف سنستعمل تحويل هاف للكشف عن الخطوط في الصورة
- سنتعلم استخدام التوابع التالية: cv2.HoughLines()،cv2.HoughLinesP()

معلومات نظرية:

تحويل هاف هو تقنية شائعة لاكتشاف أي شكل يمكن تمثيله بمعادلة رياضية، حيث يمكنه اكتشاف الشكل حتى ولو كان الشكل متقطع أو مشوه قليلا، سنرى كيف يعمل هذا التحويل للكشف عن الخطوط التي في الصورة.

الخط يمثل رياضيا بالمعادلة $\mathbf{y = mx + c}$ أو يمكن تمثيله بالشكل البارمتري $\mathbf{\rho = x \cos\theta + y \sin\theta}$ حيث ρ تمثل مسافة المستقيم العمودي من المبدأ على الخط، و θ تمثل الزأوية بين هذا المستقيم العمودي والمحور الأفقي (x)، وقياس هذه الزأوية يكون بعكس عقارب الساعة.



نتيجة لما سبق، إذا الخط مر أسفل المبدأ ستكون قيمة ρ موجبة والزأوية ستكون أقل من ١٨٠°، أما إذا مر فوق المبدأ الزأوية بدل أن نقول إن زأويته أكبر من ١٨٠ سنأخذ زأويته بحيث تكون أقل من ١٨٠° ولكن ρ ستأخذ قيمة سالبة، وأي خط عمودي زأويته ستكون ٠° درجة وأي خط أفقي ستكون زأويته ٩٠° درجة.

كيف تتم عملية تحويل هاف للخط:

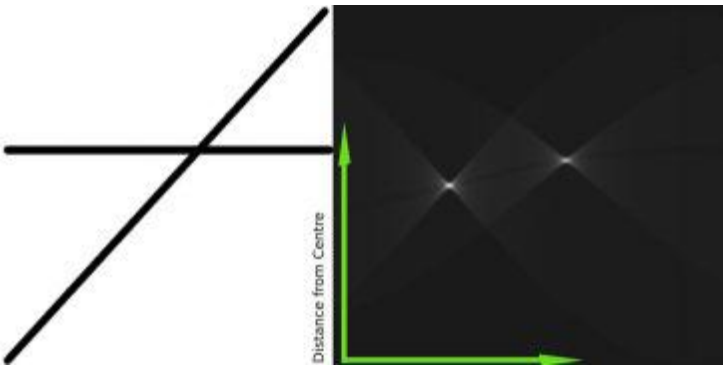
أولا علينا تمثيل كل خط وفق بارامترين هما (ρ, θ). في البداية لننشأ مصفوفة 2D أو مراكم (مُجَمِعْ) (للاحتفاظ بقيمة البارامترين ρ وθ) ونقوم في البداية بتصفيرهم، سنجعل صفوف المصفوفة rows للدلالة على ρ، وأعمدة المصفوفة columnsللدلالة على θ.

- ❖ حجم المصفوفة يعتمد على الدقة التي نحتاجها. لنفترض أننا نريد دقة الزأوية تسأوي ١ْ درجة، سنحتاج لمصفوفة بـ ١٨٠ عمود.
- ❖ بما أن أكبر مسافة ممكنة لـ ρ هي قطر الصورة، لذلك للحصول على دقة ١ بيكسل، سنحتاج لمصفوفة عدد صفوفها يسأوي قطر الصورة.
- ❖ لنفرض لدينا صورة بحجم 100x100 فيها خط أفقي بالوسط. خذ أول نقطة لك من الخط مع معرفتك لإحداثيات للنقطة (x,y) التي اخترتها.
- ❖ عوض قيم θ=0,1,2,.... ,180 في معادلة الخط، ستحصل على قيمة ρ بحسب قيمة θ التي عوضناها.
- ❖ في كل زوج (ρ, θ) يمكنك زيادة قيمة المراكم بمقدار ١ في خلايا (ρ, θ) المقابلة لها.حتى الآن يوجد في المراكم الخلية 1= (50,90) التي تكون بجانب الخلايا الأخرى التي في المراكم.
- ❖ الآن خذ النقطة الثانية من على الخط وافعل نفس الشيء الذي فعلناه في السابق، ستشاهد زيادة في قيمة الخلايا المقابلة لـ (ρ, θ) التي حصلت عليها، هذه المرة الخلية ستصبح قيمتها ٢ = (50,90).

ما تفعله بالضبط بأنك تصوت لقيم (ρ,θ) (في كل نقطة نأخذها يزيد عدد الأصوات كما في الانتخابات) (عن طريق زيادة عدد الأصوات سنعرف عدد نقاط الخط فنعلم طول الخط)، نقوم بتنفيذ نفس العملية السابقة على جميع نقاط الخط، وعند كل نقطة ستزيد قيمة الخلية (50,90) أو سيزيد صوتها، بينما الخلايا الأخرى قد يزيد عدد أصواتها أو لا، وبهذه الطريقة سنصل للنهاية وستحصل الخلية (50,90) على قيمة أعلى صوت.

لذلك إذا كنت تبحث في المراكم عن أعلى صوت ستحصل على القيمة (50,90) التي ستخبرنا بأن هناك خط في الصورة على مسافة ٥٠ من المبدأ وبزأوية ٩٠ درجة.

وهذه هي آلية عمل تحويل هاف للخط، ويمكن تطبيقه ايضاً باستخدام مكتبة Numpy، وتظهر القيم المختارة في المراكم كنقاط ذروة بيضاء فيم إذا مثلناه بصورة رمادية.



الصورة التي على اليمين توضح نتيجة تحويل هاف للمستقيمين الموضحين على الشكل اليساري.
لزيادة فهمك لألية عمل تحويل هاف يمكنك الاطلاع على ما يلي:

- **Image Courtesy: Amos Storkey** (شاهد الرسمة المتحركة التي تشرح كيف يتم زيادة قيمة الخلايا في المراكم)
- **Image courtesy: Wikipedia**
- **Explain Hough Transformation stackoverflow** (يحوي هذا الموقع على شرح مبسط على شكل رسوم توضيحية)

- **تحويل هاف في OpenCv:**

كل شيء تم شرحه سابقا موجود في التابع cv2.HoughLines()، وهو ببساطة يعيد مصفوفة لقيم (ρ, θ)، حيث ρ تقاس بالبيكسل، وθ تقاس بالراديان.

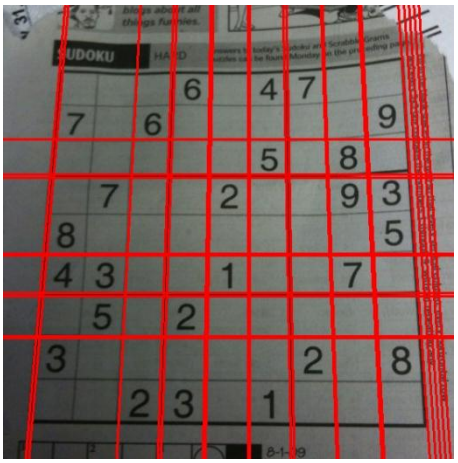
المتغير الأول يمثل صورة الدخل والتي يجب أن تكون بالصيغة الثنائية (binary image)، لذلك نطبق التعتيب أو نستخدم مكتشف حواف كاني للحصول على صورة ثنائية وذلك قبل تطبيق تحويل هاف.

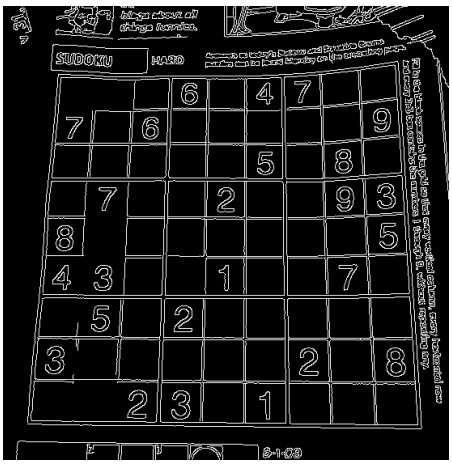
ثاني وثالث متغير هما ρ وθ على الترتيب. رابع متغير هو العتبة، والتي تعني أقل صوت يجب أن نحصل عليه لنعتبره خط. تذكر عدد الأصوات يعتمد على عدد النقاط على الخط، وهذا يعني أصغر طول للخط الذي يجب اكتشافه.

مثال١:

```
import cv2
import numpy as np

img = cv2.imread('dave.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)

lines = cv2.HoughLines(edges,1,np.pi/180,200)
for rho,theta in lines[0]:
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))

    cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)

cv2.imshow('Cany',edges)
cv2.imshow('res',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:

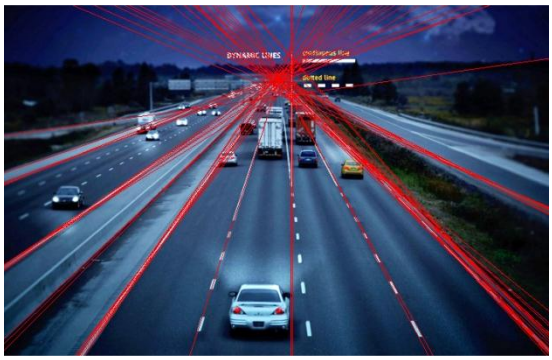


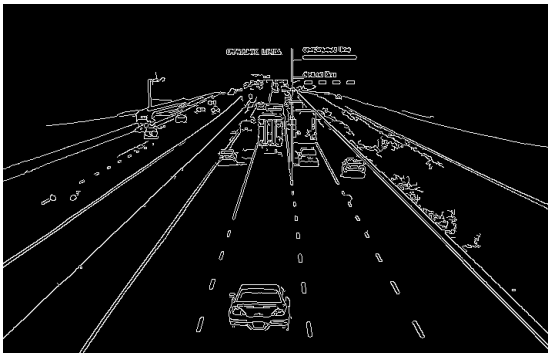


مثال٢:

```
import cv2
import numpy as np

img = cv2.imread('car2.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,500,apertureSize = 3)

lines = cv2.HoughLines(edges,1,np.pi/180,120)
for rho,theta in lines[0]:
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))

    cv2.line(img,(x1,y1),(x2,y2),(0,0,255),1)

cv2.imshow('Binary Image',edges)
cv2.imshow('res',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:



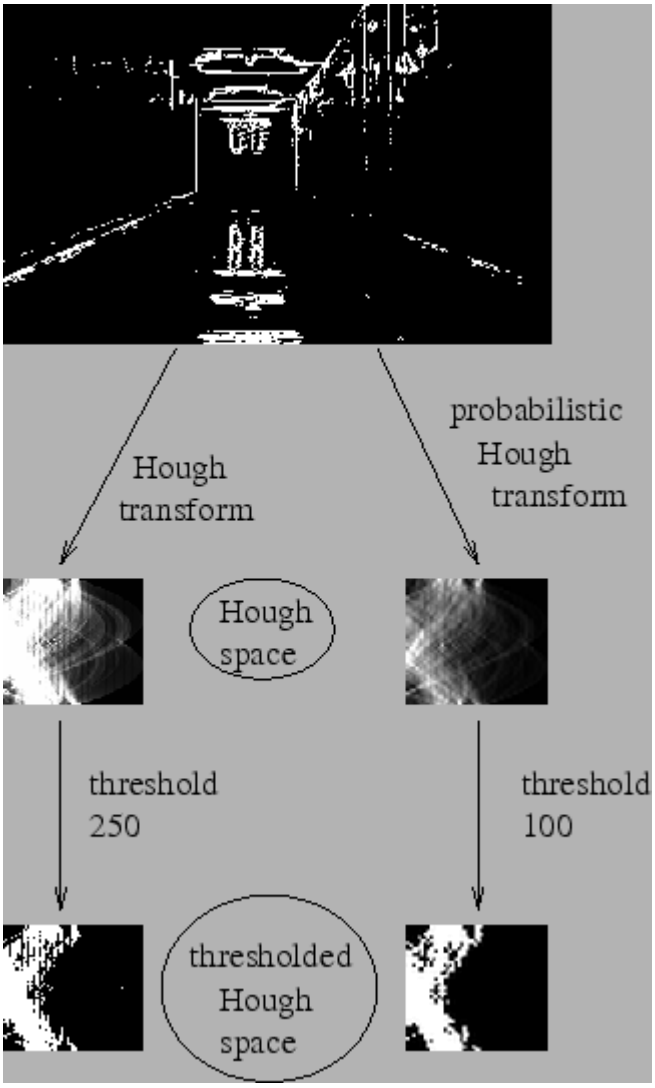


- **تحويل هاف الاحتمالي Probabilistic Hough Transform:**

في تحويل هاف السابق لاحظنا بأنه لاكتشاف مستقيم ببارامترين أنجزنا الكثير من الحسابات، على كل حال هناك تحسين لهذه التقنية ويتم ذلك بأخذ نقاط عشوائية بدل أخذ النقاط كلها، وهذا لن يؤثر على النتيجة، فكل ما علينا هو تخفيض قيمة العتبة.

شاهد للصورة التالية التي تقارن بين تحويل هاف وتحويل هاف الاحتمالي.

الصوة مأخوذة من هذا الموقع Image Courtesy: **Franck Bettinger's home page**



.

استخدام مكتبة OpenCv في اكتشاف الخطوط يعتمد على قوة الخطوط المكتشفة لذلك سنستعمل تحويل هاف الاحتمالي المتدرج.

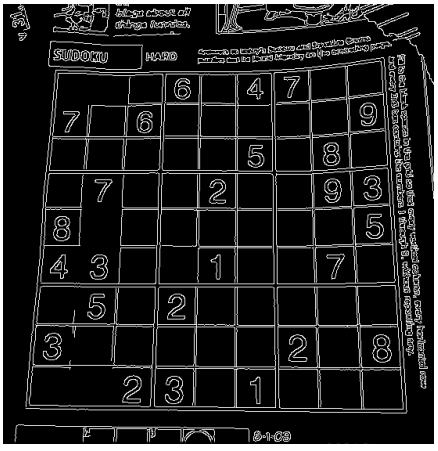
سنستعمل التابع cv2.HoughLinesP() لاكتشاف الخطوط القوية. يملك هذا التابع متغيرين هما:

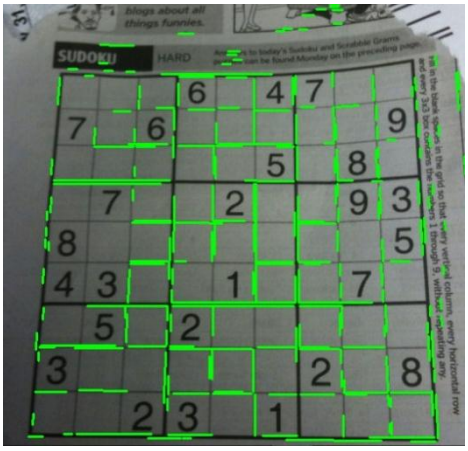
- minLineLength: أقل طول للخط ، ويرفض كل خط دون هذه القيمة.
- maxLineGap: المسافة الأعظمية المسموحة بين أجزاء الخط لاعتباره خطاً واحداً.

وأفضل شيء في هذا التابع بأنه يعيد لنا بشكل مباشر نقطة بداية الخط ونقطة نهايته، أما في الحالة السابقة حصلنا فقط على بارمترات الخطوط، وكان هناك صعوبة في إيجاد كل النقاط. أما الآن أصبح بإمكاننا أن نحصل على هذه النقاط بسهولة.

```
import cv2
import numpy as np

img = cv2.imread('dave.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)

minLineLength = 100
maxLineGap = 10
lines = cv2.HoughLinesP(edges,1,np.pi/180,100,minLineLength,maxLineGap)

for x1,y1,x2,y2 in lines[0]:
    cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)

cv2.imshow('Binary Image',edges)
cv2.imshow('res',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

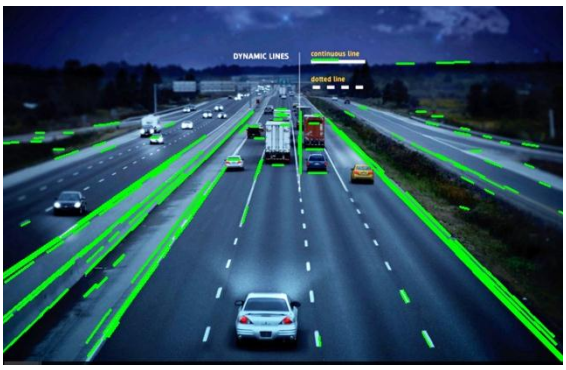
النتيجة:

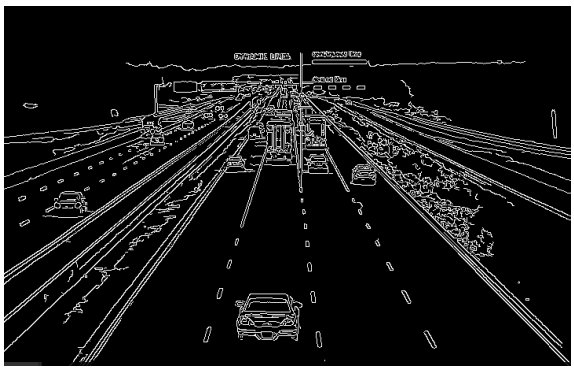




مثال٢:

```
import cv2
import numpy as np

img = cv2.imread('car2.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)

minLineLength = 100
maxLineGap = 10
lines = cv2.HoughLinesP(edges,1,np.pi/180,100,minLineLength,maxLineGap)

for x1,y1,x2,y2 in lines[0]:
    cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)

cv2.imshow('Binary Image',edges)
cv2.imshow('res',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```





مراجع إضافية:

- Hough Transform on Wikipedia

تحويل هاف للدائرة Hough Circle Transform

الهدف:

- سنتعلم استخدام تحويل هاف للإيجاد الدوائر في الصورة.
- سنتعلم استخدام التابع:cv2.HoughCircles() .

نظري:

الدائرة تمثل رياضيا كالتالي $(x - x_{center})^2 + (y - y_{center})^2 = r^2$
حيث (x_{center} , y_{center}) هما مركز الدائرة، و r هو نصف قطر الدائرة.

نلاحظ وجود ثلاث بارامترات لذلك سنحتاج لمراكم ثلاثي البعد 3D لإجراء تحويل هاف، ولكن هذا الأمر لن يكون فعال بشكل جيد، لذلك مكتبة OpenCV تستخدم طريقة هاف للتدرج (Hough Gradient Method)، والتي تعتمد على تدرج الحواف (زيادة الحواف).

```
cv2.HoughCircles(image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]]])
```

التابع الذي سنستخدمه هنا هو cv2.HoughCircles(). متغيراته هي:

Image: صورة الدخل. يجب أن تكون الصورة في المستوي الرمادي.

Method: طريقة المستخدمة لاكتشاف الدوائر التي في الصورة ، حاليا سنتستعمل الطريقة cv2.HOUGH_GRADIENT.

dp: هذا البارامتر يمثل نسبة العكس ويؤثر على قرار المراكم بتحديد دقة الصورة. فإذا كان dp=1 فسيحدد المراكم دقة الصورة كما في صورة الدخل، أما إذا كان dp=2 فسيحدد المراكم دقة الصورة بنصف حجم الصورة الأصلية.

minDist: أقل مسافة مسموحة بين مركز الدائرة المكتشفة. إذا كانت قيمة هذا البارمتر صغيرة جدا ستلاحظ بأن هناك دوائر كثيرة بجوار بعضها. وإذا كانت المسافة كبيرة جداً ستلاحظ اختفاء بعض الدوائر.

circle_storage: في لغة ال C يمثل تابع ذاكرة التخزين التي سيخزن سلسلة الدوائر المكتشفة (لا يهمنا)

param1: البارمتر الثاني يحدد وظيفة هذا البارمتر. وبما أننا سنختار طريقة CV_HOUGH_GRADIENT. فهذا البارمتر سيمثل أعلى قيمة للعتبة في مكتشف الحواف كاني، أما أصغر قيمة للعتبة ستكون أصغر بمرتين من قيمة العتبة العليا.

param2: البارمتر الثاني يحدد وظيفة هذا البارمتر. وبما أننا سنختار طريقة CV_HOUGH_GRADIENT. فهذا البارمتر يحدد قيمة عتبة المراكم. إذا اخترنا أصغر قيمة للعتبة سنحصل على أكبر عدد من الدوائر المكتشفة (بما في ذلك الدوائر الكاذبة). وإذا اخترنا أكبر قيمة للعتبة سنحصل على أكبر احتمال لأن تكون فعلاً دائرة مكتشفة.

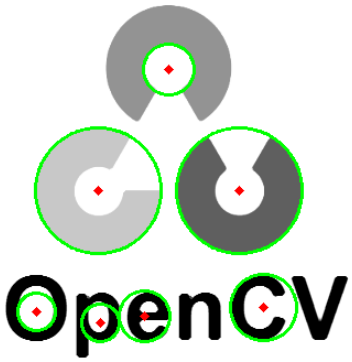
minRadius: أصغر قياس لنصف قطر الدائرة.

maxRadius: أكبر قياس لنصف قطر الدائرة.

الكود:

```
import cv2
import numpy as np

img = cv2.imread('OpenCV_Logo.png',0)
img = cv2.medianBlur(img,5)
cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)

circles = cv2.HoughCircles(img,cv2.cv.CV_HOUGH_GRADIENT,1,20,
param1=50,param2=30,minRadius=0,maxRadius=0)

circles = np.uint16(np.around(circles))

for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv2.circle(cimg,(i[0],i[1]),2,(0,0,255),3)

cv2.imshow('detected circles',cimg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

النتيجة:



ملاحظة: في مكتبة **OpenCv** الإصدار ٢ قم بتمرير المتغير الثاني للتابع بالشكل **cv2.cv.CV_HOUGH** أم إذا كان إصدار مكتبة **OpenCv** الذي تستعمله هو الإصدار ٣ فقم بتمرير المتغير الثاني بالشكل **.cv2.HOUGH_GRADIENT**

ملاحظة: مررنا للتابع **cv2.HoughCircles** البارمترين **minRadius** و **maxRadius**بقيمتين **0,0**، وهذا يعني بأن يأخذ كل الدوائر سواء كانت صغيرة أم كبيرة.

تجزئة الصورة باستخدام خوارزمية Watershed

الهدف:

- تعلم تقسيم الصورة بطريقة التجمعات المائية(Watershed).
- تعلم استخدام التابع cv2.watershed().

مقدمة نظرية:

خوارزميةwatershed هي خوارزمية تقليدية تستخدم لتجزئة الصورة لأقسام وتفيدنا هذه الخوارزمية عند استخراج الأجسام المتلامسة أو المتداخلة فيما بينها كالقطع لنقدية في الصورة التي في الأعلى.

يمكننا تمثيل أي صورة رمادية بشكل سطح طوبوغرافي بحيث الشدات اللونية العالية تمثل القمم والتلال والشدات اللونية المنخفضة تمثل الوديان. سنبدأ بملء كل الوديان المنعزلة (القيمة الصغرى المحلية) بالماء وكل وداي بلون مميز عن الأخر (الوسم). سترتفع المياه اعتمادا على ارتفاع القمم (التدرجات اللونية) المحيطة بالوديان وستبدأ المياه الملونة بألوان مختلفة بالاختلاط مع بعضها، ولتجنب هذا الأمر سنبني حواجز عند الأماكن التي يتم فيها الاختلاط. ونستمر بصب الماء وبناء الحواجز حتى غمر كل القمم بالماء، وعندها فإن الحواجز التي تم بناءها ستعطينا نتيجة التقطيع وهذا هو المقصود بكلمة watershed (مجمعات المياه). لتفهم كيف يتم هذا الأمر بشكل أوضح يمكنك زيارة صفحة الويب CMM webpage on watershed التي تحوي على رسوم متحركة وتوضيحية.

العملية السابقة قد تعطينا تقطيعا مفرطا بسبب وجود ضجيج في الصورة، لذلك وفرت OpenCv للمستخدم بأن يحدد المناطق التي يريد دمجها والمناطق التي لا يريد دمجها. وهذا هو تقطيع الصورة التفاعلي، حيث نحدد المناطق التي تشكل الجسم الأمامي بلون وحد، ثم نحدد المناطق التي تشكل خلفية الجسم الأكيدة بلون مختلف، أما المناطق التي لم نتأكد منها فنحددها بقيمتها ومن ثم نطبق الخوارزمية، بعد التطبيق سنرى الحدود ستتحدث، وفي النهاية ستحدد حدود الجسم وستأخذ القيمة 1- .

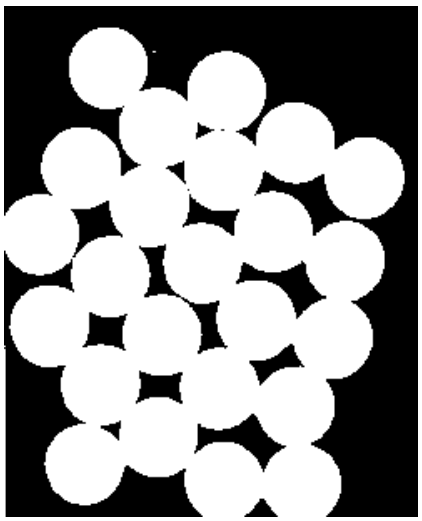
الكود:

في المثال التالي سنشاهد كيف سنستخدم تحويل المسافة مع خوارزمية watershed لتقطيع (لتجزئة) الأجسام المتلامسة. ليكن لدينا الصورة التالية للنقود، كل قطعة نقدية متلامسة مع الأخرى، وإذا طبقنا التعتيب ستبقى كما هي متلامسة.

سنبدأ بإيجاد الحدود التقريبية للنقود وذلك باستخدام تعتيب أوتسو.

```
import cv2
import numpy as np

img = cv2.imread('coins.png')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh =
cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
```

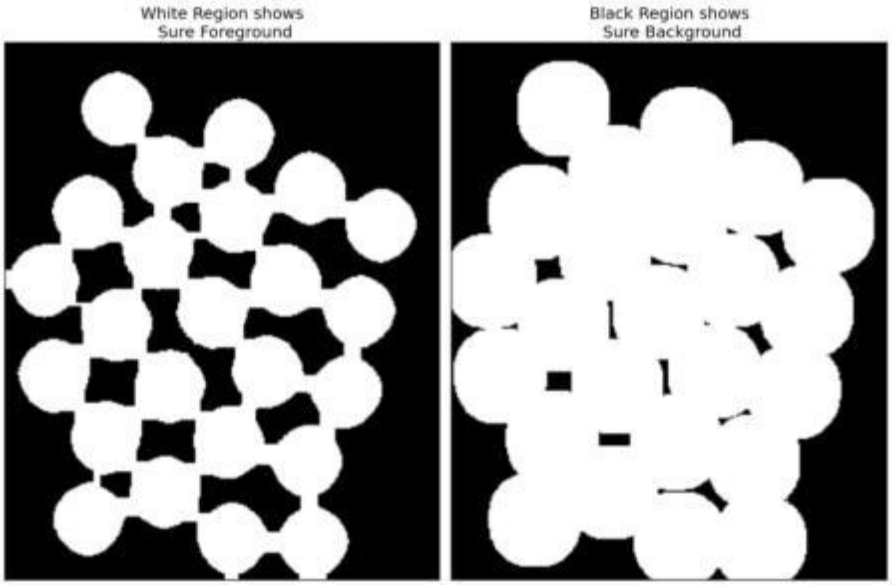
النتيجة:



الآن سنحتاج لإزالة أي ضجيج باللون الأبيض في الصورة، لذلك سنقوم بتطبيق عملية الفتح على الشكل السابق، ولإزالة الثقوب السوداء الصغيرة في الأجسام نقوم بعملية الإغلاق.والآن نحن متأكدين بأن المناطق القريبة من مركز الجسم هي مناطق أمامية، والمناطق البعيدة عن الجسم هي الخلفية.

المناطق الوحيدة التي لسنا متأكدين منها هل هي أمامية أم خلفية هي منطقة حدود العملة.
لذلك نحتاج لاستخراج المناطق المتأكدين على أنها قطعة النقود (تمثل المنطقة الأمامية)، سنستعمل عملية الحت Erosion لإزالة البيكسلات المحيطة بقطعة النقود وسيكون هذا االأمر مفيد جداً في حال كانت القطع لا تمس بعضها، أما في حال القطع تمس بعضها فسنحتاج لعمل تحويل للمسافة ثم نطبق عتبة مناسبة.

بعد ذلك علينا إيجاد منطقة الخلفية الأكيدة، لذلك سنقوم بعملية التمديد dilate للنتيجة، وبهذا يمكننا التأكد من أن المنطقة التي في الخلفية هل هي خلفية أكيدة أم لا، لأن منطقة الحدود مزالة بين الجسم والخلفية. شاهد الصورة التالية التي توضح الأمرين السابقين.



المناطق المتبقية التي لا نعرف هل هي جسم أم خلفية (هي المناطق التي تلتقي بها القطع النقدية والخلفية أو حتى القطع النقدية مع بعضها)، خوارزمية watershed ينبغي أن تكون قادرة على إيجادها.

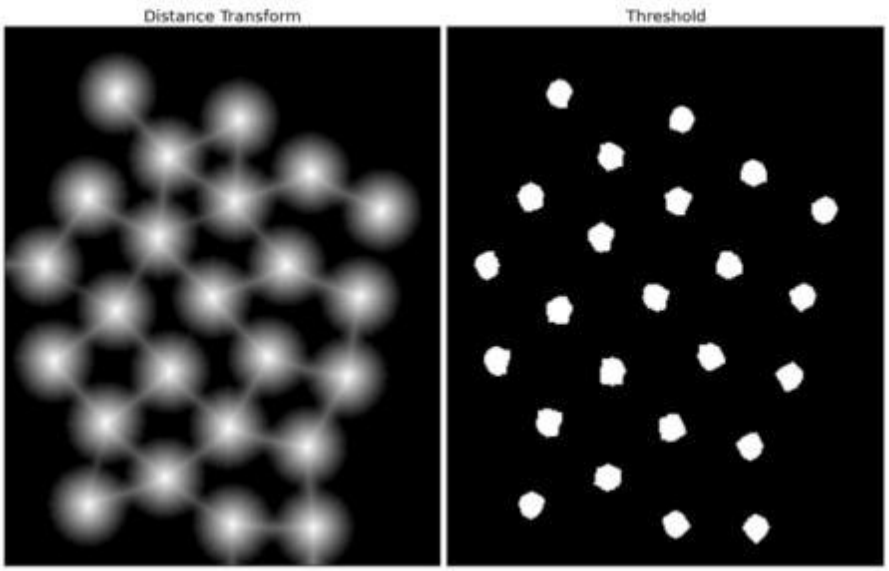
ندعو هذه المناطق المتبقية بالحدود، ويمكن معرفتها من خلال طرح الخلفية الأكيدة من الأمامية الأكيدة.

```
# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)

# sure background area
sure_bg = cv2.dilate(opening,kernel,iterations=3)

# Finding sure foreground area
dist_transform = cv2.distanceTransform(opening,cv2.cv.CV_DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)

# Finding unknown region
sure_fg = np.uint8(img)
unknown = cv2.subtract(sure_bg,sure_fg)
```

ملاحظة: في التابع **distanceTransform()** ضمن مكتبة **OpenCv** الإصدار ٣ بدل أن تمرر لهذا التابع المتغبير **cv2.cv.CV_DIST_L2** قم بتمرير المتغير **cv2.DIST_L2**.

النتيجة:

وبتعتيب الصورة نحصل على مناطق القطع النقدية والتي متأكدين من أنها قطع نقدية ويتم فصل القطع النقدية عن بعض (في بعض الأحيان قد تكون مهتم فقط باستخراج الجزء الأمامي دون فصل الأجسام عن بعضها. في هذه الحالة لا تحتاج لأن تستعمل التحويل المسافي، فقط ما تحتاجه هو استخدام عملية الحت Erosion لاستخراج المنطقة الأمامية الأكيدة).



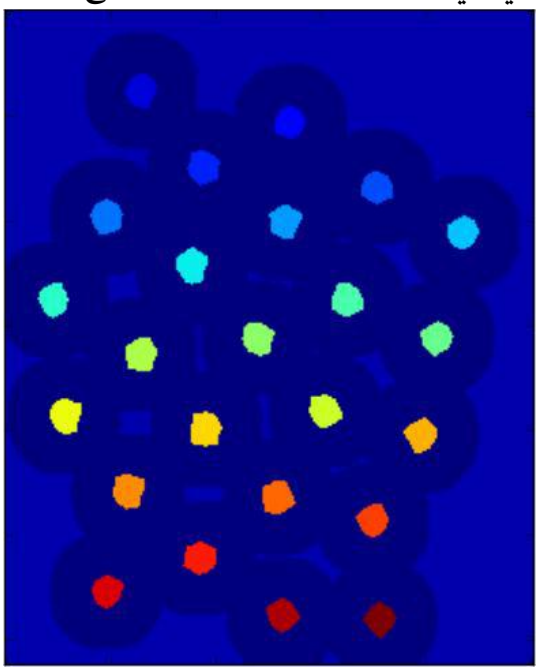
الأن أصبحنا متأكدين من مناطق القطع النقدية والخلفية. لذلك سننشأ علامة Maker (مصفوفة بنفس حجم الصورة الأصلية ولكن بنوع بيانات int32) ونوسم المناطق داخلها.المناطق (الأمامية أو الخلفية) المتأكدين منها نحددها بأي عدد صحيح موجب، ولكن بأرقام مختلفة لكل جسم. والمناطق التي لسنا متأكدين منها ندع قيمتها صفر.

ولذلك سنستخدم التابع cv2.connectedComponents() والذي يحدد خلفية الصورة بالقيمة صفر، والأجسام الأخرى يحددها بأعداد صحيحة ابتداءً من 1.ولكن إذا كانت قيمة الخلفية 0 ستعتبرها خوارزمية watershed غير معلومة، لذلك بدلاً من ذلك سنحدد المناطق غير المعلومة بـ 0.

```
# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)

# Add one to all labels so that sure background is not 0, but 1
markers = markers+1

# Now, mark the ' of unknown with zero
markers[unknown==255] = 0
```

النتيجة:

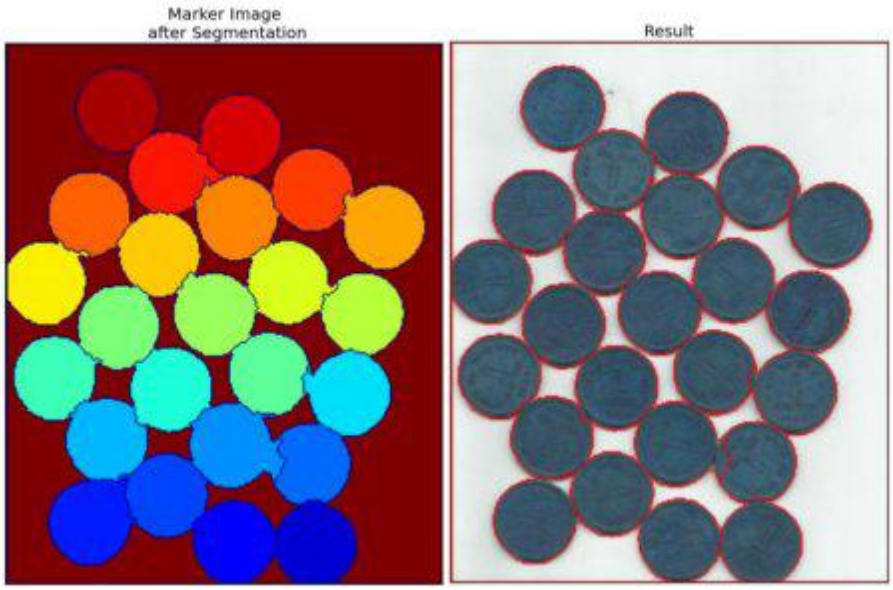
الصورة التالية تبين النتيجة التي تعرض خريطة الألوان JET، حيث الأزرق الغامق يعرض المناطق الغير معلومة(المجهولة)، والقطع النقدية المتأكدين منها تلون بألوان مختلفة (قيم مختلفة)، يتم عرض المناطق المتبقية والتي هي خلفيات أكيدة بلون أزرق أفتح بالنسبة للمناطق المجهولة.



الآن لدينا علامة Makerجاهزة، وحان الوقت لتنفيذ الخطوة النهائية وتطبيق خوارزمية watershed، عندها صورة العلامة Maker ستتعدل، والمنطقة الحدودية ستأخذ القيمة 1- .

```
markers = cv2.watershed(img,markers)
img[markers == -1] = [255,0,0]
```

لاحظ النتيجة سترى أن بعض القطع تم تقطيعها بشكل مثالي وأخرى ليست كذلك.



مراجع إضافية:

- CMM page on [Watershed Transformation] (يحوي هذ المرجع على صور متحركة مع العديد من الأمثلة التي تبسط لشرح)

الاشتقاق الأمامي التفاعلي باستخدام خوارزمية GrabCut

الهدف:

- تعلم خوارزمية GrabCut لاستخراج الأجسام الأمامية في الصورة.
- سننشئ تطبيق تفاعلي باستخدامها

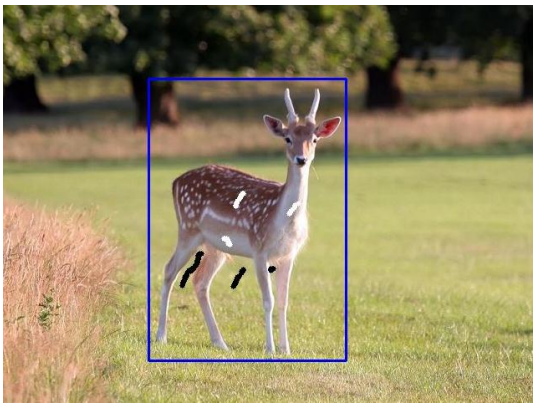
النظري:

الهدف من خوارزمية GrabCut استخراج الأجسام الأمامية التي في الصورة بأقل تفاعل من المستخدم. كيف يقوم المستخدم بتنفيذ هذه الخوارزمية؟

في البداية يرسم المستخدم مستطيل حول المنطقة الأمامية (ينبغي أن تكون المنطقة الأمامية (الجسم) تماما داخل المستطيل)، ومن ثم يقوم بتكرار تطبيق الخوارزمية حتى الوصول لأفضل نتيجة.

لكن في بعض الأحيان قد لا نحصل على النتيجة المرغوبة بسبب الأخطاء الحاصلة عند تعليم الخلفية كأمامية، أو بالعكس. سنحتاج في هذه الحالة لإعادة تعليم للنتيجة، مثل التحديد يدويا بضربات (التحديد بكبسة الفأرة) تحدد أماكن الأخطاء في الصورة الناتجة، لذلك بعد القيام بهذه العملية نحصل على نتيجة أدق.

شاهد المثال التالي الذي سنقوم فيه أولا برسم مستطيل حول الجسم، ومن ثم نقوم بإضافة اللمسات الأخيرة عن طريق ضربات الفأرة. حيث الضربات باللون الأبيض لتحديد المنطقة الأمامية، والضربات التي باللون لأسود لتحديد الخلفية، ومن ثم سنحصل على نتيجة جيدة.

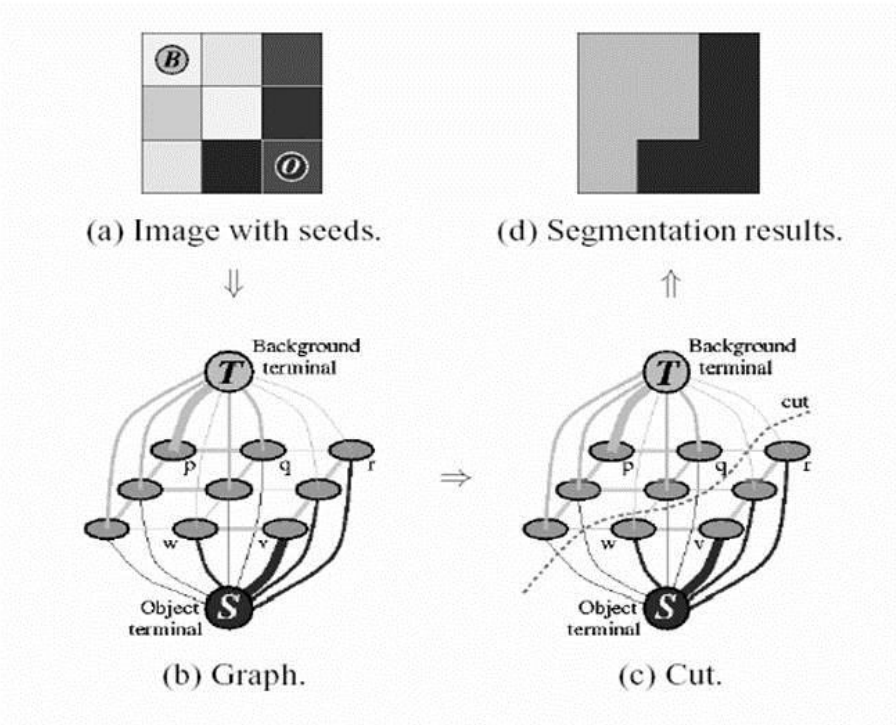


ولكن ما الذي حلَّ بالخلفية؟

- أولاً يرسم المستخدم مستطيل وكل شيء خارجه يعد خلفية أكيدة، لذلك يجب تضمين الجسم كاملا داخل المستطيل. كل شيء دخل المستطيل غير معروف فحسب البيانات التي يدخلها المستخدم يتم تعيين المنطقة الأمامية والخلفية.
- يقوم الحاسوب بعد ذلك بإعطاء علامات بالاعتماد على البيانات المعطاة، بحيث يقوم بتحديد المنطقة الخلفية من الأمامية.
- حاليا سنستخدم نمط الخلائط الغأوسي (GMM) لتحديد نمط (شكل) المنطقة الخلفية والأمامية.
- اعتماد ا على البيانات المعطاة يتعلم GMM وينشأ توزعا جديدا للبيكسلات، حيث تصنف البيكسلات الغير معرفة إما كمنطقة أمامية محتملة أو خلفية محتملة وذلك بالاعتماد على علاقتها مع البيكسلات المعرفة بدلالة إحصائيات الألوان (مثل التجميع بعناقيد).
- يتم بناء الرسم البياني من توزع البيكسلات هذا. تتمثل العقدة بالبيكسل. تضاف عقدتين إضافية وهما عقدة المصدر وعقدة المصرف. كل بيكسل أمامي يتصل بعقدة المصدر وكل بيكسل خلفي يتصل بعقدة المصرف.
- وزن(قيمة) الحواف التي تتصل بعقدة المنبع /المصرف تعطي احتمالية أن يكون البيكسل ينتمي للمنطقة الأمامية / الخلفية. تحدد الأوزان بين البيكسلات بمعلومات الحافة أو تشابه البيكسل، فإذا كان هناك فرق كبير بألوان البيكسلات المترابطة فالحافة بينهم ستكسب وزناً أخف.

- خوارزمية mincutتستعمل لتقطيع الرسم البياني، حيث تقطعه لعقدتي منبع ومصرف بأقل تابع تكلفة. وتابع التكلفة هو مجموع كل أوزان الحواف المقطوعة، وبعد القطع، كل البيكسلات المتصلة بعقدة المصدر تصبح أمامية، والمتصلة بعقدة المصرف تصبح خلفية.
- تستمر العملية حتى يتقارب التصنيف.

الصورة التالية توضح هذه العملية.



مثال توضيحي:

لتطبيق خوارزمية GrabCut في OpenCv نستخدم التابع()cv2.grabCut ، ومتغيرات هذا التابع هي:

- img: صورة الدخل
- mask: قناع الصورة والذي يحدد المناطق الأمامية والخلفية أو احتماليتها (أي احتمال هل هي امامية ام خلفية) ، ويتم ذلك عن طريق الأعلام (flags) التالية:
- cv2.GC_PR_FGD cv2.GC_BGD, cv2.GC_FGD, cv2.GC_PR_BGD,
- أو ببساطة نمرر الأرقام التالية 0,1,2,3 بالترتيب.

- rect: وهي إحداثيات المستطيل الذي يتضمن الجسم الأمامي بالصيغة (x, y, w, h).
- bdgModel, fgdModel: هي مصفوفات تستخدمها الخوارزمية داخليا ،يمكنك فقط تمرير مصفوفة صفرية بصيغة np.float64 وبحجم (1.65).
- iterCount: عدد المرات التي يجب أن تكرر الخوارزمية فيها.
- mode: يجب أن تكون cv2.GC_INIT_WITH_RECT أو cv2.GC_INIT_WITH_MASK أو وضع مركباً والذي يحدد ما إذا كان سيتم تحديد الجسم الامامي عن طريق رسم مستطيل أو عن طريق ضربات (كبسات) أزرار الفأرة.

في البداية سنبدأ بوضع المستطيل، سنحمل الصورة، وننشأ قناع للصورة، وننشأ المصفوفتين bdgModel وfgdModel، ثم نمرر بارمترات المستطيل (إحداثياته وطوله وعرضه).نسمح بتشغيل الخوارزمية ٥ مرات.

النمط mode يجب أن يكون cv2.GC_INIT_WITH_RECT لأننا نستخدم مستطيل.ثم نشغل الـ GrabCut، عند ذلك تُعَدَل صورة القناع.

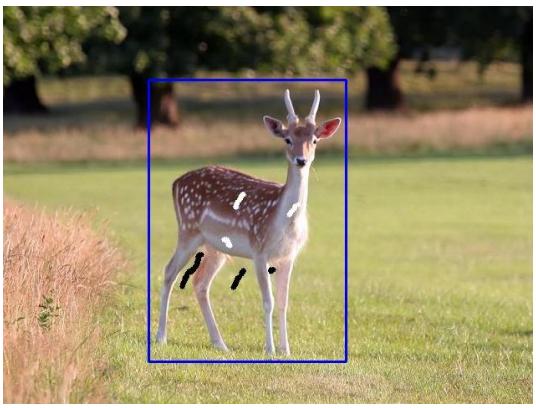
في الصورة الجديدة المعدلة للقناع، البيكسلات سوف تعلم بأربع أعلام تدل على الخلفية أو المقدمة (كما في السابق).بعدها نعدل القناع بحيث تصبح البيكسلات التي أرقامها ٠ و٢ خلفية أكيدة (نضع فيهم القيمة صفر)، أما البيكسلات بأرقام ١ و٣ تكون أمامية أكيدة (نضع فيها القيمة ١).

الآن أصبح لدينا القناع النهائي جاهز. كل ما علينا ضربه بالصورة الأصلية لنحصل على نتيجة التقطيع المرغوبة.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('deer.jpg')
mask = np.zeros(img.shape[:2],np.uint8)

bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)

rect = (50,50,450,290)
cv2.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_RECT)

mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
img = img*mask2[:,:,np.newaxis]

plt.imshow(img),plt.colorbar(),plt.show()
```

لاحظ بأنه ظهر في الصورة قسم من الخلفية لا نريد إظهاره لذلك يجب علينا استخدام الطريقة الثانية والتي تتم بتحديد القناع عبر أداة خاصة (كالمأوس) بحيث نحدد الأجسام الأكيدة باللون الأبيض والخلفية الأكيدة باللون الأسود ونكتب ما يلي:

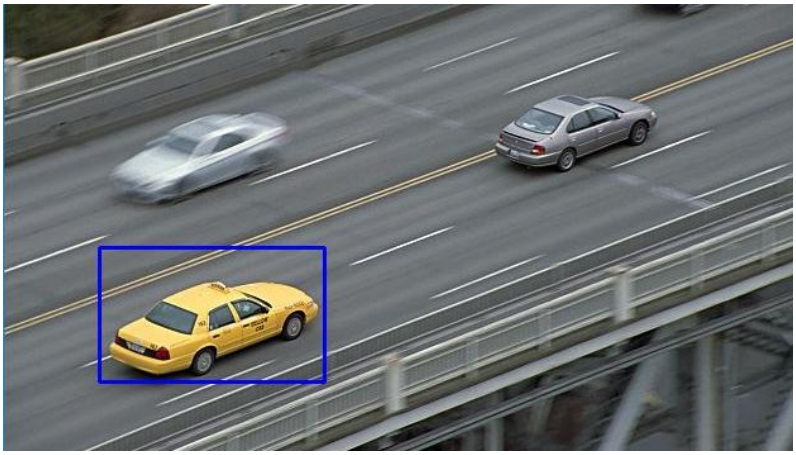
```
# newmask is the mask image I manually labelled
newmask = cv2.imread('newmask.png',0)

# whereever it is marked white (sure foreground), change mask=1
# whereever it is marked black (sure background), change mask=0
mask[newmask == 0] = 0
mask[newmask == 255] = 1

mask, bgdModel, fgdModel =
cv2.grabCut(img,mask,None,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_MASK)

mask = np.where((mask==2)|(mask==0),0,1).astype('uint8')
img = img*mask[:,:,np.newaxis]
plt.imshow(img),plt.colorbar(),plt.show()
```

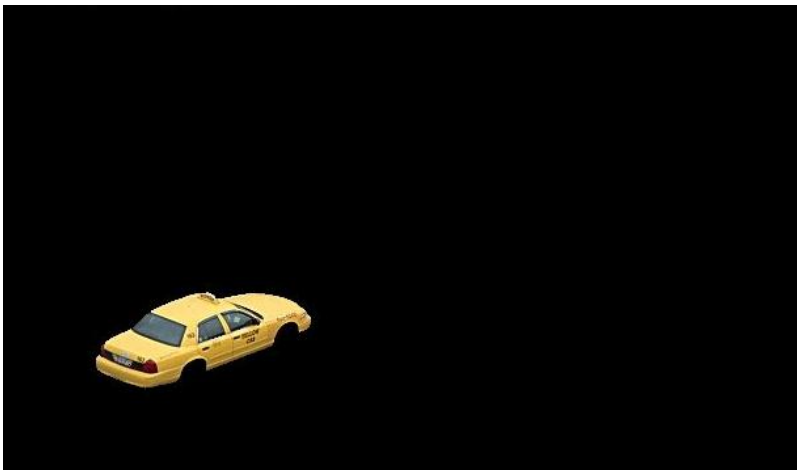


يمكنك تحميل كود استخراج الجسم لأي صورة عن طريق تحديد المنطقة الأمامية من الخلفية بواسطة ضربات الفأرة من خلال الرابط التالي:

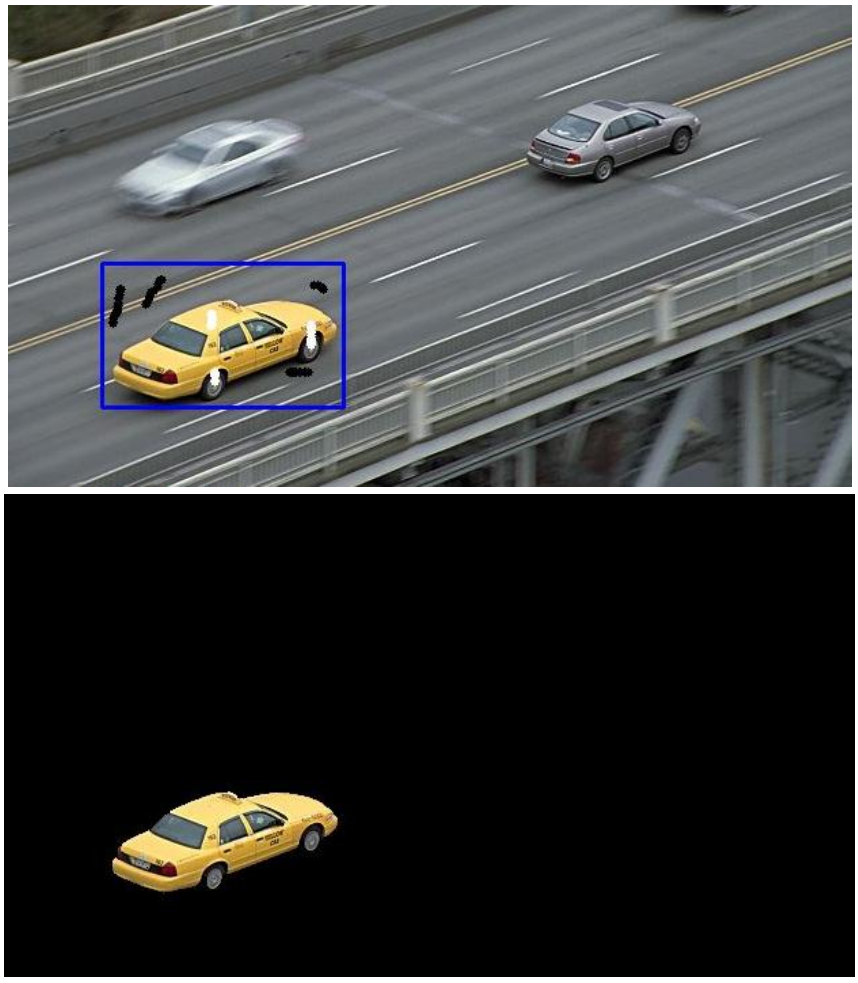
https://github.com/downingstreet/GrabCut/blob/master/grabcut.py

كل ما عليك اتباع الخطوات التي يعرضها الكود عند تشغيله.





لاحظ في الصورة الناتجة السابقة بأنه تم اختفاء دواليب السيارة لذلك سنقوم بعمل ضربات بزر الفأرة لتحديد الدواليب كنطقة أمامية.



لاحظ الآن كيف ظهرت دواليب السيارة في الصورة.

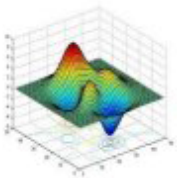
الفصل الرابع

" الإبداع هو النظر في المألوف بطريقة غير مألوفة "

أحمد الشقيري

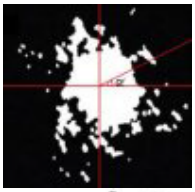
الفصل الرابع: الإطارات في OpenCv

الهدف:

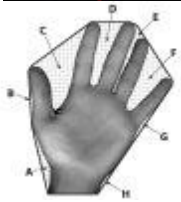


- تعلم إيجاد ورسم ملامح الشكل.

- تعلم إيجاد ملامح مختلة للإطارات، مثل المساحة، المحيط، محيط مستطيل، .....الخ



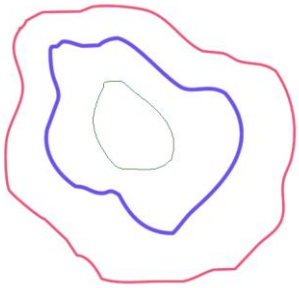
- تعلم إيجاد خصائص الإطارات، مثل الصلابة أي الكثافة، .....



- تعلم إيجاد تشوهات التحدب، نقطة مضلع الاختبار، تطابق الأشكال المختلفة

- التعرف على التدرج الهرمي للإطارات.

الإطارات (ملامح الجسم) في OpenCv



الهدف:

- فهم ما هي الإطارات.
- تعلم إيجاد ورسم الإطارات.
- سنتعلم استخدام التوابع: cv2.findContours()،cv2.drawContours() .

ما هي الإطارات؟

يمكن شرحها ببساطة بأنها عبارة عن منحنيات تضم كل النقاط المتصلة مع بعضها (على طول الحدود)، والتي تملك نفس اللون أو الشدة، وهذه الإطارات هي أدوات مفيدة لتحليل الشكل، واكتشاف الأجسام والتعرف عليها.

ملاحظات هامة:

- للحصول على أفضل نتيجة استخدم الصور الثنائية، لذلك قبل أيجاد الإطارات سنستخدم عملية التعتيب أو مكتشف حواف كاني.
- تابع إيجاد الحواف يغير الصورة الأصلية، لذلك إذا أردت الحفاظ على الصورة قم بعمل نسخة احتياطية للصورة وخزنه في متغير آخر.
- إيجاد الإطارات في OpenCv يشبه إيجاد جسم أبيض من خلفية سوداء، لذلك يجب أن يكون الكائن الذي تبحث عنه باللون الأبيض والخلفية يجب أن تكون سوداء (لا تنسى هذا الأمر).
- إذا كانت الصورة الثنائية غير واضحة يمكنك الاستعانة بالتحويلات التي تتم على الشكل مثل عملية التمديد Dilation.

دعنا نشاهد كيف سنوجد إطارات (ملامح) صورة ثنائية في الكود التالي الذي سيوضح لنا طريقة إيجاد الإطارات:

```
import numpy as np
import cv2
im = cv2.imread('test.jpg')

imgray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(imgray,127,255,0)
# Get all the contours found in the thresholdimage
# syntax for opencv2
contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
# syntax for opencv3
#thresh ,contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

شرح الكود:

```
imgray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(imgray,127,255,0)
```

سنحول الصورة للون الرمادي ثم سنقوم بتحويل الصورة للصيغة الثنائية (أبيض وأسود) وذلك عن طريق التعتيب.

```
contours, hierarchy =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

نقوم بعملية إيجاد الإطارات عن طريق التابع cv2.findcontours.

شرح التابع cv2.findcontours():

- المتغير الأول يمثل صورة الدخل (يجب أن تكون بالصيغة الثنائية).
- المتغير الثاني يمثل نمط استرداد الإطار (سنشرحه لاحقا).
- المتغير الثالث هو تقريب الإطار (سنشرحه لاحقا).

وقيم الخرج التي يعيدها التابع هي الإطارات contours وتسلسل الإطارات الهرمي hierarchy.

الإطارات المكتشفة تكون على شكل قائمة بايثون، وكل إطار مفرد هو عبارة عن مصفوفة Numpy لإحداثيات نقاط حدود الجسم.

كيف ترتسم الإطارات؟

لرسم الإطارات نستخدم التابع cv2.drawContours. ويمكننا أن تستخدم هذا التابع لرسم أي شكل لدينا نقاطه المحيطة.

شرح التابع cv2.drawContours:

- المتغير الأول هو صورة الدخل.
- المتغير الثاني هو الإطارات المطوب رسمها والتي تمرر لهذا المتغير كقائمة بايثون.
- المتغير الثالث هو فهرس الإطارات (مفيد عند رسم إطار مفرد بحيث نمرر له ترتيب الإطار المراد رسمه، لرسم كل الإطارات نمرر لهذا المتغير القيمة -1) .
- المتغيرات الأخرى هي اللون والسماكة.... الخ.

لرسم كل الإطارات التي في الصورة نكتب:

```
img = cv2.drawContours(img, contours, -1, (0,255,0), 3)
```

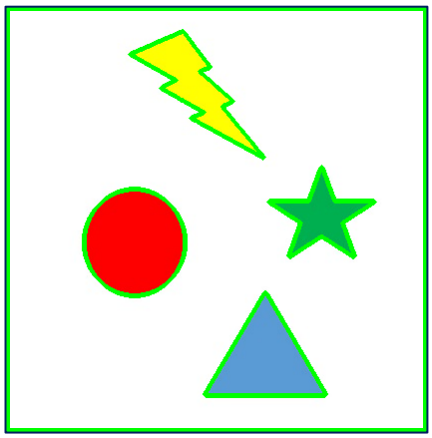
لرسم إطار مفرد نكتب:

```
img = cv2.drawContours(img, contours, 3, (0,255,0), 3)
```

في معظم الأحيان سنستخدم أسلوب أفضل لرسم الإطارات وهو كالتالي:

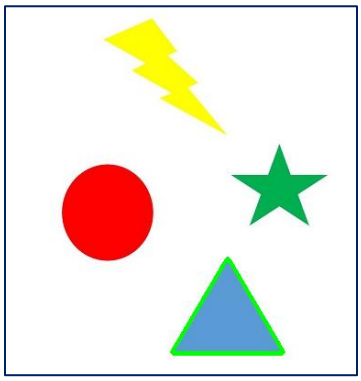
```
cnt = contours[4]
img = cv2.drawContours(img, [cnt], 0, (0,255,0), 3)
```

```
import numpy as np
import cv2
img = cv2.imread('shape.png')

imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(imgray,127,255,0)
contours, hierarchy =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)

cv2.drawContours(img, contours, -1, (0,255,0), 3)

cv2.imshow('counter',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

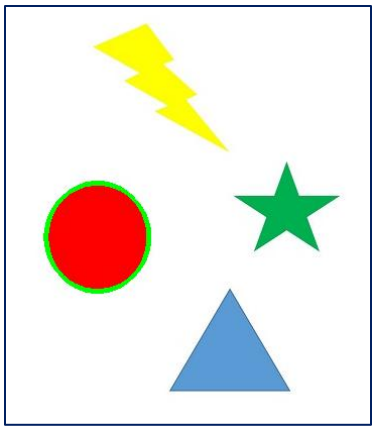


ملاحظة هامة: يجب اختيار درجة التعتيب بعناية حتى نحصل على إطارات الصورة التي نريدها.

الصورة السابقة تحتوي على 5 إطارات. يمكننا رسم إطار مفرد حول الجسم الذي نريده وذلك عن طريق المتغير الثالث للتابع cv2.drawContours.



فإذا أردنا رسم الإطار الثاني نمرر للمتغير الثالث القيمة contours[1].



لرسم الإطار الثالث نمرر له القيمة contours[2] وهكذا.

طريقة تقريب الإطار:

هذا هو المتغير الثالث في تابع إيجاد الإطارات cv2.findcontours().

ما هي وظيفته وما هي الفائدة منه؟

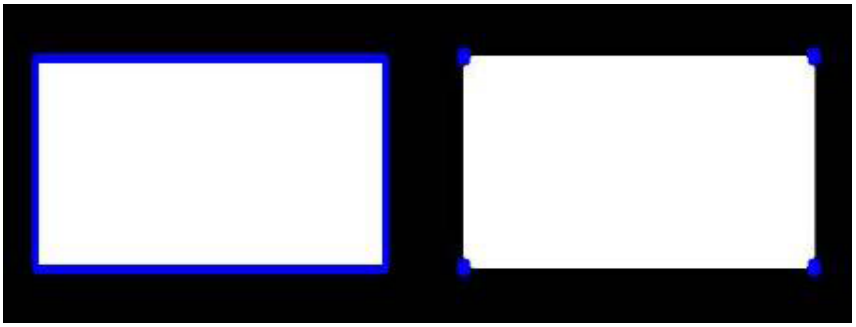
قلنا سابقاً أن بأن الإطارات عبارة عن حدود الشكل والتي تملك نفس الشدة اللونية. كما قمنا بتخزين إحداثيات النقاط التي تشكل حدود الشكل، ولكن هل خزنا جميع إحداثيات النقاط التي تشكل هذه الحدود؟

هذا هو ما سيتم تحديده باستخدام طريقة تقريب الإطارات.إذا مررنا لهذا المتغير القيمة cv2.CHAIN_APPROX_NONE سوف يخزن جميع نقاط الحدود.

ولكن هل نحتاج لتخزين جميع النقاط؟

على سبيل المثال أوجد الإطار لخط مستقيم، هل تحتاج لكل النقاط التي على الخط حتى نستطيع تمثيل هذا الخط؟ نحن بحاجة فقط لنقطتين، نقطة في بداية الخط ونقطة أخرى في نهايته، وهذا ما يفعله التابع cv2.CHAIN_APPROX_SIMPLE فهو يزيل كل نقطة زائدة عن الحاجة، وبالتالي سيتم التوفير في حجم الذاكرة.

الصورة التالية توضح ما تعلمناه، حيث في المستطيل اليساري طبقنا التابع cv2.CHAIN_APPROX_NONE فتم تحديد جميع نقاط المستطيل (باللون الأزرق)، أما في الصورة اليمينية للمستطيل طبقنا التابع cv2.CHAIN_APPROX_SIMPLE فقام بتحديد فقط النقاط الضرورية لرسم المستطيل وهي الأربع النقاط الملونة بالأزرق.



خصائص الإطارات

الهدف:

- تعلم إيجاد الخصائص المختلفة للإطارات مثل المساحة، المحيط، المركز، المربع المحيط.
- سنتعلم استخدام العديد من التوابع التي لها علاقة بالإطارات

١. عزوم الصورة Moments:

العزوم تساعدنا على حساب بعض الخصائص مثل مركز كتلة جسم، مساحة جسم، ..... الخ.
للمزيد من المعلومات عن العزوم راجع صفحة ويكيبيديا Image Moments
العزوم بشكل أساسي هي الخصائص التي تصف شكل الصورة وأمور أخرى كالمساحة والتوجه والسماكة والالتواء و ..... الخ. يمكننا استخدام هذه الخصائص للتعرف على الأشكال التي في الصورة.
عزوم الصورة هي نفس فكرة العزم في الميكانيك. أول عزم سيعطيك مركز الكتلة، حيث الكتلة تمثل شدة(كثافة) البيكسل، وثاني عزم يخبرنا كيف تختلف هذه الكتلة حول مركز الكتلة أي أن الكتلة في المركز غير الكتلة حوله (عند دراسة منطقة من الجسم). بنفس الطريقة التي تحصل فيها على عطالة لجسم على أرض الواقع، يمكنك الحصول إحدى عزوم الصورة. وهذا يعطيك المحأور الرئيسية(x,y) للشكل الذي تريد وصفه.
التابع cv2.moments() يقدم فهرس بكل العزوم المحسوبة، شاهد المثال التالي:

```
import cv2
import numpy as np
img = cv2.imread('star.jpg',0)

ret,thresh = cv2.threshold(img,127,255,0)
contours,hierarchy = cv2.findContours(thresh, 1, 2)

cnt = contours[0]
M = cv2.moments(cnt)

print M
```

من قيم العزوم هذه يمكننا الحصول على بيانات مفيدة مثل المساحة، المركز، ..... الخ

المركز يعطى بالعلاقة: $C_x = \frac{M_{10}}{M_{00}}$ و $C_y = \frac{M_{01}}{M_{00}}$

يمكن تنفيذ هذه العلاقة بصيغة برمجية كما يلي:

```
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])

print cx, cy
```

٣٤٣ ٥٤٤

٢. مساحة الإطار Contour Area:

تحسب مساحة الإطار من خلال التابع cv2.contourArea() أو M['m00'].

```
area = cv2.contourArea(cnt)
print area
```

٦٧٨.5

٣. محيط الإطار Contour Perimeter:

يسمى أيضا طول القطر، يتم حسابه من خلال التابع cv2.arcLength().

متغيره الثاني يحدد فيما إذا كان الإطار مغلق (إذا مررنا له TRUE) أم فقط منحني.

```
perimeter = cv2.arcLength(cnt, True)
print perimeter
```

١٢٨.٦٦٩٠٤٦٤٠٢

٤. تقريب الإطار Contour Approximation:

يقرب شكل الإطار لشكل آخر مع عدد رؤوس أقل بالاعتماد على الدقة التي نحددها، وذلك بتطبيق خوارزمية دوغلاس بوكر Douglas-Peucker .

لفهم ذلك، افترض بأنك تبحث عن مربع في صورة ما، ولكن بسبب مشكلة ما حصلت على مربع مشوه (كما هو معروض في الصورة التالية)، عندها يمكننا استعمال تابع تقريب الإطار للحصول على المربع المطلوب.

المتغير الثاني لتابع تقريب الإطار (epsilon) هو قيمة أعظم خطأ مسموح به للحصول على التقريب المطلوب، ويجب اختياره بعناية للحصول على النتيجة المرغوبة.

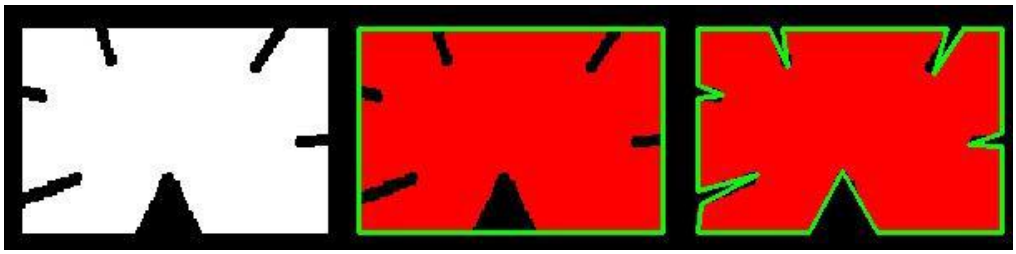
```
epsilon = 0.1*cv2.arcLength(cnt,True)
approx = cv2.approxPolyDP(cnt,epsilon,True)

cv2.drawContours(img, [approx], -1, (0,255,0), 3)
```

في الصورة التي في الوسط، الخط الأخضر يمثل تقريب الإطار.

في الصورة الوسطى نسبة التقريب هي 10% من طول المنحني (epsilon = 10% of arc length).

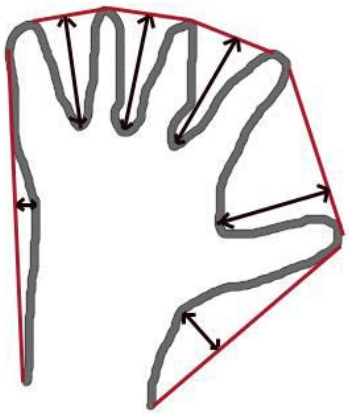
أما في الصورة اليمينية فنسبة التقريب 1% من طول المنحني (epsilon = 1% of the arc length).

المتغير الثالث لتابع تقريب الإطار cv2.approxPolyDP() يحدد فيما إذا كان المحني مغلق أم لا.



المربع المشوه	تقريب الإطار بنسبة ١٠%	تقريب الإطار بنسبة ١%

٥. غلاف التحدب Convex Hull:



غلاف التحدب يبدو لنا كتقريب الإطار، ولكنه ليس كذلك (قد تكون النتائج متشابهة في بعض الحالات). فهو يعني مقاطع المنحنيات البارزة(المنتفخة)(المحدبة)، وأيضا المقاطع المقعرة. المقاطع المقعرة (المنتفخة لجهة الداخل) تدعى بعيوب المنحني.

على سبيل المثال شاهد الصورة التالية، الخط الأحمر يعرض غلاف التحدب لشكل اليد.

الأسهم ذو الاتجاهين التي في الصورة تظهر عيوب التحدب.كل الاختلافات عن شكل المنحني الكلي المحدب يمكن إيجادها وفق التابع:

```
hull = cv2.convexHull(points[, hull[, clockwise[, returnPoints]])
```

شرح التابع cv2.convexHull ():

تابع حساب غلاف التحدب، يأخذ المتغيرات التالية:

- المتغير الأول points[: نمرر لهذا المتغير نقاط الإطار.
- المتغير الثاني hull[: متغير خرج تحدب الغلاف، غالبا نتجنبه.
- المتغير الثالث clockwise[: اتجاه العلم: إذا كان TRUE سيكون خرج تحدب الغلاف مع عقارب الساعة وإلا بالعكس.
- المتغير الرابع returnPoints[: نقاط الاستعادة، افتراضيا قيمتها TRUE، تعيد إحداثيات نقاط تحدب الغلاف، أما إذا كانت قيمتها FALSE تعيد مؤشرات لنقاط تحدب لغلاف.

لذلك للحصول على محدب الغلاف كما في الصورة السابقة يكتفي بكتابة ما يلي:

```
hull = cv2.convexHull(cnt)
```

مثال:

```
import numpy as np
import cv2
img = cv2.imread('hand.jpg')
imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

ret,thresh = cv2.threshold(imgray,245,255,0)
thresh_inv=cv2.bitwise_not(thresh)

contours, hierarchy =
cv2.findContours(thresh_inv,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)

hull = cv2.convexHull(contours[0])
cv2.drawContours(img, hull, -1, (0,•, 255), 3)

cv2.imshow('Image_hull',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

النقاط التي باللون الأحمر هي نقاط غلاف التحدب(hull).
لو وصلنا بين هذه النقاط سنحصل على غلاف التحدب.

ولكن إذا أردنا إيجاد عيوب التحدب نحتاج أن نمرر القيمة False للمتغير الرابع في لتابع () cv2.convexHull (returnPoints = False).

لفهم لذلك سنأخذ صورة المستطيل الذي في الصورة السابقة. في البداية أوجدنا إطار هذا المستطيل، الآن سنوجد غلاف التحدب لهذا المستطيل مع وضع قيمة TRUE في المتغير الرابع لتابع حساب غلاف التحدب (returnPoints = True) فنحصل على النتيجة التالية [[[234 202]], [[51 202]], [[51 79]], [[234 79]]] والتي تمثل النقاط الأربع لزوايا المستطيل .

الآن إذا فعلنا نفس الخطوات السابقة ولكن بدل وضع قيمة TRUE سنضع قيمة FALSE (returnPoints = False) سوف نحصل على النتيجة التالية [[129],[67],[0],[142]] والتي تمثل فهرس (دليل) النقاط المقابلة لها في الإطار ، على سبيل المثال لنفحص نتيجة قيمة ما:
cnt [129] = [[234, 202]] حصلنا على نفس النتيجة السابقة (ونفس الأمر مطبق على النقاط الأخرى)

سنشاهد هذا الأمر مرة أخرى عند التحدث عن عيوب التحدب (convexity defects).

٦. التحقق من التحدب Checking Convexity:

يتم التحقق فيما إذا كان المنحني محدب أم لا، عن طريق التابع cv2.isContourConvex().
هذا التابع يعيد قيمة إما TRUE أو FALSE.

```
k = cv2.isContourConvex(cnt)
```

٧. المستطيل المحيط Bounding Rectangle:

له نوعان هما المستقيم والمدور.

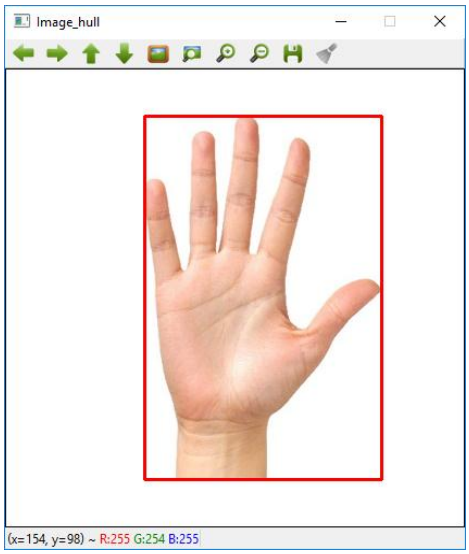
A. المستطيل المحيط المستقيم:

يعطي مستطيل مستقيم بدون أي تدوير، لذلك سيكون حجمه أصغري.
يحسب هذا المستطيل من خلال التابع cv2.boundingRect().
يعطينا هذا التابع إحداثيات الزأوية اليسرى العليا للمستطيل، وعرض المستطيل وطوله.

```
x,y,w,h = cv2.boundingRect(cnt)
cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
```

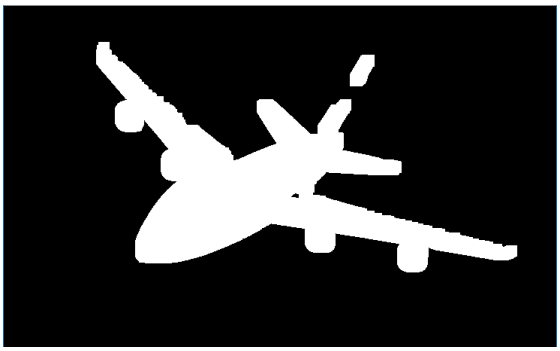
مثال١: في هذا المثال قبل رسم المستطيل المحيط سنجعل في الصورة الثنائية لون اليد باللون الأبيض ولون الخلفية بالأسود لذلك سنقوم بعكس عملية التعتيب (thresh_inv).

```
import numpy as np
import cv2
img = cv2.imread('hand.jpg')
imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

ret,thresh = cv2.threshold(imgray,245,255,0)
thresh_inv=cv2.bitwise_not(thresh)

contours, hierarchy =
cv2.findContours(thresh_inv,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
cnt =contours[0]
hull = cv2.convexHull(cnt)

x,y,w,h = cv2.boundingRect(cnt)
cv2.rectangle(img,(x,y),(x+w,y+h),(0,٠,٢٥٥),2)

#cv2.drawContours(img, contours[0], -1, (0,0, 255), 3)

cv2.imshow('Image_hull',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

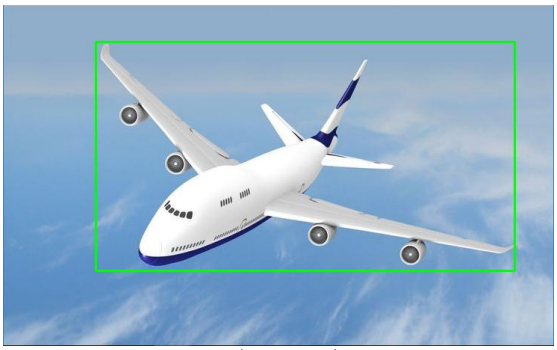


مثال٢: في هذا المثال الصوة الثنائية ستكون غير واضحة لذا سنلجأ لعملية التمديد Dilation.

```
import numpy as np
import cv2
img = cv2.imread('airplan.jpg')
imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

ret,thresh = cv2.threshold(imgray,207,255,0)
thresh_inv=cv2.bitwise_not(thresh)

kernel = np.ones((15,15),np.uint8)
dilation = cv2.dilate(thresh,kernel,iterations = 1)
cv2.imshow('dilation',dilation)

contours, hierarchy =
cv2.findContours(dilation,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
cnt =contours[1]

#draw hull Rectangle
x,y,w,h = cv2.boundingRect(cnt)
cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)

cv2.imshow('Image_hull',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



تعتيب + تمديد



المستطيل المحيط المستقيم

B. المستطيل المحيط المدور:

هنا يرسم المستطيل المحيط بأصغر مساحة، ويأخذ المستطيل زأوية دوران (أي يكون المستطيل مدار بزأوية ما).يحسب هذا المستطيل من خلال التابع cv2.minAreaRect().

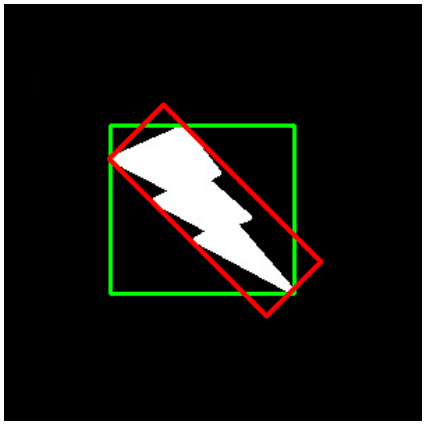
يعطينا هذا التابع إحداثيات الزأوية اليسرى العليا للمستطيل، وطول وعرض المستطيل، بالإضافة لذلك زأوية يعطينا زأوية دوران المستطيل. لرسم المستطيل سنحتاج معرفة زوايا المستطيل الأربع، يمكننا الحصول على هذه الزوايا من خلال التابع cv2.boxPoints().

```
rect = cv2.minAreaRect(cnt)
box = cv2.cv.BoxPoints(rect)

box = np.int0(box)
cv2.drawContours(im,[box],0,(0,0,255),2)
```

ملاحظة:

إذا أردت تطبيق التابع cv2.cv.BoxPoints ضمن مكتبة OpenCvالإصدار ٣ عليك أن تستبدله بالتابع cv2.boxPoints().

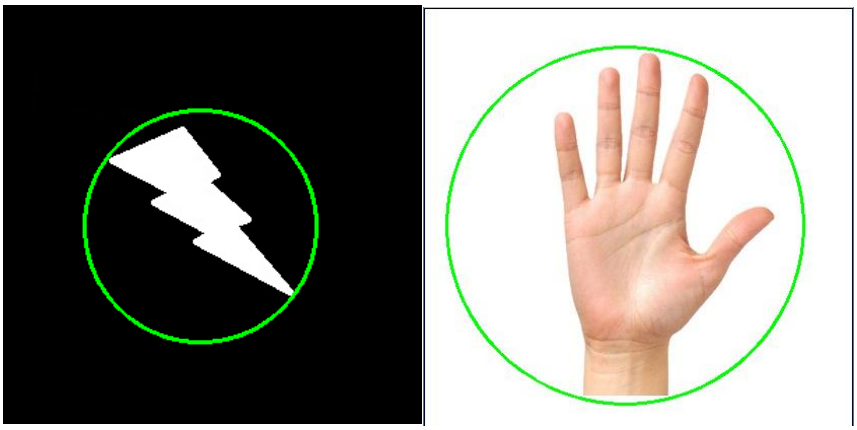
الصورة التالية تعرض نوعي المستطيل المحيط (المستقيم والمدور). المستطيل المحيط المستقيم ملون باللون الأخضر، أما المستطيل المحيط المدور ملون باللون الأحمر.



٨. الدائرة المحيطة الأصغرية Minimum Enclosing Circle:

هي الدائرة التي تغطي الجسم بأقل مساحة، ويتم ذلك عن طريق التابع cv2.minEnclosingCircle().

```
(x,y),radius = cv2.minEnclosingCircle(cnt)
center = (int(x),int(y))
radius = int(radius)
cv2.circle(img,center,radius,(0,255,0),2)
```

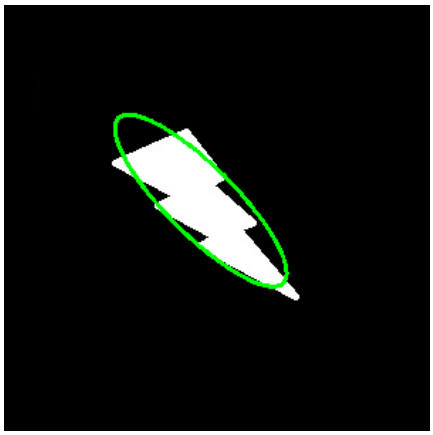


٩. القطع الناقص المتلائم مع الإطار Fitting an Ellipse:

يتم إحاطة الجسم بقطع ناقص.

يتم رسم هذ القطع عن طريق التابع cv2.fitEllipse().

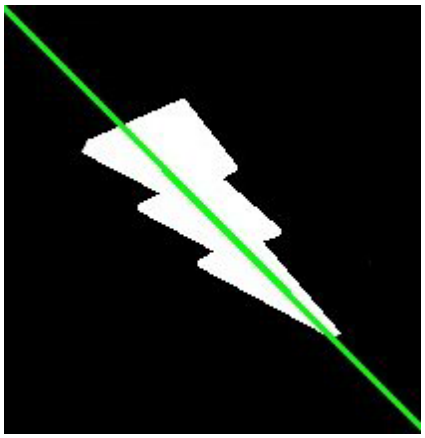
```
ellipse = cv2.fitEllipse(cnt)
im = cv2.ellipse(im,ellipse,(0,255,0),2)
```



١٠. الخط المتلائم مع الإطار Fitting a Line:

يتم رسم هذا الخط عن طريق التابع cv2.fitLine()

```
rows,cols = img.shape[:2]
[vx,vy,x,y] = cv2.fitLine(cnt, cv2.DIST_L2,0,0.01,0.01)
lefty = int((-x*vy/vx) + y)
righty = int(((cols-x)*vy/vx)+y)
img = cv2.line(img,(cols-1,righty),(0,lefty),(0,255,0),2)
```



المزيد من خصائص الإطارات Contour Properties

سنتعلم استخدام خصائص الجسم المستعملة بكثرة مثل الصلابة والقطر المكافئ وقناع الصورة والكثافة المتوسطة.

١- نسبة الأبعاد Aspect Ratio:

تمثل نسبة العرض على الارتفاع للمستطيل المحيط بالجسم.

$$\text{Aspect Ratio} = \frac{Width}{Height}$$

```
x,y,w,h = cv2.boundingRect(cnt)
aspect_ratio = float(w)/h
```

٢- الامتداد (الحدود) Extent:

الامتداد هو نسبة مساحة الإطار على مساحة المستطيل المحيط.

$$Extent = \frac{Object\ Area}{Bounding\ Rectangle\ Area}$$

```
area = cv2.contourArea(cnt)
x,y,w,h = cv2.boundingRect(cnt)
rect_area = w*h
extent = float(area)/rect_area
```

٣- الصلابة (الشدة) Solidity:

هي نسبة مساحة الإطار على مساحة المنطقة المحدبة.

$$\text{Solidity} = \frac{\text{Contour Area}}{\text{Convex Hull Area}}$$

```
area = cv2.contourArea(cnt)
hull = cv2.convexHull(cnt)
hull_area = cv2.contourArea(hull)
solidity = float(area)/hull_area
```

٤- القطر المكافئ Equivalent Diameter:

هو قطر الدائرة التي مساحتها من نفس مساحة الإطار.

$$\text{Equivalent Diameter} = \sqrt{\frac{4 \times \text{Contour Area}}{\pi}}$$

```
area = cv2.contourArea(cnt)
equi_diameter = np.sqrt(4*area/np.pi)
```

٥- التدوير:

هي الزأوية التي تم تدوير الجسم بها. الكود التالي يعطي المحور الاكبر والاصغر والأطوال.

```
(x,y),(MA,ma),angle = cv2.fitEllipse(cnt)
```

٦- القناع ونقاط البيكسل Mask and Pixel Points:

في بعض الأحيان قد نحتاج لاستخدام جميع النقاط التي تشكل هذا الجسم، يمكن معرفة هذه النقاط من خلال ما يلي:

```
mask = np.zeros(imgray.shape,np.uint8)
cv2.drawContours(mask,[cnt],0,255,-1)
pixelpoints = np.transpose(np.nonzero(mask))
#pixelpoints = cv2.findNonZero(mask)
```

هناك طريقتين لمعرفة النقاط التي تشكل الجسم:

- الطريقة الأولى باستخدام توابع مكتبة Numpy.
- الطريقة الثانية باستخدام توابع مكتبة OpenCv (أخر سطر تعليق في الكود السابق).

النتيجة نفسها بالطريقة الأولى مع اختلاف بسيط، فمكتبة Numpyتعطي الإحداثيات بصيغة (سطر، عمود)، بينما مكتبة OpenCvتعطي الإحداثيات بصيغة (X،Y).

٧- المتوسط اللوني أو الشدة المتوسطة Mean Color or Mean Intensity:

نستطيع إيجاد المعدل الوسطي للون. أو يمكننا أيجاد الشدة اللونية الوسطية للجسم في المستوي الرمادي. سنقوم باستعمال القناع مرة أخرى للقيام بذلك.

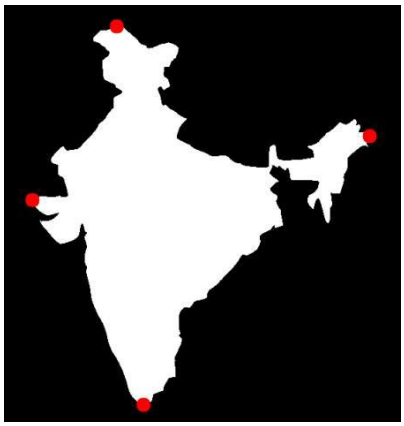
```
mean_val = cv2.mean(im,mask = mask)
```

٨- نقاط الشدة Extreme Points:

نقاط الشدة تعني نقاط الجسم العلوية والسفلية وأقصى اليمين وأقصى اليسار.

```
leftmost = tuple(cnt[cnt[:,:,0].argmin()][0])
rightmost = tuple(cnt[cnt[:,:,0].argmax()][0])
topmost = tuple(cnt[cnt[:,:,1].argmin()][0])
bottommost = tuple(cnt[cnt[:,:,1].argmax()][0])
```

على سبيل المثال، إذا طبقنا نقاط الشدة على خريطة مدينة أنديان سنحصل على النتيجة التالية:



توابع الإطارات الإضافية Contours More Functions

الهدف:

- عيوب التحدب وكيفية إيجادها
- إيجاد أقل مساقة من نقطة إلى المضلع.
- مقارنة الأشكال المختلفة.

النظري + الكود:

١- عيوب التحدب Convexity Defects:

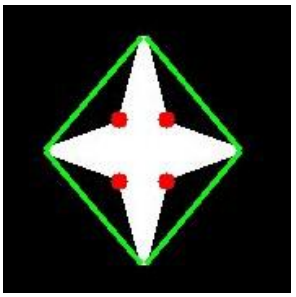
لقد تعلمنا في الفصل السابق الذي يتكلم عن الإطارات ما هو غلاف التحدب. أي انحراف(فرق) لسطح الجسم عن غلاف المحدب يعتبر عيب تحدب. مكتبة OpenCvوفرت لنا تابع جاهز لإيجاد هذه العيوب cv2.convexityDefects().

```
hull = cv2.convexHull(cnt,returnPoints = False)
defects = cv2.convexityDefects(cnt,hull)
```

```
[[[ 267   370   313 22563]]

 [[ 370   372   371   142]]

 [[ 373   377   374   114]]

 [[ 377   479   450 30081]]

 [[ 264   266   265   114]]]
```

ملاحظة: تذكر بأننا مرننا للمتغير **returnPoints** في التابع **cv2.convexHull** القيمة **FALSE** لكي نوجد عيوب التحدب (**returnPoints = False**).

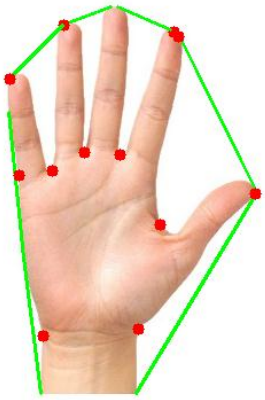
يعيد التابع convexityDefects () مصفوفة بحيث كل صف يحوي القيم التالية[نقطة البداية، نقطة النهاية، أبعد نقطة، المسافة التقريبية لأبعد نقطة]. القيم الثلاث الأولى تعاد بشكل مؤشر لموضع النقطة على الإطار.



سنكتب برنامج توضيحي يقوم برسم مستقيم من نقطة البداية إلى نقطة النهاية، ويرسم دائرة حول أبعد نقطة.

تذكر القيم الثلاث الأولى أعيدت كمؤشر لرقم الإطار (قيم عبوب التحدب لم تعاد بشكل إحداثيات). لذلك يجب أن نحضر هذه القيم دائما من المصفوفة التي تحتوي على إحداثيات الإطار المحدد (رقم الإطار).

```
import numpy as np
import cv2
img = cv2.imread('hand.jpg')
imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

ret,thresh = cv2.threshold(imgray,230,255,0)
thresh_inv=cv2.bitwise_not(thresh)

contours, hierarchy = cv2.findContours(thresh_inv,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
cnt =contours[0]

hull = cv2.convexHull(cnt,returnPoints = False)
defects = cv2.convexityDefects(cnt,hull)
#print defects
#print defects.shape
for i in range(defects.shape[0]):
    s,e,f,d = defects[i,0]
    start = tuple(cnt[s][0])
    end = tuple(cnt[e][0])
    far = tuple(cnt[f][0])
    cv2.line(img,start,end,[0,255,0],2)
    cv2.circle(img,far,5,[0,0,255],-1)

#cv2.drawContours(img, contours[0], -1, (0,0, 255), 3)

cv2.imshow('Image_hull',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



النتيجة:

شرح الكود:

في هذا الكود لفهم وظيفة كل سطر بشكل جيد قم باستخدام التعليمة print ليسهل عليك فهم الكود.

```
ret,thresh = cv2.threshold(imgray,230,255,0)
thresh_inv=cv2.bitwise_not(thresh)
```

قمنا بعملية التعتيب فظهرت اليد بلون اسود والخلفية بلون ابيض، لذلك قمنا بعكس نتيجة التعتيب عن طريق التابع cv2.bitwise_not() حتى تظهر اليد باللون الأبيض.نقوم بإيجاد الإطارات عن طريق التابع cv2.findContours().

```
contours, hierarchy = cv2.findContours(thresh_inv,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
cnt =contours[0]
```

ثم نخزن الإطار الأول (هنا يمثل الإطار حول اليد) في متغير سميناه cnt.

```
hull = cv2.convexHull(cnt,returnPoints = False)
```

نوجد النقاط التي تكون غلاف التحدب للإطار عن طريق التابع cv2.convexHull ونمرر للبارمتر الثاني لهذا التابع القيمة False حتى نستطيع أيجاد عيوب التحدب.

```
defects = cv2.convexityDefects(cnt,hull)
```

نوجد النقاط التي تشكل عيوب تحدب الإطار عن طريق التابع cv2.convexityDefects() الذي سيعيد لنا أربع قيم، (مثلا [٢٢٥٦٣ ٣١٣ ٣٧٠ ٢٦٧]) أول ثلاث قيم منها تؤشر إلى موضع نقاط عيوب التحدب.

```
for i in range(defects.shape[0]):
```

نقوم بتكرار الحلقة حسب عدد نقاط عيوب التحدب (في مثالنا يوجد ١٢ نقطة لذلك تم تكرير حلقة for ١٢ مرة)

باقي الكود لن اشرحه حأول فهمه لوحدك عن طريق تعليمة print.

٢- نقطة مضلع الاختبار Point Polygon Test:

التابع cv2.pointPolygonTest() يوجد أقصر مسافة بين نقطة في الصورة والإطار.
فهو يفيدنا في تحديد موقع النقطة بالنسبة للإطار، هل هي داخل محيط الإطار أم خارجه أم على الإطار. يعيد هذا التابع قيمة سالبة إذا كانت النقطة خارج محيط الإطار.

على سبيل المثال يمكنا التحقق من النقطة (50,50) كما يلي:

```
dist = cv2.pointPolygonTest(cnt,(50,50),True)
```

ثالث متغير في التابع cv2.pointPolygonTest() هو measureDist، إذا كانت قيمته TRUE سيوجد بعد النقطة، وإذا كانت قيمته False سيوجد هل النقط داخل الإطار أم داخله أم على الإطار.
إذا أعاد التابع القيمة (1.0) فالنقطة داخل الإطار، وإذا أعاد القيمة (-1.0) فالنقطة خارج الإطار، وإذا أعاد القيمة (0.0) فالنقطة على الإطار.

ملاحظة: إذا لم ترغب بإيجاد المسافة تأكد بأن تجعل قيمة هذا المتغير **False**، (لأن عملية حساب المسافة تأخذ وقت طويل) مما يجعل زمن تنفيذ التابع أسرع بضعفين أو ثلاثة.

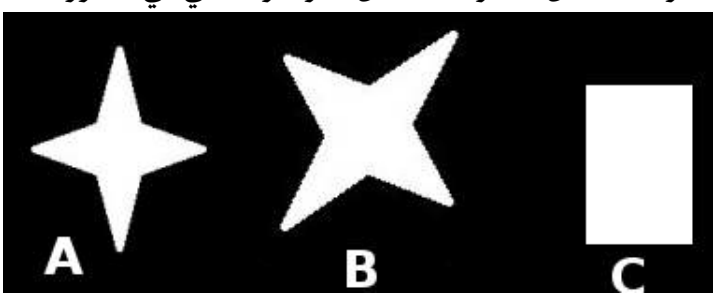
٣- مقارنة الأشكال Match Shapes:

مكتبة OpenCv توفر لنا التابع cv2.matchShapes() الذي يمكننا من المقارنة بين شكلين أو إطارين، ويعيد قيمة لنتيجة المقارنة توضح هذا التشابه، وكلما كانت نتيجة المقارنة أقل كلما كان التشابه أكبر.

تحسب هذه القيمة اعتمادا على قيم العزوم، ويوجد أيضا هناك طرق مختلفة لإيجاد قيمة نتيجة المقارنة. الكود التالي يوضح هذه الطرق:

```
import cv2
import numpy as np

img1 = cv2.imread('star1.jpg',0)
img2 = cv2.imread('star2.jpg',0)

ret, thresh = cv2.threshold(img1, 127, 255,0)
ret, thresh2 = cv2.threshold(img2, 127, 255,0)
contours,hierarchy = cv2.findContours(thresh,2,1)
cnt1 = contours[0]
contours,hierarchy = cv2.findContours(thresh2,2,1)
cnt2 = contours[0]

ret = cv2.matchShapes(cnt1,cnt2,1,0.0)
print ret
```

قمت بتطبيق تابع مقارنة الأشكال لمقارنة الأشكال A وB وC التي في الصورة التالية:



فكانت النتيجة كالتالي:

- مقارنة الشكل A مع نفسه كانت نتيجة المقارنة = 0.0
- مقارنة الشكل A مع الشكل B فكانت نتيجة المقارنة = 0.001946
- مقارنة الشكل Aمع الشكل Cفكانت نتيجة المقارنة = 0.326911

تدوير الصورة لن يؤثر كثيرا على نتيجة المقارنة.

شاهد أيضاَ:

Hu-Moments هي سبعة عزوم لا تتأثر بالدوران أو تغير الحجم. يمكنك إيجاد هذه العزوم من خلال التابع cv2.HuMoments().

```
a = cv2.HuMoments(contours)
print a ,'\n', len(a)
```

تمرين:

قارن بين صورتي حرفين أو رقمين باستخدام التابع cv2.matchShapes() (هذا المثال يمثل أبسط خطوة نحو الـ OCR).

الـ OCR باختصار عبارة عن خوارزمية لتحويل الأرقام أو الأحرف المرسومة إلى نص).

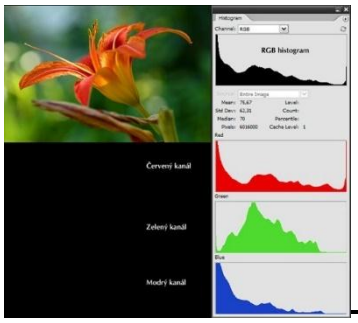
الفصل الخامس

” الجميع عباقرة لكن إن حكمت على قدرة سمكة في تسلق الشجرة
ستعيش حياتها كلها وهي تؤمن بأنها غبية “

ألبرت أينشتين

الفصل الخامس: المخططات البيانية في OpenCv

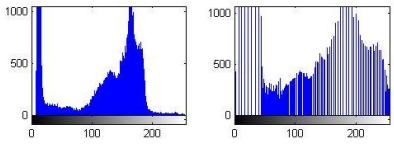
المخططات البيانية -١: البحث، الرسم، التحليل!!!



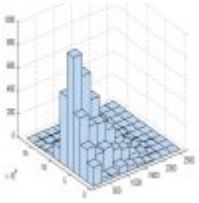
- تعلم البحث عن الإطار ورسمه

المخططات البيانية -٢: معادلة الرسم البياني.

- تعلم تحقيق التكافؤ للمخططات البيانية للحصول على أفضل نسبة تباين للصو



المخططات البيانية ثنائية البعد 2D Histograms:



- تعلم وإيجاد المخطط البياني ثنائي البعد.

الإسقاط الخلفي للمخطط البياني:

- تعلم إيجاد المخطط البياني للإسقاط الخلفي لاكتشاف جسم ملون بلون معين في الصورة.

المخططات البيانية في OpenCv

الإيجاد، الرسم، التحليل في المخططات البيانية:

الهدف:

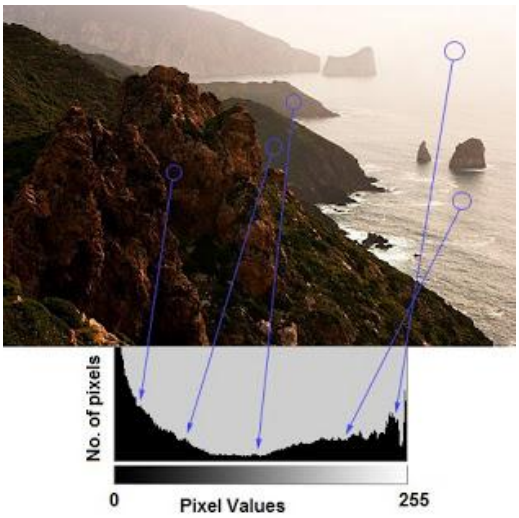
- إيجاد المخططات البيانية باستخدام توابع كل من OpenCvوnumpy.
- رسم المخططات البيانية باستعمال توابع OpenCvوMatplotlib
- سنتعلم استخدام التوابع: cv2.calcHist()، np. histogram ()

النظري:

المخطط البياني يعطينا فكرة عامة عن الصورة، يتمثل بقيم البيكسلات على المحور x وعددها الموافق على المحور Y.

هو فقط عبارة عن طريقة أخرى لفهم الصورة. بالنظر للمخطط البياني للصورة فإننا نحصل على فكرة عامة عن توزع الشدة اللونية في الصورة، وأغلب تطبيقات معالجة الصورة اليوم لا تخلو من استعمال المخططات البيانية.

الصورة التالية مأخوذة من موقع جامعة كامبريدج Cambridge in Color website وتتضمن الصورة مع مخططها البياني. مع العلم أن المخطط البياني مرسوم للصورة الرمادية وليس الملونة.



المنطقة اليسرى من المخطط البياني تعرض كمية البيكسلات التي لونها غامق (عميق)، والمنطقة اليمنى تظهر كمية البيكسلات الأفتح. وبذلك نلاحظ في الصورة بأن البيكسلات الأفتح أقل من البيكسلات الأغمق، أما بالنسبة للبيكسلات التي في الوسط فكميتها قليلة.

- **إيجاد المخطط البياني Find Histogram:**

بعد أن أخذنا فكرة عن المخططات البيانية، يمكننا الآن أن نعرف طريقة إيجادها عبر كل من مكتبة OpenCvوNumpy، ولكن قبل البدء نحتاج لفهم بعض المصطلحات المرتبطة بالمخطط البياني:

BINS: المخطط البياني السابق يظهر عدد البيكسلات لكل قيمة ويظهر أيضا شدتها اللونية، وبالتالي سنحتاج لـ 265 قيمة للإظهار، ولكن ماذا نفعل إذا أردنا أن نحسب المخطط البياني لمجموعة بيكسلات تقع قيمتها في مجال جزئي محدد؟ على سبيل المثال، نريد إيجاد عدد البيكسلات التي شدتها اللونية بين 0-15 ثم من 16-31 ثم ،.....، ثم من 240إلى 255. سنحتاج فقط لـ 16 قيمة لتمثيل المخطط البياني وهذا ما سنشاهده في بعض الأمثلة في OpenCV Tutorials on histograms .

لذلك ما يجب علينا فعله هو جمع القيم التي في المخطط البياني الكلي الْمشَكِلة لقيمة واحدة في المخطط البياني المصغر، أي قسم المخطط الكلي لـ 16 قسم متسأوي، وعندها يدعى كل قسم بـ BIN. في المخطط البياني الرئيسي للصورة السابقة كان عدد القيم 256 قيمة، ويرمز لقيم الـ BINS السابقة بـ histSize.

DIMS: عدد البارامترات (المعايير) التي تجمع البيانات. في المخطط السابق أخذنا فقط شدة البيكسل بعين الاعتبار لذلك ستكون قيمتها هنا 1.

RANGE: هو مجال الشدات اللونية التي نريد قياسها، بالعادة تكون من 0إلى 256 أي المجال كامل (جميع قيم الشدات (العمق اللوني)).

- **حساب المخطط البياني في OpenCv:**

سنستخدم الآن التابع cv2.calcHist() لإيجاد المخطط البياني، سنتعرف على هذا التابع وعلى متغيراته:

```
cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])
```

المتغير الأول image: يمثل الصورة الأصلية من نوع unit8 أو float32، ويجب أن تكتب ضمن أقواس، مثل "[img]".

المتغير الثاني channels: دليل القناة التي نريد أن نأخذ فيها المخطط البياني لمستوي الصورة، في حال كانت الصورة رمادية تكون القناة واحدة، أما في حالة الصورة الملونة فيمكننا تحديد القناة اللونية (R أو G أو B) عن طريق تمرير القيمة التي نريدها 0 أو 1 أو 2 (كل رقم يمثل قناة).

المتغير الثالث mask: قناع الصورة، إذا أردت إيجاد المخطط البياني لكامل الصورة تكون قيمته None، أما إذا أردت إيجاد المخطط البياني لمنطقة محددة فعليك إنشاء قناع صورة مناسب وتمريرها لهذا المتغير.

المتغير الرابع histSize: يمثل عدد الـBINS ، ويجب كتابته ضمن أقواس []، للحصول على المجال الكامل نمرر [256].

المتغير الخامس range: هذا المتغير يمثل المجال وبشكل افتراضي يكون المجال من [0,256].

سنبدأ بالمثال التالي بإيجاد المخطط البياني لصورة رمادية.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('sonic.jpg',0)

hist = cv2.calcHist([img],[0],None,[256],[0,256])
print hist.shape
```

```
(256, 1)
```

المخطط البياني هنا هو مصفوفةبحجم 256x1 ، كل قيمة توافق عدد من البيكسلات في الصورة مع قيمة البيكسل المقابلة لها.

- **حساب المخطط البياني في Numpy:**

سنستخدم التابع np. histogram () الموجود ضمن مكتبة Numpy في إيجاد المخطط البياني.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('home.jpg',0)
hist,bins = np.histogram(img.ravel(),256,[0,256])

print hist.shape
```

```
(256, 1)
```

شرح:

np.histogram()

ravel()

سنحصل على نتيجة مثل نتيجة التابع السابق (cv2.calcHist).

تملك مكتبة Numpy التابع np.bincount() وهو أسرع بـ 10X مرة من np.histogram()، لذلك في المخطط البياني الأحادي البعد يفضل استخدام التابع np.bincount()، ولكن لا تنسى أن تضع minlength= 256 في هذا التابع.

مثلاً:

```
hist = np.bincount(img.ravel(),minlength=256)
```

ملاحظة: توابع **OpenCv** أسرع بحوالي **40X** مرة من **np.histogram** ، لذلك استعمل توابع مكتبة **OpenCv** عند رسم المخطط البياني للصورة.

- **رسم المخطط البياني Plotting Histograms:**

هناك طريقتين للرسم:

- طريقة قصيرة: باستخدام توابع الرسم في Matplotlib.
- طريقة طويلة: باستخدام توابع الرسم في OpenCv.

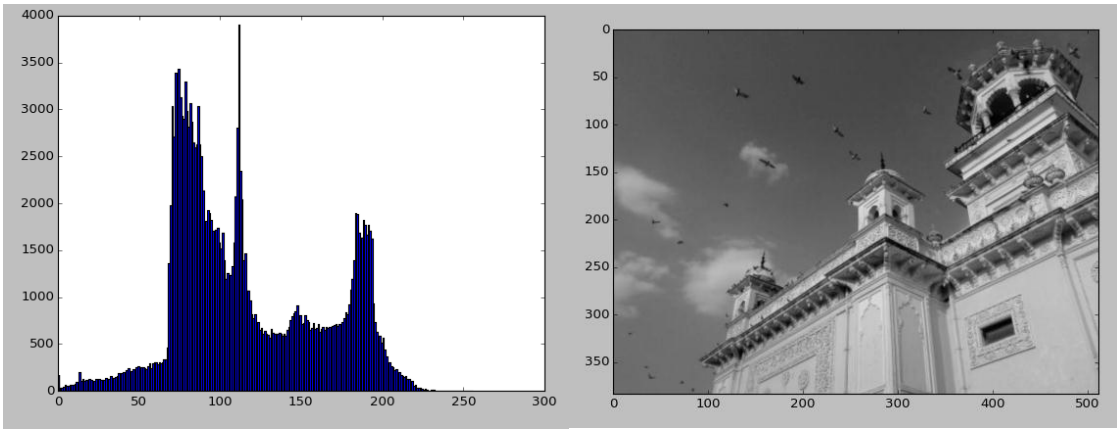
١ – الرسم باستخدام Matplotlib:

توفر لنا مكتبة Matplotlibتابع جاهز لرسم المخططات البيانية:
matplotlib.pyplot.hist(). هذا التابع يوجد مباشرة المخطط البياني ويرسمه، ولا نحتاج لاستخدام cv2.calcHist ولا np.histogram لرسم المخطط البياني.

شاهد الكود التالي:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('home.jpg',0)

plt.hist(img.ravel(),256,[0,256])
#plt.imshow(img, cmap = 'gray')
plt.show()
```

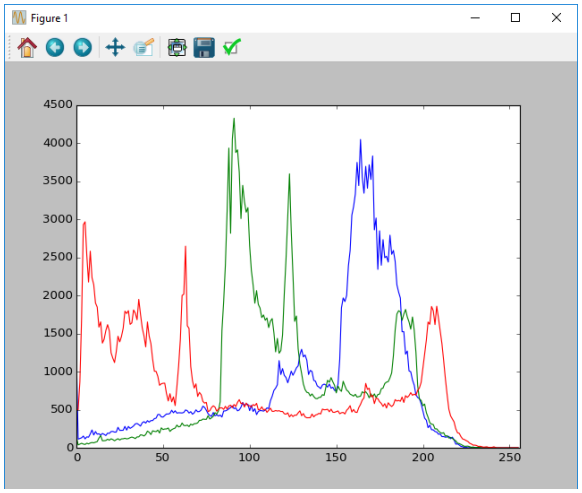
النتيجة:



يمكنك أيضا استخدام الرسم العادي في Matplotlibلرسم المخططات البيانية للصور الملونة .
ستحتاج في البداية إيجاد بيانات المخطط البياني، كما في الكود التالي:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('home.jpg')
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```

النتيجة:



بإمكانك التحقق من الصورة السابقة بحيث ستجد اللون الأزرق نسبته عالية بسبب السماء

٢- الرسم باستخدام OpenCv:

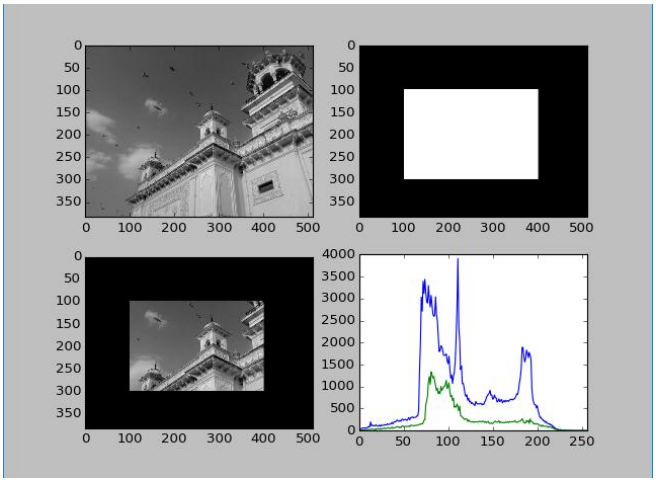
تطبيق القناع Application of Mask:

لقد استخدمنا فيما سبق التابع cv2.calcHist() لإيجاد المخطط البياني لكامل الصورة، أما إذا أردنا إيجاد المخطط البياني لمنطقة محددة فعليك إنشاء صورة بيضاء (القناع) في مكان المنطقة التي نريد رسم المخطط البياني لها، والمناطق الباقية من الصورة نجعلها سوداء. ثم نمرر هذا القناع. (شاهد الصورة التالية)

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('home.jpg',0)
# create a mask
mask = np.zeros(img.shape[:2], np.uint8)
mask[100:300, 100:400] = 255
masked_img = cv2.bitwise_and(img,img,mask = mask)
# Calculate histogram with mask and without mask
# Check third argument for mask
hist_full = cv2.calcHist([img],[0],None,[256],[0,256])
hist_mask = cv2.calcHist([img],[0],mask,[256],[0,256])

plt.subplot(221), plt.imshow(img, 'gray')
plt.subplot(222), plt.imshow(mask,'gray')
plt.subplot(223), plt.imshow(masked_img, 'gray')
plt.subplot(224), plt.plot(hist_full), plt.plot(hist_mask)
plt.xlim([0,256])

plt.show()
```

النتيجة:

المخطط البياني الأزرق يمثل المخطط البياني لكل الصورة، والمخطط البياني الأخضر يمثل المخطط البياني لمنطقة القناع.



مراجع إضافية:

Cambridge in Color website

تسوية المخطط البياني Histogram Equalization

الهدف:

في هذا القسم:

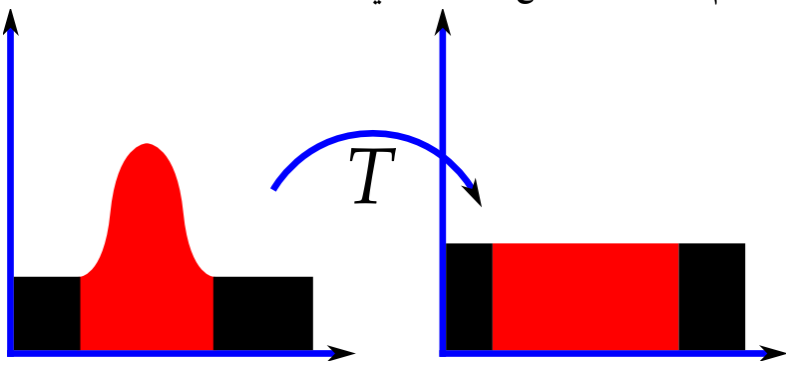
- سنتعلم ما هو مفهوم تسوية المخطط البياني، وكيف نستخدمه لتحسين تباين الصورة.

تعريف:

تسوية المخطط البياني (تسوية الهيستوغرام): هي طريقة معالجة الصورة لضبط التباين باستخدام المخطط البياني للصورة.

نظرة عامة:

فليكن لدينا صورة قيم بيكسلاتها متجمعة في نقطة واحدة في الصورة، مثلا الصورة اللامعة قيم بيكسلاتها متجمعة بالمناطق عالية القمم، ولكن الصورة الجيدة تملك توزعا منتظم للقيم، ولذلك نحتاج لنشر تلك القيم على مجال أوسع، وهذا يؤدي لتحسين تباين الصورة.



للمزيد من التفاصيل يمكنك زيارة صفحة ويكيبيديا Histogram Equalization أما هنا فسوف نشاهد تطبيق تسوية المخطط البياني عن طريق مكتبتي Numpy وOpenCv.

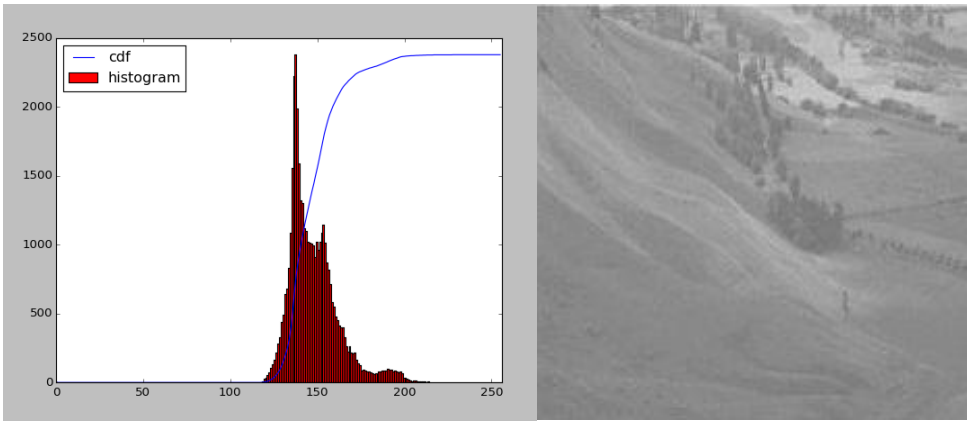
في البداية سنطبق تسوية المخطط البياني بالاستعانة بمكتبة Numpy كما في الكود التالي:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('wiki.jpg',0)
hist,bins = np.histogram(img.flatten(),256,[0,256])

cdf = hist.cumsum()
cdf_normalized = cdf * hist.max()/ cdf.max()

plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.show()
```

شرح :

flatten()

cumsum()



عند مشاهدة المخطط البياني ستجد أن المخطط يتموضع في المنطقة الأكثر إشراقاً، ونحن بحاجة لكامل الطيف، لذلك نحن بحاجة لتابع يقوم بتوزيع البيكسلات على كامل المنطقة بدل رسم البيكسلات في المنطقة الأكثر سطوعا.

والآن علينا البحث عن أخفض نقطة في المخطط البياني (ما عدا 0)، ونطبق علاقات تسوية المخطط البياني كما هو مشروح في صفحة ويكيبيديا، ويمكننا أيضا استخدام مفهوم مصفوفة القناع في Numpy، حيث تجري العمليات فقط على العناصر غير المقنعة وذلك كما يلي:

```
cdf_m = np.ma.masked_equal(cdf,0)
cdf_m = (cdf_m - cdf_m.min())*255.0/(cdf_m.max()-cdf_m.min())
cdf = np.ma.filled(cdf_m,0).astype('uint8')
```

شرح :

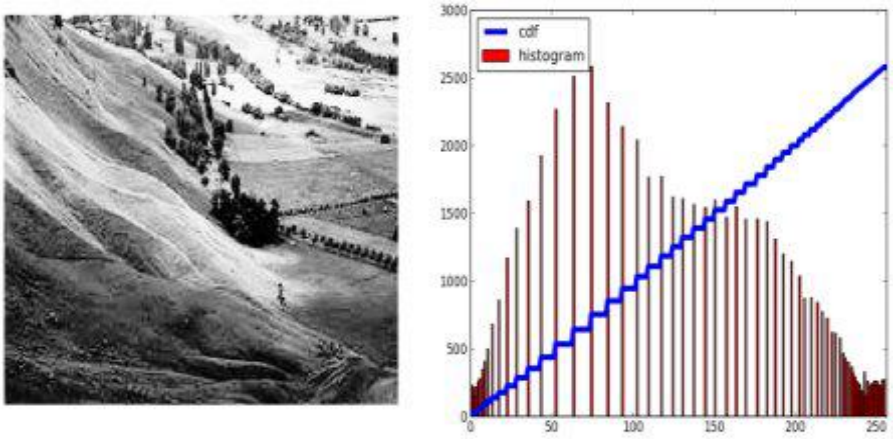
np.ma.masked_equal()

np.ma.filled()

البحث الذي يعطينا معلومات عن قيمة كل بيكسل في صورة الخرج بالنسبة لكل قيمة بيكسل في صورة الدخل. سنطبق عملية التحويل وذلك كما يلي:

```
img2 = cdf[img]
```

الآن بإمكاننا حساب المخطط البياني والـ cdf كما سبق، والنتيجة ستبدو كما يلي.



هناك ميزة مهمة وهي بأن الصورة حتى ولو كانت أغمق، فسنحصل بالنهاية على نفس النتيجة، أي يمكننا استخدامه كشروط إضاءة مرجعية، وهو مفيد في عدة حالات، مثلا في التعرف على الوجوه وقبل أن نأخذ عينات من صورة الوجه للتعرف على الوجه نقوم بتسوية المخطط البياني للصورة للحصول على صورة واضحة للتعرف على الوجه في ظروف الإضاءة المختلفة.

- **المخطط البياني المتوازن في OpenCV:**

مكتبة OpenCV توفر لنا التابع cv2.equalizeHist() الذي دخله هو صورة رمادية ويعطينا في الخرج نفس الصورة ولكن بتباين لوني أوضح.

الكود التالي يوضح ذلك:

```
img = cv2.imread('wiki.jpg',0)
equ = cv2.equalizeHist(img)
res = np.hstack((img,equ))  #stacking images side-by-side
cv2.imwrite('res.png',res)
```

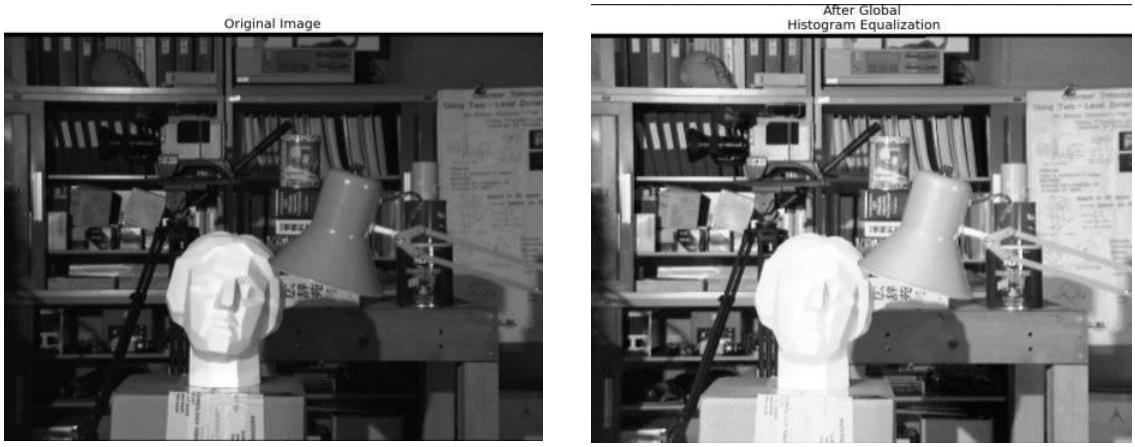
يمكنك الآن أخذ عدة صور بشروط إضاءة مختلفة وتطبق عليهم تابع تسوية المخطط البياني cv2.equalizeHist() لتحسين الصورة، وتحقق من النتيجة.

طريقة تسوية المخطط البياني تعمل بشكل جيد عندما يكون توزع الشدة اللونية لبيكسلات الصورة محصور في منطقة معينة، ولن تعمل بشكل جيد في الأماكن التي يوجد فيها اختلاف كبير في الشدة اللونية.

- **CLAHE (تسوية المخطط البياني ذو التباين المحدود والمتكيف):**

CLAHE (Contrast Limited Adaptive Histogram Equalization)

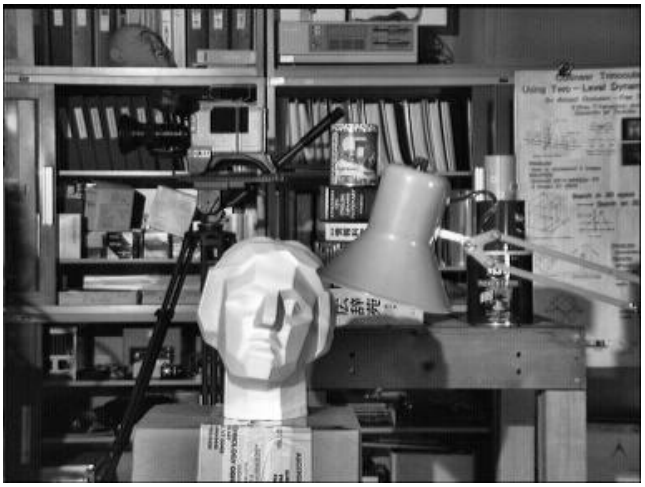
التسوية السابقة للمخطط البياني تأخذ بعين الاعتبار التباين الكلي للصورة، ولكن أحيانا قد لا تعمل معنا، فالصورة التالية تظهر صورة الدخل وخرجها بعد التسوية الشاملة للمخطط البياني: الخلفية قد تكون تحسنت في الصورة السابقة، ولكن ملامح الوجه تبدو ممحية، وهذا لأن تسوية المخطط البياني ليست مركزة على منطقة واحدة كما سبق.



لحل هذه المشكلة نستخدم تسوية المخطط البياني المتكيفة، وذلك بتقسيم الصورة لمكعبات (افتراضيا قياس المكعب 8X8 في OpenCV) ثم كل مكعب في الصورة يتم تسوية المخطط البياني له على حدى، حيث نلاحظ بأنه يأخذ بعين الاعتبار كل جزء من الصورة لوحده.

في حالة وجود ضجيج، نطبق حد للتباين، فإذا كان المخطط أعلى من حد التباين المحدد (افتراضيا 40 في OpenCV) يتم قص هذه البيكسلات وتوزيعها بشكل منتظم على المكعبات الأخرى قبل تطبيق عملية تسوية المخطط.وأخيرا يتم إجراء عملية الاستيفاء الثنائي الخطي على الحدود لتنعيم الانتقال. الكود التالي يعرض طريقة تطبيق CLAHE في OpenCV:

```
import numpy as np
import cv2
img = cv2.imread('tsukuba_l.png',0)

# create a CLAHE object (Arguments are optional).
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
cl1 = clahe.apply(img)
cv2.imwrite('clahe_2.jpg',cl1)
```



وكما نرى في النتيجة بأن تفاصيل الوجه قد تحسنت.

مراجع إضافية:

- **Wikipedia page on** Histogram Equalization
- Masked Arrays in Numpy
- How can I adjust contrast in OpenCV in C?
- How do I equalize contrast & brightness of images using OpenCV?

المخطط البياني ثنائي البعد (2D Histograms)

الهدف:

في هذه الوحدة سنتعلم إيجاد ورسم المخطط ثنائي البعد. سيفيدنا في الفصول القادمة.

مقدمة:

في السابق أوجدنا ورسمنا المخطط البياني أحادي البعد (يسمى أحادي البعد لأنه يأخذ فقط خاصية واحدة مثل قيمة شدة التدرج الرمادي للبيكسل)، أما حاليا فسنبحث عن خاصيتين لرسمها بشكل ثلاثي الأبعاد. تستعمل عادة لإيجاد المخططات البيانية للألوان حيث تكونان قيمة التدرج اللوني والتشبع لكل بيكسل.

- **المخطط البياني 2D في OpenCv:**

يحسب ببساطة جداً من خلال التابع التابع cv2.calcHist(). سنحتاج لتحويل الصورة من الفضاء BGR إلى HSV (تذكر في المخطط البياني 1D كنا نحول من الفضاء BGR إلى المستوي الرماديGRY). وهنا بارامترات التابع cv2.calcHist() ستكون كالتالي:

- channels = [0,1]: لأننا بحاجة لقناتين H و S.
- bins = [180,256]: 180للمستوي H و 256 للمستوي S.
- range = [0,180,0,256]: قيمة التدرج اللوني Hueتكون بين 0و 180، بينما قيمة الإشباع تكون بين الـ 0و 256.

شاهد الكود التالي:

```
import cv2
import numpy as np
img = cv2.imread('home.jpg')

hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
hist = cv2.calcHist([hsv], [0, 1], None, [180, 256], [0, 180, 0, 256])
```

- **المخطط البياني 2D في Numpy:**

لدى مكتبة Numpy أيضا تابع للحساب وهو np.histogram2d() (تذكر بأنه في المخطط البياني 1D استعملنا التابع np.histogram()).

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('home.jpg')

hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
hist, xbins, ybins = np.histogram2d(h.ravel(),s.ravel(),[180,256],[[0,180],[0,256]])
```

أول متغير لهذا التابع هو المستوي H، والمتغير الثاني هو المستوي S، والثالث هو عدد الـ BIN، والرابع مجالها. والآن سننتقل للخطوة التالية وهي طريقة رسم المخططات البيانية 2D.

- **رسم المخططات البيانية ثنائية البعد (Plotting 2D Histograms):**

الطريقة الأولى باستخدام cv2.imshow():

بما أن النتيجة هي مصفوفة ثنائية البعد فلذلك يمكن عرضها كصورة رمادية باستخدام التابع السابق، ولكن لن نحصل على فكرة كبيرة عن الألوان إلا إذا كنا على علم بقيم التدرج (Hue) للألوان المختلفة.

الطريقة الثانية باستخدام مكتبة Matplotlib:

يمكننا استخدام التابع plt.imshow() لرسم المخطط البياني ثنائي البعد بأشكال مختلفة، مما سيعطينا فكرة أفضل عن شدة البيكسلات المختلفة، ولكن لن يعطينا فكرة كافية عن الألوان من أول نظرة إلا إذا كنا نعرف قيمة التدرج للألوان المختلفة. ومع ذلك يفضل استعمال هذه الطريقة فهي أفضل وأبسط.

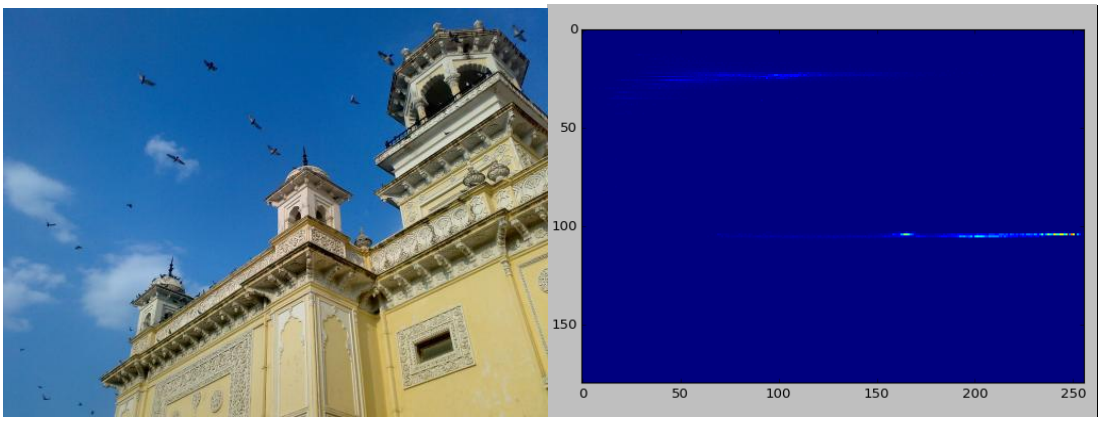
ملاحظة: عند استخدام هذا التابع تذكر وضع المتغير **nearest** في العلم **interpolation** للحصول على نتائج أفضل (**interpolation = 'nearest'**).

شاهد الكود التالي:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('home.jpg')

hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
hist = cv2.calcHist( [hsv], [0, 1], None, [180, 256], [0, 180, 0, 256] )

plt.imshow(hist,interpolation = 'nearest')
plt.show()
```

النتيجة:

الصورة التالية هي للصورة الأصلية مع مخططها البياني الملون. المحور X يعرض قيمة S (الإشباع)، والمحور Y يعرض قيمة التدرج (Hue).



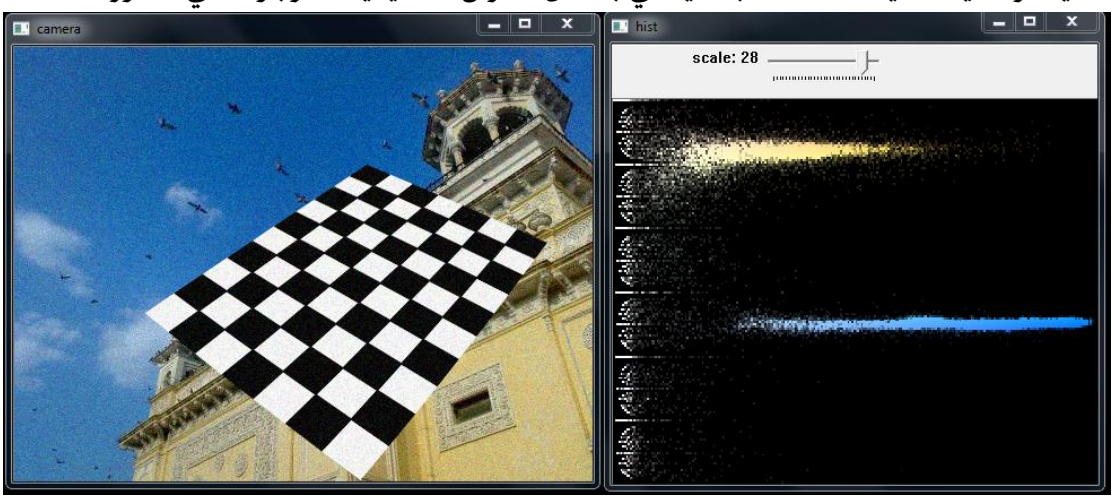
في المخطط البياني نشاهد بعض القيم العالية تقريبا عند S=200, H=100 ، وهذا يقابل لون السماء الأزرق. وأيضا يوجد هناك بعض القيم قيمتها عالية تقريبا عند S=100 , H=25 ، وهذا يقابل اللون الأصفر للقصر.

الطريقة الثالثة باستخدام أسلوب العينات في OpenCv:

وهو كما في المثال البسيط المرفق في الموقع الذي نصبت فيه مكتبة OpenCv ضمن المسار.
"C:\opencv\sources\samples\python2\color_histogram.py"

إذا شغلت هذا الكود يمكنك مشاهدة المخطط البياني الذي يعرض الألوان المتوافقة، وبكلمات بسيطة فهو يخرج الكود اللوني للمخطط البياني. النتيجة جيدة جدا (على الرغم من أنك تحتاج إلى إضافة مجموعة إضافية من الخطوط)

في هذا الكود المبرمج أنشأ خريطة ملونة في الفضاء HSV، ثم حولها للفضاء BGR. نتيجة صورة المخطط البياني تكون مضروبة بلون الخريطة. واستخدم أيضا بعض خطوات المعالجة لإزالة البيكسلات الصغيرة المعزولة، للحصول على مخطط بياني جيد. يمكنك العودة لهذا البرنامج وتشغيله وتحليله، حيث ستلاحظ بأنه يعطي بالفعل الألوان الحقيقية الموجودة في الصورة.

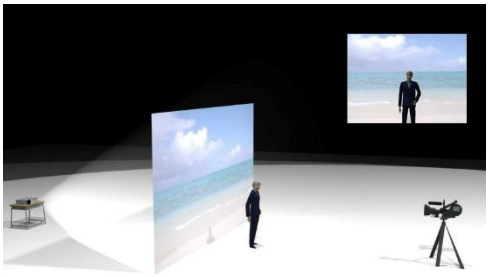


يمكنك أن ترى في المخطط البياني بوضوح ما هي الألوان الموجودة، فهناك اللون الأزرق واللون الأصفر وبعض البياض بسبب رقعة الشطرنج.

الإسقاط الخلفي للمخطط البياني Histogram Back projection

الهدف:

في هذا الفصل سنتعلم طريقة إسقاط المخطط البياني على صورة خلفية وفائدتها.



معلومات نظرية:

تعريفback-projection في علم التصوير:

هو إسقاط الصورة على خلفية شاشة نصف شفافة ليتم عرضها كما في فيلم التصوير، أو لإستخدامها كخلفية في التصوير.

سنتستعمل هذه التقنية لتقطيع الصورة أو لإيجاد الأجسام المهمة في الصورة.

بكلمات بسيطة، تعتمد هذه الخوارزمية على إنشاء صورة بنفس قياس صورة الدخل (ولكن بقناة واحدة)، بحيث كل بيكسل فيها يتوافق مع احتمال أن ينتمي للجسم المطلوب. بكلمات أكثر بساطة، صورة الخرج سيكون فيها الجسم المطلوب أكثر بياضاً بالنسبة لباقي أجزاء الصورة.

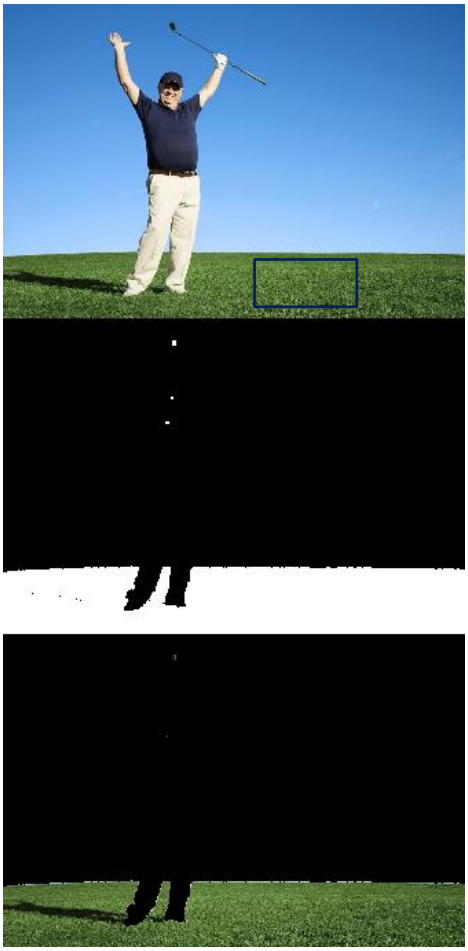
يستخدم الإسقاط الخلفي للمخطط البياني مع خوارزمية ملاحقة الأجسام Camshift (سيتم شرح خوارزمية Camshift لاحقاً).

كيف نستطيع القيام بالخطوات التي شرحناها سابقاً؟

ننشأ مخطط بياني للصورة التي تحوي على الجسم المطلوب (في صورتنا الأجسام هي الأرض واللاعب والسماء) (في الصورة التالية عيَّنا الأرض بأنها الجسم المطلوب)، الجسم يجب أن يملأ الصورة حتى نحصل على أفضل نتيجة، ويفضل استعمال المخطط البياني الملون(2D) بدلاً من المخطط الرمادي(1D)، لأن لون الجسم أفضل طريقة لتحديد الجسم مقارنة مع شدة اللون الرمادي.

ثم نقوم بإسقاط المخطط البياني لصورة الجسم خلفيا على صورة الاختبار (التي نحن بحاجة لأن نجد الجسم فيها). بكلمات أخرى سنحسب احتمال انتماء كل بيكسل للأرض ونعرضه، وبعد ذلك بعملية التعتيب نحصل على الأرضية لوحدها.

في الصورة التالية استعملت المنطقة التي بداخل المستطيل الأزرق كعينة بسيطة (الجسم المطلوب)، وأردت أن استخرج كامل الأرضية عن طريق هذ العينة باستخدام خوارزمية الإسقاط الخلفي.



- **خوارزمية الإسقاط الخلفي في Numpy:**

في البداية نحتاج أن نحسب المخطط البياني الملون للجسم الذي نريد إيجاده (وليكن 'M') والصورة التي نريد البحث فيها عن الجسم (ولتكن 'I').

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

#roi is the object or region of object we need to find
roi = cv2.imread('rose_red.png')
hsv = cv2.cvtColor(roi,cv2.COLOR_BGR2HSV)

#target is the image we search in
target = cv2.imread('rose.png')
hsvt = cv2.cvtColor(target,cv2.COLOR_BGR2HSV)

# Find the histograms using calcHist. Can be done with np.histogram2d also
M = cv2.calcHist([hsv],[0, 1], None, [180, 256], [0, 180, 0, 256] )
I = cv2.calcHist([hsvt],[0, 1], None, [180, 256], [0, 180, 0, 256] )
```

ثم نوجد النسبة $\boldsymbol{R = \frac{M}{I}}$ ، ثم نقوم بعملية backproject على R، أي استخدام R كلوحة وإنشاء صورة جديدة فيها كل بيكسل محتمل أن يكون مطابق للهدف، أي $\boldsymbol{B(x,y) = R[h(x,y), s(x,y)]}$
حيث h (hue) تمثل التدرج اللوني للبيكسل الذي إحداثياته (x,y)، وS تمثل الإشباع اللوني للبيكسل (x,y). بعد ذلك نطبق الشرط $\boldsymbol{B(x,y) = min[B(x,y), 1]}$.

```
R = M/(I+1)

h,s,v = cv2.split(hsvt)
B = R[h.ravel(),s.ravel()]
B = np.minimum(B,1)
B = B.reshape(hsvt.shape[:2])
```

ravel()
reshape()

الآن نطبق عملية اللف (الطي) مع قناع على شكل قرص دائري $\boldsymbol{B = D * B}$ حيث D تمثل القناع القرصي.

```
disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
cv2.filter2D(B,-1,disc,B)
B = np.uint8(B)
cv2.normalize(B,B,0,255,cv2.NORM_MINMAX)
```

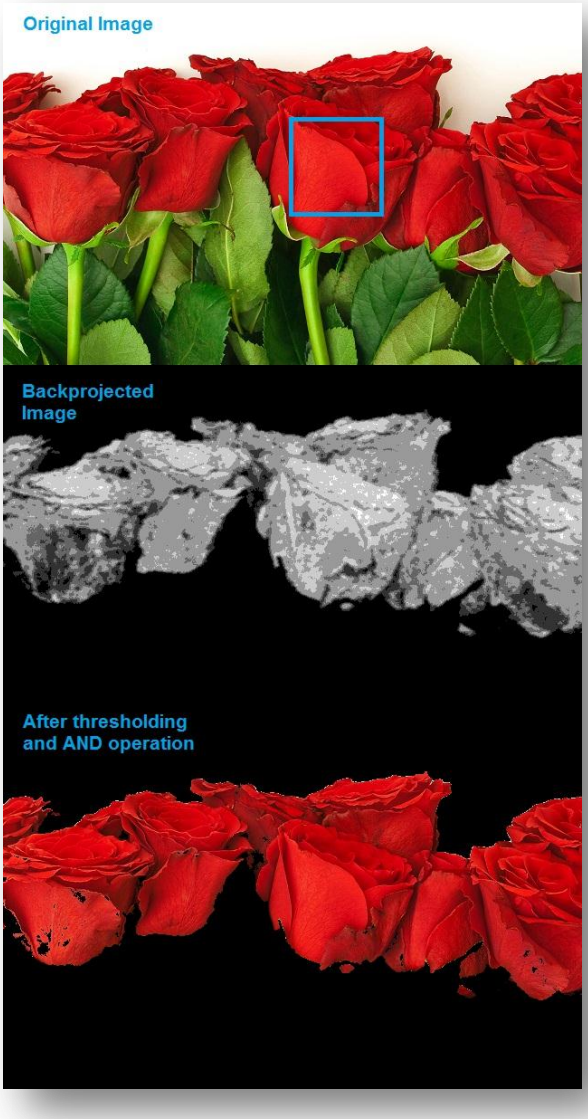
getStructuringElement()

cv2.normalize()

الآن نحصل على مكان الجسم عن طريق أعظم شدة لونية مُعطى للجسم، وبعدها يمكننا القيام بالتعتيب للحصول على نتائج أفضل.

```
ret,thresh = cv2.threshold(B,50,255,0)
```

هذا هو كل الأمر!!



- **خوارزمية الإسقاط الخلفي في OpenCv:**

توفر لنا OpenCvالتابع cv2.calcBackProject()، بارومتراته تقريبا هي نفس التابع cv2.calcHist(). أحد بارومتراته هو المخطط البياني للجسم المراد إيجاده، ويجب تعديله قبل تمريره لتابع الإسقاط الخلفي. يعيد هذا التابع الصورة المحتملة. بعد ذلك يجب علينا أن نمرر قناع قرصي على الصورة ثم نطبق عملية التعتيب. الكود التالي يوضح ذلك:

```
import cv2
import numpy as np

roi = cv2.imread('rose_red.png')
hsv = cv2.cvtColor(roi,cv2.COLOR_BGR2HSV)

target = cv2.imread('rose.png')
hsvt = cv2.cvtColor(target,cv2.COLOR_BGR2HSV)

# calculating object histogram
roihist = cv2.calcHist([hsv],[0, 1], None, [180, 256], [0, 180, 0, 256] )

# normalize histogram and apply backprojection
cv2.normalize(roihist,roihist,0,255,cv2.NORM_MINMAX)
dst = cv2.calcBackProject([hsvt],[0,1],roihist,[0,180,0,256],1)

# Now convolute with circular disc
disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
cv2.filter2D(dst,-1,disc,dst)

# threshold and binary AND
ret,thresh = cv2.threshold(dst,50,255,0)
thresh = cv2.merge((thresh,thresh,thresh))
res = cv2.bitwise_and(target,thresh)

res = np.vstack((target,thresh,res))
cv2.imshow('res',res) , cv2.waitKey(0), cv2.destroyAllWindows()
```

شرح الكود:

```
roi = cv2.imread('rose_red.png')
hsv = cv2.cvtColor(roi,cv2.COLOR_BGR2HSV)
```

قمنا بتخزين صورة الجسم المطلوب (الوردة الحمراء) في المتغير roi، ثم حولنا الصورة من الفضاء RGB إلى HSV لسهولة التعامل مع هذا الفضاء اللوني.

```
target = cv2.imread('rose.png')
hsvt = cv2.cvtColor(target,cv2.COLOR_BGR2HSV)
```

ثم قمنا بتخزين الصورة الهدف أي الصورة التي سنبحث فيها عن الجسم (صورة الورود) في المتغير target، ثم حولنا الصورة من الفضاء RGB إلى HSV لسهولة التعامل مع هذا الفضاء اللوني.

```
# calculating object histogram
roihist = cv2.calcHist([hsv],[0, 1], None, [180, 256], [0, 180, 0, 256] )
```

بعد ذلك قمنا بتهيئة المخطط البياني للصورة الجسم المطلوب وذلك عن طريق التابع cv2.calcHist().

```
# normalize histogram and apply backprojection
cv2.normalize(roihist,roihist,0,255,cv2.NORM_MINMAX)
```

```
dst = cv2.calcBackProject([hsvt],[0,1],roihist,[0,180,0,256],1)
```

هنا قمنا بعملية الاسقاط على الخلفية عن طريق التابع calcBackProject . بارامتراته هي:

cv2.calcBackProject(images, channels, hist, ranges, scale[, dst]) ⟶ dst

- المتغير الأول images: يحدد صور الدخل (هنا مررنا فقط صورة واحدة وهي الصورة الهدف التي سنبحث فيها عن الجسم)، والتي يجب أن تكون بنفس الحجم ونفس العمق اللوني CV_8U or CV_32F.
- المتغير الثاني channels يحدد القنوات التي سيطبق عليها عملية الاسقاط على الخلفية (back projection).
- المتغير الثالث hist: سنمرر له المخطط البياني لصورة الجسم المطلوب.

- المتغير الرابع range سنمرر له حدود أبعادالمخطط البياني.
- المتغير الخامس] scale: (اختياري) عامل تحديد الحجم لنتيجة عملية الاسقاط على الخلفية.
- التابع calcBackProject يحسب الاسقاط الخلفي للمخطط البياني. وهو مشابه للتابع calcHist الذي يوجد المخطط البياني عند كل موضع (x,y) حيث التابع يجمع القيم من القنوات اللونية المختارة لصورة الدخل، ويوجد البنْ (BIN) للمخطط البياني للصورة. ولكن هنا بدل زيادة عدد البن، التابع يقرأ قيم البن، ويجدولها حسب الحجم، ويخزنها في (x,y).

يمكنك إيجاد وتعقب الأجسام ذو اللون الزاهي في المشهد من خلال اتباع ما يلي:

١. قبل تعقب الجسم نقوم بتصوير الجسم المطلوب تعقبه بالكاميرا بحيث يغطي أكبر مساحة من الصورة الملتقطة. ثم نحول صورة الجسم التي التقطناها للفضاء HSV ونحسب المخطط البياني للصورة ضمن القناة H (hue). قد يتضمن المخطط البياني نقاط قوة وذلك حسب لون الجسم السائد.
٢. عند تعقب الجسم نحول كل أطار صورة (frame) من الفيديو الملتقط من الكاميرا (الذي سنبحث ضمنه عن الجسم) إلى الفضاء HSV، ثم نوجد الاسقاط الخلفي لكل frameضمن الفضاء HSV وذلك عن طريق إجراء عمليات حسابية على المخطط البياني لصورة الجسم المطلوب. ثم نقوم بإجراء عملية التعتيب على نتيجة الاسقاط الخلفي حتى نخمد الألوان الضعيفة. وأيضا يمكن أن تخمد البيكسلات التي يكون إشباعها اللوني غير كافي، كالبيكسلات العاتمة جداً أو الساطعة جداً.
٣. نوجد الأجزاء التي يوجد بينها ترابط في صورة الخرج، على سبيل المثال أكبر جزء.

هذه الخطوات تقريبا تشبه خوارزمية **Camshift** لتعقب الجسم عن طريق اللون.

```
# Now convolute with circular disc
disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
```

في هذا الأمر سنقوم بإنشاء قتاع قرصي وذلك عن طريق التابع getStructuringElement حيث نمرر له الشكل الذي نريده. هنا مررنا لهذا التابع شكل قطع ناقص (قرص)، والمتغير الثاني لهذا التابع نمرر له حجم القناع.

```
cv2.filter2D(dst,-1,disc,dst)
```

هنا سنقوم بالاستعانة بالتابع cv2.filter2D لترشيح الصورة وفقا للقناع القرصي.
المتغير الأول هو صورة الدخل، والثاني بحدد العمق اللوني لصورة الخرج وإذا كانت قيمته (-1) سنحصل على صورة بنفس العمق اللوني لصورة الدخل، والمتغير الثالث يمثل القناع الذي يجب أن يكون بقناة واحدة بمصفوفة من النوع float.

```
# threshold and binary AND
ret,thresh = cv2.threshold(dst,50,255,0)
thresh = cv2.merge((thresh,thresh,thresh))
res = cv2.bitwise_and(target,thresh)
```

وأخيرا سنقوم بعملية تعتيب الصورة الناتجة عن الإسقاط على الخلفية لإخماد الألوان التي شدتها اللونية ضعيفة.

```
res = np.vstack((target,thresh,res))
cv2.imshow('res',res) , cv2.waitKey(0), cv2.destroyAllWindows()
```

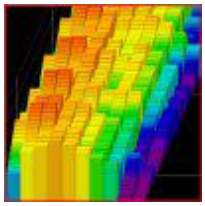
التابع np.vstack() سنستعمله لجمع ثلاثة صور في صورة واحدة، ثم سنعرض النتيجة ضمن نافذة. أي سنعرض نتائج ثلاث صور في نافذة واحدة.

الفصل السادس

”النجاح ليس نتيجة لعدم ارتكاب أي أخطاء،
ولكنه نتيجة لعدم تكرار نفس الخطأ مرتين “

جورج برنارد شو - مؤلف أيرلندي شهير.

الفصل السادس: تحويلات الصورة في OpenCV



- **تحويل فورييه:**

تعلم إيجاد تحويل فورييه للصورة.

تحويل فورييه للصورة Fourier Transform

الهدف:

- إيجاد تحويل فورييه للصورة باستخدام OpenCv.
- الاستفادة من توابع الـFFT الموجودة في Numpy.
- استخدام تحويل فورييه في بعض التطبيقات.
- تعلم استخدام التوابع التالية: cv2.dft() ،cv2.idft() .

معلومات نظرية:

يستخدم تحويل فورييه لتحليل خصائص التردد لمختلف المرشحات.

في الصور ثنائية البعد 2D يستخدم تحويل فورييه المتقطع DFT (Discrete Fourier Transform) لإيجاد المجال الترددي. أسرع خوارزمية لحساب تحويل فورييه هي FFT (Fast Fourier Transform) للحصول على تفاصيل أكثر عن هذا الموضوع يمكنك قراءة أي كتاب يتكلم عن معالجة الصورة أو الإشارة.

بالنسبة للإشارة الجيبية $\mathbf{x(t) = Asin(2\pi ft)}$ يمكننا اعتبار f هو تردد الإشارة، وإذا أخذنا مجالها الترددي بعين الاعتبار سنرى spike (الزيادة المفاجأة في الإشارة) في تردد الإشارة f. إذا أخذنا عينات من الإشارة لتشكيل إشارة متقطعة، سنحصل على نفس المجال الترددي، ولكن ستكون بشكل دوري في المجال [-π,+π] أو [0 , 2π] (أو [0 , N] بالنسبة للنقطة N في تحويل فورييه المتقطع DFT)

يمكننا أن نعتبر الصورة كالإشارة التي أخذنا عينات منها في كلا الاتجاهين x,y. لذلك عندما نأخذ تحويل فورييه للصورة في كلا الاتجاهين X وY سنحصل على التردد الذي سيمثل الصورة. بشكل بديهي، إذا كان مطال الإشارة الجيبية يتغير بسرعة كبيرة وفي وقت قصير يمكننا اعتبارها إشارة ذات تردد مرتفع، أما إذا كان مطال الإشارة يتغير ببطاء ستكون إشارة ذات تردد منخفض، يمكننا تطبيق نفس الفكرة على الصورة.

متى مطال الصورة يتغير بشكل كبير؟

يتغير بشكل كبير عند نقاط الحواف أو الضجيج، لذلك يمكننا القول بأن نقاط الحواف والضجيج ذات تردد مرتفع في الصورة. أما إذا لم يكن هناك تغير كبير في المطال ستكون النقاط ذات تردد منخفض في الصورة. والآن سنشاهد كيف سنوجد تحويل فورييه للصورة.

- **تحويل فورييه في Numpy:**

سنوجد تحويل فورييه باستخدام مكتبة Numpy. حيث تملك مجموعة FFT لإيجاد تحويل فورييه.

التابع المسؤول عن تحويل فورييه هو np.fft.fft2()، سيعطينا هذا التابع التحويل الترددي كمصفوفة عقدية.

المتغير الأول لهذا التابع هو صورة الدخل (يجب أن تكون رمادية)،

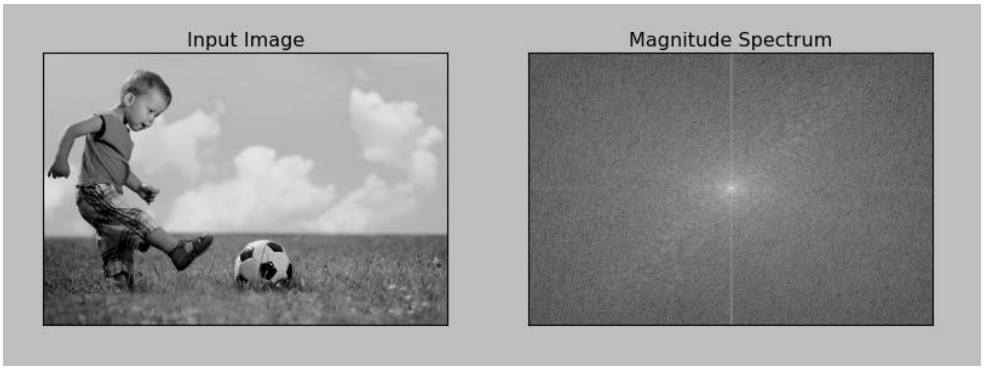
ثاني متغير خياري وهو المسؤول عن تحديد قياس مصفوفة الخرج، فإذا كان القياس أكبر من صورة الدخل سيتم تمديد الصورة بأصفار قبل حساب FFT، وإذا كان أقل من حجم صورة الدخل سيتم قص الصورة، وإذا لم نمرر أي قيمة في هذا المتغير سيكون قياس صورة الخرج بنفس قياس صور الدخل.

الآن بعد أن حصلنا على النتيجة، فإن التردد الصفري للعنصر (المركبة المستمرة DC) سيكون في أعلى الزأوية اليسرى، لذلك سنحتاج لإزاحة النتيجة بمقدار 2N في كلا الاتجاهين، وذلك من خلال التابع np.fft.fftshift() (لسهولة التحليل).

بعد ذلك علينا إيجاد حجم الطيف (magnitude spectrum) كما يلي:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('kid_football.jpg',0)

f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20*np.log(np.abs(fshift))

plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

النتيجة ستبدو كما يلي:

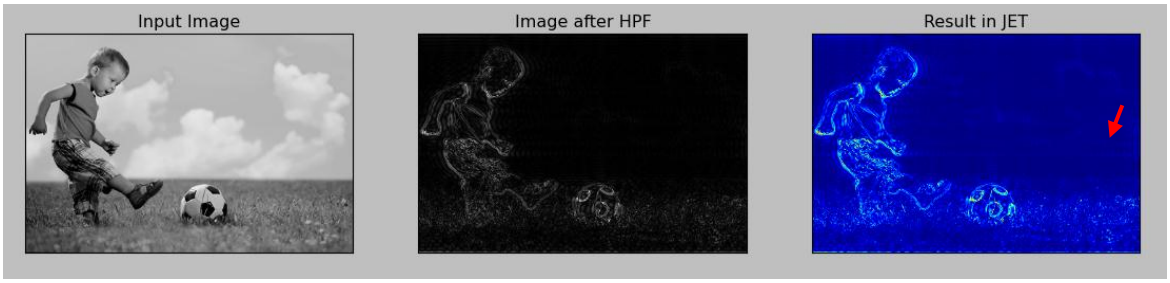


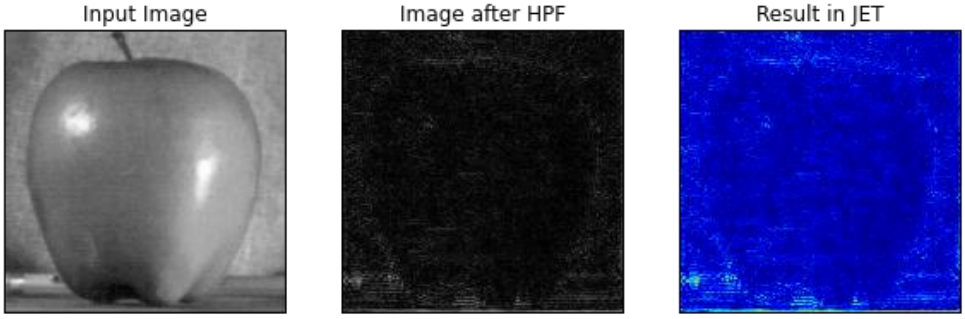
يمكننا أن نرى بأن المنطقة الأكثر بياضا هي في المركز وهذا يبين لنا بأن الترددات المنخفضة هي السائدة.

بعد إيجاد التردد من خلال التحويل الترددي يمكنك الآن القيام بعدة عمليات على المجال الترددي مثل ترشيح التردد المرتفع، وإعادة بناء الصورة أي إيجاد عكس DFT، لذلك يمكنك ببساطة أن تزيل الترددات المنخفضة عن طريق تطبيق قناع مستطيلي بحجم 60x60، ثم نطبق التحويل العكسي باستخدام التابع np.fft.ifftshift() بحيث تأتي المركبة المستمرة DC في الزأوية اليسرى العليا مرة أخرى (أي تعود الصورة لمكانها).

بعد ذلك نوجد تحويل فورييه العكسي باستخدام التابع np.ifft2()، والنتيجة أيضا ستكون أعداد عقدية، عندها يمكننا أخذ قيمتها المطلقة.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('kid_football.jpg',0)

f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)

rows, cols = img.shape
crow,ccol = rows/2 , cols/2

fshift[crow-30:crow+30, ccol-30:ccol+30] = 0
f_ishift = np.fft.ifftshift(fshift)
img_back = np.fft.ifft2(f_ishift)
img_back = np.abs(img_back)

plt.subplot(131),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(img_back, cmap = 'gray')
plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(img_back)
plt.title('Result in JET'), plt.xticks([]), plt.yticks([])
plt.show()
```





تظهر الصورة الناتجة الحواف التي في الصورة نتيجة لتطبيق مرشح التردد المرتفع، وهذا يدل أيضا على أن معظم البيانات التي في الصورة تقع في منطقة التردد المنخفض من الطيف. لقد شاهدنا كيف أوجدنا تحويل فورييه DFT وIDFT (تحويل فورييه المتقطع وعكسه) وغيرها في Numpy، وفيما يلي سنقوم بنفس الأمر ولكن ضمن مكتبةOpenCv.

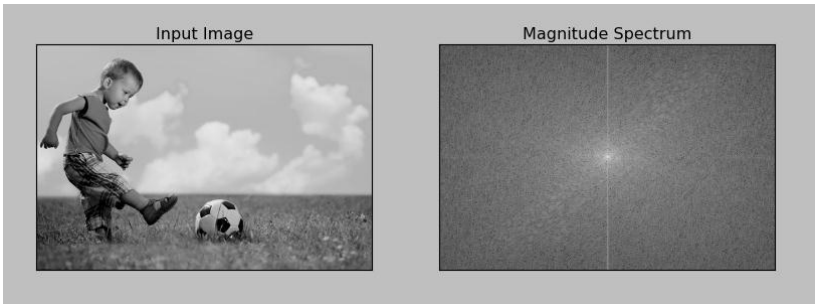
إذا لاحظت في الصور الناتجة هناك عدا الحواف نقاط ضجيج أخرى (مثل التي عند السهم الأحمر)، وذلك بسبب تأثير النافذة المستطيلة المستعملة كقناع. تسمى نقاط الضجيج هذه بتأثير الاهتزاز ringing effects، وللتغلب على هذه المشكلة استعمل المرشح الغأوسي بدل مرشح النافذة المستطيلة.

- **تحويل فورييه في OpenCv:**

مكتبة OpenCvتوفر لنا التابعيين () cv2.dft،cv2.idft() للقيام بعملية التحويل. يعيد نفس النتائج السابقة ولكن بقناتين، أول قناة تمثل القسم الحقيقي للصورة الناتجة، وثاني قناة تمثل القسم التخيلي للصورة الناتجة. صورة الدخل يجب أن تكون بالصيغة np. float32.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('kid_football.jpg',0)

dft = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

magnitude_spectrum =
20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

النتيجة:



يمكنك أيضا استخدام التابع cv2.cartToPolar() الذي يعيد كل من الطويلة والزأوية في نفس اللحظة. الآن سنقوم بإيجاد التحويل العكسي لتحويل فورييه المتقطع (IDFT)، في السابق شاهدنا كيف طبقنا مرشح ترد HPF، أما هذه المرة سنطبق مرشح LPF (الذي سوف يزيل الترددات المرتفعة من الصورة مما يؤدي لتغبيش الصورة). وفي البداية نقوم بإنشاء قناع يأخذ القيمة 1عند الترددات المنخفضة.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('kid_football.jpg',0)

dft = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

rows, cols = img.shape
crow,ccol = rows/2 , cols/2

# create a mask first, center square is 1, remaining all zeros
mask = np.zeros((rows,cols,2),np.uint8)
mask[crow-30:crow+30, ccol-30:ccol+30] = 1

# apply mask and inverse DFT
fshift = dft_shift*mask
f_ishift = np.fft.ifftshift(fshift)
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])

plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(img_back, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

النتيجة:

ملاحظة: كالعادة توابع تحويل فورييه [cv2.dft () ,cv2.idft()] لمكتبة **OpenCv** أسرع من نظيراتها في **Numpy**. ولكن توابع **Numpy** هي أكثر سهولة في الاستخدام.

الأداء المثالي لتحويل فورييه المتقطع Performance Optimization of DFT:

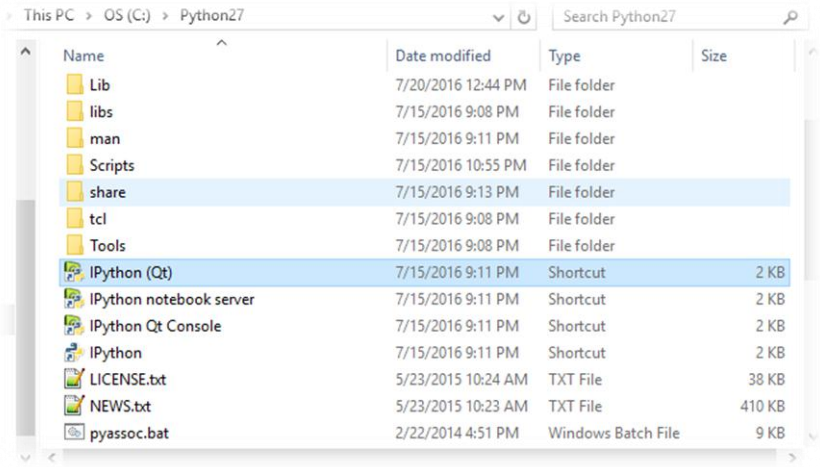
أداء تحويل فورييه المتقطع DFT جيد بالنسبة لبعض أحجام المصفوفة. فزمن التنفيذ يكون أسرع عندما تكون المصفوفة مرفوعة للقوى ٢. أيضاً المصفوفات التي قياسها يكون ضعفي أو ٣ أضعاف أو ٥ أضعاف يتم معالجتها بكفاءة تامة.

لذلك إذا كنت قلق بخصوص أداء الكود، يمكنك تعديل حجم المصفوفة لأحد الأحجام الاختيارية (وذلك بحشو المصفوفة بأصفار) قبل إيجاد تحويل فورييه المتقطع DFT. (أي نزيد حجم المصفوفة ونعبئ الحجم الزائد بأصفار). في مكتبة OpenCV يجب أن تقوم بحشو المصفوفة بأصفار بشكل يدوي، ولكن في مكتبة Numpy ستحدد القياس الجديد لحساب تحويل فورييه FFT (تحويل فورييه السريع) وسوف يتم تحشو المصفوفة بأصفار بشكل أتوماتيكي.

السؤال الآن، كيف نوجد الحجم المثالي للمصفوفة؟

مكتبة OpenCv توفر لنا التابع cv2.getOptimalDFTSize() لمعرفة الحجم المثالي. يمكنك تطبيق هذا الأمر على كل من cv2.dft() و np.fft.fft2().

دعنا نتحقق من الأداء باستخدام أداة حساب زمن التنفيذ %timeit التي توفرها لنا بيئة IPython. ستجد أداة IPython ضمن هذا المسار "C:\Python27\IPython (Qt)"




```
In [1]: cd Pictures
C:\Users\ASUS\Pictures
In [2]: import cv2
In [3]: import numpy as np
In [4]: img = cv2.imread('kid_football.jpg',0)
In [5]: rows,cols = img.shape
In [6]: print rows,cols
```

```
334 500
```

```
In [7]: nrows = cv2.getOptimalDFTSize(rows)
In [8]: ncols = cv2.getOptimalDFTSize(cols)
In [9]: print nrows, ncols
```

```
360 500
```

عند المقارنة بين النتائج ستلاحظ بأنه تم تعديل الحجم من (334,500) إلى (360,500).

الآن دعنا نحشوها بأصفار (في OpenCv نقوم بحشوها بأصفار بشكل يدوي) ونوجد زمن تحويل فورييه DFT. يمكن فعل هذا الأمر بإنشاء مصفوفة صفرية كبيرة الحجم، وننسخ بيانات الصورة إلى هذه المصفوفة، أو نستخدم التابع cv2.copyMakeBorder().

```
In [10]: nimg = np.zeros((nrows,ncols))
In [11]: nimg[:rows,:cols] = img
```

OR

```
In [12]: right = ncols - cols
In [13]: bottom = nrows - rows
In [14]: bordertype = cv2.BORDER_CONSTANT
In [15]: nimg = cv2.copyMakeBorder(img,0,bottom,0,right,bordertype, value = 0)
```

الآن سنحسب الأداء بالنسبة لمكتبة Numpy:

```
In [16]: %timeit fft1 = np.fft.fft2(img)
```

loops, best of 3: 45.5 ms per loop ١٠

```
In [17]: %timeit fft2 = np.fft.fft2(img,[nrows,ncols])
```

loops, best of 3: 8.46 ms per loop ١٠٠

بالمقارنة بين السرعتين ستجد أن أداء الكود الثاني كان أفضل فهو أسرع بحوالي ٤ مرات من الكود الأول. الآن سنجرب نفس الأمر ولكن هذه المرة باستخدام توابع مكتبة OpenCv.

```
In [18]: %timeit dft1= cv2.dft(np.float32(img),flags=cv2.DFT_COMPLEX_OUTPUT)
```

loops, best of 3: 2.22 ms per loop ١٠٠

```
In [19]: %timeit dft2= cv2.dft(np.float32(nimg),flags=cv2.DFT_COMPLEX_OUTPUT)
```

loops, best of 3: 1.73 ms per loop ١٠٠٠

أيضا بالمقارنة بين زمن تنفيذ الكود الأول والثاني ستجد أن زمن تنفيذ الكود الثاني كان أسرع. ويمكنك المقارنة أيضا بين أداء توابع مكتبة OpenCV ومكتبة Numpyفستجد أن زمن تنفيذ توابع مكتبة OpenCv أسرع بكثير من زمن تنفيذ لتوابع في Numpy.

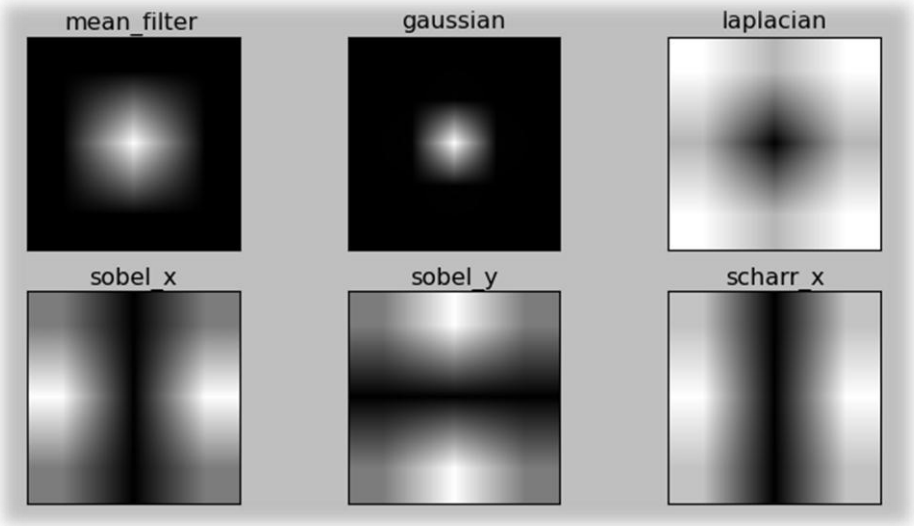
لماذا المرشح الابلاسي هو مرشح تردد مرتفع؟

لقد تم طرح هذا السؤال في بعض المنتديات، لماذا لابلاسيان هو مرشح HPF؟ ولماذا Sobel هو مرشح HPF؟

الإجابة كانت حسب تحويل فورييه.خذ فقط تحويل فورييه للابلاسيان للأحجام كبيرة لـ FFT، وقم بتحليلها بما ان القناع في المجال الزمني فتحويله للمجال الترددي سيوضح سلوكه، كالتالي:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# simple averaging filter without scaling parameter
mean_filter = np.ones((3,3))

# creating a guassian filter
x = cv2.getGaussianKernel(5,10)
gaussian = x * x.T

# different edge detecting filters
# scharr in x-direction
scharr = np.array([[-3, 0, 3],
                   [-10,0,10],
                   [-3, 0, 3]])

# sobel in x direction
sobel_x= np.array([ [-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]])

# sobel in y direction
sobel_y= np.array([ [-1,-2,-1],
                    [0, 0, 0],
                    [1, 2, 1]])

#laplacian
laplacian=np.array([[0, 1, 0],
                    [1,-4, 1],
                    [0, 1, 0]])
```

```
filters = [mean_filter, gaussian, laplacian, sobel_x,
           sobel_y, scharr]
filter_name = ['mean_filter',
                    'gaussian','laplacian',
                    'sobel_x', \
                    'sobel_y', 'scharr_x']

fft_filters = [np.fft.fft2(x) for x in filters]
fft_shift = [np.fft.fftshift(y) for y in fft_filters]
mag_spectrum = [np.log(np.abs(z)+1) for z in fft_shift]

plt.figure(figsize=(10,5))
for i in xrange(6):
   plt.subplot(2,3,i+1),plt.imshow(mag_spectrum[i],cmap = 'gray')
   plt.title(filter_name[i]), plt.xticks([]), plt.yticks([])

plt.show()
```



من الصور الناتجة يمكننا بوضوح تقرير فيما إذا كان المرشح ممر للترددات العالية أم المنخفضة.

مراجع إضافية:

- by Steven LeharAn Intuitive Explanation of Fourier Theory
- at HIPRFourier Transform
- What does frequency domain denote in case of images?

الفصل السابع

" إن المهام العظام يمكن إنجازها حين يستغل الإنسان وقته بكفاءة "

د.إبراهيم الفقي

الفصل السابع: ميزة الكشف والوصف Feature Detection and Description

❖ فهم ميزات الصورة

ما هي الميزات الرئيسية في الصورة؟ كيف يمكن إيجاد هذه الميزات حتى تكون مفيدة لنا؟

❖ مكتشف زوايا هاريس Harris Corner Detection

الزوايا هي ميزة جيدة ولكن كيف يمكن إيجادها.

❖ مكتشف زوايا Shi-Tomasi & ميزات جيدة للتعقب

سنتعلم كيفية اكتشاف زوايا Shi-Tomasi.

❖ مقدمة إلى SIFT (ميزة التحويل الثابتة القياس) (Scale-Invariant Feature Transform)

مكتشف زوايا هاريس غير جيد كفاية عند تغير قياس الصورة، نتيجة لذلك قامت المطورة Lowe بوضع طريقة لإيجاد العناصر الثابتة القياس. وأطلقت عليها اسم SIFT.

مقدمة إلى SURF (الميزات القوية المسرعة) (Speeded-Up Robust Features)

إن ميزة SIFT جيدة، ولكنها غير سريعة بما فيه الكفاية، لذلك تم انشاء نسخة أسرع وأطلق عليها اسم SURF.

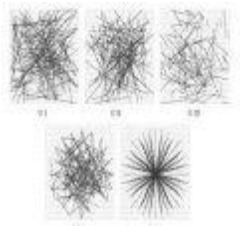
❖ FASTأسرع خوارزمية لاكتشاف الزوايا

كل طرق الميزات السابقة جيدة في بعض الأحيان، ولكنها ليست سريعة بما فيه

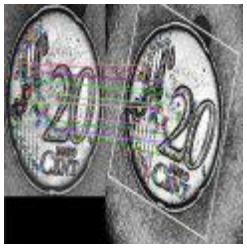
الكفاية لتعمل مع التطبيقات التي تعمل بالوقت الحقيقي) مثل السيارة ذاتية القيادة و ...)، لذلك تم إيجاد خوارزمية FAST والتي تعمل في الوقت الحقيقي.

❖ BRIEF(الميزات الابتدائية المستقلة القوية الثنائية)

(Binary Robust Independent Elementary Features)



بسبب استهلاك الذاكرة الكبير من قبل SIFT تم إيجاد BRIEF التي توفر في الذاكرة وتعطينا سرعة أكبر بالمطابقة.



❖ ORB (موجه FAST ومدور BRIEF) (Oriented FAST and Rotated BRIEF)

SIFT و SURF يقومون بعمل جيد، ولكن لمشكلة بأنه غير مجاني ويجب دفع بضع دولارات كل سنة.

لحل هذه المشكلة وفرت لنا مكتبة OpenCv بديل عن SIFT و SURF وهو ORB.

❖ ميزة المطابقة Feature Matching

لقد تعلمنا كثيرا على ميزة الكشف والوصف، والآن حان الوقت لنتعلم كيف نطابق مختلف الأوصاف.

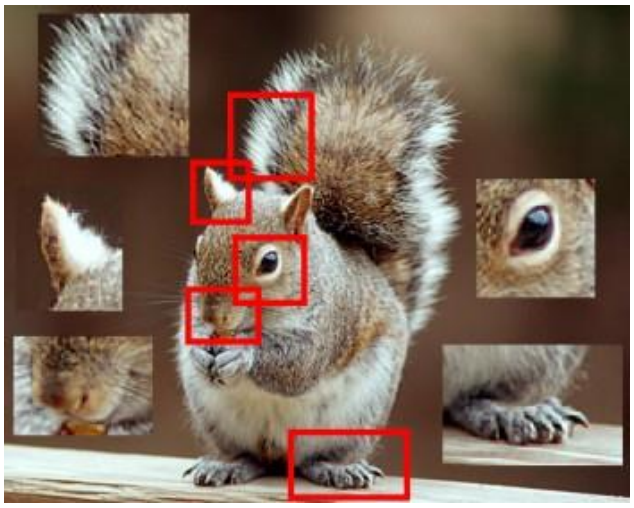
مكتبة OpenCv توفر لنا تقنيتين لهذا الأمر وهما:

مطابقة Brute-Force ومطابقة FLANN based.

❖ ميزة المطابقة + Homography لإيجاد الأجسام

الآن أصبحنا نعرف ما هي ميزة المطابقة دعنا الآن ندمجها مع الموديل calib3d لإيجاد الأجسام في الصورة المعقدة.

فهم الميزات Understanding Features



الهدف:

- في هذا الفصل سنحأول فقط فهم الميزات الرئيسة في الصورة (الخصائص المميزة في الصورة)، ولنعرف لماذا هي مهمة، ولماذا ميزة الزوايا هي ميزة مهمة.

الشرح:

في لعبة تركيب الصور نحتاج لتجميع القطع الصغيرة من الصور لتشكيل صورة كبيرة. كيف يستطيع الإنسان تمييز القطع المنفصلة المشكلة للصورة الواحدة؟ وهل يمكن للحاسوب نظرياً بأن يملك مثل هذه القدرة؟ وإذا أمكن هذا لماذا لا يمكننا إنشاء برنامج يقوم بجمع عدة صور لمناظر الحياة الحقيقية الطبيعة في صورة واحدة كبيرة؟ أو يمكننا إنشاء برنامج يصنع مجسم ثلاثي الأبعاد 3D من مجموعة صور لمجسم؟

جميع الأسئلة السابقة تركز حول كيف يستطيع الحاسب تجميع صور مختلفة لنفس المشهد في صورة واحدة؟

الجواب: نحن نبحث عن أنماط محددة أو ميزات فريدة والتي يمكن بسهولة تتبعها، ومقارنتها بسهولة.

إذا أردنا تعريف هذه الميزة، فمن الصعب التعبير عنها بكلمات ولكننا نعرف ما هي، فالقدرة على التمييز موجودة فينا بالأصل وحتى عند الأطفال الصغار لذلك تجدهم يستطيعون أن يلعبوا لعبة تركيب الصورة المفككة، بحيث يستطيعوا أن يجمعوا القطع ويحددوا التشابه فيما بينها.

والسؤال الآن ما هي هذه الميزات؟

الجواب: يجب أن يكون مفهوما من قبل الحاسب أيضا!

حسناً، فمن الصعب القول كيف تجد البشر هذه الميزات، لأنها مبرمجة مسبقا في الدماغ، ولكن إذا نظرنا بعمق إلى الصورة وبحثنا عن أنماط مختلفة، سنجد شيء مثير للاهتمام. على سبيل المثال لنأخذ الصورة التالية:

نلاحظ بأن القطع الستة الموجودة في اعلى الصورة من A ← F هي موجودة ضمن الصورة.

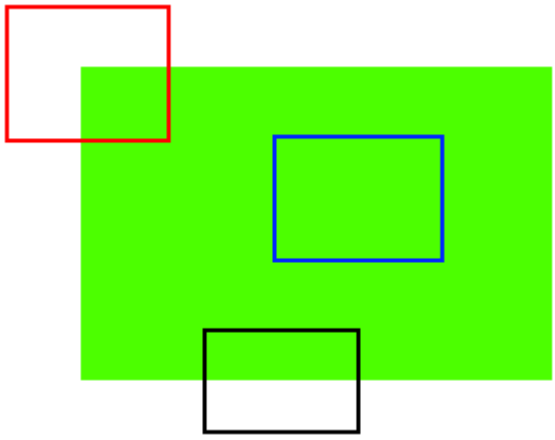
القطعة A و Bعبارة عن أسطح مستوية، وتنتشر في مناطق كثيرة من الصورة لذلك فمن الصعب العثور عليها في مكان محدد في الصورة.

القطعة CوD تمثل حواف المبنى، ويمكنك أن تجد موقعها بشكل تقريبي ولكن أن توجد مكانها المحدد فهذا صعب نوعا ما. لأن هذه القطعة موجودة على طول الحافة، وهي نفسها في كل مكان.

نلاحظ من طبيعة الحافة بأن ميزة الحواف أفضل من الأسطح المستوية، ولكنها ليست جيدة بما فيه الكفاية.

القطعتين E وF تمثل بعض زوايا المبنى، ويمكن العثور عليها بسهولة، لأن الزوايا عند تحريكها في أي اتجاه ستبدو مختلفة، لهذا يمكن اعتبارها ميزة جيدة.

الآن سننتقل للشرح التالي حتى نفهم بشكل أفضل:



الشرح بالضبط مثل فوق.
فالبقعة الزرقاء تمثل السطح المستوي، ومن الصعب العثور عليها وتعقبها، وأينما حركتها ستبدو نفسها. البقعة السوداء تمثل الحافة، إذا حركتها بشكل عمودي ستتغير، بينما إذا حركتها بشكل أفقي لن تتغير (وهذا الأمر تماماً مثل السابق فعندما نحرك الحافة بشكل موازي لطول الحافة ستبدو نفسها).

البقعة الحمراء تمثل الزأوية، أينما حركتها ستبدو بمنظر مختلف، لذلك نعتبر الزوايا ذو ميزة جيدة في الصورة (ليست الزوايا لوحدها هي ميزة جيدة، ففي بعض الحالات تعتبر النقاط أيضا ذو ميزة جيدة).

بعد أن أجبنا على السؤال الأول وهو، ما هي الميزات؟

السؤال الذي يطرح نفسه الآن، كيف نوجد هذه الميزات؟ أو كيف نوجد الزوايا؟
الجواب: من خلال الحدس، حيث نبحث عن مناطق في الصورة تتغير أكثر من غيرها عند تحريكها، ونعبر عن هذا بلغة الحاسوب بما يسمى اكتشف الميزات Feature Detection، وهذا ما سنتدأوله في الفصول القادمة.

لنفترض بأننا أوجدنا الميزات التي في الصورة، فبعد إيجادنا لها، يجب علينا مطابقتها مع صورة أخرى، ويتم ذلك بأخذ منطقة حول البقعة المميزة ونقوم بوصفها، (مثلا الجزء الذي في الأعلى

يمثل السماء الزرقاء، والقسم السفلي يمثل منطقة البناء، على البناء هناك بعد الزجاج، ...الخ)، وبشكل مشابه لهذا الأمر يقوم الحاسوب بذلك، ويسمى هذا بوصف الميزة Feature Description. بعد الحصول على الميزة والوصف يتم استخدام ذلك إما لترتيب الميزات أو للوصول إلى الصور أو لفعل أي شيء نريده.

لإيجاد الميزات ووصفها ومطابقتها في OpenCv نستخدم عدة خوارزميات متنوعة.

مكتشف زوايا هاريس Harris Corner Detection

الهدف:

- فهم مفهوم مكتشف زأويا هاريس
- سنتعلم استخدام التوابع التالية:cv2.cornerHarris()، cv2.cornerSubPix() .

مقدمة نظرية:

في الفصل السابق رأينا بأن الزوايا هي مناطق في الصورة فيها اختلاف كبير في الكثافة (الشدة) اللونية في جميع اتجاهات هذه المنطقة.

في هذه الخوارزمية سنقوم بتحويل الأفكار البسيطة السابقة حول الميزات لصيغة رياضية، فهذه الخوارزمية تقوم بشكل أساسي على حساب الاختلاف بالشدة عند تحريك (u,v) بكل الاتجاهات، وهذا ما يعبر عنه رياضيا كما يلي:

$$E(u,v) = \sum_{x,y} w(x,y)\,[I(x+u, y+v) - I(x,y)]^2$$

Window function شدة الإزاحة الشدة

تابع النافذة window function إما نافذة مستطيلة أو نافذة غأوسية، الغاية منه إعطاؤنا وزن البيكسلات التي تحت المستطيل. علينا الاستفادة من قيمة التابع $E(u,v)$ لإيجاد الزوايا. نطبق منشور تايلور الموسع على المعادلة السابقة وبالقيام ببعض الخطوات الرياضية نحصل على ما يلي:

$$E(u,v) \approx [u,v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$

حيث $M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$

I_x , I_y هما مشتقي الصورة بالاتجاهين x وy (يمكن إيجادهما بسهولة من خلال التابع cv2.Sobel()). (للتذكير تابع سوبيلSobel مر معنا مسبقا)

بعد ذلك نصل للجزء الرئيسي، والذي يحدد احتمال أن تحوي النافذة زأوية أو لا تحوي.

$$R = det(M) - K(trace(M))^2$$

حيث:

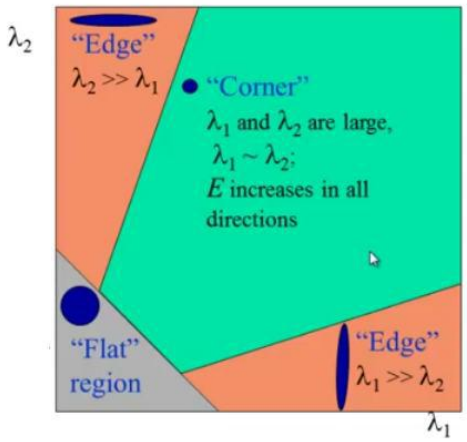
$$det(M) = \lambda_1 \lambda_2$$
$$trace(M) = \lambda_{1+} \lambda_2$$

λ_1 , λ_2 هما القيم الذاتية لـ M.

هذه القيم الذاتية هي التي تقرر فيما إذا كانت المنطقة هي زأوية أم حافة أم مستوي.

- عندما تكون |R| صغيرة، وهذا يحدث عندما λ_1 و λ_2 تكون قيمتهم صغيرة ← المنطقة مستوية.
- عندما تكون R<0، وهذا يحدث عندما $\lambda_2 \ll \lambda_1$أو بالعكس أي $\lambda_1 \ll \lambda_2$ ← المنطقة حافة.
- عندما تكون R كبيرة، وهذا يحدث عندما λ_1 وλ_2 تكون قيمتهم كبيرة و $\lambda_1 \approx \lambda_2$ ← المنطقة زأوية.

يمكن تمثيل هذه القيم على شكل صورة، كما يلي:



صورة الدخل لمكتشف زوايا هاريس يجب أن تكون صورة في المستوي الرمادي. وبأخذ العتبة لها بالشكل المناسب ستعطينا الزوايا التي في الصورة. سنقوم باستخدام مكتشف زوايا هاريس على صورة بسيطة.

- **مكتشف زأوية هاريس في OpenCV:**

توفر لنا مكتبة OpenCv التابع cv2.cornerHarris() لهذا الغرض. متغيراته هي:

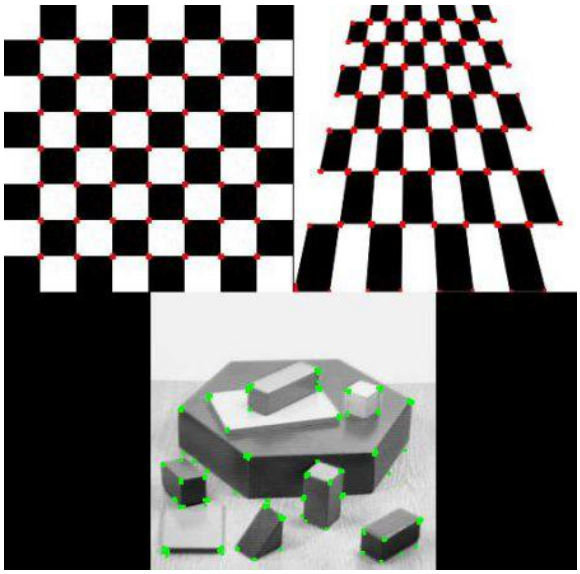
- Img: صورة الدخل ، يجب أت تكون رمادية ومن النوع float32.
- blockSize: قياس الجوار بالنسبة لاكتشاف الحواف.
- Ksize: بارامتر الفتحة لمشتق سوبيل sobel المُسْتخدم.
- K: البارامتر الحر لمكتشف هاريس في المعادلة.

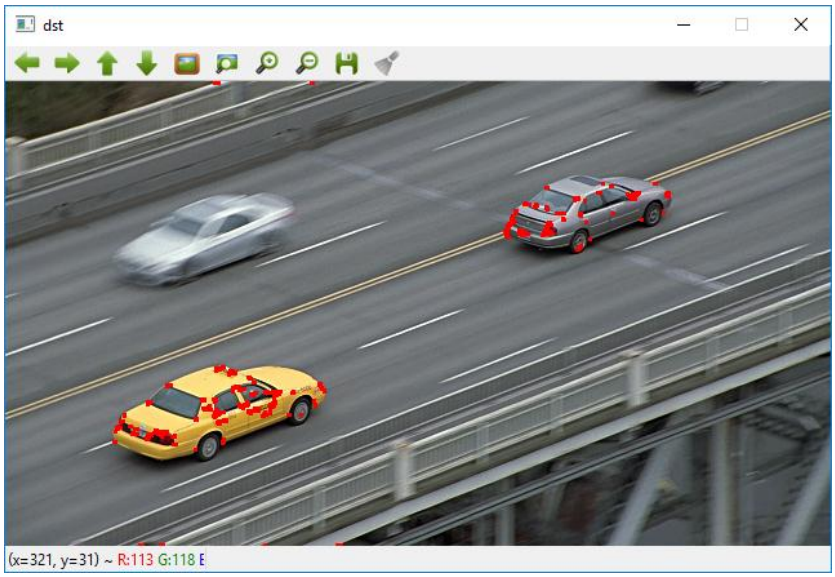
```
import cv2
import numpy as np
filename = 'car.jpg'
img = cv2.imread(filename)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

gray = np.float32(gray)
dst = cv2.cornerHarris(gray,2,3,0.04)

#result is dilated for marking the corners, not important
dst = cv2.dilate(dst,None)
```

```
# Threshold for an optimal value, it may vary depending on the image.
img[dst>0.01*dst.max()]=[0,0,255]
```

```
cv2.imshow('dst',img)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

النتيجة:





شرح الكود:

في البداية قمنا باستدعاء مكتبة OpenCv ومكتبة Numpy، ثم قمنا بتخزين الصورة في المتغير filename، ثم قمنا بقراءة الصورة عن طريق التابع cv2.imread()، ثم قمنا بتحويل الصورة إلى المستوي الرمادي عن طريق التابع cv2.imread()، ثم قمنا بتحويل الصورة للصيغة float32 وذلك عن طريق التابع np. float32() .

```
dst = cv2.cornerHarris(gray,2,3,0.04)
```

ثم قمنا باكتشاف زوايا هاريس عن طريق التابع cv2.cornerHarris() ولقد شرحنا سباقا ما هي البارامترات التي يأخذها. (ملاحظة لفهم كل متغير بشكل أوضح جرب تغيير كل قيمة لوحدها وشاهد ما الذي سيحصل في الصورة)

```
dst = cv2.dilate(dst,None)
```

التابع cv2.dilate() هو تابع التمديد (تم شرحه مسبقا في فصل التحولات المورفولوجية) فقط استخدمناه لتوضيح العلامات الملونة باللون الأحمر والتي تدل على زوايا هاريس (هذا السطر غير مهم يمكنك عدم كتابته).

```
# Threshold for an optimal value, it may vary depending on the image.
img[dst>0.01*dst.max()]=[0,0,255]
```

هذه التعليمة مسؤولة عن تلوين أماكن زوايا هاريس باللون الأحمر، حيث القيمة [0,0,255] تمثل اللون الأحمر في الفضاء BGR، وهذا السطر أيضا يحدد العتبة التي نريد إظهارها لهذه الزوايا، حيث قمنا هنا بتحديد قيمة العتبة الدنيا بـ ٠.٠١.

(جرب تكبير وتصغير قيمة العتبة وشاهد ما الذي سيحدث_ سترى بأنه كلما قمت بتخفيض العتبة الدنيا سيزيد عدد النقاط المعلمة باللون الأحمر لزوايا هاريس على الصورة، وكلما قمنا بزيادة هذه العتبة سينخفض عدد النقاط المعلمة باللون الأحمر لزوايا هاريس).

يمكنك استبدال سطر الأوامر هذا بالتعليمة التالية:

```
ret, dst = cv2.threshold(dst,0.01*dst.max(),255,0)
```

```
cv2.imshow('dst',img)
```

وفي النهاية نقوم بعرض الصورة ضمن نافذة اسمها 'dst' وذلك عن طريق التابع cv2.imshow().

- **دقة الزأوية مع البيكسل الفرعي Corner with SubPixel Accuracy:**

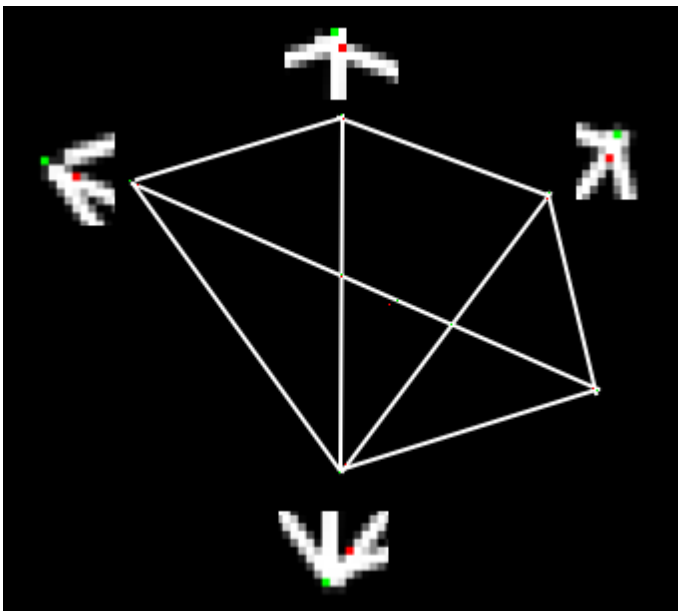
أحيانا نحتاج لدقة أكبر عند اكتشاف الزوايا.

مكتبة OpenCvتوفر لنا هذا الأمر من خلال التابع cv2.cornerSubPix(). فللحصول على دقة أكبر نمرر ناتج اكتشاف زوايا هاريس للتابع cv2.cornerSubPix والذي يقوم بترشيح الناتج، للحصول على نتائج أدق.

```
import cv2
import numpy as np

filename = 'car.jpg'
img = cv2.imread(filename)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

# find Harris corners
gray = np.float32(gray)
dst = cv2.cornerHarris(gray,2,3,0.04)
dst = cv2.dilate(dst,None)
ret, dst = cv2.threshold(dst,0.01*dst.max(),255,0)
dst = np.uint8(dst)

# find centroids
ret, labels, stats, centroids = cv2.connectedComponentsWithStats(dst)

# define the criteria to stop and refine the corners
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.001)
corners = cv2.cornerSubPix(gray,np.float32(centroids),(5,5),(-1,-1),criteria)

# Now draw them
res = np.hstack((centroids,corners))
res = np.int0(res)
img[res[:,1],res[:,0]]=[0,0,255]
img[res[:,3],res[:,2]] = [0,255,0]

cv2.imshow('dst',img)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

ملاحظة: التابع **cv2.connectedComponentsWithStats** غير موجود في مكتبة **OpenCV** الإصدار ٢.

النتيجة:



مكتشف زوايا Shi-Tomasi & ميزات جيدة للتعقب

الهدف:

- سنتعلم استخدام كاشف آخر للزوايا وهو: مكتشف الزأوية Shi-Tomasi.
- سنتعلم استخدام التابع التالي:cv2.goodFeaturesToTrack() .

النظري:

مكتشف الزوايا Shi-Tomasi يعطي نتيجة أفضل مقارنة مع مكتشف الزوايا هاريس.

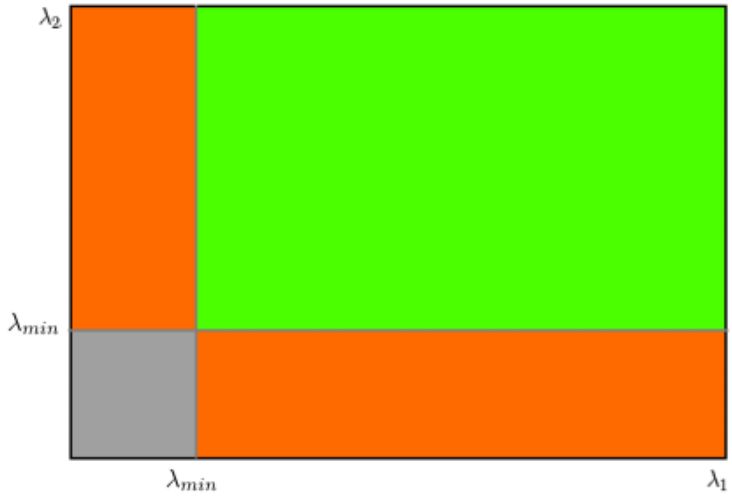
في مكتشف هاريس تم أخذ تابع التحصيل كما يلي:

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

بدلاً عن ذلك اقترح Shi-Tomasiالتالي:

$$R = min(\lambda_1, \lambda_2)$$

إذا كانت أكبر من قيمة العتبة نعتبرها زأوية. وإذا رسمناها في الفضاء $\lambda_1 - \lambda_2$ كما فعلنا مع مكتشف الزوايا هاريس فسنحصل على الصورة التالية.



نلاحظ من الصورة بأنه فقط عندما تكون قيمة λ_1 و λ_2 أعلى من قيمة الحد الأدنى λ_{min}نعدها زأوية (المنطقة الملونة باللون الأخضر).

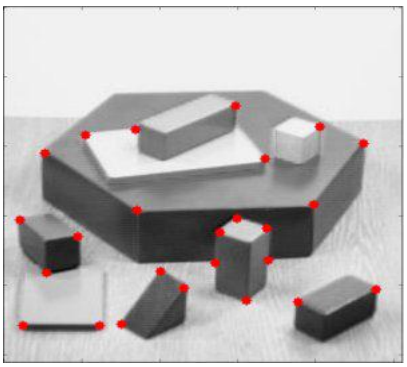
الكود:

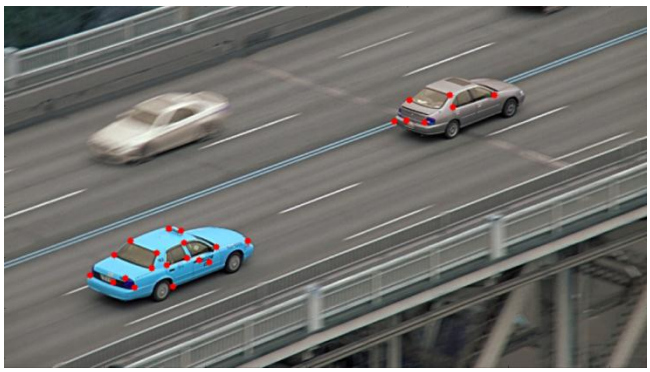
مكتبة OpenCv تحوي التابع cv2.goodFeaturesToTrack()، والذي يُوْجِدْ الزوايا الأقوى N في الصورة عن طريق مكتشف Shi-Tomasi (أو مكتشف زوايا هاريس إذا استعملته). الصورة يجب أن تكون في مستوى اللون الرمادي. قم بتحديد عدد الزوايا التي تريد إيجادها، ثم حدد مستوى الدقة، والذي يدل على الحد الأدنى لدقة الزأوية، بحيث إن قيمة أي زأوية تكون تحت الحد الأدنى ترفض (أي تأخذ القيمة 0). ثم نزود التابع بأقل مسافة مسموحة بين الزوايا المكتشفة.

كل هذه المعطيات تجعل التابع يوجد الزوايا التي في الصورة بحيث كل زأوية تحت مستوى الدقة المطلوبة ترفض، ثم تخزن الزوايا المتبقية اعتمادا على مستوى الدقة وبترتيب تنازلي. ثم يأخذ التابع أول أقوى زأوية، ويتخلص من كل الزوايا القريبة من مجال المسافة الأقل من المسموح، ويعيد الزوايا القوية N. في المثال التالي سنحأول إيجاد أفضل 25 زأوية:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('car.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

corners = cv2.goodFeaturesToTrack(gray,25,0.01,10)
corners = np.int0(corners)

for i in corners:
   x,y = i.ravel()
   cv2.circle(img,(x,y),3,255,-1)

plt.imshow(img),plt.show()
```

النتيجة:





من نتيجة الصورة نستنتج بأن هذا التابع مناسب أكثر للتعقب tracking، وهذا ما سنراه في الفصول القادمة.

شرح الكود:

في البداية نقوم بقراءة الصورة وتخزينها في المتغير img، ثم نقوم بتحويل الصورة لمستوي الون الرمادي.

```
corners = cv2.goodFeaturesToTrack(gray,25,0.01,10)
```

نقوم بإيجاد الزوايا الأقوى التي في الصورة وذلك عن طرق التابع cv2.goodFeaturesToTrack().

متغيرات هذا التابع هي:

- المتغير الأول هو صورة الدخل ويجب أن تكون في المستوي الرمادي.
- المتغير الثاني يحدد عدد الزوايا التي نريد اكتشافها.
- المتغير الثالث يحدد مستوى دقة الزوايا المختارة، حيث يحدد العتبة الدنيا للزوايا المختارة.
- المتغير الرابع يحدد أقل مسافة مسموحة بين الزوايا المكتشفة.

```
corners = np.int0(corners)
```

التابع np.int0() وظيفته هنا إلغاء الأرقام التي بعد الفاصلة للقيمة التي نمررها له. فمثلا إذا كان الرقم 5.76 يصبح الرقم ٥.

```
corners=5.76
x = np.int0(corners)
print x
```

```
5
```

مثال آخر:

```
corners= np.array([5.76,3.5,6.78,7])
x = np.int0(corners)
print x
```

```
[5 3 6 7]
```

```
for i in corners:
    x,y = i.ravel()
    cv2.circle(img,(x,y),3,255,-1)
```

ثم قمنا بعمل حلقة for حتى نخزن إحداثيات الزوايا التي حصلنا عليها في المتغيرين x وy، ثم نقوم بتمرير هذه الإحداثيات لتابع رسم الدوائر cv2.circle() حتى نعلم نقاط الزوايا بدوائر ملونة باللون الأحمر. التابع ravel () في مكتبة numpy مسؤول عن تحويل المصفوفة ثنائية البعد 2D إلى مصفوفة أحادية البعد 1D.

مثال:

```
a = np.array([[1,2], [4,5]])
a.ravel()
```

```
[1 2 4 5]
```

هنا قمنا بتحويل المصفوفة ثنائية البعد 2D إلى 1D حتى نستطيع الحصول على إحداثيات x لوحدها وإحداثيات y لوحدها لنمررها لتابع رسم الدوائر.

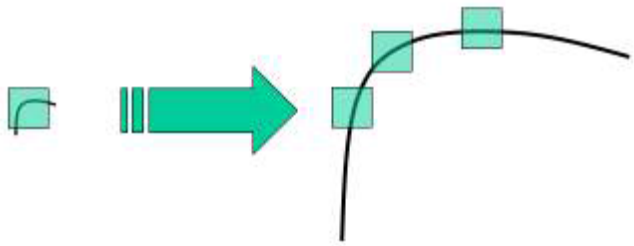
مقدمة إلى SIFT (Scale-Invariant Feature Transform)

الهدف:

- سنتعرف على خوارزمية SIFT.
- تعلم إيجاد النقاط الرئيسية لـ SIFT و مواصفاتها.

النظري:

في الفصلين السابقين شاهدنا بعض مكتشفات الزوايا كمكتشف هاريس، وشاهدنا كيف أن ميزات الصورة كميزة الزأوية تبقى كما هي عند تدوير الصورة، أي أن الزأوية لا تتأثر بالتدوير، ولكن ما الذي يحدث لميزة الزأوية إذا غيرنا حجم الصورة؟ الجواب يمكن أن تصبح ليست بزأوية، فعلى سبيل المثال شاهد الصورة التالية، ستشاهد زأوية في صورة صغيرة داخل نافذة صغيرة (النافذة تمثل القناع). إذا قمنا بتكبير الصورة مع الاحتفاظ بنفس حجم قناع النافذة ستعتبر مسطحة أي ليست بزأوية. وهذا يدل على أن زوايا هاريس تتأثر عند تغير حجم الصورة.



بسبب مشكلة تأثر زوايا هاريس عند تغير الحجم، قام المطورون بإيجاد خوارزمية جديدة تحافظ على ميزة الزوايا عند تغير الحجم وأطلقوا عليها ضمن أوراق البحث اسم SIFT (اختصار لكلمة: ميزة التحويل الثابتة الحجم)

تتألف هذه الخوارزمية من أربع مراحل وهي:

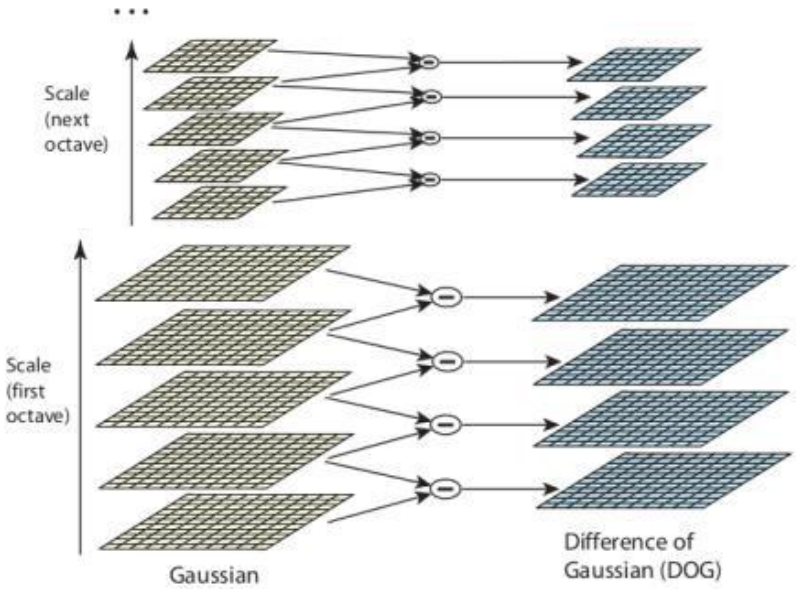
- اكتشاف القيمة العظمى للحجم.
- تحديد النقاط الأساسية التي تمثل الزوايا.
- تعيين الاتجاه (لإزالة تأثيرات الدوران والحجم).
- إنشاء واصف (باستخدام المخططات البيانية للاتجاهات).

١- مكتشف القيمة العظمى لقياس الحجم (Scale-space Extrema Detection) :

من الواضح من الصورة السابقة بأننا لا نستطيع استخدام نفس قناع النافذة للكشف عن النقاط الرئيسية (الزوايا) التي في الصور ذات الأحجام مختلفة. يمكن استخدام نفس النافذة للكشف عن الزوايا الأصغر من النافذة، ولكن للكشف عن الزوايا الكبيرة سنحتاج لنافذة أكبر، لهذا يتم استعمال مرشح قياس الحجم (Scale-space). والذي يعتمد على إيجاد لابلاسيان الغأوسي(LoG) للصورة مع قيم حجم مختلفة للصورة (σ).

لابلاسيان الغأوسي يعمل كمكتشف للنقاط، بحيث يكتشف النقاط حتى ولو تم تغيير حجم الصورة. بكلمات مختصرة، σ تمثل معامل القياس(الحجم)، على سبيل المثال في الصورة السابقة بتطبيق القناع الغأوسي على حجم صغير للصورة سنحصل على قيمة عالية لأصغر زأوية، بينما إذا طبقنا القناع الغأوسي على نفس الصورة ولكن بحجم أكبر سنحصل على قيمة تتناسب تماما مع كِبَرْ الزأوية.

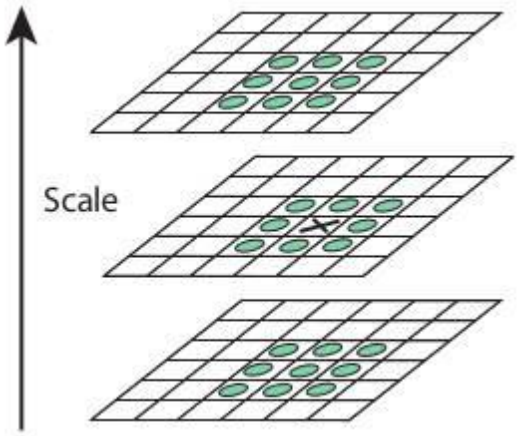
لذلك يمكننا إيجاد أكبر قيمة حالية للقياس والحجم، عندها سنحصل على قائمة من القيم (x , y , σ) والتي تعني وجود نقاط زوايا محتملة عند الإحداثية (x , y) عند الحجم σ. ولكن هذه العملية تستهلك وقت، لذلك سنلجأ لاستخدام الفرق الغأوسي في خوارزمية SFIT لأن هذا الأمر يعد أبسط ولكن يقوم بالمهمة المطلوبة. يتم هذا الأمر عن طريق طرح التشويش الغأوسي (Gaussian blurring) للصورة بحجمين مختلفين للصورة. هذه العملية تتم على قيم ثمانية مختلفة للصورة في الهرم الغأوسي، وهذا يتمثل في الصورة التالية:



number of octaves= 4، number of scale levels = 5

بعد إيجاد الفرق الغأوسي (DoG) يتم البحث عن القيمة العظمى الحالية ضمن المستويات المختلفة، فعلى سبيل المثال يتم مقارنة بيكسل واحد مع ٨ بيكسلات مجأورة له ويتم مقارنته أيضا مع ٩ بيكسلات من المستوى الذي بعده ومع ٩ بيكسلات من المستوى الذي قبله، وفي حالة وجود هذه القيمة العظمى هناك احتمالية ان تكون النقاط الرئيسية تمثل زأوية.

وهذا يعني بشكل أساسي بأن النقاط الرئيسية هي أفضل ممثل عند تغير حجم الصورة، وهذا ما توضحه الصورة التالية:



٢- تحديد النقاط الأساسية (Keypoint) التي تمثل الزوايا:

بعد إيجاد النقاط الرئيسية المحتمل أن تكون زوايا، يجب القيام بتصفيتها للحصول على النقاط الأكثر احتمالاً أن تكون زوايا، ويتم هذا بمساعدة منشور تايلور، وذلك عبر تحديد عتبة معينة للقيم العظمى وأي قيمة تكون أقل من هذه العتبة ترفض، وتدعى هذه العتبة بعتبة التباين contrastThreshold.

الفرق الغأوسي DoG لديه حساسية عالية للحواف لذلك يجب إزالتها، لهذا يتم استخدام مفهوم مشابه لمكتشف زوايا هاريس، وذلك عن طريق استخدام مصفوفة 2x2 Hessian (H) والتي تقارن القيم الذاتية للنقاط، فإذا كان الناتج أكبر من عتبة معينة تدعى edgeThreshold في OpenCv فإنه يتم شطب النقطة، وبالتالي فإن النقاط الباقية هي النقاط المهمة القوية.

٣- اتجاه التعيين Orientation Assignment:

الآن يتم تعيين الاتجاه لكل من النقاط الرئيسية لتحقيق الثبات أيضا ولكن هذه المرة عند تدوير الصورة. يتم أخذ المنطقة المجأورة حول موقع النقاط الرئيسية وذلك اعتمادا على الحجم. ويتم حساب مقدار التدرج واتجاهه في تلك المنطقة. يتم انشاء مخطط بياني للتوجه له 36 بن (BIN) تشمل ٣٦٠ درجة. تأخذ قيمها بحسب مقدار الدوران والنافذة الدائرية الغأوسية الوزن مع σ تسأوي ١.٥ مرة من حجم النقاط الرئيسية. يتم أخذ أعلى قمة في المخطط البياني وأي قمة فوق 80% يتم أخذها بعين الاعتبار لحساب التوجه. إنه ينشأ نقاط أساسية بنفس الموقع والحجم ولكن باتجاهات مختلفة، وهذا يفيد في استقراريه المطابقة.

٤- واصف النقطة الرئيسية:

الآن يتم إنشاء واصف للنقطة الرئيسية. ويكون بحجم 16x16 وحول النقطة الرئيسية. يتم تقسيمه إلى ١٦ مربع فرعي بحجم 4x4. ولكل منها يتم إنشاء مخطط بياني بـ ٨ بن للدوران. لذلك يوجد ١٢٨ بن للقناع الكلي. تمثل هذه البنات (BINS) بشعاع لوصف النقطة الرئيسية. بالإضافة لذلك يتم اتخاذ عدة تدابير للتحقق من قوتها عند حدوث تغيرات في الإضاءة والدوران.

٥- مطابقة النقطة الرئيسية:

يتم مطابقة النقاط الرئيسية بين صورتين عن طريق تحديد أقرب النقاط المجأورة. ولكن في بعض الحالات قد يكون هناك مطابقتان قريبتان جدا من بعضهما. وهذا قد يحدث بسبب الضجيج أو لأسباب أخرى، وفي هذه الحالة تأخذ نسبة أقرب مسافة إلى ثاني أقرب مسافة..، فإذا كانت أكبر من ٠.٨ يتم رفض النتيجة، وهذا يزيل 90% من نتائج المطابقة الخاطئة بينما يرجع 5% من نتائج المطابقة الصحيحة.

هذا هو ملخص خوارزمية SFIT ولتفاصيل أكثر يمكنك العودة لأوراق البحث الخاصة بهذه الخوارزمية تذكر بأن هذه الخوارزمية لها حقوق نشر، لذلك هي في القسم الغير مجاني لـ OpenCv.

ملاحظة: خوارزمية **SFIT** غير مجانية لذلك سنلاحظ بأنها موجودة في اصدار **OpenCv** القديم **v2** أما في الإصدار الأحدث **v3** فهي غير موجودة.

- **SIFT في OpenCv:**

دعنا نبدأ بتعلم الوظائف المتاحة لSIFT في مكتبة OpenCv. لنبدأ مع النقاط الأساسية ونستفاد منها. في البداية علينا بناء (إنشاء) جسم SFIT، ثم نقوم بتمرير البيانات المختلفة إليه وهذا أمر اختياري، وقد تم شرح ذلك بشكل جيد في أوراق البحث الخاصة بهذه الخوارزمية.

```
import cv2
import numpy as np

img = cv2.imread('car.jpg')
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT()
kp = sift.detect(gray,None)

img=cv2.drawKeypoints(gray,kp)

cv2.imwrite('sift_keypoints.jpg',img)

cv2.imshow('image',img)
cv2.waitKey()
cv2.destroyAllWindows()
```

النتيجة:

شرح الكود:

أولاً قمنا بقراءة الصورة ثم قمنا بتحويلها لمستوى اللون الرمادي.

```
sift = cv2.SIFT()
```

ثم قمنا بإنشاء جسم خوارزمية SFIT وذلك عن طريق التابع cv2.SIFT().

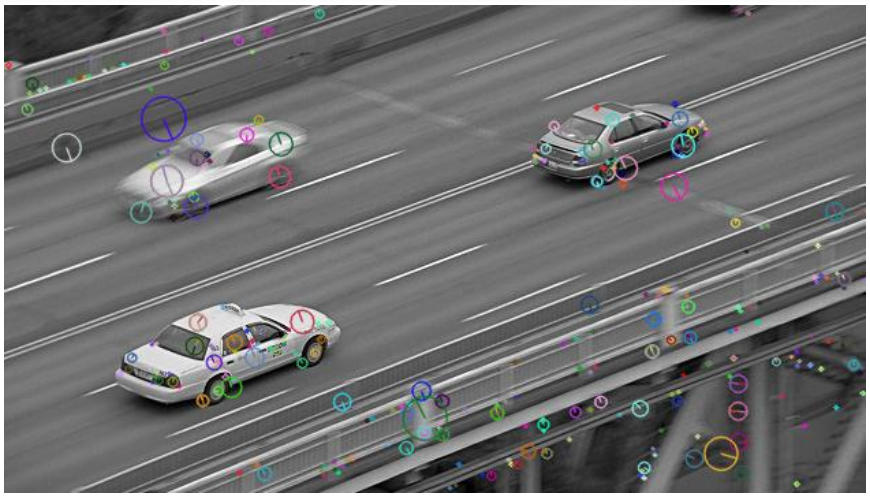
```
kp = sift.detect(gray,None)
```

التابع sift.detect() يوجد النقاط الرئيسية التي في الصورة. يمكنك تمرير قناع إذا أردت البحث عنها فقط في جزء محدد. كل نقطة رئيسية لها تركيب خاص وهي تحوي على العديد من الخصائص مثل الإحداثيات (x,y)، وحجم الجوار ذو المعنى، والزأوية التي تحدد الاتجاه، والاستجابة التي تحدد قوة النقاط الرئيسية.

```
img=cv2.drawKeypoints(gray,kp)
```

مكتبة OpenCv أيضا توفر لنا التابع cv2.drawKeyPoints() والذي يقوم برسم دوائر صغيرة في مواقع النقاط الرئيسية. إذا مررت العلم
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS لهذا التابع سوف يرسم دائرة بحجم النقطة الأساسية وسيعرض أيضا الاتجاه. شاهد المثال التالي.

```
img=cv2.drawKeypoints(gray,kp,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```



الآن لنحسب الوصف. مكتبة OpenCvتوفر لنا طريقتين للقيام بحساب الوصف:

١. بما أننا سابقا أوجدنا النقاط الرئيسية، يمكنك إذاً استدعاء التابع sift.compute() والذي يحسب الوصف من خلال النقاط الرئيسية التي أوجدناها. مثل: kp,des = sift.compute(gray,kp).

٢. إذا لم تقم بإيجاد النقاط الرئيسية من قبل، عندها يمكنك إيجاد النقاط الرئيسية والوصف في خطوة واحدة من خلال التابع sift.detectAndCompute() . وذلك كما في الكود التالي:

```
sift = cv2.SIFT()
kp, des = sift.detectAndCompute(gray,None)
```

هنا kp تمثل قائمة(مصفوفة) بالنقاط الأساسية، وdes تمثل مصفوفة من الشكل <عدد النقاط الرئيسية × ١٢٨> لذلك سنحصل على النقاط الأساسية والوصف. وفي الفصول القادمة سنتعلم كيف نطابق بين النقاط الرئيسية في صور مختلفة.

مقدمة إلى SURF (تسريع الميزات القوية) (Speeded-Up Robust Features)

مقدمة:

إن ميزة SIFT جيدة، ولكنها غير سريعة بما فيه الكفاية، لذلك تم انشاء نسخة أسرع وأطلق عليها اسم SURF

الهدف:

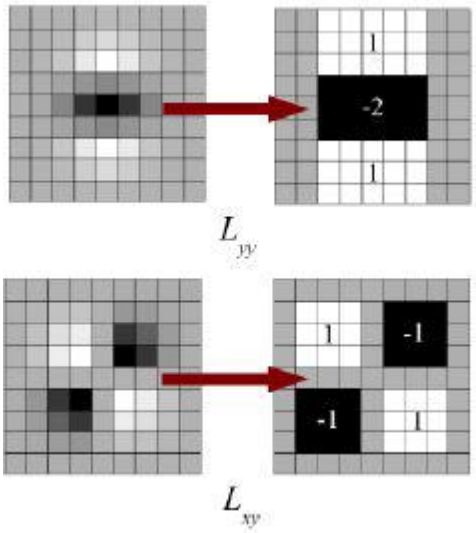
- سنتعلم أساسيات SURF.
- سنتعلم استخدام توابع SURF في OpenCv.

مقدمة نظرية:

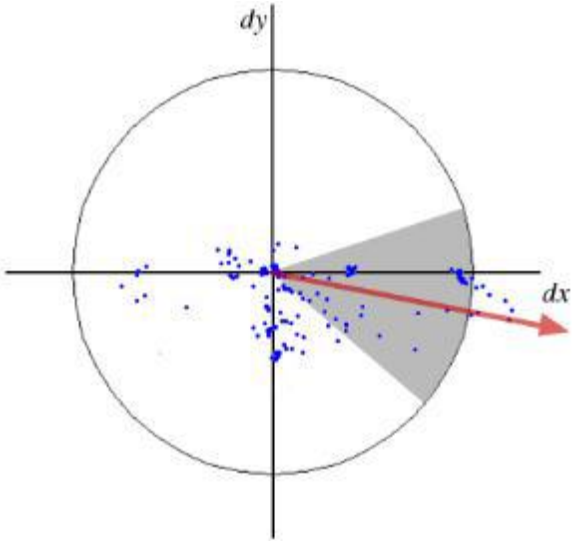
في الفصل السابق شاهدنا كيف أن خوارزمية SFIT تفيدنا في تحديد النقاط الرئيسية والوصف، ولكنها كانت بطيئة نسبياً والمستخدم بحاجة لنسخة أسرع، لذلك قام عدة مطورين بإيجاد نسخة جديدة مسرعة وأطلقوا عليها ضمن أوراق البحث الخاصة بهم اسم SURF " Speeded Up Robust Features".

في SFIT شاهدنا كيف تم تقريب لابلاسيان الغأوسي بإيجاد الفرق الغأوسي لإيجاد حجم الصورة، (للتذكير: لابلاسيان الغأوسي يعمل كمكتشف للنقاط الرئيسية بحيث يكتشف هذه النقاط حتى ولو تغير حجم الصورة) أما في SURF فتم تبسيط هذا الأمر أكثر بحساب المرشح الصندوقي.

الصورة التالية توضح هذا التقريب. الميزة الوحيدة لهذا التقريب بأنه عند القيام بطي الصورة مع المرشح الصندوقي فإنه يمكن بسهولة حسابه عن طريق مكاملة الصور. ويمكن أن يحسب بنفس الأمر من أجل أحجام مختلفة. وتعتمد SURF أيضا على العوامل المُحَدَدة لمصفوفة هيسيان Hessian لكل من الموضع والقياس.



لحساب الاتجاه، تستخدم SURF الاستجابات الموجية (wavelet responses) في كل من الاتجاهين الأفقي والعمودي وبجوار بحجم ٦ أضعاف القياس (6S)، ونطبق أيضاً الأوزان الغأوسية المناسبة على ذلك. ثم يتم رسم الناتج كنقاط في الفضاء كما هو واضح في الصورة التالية.



ويتم تقدير الاتجاه الغالب عن طريق حساب مجموع كل الاستجابات ضمن زأوية متغيرة بقيمة ٦٠ درجة.

الشيء المثير للاهتمام هو أن الاستجابة الموجية يمكن إيجادها بسهولة بالغة عند أي حجم للصورة.

في العديد من التطبيقات ثبات الدوران غير مطلوب لذلك لا حاجة لإيجاد هذا التوجه، مما يؤدي لتسريع زمن تنفيذ الخوارزمية.

SURF يوفر وظيفة تدعى Upright-SURF أو U-SURF تحسن السرعة ، وهي فعالة لحوالي $\pm 15^{\circ}$ درجة.

في مكتبة OpenCv هناك علم (بارامتر) يحدد هل نستخدمها أم لا، فإذا كانت قيمة العلم ٠ سيحسب التوجه، وإذا كانت قيمة العلم ١ لن يحسب التوجه وبالتالي سيكون أسرع.

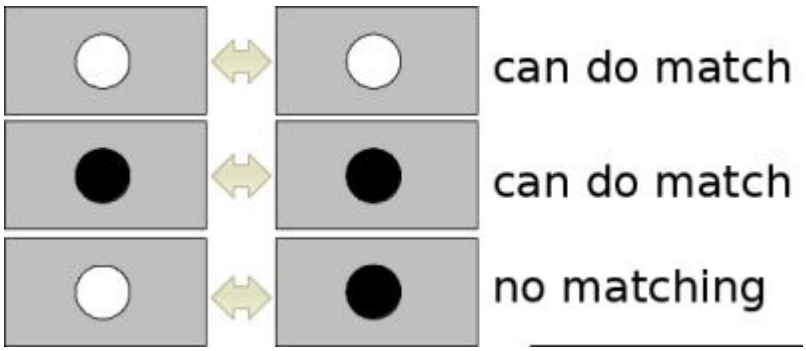
بالنسبة لوصف الميزات، SURF تستخدم الاستجابة الموجية في الاتجاه الأفقي والعمودي (نستخدم متكاملة الصورة مرة أخرى لجعل هذا يتم بسهولة).
تأخذ النقاط المجاورة حول النقاط الأساسية بالقياس 20s×20s حيث s يمثل القياس. ثم تقسم على 4×4مقطع فرعي. ثم نأخذ الاستجابة الأفقية والعمودية لكل منطقة فرعية، ونشكل متجهة (شعاع) مثل هذا

$$\boldsymbol{v} = (\sum \boldsymbol{d_x}, \sum \boldsymbol{d_y}, \sum |\boldsymbol{d_x}|, \sum |\boldsymbol{d_y}|).$$

عندما يتم تمثيلها بمتجهة، يعطي SURF ميزة وصف بـ ٦٤ بعد. ولكن كلما قللناها حصلنا على سرعة أكثر ولكن على حساب الدقة.

للحصول على دقة أكبر، SURFيملك نمط آخر للوصف بـ ١٢٨ بعد. جمع $\boldsymbol{d_x}$ و $|\boldsymbol{d_x}|$ يحسب بشكل منفصل عندما $\boldsymbol{d_y} < \boldsymbol{0}$ و$\boldsymbol{d_y} \geq \boldsymbol{0}$. وبشكل مشابه، جمع $\boldsymbol{d_y}$ و $|\boldsymbol{d_y}|$ يحسب بشكل منفصل وفقاً لإشارة $\boldsymbol{d_x}$.
وبالتالي عند مضاعفة عدد الميزات لا يضاف عمليات حسابية معقدة.
مكتبةOpenCv توفر لنا العمل على كلا النمطين وذلك بضبط قيمة العلم بين ٠ و١، بحيث العلم ٠ يوضع بالنسبة لـ ٦٤ بعد، أما العلم 0 فيوضع بالنسبة لـ ١٢٨ بعد. (عند عدم وضع أي قيمة للعلم يكون بشكل افتراضي ١٢٨).

هناك تحسين آخر وهو باستخدام إشارة لابلاسيان (أثر مصفوفة هيسيان Hessian) ... وهذا لا يزيد التعقيد الحسابي لأنه محسوب أصلاً أثناء الاكتشاف، وهذه الإشارة توضح الخلفيات العاتمة وراء الأجسام البيضاء أو بالعكس، وعند المطابقة فقط سنقارن الميزات فيما إذا كانت تملك نفس التباين (كما هو معروض في الصورة التالية). هذه المعلومات القليلة تسمح بتسريع عملية المطابقة، دون التقليل من أداء الوصف.



بكلمات مختصرة، إن SURF تعطي العديد من الميزات في كل مرحلة، حيث يظهر التحليل بأنها أسرع بحوالي ٣ أضعاف من SFIT، بينما الأداء يماثل SFIT. وSURF جيدة للتعامل مع الصور المدورة والضبابية(المشوشة)، ولكنها غير جيدة في التعامل مع التغير في ناحية الرؤية والإضاءة.

- **SURF في OpenCv:**

مكتبة OpenCv توفر وظائف لـ SURF تماما مثل SFIT.

في SURF هناك بعض الشروط خيارية مثل صفة البعد: ١٢٨/٦٤ بعد، الاتجاه: عمودي (أي بدون اتجاه) \ عادي (أي مع اتجاه). كل هذه التفاصيل شرحت ضمن أوراق البحث. ويستخدم التابعين SURF.compute()،SURF.detect() لإيجاد النقاط الرئيسية والوصف.

في الأمثلة التالية سنتعلم إيجاد النقاط الرئيسية والوصف ونرسمها:

أولا نقوم بتحميل الصورة.

```
>>> img = cv2.imread('fly.png',0)
```

ثم ننشأ جسم SURF. يمكنك تحديد البارامترات لاحقا

هنا ضبطنا عتبة مصفوفة هيسيان بـ ٤٠٠.

```
>>> surf = cv2.SURF(400)
```

نوجد النقاط الأساسية والوصف بشكل مباشر

```
>>> kp, des = surf.detectAndCompute(img,None)
>>> len(kp)
```

Out: 699

٦٩٩ نقطة عدد ضخم للرسم لذلك نزيد العتبة لنصل لـ ٥٠ نقطة، ولكن عند المطابقة سنحتج كل هذه النقاط ولكن ليس الآن.

```
>>> print surf.hessianThreshold
```

Out: 400.0

سنجعل قيمة العتبة مثلا ٥٠٠٠٠. تذكر هذا فقط لرسم النقاط على الصورة.

```
>>> surf.hessianThreshold = 50000

>>>kp, des = surf.detectAndCompute(img,None)

>>>print len(kp)
```

Out: 47

العدد مناسب أقل من ٥٠ لذلك سنرسمه على الصورة.

```
>>> img2 = cv2.drawKeypoints(img,kp,None,(255,0,0),4)
>>> plt.imshow(img2),plt.show()
```

الآن سنطبق U-SURF لذلك لن تجد هناك اتجاه. (الصورة التالية توضح ذلك). ثم نفحص علم الاتجاه ونتأكد هل هو عمودي أم لا، فإذا كانت قيمته False نجعلها TRUE (هذا يعني عدم وجود اتجاه).

```
>>> print surf.upright
```

False

```
>>> surf.upright = True
```

نعيد حساب ميزات التقاط ونرسمها.

```
>>> kp = surf.detect(img,None)
>>> img2 = cv2.drawKeypoints(img,kp,None,(255,0,0),4)
```

```
>>> plt.imshow(img2),plt.show()
```

شاهد النتيجة سترى بأن جميع الاتجاهات المعروضة هي بنفس الاتجاه. هذا يعطي سرعة في وقت التنفيذ أكثر من السابق.

أخيراً سنفحص قياس الواصف ونغيره إلى ١٢٨ بعد فقط إذا كان البعد ٦٤-بعد. ثم نوجد قياس الواصف.

```
>>> print surf.descriptorSize()
```

Out:64

هذا يعني أن قيمة علم "التمديد" قيمته هي FALSE.

```
>>> surf.extended
```

Out: FALSE

بما أن قيمة العلم FALSE فنجعلها TRUE لنحصل على وصف ب ١٢٨- بعد.

```
>>> surf.extended
>>>kp, des = surf.detectAndCompute(img,None)
>>>print surf.descriptorSize()
```

Out: 128

```
>>> print des.shape
```

لا تنسى بأن قسم المطابقة سنقوم به في فصل آخر. Out:(٦٤، ١٢٨)

خوارزمية FAST لاكتشاف الزوايا

مقدمة:

كل طرق الميزات السابقة جيدة في بعض الأحيان، ولكنها ليست سريعة بما فيه الكفاية حتى تعمل مع التطبيقات التي تعمل بالوقت الحقيقي (مثل السيارة ذاتية القيادة، وتعقب جسم في الوقت الحقيقي مثل لوحة المركبات المخالفة، .....الخ)، لذلك تم إيجاد خوارزمية FAST والتي تعمل في الوقت الحقيقي Real-time.

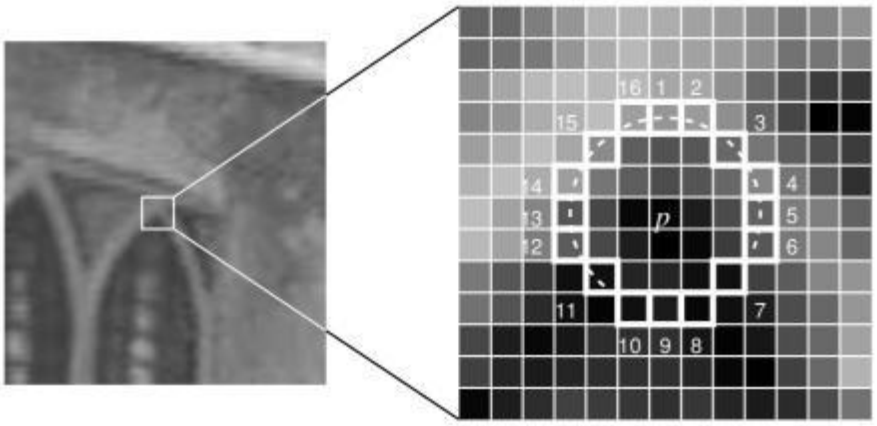
الهدف:

- سنفهم أساسيات خوارزمية FAST.
- سنوجد الزوايا باستخدام توابع OpenCv الخاصة بخوارزمية FAST.

مقدمة نظرية:

ميزة الاكتشاف باستخدام FAST:

١. اختر بيكسل p في الصورة حيث سيتم تعريفه كنقطة أساسية أو غير أساسية. ولتكن شدة البيكسل Ip.
٢. حدد قيمة عتبة مناسبة t.
٣. اعتبر أن هناك دائرة بـ ١٦ بيكسل حول البيكسل (شاهد الصورة التالية).



٤. البيكسل p يكون زأوية إذا وُجِدَ مجموعة n من البيكسلات المتجأورة في الدائرة (من الـ ١٦ بيكسل) بحيث تكون كلها أسطع من $Ip + t$ أو أعتم من $\mathbf{I_p t}$ (كما هو مبين في الصورة، ستجد خطوط متقطعة بيضاء). هنا تم اختيار n لتكون ١٢.

٥. تم اقتراح اختبار السرعة العالية high-speed test لاستبعاد عدد كبير من النقاط التي ليست بزأوية. وهذا الاختبار فقط يفحص أربعة بيكسلات وهي 13, 5, 9 , 1(أولا يفحص البيكسل ١ و٩ فيما إذا كانا ساطعين أم عاتمين. إذا كان كذلك يفحص ٥ و١٣). حتى يكون البيكسل p زأوية، يجب أن يكون ثلاثة من هذه البيكسلات على الأقل أكثر سطوعا من Ip $+ t$ أو أغمق من $\mathbf{I_p t}$ ، وإلا فلن يكون البيكسل p زأوية. هذا المكتشف بحد ذاته أدائه عالي، ولكن هناك عدة نقاط ضعف.

أول ثلاث نقاط تتحدد باستخدام تقنيات تعليم الألة. وتحل باستخدام الإخماد الا أعظمي.

الملخص:

هذه الخوارزمية أسرع بعدة مرات من مكتشفات الزوايا السابقة، ولكنها لا تتحمل مستويات عالية من الضجيج. وهذه الخوارزمية تعتمد على العتبة.

- **FAST في OpenCv:**

FAST يستدعى كأي مكتشف ميزات آخر في OpenCv. حيث يمكنك تحديد قيمةالعتبة، ويمكننا تطبيق الإخماد الغير أعظمي أو عدم تطبيقه، والجوار الواجب استخدامه، وله ثلاثة أعلام وهي:

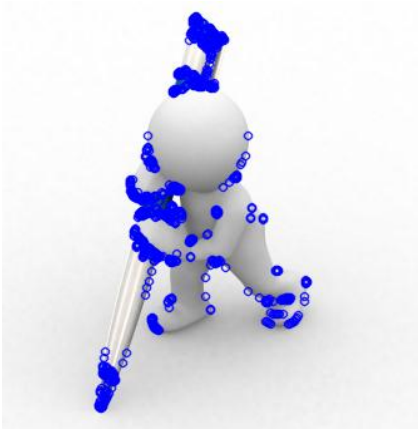
- **cv2.FAST_FEATURE_DETECTOR_TYPE_5_8**

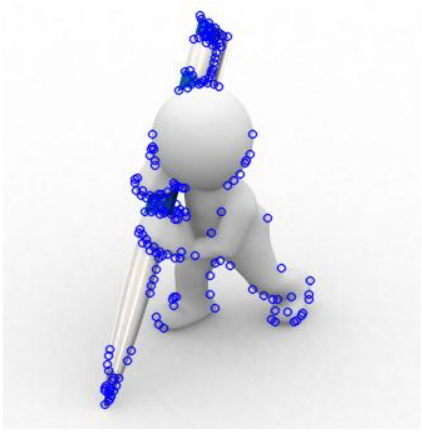
- cv2.FAST_FEATURE_DETECTOR_TYPE_7_12
- cv2.FAST_FEATURE_DETECTOR_TYPE_9_16

الكود التالي يبين كيف نكتشف ونرسم النقاط الميزة في FAST.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('Writing.jpg')

# Initiate FAST object with default values
fast = cv2.FastFeatureDetector()

# find and draw the keypoints
kp = fast.detect(img,None)
img2 = cv2.drawKeypoints(img, kp, color=(255,0,0))

# Print all default params
print "Threshold: ", fast.getInt('threshold')
print "nonmaxSuppression: ", fast.getBool('nonmaxSuppression')
#print "neighborhood: ", fast.getInt('type')
print "Total Keypoints with nonmaxSuppression: ", len(kp)


# Disable nonmaxSuppression
fast.setBool('nonmaxSuppression',0)
kp = fast.detect(img,None)

print "Total Keypoints without nonmaxSuppression: ", len(kp)
img3 = cv2.drawKeypoints(img, kp, color=(255,0,0))

cv2.imshow('max Suppression',img2)
cv2.imshow(' non max Suppression',img3)
cv2.waitKey()
cv2.destroyAllWindows()
```

شاهد النتيجة: الصورة التي على اليسار تعرض FAST مع الإخماد الغير أعظمي، بينما الصورة التي على اليمين تعرض FAST بدون الإخماد الغير أعظمي.





BRIEF (الميزات الابتدائية المستقلة القوية الثنائية)

(Binary Robust Independent Elementary Features)

مقدمة:

بسبب استهلاك الذاكرة الكبير من قبل SIFT تم إيجاد BRIEF التي توفر في الذاكرة وتعطينا سرعة أكبر بالمطابقة.

الهدف:

- في هذا الفصل سنتعلم أساسيات خوارزمية BRIEF.

مقدمة نظرية:

نحن نعلم بأن SFITتستخدم ١٢٨-بعد شعاع للوصف. لأنها تستعمل أرقام للنقاط من نوع float، تأخذ بشكل أساسي ٥١٢ بايت. بشكل مشابه SURF أيضا تأخذ أقل شيء ٢٥٦ بايت (من أجل ٦٤-بعد). إنشاء مثل هذه الشعاع لآلاف الميزات يستهلك الكثير من الذاكرة وهذا شيء غير عملي وخاصة في الأنظمة المدمجة والتي تكون ذاكرتها محدودة. وكلما كانت الذاكرة التي استهلكناها أكبر عندها عملية المطابقة ستأخذ وقت أطول.

ولكن كل هذه الأبعاد قد لا يكون هناك حاجة إليها عند المطابقة الفعلية. لذلك يمكننا أن نضغطها بعدة طرق مثل PCA وLDA. وهناك طرق أخرى مثل التجزئة باستخدام LSH (التجزئة الحساسة المحلية) وتستخدم لتحويل صفات SFITمن أرقام للنقاط من نوع floatإلى سلاسل ثنائية binary strings. هذه السلاسل الثنائية تستخدم لمطابقة الميزات باستخدام مسافة المبالغة Hamming distance. وهذا يعطينا سرعة أفضل، لأن إيجاد المسافة المبالغة يستعمل فيه فقط XOR وعداد للبيت bit count، والتي تكون سريعة جدا في المعالجات الحديثة مع تعليمات SSE. ولكن هنا سنحتاج لإيجاد الأوصاف في البداية، وفقط عندئذ يمكننا تطبيق عملية التجزئة، وهذا لا يؤدي لحل مشكلة استهلاك الذاكرة التي واجهتنا في الأول.

BRIEF تزودنا باختصار لإيجاد السلسلة الثنائية بشكل مباشر بدون إيجاد الأوصاف. يأخذ صورة مهذبة للتصحيح ويختار مجوعة من nd (x,y) أزواج الموقع بطريقة فريدة من نوعها (وهذا الأمر موضح ضمن أوراق البحث الخاصة بخوارزمية BRIEF). ثم تتم مقارنة شدة بعض البيكسلات عند هذه الأزواج. على سبيل المثال، ليكن في البداية موضع الأزواج p وq. إذا كانت شدة البيكسل p أكبر من شدة البيكسل q عندها النتيجة ستكون ١، وغير ذلك ستكون النتيجة ٠. وهذا يطبق على كل مواضع الأزواج nd لنحصل على أبعاد nd سلسلة البت.

ويوفر هذه الـ ndيمكن أن تكون ١٢٨ أو ٢٥٦ أو ٥١٢، ومكتبة OpenCvتدعم كل هذا، ولكن بشكل افتراضي سوف تكون ٢٥٦ (مكتبة OpenCv تمثل ذلك في بايتات، لذلك القيمة ستكون ١٦ و٣٢ و٦٤). حتى عندما تحصل على هذا يمكنك استخدام المسافة المبالغة لمطابقة هذه الأوصاف. هناك نقطة هامة وهي بأن BRIEF واصف الميزات، لا يزودنا بأي طريقة لإيجاد الميزات، لذلك يجب أن نستعمل أي مكتشف للميزات مثل SFIT,SURF.

بكلمات مختصرة: إن BRIEF هو أسرع طريقة لحساب ميزة الوصف والمطابقة. ويوفر لنا نسبة تمييز وتعرف عالية مالم يكن الجسم تم تدويره بشكل كبير.

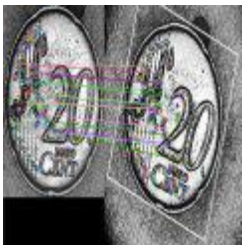
- **BRIEF في OpenCv:**

الكود التالي يوضح حساب الأوصاف ل BRIEF بمساعدة كاشف الأوصاف CenSurE (يدعى الكاشف CenSurE بكاشف النجمة STAR في OpenCv).

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('car.jpg')

# Initiate STAR detector
star = cv2.FeatureDetector_create("STAR")

# Initiate BRIEF extractor
brief = cv2.DescriptorExtractor_create("BRIEF")

# find the keypoints with STAR
kp = star.detect(img,None)

# compute the descriptors with BRIEF
kp, des = brief.compute(img, kp)

print brief.getInt('bytes')
print des.shape

img2 = cv2.drawKeypoints(img, kp, color=(255,0,0))

cv2.imshow('BRIEF ',img2)
cv2.waitKey()
cv2.destroyAllWindows()
```

التابع brief.getInt('bytes') يعطي قياس nd المستعمل في البايت. بشكل افتراضي يكون ٣٢.
المرحلة التالية التي سنقوم بها هي المطابقة والتي سنتنأولها في الفصول القادمة.

ORB (موجه FAST ومدور BRIEF) (Oriented FAST and Rotated BRIEF)



مقدمة:

SIFT و SURF يقومون بعمل جيد، ولكن المشكلة بأنهما غير مجانيتان ويجب دفع بضع دولارات كل سنة. لحل هذه المشكلة وفرت لنا مكتبة OpenCvبديل عن SIFT و SURFوهو ORB.

الهدف:

- في هذا الفصل سنتعلم أساسيات ORB.

معلومات نظرية:

ORB بشكل أساسي عبارة عن دمج لمكتشف النقاط الرئيسية FAST مع الواصف BRIEF مع العديد من التعديلات لتحسين الأداء. في البداية تستعمل FAST لإيجاد النقاط الرئيسية ثم يتم تطبيق مقياس زوايا هاريس للعثور على أفضل النقاط N بينهم. كم أنها تستخدم الهرم للحصول على الميزات(الزوايا) مع أحجام مختلفة للصورة. ولكن هناك مشكلة واحدة وهي أن FAST لا تحسب التوجه. فماذا نفعل إذا لنحصل على ميزات ثابتة حتى ولو قمنا بتدوير الصورة (ثبات في الدوران)؟

قام المطور بالتعديل التالي، حيث قام بحساب كثافة الوزن المتوسط للبقعة مع الزأوية الحالية عند المركز. اتجاه الشعاع من نقطة الزأوية إلى المركز يعطي التوجه. لتحسين ثبات الدوران، نحسب العزوم بالنسبة للمحورين x وy والتي يجب أن تكون في منطقة دائرية نصف قطرها r، حيث r هو حجم البقعة.

الآن بالنسبة للوصف فإن ORB تستخدم الواصف BRIEF. ولكن كما شاهدنا في الفقرة السابقة فإن BRIEF أداءه ضعيف عند دوران الصورة. لذلك ما ستفعله ORB هي توجيه"steer" BRIEF

وفقا لتوجهات النقاط الرئيسية. للحصول على أي ميزة لمجموعة n من الاختبارات الثنائية binary test عند الموضع (x_i, y_i)، نعرف مصفوفة بحجم 2×n. S تحوي على إحداثيات هذه البيكسلات. ثم باستخدام التوجيه لهذه البقعة،θ ، يتم إيجاد دوران المصفوفة والإحداثيات S للدوران، لنحصل على التوجيه (الدوران) صيغة $S\theta$.

ORB تقدر مقدار زيادة الزأوية بـ $\frac{2\pi}{30}$(١٢ درجة) ، وينشأ جدول بحث من نماذج BRIEF المحسوبة سابقا. طالما أن توجه النقاط الرئيسية θ متناسقة عند العرض، ومجموعة نقاط التصحيح $S\theta$ سوف تستخدم لحساب الوصف.

BRIEFيملك خاصية هامة وهي أن كل ميزة بت لديها تباين (فرق) كبير ومتوسط تقريبا ٠.٥. ولكن بمجرد أنها موجهة جنبا إلى جنب مع أبعاد النقاط الأساسية، تفقد هذه الخاصية وتصبح أكثر توزيعاً. التباين العالي يجعل الميزة أكثر وضوحاً، لأن استجابته تتغير بشكل مختلفا تبعا للمدخلات.

خاصية أخرى مرغوبة وهي أن تكون الاختبارات غير مترابطة بحيث كل اختبار يساهم في النتيجة. لحل كل هذه، ORB يقوم بتشغيل البحث الشجع بين جميع الاختبارات الثنائية الممكنة لإيجاد اختبار ثنائي واحد يملك تباين عالي وتكون قريبة من ٠.٥ بالإضافة إلا كونها غير مترابطة، وهذه النتيجة تدعى rBRIEF.

بالنسبة لواصف المطابقة، LSH متعددة الاختبار تحسن من LSH التقليدي، وتستخدم. أوراق البحث تقول إن ORB أسرع بكثير من SURF وSIFT، وواصف ORB يعمل بشكل أفضل من SURF. ORB هو خيار جيد في الأجهزة المنخفضة الاستهلاك للطاقة.

- **ORB في OpenCv:**

كما في العادة يجب أولا إنشاء الجسم لذا سننشأ جسم ORB من خلال التابع () cv2.ORB أو الواجهة الشائعة feature2d. هذا التابع يحوي على عدد من البارامترات الاختيارية.

- nFeature: (عدد الميزات) أكثر بارامتر مفيد وهو يدل على أكبر عدد من الميزات التي سيتم الاحتفاظ بها (افتراضيا تكون ٥٠٠).
- scoreType: (نوع النقاط) يدل على ما إذا كان محصلة النقاط لهاريس أو FAST لتصنيف الميزات (افتراضيا تكون محصلة النقاط لهاريس).
- WTA_K : لتحديد عدد النقاط التي تعطي توجيه كل عنصر من عناصر الواصف BRIEF. (افتراضيا تكون نقطتين أي يختار نقطتين في كل مرة).
- NORM_HAMMING: المسافة بين النقاط التي تستخدم من أجل المطابقة، فإذا كانت WTA_K ٣ أو ٤، هذا يعني بأنها ٣ أو ٤ نقاط لإنتاج الواصف BRIEF. ثم يتم تعريف مسافة المطابقة بـ NORM_HAMMING2.

الكود التالي يوضح استخدام ORB.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('car.jpg')

# Initiate STAR detector
orb = cv2.ORB()

# find the keypoints with ORB
kp = orb.detect(img,None)

# compute the descriptors with ORB
kp, des = orb.compute(img, kp)

# draw only keypoints location,not size and orientation
img2 = cv2.drawKeypoints(img,kp,color=(0,255,0), flags=0)


cv2.imshow('ORB ',img2)
cv2.waitKey()
cv2.destroyAllWindows()
```

مراجع إضافية:

- Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary R. Bradski: ORB: An efficient alternative to SIFT or SURF. ICCV 2011: 2564-2571.

تطابق الميزات Feature Matching

مقدمة:

لقد تعلمنا كثيرا عن ميزة الكشف والوصف، والآن حان الوقت لنتعلم كيف نطابق مختلف الأوصاف. مكتبة OpenCv توفر لنا تقنيتين لهذا الأمر وهما:

مطابقة Brute-Force ومطابقة FLANN based.

الهدف:

- سنتعلم كيف سنطابق الميزات بين صورتين.
- سنستخدم مطابقة Brute-Force ومطابقة FLANN based.

أساسيات المطابقة Brute-Force:

المطابقة Brute-Force هي مطابقة بسيطة، فهي تأخذ الواصف لميزة واحدة في المجموعة الأولى وتطابقها مع ميزات أخرى في مجموعة ثانية بإجراء بعض الحسابات على المسافة، وأقرب واحدة يتم اخذها (أعادتها return).

بالنسبة للمطابقة BF، يجب علينا في البداية إنشاء جسم مطابقة BF باستخدام التابع cv2.BFMatcher(). هذا التابع يأخذ بارامترين اختياريين، البارامتر الأول هو normType، وظيفته تحديد قياس المسافة التي سيتم استخدامها (بشكل افتراضي تكون cv2.NORM_L2)، هذا البارامتر جيد بالنسبة ل SFIT وSURF أما بالنسبة للسلسلة الثنائية التي تبني الواصف مثل ORB,BRIEF,RBISK ينبغي أن نمرر القيمةcv2.NORM_HAMMING، والتي تستخدم

المسافة المبالغة للقياس. إذا ORB استخدمت VTA_K ==3 or 4 عندها يجب أن نمرر القيمة cv2.NORM_HAMMING2.

ثاني بارامتر هو المتغير المنطقي crossCheck وتكون قيمته بشكل افتراضيFALSE . إذا جعلنا قيمته TRUE عندها المطابق سيعيد فقط هذه المطابقة مع القيمة (i,j) من النوع الواصف i-th في المجموعة A، والواصف j-th في المجموعة B كأفضل مطابقة، والعكس بالعكس. إن الميزتين في كلا المجموعتين يجب أن تتطابق مع الأخرى.

بعد إنشاء الجسم، هناك طريقتان هامتان هما BFMatcher.match() and BFMatcher.knnMatch()، أول طريقة تعيد أفضل مطابقة، وثاني طريقة تعيد k مطابقة أقل، حيث k تتحدد من قبل المستخدم. يمكن أن تكون مفيدة عندما نحتاج بأن نقوم بعمل إضافي على ذلك.

كما كنا نستخدم التابع cv2.drawKeypoints() لرسم النقاط، سنستخدم التابع cv2.drawMatches() الذي يساعدنا على رسم المطابقات. فهو يضع الصورتين بشكل أفقي ويرسم الخطوط من أول صورة لثاني صورة ليظهر أفضل المطابقات. هناك أيضا التابع cv2.drawMatchesKnn والذي يرسم كل K لأفضل المطابقات. إذا k=2 سوف يرسم خطين مطابقة لكل نقطة أساسية. لذلك يجب أن نمرر قناع إذا أردنا أن نرسم بشكل انتقائي.

دعنا نأخذ مثال واحد على كل من SURF وORB (كل واحد منهم يستعمل قياسات مسافة مختلفة).

ملاحظة: في هذ البحث قم بتنصيب **opencv3.0** وضع بدل التابع **ORB ()** التابع **ORB_create ()**.
التابع **drawMatches** غير موجود في مكتبة **OpenCv** الإصدار ٢.

لذلك إذا أردت استعمال الإصدار ٢ من OpenCv قم بكتابة التابع drawMatches بشكل يدوي كما هو مشروح في الرابط التالي:

https://www.codementor.io/tips/5193438072/module-object-has-no-attribute-drawmatches-opencv-python

مطابقة القوة المسيطرة مع الواصف ORB:

هنا سنشاهد أبسط مثال وهو كيف نطابق الميزات بين صورتين. في هذه الحالة. لدينا queryImage وtrainImage. سنحأول إيجاد queryImage في trainImage باستخدام ميزة المطابقة. (الصور هي box.png و box_in_scene.png).

(**queryimage** هي صورة الجسم المطلوب، و**trainImage** هي الصورة التي سنبحث ضمنها عن الجسم المطلوب).

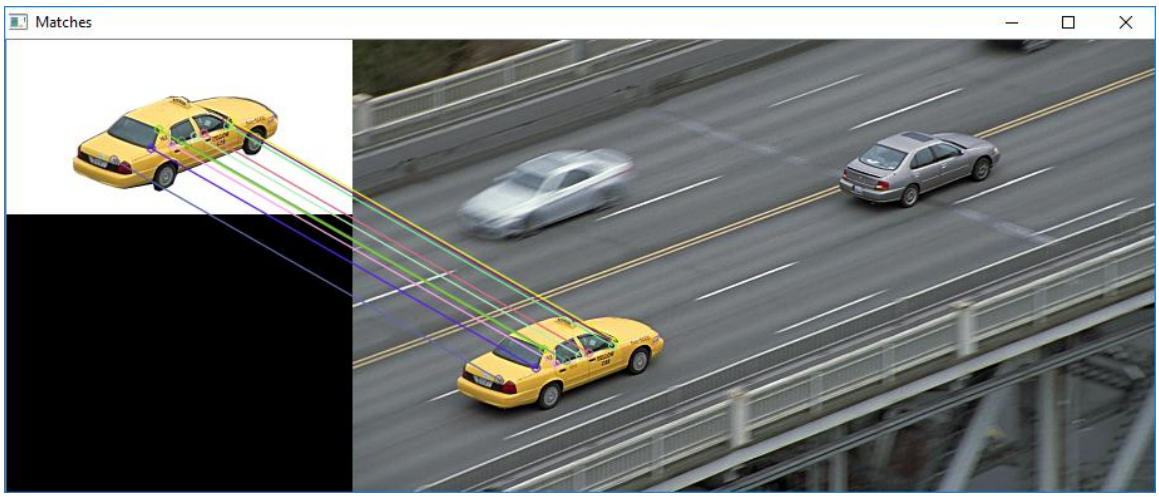
سنستخدم الواصف SFIT لمطابقة الميزات، لذلك دعنا نبدأ بتحميل الصورة وإيجاد الأوصاف.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img1 = cv2.imread('carr.png') # queryImage
img2 = cv2.imread('car.jpg') # trainImage

# Initiate SIFT detector
orb = cv2.ORB_create()

# find the keypoints and descriptors with SIFT
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)
```

ثم سنقوم بإنشاء جسم مطابقة BF مع مسافة قياس cv2.NORM_HAMMING (لأننا نستعمل ORB) و نقوم بتفعيل crossCheck للحصول على نتائج أفضل، ثم نستعمل طريقةmatch() لنحصل على أفضل مطابقة بين لصورتين. ثم نقوم بترتيب مسافاتها بشكل تصاعدي لأن المطابقة ستصبح أفضل . ثم سنرسم أول عشر مطابقات (فقط من اجل وضوح التطابق في الرسم). يمكنك زيادة عدد المطابقات كما تريد.

```
# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches[:10],None, flags=2)

cv2.imshow('Matches ',img3)
cv2.waitKey()
cv2.destroyAllWindows()
```

شاهد النتيجة التي حصلنا عليها:



ما هو جسم المطابقة؟

الناتج من matches = bf.match(des1,des2) خط من قائمة الأجسام DMatch.

الجسم DMatch يملك الصفات التالية:

١. DMatch.distance: المسافة بين نقاط الوصف، كلما كانت أقل كلما كان أفضل.
٢. DMatch.trainIdx: قائمة بنقاط الوصف لمطابقة الصورة train (الصورة التي سنبحث فيها عن الجسم).
٣. DMatch.queryIdx: قائمة بنقاط الوصف لمطابقة الصورة query (صورة الجسم المطلوب).
٤. DMatch.imgIdx: قائمة بنقاط الوصف للصورة img ...

- **مطابقة FLANN based:**

مطابقة FLANN تستند على مكتبة FAST لتقريب النقاط المجأورة، وتحوي على مجموعة من الخوارزميات المثالية لتسريع البحث عن النقطة الأقرب في قواعد البيانات الكبيرة، والميزات والأبعاد الثانوية. هذه المطابقة تعمل بسرعة أكبر من BFMatcher بالنسبة لقواعد البيانات الكبيرة.

شاهد المثال التالي الذي سنستعمل فيه المطابقة FLANN based.

بالنسبة لمطابقة FLANN based نحن بحاجة لتمرير مترجمين والذين يحددان الخوارزمية التي يجب أن تستعمل. بارومتراته مترابطة مع بعضها. أول بارامتر هو IndexParams. بالنسبة لمختلف الخوارزميات، يتم شرح المعلومات التي يتم تمريرها في ورقة البحث FLANN.

ملخص:

بالنسبة لخوارزميات مثل SFIT وSURF يتم تمرير القيم التالية:

```
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
```

أما بالنسبة لخوارزمية مثل ORB نمرر القيم التالية (يفضل أن تمرر القيم الموجودة في ورقة البحث. ولكن في بعض الأحيان قد لا تعطينا النتيجة المرغوبة لذلك سنمرر قيم أخرى تعمل بشكل جيد).

```
index_params= dict(algorithm = FLANN_INDEX_LSH,
                   table_number = 6, # 12
                   key_size = 12, # 20
                   multi_probe_level = 1) #2
```

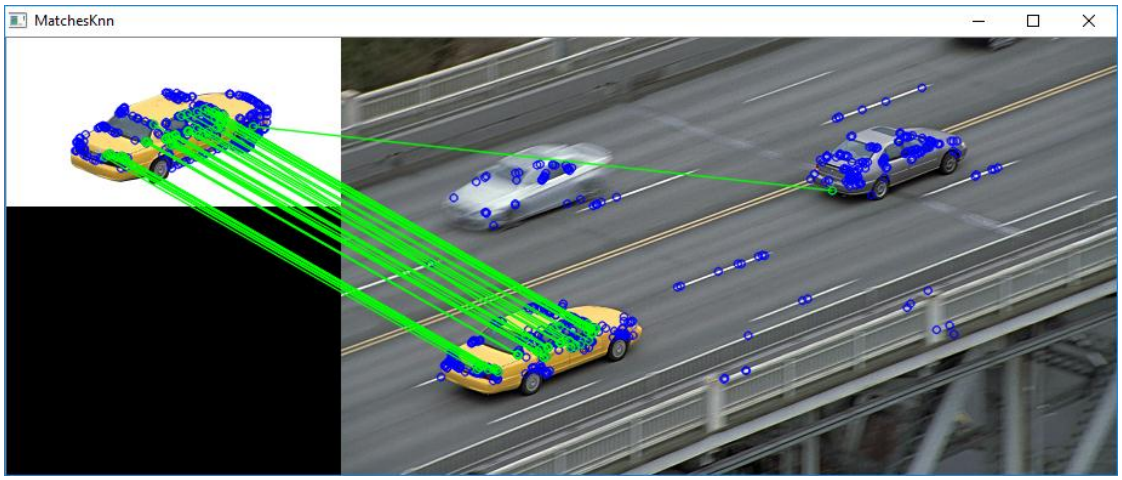
ثاني مترجم هو SearchParams، هذا المترجم يحدد عدد مرات شجرة الفهرس التي ينبغي اجتياز تكرارها. أعلى القيم تعطي دقة أفضل، ولكن يأخذ أيضا المزيد من الوقت. إذا كنت تريد تغير القيمة مرر:

```
search_params = dict(checks=100)
```

مع هذه المعلومات نحن على ما يرام.

```
import numpy as np
import cv2

img1 = cv2.imread('carr.png') # queryImage
img2 = cv2.imread('car.jpg') # trainImage

# Initiate SIFT detector
orb = cv2.ORB_create()

# find the keypoints and descriptors with SIFT
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

# FLANN parameters
FLANN_INDEX_LSH = 6
#help(dict)
index_params= dict(algorithm = FLANN_INDEX_LSH,
                   table_number = 6, # 12
                   key_size = 12, # 20
                   multi_probe_level = 1) #2

search_params = dict(checks=50) # or pass empty dictionary

flann = cv2.FlannBasedMatcher(index_params,search_params)

matches = flann.knnMatch(des1,des2,k=2)

# Need to draw only good matches, so create a mask
matchesMask = [[0,0] for i in xrange(len(matches))]

# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        matchesMask[i]=[1,0]

draw_params = dict(matchColor = (0,255,0),
    singlePointColor = (255,0,0),
    matchesMask = matchesMask,
    flags = 0)

img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,matches,None,**draw_params)

cv2.imshow('MatchesKnn ',img3)
cv2.waitKey()
cv2.destroyAllWindows()
```

النتيجة:



شرح الكود:

الأسطر الأولى في الكود تم شرحها مسبقاً، حيث قمنا بإنشاء جسم المطابقة ORB ثم قمنا بإيجاد النقاط الأساسية والوصف لكل من الصورتين.

```
# FLANN parameters
FLANN_INDEX_LSH = 6
index_params= dict(algorithm = FLANN_INDEX_LSH,
                   table_number = 6, # 12
                   key_size = 12, # 20
                   multi_probe_level = 1) #2
```

index_params يمثل قيمة البارامتر الأول للتابع FlannBasedMatcher()، وظيفته إنشاء فهرس للبحث عن البيانات الأولية لكل نقطة. إن إنشاء مؤشر (فهرس) لكل نقطة سيسهل علينا عملية البحث.

المتغير الأول للفهرس هو algorithm وظيفته تحديد الخوارزمية المستعملة لبناء الفهرس، ويأخذ أحد لقيم التالية: 'linear', 'kdtree', 'kmeans', 'composite' or 'autotuned'.

- الخيار linear لا ينشأ أي فهرس، وإنما يستخدم المطابقة BF للبحث عن النقاط الأساسية.
- الخيار kdtree ينشأ فهرس واحد أو أكثر بشكل مخطط شجري ويتم ذلك بشكل عشوائي.
- الخيار kmeans ينشأ فهرس على شكل تسلسل هرمي. باقي الخيارات مشروحة في ورقة البحث FLANN. ضمن الفقرة 3.3.1

- والمتغير الثاني هو table_number عدد أقسام الجدأول التي سنستعملها (عادة تكون بين ١٠ و٣٠).
- المتغير الثالث key_size يمثل قياس مفتاح التقسيم بالبتات (bit).

للاطلاع على شرح هذه المتغيرات انتقل لموقع لويب الخاص بشرح توابع OpenCV

http://docs.opencv.org

```
search_params = dict(checks=50) # or pass empty dictionary
```

، هذا السطر يحدد عدد مرات شجرة الفهرس التي ينبغي اجتياز تكرارها. كلما كانت القيمة أعلى كلما كانت الدقة أفضل، ولكن ستأخذ أيضا المزيد من الوقت. لذلك قم بزيادتها فقط عند الحاجة لذلك.

```
flann = cv2.FlannBasedMatcher(index_params,search_params)
```

التابع المسؤل عن تحديد معايير المطابقة. نمرر لهذا التابع القيم التي عرفناها وهما index_params وsearch_params.

```
matches = flann.knnMatch(des1,des2,k=2)
```

إيجاد أفضل مطابقة بين كل واصف للصورتيين.

ميزة المطابقة + Homography لإيجاد الأجسام

مقدمة:

الآن أصبحنا نعرف ما هي مطابقة الميزات، دعنا الآن ندمجها مع النمط calib3d لإيجاد الأجسام في الصورة المعقدة.

الهدف:

- سندمج بين ميزة المطابقة وfindHomography من النمط calib3d لإيجاد الأجسام المعروفة في الصورة المعقدة.

أساسيات:

نفعل ما فعلناه في الفصل السابق. نوجد بعض النقاط المميزة في الصورة query، ونوجد أيضاً النقاط المميزة في الصورة train، ونوجد أفضل مطابقة بينهم. بكلمات مختصرة سنوجد بعض أجزاء جسم ما في صورة أخرى مشوشة. هذه المعلومات غير كافية لإيجاد الجسم بالضبط في الصورة query.

لذلك سنستخدم النمط calib3d أي cv2.findHomography(). إذا مررنا مجموعة من النقاط من كلا الصورتين، سوف يوجد التحويل مبتغاه في ذلك الجسم. بعد ذلك يمكننا استخدام التابع cv2.perspectiveTransform() لإيجاد الجسم. إنها تحتاج فهرس بأربع نقاط صحيحة لإيجاد هذا التحول.

لقد شاهدنا كيف أنه يمكن أن يحصل بعض الأخطاء المحتملة عند عملية المطابقة والتي قد تؤثر على النتيجة. لحل هذه المشكلة نستخدم خوارزمية RANSAC أو LEAST_MEDIAN (وذلك عن طريق الأعلام). حيث نسمي المطابقة الجيدة التي تقدم تقديرات صحيحة inliers والمتبقية تسمى بـ outliers.

التابع cv2.findHomography() يعيد القناع الذي يحدد هل النقاط inliers أم outliers. دعونا نقم بذلك!!

الكود:

نوجد في البداية كما اعتدنا سمة SFIT في الصورة ونطبق نسبة الاختبار (ratio test) لإيجاد أفضل مطابقة.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

MIN_MATCH_COUNT = 10

img1 = cv2.imread('box.png',0) # queryImage
img2 = cv2.imread('box_in_scene.png',0) # trainImage

# Initiate SIFT detector
sift = cv2.SIFT()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)

flann = cv2.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)

# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
```

الآن نضبط على الأقل شروط أربع مطابقات (تُحَدَّدْ من قبل العلم MIN_MATCH_COUNT) يجب أن تكون موجودة لإيجاد الجسم. وإلا بكل بساطة نعرض رسالة تقول إن المطابقات الموجودة حاليا غير كافية.

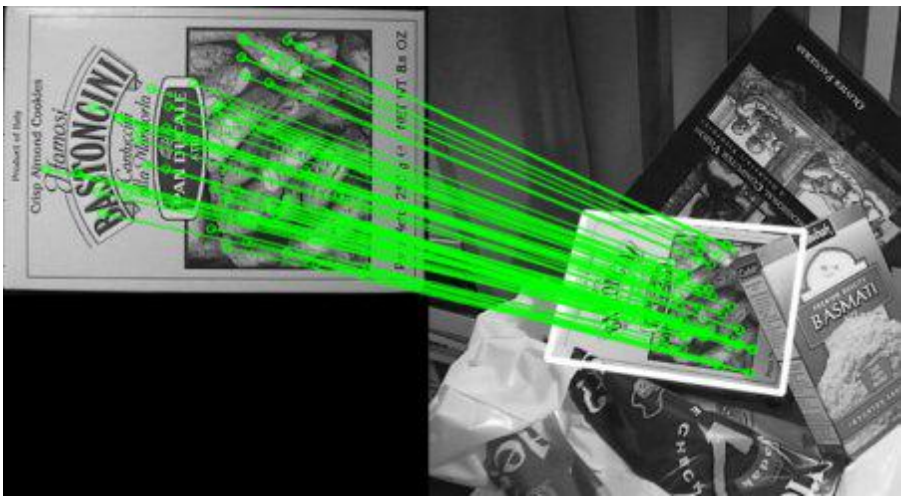
إذا تم العثور على ما يكفي من المطابقات نستخرج مواضع النقاط الأساسية للمطابقات في كلا الصورتين. التي تم تمريرها ليجد التحويل مبتغاه. حالما نحصل على مصفوفة التحويل 3×3، نستخدمه لتحويل الزوايا من الصورة query إلى ما يقابلها من النقاط في الصورة train. ثم نرسمها.

```
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)

    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()

    h,w = img1.shape
    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
    dst = cv2.perspectiveTransform(pts,M)

    img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE_AA)
else:
    print "Not enough matches are found - %d/%d" % (len(good),MIN_MATCH_COUNT)
    matchesMask = None
```

وأخيرا نرسم inliers (إذا تم بنجاح إيجاد الجسم) أو نقاط المطابقة الأساسية (إذا فشل في إيجاد الجسم)

```
draw_params = dict(matchColor = (0,255,0), # draw matches in green color
                   singlePointColor = None,
                   matchesMask = matchesMask, # draw only inliers
                   flags = 2)

img3 = cv2.drawMatches(img1,kp1,img2,kp2,good,None,**draw_params)

plt.imshow(img3, 'gray'),plt.show()
```

شاهد النتيجة التالية:



الفصل الثامن

”الرجل الذي يستخدم مهاراته وخياله البناء لمعرفة أقصى ما يمكن أن يقدمه مقابل دولار واحد بدلاً من التفكير في أقل ما يمكن أي يقدمه مقابل نفس الدولار، حتماً سينجح“

هنري فورد – مؤسس شركة فورد للسيارات

الفصل الثامن: تحليل الفيديو Video Analysis

خوارزمية Meanshift وCamshift:

شاهدنا في الأمثلة السابقة طريقة التعقب اعتماداً على اللون، وهذه الطريقة بسيطة جداً. أما الآن سنشاهد خوارزمية أفضل مثل Meanshift والنسخة المطورة منها Camshiftلإيجاد اللون وتعقبه.

التدفق البصري Optical Flow:

الآن سنناقش مفهوم هام وهو "Optical Flow " (التدفق البصري) والذي له علاقة مع الفيديو وتطبيقات عديدة.

طرح الخلفية Background Subtraction:

نحتاج في عدة تطبيقات أن نستخرج الخلفية لتنفيذ عدة عمليات مثل تعقب كائن.

خوارزمية Meanshift وCamshift:

مقدمة:

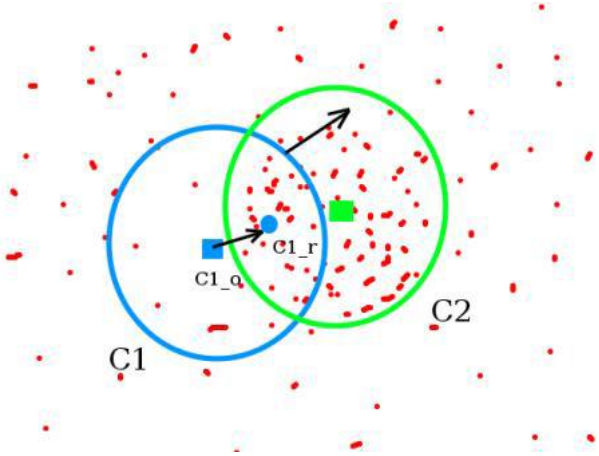
شاهدنا في الأمثلة السابقة طريقة التعقب اعتماداً على اللون، وهذه الطريقة بسيطة جداً. أما الآن سنشاهد خوارزمية أفضل مثل Meanshift والنسخة المطورة منها Camshiftلإيجاد اللون وتعقبه.

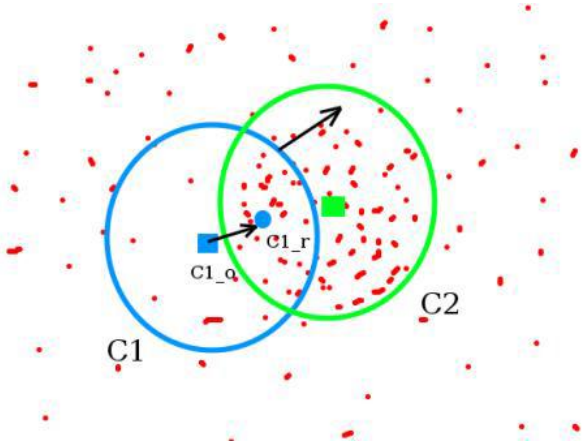
الهدف:

- سنتعلم طريقة إيجاد الأجسام في الفيديو باستخدام خوارزميتي Meanshift و Camshift

- **خوارزمية Meanshift:**

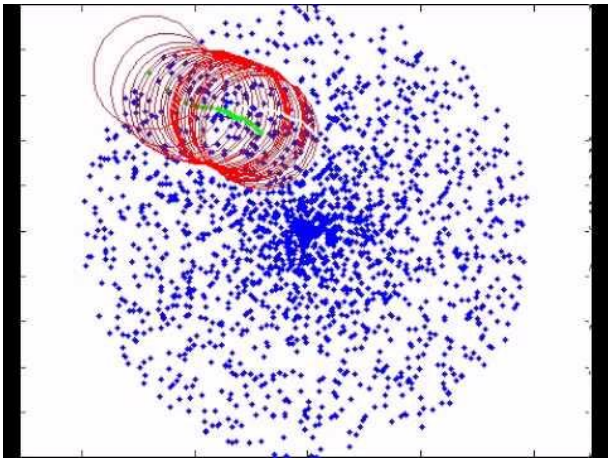
اعتبر بأنك تملك مجموعة من النقاط (توزع البيكسلات لهذه النقاط يكون كما في الاسقاط الخلفي للمخطط البياني). نقوم بإنشاء نافذة صغيرة (دائرة) ويجب علينا تحريك هذه النافذة إلى المنطقة التي تكون فيها شدة البيكسلات أعظمية (المنطقة التي يكون فيها عدد النقاط أعظمي)، وهذا الأمر موضح في الصورة التالية:





النافذة الأولية تُعرض على شكل نافذة دائرية لونها أزرق اسمها "C1". يتم وضع علامة في مركزها الأصلي على شكل مستطيل باللون الأزرق لنسمه "C1_o". ولكن إذا أوجدنا مركز كتلة النقاط داخل هذه النافذة سوف نحصل على نقطة أسمها "C1_r" (معلمة بدائرة زرقاء صغيرة) والتي تمثل المركز الحقيقي لهذه النافذة.

من المؤكد أن مركز نافذة الدائرة الأصلي لا يتطابق مع مركز كتلة النقاط (المركز الحقيقي) لذلك يتم تحريك النافذة الدائرية وجعل مركزها الأصلي يتطابق مع المركز الحقيقي، ثم نقوم مرة أخرى بإيجاد النقطة المركزية الجديدة والمركز الحقيقي. في الغالب لن يتطابق المركز الأصلي للنافذة الدائرية مع المركز الحقيقي، لذلك نحرك النافذة الدائرية مرة أخرى ليطابق مركزها مع المركز الحقيقي، ونستمر في تكرار هذا الأمر حتى يصبح مركز النافذة الدائرية منطبقا مع المركز الحقيقي (مع أصغر خطأ مقبول). لذلك أخيراً ما نحصل عليه هو نافذة فيها توزع البيكسلات أعظمي. هذه النافذة الأخيرة معلمة بدائرة باللون الأخضر أسمها "C2"، بمشاهدة الصورة السابقة ستجد أن هذه الدائرة تحوي أكبر عدد من النقاط.



Meanshift في OpenCv:

لاستعمال خوارزمية Meanshift في OpenCv، سنحتاج في البداية إلى تهيئة الهدف، وإيجاد مخططه البياني لنتمكن من القيام بعملية الإسقاط الخلفي (backproject) للهدف على كل إطار frameلحساب Meanshift. وسنكون أيضا بحاجة لتوفير الموقع الأصلي للنافذة. بالنسبة للمخطط البياني نأخذ فقط التدرج اللوني. وأيضا لتجنب القيم الخاطئة بسبب ضعف الإضاءة نقوم بإهمال قيم الإضاءة المنخفضة عن طريق التابع cv2.inRange().

```
import numpy as np

import cv2

cap = cv2.VideoCapture('slow.flv')

# take first frame of the video
ret,frame = cap.read()

# setup initial location of window
r,h,c,w = ٥٠,١٠٠,٢٥,١٠٠ # simply hardcoded the values
track_window = (c,r,w,h)

# set up the ROI for tracking
roi = frame[r:r+h, c:c+w]
hsv_roi = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((0., 60.,32.)), np.array((180.,255.,255.)))
roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)

# Setup the termination criteria, either 10 iteration or move by atleast 1 pt
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )
while(1):
    ret ,frame = cap.read()

    if ret == True:
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

        # apply meanshift to get the new location
        ret, track_window = cv2.meanShift(dst, track_window, term_crit)

        # Draw it on image
        x,y,w,h = track_window
        cv2.rectangle(frame, (x,y), (x+w,y+h), 255,2)
        cv2.imshow('img2',frame)
```

```
    k = cv2.waitKey(60) & 0xff
    if k == 27:
        break
    else:
        cv2.imwrite(chr(k)+".jpg",frame)
else:
    break

cv2.destroyAllWindows()
cap.release()
```

شرح الكود:

انصحك قبل قراءة شرح هذا الكود أن تقرأ شرح الكود الموجود في بحث الاسقاط الخلفي للمخطط البياني في الصفحات ١٥٥،١٥٦،١٥٧ حتى تفهم الكود التالي جيداً.

```
# setup initial location of window
r,h,c,w = 50,100,25,100 # simply hardcoded the values
track_window = (c,r,w,h)
```

هذين السطرين من الكود لتهيئة النافذة التي ستتضمن الجسم المطلوب تعقبه. هنا اخترنا أن يبدأ رسم المستطيل من الإحداثية (50,100) بطول وعرض ١٠٠ أي الجسم الذي ستم تعقبه سيكون ضمن هذا المستطيل.

```
roi = frame[r:r+h, c:c+w]
hsv_roi = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((0., 60.,32.)), np.array((180.,255.,255.)))
```

حددنا المنطقة من الفيديو التي سبدأ عندها لرسم roi، ثم حولنا الفيديو للفضاء اللوني HSV لسهولة تحديد المجال اللوني الذي سنتعقبه، ثم حددنا المجال اللوني للقناع mask بين مجال لوني معين.

```
roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
```

سنستخدم التابع cv2.calcHist() ليعطينيا فكرة عن توزع الشدة اللونية لبيكسلات صورة الجسم المطلوب ضمن القناع mask (مر معنا مسبقاً)

المتغير الأول صورة الدخل، والثاني يحدد القناة اللونية التي سنحسب له المخطط البياني، والثالث يمثل القناع، والرابع يحدد قيمة الBIN والخامس يحدد المجال.

```
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)
```

```
# Setup the termination criteria, either 10 iteration or move by atleast 1 pt
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )
```

هذا المتغير سنمرره للتابع إيجاد الجسم cv2.meanShift() (انظر للأسفل). وظيفته هي ضبط معيار إيقاف نافذة البحث عن تكرار البحث، سنقوم هنا بضبط معيار تكرار البحث على ١٠ تكرارات، وبمعيار تحرك نافذة البحث بما لا يقل عن نقطة.

```
while(1):
   ret ,frame = cap.read()
```

سنشأ حلقة while(1) تكرارية لا نهائية حتى نقرأ جميع إطارات الفيديو، لنحول كامل إطارات الصورة المكونة للفيديو من الفضاء اللوني RGB إلى HSV ، ثم لنطبق عليها عملية الاسقاط الخلفي، ثم لنطبق خوارزمية Camshift، ثم لنرسم مستطيل حول الجسم لمطلوب الذي نتعقبه.

```
   if ret == True:
      hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
      dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)
```

المتغير ret ستكون قيمته True ما دام الفيديو لم ينتهي، وعند انتهائه ستصبح قيمته False. بعد أن اختبرنا الشرط بأنه محقق سنحول كل إطار صورة للفيديو إلى الفضاء HSV. ونطبق على منطقة صورة الجسم الذي سنتعقبه عملية الاسقاط المخطط البياني على الخلفية.

(تم شرح التابع **cv2.calcBackProject** مسبقاً في فصل الإسقاط الخلفي للمخطط البياني)

```
      # apply meanshift to get the new location
      ret, track_window = cv2.meanShift(dst, track_window, term_crit)
```

هنا سنطبق خوارزمية Meanshift حتى نعرف الموضع الذي يتحرك إليه الجسم.

متغيرات التابع cv2.meanShift هي:

cv2.meanShift(probImage, window, criteria) ⟶ retval, window

- المتغير الأول: سنمرر له نتيجة الإسقاط الخلفي للمخطط البياني للجسم المطلوب.
- المتغير الثاني: سنمرر له نافذة البحث الأولية (أول مكان في الصورة سنبدأ عنده البحث عن الجسم).
- المتغير الثالث: يحدد الشرط الذي سيتم فيه إيقاف نافذة البحث عن تكرار البحث، ومعيار تحرك افذة البحث (مقدار انزياحها).

(انظر في الأعلى لشرح هذا المتغير)

سيعيد لنا هذا التابع الموقع الجديد لنافذة البحث وحجمها (track_window).

```
# Draw it on image
x,y,w,h = track_window
cv2.rectangle(frame, (x,y), (x+w,y+h), 255,2)
```

سنخزن موضع نافذة البحث الجديدة وقيمتها في x,y,w,h، ثم سنرسم مستطيل في مكان نافذة البحث.

- **خوارزمية Camshift:**

هل شاهدت بدقة النتيجة الماضية؟ هناك مشكلة وهي أن النافذة تملك نفس القياس سواء كانت السيارة أبعد أو أقرب من الكاميرا. وهذا الأمر غير جيد فنحن بحاجة لأن يتكيف الإطار مع حجم ودوران الهدف. لحل هذه المشكلة تم إيجاد خوارزمية Camshift (Continuously Adaptive Meanshift) من قبل Gary Bradsky في ورقة بحثه تحت عنوان

" Computer Vision Face Tracking for Use in a Perceptual User Interface"

آلية عمل هذه الخوارزمية:

يُطَبق أولا خوارزمية Meanshift، وبمجرد أن تتطابق Meanshift، يتم تحديث قياس الإطار وفق المعادلة $s = 2 \times \sqrt{\frac{M_{00}}{256}}$. وتحسب أيضا الاتجاه لأفضل قطع ناقص مناسب لها. نقوم مرة أخرى بتطبيق Meanshiftمع القياس الجديد وتقوم بنفس الخطوات السابقة حتى تتحقق الدقة المطلوبة.

Camshift في OpenCv:

هي تقريبا نفس Meanshift، ولكنها تعيد دوران المستطيل (هذه هي نتيجتنا) وبارامترات النافذة (تستخدم لتمرر كنافة البحث عن التكرار التالي). شاهد الكود التالي.

```
import numpy as np
import cv2

cap = cv2.VideoCapture('slow.flv')

# take first frame of the video
ret,frame = cap.read()

# setup initial location of window
r,h,c,w = 50,25,50,25 # simply hardcoded the values
track_window = (c,r,w,h)

# set up the ROI for tracking
roi = frame[r:r+h, c:c+w]
hsv_roi = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((0., 60.,32.)), np.array((180.,255.,255.)))
roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)
```

```
# Setup the termination criteria, either 10 iteration or move by atleast 1 pt
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 1, 1 )
while(1):
    ret ,frame = cap.read()

    if ret == True:
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

        # apply meanshift to get the new location
        ret, track_window = cv2.meanShift(dst, track_window, term_crit)

        # Draw it on image
        pts = cv2.boxPoints(ret)
        pts = np.int0(pts)
        cv2.polylines(frame,[pts],True, 255,2)
        cv2.imshow('img2',frame)

        k = cv2.waitKey(60) & 0xff
        if k == 27:
            break
        else:
            cv2.imwrite(chr(k)+".jpg",frame)
    else:
        break

cv2.destroyAllWindows()
cap.release()
```

النتيجة:

شرح الكود:

نفس شرح الكود السابق مع اختلاف فقط في تابع الرسم

```
# Draw it on image
pts = cv2.boxPoints(ret)
pts = np.int0(pts)
cv2.polylines(frame,[pts],True, 255,2)
cv2.imshow('img2',frame)
```

Opencv 3.0

boxPoints(...)

boxPoints(box[, points]) -> points

opencv 2.4.11

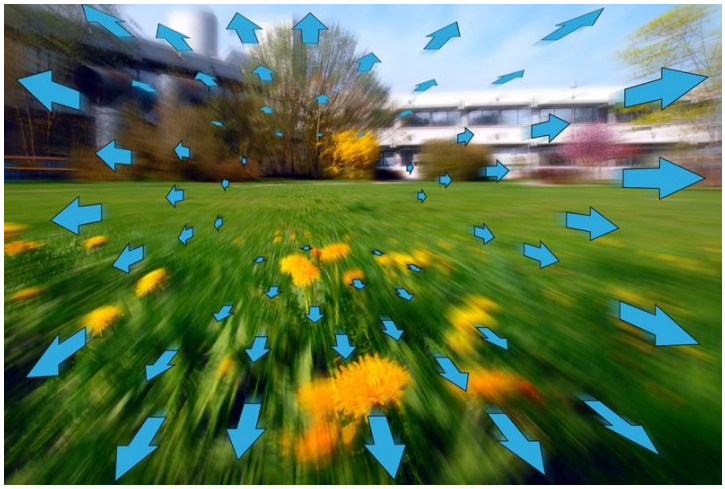
BoxPoints(...)

BoxPoints(box) -> points

مراجع إضافية:

- صفحة ويكيبيديا Camshift (يوجد فيه رسمتين توضيحيتين متحركتين).

التدفق البصري Optical Flow:



مقدمة:

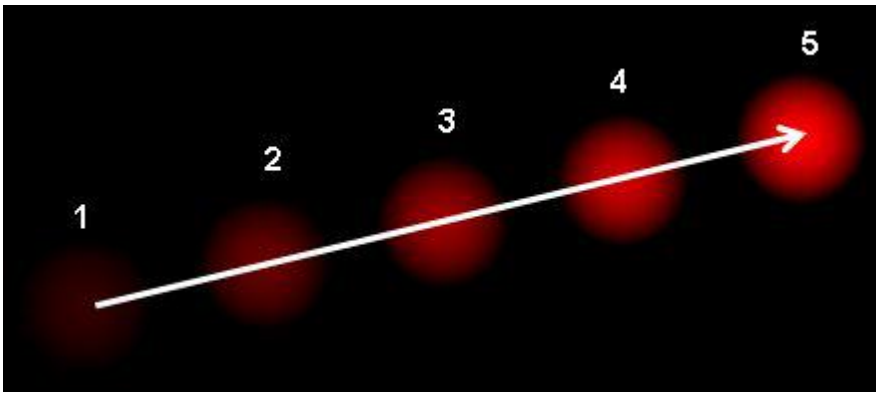
الآن سنناقش مفهوم هام وهو "Optical Flow " (التدفق البصري) الذي يهتم بالفيديو بشكل خاص، وبعدة تطبيقات أخرى.

الهدف:

- فهم مفهوم التدفق البصري وكيفية تقديره باستخدام طريقة Kanade method.
- سنتعلم استخدام التابع cv2.calcOpticalFlowPyrLK() لتعقب النقاط المميزة في الفيديو.

التدفق البصري:

التدفق البصري هو نمط الحركة الظاهرية لأجسام الصورة بين إطارين صورة frames متتاليين الناجم عن حركة الجسم أو الكاميرا. ويكون عبارة عن حقل من الأشعة ثنائية البعد، وكل شعاع عبارة عن شعاع إزاحة يبين حركة النقاط بين إطار الصورة الأول والثاني. شاهد الصورة التالية.



هذه الصورة تعرض انتقال الكرة في ٥ إطارات للصورة. يظهر السهم المعروض شعاع الإزاحة لانزياح الكرة. و يمكن استخدام التدفق البصري في العديد من التطبيقات كالمجالات التالية:

- معرفة طبيعة الحركة.
- ضغط الفيديو.
- تحقيق استقرار الفيديو (Video Stabilization).

التدفق البصري يعتمد على عدة فرضيات

١. شدة بيكسلات الجسم لا تتغير بين إطاري صورة متتاليين.
٢. البيكسلات المتجأورة لها حركة مماثلة.

لنعتبر البيكسل الذي في أول إطار للصورة هو $I(x, y, t)$ (هنا سنفحص بعد جديد ألا وهو الوقت t، في السابق تعاملنا مع الصور ولم نتعامل مع الفيديو لذلك لم نكن بحاجة لعنصر الوقت). البيكسل يتحرك مسافة قدرها (dx, dy) في الإطار التالي للصورة ويأخذ زمن قدره dt. وبما أن هذا البيكسل هو نفسه فشدة البيكسل لم تتغير، عندها يمكننا القول بأن:

$$\boldsymbol{I(x, y, t) = I(x + dx\,, y + dy\,, t + dt)}$$

سنأخذ منشور تايلور التقريبي للقسم اليميني من المعادلة، ونزيل العوامل المشتركة، ونقسم على dt فنحصل على المعادلة التالية: $\boldsymbol{f_x u + f_y v + f_t = 0}$ وتسمى هذه المعادلة بمعادلة التدفق البصري Optical Flow.

حيث:

$$\boldsymbol{f_y = \frac{\partial f}{\partial x}\ ;\ f_x = \frac{\partial f}{\partial x}}$$

$$v = \frac{dy}{dt} \; ; \quad u = \frac{dx}{dt}$$

في هذه المعادلة يمكننا إيجاد f_x و f_y فهما يمثلان التدرجات اللونية التي الصورة. وبشكل مماثل فإن f_t تمثل قيمة التدرج اللوني بعد فترة من الوقت. ولكن (u,v) غير معروفة لذلك لا يمكننا حل هذه المعادلة مع وجود متغيرين غير معروفين. لذلك تم توفير عدة طرق لحل هذه المشكلة، وإحدى هذه الطرق هي لوكاس كاندي Lucas-Kanade.

طريقة لوكاس كاندي:

لقد شاهدنا هذه الفرضية من قبل. إن كل البيكسلات المجأورة سوف تملك نفس الحركة. إن طريقة لوكاس تقوم بأخذ بقعة بحجم 3x3 حول هذه النقطة. لذلك كل ٩ نقاط ستملك نفس الحركة، عندها يمكننا إيجاد (f_x, f_y, f_t) لهذه النقاط التسعة. حتى الآن تبقى المشكلة بحل ٩ معادلات مع متغيرين غير معروفين. المعادلة التالية هي الحل النهائي لحل مشكلة جملة معدلتين مع مجهولين إثنين، وبحل هذه المعادلة نحصل على الحل.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{xi}^2 & \sum_i f_{xi} f_{yi} \\ \sum_i f_{xi} f_{yi} & \sum_i f_{yi}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{xi} f_{ti} \\ -\sum_i f_{yi} f_{ti} \end{bmatrix}$$

(مكتشف زوايا هاريس يقول لن أن أفضل نقاط التعقب هي الزوايا).

- **طريقة لوكاس كاندي- للتدفق البصري في OpenCv:**

توفر لنا مكتبة OpenCv حل جميع المعادلات السابقة في تابع واحد وهو cv2.calcOpticalFlowPyrLK().

سنقوم بإنشاء تطبيق بسيط لتعقب بعض نقاط في الفيديو. لتحديد هذه النقاط نستخدم التابع cv2.goodFeaturesToTrack(). نأخذ أول إطار للصورة، ونحدد بعض نقاط زوايا Shi-Tomasi، ثم بشكل متكرر نتتبع هذه النقاط باستخدام طريقة لوكاس كاندي للتدفق البصري.

بالنسبة للتابع cv2.calcOpticalFlowPyrLK() فإننا نمرر له إطار الصورة السابق والنقاط السابقة وإطار الصورة التالي. ويعيد لنا هذا التابع النقاط التالية(القادمة) جنبا إلى جنب مع رقم يبين حالة

كل نقطة، فإذا تم إيجاد النقطة التالية يكون رقم الحالة المُبَين لوضع النقطة ١، وإلا سيعيد قيمة ٠.
ونكرر تمرير النقاط التالية كما مررنا النقاط السابقة في الخطوة التالية. شاهد الكود التالي:

```
import numpy as np
import cv2
cap = cv2.VideoCapture('slow.mp4')

# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 100,
                       qualityLevel = 0.3,
                       minDistance = 7,
                       blockSize = 7 )

# Parameters for lucas kanade optical flow
lk_params = dict( winSize = (15,15),
                  maxLevel = 2,
criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Create some random colors
color = np.random.randint(0,255,(100,3))

# Take first frame and find corners in it
ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **feature_params)

# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)

while(True):
    ret,frame = cap.read()
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)

    # Select good points
    good_new = p1[st==1]
    good_old = p0[st==1]

    # draw the tracks
    for i,(new,old) in enumerate(zip(good_new,good_old)):
        a,b = new.ravel()
        c,d = old.ravel()
        cv2.line(mask, (a,b),(c,d), color[i].tolist(), 2)
        cv2.circle(frame,(a,b),5,color[i].tolist(),-1)
```

```
    img = cv2.add(frame,mask)

    cv2.imshow('frame',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
    # Now update the previous frame and previous points
    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1,1,2)

cv2.destroyAllWindows()
cap.release()
```

هذا الكود لا يتحقق فيما إذا كانت النقاط الأساسية صحيحة أم لا. وحتى إذا اختفت أي نقطة مميزة في الصورة، هناك فرصة لإيجاد التدفق البصري في النقطة القادمة التي تبدو قريبة منها. لذلك يعتبر هذا تعقب قوي، ونقاط الزأوية يجب أن يتم البحث عنها خلال فترات معينة. مكتبة OpenCV وفرت لنا نماذج جاهزة لهذا الأمر مثل إيجاد النقاط المميزة كل ٥ إطارات للصورة، وتشغيل الفحص الخلفي لنقاط التدفق البصري لنحصل فقط على النقاط الجيدة منها فقط.

"C:\opencv\sources\samples\python2\lk_track.py"

شاهد النتيجة التي حصلنا عليها:

شرح الكود:

```
# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 100,
                       qualityLevel = 0.3,
                       minDistance = 7,
                       blockSize = 7 )
```

التابع dic وظيفته تخزين البيانات كقائمة كما في قواعد المعطيات. وبدلاً من استخدام الأرقام فقط للوصول للبيانات يمكنك استخدام أي شيئ تقريبا. فهذا يتيح لك التعامل مع الديكت (dict) كقاعدة بيانات لتخزين البيانات وتنظيمها.

هنا عرفنا قاعدة معطيات باسم feature_params وخزنا بداخلها العناصر maxCorners وqualityLevel وminDistance وblockSize وأعطينا لكل عنصر قيمة معينة.

```
# Parameters for lucas kanade optical flow
lk_params = dict( winSize = (15,15),
                  maxLevel = 2,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))
```

أيضا هنا قمنا بتعريف قاعدة معطيات باسم lk_params وخزنا بداخله العناصر winSize وmaxLevel وcriteria.

```
# Create some random colors
color = np.random.randint(0,255,(100,3))
```

التابع numpy.random.randint() يعيد قيمة متغير عشوائي من النوع integer بين القيمة low و hight يأخذ الشكل(numpy.random.randint(*low*, *high*, *size*, *dtype*

```
# Take first frame and find corners in it
ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **feature_params)
```

الآن سنوجد نقاط الزوايا القوية في كل frame وذلك عن طريق التابع cv2.goodFeaturesToTrack() الذي يأخذ المتغيرات التالية:

cv2.goodFeaturesToTrack(image, maxCorners, qualityLevel, minDistance[, corners[, mask[, blockSize[, useHarrisDetector[, k]]]]]) → corners

- Image: الصورة المطلوب حساب نقاط الزاوية القوية فيها.
- Mask: (خياري) إذا أردت اكتشاف نقاط الزاوية لمنطقة محددة من الصورة عندها يمكنك تمرير قيمة القناع المطلوب، هنا لن نستعمله لذلك سنمرر القيمة None.

المتغيرات البقية قمنا بتعريفها في بداية الكود على شكل قاعدة معطيات باسم feature_params وهذه المتغيرات هي:

- maxCorners: أقصى عدد للزوايا التي سيعيدها. إذا وجد عدد أكبر من أقصى قيمة سيتم إرجاع نقاط الزوايا الأقوى منها.
- qualityLevel: هذا البارامتر يحدد نوعية الحد الأدنى المقبول للزأوية.
- minDistance: أقل مسافة مسموحة بين الزوايا
- blockSize:

```
# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)
```

في هذا السطر سننشأ قناع للصورة والذي سيساعدنا في عملية الرسم. التابع numpy.zeros_like يعيد مصفوفة صفرية (أي صورة سوداء) بنفس شكل ونوع الصورة المعطاة .

```
while(True):
    ret,frame = cap.read()
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

ما دام الفيديو لم ينتهي سنحول كل إطار صورة فيه إلى المستوي الرمادي.

```
    # calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)
```

الآن سنقوم بحساب التدفق البصري وذلك عن طريق التابع cv2.calcOpticalFlowPyrLK الذي يأخذ المتغيرات التالية:

cv2.calcOpticalFlowPyrLK(prevImg, nextImg, prevPts[, nextPts[, status[, err[, winSize[, maxLevel[, criteria[, flags[, minEigThreshold]]]]]]]]) → nextPts, status, err

- prevImg: الصورة السابقة. (الفريم السابق)
- nextImg: الصورة التالية. (الفريم التالي)
- prevPts: متجهات (أشعة) إحداثيات النقاط التي نحتاج أن نوجد التدفق لها (سنمرر له إحداثيات نقاط الزوايا القوية). يجب أن تكون إحداثيات النقاط بالصيغة float.

- winSize: يحدد قياس نافذة البحث عند كل مستوى هرمي.
- maxLevel: أقصى عدد لمستويات الهرم. إذا مررنا له القيمة ٠ عندها لن يستخدم الأهرامات (مستوي واحد فقط)، وإذا مررنا القيمة ١ سيستخدم مستويين هرميين.
- criteria: يحدد معيار إنهاء خوارزمية البحث المتكررة.
- ويعيد لنا هذا التابع القيم التالية:
- nextPts: يعطينا الموضع الجديد للميزات المدخلة (الزوايا) في الصورة الجديدة (التالية)
- Status: حالة وضع متجهات الخرج. كل متجهة تأخذ القيمة ١ إذا وجد تدفق في النقطة المميزة، وإلا ستأخذ القيمة ٠. (سيفيدنا هذا المتغير في تحديد نقاط الزوايا اميزة التي يتم فيها التدفق (نقطة متحركة) من تلك التي التي لا يوجد فيها تدفق (نقطة ساكنة)).
- Err: الأخطاء الناجمة عن المتجهة.

```
# Select good points
good_new = p1[st==1]
good_old = p0[st==1]
```

هنا قمنا بأخذ نقاط التدفق الجيدة وحذفنا النقاط التي توقف فيه التدفق (النقاط التي تكون حالتها تساوي الصفر Status=0 تم حذفها). ولقد عرفنا النقاط المميزة التي حالة التدفق فيها جيدة أم لا عن طريق تابع التدفق cv2.calcOpticalFlowPyrLK() الذي يعيد لنا قيمة الـ status إما ١ إن كان هناك تدفق في النقطة المميزة وإلا ٠ إن لم يكن هناك تدفق. (هنا المتغير status هو st).

```
# draw the tracks
for i,(new,old) in enumerate(zip(good_new,good_old)):
    a,b = new.ravel()
    c,d = old.ravel()
    cv2.line(mask, (a,b),(c,d), color[i].tolist(), 2)
    cv2.circle(frame,(a,b),5,color[i].tolist(),-1)
```

في هذا الأمر سنقوم برسم نقاط التعقب ومسارات حركتها.

- التابع enumerate() يعيد عدد الأجسام (عدد المجموعات).
- التابع zip() يعيد هذا التابع قائمة بالمجموعات.
- التابع ravel() يستخدم لتحويل المصفوفة سواء كانت 3D أو 2D إلى مصفوفة أحادية البعد 1D.

سنقوم هنا بتخزين الإحداثيات الجديدة للنقاط المميزة (x,y)في المتغيرين a,b، والإحداثيات القديمة في المتغيرين c,d. ثم سنمرر الإحداثيات القديمة والجديدة لتابع رسم الخط ليرتسم على الفيديو مسار تدفق الإحداثية، ونمرر الإحداثية الجديدة لتابع رسم الدائرة حتى يرسم دائرة في مكان الإحداثية الجديدة للنقطة المميزة.

```
img = cv2.add(frame,mask)
```

هذا السطر يفيدنا فقط في عملية الرسم.

- **كثافة التدفق البصري في OpenCv:**

طريقة لوكاس كاندي تحسب التدفق البصري لمجموعة ميزات متفرقة (في مثالنا، اكتشاف الزوايا باستخدام خوارزمية Shi-Tomasi). مكتبة OpenCv توفر خوارزمية أخرى لإيجاد كثافة التدفق البصري. تقوم هذه الخوارزمية على حساب التدفق البصري لكل النقاط في إطار صورة. فهي تبنى على خوارزمية Gunner Farneback's (لمعرفة ما هي خوارزمية Gunner Farneback's اطلع على ورقة البحث التي كتبها Gunner Farneback بعنوان "Two-Frame Motion Estimation Based on Polynomial Expansion").

المثال التالي يعرض كيفية إيجاد كثافة التدفق البصري باستخدام الخوارزمية Farneback. سنحصل على مصفوفة بـ ٢ قناة مع شعاع التدفق البصري (u,v). سنوجد حجمها واتجاهها. سنقوم

بتلوين نتيجة الكود كما هو واضح في الصورة التالية لنحصل على نتيجة تصور أفضل. الاتجاه يتوافق مع قيم Hue للصورة. الحجم يتوافق مع قيمة المستوى. شاهد الكود التالي:

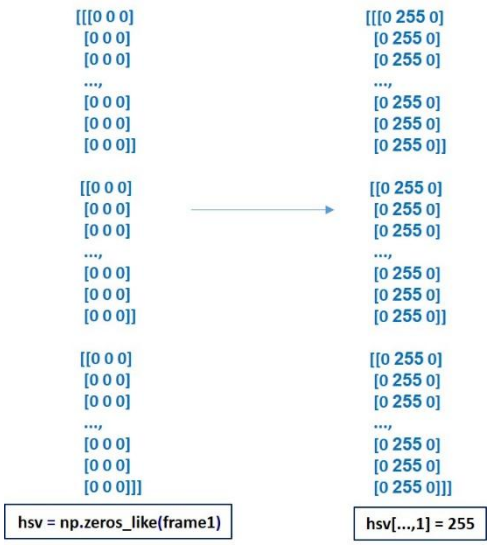
```
import cv2
import numpy as np

cap = cv2.VideoCapture("vtest.avi")

ret, frame1 = cap.read()
prvs = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)
hsv = np.zeros_like(frame1)
hsv[...,1] = 255

while(1):
    ret, frame2 = cap.read()
    next = cv2.cvtColor(frame2,cv2.COLOR_BGR2GRAY)

    flow = cv2.calcOpticalFlowFarneback(prvs,next, 0.5, 3, 15, 3, 5, 1.2, 0)

    mag, ang = cv2.cartToPolar(flow[...,0], flow[...,1])
    hsv[...,0] = ang*180/np.pi/2
    hsv[...,2] = cv2.normalize(mag,None,0,255,cv2.NORM_MINMAX)
    rgb = cv2.cvtColor(hsv,cv2.COLOR_HSV2BGR)
    cv2.imshow('frame2',rgb)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
    elif k == ord('s'):
        cv2.imwrite('opticalfb.png',frame2)
        cv2.imwrite('opticalhsv.png',rgb)
    prvs = next

cap.release() , cv2.destroyAllWindows()
```

النتيجة:

شرح الكود:

```
ret, frame1 = cap.read()
prvs = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)
hsv = np.zeros_like(frame1)
hsv[...,1] = 255
```

في البداية نقوم بقراءة الفيديو، ثم نحوله للمستوي الرمادي، بعد ذلك نقوم بتعريف مصفوفة صفرية (أي صورة سوداء) باسم hsv بنفس أبعاد الفيديو، وبما أن كل إطار صورة في الفيديو ملون فهو يأخذ ثلاث قنوات (أي مصفوفة ثلاثية البعد 3D) إذا ستكون أيضا المصفوفة الصفرية hsv التي عرفناها مكونة من ثلاث قنوات، وكل هذه القنوات تأخذ القيمة ٠. في الأمر hsv[...,1] = 255 سنقوم بوضع القيمة ٢٥٥ للقناة الثانية (سيصبح لدينا بدل الصورة السوداء صورة خضراء).



```
while(1):
    ret, frame2 = cap.read()
    next = cv2.cvtColor(frame2,cv2.COLOR_BGR2GRAY)
```

```
flow = cv2.calcOpticalFlowFarneback(prvs,next, 0.5, 3, 15, 3, 5, 1.2, 0)
```

التابع cv2.calcOpticalFlowFarneback() يحسب لنا كثافة التدفق البصري باستخدام خوارزمية غورنار فاربيناك (Gunnar Farneback's) ويأخذ الشكل التالي:

cv2.calcOpticalFlowFarneback(prev, next, pyr_scale, levels, winsize, iterations, poly_n, poly_sigma, flags[, flow]) → flow

- prev: صورة الدخل يجب أن تكون بقناة واحدة.
- Next: ثاني صورة دخل. يجب أن تكون بنفس حجم ونوع الصورة الأولى (prev).
- pyr_scale: هذا البارمتر يحدد حجم الصورة فإذا كانت قيمته (<1) يعني سيبني الأهرامات لكل صورة، أما إذا كانت قيمته pyr_scale=0.5 هذا يعني الهرم سيكون تقليدي أي سيكون كل مستوى أصغر بمرتين من المستوي الذي قبله.
- Levels: عدد مستويات الهرم بما في ذلك الصورة الأولى. إذا مررنا لهذا البارمتر القيمة ١ هذا يعني بأنه لا يوجد مستويات إضافية سيتم إنشاؤها وفقط سيستخدم الصورة الأصلية.
- Winsize: حجم النافذة. كلما كانت القيمة أكبر ستكون قوى الخوارزمية أقوى للتغلب على تشويش الصورة، وإعطاء فرصة أكبر للكشف عن الحركات السريعة، ولكنها تعطي حركة أكثر ضبابية (blurred motion field).
- Iterations: عدد مرات تكرار الخوارزمية عند كل مستوي هرمي.
- poly_n: حجم البيكسلات المجأورة التي تستخدم للبحث عن تمدد متعدد الحدود (polynomial) لكل بيكسل. إذا مررنا أكبر قيمة هذا يعني أن الصورة سيصبح سطحها انعم (smoother) وتزيد مرونة الخوارزمية مما يعطيها قوى، ولكن تزيد أيضا ضبابية حقل الحركة، بشكل عام تكون قيمته ٥ أو ٧.
- poly_sigma: نمرر له القيمة ١.١ إذ كان poly_n=5، والقيمة ١.٥ إذا كان poly_n=7.

- flags: علم التشغيل. يتكون مما يلي:
  - OPTFLOW_USE_INITIAL_FLOW
  - OPTFLOW_FARNEBACK_GAUSSIAN: يستخدم الفلتر الغوسي بدل الفلتر الصندوقي لنفس حجم التدفق البصري المعتبر. هذا الخيار يعطينا دقة أفضل للتدفق من الفلتر الصندوقي، ولكن على حساب السرعة عادة. عند اختيار النافذة الغأوسية يجب أن نعطي المتغير Winsize قيمة كبيرة لنحصل على نفس المستوى من الشدة.
- flow: يحسب تدفق الصورة التي تملك نفس قياس الصورة الأولى (prev) والنوع CV_32FC2. لن نستخدمه. أحيانا إن لم نستخدمه يجب أن تمرر مكانه القيمة None.

```
mag, ang = cv2.cartToPolar(flow[...,0], flow[...,1])
```

cv2.cartToPolar

cv2.cartToPolar(x, y[, magnitude[, angle[, angleInDegrees]]]) → magnitude, angle

```
hsv[...,0] = ang*180/np.pi/2
hsv[...,2] = cv2.normalize(mag,None,0,255,cv2.NORM_MINMAX)
rgb = cv2.cvtColor(hsv,cv2.COLOR_HSV2BGR)
```

مكتبة OpenCv توفر نماذج متقدمة عن التدفق البصري، لمعرفة هذه النماذج شاهد النموذج الموجود ضمن مكتبة OpenCv في الملف الذي مساره "C:\opencv\sources\samples\python2\opt_flow.py"

أمثلة إضافية:

- **"C:\opencv\sources\samples\python2\lk_track.py"**
- **"C:\opencv\sources\samples\python2\opt_flow.py"**

طرح الخلفية Background Subtraction:

مقدمة:

نحتاج في عدة تطبيقات أن نستخرج الخلفية لتنفيذ عمليات عدة مثل تعقب كائن.

الهدف:

- سنتعرف على طرق طرح الخلفية التي توفرها مكتبة OpenCv.

أساسيات:

طريقة طرح الخلفية هي خطوة أساسية في العديد من التطبيقات التي تعتمد بشكل رئيسي على معالجة الرؤية الحاسوبية. على سبيل المثال، في حالة عد الزوار عن طريق الكاميرا يتم عد الزوار الداخلين والخارجين من الغرفة، أو إحصاء حركة المرور أو معرفة المعلومات عن المركبات التي تمر أمام الكاميرا. في كل هذه الحالات نحتاج استخراج شخص أو مركبة لوحده. فأنت بحاجة لهذه التقنية لاستخراج الحركة الأمامية من الخلفية (كشخص أو مركبة) وفصلها عن الخلفية.

إذا كان لديك صورة الخلفية لوحدها مثل صورة الغرفة بدون الأشخاص أو صورة للطريق دون المركبات، يمكن فعل هذا بسهولة عن طريق طرح الصورة الجديدة من خلفيتها، عندها ستحصل على الجسم الأمامي لوحده.

ولكن في معظم الحالات لا نملك الخلفية لوحدها حتى نستطيع القيام بعملية الطرح، لذلك سنحتاج لاستخراج الخلفية من أي صورة لدينا. عملية استخراج الخلفية تصبح أكثر تعقيدا عند وجود ظل (مثل ظل المركبة)، لأن الظل أيضا يتحرك مع الجسم. الطرح البسيط سيحدد الخلفية الأمامية (الجسم + الظل). أن تستخرج الجسم لوحده هذا شيء معقد.

لقد تم إيجاد عدة خوارزميات لحل هذه المشكلة، حيث قامت مكتبة OpenCv بطرح ثلاث خوارزميات من السهل على المستخدم استخدامها، سنقوم بشرح هذه الخوارزميات الثلاث كل واحدة لوحدها.

١- خوارزمية طرح الخلفية MOG:

هذه الخوارزمية تقوم بفصل المزيج خلفية\أمامية عن طريق المزائج الغأوسية. وقد تم عرض هذا في ورقة البحث " An improved adaptive background mixture model for real-time tracking with shadow detection" للمطورين P. KadewTraKuPong و R. Bowden في عام ٢٠٠١.

فهذه الطريقة تعتمد على مزج كل بيكسل في الخلفية ومزجه مع التوزيعات الغأوسي K (k=3 to 5).شدة المزيج تمثل مدة الوقت لحالة الألوان التي تبقى في المشهد. الخلفية المحتملة هي تلك التي ألوانها تبقى لفترة أطول دون تغير.

عند كتابة الكود سنحتاج لإنشاء جسم الخلفية وذلك باستخدام التابع cv2.createBackgroundSubtractorMOG().

هذا التابع يملك بعض البارامترت الاختيارية مثل مدة البقاء (للون)، وعدد الأمزجة، والعتبة (history، threshold، number of gaussian mixtures).

إذا لم يتم تعريف هذه القيم فإنها تأخذ قيم افتراضية بشكل تلقائي. بالنسبة لداخل حلقة الفيديو فإننا سنستخدم التابع backgroundsubtractor.apply() للحصول على قناع للأجسام الأمامية.

شاهد الكود التالي:

```
import numpy as np
import cv2

cap = cv2.VideoCapture('vtest.avi')
fgbg = cv2. BackgroundSubtractorMOG()
while(1):
    ret, frame = cap.read()
    fgmask = fgbg.apply(frame)
    cv2.imshow('frame',fgmask)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

ملاحظة: في **OpenCV** الإصدار ٣ استخدم التابع **cv2.createBackgroundSubtractorMOG()** بدل التابع **cv2. BackgroundSubtractorMOG()**.

النتيجة: موجودة في الأسفل حيث سنقارن بين جميع خوارزميات طرح الخلفية.

شرح الكود:

```
fgbg = cv2. BackgroundSubtractorMOG()
```

في البداية قمنا بتعريف جسم الخلفية وتخزينه في المتغير fgbg وذلك عن طريق التابع cv2.BackgroundSubtractorMOG() الذي يأخذ المتغيرات التالية:

```
cv2.BackgroundSubtractorMOG([history, nmixtures, backgroundRatio[, noiseSigma]]) →
<BackgroundSubtractorMOG object>
```

- history: طول مدة الحفظ (اظن مدة بقاء اللون في مكانه).
- nmixtures: عدد الخلائط(المزائج) الغأوسية.
- backgroundRatio: نسبة الخلفية.
- noiseSigma: شدة الضجيج.

```
while(1):
    ret, frame = cap.read()
    fgmask = fgbg.apply(frame)
    cv2.imshow('frame',fgmask)
```

قمنا بفتح حلقة while لنقوم بعملية طرح الخلفية لجميع إطارات الصورة المكونة للفيديو. في كل تكرار للحلقة نحصل على المنطقة الأمامية من الجسم ملونة بالأبيض وذلك عن طريق تطبيق تابع طرح الخلفية من الأجسام الأمامية على كل إطار في الفيدو عن طريق التعليمة () apply.

٢- خوارزمية طرح الخلفية MOG٢:

أيضا المزائج الغأوسية تعتمد على خوارزمية فصل المنطقة الأمامية عن الخلفية. تم وضع هذه الخوارزمية بناءً على ورقتي البحث للمطور Z. Zivkovic بعنوان " Improved adaptive Gausian mixture model for background subtraction" كتبت عام ٢٠٠٤، و " Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction" كتبت عام ٢٠٠٦.

إحدى الميزات الهامة لهذه الخوارزمية بأنها تختار العدد المناسب للتوزع الغأوسي لكل بيكسل (تذكر بأنه في أخر حالة أخذنا K للتوزع الغأوسي على كامل الخوارزمية وليس لكل بكسل)، وهذا يوفر القدرة على التكيف بشكل أفضل مع المشاهد المختلفة التي تغيير فيها شدة الإضاءة.

في الحالة السابقة، أنشأنا جسم طرح الخلفية. هنا نملك الخيار فيما إذا كنا نريد تحديد الظل لعرضه أم لا. لتحديد الظل نكتب detectShadows = True (القيمة الافتراضية تكون True)، وهذا يحدد ويعلم الظلال، ولكن يخفف من سرعة التنفيذ. الظلال سوف تعلم باللون الرمادي.

```
import numpy as np
import cv2
cap = cv2.VideoCapture('vtest.avi')

fgbg = cv2. createBackgroundSubtractorMOG2()

while(1):
    ret, frame = cap.read()

    fgmask = fgbg.apply(frame)

    cv2.imshow('frame',fgmask)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

(النتيجة ستعطى في الأسفل)

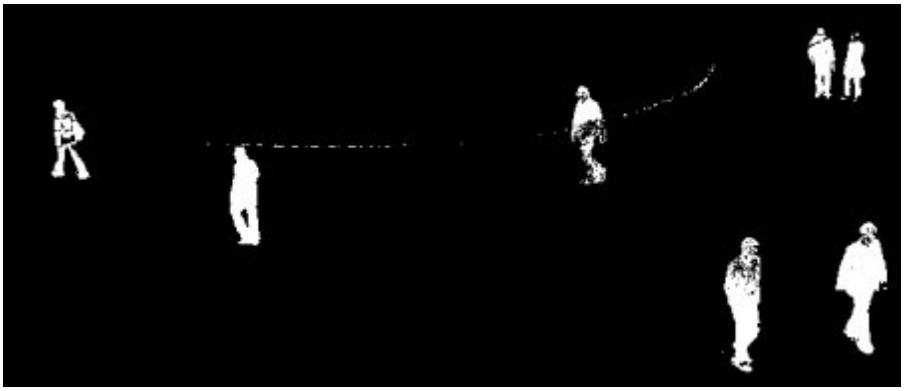
ملاحظة: في إصدار **OpenCv V2** قم باستخدام التابع **BackgroundSubtractorMOG2()** بدلا من التابع **.createBackgroundSubtractorMOG2()**

النتائج:

إطار الصورة الأصلي

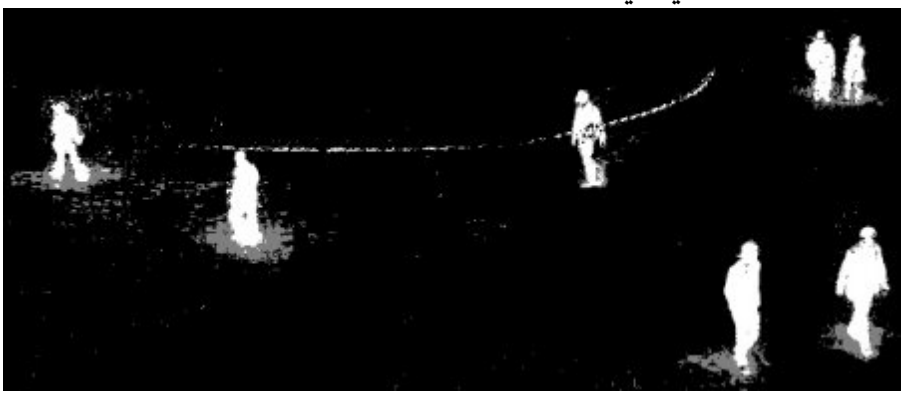
الصورة التالية تعرض إطار الصورة رقم ٢٠٠ من الفيديو.

نتيجة الطرح الخلفي لخوارزمية MOG:



نتيجة الطرح الخلفي لخوارزمية MOG2:

المنطقة الملونة باللون الرمادي هي منطقة الظلال.



الفصل التاسع

" أن تتعثر فهذا يعني أنك تسير في الطريق، فلم أسمع بأحد يتعثر وهو لا يتحرك "

تشارلز كيترينج - مهندس ومخترع أمريكي

الفصل التاسع: اكتشاف الأجسام Object Detection

اكتشاف الأجسام باستخدام خوارمية Haar Cascades:

سنتعلم تحديد الوجه باستخدام خوارزمية كاسكيد هار (Haar Cascades).

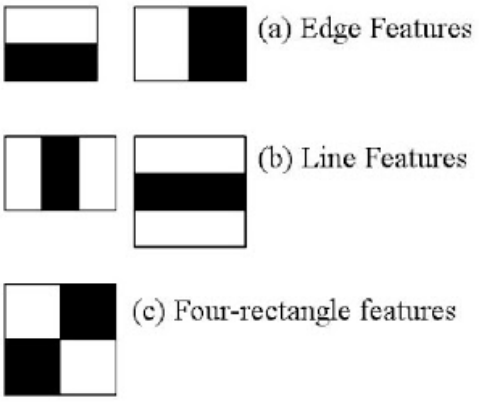
الهدف:

- سنتعلم أساسيات اكتشاف الوجه اعتمادا على مزايا هار Haar ....
- وسنقوم بنفس الشيئ لاكتشاف العين.

مقدمة نظرية:

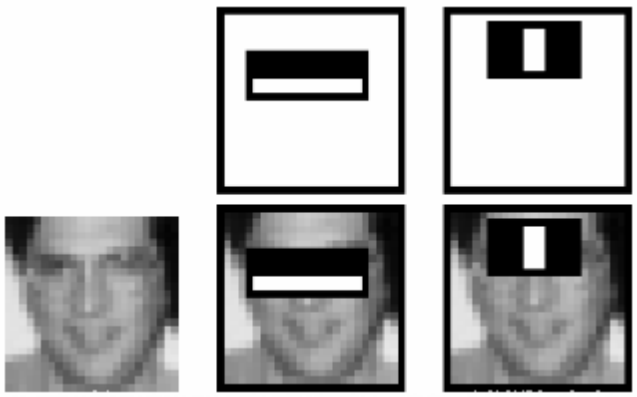
سنكتشف الأجسام باستخدام خوارزمية Haar والتي تعتمد على استخدام عدة مصنفات، وتعتبر هذه الطريقة عملية لاكتشاف الأجسام. تم اقتراح هذه الخوارزمية من قبل Paul Viola و Michael Jones ضمن حلقة بحثهم " Rapid Object Detection using a Boosted Cascade of Simple Features" التي كتبوها عام ٢٠٠١. هذه الخوارزمية تعتمد على تدريب الجهاز على التعلم اعتمادا على المقاربة، حيث يتم تدريب التابع على عدة صور إيجابية وسلبية. ثم نستخدم هذا التابع المدرب للكشف عن الأجسام في صور أخرى.

هنا سوف نستعمله لاكتشاف الوجه. سنحتاج في البداية لعدة صور إيجابية (صور لوجوه) وصور سلبية (صور بدون وجوه) لندرب المصنف (classifier) على اكتشاف الوجه. ثم سنحتاج لاستخراج الميزات منها. لذلك سنستخدم ميزات Haar كما هو مبين في الصورة التي في الأسفل. كل ميزة تحصل على قيمة واحدة من خلال طرح مجموعة البيكسلات التي تحت المستطيل الأبيض من مجموع البيكسلات التي تحت المستطيل الأسود.



الآن سيتم استخدام كل الأحجام والأماكن الممكنة للقناع لحساب العديد من الميزات (تخيل كم عدد الحسابات التي سنحتاجها؟ فحتى عند استخدام نافذة قناع بحجم 24x24 سيعطينا نتيجة لأكثر من ١٦٠٠٠٠ ميزة). لحساب كل ميزة سنحتاج لإيجاد مجموع البيكسلات التي تحت المستطيل الأبيض والأسود. لحل هذه المشكلة عرضوا صورة متكاملة (غير متجزئة)، هذا الأمر يبسط حساب مجموع البيكسلات، عدد البيكسلات قد يكون كبير لذلك ستشارك في هذه العملية ٤ بيكسلات فقط. أليس هذا الأمر أفضل؟ فهو يجعل الأمر يتم بسرعة هائلة.

لكن من بين كل هذه الميزات حسبنا عدد من البيكسلات ليس لها علاقة. على سبيل المثال لنفرض لدينا الصورة التي في الأسفل. الصف العلوي يظهر ميزيتين جيدتين. أول ميزة تبدو أنها تركز على خاصية منطقة العينين والتي تكون غالبا أغمق من منطقة الأنف والخدين. أما الميزة الثانية تركز على خاصية منطقة العينين تكون أغمق من جسر النظارتين والأنف. ولكن النوافذ التي يتم تطبيقها على الخدين أو أي منطقة أخرى لا تهمنا. فكيف سنتختار أفضل الميزات من بين ١٦٠,٠٠٠ ميزة؟



لهذا الأمر سنطبق كل ميزة على جميع صور التدريب. لكل ميزة ستوجد أفضل عتبة التي سوف تصنف الوجوه الإيجابية والسلبية. ولكن من المعلوم بأنه سيكون هناك أخطاء أو سوء في التصنيف. علينا تحديد الميزات مع أدنى معدل للخطأ، وهذا يعني بأنها ستختار أفشل الميزات لتصنف الوجه من عدم الوجه في الصورة. (العملية ليست بهذه البساطة. فكل صورة ستأخذ وزنا متسأويا في البداية. ويعد كل تصنيف سيتم زيادة أوزان الصور المصنفة بشكل خاطئ. ثم تتم مرة أخرى نفس العملية، ويتم حساب معدل الأخطاء الجديدة، والأوزان الجديدة أيضاً. تستمر هذه العملية حتى يتم التحقق من معدل الدقة أو الخطأ المطلوب إيجاده لعدد من الميزات)

التصنيف النهائي يعطينا مجموع أوزان التصنيفات الضعيفة. تدعى ضعيفة لأنه لوحدها لا تستطيع تصنيف الصورة.، ولكن مع التصانيف الأخرى سيتشكل لدينا مصنف قوي. تخبرنا ورقة البحث بأنه عند توفر ميزات حتى ٢٠٠ ميزة سنحصل على دقة اكتشاف 95%. العدد النهائي اذي حصلنا عليه كان ٦٠٠٠ ميزة (تخيل كيف خفضنا بهذه العملية التي قمنا بها عدد الميزات من ١٦٠,٠٠٠ إلى ٦٠٠٠، وهذا مكسب كبير لنا بالذات من ناحية سرعة التنفيذ).

يمكنك الأن أخذ الصور. وتستعمل كل نافذة بحجم 24x24. وتطبق ٦٠٠٠ ميزة عليها. تحقق فيما إذا كان وجه أم لا. أليست هذه الطريقة غير عملية ومضيعة للوقت؟ نعم هي غير فعالة لذلك أوجد الكاتب حل لذلك.

في الصورة معظم المناطق هي ليست منطقة وجه، لذلك أفضل طريقة بأن تفحص النافذة هل هي منطقة وجه فإذا لم تكن يتم حذفها فورا ولا يتم معالجتها مرة أخرى. بهذه الطريقة نوفر الوقت للتحقق من إمكانية وجود وجه في منطقة معينة.

لذلك الأمر تم إيجاد مفهوم Cascade of Classifiers (تسلسل المصنفات). فبدلا من تطبيق ٦٠٠٠ ميزة على نافذة، نجمع هذه لميزات على مراحل(stages) مختلفة من المصنف (Classifiers) ونطبق ميزة تلو الأخرى. (عادة المراحل الأولى ستحوي أقل عدد من الميزات). إذا أخفقت النافذة المرحلة الأولى يتم استبعادها.

المؤلف قد اكتشف أكثر من ٦٠٠٠ ميزة خلال ٣٨ مرحلة حيث تم اكتشاف عدد من الميزات في أول ٥ مراحل وذلك كما في الترتيب التالي: 1,10,25,25,50

- **خوارزمية Harr Cascade في OpenCv:**

توفر لنا مكتبة OpenCv المدرب (trainer) الذي يعطينا القدرة على تدريب الخوارزمية لإنشاء مصنف (classifier) الذي سيتتضمن المعلومات (الصفات المميزة) التي ستفيدنا في التعرف على الجسم الذي تم تدريب الخوارزمية للتعرف عليه، وتوفر لنا OpenCv أيضا المكتشف (detector)

الذي يمكننا من اكتشاف وجود الأجسام. سنتعرف على مصطلح جديد اسمه المصنف classifier وهو يتضمن مجموعة صفات لتمييز جسم معين، فمثلا إذا كان عندنا مصنف للسيارة فهذا يعني بأنه يحوي على الصفات المميزة التي تدلنا على أن الجسم هو سيارة.

مكتبة OpenCv توفر لك القدرة على تدريب مصنفك classifier على أي جسم مثل السيارة، الطائرة، الحشرات، الحيوانات. بحيث يمكنك إنشاء المصنف الخاص بك. لمعرفة كل تفاصيل إنشاء المصنف classifier للجسم الذي تريد اتبع الرابط Cascade Classifier Training. والآن سنتعامل مع المكتشف detector وبعد ذلك سأشرح طريقة تدريب الخوارزمية بأسلوب سهل.

مكتبة OpenCv تحتوي على عدة مصنفات(classifier) مدربة وجاهزة كالوجه والعين والفم.

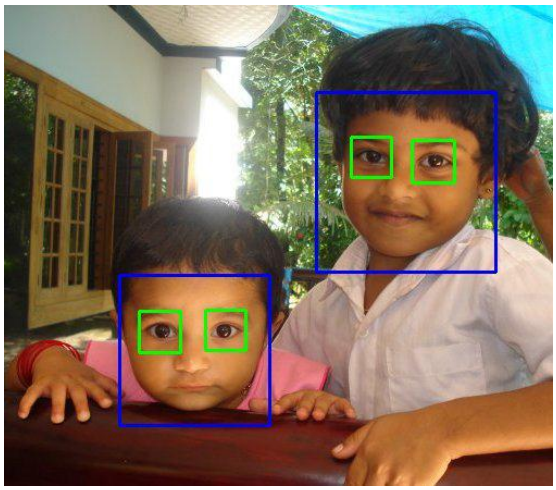
هذه المصنفات موجودة بصيغة XML ضمن مجلد OpenCv ضمن مسار مجلد OpenCv التالي opencv\sources\data\haarcascades. دعنا الآن ننشأ مكتشف للوجه والعين مع OpenCv.

في البدية سنحتاج تحميل الـ classifier المطلوب، ثم نحمل صورة الدخل (أو الفيديو) في المستوي اللوني الرمادي grayscale.

```
import numpy as np
import cv2
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
img = cv2.imread('sachin.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

الآن سنوجد الوجه في الصورة. إذا كان الوجه موجود سيتم إعادة موضع اكتشاف الوجه كمستطيل بالصيغة Rect(x,y,w,h). وعندما نحصل على الموضع يمكننا إنشاء ROI (المنطقة المهمة) ، ونبحث ضمن هذه البقعة التي تم اكتشاف الوجه فيها عن العين (لأن العين دائما موجودة في الوجه، مما يعطينا دقة في البحث).

شاهد النتيجة:



شرح الكود:

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
```

نحمل المصنف الذي يحتوي على مجموعة الصفات المميزة لوجه الأنسان (من الأمام) ونخزنه في التابع الذي أنشأناه وهو face_cascade، ونحمل أيضا المصنف الذي يحتوي على مجموعات الصفات المميزة للعين ونخزنها في التابع eye_cascade.

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

حتى نكتشف الوجه في OpenCV نستخدم طريقة method تدعى detectMultiScale والتي تستخدم لاكتشاف الأجسام في الصورة. لاكتشاف وجود الوجه في الصورة سنمرر للتابع detectMultiScale () أربع متغيرات وهي:

- Image: الصورة التي سنبحث ضمنها عن الوجه (gray).
- scaleFactor: عامل القياس للهرم المستخدم لاكتشاف وجه الإنسان (image pyramid).

إذا زدنا قيمة الـ scaleFactor ستزداد سرعة المكتشف، ولكن يمكن أن توثر على دقة الاكتشاف.

- (could harm our true-positive detection accuracy)
- وكلما أنقصنا من من قيمة عامل قياس الهرم (scaleFactor) ستنخفض سرعة المكتشف، ولكن ستزيد دقة المكتشف. ربما أيضا يزيد معدل اكتشاف الصور الكاذبة. لمزيد من المعلومات راجع فقرة sA note on Haar cascade الموجودة على الرابط التالي
- http://www.pyimagesearch.com/2016/06/20/detecting-cats-in-images-with-opencv/

- minNeighbors: للتحكم بأقل عدد من المستطيلات المحيطة المكتشفة في منطقة معينة بالنسبة للمنطقة التي يمكن ان تعتبر وجه انسان. هذا البارمتر مفيد جدا للتخفيف من معدل اكتشاف الصور الكاذبة.
- (pruning false-positive detections)
- minSize: أقل طول وعرض مسموح للمستطيل المحيط المكتشف للوجه (مثال 75 x 75) (في الكود السابق لم نستعمل هذا المتغير).

التابع () detectMultiScale يعيد مستطيل بالشكل (إحدثيات x,y للزأوية العليا اليسرى للمستطيل، وطوله وعرضه).

بعض الملاحظات على خوارزمية Haar cascades:

هذه الخوارزمية قادرة على اكتشاف الأجسام في الصورة بغض النظر عن موقعها وحجمها، وأيضا يمكنها الكشف عن الأجسام في الوقت الحقيقي في الأجهزة الحديثة.

مشاكل مصنف هار Haar cascades:

هناك مشكلة في مصنفات هار (Haar cascades) فعند تكبير قيمة scaleFactor ستصبح أسرع عملية اكتشاف الجسم ولكن المشكلة بأن الدقة تنخفض، والمشكلة الأخرى عند تقليل قيمة scaleFactor يصبح مكتشف القياس أبطأ وتزيد الدقة ولكن بالمقابل تزيد عدد الصور الكاذبة المكتشفة (أي الصورة التي لا تمثل وجه).

لذلك عند ضبط البارمترات للتابع () detectMultiScale سنحتاج الكثير من الوقت، وقد تحتاج لضبط جديد عند اختلاف الصورة. بسبب هذه المشاكل يتم تطبيق Histogram of Oriented Gradients + Linear SVM detection بدلا منها.

HOG + Linear SVM أسهل في ضبط المتغيرات وأفضل، وتتميز بأن نسبة اكتشاف الصور الكاذبة قليل جدا. ولكن هناك سيئة واحدة فيها وهي صعوبة الحصول على النتيجة في الوقت الحقيقي (Real-time) بسبب تطبيق HOG +Linear SVM معا.

مراجع إضافية:

1. **Video Lecture on Face Detection and Tracking**
2. **An interesting interview regarding Face Detection by Adam Harvey**
3. **http://www.pyimagesearch.com/2016/06/20/detecting-cats-in-images-with-opencv/**

- **خطوات تدريب الخوارزمية للتعرف على الشكل المطلوب:**

في الخطوات التالية سأشرح طريقة إنشاء سلسلة من المصنفات (classifiers) اعتمادا على الميزات مثل ميزة هار Haar، والتي تعد التقنية الأكثر شيوعا في الرؤية الحاسوبية لاكتشاف الوجه والعين. كل الأدوات المطلوبة والصور الموجبة والسالبة موجودة على الرابط:

https://www.cs.auckland.ac.nz/~m.rezaei/Tutorials/Haar-Training.zip

خطوات تدريب الخزارزمية لإنشاء classifier:

- جمع صور التدريب الموجبة والسالبة.
- تعليم الصور الموجبة باستخدام الأداة objectmarker.exe أو ImageClipper.
- إنشاء ملف .vev (vector) اعتمادا على الصور الموجبة المعلمة اعتماد على الأداة createsamples.exe.
- تدريب المصنف باستخدام haartraining.exe.
- تشغيل المصنف باستخدام الأداة cvHaarDetectObjects().

الخطوة ١: جمع قاعدة بيانات الصور

سنقوم بجمع حوالي ٢٠٠ صورة إيجابية (موجود فيها الجسم المطلوب) و٢٠٠ صورة سلبية (لا يوجد فيها الجسم). يمكنك الحصول على الصور إما من الأنترنت أو من خلال تصوير عدة فيديوات وتقسيمها لصور.

الصور الموجبة تعني الصورة الصورة التي تحوي على الجسم (على سبيل المثال الوجه أو العين)، والصور السلبية هي التي لا تتضمن الجسم.وجود عدد كبير من الصور الإيجابية والسلبية (الخلفية) يعطينا دقة أكبر في المصنف(classifier).

الخطوة ٢: ترتيب الصور السلبية

ضع صور الخلفية ضمن مجلد Haar Training\training\negative وشغل الملف create_list.bat

```
dir /b *.jpg >bg.txt
```

بعد تشغيل هذا الملف سنحصل على ملف نصي باسم bg.txt يحوي على أسماء الصور كما يلي:

```
image1200.jpg
image1201.jpg
image1202.jpg
...
```

سنحتاج هذا الملف النصي الذي يحوي على بيانات الصور السلبية لتدريب المصنف.

الخطوة ٣: القص & تعليم الصور الموجبة

في هذه الخطوة سنحتاج لإنشاء ملف بيانات (vector file) يحوي على أسماء الصور الموجبة بالإضافة لموضع الجسم في كل صورة. يمكننا إنشاء هذا الملف عن طريق أداتين هما:

Objectmarker أو Image Clipper.

أداة Objectmarker أكثر بساطة وأسرع أما الأداة الثانية Image Clipper نوعا ما أكثر مرونة ولكن ستأخذ معك وقت أكبر عند تعليم الأجسام في الصور.

هنا سنستخدم أداة Objectmarker.

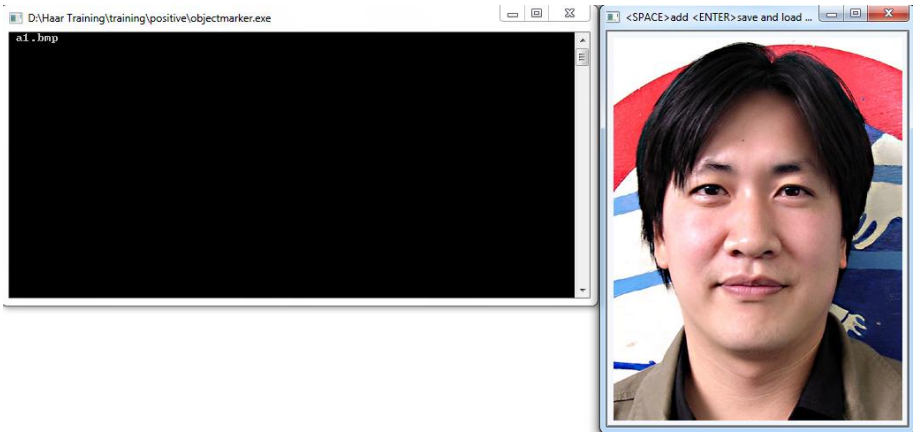
ضع الصور الموجبة ضمن المجلد \training\positive\rawdata...

ضمن المجلد \training\positive.... هناك ملف أسمه objectmaker.exe سنحتاجه لتعليم الأجسام في الصورالموجبة.

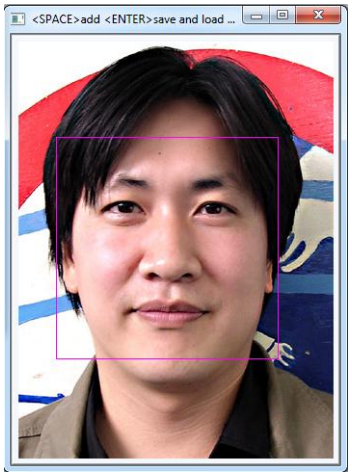
لكي يعمل الملف objectmaker.exe تأكد من وجود الملفين cv.dll و highgui.dll ضمن مسار الملف.

قبل تشغيل objectmaker.exe تأكد بأنك مرتاح ولديك وقت كافي لتعلم بعناية الأجسام لعشرات أو مئات الصور الموجبة.

كيف ستعلم الجسم؟ شغل الملف objectmaker.exe ستشاهد نافذتين كما في الأسفل: نافذة لعرض الصورة ونافذة لعرض اسم الصورة.

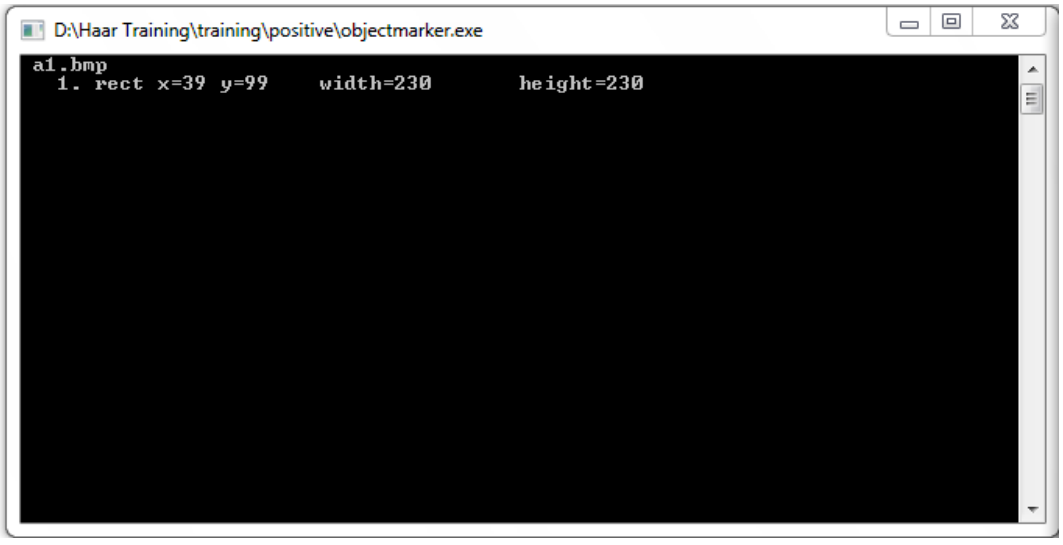


١) انقر على الزأوية العليا لمنطقة الجسم (مثل الوجه) وأمسك المأوس حتى تعلم منطقة الجسم.
٢) ستشاهد مستطيل حول الجسم. إذا لم يعجبك التحديد اضغط أي زر لإزالة التحديد (ما عدا زر Space وEnter)



ملاحظة هامة: عند رسم المستطيل المحيط بالجسم اهتم أن تبدأ برسم المربع إما من الزأوية اليسرى العليا أو من الزأويا السفلى اليمنى. إذا بدأت الرسم من الزأويتين الأخريتين فلن يكتب إحداثيات الجسم المحدد ضمن الملف **info.txt** أي كأنك لم تحدد الجسم.

إذا رسمت المستطيل بشكل صحيح أضغط على الزر SPACE . عن ذلك سيظهر موضع المستطيل وقياسه على النافذة السوداء (كما هو واضح في الصورة)



أعد الخطوات من a إلى c إذا كان هناك عدة أجسام في الصورة (مثل عدة وجوه).

عندما تنتهي من تحديد الأجسام في الصورة الحالية أضغط الزر Enter.

كرر الخطوات من a إلى e حتى تنتهي من تعليم جميع الأجسام في الصور الموجبة.

إذا شعرت بأنك تعبت وتريد التوقف عن تعليم الصور اضغط زر ESC عندها سيتم إنشاء ملف نصي باسم info.txt.

تحذير: عندما تخرج من الأداة **ojectmaker** وتريد إكمال تعليم الصور الموجبة فيما بعد قم بحفظ الملف **info.txt** الحالي الذي أنشأته في مجلد أخر أو غير أسمه (في كل مرة تخرج منها من الأداة قم بتغيير اسم الملف إلى **info1.txt, info2.txt**). لأنه في كل مرة ستشغل فيها الأداة **ojectmaker.exe** سيتم الكتابة فوق الملف **info.txt** الحالي وستذهب كل البيانات التي عملتها. وبعد الأنتهاء من تعليم جميع الصور قم بدمج جميع بيانات الملفات النصية التي أنشأتها ضمن ملف نصي واحد باسم **info.txt**.

داخل الملف النصي info.txt ستشاهد بعض المعلومات مثل:

```
rawdata\image1200.bmp 1 34 12 74 24
rawdata\image1201.bmp 3 35 25 70 39 40 95 80 92 120 40 45 36
rawdata\image1202.bmp 2 10 24 90 90 45 68 99 82
```

أول رقم يمثل عدد الأجسام الموجودة في الصورة المعطاة، فعلى سبيل المثال في السطر الثاني الرقم ٣ يدل على أن هناك ثلاثة أجسام في هذه الصورة (كثلاثة وجوه). الأرقم الأربعة التالية الملونة بالأخضر تعطينا موضع أول جسم في الصورة (إحداثيات الزأويا العليا اليسرى للمستطيل , x= 35 y=25 وطول وعرض المستطيل Width=70 ,Hight=39) والأرقام الأربعة التالية المونة بالأحمر تمثل موقع لجسم الثاني، والأرقام الملونة بالأزرق تمثل الجسم الثالث، وهكذا ...

وجود خطأ على سبيل المثال كما في هذا السطر: **rawdata/d19.bmp 3 83 119 185 183** يمكن أن يؤدي إلى حدوث خطأ كبير غير ملحوظ عندما تدرب المصنف للتعرف على الشكل (فعند حصول الخطأ في سطر معين يمكن أن يؤدي لعدم قراءة الأسطر التي تلي الخطأ). قبل البدء في الخطوة التي تليها تأكد أن كل أسطر الملف **info.txt** صحيحة.

الخطأ في المثال السابق هو بالعدد ٣ فيجب أن يكون العدد ١. هذا النوع من الخطأ يمكن أن يحدث عندما تدمج ملفات info.txt معا.

الخطوة ٤: إنشاء ملف vector للصور الموجبة

ضمن المجلد \training\ . . هناك ملف باتش(batch) اسمه
samples_creation.bat

يختوي هذا الملف على الأوامر التالية:

```
createsamples.exe -info positive/info.txt -vec vector/facevector.vec -num  200 -w 24 -h 24
```

البارمترات الرئيسية هي:

-info positive/info.txt	**Path for positive info file**
-vec vector/facevector.vec	**Path for the output vector file**
-num 200	**Number of positive files to be packed in a *vector file***
-w 24	**Width of objects**
-h 24	**Height of objects**

ملف الباتش هذا يحمل الملف النصي info.txt ويحزم صور الأجسام في ملف vector باسم نختاره، مثلا facevector.vec. بعد تشغيل ملف الابتش سوف تحصل على ملف باسم facevector.vec داخل المجلد \training\vector..

لاحظ: لتشغيل الملف **creatsample.exe** ستحتاج لأن يكون ضمن مسار المجلد الملفات التالية
cv097.dll, cxcore097.dll, highgui097.dll, and libguide40.dll

الخطوة ٥: تدريب Harr (Haar-Training)

ضمن المجلد training\.. يمكنك تعديل الباتش haartraining.bat :

```
haartraining.exe -data cascades -vec vector/vector.vec -bg  negative/bg.txt  -npos 200 -nneg
200 -nstages 15 -mem 1024 -mode ALL -w 24 -h 24 –nonsym
```

-data cascades	**Path and for storing the cascade of classifiers**
-vec data/vector.vec	**Path which points the location of vector file**
-bg negative/bg.txt	**Path which points to background file**
-npos 200	**Number of positive samples ≤ no. positive bmp files**
-nneg 200	**Number of negative samples (patches) ≥ npos**
-nstages 15	**Number of intended stages for training**
-mem 1024	**Quantity of memory assigned in MB**
-mode ALL	**Look literatures for more info about this parameter**
-w 24 -h 24	**Sample size**
-nonsym	**Use this if your subject is not horizontally symmetrical**

القياس W- و H- في الباتش harrtraining.bat ينبغي أن يكون نفسه في الباتش sample-creation.bat.

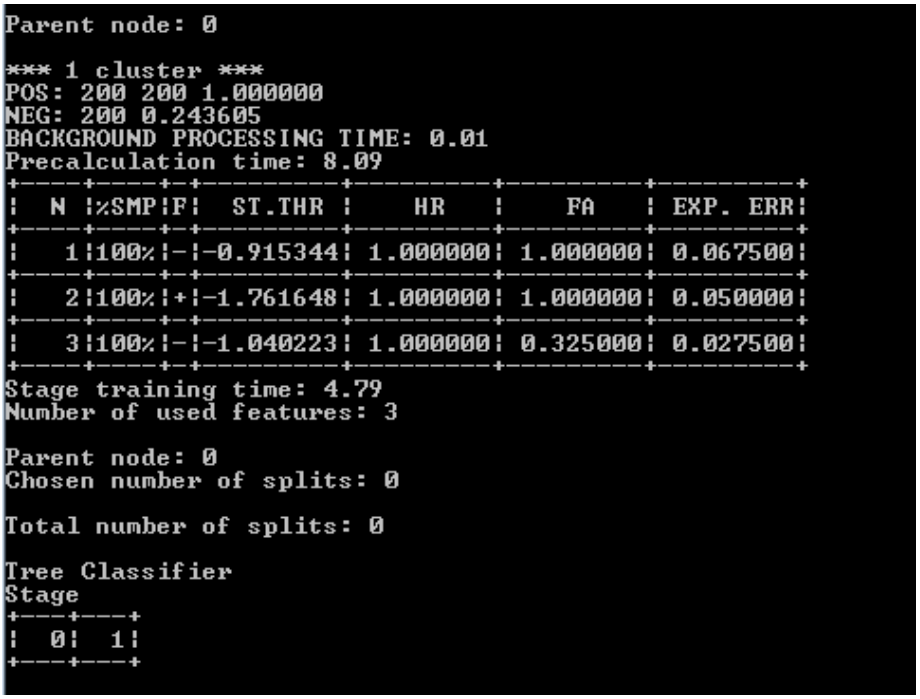
Harrtraining.exe يقو م بجمع مجموعة جديدة من العينات السلبية في كل مرحلة، و nneg- يحدد الحد الأقصى لحجم المجموعة. إنه يستخدم معلومات المرحلة السابقة ليحدد أي العينات مرشحة "candidate samples" حتى يتم تصنيفها بشكل صحيح. في نهاية التدريب عندما تصبح نسبة العينات التي تم تصنيفها بشكل صحيح من العينات المرشحة أقل من $FR^{stage\ no}$ لذلك :

لذلك مهما كان عدد المراحل (nstages) المحددة في haartraining.bat. البرنامج قد ينتهي باكرا (أي قبل نهاية عدد المراحل المحددة) إذا وصل للشرط السابق. عادة إنها إشارة جيدة لدقة عملية التدريب لدينا، ولكن أيضاً يمكن أن يحدث عندما تكون عدد الصور الموجبة ليس كافي (مثلا أقل من ٥٠٠).

لاحظ: لتشغل الملف **haartaining.exe** ستحتاج لأن يكون ضمن مسار المجلد الملفات التالية:

cv097.dll, cxcore097.dll, and highgui097.dll

عندما تشغل haartraning.bat ستشاهد بعض المعلومات المشابهة للصورة التالية:



المعطيات التي في الصورة هي للمرحلة الأولى من التدريب.

Parent node:	**Defines the current stage under training process**
N:	**Number of used features in this stage**
%SMP:	**Sample Percentage (Percentage of sample used for this feature)**
F:	**"+" if flipped (when symmetry applied) and "– "if not**
ST.THR:	**Stage Threshold**
HR:	**Hit Rate based on the stage threshold**
FA:	**False Alarm based on the stage threshold**
EXP.	**ERR: Exponential Error of strong classifier**

الصورة التالية تعرض المرحلة العاشرة 10^{th} من مراحل تدريب المصنف.

```
Parent node: 9

*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 0.000574246
BACKGROUND PROCESSING TIME: 2.60
Precalculation time: 7.99
+----+----+-+---------+---------+---------+---------+
|  N |%SMP|F|  ST.THR |    HR   |    FA   | EXP. ERR|
+----+----+-+---------+---------+---------+---------+
|   1|100%|-|-0.554502| 1.000000| 1.000000| 0.207500|
+----+----+-+---------+---------+---------+---------+
|   2|100%|+|-0.883580| 1.000000| 1.000000| 0.180000|
+----+----+-+---------+---------+---------+---------+
|   3|100%|-|-1.647806| 1.000000| 1.000000| 0.122500|
+----+----+-+---------+---------+---------+---------+
|   4| 83%|+|-1.357607| 1.000000| 0.785000| 0.095000|
+----+----+-+---------+---------+---------+---------+
|   5| 91%|-|-1.956339| 1.000000| 0.810000| 0.100000|
+----+----+-+---------+---------+---------+---------+
|   6| 76%|+|-1.634170| 1.000000| 0.630000| 0.055000|
+----+----+-+---------+---------+---------+---------+
|   7| 73%|-|-1.235653| 1.000000| 0.435000| 0.057500|
+----+----+-+---------+---------+---------+---------+
Stage training time: 10.40
Number of used features: 7

Parent node: 9
Chosen number of splits: 0

Total number of splits: 0

Tree Classifier
Stage
+---+---+---+---+---+---+---+---+---+---+---+
|  0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10|
+---+---+---+---+---+---+---+---+---+---+---+
```

الخطوة ٦: إنشاء ملف الـ XML

بعد الانتهاء من مراحل تدريب Harr ستجد ضمن المجلد /training/cascades/.. مجموعة من المجلدات مرقمة من الرقم ٠ إلى رقم المراحل التي تم تحديدها ضمن الملف haartraining.bat.

في كل مجلد من المجلدات السابقة يجب أن يكون هناك ملف نصي باسم AdaBoostCARTHaarClassifier.txt. قم بنسخ جميع المجلدات إلى المجلد /cascade2xml/data/..

الأن يجب علينا جميع المراحل التي تم إنشائها (المصنفات) ضمن ملف واحد XML والذي سيكون الملف النهائي.

شغل ملف الباتش convert.bat الموجود ضمن المجلد /cascade2xml/..

يتضمن ملف الباتش هذا التعليمات التالية:

haarconv.exe data myfacedetector.xml 24 24

myfacedetecor.xml هو اسم ملف الخرج والرقمين 24 24 يحددان W و H.

الأن أصبح لدينا ملف ال XML الخاص بنا، والذي يمكنك من خلاله اكتشاف الأجسام.

تطبيقات عملية

قد يسأل أحدكم بأننا تعلمنا معالجة الصور فكيف نستطيع الآن ربطها مع الأجهزة الإلكترونية؟
هناك فكرة يجب أن تفهمها حتى تستطيع أن تربط أي مشروع معالجة صورة مع العناصر والأجهزة الألكترونية وهي:

عندما تستطيع رسم مستطيل أو دائرة في مكان الجسم المطلوب البحث عنه فأنت تعرف إذا إحداثيات (X,Y) لمكان وجود الجسم على الشاشة، فإذا كل ما تحتاجه هو تمرير هذه الإحداثيات للجهاز الكهربائي أو الإلكتروني كمحرك السيرفو مثلا، فيتم تحريك السيرفو بناءاً على هذه الإحداثيات. التطبيقات التالية توضح بعضا من هذه الأفكار.

مثال: ملاحقة كرة

سنتعلم في هذا التطبيق طريقة التحكم بمحركين يتحكمان بكاميرا لملاحقة كرة عن طريق لونها.
الهدف من التطبيق هو فهم كيفية استخدام مكتبة OpenCv للتحكم بالعناصر الكهربائية والإلكترونية. مخططات المشروع والكود والصور متوفرة على الرابط:

https://github.com/MicrocontrollersAndMore/Raspberry_Pi_2_and_OpenCV_3_Tutorial_Part_1

الكود متوفر على الرابط:
https://github.com/MicrocontrollersAndMore/Raspberry_Pi_2_and_OpenCV_3_Tutorial_Part_1/blob/master/pan_and_tilt_tracker.py

الموقع التالي يحتوي على العديد من أفكار المشاريع التي يمكنك تنفيذها:
/https://www.intorobotics.com/20-hand-picked-raspberry-pi-tutorials-in-computer-vision

المراجع الرئيسية

- **https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html**
- **http://docs.opencv.org/master/d6/d00/tutorial_py_root.html**
- **http://docs.opencv.org/2.4/index.html**
- **http://docs.opencv.org/3.0-beta/modules/refman.html**
- **http://opencvpython.blogspot.com/**
- **http://www.pyimagesearch.com/**
- **http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html**
- **http://alereimondo.no-ip.org/OpenCV/34**
- **https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/**
- **http://breakinpointteam.blogspot.com/2015/12/opencv.html**
- **http://destovskyson.blogspot.com/**
- كتاب راسبيري باي ببساطة للمهندس عبد الله علي عبد الله
- **http://breakinpointteam.blogspot.com/2015/12/opencv.html**
- **https://www.youtube.com/playlist?list=PLd3hlSJsX_Imk_BPmB_H3AQjFKZS9XgZm**
- **https://www.youtube.com/user/UCFCRCV/featured**