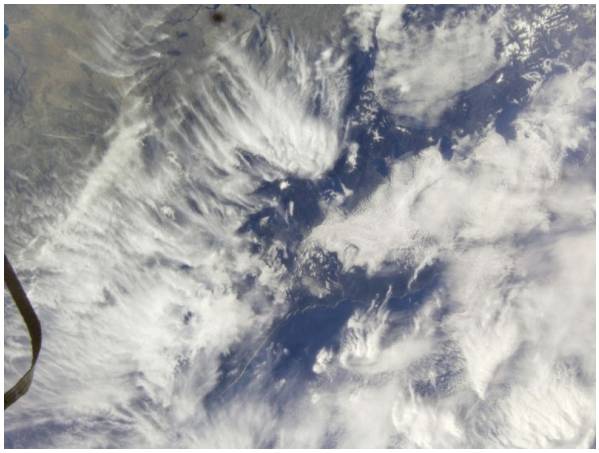
Projets

Calculer la vitesse de l'ISS à partir de photos

Calculer la vitesse de l'ISS à partir de données-image

Étape 1 Introduction

Dans ce projet, vous allez utiliser des photos prises par un ordinateur de vol Astro Pi embarqué sur la Station spatiale internationale (ISS) pour estimer la vitesse à laquelle l'ISS tourne autour de la Terre.

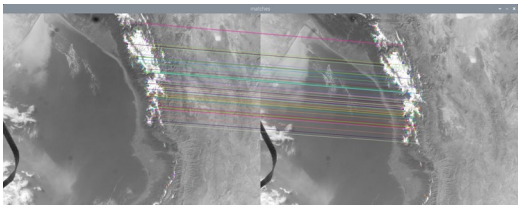


> Le challenge européen Astro Pi offre aux jeunes l'opportunité de conduire des expériences scientifiques dans l'espace en codant un programme informatique exécuté sur les ordinateurs Raspberry Pi installés à bord de la Station spatiale internationale (ISS).

Vous allez :

- Extraire des données EXIF à partir d'images
- Utiliser OpenCV pour calculer la distance entre des caractéristiques similaires sur
- deux images Calculer la vitesse de l'ISS

L'image ci-dessous montre deux photos prises depuis l'ISS, avec des lignes qui relient des caractéristiques similaires. En mesurant la distance en pixels entre les caractéristiques qui se sont déplacées, vous pouvez calculer la vitesse de déplacement de la caméra, et donc déterminer la vitesse de déplacement de l'ISS.



Pour ce projet, vous aurez besoin :

Matériel

- D'un ordinateur capable de faire fonctionner Python ou d'un navigateur Internet et d'un accès à repl.it (https://replit.com)

Logiciel

- De Thonny : ce projet peut être réalisé à l'aide de l'éditeur de code Python Thonny, qui peut être installé sous Linux, Windows ou Mac.

Installer Thonny

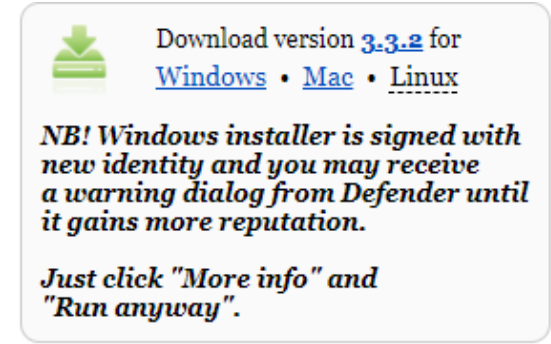
Installer Thonny sur un Raspberry Pi

- Thonny est déjà installé sur Raspberry Pi OS, mais peut avoir besoin d'être mis à jour pour avoir la dernière version.
- Ouvrez une fenêtre du terminal en cliquant sur l'icône située en haut à gauche de l'écran ou en appuyant simultanément sur les touches Ctrl+Alt+T.
- Dans la fenêtre, tapez la commande suivante pour mettre à jour votre système d'exploitation et Thonny :

```
sudo apt update && sudo apt upgrade -y
```

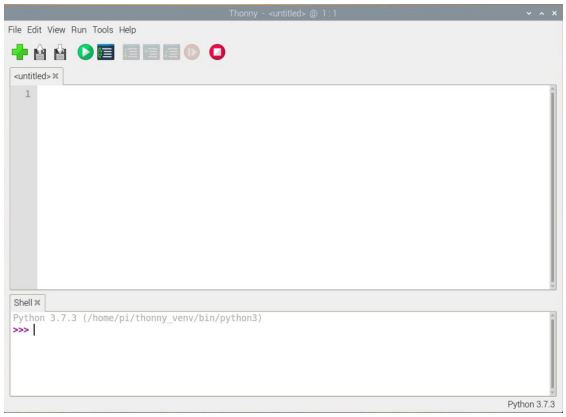
Installer Thonny sur d'autres systèmes d'exploitation

- Sous Windows, macOS et Linux, vous pouvez installer la dernière version de l'IDE Thonny ou mettre à jour la version existante. Dans un navigateur Web, accédez à **thonny.org** (https://thonny.org/)
- Dans le coin supérieur droit de la fenêtre du navigateur, vous verrez des liens de téléchargement pour Windows et macOS, et les instructions pour Linux.
- Téléchargez les fichiers appropriés et exécutez-les pour installer Thonny.



Ouvrir Thonny

Ouvrez Thonny depuis votre lanceur d'application. Vous devriez obtenir une fenêtre qui ressemble à ceci :



Vous pouvez utiliser Thonny pour écrire du code Python standard. Saisissez ce qui suit dans la fenêtre principale, puis cliquez sur le bouton **Run** (Exécuter) (il vous sera demandé d'enregistrer le fichier).

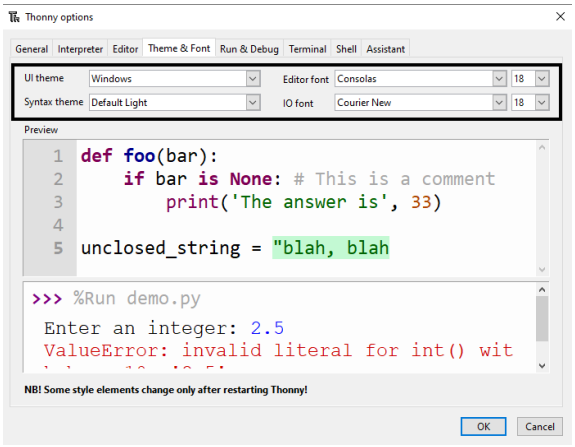
```
print('Hello World!')
```

Modifier le thème et la police dans Thonny

Thonny vous permet de modifier le thème et la police du logiciel. Cette fonctionnalité permet d'augmenter la taille de la police et de modifier les couleurs de l'arrière-plan et du texte en fonction de vos besoins.

Pour modifier le thème et la police :

- Cliquez sur Tools (Outils) -> Options.
- Cliquez sur l'onglet Theme & Font (Thème et Police).
- Cliquez sur les cases déroulantes pour chaque option jusqu'à ce que vous trouviez les paramètres qui correspondent le mieux à vos besoins.



- Cliquez sur OK lorsque vous avez terminé.

Attention : Privilégiez les polices simples et claires. Si vous utilisez une police de type écriture manuelle, cela peut rendre la lecture et le débogage difficiles.

Étape 2 Utiliser les données Exif pour trouver un décalage temporel

> Le format de fichier image échangeable (Exif, pour *exchangeable image file format*) est une norme qui définit les formats des fichiers image et son, et les différentes balises qui peuvent être stockées dans le fichier. Ces balises peuvent inclure le moment où la photo a été prise, l'endroit où elle a été prise et le type de caméra utilisé.

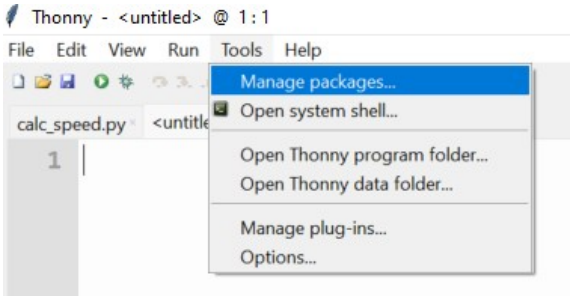
Pour recueillir des données Exif à partir d'une photographie, vous avez besoin d'une bibliothèque Python appelée `exif`.

Ouvrez l'IDE Python Thonny et cliquez sur **Tools (Outils) > Manage packages (Gérer les paquets),** puis recherchez et installez la bibliothèque exif.

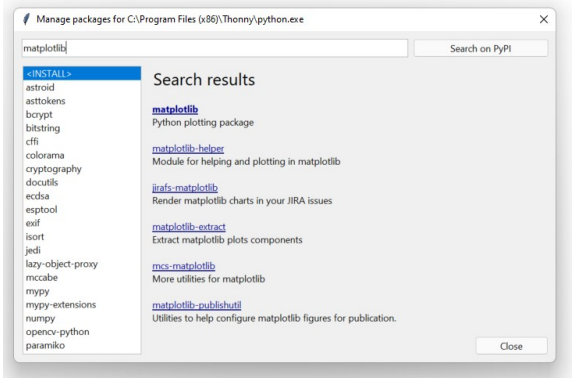
Installer des paquets Python avec Thonny

Installer des paquets Python avec Thonny

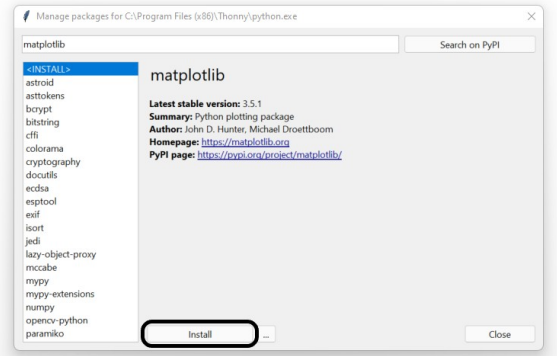
- Ouvrez Thonny depuis votre menu d'applications.
- Cliquez sur **Tools** (Outils) dans la barre de menu, puis sélectionnez **Manage packages** (Gérer les paquets).



- Utilisez la zone de recherche pour trouver le paquet que vous recherchez.



- Sélectionnez le paquet que vous souhaitez installer, puis cliquez sur le bouton **Install** (Installer).



- Une fenêtre s'ouvre et affiche l'avancement de l'installation de votre paquet. Une fois l'installation terminée, le paquet sera disponible dans vos programmes et vous pourrez l'utiliser.

Dans l'éditeur de code principal, ajouter les deux lignes de code suivantes. iss_speed.py

```
```

Enregistrez votre fichier sous `iss_speed.py`.

Vous aurez besoin de quelques images prises depuis l'unité Astro Pi embarquée à bord de l'ISS. Vous pouvez télécharger un fichier zip des photos en cliquant sur **ce lien** (https://rpf.io/p/en/astropi-iss-speed-go). Une fois que les photos ont été téléchargées, faites un clic droit sur le dossier dans vos **Téléchargements** et décompressez le dossier. **Ensuite, déplacez les photos à l'endroit où vous avez enregistré votre script Python.**

Sous vos `lignes importées`, créez une fonction pour déterminer le moment auquel une photo a été prise. Il faudra un argument, qui sera le nom du fichier de la photo.

iss_speed.py

```
from exif import Image
from datetime import datetime


def get_time(image):
```

L'image doit être ouverte puis convertie en un objet `Image` qui fait partie du paquet `exif` .

iss_speed.py

```
from exif import Image
from datetime import datetime


def get_time(image):
    with open(image, 'rb') as image_file:
        img = Image(image_file)
```

Vous pouvez jeter un œil à toutes les données Exif enregistrées dans le fichier image.

iss_speed.py

```
from exif import Image
from datetime import datetime


def get_time(image):
    with open(image, 'rb') as image_file:
        img = Image(image_file)
        for data in img.list_all():
            print(data)
```

Vous pouvez tester votre fonction en utilisant le nom de l'une des images que vous avez téléchargées. Cela doit se faire sous la forme d'une chaîne.

iss_speed.py

```
from exif import Image
from datetime import datetime


def get_time(image):
    with open(image, 'rb') as image_file:
        img = Image(image_file)
        for data in img.list_all():
            print(data)


get_time('photo_0683.jpg')
```

Exécutez votre code, vous devriez obtenir un résultat qui ressemble à ceci :

```
image_width
image_height
make
model
x_resolution
y_resolution
resolution_unit
datetime
y_and_c_positioning
_exif_ifd_pointer
_gps_ifd_pointer
compression
jpeg_interchange_format
jpeg_interchange_format_length
exposure_time
exposure_program
photographic_sensitivity
exif_version
datetime_original
datetime_digitized
components_configuration
shutter_speed_value
brightness_value
metering_mode
flash
maker_note
flashpix_version
color_space
pixel_x_dimension
pixel_y_dimension
_interoperability_ifd_Pointer
exposure_mode
white_balance
gps_latitude_ref
gps_latitude
gps_longitude_ref
gps_longitude
```

Les données nécessaires à ce projet sont `datetime_original`. Elles peuvent être enregistrées sous forme de chaîne, puis doivent être converties en objet `datetime` de manière à pouvoir effectuer les calculs avec celui-ci.
iss_speed.py

```
from exif import Image
from datetime import datetime


def get_time(image):
    with open(image, 'rb') as image_file:
        img = Image(image_file)
        time_str = img.get("datetime_original")
        time = datetime.strptime(time_str, '%Y:%m:%d %H:%M:%S')
    return time


print(get_time('photo_0683.jpg'))
```

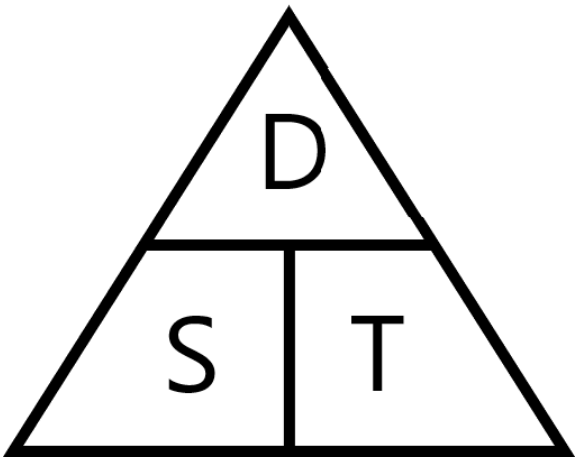
Lorsque vous exécutez ce code, vous devriez obtenir un résultat

```
2023-05-08 15:31:57
```

Enregistrez votre projet

Étape 3 Déterminer le décalage temporel entre deux photos

Vous pouvez calculer la vitesse de déplacement d'un objet en divisant la distance parcourue par la durée du déplacement.



Donc, pour calculer la vitesse de l'ISS à partir de photos, vous devez savoir combien de temps s'est écoulé entre le moment où chaque photo a été prise.

Supprimez l'appel à afficher (`print`) les résultats de votre fonction `get_time`. iss_speed.py

```
from exif import Image
from datetime import datetime


def get_time(image):
    with open(image, 'rb') as image_file:
        img = Image(image_file)
        time_str = img.get("datetime_original")
        time = datetime.strptime(time_str, '%Y:%m:%d %H:%M:%S')
    return time
```

Créez une nouvelle fonction appelée `get_time_difference`. Deux arguments sont nécessaires et ce seront les noms des fichiers des deux images.

iss_speed.py

```
def get_time_difference(image_1, image_2):
```

Utilisez votre fonction `get_time` pour obtenir les heures à partir des données Exif de chacune des deux images. iss_speed.py

```
def get_time_difference(image_1, image_2):
    time_1 = get_time(image_1)
    time_2 = get_time(image_2)
```

Faites la soustraction des deux heures et effectuez un test en les affichant (print). iss_speed.py

```
def get_time_difference(image_1, image_2):
    time_1 = get_time(image_1)
    time_2 = get_time(image_2)
    time_difference = time_2 - time_1
    print(time_difference)
```

Vous pouvez exécuter votre fonction en l'appelant avec deux noms d'image différents.

iss_speed.py

```
def get_time_difference(image_1, image_2):
    time_1 =  get_time(image_1) time_2
    = get_time(image_2) time_difference
    = time_2 - time_1
    print(time_difference)


get_time_difference('photo_0683.jpg', 'photo_0684.jpg')
```

Exécutez votre code et si vous avez utilisé les deux images ci-dessus, vous devriez obtenir un résultat qui ressemble à ceci :

La fonction doit renvoyer le temps en secondes, sous la forme d'un entier. Le paquet `datetime` permet de faire la conversion facilement.

iss_speed.py

```
def get_time_difference(image_1, image_2):
    time_1 = get_time(image_1)
    time_2 = get_time(image_2)
    time_difference = time_2 - time_1
    return time_difference.seconds
```

Pour tester votre code, vous pouvez `afficher (print)` le résultat de la nouvelle fonction.

iss_speed.py

```
13 def get_time_difference(image_1, image_2):
14     time_1 = get_time(image_1) time_2
15     = get_time(image_2)
16     time_difference = time_2 - time_1
17     return time_difference.seconds
18
19
20 print(get_time_difference('photo_1754.jpg',   'photo_1755.jpg'))
```

Votre résultat doit ressembler à ceci, en fonction des photos que vous avez choisies :

```
>>> 9
```

Enregistrez votre projet

Étape 4 Rechercher les caractéristiques concordantes

L'étape suivante consiste à trouver les caractéristiques concordantes sur les deux images. Pour cela, des algorithmes sont proposés dans OpenCV.

Supprimez l'instruction `print` de votre code.

iss_speed.py

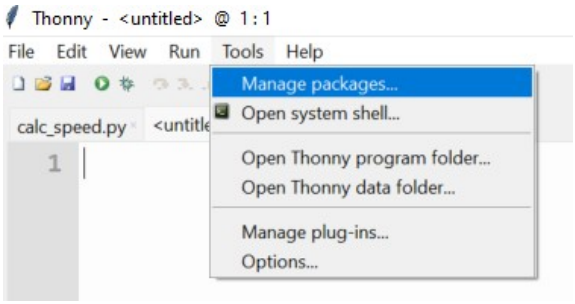
```
13 def get_time_difference(image_1, image_2):
14     time_1 = get_time(image_1)
15     time_2 = get_time(image_2)
16     time_difference = time_2 - time_1
17     return time_difference.seconds
```

Installez le paquet `opencv-python` dans Thonny.

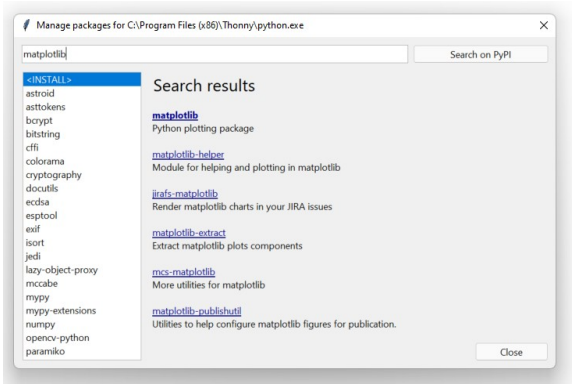
Installer des paquets Python avec Thonny

Installer des paquets Python avec Thonny

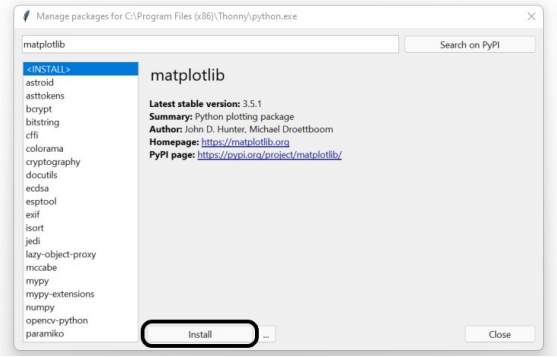
- Ouvrez Thonny depuis votre menu d'applications.
- Cliquez sur **Tools** (Outils) dans la barre de menu, puis sélectionnez **Manage packages** (Gérer les paquets).



- Utilisez la zone de recherche pour trouver le paquet que vous recherchez.



- Sélectionnez le paquet que vous souhaitez installer, puis cliquez sur le bouton **Install** (Installer).



- Une fenêtre s'ouvre et affiche l'avancement de l'installation de votre paquet. Une fois l'installation terminée, le paquet sera disponible dans vos programmes et vous pourrez l'utiliser.

Importez le paquet `cv2` et le paquet `math` intégré en haut de votre script.

iss_speed.py

```
from exif import Image
from datetime import datetime
import cv2
import math
```

Supprimez l'appel à `print(get_time_difference('photo_07464.jpg', 'photo_07465.jpg'))` de la ligne 22.

Les images doivent être converties en objets OpenCV afin de pouvoir les traiter. Ajoutez donc une fonction qui prend les deux images comme arguments, puis retourne ces objets.

iss_speed.py

```
def convert_to_cv(image_1, image_2):
    image_1_cv = cv2.imread(image_1, 0)
    image_2_cv = cv2.imread(image_2, 0)
    return image_1_cv, image_2_cv
```

Les objets OpenCV retournés peuvent maintenant être utilisés par d'autres classes et méthodes dans le paquet OpenCV. Pour ce projet, l'algorithme ORB (Oriented FAST and Rotated BRIEF) peut être utilisé. Cet algorithme permet de détecter les **points d'intérêt** sur une ou plusieurs images. Si les images sont similaires, les mêmes points d'intérêt doivent être identifiés dans chaque image, même si certaines caractéristiques ont été déplacées ou modifiées. ORB peut également assigner des **Descripteurs** aux points d'intérêt. Ceux-ci contiennent des informations sur le point d'intérêt, comme sa position, sa taille, sa rotation et sa luminosité. En comparant les descripteurs entre les points d'intérêt, on peut calculer les changements d'une image à l'autre.

Écrivez une fonction pour trouver les points d'intérêt et les descripteurs des deux images. Il faudra trois arguments : les deux premiers sont les objets image OpenCV, et le dernier est le nombre maximal de caractéristiques que vous souhaitez rechercher.

iss_speed.py

```
28  def calculate_features(image_1, image_2, feature_number):
29      orb = cv2.ORB_create(nfeatures = feature_number)
30      keypoints_1, descriptors_1 = orb.detectAndCompute(image_1_cv, None)
31      keypoints_2, descriptors_2 = orb.detectAndCompute(image_2_cv, None)
32      return keypoints_1, keypoints_2, descriptors_1, descriptors_2
```

Maintenant que vous avez les points d'intérêt et leurs descripteurs, ils doivent être mis en correspondance entre les deux images. Cela vous indiquera si un point d'intérêt de la première image correspond au même point d'intérêt sur la seconde image. La méthode la plus simple consiste à utiliser la force brute.

> Un algorithme de force brute signifie que l'ordinateur va essayer toutes les combinaisons possibles. C'est comme essayer de déverrouiller un téléphone protégé par un code PIN en commençant par `0000`, puis `0001` et ainsi de suite jusqu'à ce qu'il se déverrouille ou que vous arriviez à `9999`.

Dans ce contexte, la **force brute** signifie que vous prenez un descripteur de la première image et que vous essayez de l'associer à **tous** les descripteurs de la deuxième image. Une concordance sera trouvée ou non. Ensuite, vous prenez le deuxième descripteur de la première image et vous répétez le processus, et ainsi de suite jusqu'à ce que vous ayez comparé chaque descripteur de la première image à ceux de la deuxième image.

Écrivez une fonction qui prend les deux ensembles de descripteurs et tente de trouver des

concordances par force brute. iss_speed.py

```
35  def calculate_matches(descriptors_1, descriptors_2):
36      brute_force = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
37      matches = brute_force.match(descriptors_1, descriptors_2)
38      matches = sorted(matches, key=lambda x: x.distance)
39      return matches
```

Maintenant que vous avez vos concordances, vous pouvez exécuter toutes vos fonctions et examiner le résultat.

Attribuez les deux images que vous souhaitez utiliser et ajoutez des appels de fonction à la fin de votre script pour exécuter vos fonctions et afficher les concordances.

iss_speed.py

```
image_1 = 'photo_0683.jpg'
image_2 = 'photo_0684.jpg'


time_difference = get_time_difference(image_1, image_2) # Get time difference between images
image_1_cv, image_2_cv = convert_to_cv(image_1, image_2) # Create OpenCV image objects
keypoints_1, keypoints_2, descriptors_1, descriptors_2 = calculate_features(image_1_cv, image_2_cv, 1000) # Get keypoints and descriptors
matches = calculate_matches(descriptors_1, descriptors_2) # Match descriptors
print(matches)
```

Le résultat devrait ressembler à ceci :

```
[< cv2.DMatch 0x11b34cb30>, < cv2.DMatch 0x11b2db8b0>, < cv2.DMatch 0x11b2dbef0>...
```

Il s'agit d'une liste de concordances avec le point d'intérêt. Cependant, ce n'est pas très utile, mais à l'étape suivante, vous pouvez tracer les concordances sur les images et les visualiser.

Enregistrez votre projet

Étape 5 Afficher les caractéristiques concordantes

Les concordances peuvent être affichées sur les deux images, avec des lignes reliant chacun des points d'intérêt mis en correspondance.

Supprimez le dernier appel `print` à la fin de votre

script. iss_speed.py

```
time_difference = get_time_difference(image_1, image_2) # Get time difference between images
image_1_cv, image_2_cv = convert_to_cv(image_1, image_2) # Create OpenCV image objects
keypoints_1, keypoints_2, descriptors_1, descriptors_2 = calculate_features(image_1_cv, image_2_cv, 1000) # Get
keypoints and descriptors
matches = calculate_matches(descriptors_1, descriptors_2) # Match descriptors
```

Créez une fonction, en dessous de vos autres fonctions, qui prend pour arguments les deux objets image OpenCV, les points d'intérêt et les concordances.

iss_speed.py

```
def display_matches(image_1_cv, keypoints_1, image_2_cv, keypoints_2, matches):


time_difference = get_time_difference(image_1, image_2) # Get time difference between images
image_1_cv, image_2_cv = convert_to_cv(image_1, image_2) # Create OpenCV image objects
keypoints_1, keypoints_2, descriptors_1, descriptors_2 = calculate_features(image_1_cv, image_2_cv, 1000) # Get
keypoints and descriptors
matches = calculate_matches(descriptors_1, descriptors_2) # Match descriptors
```

Ensuite, tracez des lignes entre les points d'intérêt dont les

descripteurs correspondent. iss_speed.py

```
def display_matches(image_1_cv, keypoints_1, image_2_cv, keypoints_2, matches):
    match_img = cv2.drawMatches(image_1_cv, keypoints_1, image_2_cv, keypoints_2, matches[:100], None)
```

Les images peuvent maintenant être redimensionnées et affichées côte à côte sur votre écran, avec les lignes tracées entre les concordances.

iss_speed.py

```
def display_matches(image_1_cv, keypoints_1, image_2_cv, keypoints_2, matches):
    match_img = cv2.drawMatches(image_1_cv, keypoints_1, image_2_cv, keypoints_2, matches[:100], None)
    resize = cv2.resize(match_img, (1600,600), interpolation = cv2.INTER_AREA)
    cv2.imshow('matches', resize)
```

Pour terminer la fonction, le script doit attendre qu'une touche soit enfoncée, puis fermer l'image. iss_speed.py

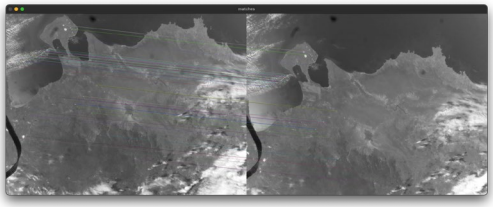
```
46  def display_matches(image_1_cv, keypoints_1, image_2_cv, keypoints_2, matches):
47      match_img = cv2.drawMatches(image_1_cv, keypoints_1, image_2_cv, keypoints_2, matches[:100], None)
48      resize = cv2.resize(match_img, (1600,600), interpolation = cv2.INTER_AREA)
49      cv2.imshow('matches', resize)
50      cv2.waitKey(0)
51      cv2.destroyWindow('matches')
```

Toutes ces fonctions doivent maintenant être appelées dans l'ordre, de sorte que vous puissiez voir le résultat. Au bas de votre script, ajoutez les lignes suivantes :

iss_speed.py

```
time_difference = get_time_difference(image_1, image_2) # Get time difference between images
image_1_cv, image_2_cv = convert_to_cv(image_1, image_2) # Create OpenCV image objects
keypoints_1, keypoints_2, descriptors_1, descriptors_2 = calculate_features(image_1_cv, image_2_cv, 1000) # Get keypoints and descriptors
matches = calculate_matches(descriptors_1, descriptors_2) # Match descriptors
display_matches(image_1_cv, keypoints_1, image_2_cv, keypoints_2, matches) # Display matches
```

Exécutez votre code et vous devriez obtenir une image comme celle ci-dessous. Cliquez dans la fenêtre et appuyez sur n'importe quelle touche pour quitter l'affichage d'image.



Enregistrez votre projet

Étape 6 Rechercher les coordonnées concordantes

Maintenant que les caractéristiques identiques sur chaque image ont été associées, vous devez récupérer les coordonnées de ces caractéristiques.

Créez une nouvelle fonction qui prend pour arguments les deux ensembles de points d'intérêt et la liste des concordances. iss_speed.py

```
54 def find_matching_coordinates(keypoints_1, keypoints_2, matches):
```

Créez deux listes vides pour stocker les coordonnées de chaque caractéristique mise en correspondance dans chacune des images. iss_speed.py

```
54 def find_matching_coordinates(keypoints_1, keypoints_2, matches):
55     coordinates_1 = []
56     coordinates_2 = []
```

La liste des concordances contient de nombreux objets OpenCV de chaque concordance pour chaque image. `match` . Vous pouvez itérer sur la liste pour trouver les coordonnées

Ajoutez une boucle `for` pour récupérer les coordonnées (`x1`, `y1`, `x2`, `y2`) de chaque concordance. iss_speed.py

```
54 def find_matching_coordinates(keypoints_1, keypoints_2, matches):
55     coordinates_1 = []
56     coordinates_2 = []
57     for match in matches:
58         image_1_idx = match.queryIdx
59         image_2_idx = match.trainIdx
60         (x1,y1) = keypoints_1[image_1_idx].pt
61         (x2,y2) = keypoints_2[image_2_idx].pt
```

Ensuite, ces coordonnées peuvent être ajoutées aux deux listes de coordonnées, et les deux listes peuvent être renvoyées. iss_speed.py

```
54  def find_matching_coordinates(keypoints_1, keypoints_2, matches):
55      coordinates_1 = []
56      coordinates_2 = []
57      for match in matches:
58          image_1_idx = match.queryIdx
59          image_2_idx = match.trainIdx
60          (x1,y1) = keypoints_1[image_1_idx].pt
61          (x2,y2) = keypoints_2[image_2_idx].pt
62          coordinates_1.append((x1,y1))
63          coordinates_2.append((x2,y2))
64      return coordinates_1, coordinates_2
```

Ajoutez un appel de fonction en bas de votre script pour stocker les données de sortie de la fonction. Ajoutez une ligne pour afficher la première paire de coordonnées de chaque liste, et exécutez votre programme. iss_speed.py

```
67  time_difference = get_time_difference(image_1, image_2) # Get time difference between images
68  image_1_cv, image_2_cv = convert_to_cv(image_1, image_2) # Create OpenCV image objects
69  keypoints_1, keypoints_2, descriptors_1, descriptors_2 = calculate_features(image_1_cv, image_2_cv, 1000) # Get
70  keypoints and descriptors
71  matches = calculate_matches(descriptors_1, descriptors_2) # Match descriptors
72  display_matches(image_1_cv, keypoints_1, image_2_cv, keypoints_2, matches) # Display matches
73  coordinates_1, coordinates_2 = find_matching_coordinates(keypoints_1, keypoints_2, matches)
    print(coordinates_1[0], coordinates_2[0])
```

Votre résultat devrait ressembler à ceci :

```
(666.8699340820312, 629.5451049804688) (661.8932495117188, 1062.512939453125)
```

Enregistrez votre projet

Étape 7 Calculer la distance des caractéristiques

Lorsque les coordonnées des caractéristiques concordantes sont enregistrées, il est possible de calculer la distance entre les coordonnées des caractéristiques concordantes. Il s'agira toutefois de la distance sur les images, donc il faudra convertir cette distance en kilomètres équivalents sur Terre, puis la diviser par l'écart de temps entre les photos, afin de calculer la vitesse.

Créez une fonction pour calculer la distance moyenne entre les coordonnées concordantes. Appelez-la `calculate_mean_distance`. Deux arguments sont nécessaires et ce seront les deux

listes de coordonnées. iss_speed.py

```
67  def calculate_mean_distance(coordinates_1, coordinates_2):
```

La fonction `zip` de Python prend les éléments des deux listes et les regroupe. Donc, l'élément 0 de la première liste est combiné à l'élément 0 de la deuxième liste. Ensuite, les premiers éléments de chacune des listes sont combinés. L'objet de liste zippé peut facilement être reconverti en une seule liste simple.

Commencez par créer une variable pour stocker la somme de toutes les distances entre les coordonnées et appelez-la `all_distances`. Ensuite, vous pouvez `zipper` les deux listes, puis reconvertir l'objet zippé en une liste.

iss_speed.py

```
67  def calculate_mean_distance(coordinates_1, coordinates_2):
68      all_distances = 0
69      merged_coordinates = list(zip(coordinates_1, coordinates_2))
```

Pour voir ce qui s'est passé ici, vous pouvez ajouter des appels `print` pour voir le détail des listes.

Ajoutez trois appels `print` pour voir un élément de `coordinates_1`, `coordinates_2` et `merged_coordinates`.

iss_speed.py

```
67  def calculate_mean_distance(coordinates_1, coordinates_2):
68      all_distances = 0
69      merged_coordinates = list(zip(coordinates_1, coordinates_2))
70      print(coordinates_1[0])
71      print(coordinates_2[0])
72      print(merged_coordinates[0])
```

Ajoutez un appel de votre fonction en bas de votre

```
81  average_feature_distance = calculate_mean_distance(coordinates_1, coordinates_2)
```

Lorsque vous exécutez le code, vous devriez obtenir un résultat

```
(666.8699340820312, 629.5451049804688)
(661.8932495117188, 1062.512939453125)
((666.8699340820312, 629.5451049804688), (661.8932495117188, 1062.512939453125))
```

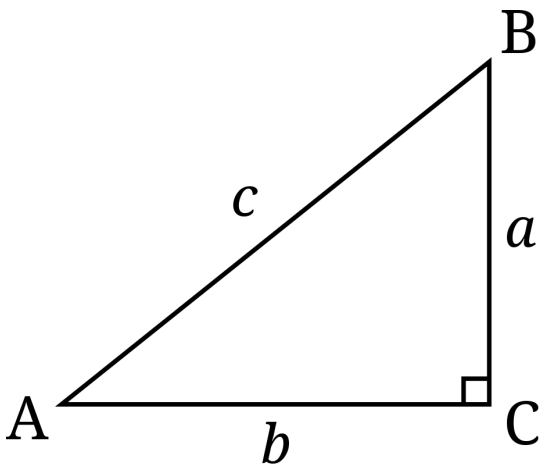
Normalement, vous voyez que les coordonnées `x` et `y` de chaque caractéristique de chaque image ont été combinées. Cela facilitera l'itération sur la nouvelle liste.

Supprimez les appels `print` et ajoutez une boucle `for` pour itérer sur les coordonnées fusionnées (`merged_coordinates`), et calculer les différences entre les coordonnées `x` et `y` de chaque image.

iss_speed.py

```
67  def calculate_mean_distance(coordinates_1, coordinates_2):
68      all_distances = 0
69      merged_coordinates = list(zip(coordinates_1, coordinates_2))
70      for coordinate in merged_coordinates:
71          x_difference = coordinate[0][0] - coordinate[1][0]
72          y_difference = coordinate[0][1] - coordinate[1][1]
```

Observez l'image suivante :



La distance entre les points **A** et **B** est la longueur de la ligne **c**. Il s'agit de l'hypoténuse.
L'hypoténuse (`hypot`) peut être calculée à l'aide du paquet `math`

Calculez la distance entre les deux points et ajoutez-la à la variable `all_distances`.

iss_speed.py

```
def calculate_mean_distance(coordinates_1, coordinates_2):
    all_distances = 0
    merged_coordinates = list(zip(coordinates_1, coordinates_2))
    for coordinate in merged_coordinates:
        x_difference = coordinate[0][0] - coordinate[1][0]
        y_difference = coordinate[0][1] - coordinate[1][1]
        distance = math.hypot(x_difference, y_difference)
        all_distances = all_distances + distance
```

Renvoyez la distance moyenne entre les caractéristiques en divisant `all_distances` par le nombre de caractéristiques concordantes, c'est-à-dire par la longueur de la liste `merged_coordinates`.

iss_speed.py

```
def calculate_mean_distance(coordinates_1, coordinates_2):
    all_distances = 0
    merged_coordinates = list(zip(coordinates_1, coordinates_2))
    for coordinate in merged_coordinates:
        x_difference = coordinate[0][0] - coordinate[1][0]
        y_difference = coordinate[0][1] - coordinate[1][1]
        distance = math.hypot(x_difference, y_difference)
        all_distances = all_distances + distance
    return all_distances / len(merged_coordinates)
```

Ajoutez un appel de fonction en bas de votre code pour calculer la distance moyenne, puis affichez le résultat.

iss_speed.py

```
time_difference = get_time_difference(image_1, image_2) # Get time difference between images
image_1_cv, image_2_cv = convert_to_cv(image_1, image_2) # Create OpenCV image objects
keypoints_1, keypoints_2, descriptors_1, descriptors_2 = calculate_features(image_1_cv, image_2_cv, 1000) # Get
keypoints and descriptors
matches = calculate_matches(descriptors_1, descriptors_2) # Match descriptors
display_matches(image_1_cv, keypoints_1, image_2_cv, keypoints_2, matches) # Display matches
coordinates_1, coordinates_2 = find_matching_coordinates(keypoints_1, keypoints_2, matches)
average_feature_distance = calculate_mean_distance(coordinates_1, coordinates_2)
print(average_feature_distance)
```

Lorsque vous exécutez votre code, vous devez obtenir une réponse comme « 504.08973470622516 ».

Enregistrez votre projet

Étape 8 Calculer la vitesse moyenne

Maintenant que les distances des caractéristiques ont été calculées pour les deux photos, il faut les convertir en distance réelle entre les caractéristiques sur Terre. Une fois ce calcul effectué, le décalage temporel entre les deux photos peut être utilisé pour calculer la vitesse de la caméra et, par conséquent, la vitesse de l'ISS.

Supprimez l'appel `print` en bas de votre script.

Créez une nouvelle fonction pour calculer la vitesse de l'ISS. Il convient d'utiliser la distance des caractéristiques (`feature_distance`), un facteur `GSD` et le décalage temporel (`time_difference`) comme arguments. iss_speed.py

```
78  def calculate_speed_in_kmps(feature_distance, GSD, time_difference):
```

Vous pouvez utiliser **ce site Web** (https://www.3dflow.net/ground-sampling-distance-calculator/) pour calculer l'échelle entre la distance en pixels et la distance sur Terre. La distance d'échantillonnage au sol (GSD) est exprimée en centimètres/pixels. Or, vous avez besoin de la distance en kilomètres et un kilomètre contient 100 000 cm. Si vous utilisez des photos d'une résolution différente de celle des exemples présentés ici, ou des photos prises avec une caméra différente, vous devrez recalculer le GSD.

Calculez la distance en multipliant la distance de la caractéristique en pixels par le `GSD`, puis en divisant le résultat par 100 000.

iss_speed.py

```
78  def calculate_speed_in_kmps(feature_distance, GSD, time_difference):
79      distance = feature_distance * GSD / 100000
```

La vitesse peut alors être calculée en divisant par le décalage temporel (`time_difference`) entre les deux images, et en renvoyant la vitesse.

iss_speed.py

```
78  def calculate_speed_in_kmps(feature_distance, GSD, time_difference):
79      distance = feature_distance * GSD / 100000
80      speed = distance / time_difference
81      return speed
```

Le site de la distance d'échantillonnage au sol (https://www.3dflow.net/ground-sampling-distance-calculator) donne un `GSD` de 12648 pour la caméra haute qualité embarquée sur l'ISS qui prend des photos de la même résolution que ces exemples (4056 x 3040). Si vous utilisez d'autres photos, vous devrez recalculer le GSD si elles ont une résolution différente ou si elles ont été prises avec une caméra différente.

Appelez votre fonction, puis affichez le résultat à la fin de votre programme. iss_speed.py

```
time_difference = get_time_difference(image_1, image_2) # Get time difference between images
image_1_cv, image_2_cv = convert_to_cv(image_1, image_2) # Create OpenCV image objects
keypoints_1, keypoints_2, descriptors_1, descriptors_2 = calculate_features(image_1_cv, image_2_cv, 1000) # Get keypoints and descriptors
matches = calculate_matches(descriptors_1, descriptors_2) # Match descriptors
display_matches(image_1_cv, keypoints_1, image_2_cv, keypoints_2, matches) # Display matches
coordinates_1, coordinates_2 = find_matching_coordinates(keypoints_1, keypoints_2, matches)
average_feature_distance = calculate_mean_distance(coordinates_1, coordinates_2)
speed = calculate_speed_in_kmps(average_feature_distance, 12648, time_difference)
print(speed)
```

Avec les deux images utilisées dans ce projet, une valeur de `7,084` est renvoyée, ce qui n'est pas loin de la vitesse de 7,66 km/s à laquelle l'ISS se déplace réellement.

Enregistrez votre projet

Étape 9 Améliorer votre programme

Pour affiner votre code et peut-être calculer une estimation plus juste de la vitesse de l'ISS, vous pouvez essayer d'étudier les éléments suivants :

- Utiliser un autre **algorithme de détection des caractéristiques dans OpenCV** (https://docs.opencv.org/3.4/db/d27/tutorial_py_table_of_contents_feature2d.html).
- Choisir un autre nombre de caractéristiques
- concordantes à utiliser. Utiliser plus de deux photos.
- Vérifier le temps d'écriture d'une photo sur le disque afin d'obtenir une valeur plus précise du temps
- écoulé entre les photos. La courbure de la Terre a-t-elle un effet sur les valeurs réelles de distance parcourue ?
- La hauteur de la caractéristique identifiée a-t-elle également un effet ?
- L'angle de déplacement (en diagonale sur l'image) a-t-il un impact qui doit être corrigé ?
  – Si vos caractéristiques concordantes sont des nuages, pouvez-vous compenser le fait qu'ils puissent aussi se déplacer ?

Étape 10 Et la suite ?