Belgium

esero

Fiche pour l'enseignant

Mission Space Lab

Phase 2 : Estimation de la vitesse de la Station Spatiale Internationale

Cette deuxième phase sert de guide pas à pas dans la réalisation d'un programme Python qui va réaliser une première estimation de la vitesse de l'ISS.

esa

belspo

La Scientothèque
L'ÉGALITÉ DES CHANCES PAR LES SCIENCES

ULB

Avec le soutien financier de

Mission Space Lab

Phase 2 : Estimation de la vitesse de la Station Spatiale Internationale

Guide de l'enseignant

La Scientothèque
L'ÉGALITÉ DES CHANCES PAR LES SCIENCES

ULB

Table des matières

Vue d'ensemble de l'activité

Public	Matières	Durée
S3 à S6	Informatique	1 à 2 périodes

Résumé

Dans cette activité, une introduction aux éléments les plus important du langage Python est présentée. Cela permet d'acquérir les notions minimales pour affronter les activités suivantes de Mission Space Lab.

Objectifs d'apprentissage

-

Matériel

- Ordinateur
- Accès à Internet
- Compte Google (Gmail ou autre) *(facultatif)*

Métiers STEM en lien

- Informaticien
- Chercheur
- Ingénieur
-

Auteurs : La Scientothèque (ESERO Belgium)

Date de publication : Septembre 2024

INTRODUCTION

Mission Space Lab (MSL)

Le projet Mission Space Lab (MSL), proposé par ESERO aux professeurs du secondaire, vise à éveiller l'intérêt des élèves pour les sciences spatiales et l'exploration de l'espace. Il s'agit d'une initiative concrète qui permet aux jeunes de se confronter à la recherche scientifique en conditions réelles, à bord de la Station spatiale internationale (ISS).

Ce projet poursuit plusieurs objectifs pédagogiques ambitieux :

- **Sensibiliser** les élèves aux merveilles de l'univers et aux enjeux de l'exploration spatiale.
- **Permettre** aux jeunes de réaliser des expériences scientifiques concrètes en microgravité, un environnement hors du commun.
- **Développer** leurs compétences en programmation, analyse de données et résolution de problèmes.
- **Favoriser** la collaboration et le travail en équipe pour mener à bien un projet scientifique complexe.

C'est pourquoi, le projet est séparée en trois phases distinctes :

1. **Une introduction à la programmation Python** : elle permet de construire des fondations solides pour entammer les prochaines phases tout en développant des compétences utiles.
2. **Une première estimation de la vitesse de l'ISS** : Cette activité propose un parcours guidé pour l'estimation de l'ISS permettant de construire une base solide pour la suite.
3. **Des activités d'approfondissement** : elles visent une meilleure compréhension de la problématiques par une étude sur papier et l'exploration de pistes de solutions pour l'amélioration de l'estimation de la vitesse de l'ISS

La deuxième phase et lien avec la dernière phase

Cette deuxième phase se concentre sert de guide pas à pas dans la réalisation d'un programme Python qui va réaliser une première estimation de la vitesse de l'ISS. Le code produit l'issue de cette phase servira de base pour les futures améliorations du code permettant ainsi de prendre part au challenge de la Mission Space Lab (MSL).

En fin du document de la fiche élève, quelques suggestions sont proposés en guise de piste d'amélioration du code. Ces améliorations peuvent utiliser le code proposé dans ce guide comme base. Pour participer au challenge MSL, il convient cependant de respecter quelques règles brièvement décrites plus loin dans la section *Déroulement du challenge MSL*.

La dernière phase propose de s'éloigner des considérations informatiques pour comprendre ce qui a été réalisé lors de cette deuxième phase par une modélisation théorique. Ainsi, la vitesse prédite par plusieurs modèles physiques sont comparés au modèle numérique. L'incertitude sur l'estimation est abordée comme outil d'amélioration pratique.

Outils mis à disposition

Il est possible de réaliser cette phase de deux manières différentes :

- **Avec Google Collab :** c'est un environnement de programmation en ligne accessible depuis un navigateur web et sans aucune installation. Son avantage principal est de permettre de mélanger texte et code. Le code de l'activité peut ainsi être progressivement écrit et testé dès qu'il est expliqué, évitant de d'éparpiller son attention sur plusieurs supports/environnements.
- **Avec Thonny :** c'est l'environnement de développement intégré (IDE) préconisé par l'ESA pour la réalisation de l'ensemble du projet. Il permet l'exécution de code localement sans connexion internet et propose un simulateur Astro Pi pour tester le code dans des conditions réalistes avant l'envoi à l'ESA. Une fiche pédagogique fourni les instructions

Caractéristique	Google Collab	IDE Thonny
Pas d'installation requise	✔	✘
Simulateur Astro Pi	✘	✔
Configuration non requise	✔	✘
Internet non requis	✘	✔
Fiche élève non requis	✔	✘
Compte Google non requis	✘	✔

Pour la deuxième phase, **il est conseillé de s'habituer plus proprement à l'interface Thonny** pour pouvoir se préparer à la réalisation du challenge qui en fera grand usage. Cependant, un notebook Collab est mis à disposition si votre objectif est de rapidement se familiariser avec la méthode d'estimation de la vitesse proposée avant de coder de manière plus conventionnelle.

⚠ *Pour les instructions relatives à l'installation et la configuration de Thonny, veuillez vous référer à la fiche du professeur de la première phase.*

Voici le lien vers le notebook Google Collab de cette deuxième phase :

https://colab.research.google.com/drive/1YYopymmfjVVfq4afH0j964zkTmrIrXP8?usp=sharing

Son utilisation est tout à fait identique à celui de la phase précédente. Pour une rapide introduction à Google Collab, veuillez consulter la fiche du professeur de la première phase.

⚠ *Dans le cadre du challenge Mission Space Lab, seuls les codes utilisant la liste limitée de modules autorisés pourront être acceptés et envoyés à la Station Spatiale Internationale. Une liste des modules interdits est listé ici.*

DÉROULEMENT DU CHALLENGE MSL

Présentation de l'Astro Pi

À bord de l'ISS, l'estimation de la vitesse sera réalisée par l'ordinateur Astro Pi sur base du code Python qui sera réalisé en classe lors de cette phase.

Les AstroPi sont des micro-ordinateurs RaspberryPi muni de toute une série de capteurs:

- Capteur de température
- Capteur de niveau d'humidité
- Un axéléromètre et gyroscope 3 axes
- Un capteur de pression
- Un capteur de luminosité et de couleur
- Un capteur infrarouge passif (PIR)

L'ensemble est emballé dans une coque personnalisée imprimée en 3D, assurant ainsi l'intégrité de l'ensemble face aux chocs et aux manipulations hasardeuses. Le micro-ordinateur RaspberryPi 4 dispose de capacités similaires à un ordinateur portable basique dans un espace très réduit. La puissance de calcul disponible est considérable et permet ainsi la réalisation d'opérations avancées comme le traitement complexe d'images ou l'intelligence artificielle. Cette puissance sera mise à contribution dans le cadre de ce projet.

Pour plus d'informations au sujet de l'Astro Pi, veuillez consulter les liens fourni plus loin.

Calendrier du challenge

Mission Space Lab se déroule en deux temps :

- ***Étape de création*** : écriture du programme qui va collecter les données nécessaires et réaliser les calculs. Pour être déployé, des exigences devront être respectées. Celles-ci sont listées plus loin. Il faut s'inscrire sur la page officielle pour participer.

- ***Déploiement*** : Le programme, s'il est conformes aux exigences, sera déployé sur la Station Spaciale Internationale. Les programmes ayant été sélectionnés se verront attribué du statut « *flight status* » et les équipes correspondantes seront notifiés.

Après le déploiement, les équipes ayant reçu le « *flight status* » reçevront un certificat de participation et l'ensemble des données capturés à bord de l'ISS. Ils seront de plus invités à une ***séance de Question&Réponses avec un astronaute de l'ESA***.

Quels sont les attendus du programme ?

- Votre programme doit produire une sortie numérique de la vitesse moyenne que l'ISS voyage. C'est ce que l'on appelle aussi l'amplitude de la vitesse.
- La sortie que votre programme produit doit être un fichier .txt.
- La production numérique ne devrait pas utiliser plus de 5 numéros significatifs (5 chiffres au total, y compris les virgules décimales, par exemple 1,2345 km/s).
- La vitesse doit être indiquée en kilomètres par seconde (km/s).

Voici un exemple de code enregistrant la vitesse dans le format adéquat dans un fichier :

```
vitesse = 7.1234567890  # Remplacer avec votre estimation

# Formattons vitesse pour qu'elle ait une précision
# de 5 chiffres significatifs
vitesse_formatte = "{:.4f}".format(vitesse)

# Créons une chaîne de caractère que l'on écrira dans le fichier
chaine_sortie = vitesse_formatte

# Ecrire dans le fichier
chemin_fichier = "result.txt"  # Replacer avec fichier voulu
with open(chemin_fichier, 'w') as file:
    file.write(chaine_sortie)

print("Données écrites dans ", chemin_fichier)
```

Votre programme devrait mesurer la vitesse linéaire moyenne que l'ISS voyage autour de la Terre, pas combien l'ISS tourne.

Liste des exigences du programme pour reçevoir le « flight status »

En vue du déploiement, le programme doit scrupuleusement se conformer à certaines exigences.

Prescriptions : Généralités

- Votre expérience ne repose pas sur l'interaction avec un astronaute.

- Votre programme est écrit en Python 3 et est nommé **main.py**. Il doit s'exécuter sans erreurs lorsqu'il est exécuté sur la ligne de commande du système d'exploitation de vol en utilisant python3 main.py.
- Votre programme ne s'appuie sur aucune autre bibliothèque que celles énumérées dans le guide du créateur du Laboratoire Space Lab (rpf.io/msimsl-creators).
- Votre programme surveille son temps d'exécution et s'arrête après 10 minutes.
- Il n'y a pas de mauvais langage, de grossièreté ou de déclarations politiques dans votre programme.
- Votre programme utilise au moins un capteur Sense HAT ou la caméra.
- Votre programme est téléchargé dans un fichier zip. Si vous avez des fichiers supplémentaires nécessaires au fonctionnement de votre expérience, ils peuvent également être inclus dans le fichier zip, mais le fichier zip doit contenir un fichier appelé **main.py**, qui doit être la façon dont votre programme est exécuté.
- Votre programme n'utilise pas la matrice LED d'Astro Pi.
- Votre programme n'affiche pas de drapeaux pendant sa course.
- Votre programme n'est pas autorisé à conserver plus de 42 images à la fin des 10 minutes, bien qu'il puisse stocker plus que cela pendant qu'il est en cours d'exécution.
- Votre programme zippé ne doit pas être supérieur à 3MB, à moins qu'il n'inclue un modèle d'apprentissage automatique TensorFlow Lite (.tflite), auquel cas votre programme zippé ne doit pas être supérieur à 7MB.

Prescriptions : Sécurité

- Votre programme est bien documenté et facile à comprendre, et il n'y a aucune tentative de cacher ou d'obscurcir ce que fait un morceau de code.
- Votre programme ne démarre pas un processus système, ou n'exécute pas un autre programme ou toute commande habituellement saisie sur le terminal (par exemple vcgencmd).
- Votre programme n'utilise pas la mise en réseau.
- Votre programme n'inclut pas le code malveillant (c'est-à-dire le code qui tente délibérément de perturber la fonctionnalité du système).

Exigences: Fichiers et fils

- Votre programme n'utilise pas de threads, ou s'il le fait, il ne le fait qu'en utilisant la bibliothèque de threading; les threads sont gérés avec soin et fermé proprement, et leur utilisation est clairement expliquée par les commentaires dans le code.
- Votre programme ne sauvegarde des données que dans le même dossier où votre fichier Python principal est, comme décrit dans le guide du créateur du Laboratoire Space Lab (c'est-à-dire en utilisant la variable spéciale 'file'); votre programme ne tente pas de créer de nouveaux répertoires pour stocker vos données, et aucun nom de chemin absolu n'est utilisé.
- Votre programme fonctionne sans erreurs et ne soulève aucune exception non manipulée.
- Tous les fichiers que votre programme crée ont des noms qui n'incluent que les lettres, les nombres, les points (.), les tirets (-), ou les soulignements (_).
- Votre programme n'utilise pas plus de 250 Mo de place pour stocker des données.
- En plus de contenir votre fichier **main.py**, le fichier zip que vous soumettez ne doit contenir que les types de fichiers suivants: .py, .main, .csv, .txt, .jpg, .png, .yuv, .json, .toml, .yaml, .tflite.
- En plus de votre fichier **result.txt**, la sortie de votre programme ne doit inclure que les types de fichiers suivants: .csv, .txt, .log, .jpg, .png, .yuv, .raw (camera), .h264, .json, .toml, .yaml.

Les programmes qui ne respectent pas cette liste de contrôle seront disqualifiés.

Pour plus d'informations sur le challenge, **veuillez consulter la page consacrée.**

Méthode automatique
Un script a été créé pour automatiser la vérification des exigences. Téléchargez les scripts check_requirements.sh et check_sensehat_function_usage.py sur ce site et placez les dans le dossier où se trouve votre fichier de code main.py. Exécutez dans un terminal du Raspberry Pi :

```
chmod +x check_remquirements.sh && ./check_requirements.sh
```

ensuite

```
python check_sensehat_functions_usage.py
```

Des messages s'afficheront si une exigence n'est pas respectée. Cette méthode ne fonctionne cependant nativement que sur Raspberry Pi, Mac ou Linux. Sur Windows, elle ne fonctionne pas nativement à moins d'installer Windows Subsystem for Linux par exemple.

Liens

Presque toutes les pages proposés ici sont uniquement en anglais actuellement.

Page officielle du challenge Mission Space Lab
https://astro-pi.org/mission-space-lab/

Guide du créateur
https://projects.raspberrypi.org/en/projects/mission-space-lab-creator-guide

Livre des règles
https://astro-pi.org/mission-space-lab/rulebook

Guide du mentor
https://astro-pi.org/mission-space-lab/mentor-guide

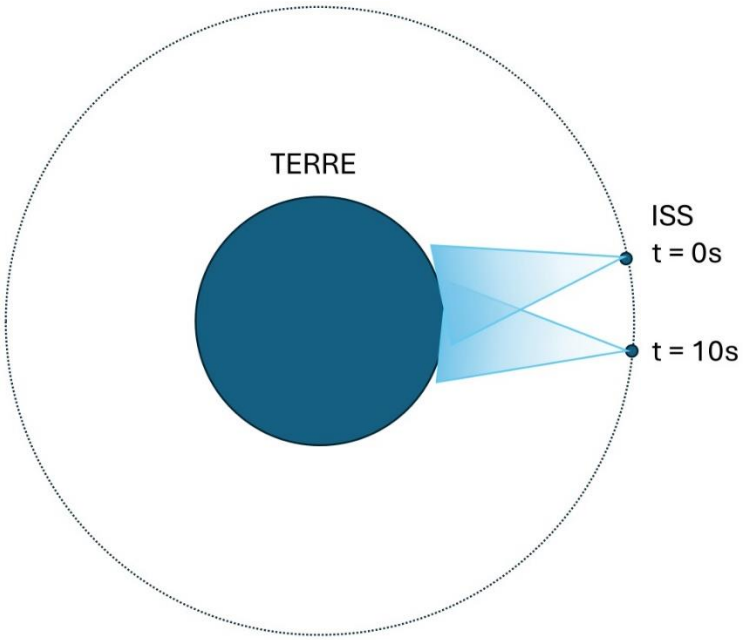
Débuter avec le module caméra
https://projects.raspberrypi.org/fr-FR/projects/getting-started-with-picamera

Liste des modules interdits
https://docs.google.com/spreadsheets/d/1EoVzgA8gOiDXsJ1k9dQBdPyFC8U3bXFca2dRmdKNbcI/edit?gid=0#gid=0

Débuter avec le SenseHAT
https://projects.raspberrypi.org/en/projects/getting-started-with-the-sense-hat

Détermination de la vitesse de l'ISS

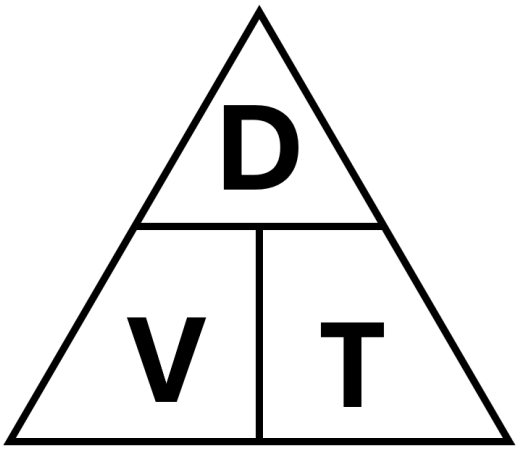
L'idée est la suivante : en partant de photos de la Terre prises depuis l'ISS, comment peut-on déterminer la vitesse de la station ?



Pour rappel, la vitesse est définie par

$$v = \frac{\Delta x}{\Delta t} = \frac{x_f - x_t}{t_f - t_i}$$

exprime que la vitesse est la distance parcourue sur un certain laps de temps. Elle peut aussi être visualisée par un diagramme mnémotechnique comme celui-ci :



où l'on voit clairement que la vitesse V est bien le rapport d'une distance D sur un temps T écoulé. Donc il faut trouver une façon de mesurer la distance parcourue par l'ISS avec des photos de la Terre prises depuis l'ISS. Pour y arriver, plusieurs étapes sont nécessaires :

1. Il faut pouvoir mesurer la durée écoulée entre deux photos ou déterminer le moment exact de prises de chaque image pour en déduire le laps de temps entre les deux.
2. Il faut pouvoir mesurer la distance parcourue sur l'image :
    a. Des caractéristiques communes entre les images doivent être identifiées et leurs coordonnées déduites.
    b. La distances entre les coordonnées des points doit être calculée.
    c. La distance obtenue sur l'image étant exprimée en pixels, il faudra le convertir en km.
3. La vitesse est calculée comme le rapport de la distance parcourue sur le laps de temps entre les deux images.

Ce tutoriel ne présente que les grandes lignes et l'estimation obtenue est déjà proche de la réalité. Il y a cependant encore beaucoup d'améliorations que l'on peut apporter pour affiner l'estimation de la vitesse. C'est là que tu entres en jeu !

1. Extraire les métadonnées des photos

De nos jours, les photos numériques prises sur les smartphones comme les caméras numériques professionnelles permettent d'intégrer dans les images des informations supplémentaires au-delà de la simple photo elle-même. Dans le passé, les appareils argentiques comme les très populaires Polaroid permettaient de produire une photo instantanément imprimée.

Si nous voulions nous rappeler de la date, l'heure ou la position exacte de la prise de la photo, il fallait se rappeler de le noter au dos de chaque photo. C'est un inconvénient qui disparaît avec les appareils numériques modernes qui enregistrent ces informations et plus encore sur les photos au moment de leur prise. C'est ce que l'on appelle des méta-données.

Dans l'éditeur de code, écrivons les lignes suivantes pour les importations qui nous seront utiles :

```
from exif import Image
from datetime import datetime
```

Enregistrons le fichier Python sous le nom `main.py` dans un lieu que l'on pourra facilement retrouver par la suite. Le

module `exif` contient `Image`, une fonction qui va nous permettre d'ouvrir l'image mais aussi d'accéder à ses méta-données. Le module `datetime` permet d'interpréter les informations de date et d'horaire et de les transformer. Prenons maintenant une photo déjà prise.

NOTE

Pour ce tutoriel, il est recommandé de **télécharger des exemples de photos prises depuis l'ISS ici** et décompresser le contenu de l'archive dans le répertoire où se trouve le fichier Python.

Définissons à présent une fonction qui va nous fournir la date et l'heure de prise d'une photo passée en argument. Dans le même fichier Python, écrivons après les importations :

```
def obtenir_temps(image):
```

La première étape consiste à ouvrir le fichier image :

```
def obtenir_temps(image):
  with open(image, 'rb') as fichier_image:
```

Ensuite, on lit le contenu du fichier avec la fonction `Image()` qui nous permettra d'extraire les méta-données:

```
def obtenir_temps(image):
  with open(image, 'rb') as fichier_image:
    img = Image(fichier_image)
```

`Image()` nous renvoie un objet que l'on a appelé `img` qui contient l'ensemble des information de l'image organisée d'une manière structurée et facile à utiliser. Dans un premier temps, on peut s'amuser à lister l'ensemble des méta-données à notre disposition. Elle nous est fournie sous forme de liste Python prête à l'emploi par la méthode `.list_all()`. Il nous suffit d'itérer sur chaque élément de la liste avec une boucle `for` et de l'afficher.

```
def obtenir_temps(image):
  with open(image, 'rb') as fichier_image:
    img = Image(fichier_image)
    for metadonnee in img.list_all():
      print(metadonnee)
```

Pour voir cette liste, il reste à exécuter la fonction en l'appelant avec le nom d'un des fichiers images téléchargées précédemment :

```
def obtenir_temps(image):
  with open(image, 'rb') as fichier_image:
    img = Image(fichier_image)
    for metadonnee in img.list_all():
      print(metadonnee)

obtenir_temps('photo_0683.jpg') # Le nom du fichier image est passée en argument
```

Nous obtenons le résultat suivant

```
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
```

On peut voir que les informations disponibles sont très nombreuses et variées et pourraient potentiellement être utiles pour la suite. Cependant, nous sommes ici uniquement intéressés par la date et le temps de prise original. La métadonnée `datetime_original` correspond à cette description. Pour extraire l'information, il faut utiliser la méthode `.get('nom_metadonnee')` en lui donnant le nom de la métadonnée voulue en argument. Pour la méta-donnée voulue, il nous renverra une chaîne de caractères que l'on renvoie en fin de fonction. Lors de l'appel à la fonction, on affiche le résultat avec `print()`. Remplaçons la boucle précédente comme suit :

```
def obtenir_temps(image):
  with open(image, 'rb') as fichier_image:
    img = Image(fichier_image)
    temps_str = img.get('datetime_original')
  return temps_str

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

La date et le temps sera alors fournie sous la forme d'une chaîne de caractères mais il aura un format inadapté pour nous. On utilise le module `datetime` pour changer son format sous la forme `année:mois:jour heure:minute:seconde` avec la méthode `.strptime()` comme suit:

```
def obtenir_temps(image):
  with open(image, 'rb') as fichier_image:
    img = Image(fichier_image)
    temps_str = img.get('datetime_original')
    temps = datetime.strptime(temps_str, '%Y:%m:%d %H:%M:%S')
  return temps

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

A l'exécution, le résultat suivant devrait s'afficher :

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

A toi de jouer à présent !

> NOTE
>
> N'oublie pas d'enregistrer ton projet.

1.1. Exercices

1. Crée à présent une fonction qui extraie les coordonnées GPS de latitude et longitude d'une photo et les affiche les sous la forme `latitude=59.23, longitude=13.63`.

La fonction est similaire à `obtenir_temps()`: il s'agit de sélectionner les méta-données `gps_latitude` et `gps_longitude`

```
def obtenir_coordonnees(image):
  with open(image, 'rb') as fichier_image:
    img = Image(fichier_image)
    latitude = img.get('gps_latitude')
    longitude = img.get('gps_longitude')
  print("latitude: ", latitude, ", longitude: ", longitude)
```

2. Déterminer le laps de temps entre les photos

A présent, pour calculer la différence de temps entre deux photos prises successivement, il suffit d'appeler la fonction `obtenir_laps_temps()` que l'on a créé sur les images et de calculer la différence de temps obtenue. On continue le code dans le même fichier Python à la suite du code précédent.

Une nouvelle fonction devra être créée pour obtenir le laps de temps. On le définit comme suit :

```
def obtenir_laps_temps(image1, image2):
```

Pour chaque image, on va extraire le temps :

```
def obtenir_laps_temps(image1, image2):
  temps1 = obtenir_temps(image1)
  temps2 = obtenir_temps(image2)
```

Finalement, la différence de temps est calculée et renvoyée :

```
def obtenir_laps_temps(image1, image2):
  temps1 = obtenir_temps(image1)
  temps2 = obtenir_temps(image2)

  difference_temps = temps2 - temps1
  return difference_temps
```

Pour l'exécuter, nous appelons la fonction que l'on vient de créer avec les noms des images en argument et la valeur renvoyée est enregistrée dans une variable `laps_temps` que l'on affiche ensuite avec `print()`:

```
def obtenir_laps_temps(image1, image2):
  temps1 = obtenir_temps(image1)
  temps2 = obtenir_temps(image2)

  difference_temps = temps2 - temps1
  return difference_temps

# Calcul du laps de temps entre deux images
laps_temps = obtenir_laps_temps('photo_0683.jpg', 'photo_0684.jpg')
print(laps_temps)
```

Nous obtenons un affichage sous forme `heure:minute:seconde`:

```
0:00:09
```

mais ce que l'on souhaite est d'obtenir directement un laps de temps en seconde sous forme numérique plutôt que formaté comme tel. Pour cela, nous utilisons la méthode `.seconds` de notre objet `difference_temps` pour obtenir une conversion automatique en secondes lors du renvoi du résultat de notre fonction:

```
def obtenir_laps_temps(image1, image2):
  temps1 = obtenir_temps(image1)
```

```
  temps2 = obtenir_temps(image2)

  difference_temps = temps2 - temps1
  return difference_temps.seconds # On utilise la méthode ici


# Calcul du laps de temps entre deux images
laps_temps = obtenir_laps_temps('photo_0683.jpg', 'photo_0684.jpg')
print(laps_temps)
```

Le résultat correspond alors à ce qui est attendu

```
9
```

NOTE

N'oublie pas d'enregistrer ton projet.

2.1. Exercice

1. Teste avec différents fichiers images, comment sait-on si l'on a bien choisi des images qui se succèdent ou se précèdent ?

Si l'on fait pas attention, la différence de temps entre les images deviendrait négative dans le cas où les noms d'images seraient interverties. Cependant, la conversion en secondes sur l'instruction de retour `difference_temps.seconds` de la fonction `obtenir_laps_temps()` fournit une valeur entière positive. En effet, si l'on affiche directement `difference_temps`, le résultat serait

```
-1 day, 23:59:51
```

mais sa conversion en secondaire résulterait en la valeur `86391` au lieu des 9 secondes . Pour éviter cet écueil, il convient de comparer les temps avant de calculer la différence de temps comme ceci:

```
def obtenir_laps_temps(image1, image2):
  temps1 = obtenir_temps(image1)
  temps2 = obtenir_temps(image2)

  if temps2 > temps1:
    difference_temps = temps2 - temps1
  else:
```

```
    difference_temps = temps1 - temps2
return difference_temps.seconds # On utilise la méthode ici
```

3. Déterminer la distance parcourue

NOTE

Pour la suite, retire les deux instructions `print()` de la précédente section utilisé pour l'affichage des résultats.

Comme vu lors de l'introduction, la distance parcourue correspond à la différence de position en deux instants distincts. Pour deux photos prises successivement, le relief observé se déplace significativement. Pour l'œil humain, il est facile de faire ce constat mais un ordinateur n'en est pas capable directement : il faut qu'on lui donne des instructions précises pour qu'il y arrive.

3.1. Caractéristiques uniques dans les photos

Comment sommes-nous capables de remarquer le déplacement du relief ? Cela est possible car nous sommes capables de reconnaître des caractéristiques du relief qui n'ont pas changé d'une image à l'autre et de nous rendre compte que cette caractéristique a changé de place sur l'image. La caractéristique peut être la forme ondulée spécifique d'une rivière qui est facilement reconnaissable ou une montagne enneigée particulièrement visible par exemple.

Dans un ordinateur, un algorithme va imiter cette faculté humaine de retrouver des caractéristiques faciles à reconnaître et lister toutes celles qu'il peut trouver dans une image. Non seulement il va va les décrire mais il va aussi enregistrer des informations à propos de chaque caractéristique qu'il a décrit, c'est ce qu'on appelle le **descripteur** associé à la caractéristique.

L'algorithme abordé dans ce tutoriel s'appelle ORB (Oriented FAST and Rotated BRIEF). Dans notre cas, cet algorithme et l'ensemble des outils que l'on va utiliser sont tous prêts à l'emploi dans le module OpenCV qui devrait être installé préalablement. Si ce n'est pas encore le cas, installe le module `opencv-python` dans Thonny dans **Tools > Manage packages...** (ou **Outils > Gérer les paquets...** si Thonny est configuré en français).

Cela étant fait, il convient d'ajouter aux premières lignes l'importation des modules suivants :

```
from exif import Image # déjà présent auparavant
from datetime import datetime # déjà présent auparavant
import cv2 # module OpenCV
import math # module des fonctions mathématiques
```

Dans un premier temps, nous devons créer une fonction qui va ouvrir les images sous une forme acceptée par OpenCV. La fonction suivante ouvre deux images en noir et blanc dont les noms sont données en argument et renvoie les deux objets image en format OpenCV:

```
def convertir_en_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
```

Ensuite, nous créons une fonction qui va s'occuper d'identifier les caractéristiques des deux images. On a la possibilité de demander à lister un nombre précis de caractéristiques et on va utiliser cela à notre avantage. Commençons par la définition de la fonction :

```
def identifier_caracteristiques(image1, image2, nombre_caract):
```

On va ensuite créer un objet que l'on nomme `orb` qui contient l'algorithme ORB. Il suffit d'appeler `cv2.ORB_create()` du module OpenCV `cv2` en indiquant à l'argument `nfeatures` le nombre de caractéristiques voulues:

```
def identifier_caracteristiques(image1, image2, nombre_caract):
  orb = cv2.ORB_create(nfeatures=nombre_caract)
```

Pour enclencher l'algorithme sur les deux images, on va utiliser la méthode `.detectAndCompute()` sur l'objet `orb` qui demande deux arguments obligatoires dont le premier est l'objet image et le second est laissé à la valeur `None` par défaut. On renvoie alors la liste des caractéristiques et les descripteurs associées des deux photos :

```
def identifier_caracteristiques(image1, image2, nombre_caract):
  orb = cv2.ORB_create(nfeatures=nombre_caract)
  caracts_1, descripteurs_1 = orb.detectAndCompute(image1, None)
  caracts_2, descripteurs_2 = orb.detectAndCompute(image2, None)
  return caracts_1, descripteurs_1, caracts_2, descripteurs_2
```

Nous avons à présent une liste de points caractéristiques pour chacune des photos ainsi que les descripteurs associés. Il convient à présent de relier les caractéristiques qui sont communes aux deux images. On crée donc une fonction qui va comparer les caractéristiques d'une photo à toutes les caractéristiques de l'autre photo. Pour cela, le descripteur est utilisé. Donc notre fonction a besoin de la liste des descripteurs des deux photos. On définit la fonction comme suit:

```
def calculer_correspondance(descripteurs_1, descripteurs_2):
```

Un algorithme simple de correspondance (*matching* en anglais) par force brute (***brute force*** en anglais) va comparer chaque descripteur d'une image à tous ceux de l'autre jusqu'à trouver une correspondance. On crée un objet `brute_force` en appelant `cv2.BFMatcher()` de OpenCV avec des arguments par défaut adaptés :

```
def calculer_correspondance(descripteurs_1, descripteurs_2):
    brute_force = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
```

L'algorithme est enclenchée en appelant la méthode `.match()` de `brute_force` avec les descripteurs des deux images en argument. On obtient alors la liste des correspondances appelé `corresp`:

```
def calculer_correspondance(descripteurs_1, descripteurs_2):
    brute_force = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    corresp = brute_force.match(descripteurs_1, descripteurs_2)
```

Avant de renvoyer ce résultat, la liste des correspondances est triée par niveau de correspondance du plus fort au plus faible :

```
def calculer_correspondance(descripteurs_1, descripteurs_2):
    brute_force = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    corresp = brute_force.match(descripteurs_1, descripteurs_2)
    corresp = sorted(corresp, key=lambda x: x.distance)
    return corresp
```

Cela signifie que les correspondances les plus semblables sont mis en premier tandis que les moins semblables sont derniers de la liste.

Pour finir, on va vérifier que les fonctions créées sont fonctionnelles. Pour cela, il faut appeler en premier la fonction `convertir_en_cv()` en indiquant les nom des deux images. Puis on envoie le résultat à la fonction `identifier_caracteristiques()` en lui indiquant les images et que l'on veut trouver 1000 caractéristiques sur celles-ci. Finalement, on calcule la correspondance avec `calculer_correspondance()` en y indiquant les descripteurs calculés par la précédente fonction et on affiche la liste des correspondances.

```
# Calcul du laps de temps entre deux images
laps_temps = obtenir_laps_temps('photo_0683.jpg', 'photo_0684.jpg')
# Créer objet image en format OpenCV
image1_cv, image2_cv = convertir_en_cv('photo_0683.jpg', 'photo_0684.jpg')
# Obtenir les caractéristiques et les descripteurs
caracts_1, descripteurs_1, caracts_2, descripteurs_2 =
identifier_caracteristiques(image1_cv, image2_cv, 1000)
# Obtenir les correspondances entre les descripteurs
correspondances = calculer_correspondance(descripteurs_1, descripteurs_2)
# Afficher la liste des correspondances
print(correspondances)
```

A l'exécution, on devrait obtenir quelque chose de similaire à ceci

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

L'élément `< cv2.DMatch 0x11b2db8b0>` représente un objet donc nous obtenons une liste d'objets qui définit une correspondance entre les deux images. Nous allons à présent les afficher visuellement.

3.2. Affichage des correspondances entre caractéristiques

Pour commencer, enlevons la dernière instruction `print()` qui affichait la liste des correspondances.

Créons une fonction qui va afficher à l'écran les deux images, les points caractéristiques calculés et les correspondances entre ces points. Pour cela, la fonction `cv2.drawMatches()` est appelé avec les images, les caractéristiques et la liste de correspondances dont on ne sélectionne les 100 derniers éléments. La fonction se présente comme suit :

```
def afficher_correspondances(image1_cv, caracts_1, image2_cv, caracts_2, corresp):
  corresp_image = cv2.drawMatches(image1_cv, caracts_1, image2_cv, caracts_2,
corresp[:100], None)
```

Les instructions suivantes sont destinées à configurer la fenêtre qui va afficher les images pour qu'elles soient affichées côte à côte:

```
def afficher_correspondances(image1_cv, caracts_1, image2_cv, caracts_2, corresp):
  corresp_image = cv2.drawMatches(image1_cv, caracts_1, image2_cv, caracts_2,
corresp[:100], None)
  resize = cv2.resize(corresp_image, (1600,600), interpolation = cv2.INTER_AREA)
  cv2.imshow('correspondances', resize)
```

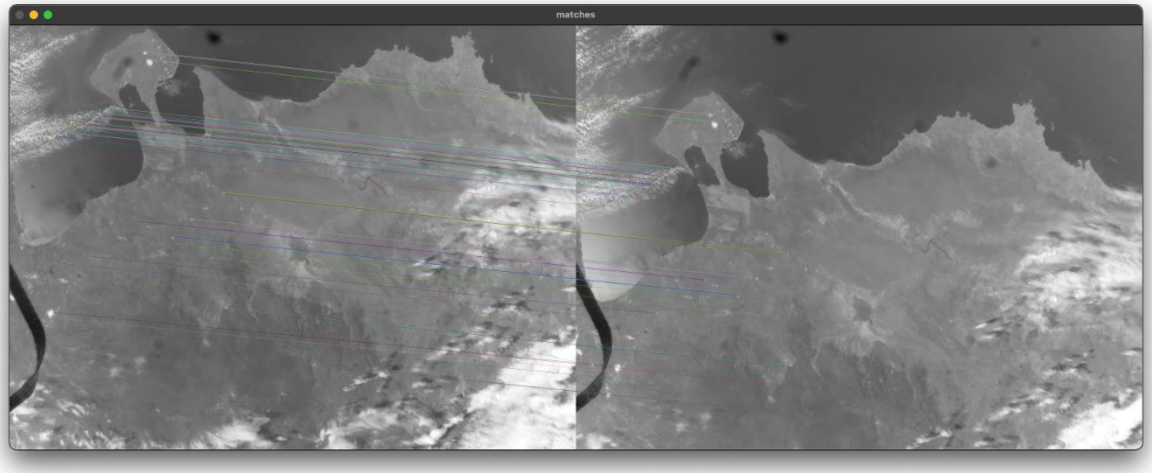
Si l'on exécute ce code tel quel, une fenêtre présentant les deux images va s'afficher mais il va se bloquer rapidement et il ne sera pas possible de le fermer normalement. On ajoute donc deux instructions supplémentaires permettant à la fenêtre de fermer lorsqu'une touche est appuyée.

```
def afficher_correspondances(image1_cv, caracts_1, image2_cv, caracts_2, corresp):
  corresp_image = cv2.drawMatches(image1_cv, caracts_1, image2_cv, caracts_2,
corresp[:100], None)
  resize = cv2.resize(corresp_image, (1600,600), interpolation = cv2.INTER_AREA)
  cv2.imshow('correspondances', resize)
  cv2.waitKey(0)
  cv2.destroyWindow('correspondances')
```

On peut à présent utiliser la fonction et l'appeler à la suite des autres appels de fonction :

```
# Calcul du laps de temps entre deux images
laps_temps = obtenir_laps_temps('photo_0683.jpg', 'photo_0684.jpg')
# Créer objet image en format OpenCV
image1_cv, image2_cv = convertir_en_cv('photo_0683.jpg', 'photo_0684.jpg')
# Obtenir les caractéristiques et les descripteurs
caracts_1, descripteurs_1, caracts_2, descripteurs_2 =
identifier_caracteristiques(image1_cv, image2_cv, 1000)
# Obtenir les correspondances entre les descripteurs
correspondances = calculer_correspondance(descripteurs_1, descripteurs_2)
# Afficher visuellement les correspondances
afficher_correspondances(image1_cv, caracts_1, image2_cv, caracts_2,
correspondances)
```

Nous devrions obtenir le résultat suivant:



NOTE

N'oublie pas d'enregistrer ton projet.

3.2.1. Exercice

1. Sur la fenêtre, on a affiché que les 100 premières correspondances. Affiche à présent l'ensemble des correspondances de la liste. Est-ce que toutes les correspondances sont pertinentes ?

Pour afficher l'ensemble des correspondances dans la fonction `afficher_correspondances()`, il suffit de retirer la sélection des 100 premiers éléments de la liste des correspondances, c.-à-

d. `corresp[:100]` pour que toute la liste soit prise en compte. La fonction devient donc

```
def afficher_correspondances(image1_cv, caracts_1, image2_cv, caracts_2, corresp):
  corresp_image = cv2.drawMatches(image1_cv, caracts_1, image2_cv, caracts_2,
corresp, None, matchesThickness=4)
  resize = cv2.resize(corresp_image, (1600,600), interpolation = cv2.INTER_AREA)
  cv2.imshow('correspondances', resize)
  cv2.waitKey(0)
  cv2.destroyWindow('correspondances')
```

3.3. Obtention des coordonnées des correspondances

A présent que les correspondances entre les caractéristiques identiques des deux images a été établie, il nous faut obtenir les coordonnées de ces points correspondants. On définit une fonction qui utilisera la liste des caractéristiques des deux images ainsi que la liste des correspondances:

```
def obtenir_coord_corresp(caracts_1, caracts_2, correspondances):
```

On crée deux listes vides pour stocker les coordonnées de chaque caractéristique mise en correspondance dans chacune des images :

```
def obtenir_coord_corresp(caracts_1, caracts_2, correspondances):
  coordonnees_1 = []
  coordonnees_2 = []
```

Pour parcourir la liste de correspondance, on utilise une boucle :

```
def obtenir_coord_corresp(caracts_1, caracts_2, correspondances):
  coordonnee_1 = []
  coordonnee_2 = []
  for correspondance in correspondances:
```

La liste de correspondance contient des objets qui peuvent nous informer quelle caractéristique d'une image a été mise en correspondance avec quelle caractéristique de l'autre image. Cela nous permet d'obtenir les coordonnées `x1, y1` et `x2, y2` pour chaque image :

```
def obtenir_coord_corresp(caracts_1, caracts_2, correspondances):
```

```
  coordonnees_1 = []
  coordonnees_2 = []
  for correspondance in correspondances:
    image_1_idx = correspondance.queryIdx
    image_2_idx = correspondance.trainIdx
    (x1,y1) = caracts_1[image_1_idx].pt
    (x2,y2) = caracts_2[image_2_idx].pt
```

Ensuite, ces coordonnées peuvent être ajoutées aux deux listes de coordonnées, et les deux listes peuvent être renvoyées :

```
def obtenir_coord_corresp(caracts_1, caracts_2, correspondances):
  coordonnees_1 = []
  coordonnees_2 = []
  for correspondance in correspondances:
    image_1_idx = correspondance.queryIdx
    image_2_idx = correspondance.trainIdx
    (x1,y1) = caracts_1[image_1_idx].pt
    (x2,y2) = caracts_2[image_2_idx].pt
    coordonnees_1.append( (x1,y1) )
    coordonnees_2.append( (x2,y2) )
    return coordonnees_1, coordonnees_2
```

Ajoutons un appel de fonction en bas du script pour stocker les données de sortie de la fonction. Ajoutons ensuite une ligne pour afficher la première paire de coordonnées de chaque liste. Exécute le programme :

```
# Calcul du laps de temps entre deux images
laps_temps = obtenir_laps_temps('photo_0683.jpg', 'photo_0684.jpg')
# Créer objet image en format OpenCV
image1_cv, image2_cv = convertir_en_cv('photo_0683.jpg', 'photo_0684.jpg')
# Obtenir les caractéristiques et les descripteurs
caracts_1, descripteurs_1, caracts_2, descripteurs_2 =
identifier_caracteristiques(image1_cv, image2_cv, 1000)
# Obtenir les correspondances entre les descripteurs
correspondances = calculer_correspondance(descripteurs_1, descripteurs_2)
# Afficher visuellement les correspondances
afficher_correspondances(image1_cv, caracts_1, image2_cv, caracts_2,
correspondances)
# Obtention des coordonnées des points mis en correpondance
coordonnees_1, coordonnees_2 = obtenir_coord_corresp(caracts_1, caracts_2,
correspondances)
```

```
print(coordonnees_1[0], coordonnees_2[0])
```

Après avoir fermé la fenêtre affichant l'image des correspondances en ayant appuyé sur une touche, le résultat suivant devrait s'afficher :

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

NOTE

N'oublie pas d'enregistrer ton projet.

3.4. Calcul de la distance des caractéristiques

NOTE

Pour la suite, efface l'instruction d'affichage `print()` précédente.

Lorsque les coordonnées des caractéristiques correspondantes sont enregistrées, il est possible de calculer la distance entre les coordonnées des caractéristiques correspondantes. Il s'agira toutefois de la distance sur les images exprimée en pixels, 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.

Créons une fonction pour calculer la distance moyenne entre les coordonnées concordantes. Appelons-la `calculer_distance_moyenne()`. Deux arguments sont nécessaires et ce seront les deux listes de coordonnées obtenues précédemment :

```
def calculer_distance_moyenne(coordonnees_1, coordonnees_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 et ainsi de suite. Ensuite, les premiers éléments de chacune des listes sont combinés. L'objet de liste zippé peut facilement être reconverti en une seule liste simple.

Commençons par créer une variable pour stocker la somme de toutes les distances entre les coordonnées et appelons-la `total_distances`. Ensuite, on peut zipper les deux listes, puis reconvertir l'objet zippé en une liste :

```
def calculer_distance_moyenne(coordonnees_1, coordonnees_2):
  total_distances = 0
  zip_coordonnees = list( zip(coordonnees_1, coordonnees_2) )
```

Pour voir ce qui s'est passé ici, vous pouvez le faire avec `print()` pour voir le détail des listes. Ajoutez trois appels `print()` pour voir un élément de `coordinates_1`, `coordinates_2` et `merged_coordinates`.

```
def calculer_distance_moyenne(coordonnees_1, coordonnees_2):
  total_distances = 0
  zip_coordonnees = list( zip(coordonnees_1, coordonnees_2) )
  print( coordonnees_1[0] )
  print( coordonnees_2[0] )
  print( zip_coordonnees[0] )
```

Pour exécuter cette fonction, appelons-le tout en bas du script:

```
calculer_distance_moyenne(coordonnees_1, coordonnees_2)
```

ce qui devrait afficher quelque chose de similaire à ceci:

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

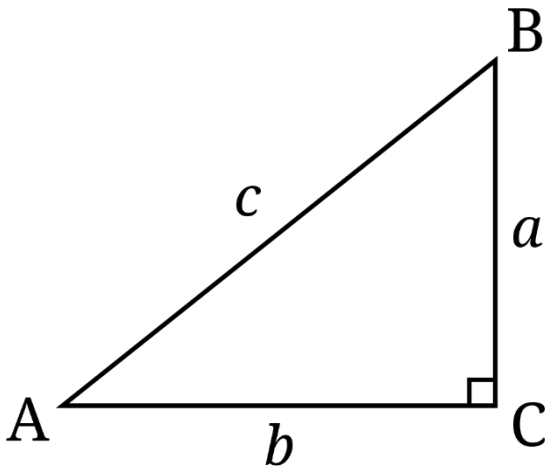
Normalement, on voit que les coordonnées `x` et `y` de chaque caractéristique de chaque image ont été combinées. Cela facilitera l'itération sur la nouvelle liste.

Avant de passer à la suite, il faut supprimer les appels `print()` et ajouter une boucle `for` pour itérer sur les coordonnées fusionnées `zip_coordonnees`, et calculer les différences entre les coordonnées `x` et `y` de chaque image.

```
def calculer_distance_moyenne(coordonnees_1, coordonnees_2):
  total_distances = 0
  zip_coordonnees = list( zip(coordonnees_1, coordonnees_2) )
  for coordonnee in zip_coordonnees:
    difference_x = coordonnee[0][0] - coordonnee[1][0]
    difference_y = coordonnee[0][1] - coordonnee[1][1]
```

Si l'on désigne par A le point correspondant à la première caractéristique et B celui du deuxième, on peut les visualiser comme

les sommets d'un triangle rectangle comme présenté dans le schéma suivant



DEFI

Trouve l'expression mathématique permettant le calcul de la distance entre deux points en connaissant les coordonnées de ces deux points.

La distance entre les points **A** et **B** est la longueur du segment **c**. Il s'agit de l'hypoténuse. L'expression mathématique est

$$distance \equiv c = \sqrt{a^2 + b^2}$$

Où a correspond à `difference_y` et b correspond à `difference_x` . L'hypoténuse peut être calculée à l'aide du module `math` avec sa fonction `.hypot()`. La différence entre les `x` et `y` calculé précédemment nous a fourni la longueur du segment **b** et **a** respectivement. Ce sont les arguments nécessaire à la fonction `.hypot()` pour obtenir l'hypoténuse, qui est la distance entre **A** et **B** recherchée.

DEFI

Connaissant l'expression mathématique de la distance trouvée précédemment, comment calculer la distance sans utiliser la fonction `.hypot()`? La fonction racine carrée est définie dans le module `math` comme `.sqrt()`.

On peut obtenir le même réponse en écrivant

```
distance = math.sqrt(difference_x*difference_x + difference_y*difference_y)
```

On l'ajoute alors à `total_distances` pour en obtenir la somme totale.

```
def calculer_distance_moyenne(coordonnees_1, coordonnees_2):
  total_distances = 0
  zip_coordonnees = list( zip(coordonnees_1, coordonnees_2) )
  for coordonnee in zip_coordonnees:
    difference_x = coordonnee[0][0] - coordonnee[1][0]
    difference_y = coordonnee[0][1] - coordonnee[1][1]
    distance =  math.hypot(difference_x, difference_y)
    total_distances = total_distances + distance
```

On renvoie la distance moyenne entre les caractéristiques en divisant `total_distances` par le nombre de caractéristiques concordantes, c'est-à-dire par la longueur de la liste `zip_coordonnees` mesurée par la fonction `len()`:

```
def calculer_distance_moyenne(coordonnees_1, coordonnees_2):
  total_distances = 0
  zip_coordonnees = list( zip(coordonnees_1, coordonnees_2) )
  for coordonnee in zip_coordonnees:
    difference_x = coordonnee[0][0] - coordonnee[1][0]
    difference_y = coordonnee[0][1] - coordonnee[1][1]
    distance =  math.hypot(difference_x, difference_y)
    total_distances = total_distances + distance
  return total_distances/len(zip_coordonnees)
```

On ajoute un appel de fonction en bas du script pour calculer la distance moyenne, puis on affiche le résultat :

```
# Calcul du laps de temps entre deux images
laps_temps = obtenir_laps_temps('photo_0683.jpg', 'photo_0684.jpg')
# Créer objet image en format OpenCV
image1_cv, image2_cv = convertir_en_cv('photo_0683.jpg', 'photo_0684.jpg')
# Obtenir les caractéristiques et les descripteurs
caracts_1, descripteurs_1, caracts_2, descripteurs_2 =
identifier_caracteristiques(image1_cv, image2_cv, 1000)
# Obtenir les correspondances entre les descripteurs
correspondances = calculer_correspondance(descripteurs_1, descripteurs_2)
# Afficher visuellement les correspondances
afficher_correspondances(image1_cv, caracts_1, image2_cv, caracts_2,
correspondances)
```

```
# Obtention des coordonnées des points mis en correpondance
coordonnees_1, coordonnees_2 = obtenir_coord_corresp(caracts_1, caracts_2, correspondances)
# Calcul de la distance moyenne à partir des coordonnées des points
distance_moyenne = calculer_distance_moyenne(coordonnees_1, coordonnees_2)
```

A l'exécution, un résultat similaire à "504.08973470622516" devrait s'afficher. Cette distance correspond en réalité à une distance en pixels sur l'image. Il faudra encore le convertir grâce à un facteur appelé GSD. C'est l'objet de la prochaine section.

NOTE

N'oublie pas d'enregistrer ton projet.

4. Calcul de la vitesse moyenne

NOTE

Pour la suite, supprime toutes les instruction `print()`.

Maintenant que les distances en pixels des caractéristiques ont été calculées pour les deux photos, il faut les convertir en distance réelle en km entre les caractéristiques sur Terre. C'est le facteur appelé ***Ground Sampling Distance*** (GSD) qui détermine la distance réelle correspondant à chaque pixel parcouru sur l'image. Cette valeur dépend de la hauteur de l'ISS, du type de caméra utilisé pour prendre les photos et de la résolution des images.

Pour une résolution de 4056x3040, la caméra embarquée sur l'ISS a un GSD de 12648 cm/px. Cela signifie que pour chaque pixel parcouru sur l'image, on a 12648 cm parcouru en réalité sur le sol.

IMPORTANT

Si la résolution de l'image est différente, il faut adapter la valeur du GSD en **utilisant un calculateur en ligne par exemple**.

Une fois ce calcul effectué, le décalage temporel entre les deux photos exprimé en secondes peut être utilisé pour calculer la vitesse de la caméra et, par conséquent, la vitesse de l'ISS.

Créons une fonction qui calcule la vitesse de l'ISS basé sur la distance moyenne parcourue en pixels, le laps de temps écoulé et le GSD:

```
def calculer_vitesse_en_kmps(distance_moyenne, GSD, laps_temps):
```

Calculons la distance réelle parcourue. Pour cela, on multiplie la distance moyenne ne pixel par le GSD pour obtenir une distance en cm. Comme on veut obtenir une distance en kilomètre et que 1 km=100000cm, il convient de ne pas oublier de diviser la valeur du GSD par 100000.

```
def calculer_vitesse_en_kmps(distance_moyenne, GSD, laps_temps):
  distance = distance_moyenne * GSD / 100000
```

La vitesse est alors simplement obtenue en divisant cette distance par le laps de temps entre les deux images. On obtient une valeur en km/s que l'on renvoie :

```
def calculer_vitesse_en_kmps(distance_moyenne, GSD, laps_temps):
  distance = distance_moyenne * GSD / 100000
  vitesse = distance / laps_temps
  return vitesse
```

Pour l'exécuter, on l'ajoute à la fin du script avec les autres appels de fonctions. On n'oublie pas d'indiquer le GSD de 12648 en argument ainsi que les autres valeurs nécessaires. Finalement, on affiche la valeur renvoyé par la fonction avec `print()`:

```
# Calcul du laps de temps entre deux images
laps_temps = obtenir_laps_temps('photo_0683.jpg', 'photo_0684.jpg')
# Créer objet image en format OpenCV
image1_cv, image2_cv = convertir_en_cv('photo_0683.jpg', 'photo_0684.jpg')
# Obtenir les caractéristiques et les descripteurs
caracts_1, descripteurs_1, caracts_2, descripteurs_2 =
identifier_caracteristiques(image1_cv, image2_cv, 1000)
# Obtenir les correspondances entre les descripteurs
correspondances = calculer_correspondance(descripteurs_1, descripteurs_2)
# Afficher visuellement les correspondances
afficher_correspondances(image1_cv, caracts_1, image2_cv, caracts_2,
correspondances)
# Obtention des coordonnées des points mis en correpondance
coordonnees_1, coordonnees_2 = obtenir_coord_corresp(caracts_1, caracts_2,
correspondances)
```

```
# Calcul de la distance moyenne à partir des coordonnées des points
distance_moyenne = calculer_distance_moyenne(coordonnees_1, coordonnees_2)
# Calcul de l'estimation de la vitesse
vitesse = calculer_vitesse_en_kmps(distance_moyenne, 12648, laps_temps)
print(vitesse)
```

Avec les deux images particulières utilisés dans ce tutoriel, une valeur de `7.084` est renvoyée, ce qui n'est pas loin des 7.66 km/s moyens de l'ISS. Pour d'autres images, des valeurs différentes pourraient être obtenues.

NOTE

N'oublie pas d'enregistrer ton projet.

5. Aller plus loin

Le code est encore incomplet : les images fournies ont été prises préalablement mais, à bord de l'ISS, il sera nécessaire de prendre des photos avec le module `picamera`. Pour y arriver, il est recommandé de suivre la ressource suivante pour acquérir une compréhension complète du module pour l'adapter aux besoins spécifiques : `https://esero.fr/wp-content/uploads/2021/04/Premiers-pas-avec-le-module-camera-_-Raspberry-Pi-Projects_FR.pdf`

Quelques exercices sont proposés pour chaque section du tutoriel. Les réaliser permettra une meilleure compréhension du code avant de tenter de l'améliorer. Il est donc conseillé d'en compléter un maximum.

La vitesse obtenue est satisfaisante mais l'estimation peut encore être améliorée. Voici quelques pistes à explorer pour améliorer l'estimation :

- Lors de la détection des caractéristiques, seul 1000 caractéristiques ont été cherchées avec algorithme ORB. Qu'en est-il si on change ce paramètre ?
- Est-ce qu'un autre algorithme donnerait de meilleurs résultats ? Une liste d'algorithme est disponible dans la documentation OpenCV

: **https://docs.opencv.org/3.4/db/d27/tutorial_py_table_of_contents_feature2d.html**

- La correspondance des caractéristiques a été faite par un algorithme de Force Brute puis triée du moins correspondant au plus correspondant mais la recherche des coordonnées correspondantes a utilisé toutes les correspondances sans tenir compte du triage. Il conviendrait d'utiliser les meilleures correspondances uniquement pour améliorer l'estimation.
- Si des nuages apparaissent, la détection des caractéristiques va être biaisée par leur présence. Comment les ignorer ? Sinon, comment minimiser leur impact en sachant qu'ils peuvent être en mouvement eux aussi ?
- Comment utiliser un modèle de Deep Learning avec l'accélérateur Google Coral peut aider dans ce projet ? Voici une ressource à ce sujet : **https://esero.fr/wp-content/uploads/2022/12/ImageClassificationCoral_RaspPiProj_FR.pdf**
- Au lieu d'utiliser que deux images, peut-on en utiliser plus ?
- Au lieu d'utiliser la méta-donnée du temps de prise de l'image, mesurer directement le temps plus précisément au moment de prendre l'image.
- La distance moyenne estimée correspond à la distance au sol et non au niveau de l'orbite de l'ISS. Comment adapter la distance mesurée ?
- La Terre n'est pas parfaitement sphérique donc sa courbure varie en fonction du lieu où se situe l'ISS. Comment cela impacte la distance moyenne estimée au sol ?
- La Terre tourne aussi à sa propre vitesse durant l'orbite de l'ISS. Comment cela impacte-t-il l'estimation de la vitesse ?
- La hauteur de l'ISS varie au cours de son orbite mais aussi au cours de la journée. Quel impact cela a-t-il et comment en tenir compte ?