

Guide Astro Pi Mission Space Lab

Le Guide Mission Space Lab accompagne les équipes dans la programmation d'expériences scientifiques sur les ordinateurs Astro Pi de la Station spatiale internationale (ISS). Ce document détaille les objectifs de la mission, énonce les contraintes techniques, oriente vers les ressources complémentaires et précise les modalités de soumission des projets.



Introduction

Avec le projet Mission Space Lab, votre défi consiste à écrire un programme Python qui collecte et enregistre des données à l'aide des capteurs ou de la caméra d'un ordinateur Astro Pi embarqué à bord de la Station spatiale internationale (ISS). Ces capteurs et cette caméra vous permettent d'étudier diverses questions relatives à l'ISS et de recueillir des données réelles provenant de l'espace.

Votre programme fonctionnera de manière autonome pendant 10 minutes à bord de l'ISS. Pendant ce laps de temps, il devra collecter des données, puis les enregistrer dans un fichier en vue d'une analyse ultérieure.

C'est à vous de décider comment utiliser ces 10 minutes. Vous pouvez explorer une question scientifique, collecter et analyser des données de capteurs, prendre des photos de la Terre ou créer un programme plus complexe combinant plusieurs de ces idées. À vous de choisir.



Avant de pouvoir être exécuté à bord de l'ISS, votre code doit passer avec succès une série de contrôles effectués par le centre de contrôle Astro Pi. Ces contrôles permettent de vérifier que votre programme fonctionne correctement et qu'il répond aux exigences de la mission.

Si votre code passe les tests avec succès, il obtiendra le certificat officiel de vol et sera programmé pour être exécuté à bord de l'ISS. S'il contient des erreurs ou s'il ne répond pas aux exigences opérationnelles fondamentales, il ne pourra pas être envoyé dans l'ISS.

Il ne s'agit pas d'un guide complet, étape par étape, pour la création de votre programme. Ce document a pour but de vous aider à vous lancer. Vous et votre équipe devrez prendre des décisions concernant les objectifs de votre projet et la manière de transposer vos idées en code.



Besoin d'aide ?

Si vous rencontrez des difficultés, n'hésitez pas à [nous contacter](#) ! Vous pouvez également participer aux différents webinaires organisés par le CNES et l'ESA au cours desquels vous pourrez poser vos questions aux équipes Astro Pi et ESERO France.

Sur quoi enquêter ?

Avec un ordinateur Astro Pi à bord de l'ISS, vous pouvez explorer de nombreuses questions scientifiques différentes. Les capteurs et la caméra vous permettent de collecter des données réelles depuis l'espace et de les utiliser pour approfondir un sujet qui vous intéresse.

Pour vous inspirer, voici quelques exemples de projets réalisés par les équipes précédentes de Mission Space Lab :

- Analyse de la végétation à l'aide de l'indice NDVI et de l'assemblage d'images
- Identification des amas d'algues dans les océans
- Mesurer le champ magnétique terrestre
- Analyse des nuages

Remarque : ces projets ont été conçus selon les anciennes règles de Mission Space Lab, qui autorisaient une durée maximale de 3 heures pour les programmes exécutés à bord de l'ISS. Dans le cadre du défi actuel, votre programme ne doit pas dépasser 10 minutes. Vous devrez donc adapter ces idées pour respecter cette limite de temps.

Ce dont vous aurez besoin

Pour mener à bien ce projet, vous aurez besoin de :

- **Un ordinateur utilisant Python 3.13 ou une version ultérieure.** Vous pouvez utiliser un Raspberry Pi ou n'importe quel ordinateur fonctionnant sous Microsoft Windows, macOS ou Linux. Vous trouverez ici [les instructions d'installation de Python](#). Une description complète de la configuration requise pour Python dans le cadre de Mission Space Lab figure plus loin dans ce guide.
- **Une connexion Internet.** Vous devrez disposer d'un accès à Internet pour utiliser la version en ligne de l'outil [Astro Pi Replay Tool](#) afin de simuler l'exécution de votre code en temps réel sur les Astro Pi de l'ISS. Vous aurez également besoin d'un accès à Internet pour envoyer votre programme.
- **Un éditeur de code.** C'est là que vous écrirez et modifierez votre code Python. N'importe quel éditeur de texte fera l'affaire, mais nous vous recommandons d'utiliser un éditeur de code dédié tel que Thonny. Dans ce guide, nos exemples utiliseront Thonny.

Les ordinateurs Astro Pi

Les Astro Pi embarqués à bord de l'ISS sont deux ordinateurs Raspberry Pi 4 8 Go modifiés, équipés d'une carte d'extension Sense HAT et d'une caméra. Ils sont placés dans une malle renforcée sur mesure en aluminium. Le Sense HAT (V2) comprend notamment des capteurs de température et d'humidité, un gyroscope, un magnétomètre, un accéléromètre et des capteurs de lumière/couleur, qui permettent de mesurer le champ magnétique local et l'accélération, par exemple. Les ordinateurs sont équipés d'une puissante caméra haute qualité Raspberry Pi dotée d'un objectif de 5 mm capable de prendre des photos incroyables de la Terre. De plus, ces ordinateurs peuvent effectuer un apprentissage automatique en temps réel grâce aux accélérateurs Coral associés. Vous pouvez en **découvrir plus sur les ordinateurs et les capteurs ici** (<https://astro-pi.org/about/the-computers>).



Pour collecter des données qui évoluent au fil du temps, à mesure que la Station spatiale internationale (ISS) tourne autour de la Terre, vous devriez vous concentrer sur des capteurs tels que la caméra, le gyroscope, le magnétomètre, l'accéléromètre ou les capteurs de lumière et de couleur.

Les mesures telles que la température et l'humidité ont tendance à rester relativement constantes à l'intérieur de l'ISS, tandis que le capteur de mouvement infrarouge passif (PIR) détecte principalement l'activité de l'équipage à proximité des ordinateurs Astro Pi. Bien que vous puissiez utiliser des capteurs de mouvement, de température et d'humidité dans votre projet, ceux-ci sont moins susceptibles de fournir des données qui varient de manière significative au cours de votre expérience de 10 minutes.

En choisissant des capteurs capables de détecter les variations environnementales, vous obtiendrez des données plus intéressantes à analyser.

Information importante

N'oubliez pas que votre programme doit fonctionner de manière entièrement autonome du début à la fin et ne peut pas dépendre des commandes du joystick ni des pressions sur les boutons effectuées par un astronaute.

L'environnement Python d'Astro Pi

Les ordinateurs Astro Pi de l'ISS disposent de la version 3.13 de Python, donc vous devrez utiliser cette version ou une version ultérieure. Si vous utilisez une version ultérieure, sachez qu'il peut y avoir de nouvelles fonctions qui fonctionnent sur votre ordinateur, mais pas sur les Astro Pi.

Outre la bibliothèque standard de Python, les ordinateurs Astro Pi intègrent plusieurs modules supplémentaires destinés à vous aider à mener à bien votre projet « Mission Space Lab ». Ceux-ci sont présentés ci-dessous, accompagnés d'exemples et de liens vers une documentation plus détaillée.

Il existe de nombreuses bibliothèques Python qui peuvent vous aider dans votre projet. Choisissez celles qui correspondent le mieux à votre projet et qui vous permettront d'atteindre vos objectifs de recherche.

Les modules (parties) de la bibliothèque standard que vous pouvez utiliser sont soumis à certaines restrictions. Les modules suivants ne sont pas autorisés. Si vous les utilisez, votre programme ne sera pas accepté :

Bibliothèques interdites

<https://docs.google.com/spreadsheets/u/0/d/1EoVzgA8gOiDXsJ1k9dQBdPyFC8U3bXFca2dRmdKNbcl/edit>

Exemples de bibliothèques utiles



Utilisation

Skyfield est un programme d'astronomie qui calcule la position des étoiles, des planètes et des satellites qui sont en orbite autour de la Terre.

Par exemple, vous pouvez utiliser Skyfield pour calculer la position actuelle de Mars :

```
from skyfield.api import Loader
from pathlib import Path

bsp_file = Path.home() / "de421.bsp"
load = Loader(bsp_file.parent)
planets = load(bsp_file.name)
mars = planets['Mars Barycenter']
ts = load.timescale()
barycentric = mars.at(ts.now())
print(barycentric)
```

Cet extrait de code fonctionne, mais le fichier d'éphémérides (de421.bsp) est trop volumineux pour être inclus dans votre charge utile finale. Pour contourner ce problème, importez les éphémérides à partir de la bibliothèque `astro_pi_orbit`, qui se chargera d'importer le fichier à votre place :

```
from skyfield.api import load
from astro_pi_orbit import de421

planets = de421
mars = planets['Mars Barycenter']
ts = load.timescale()
barycentric = mars.at(ts.now())
print(barycentric)
```

Documentation

- [Rhodesmil.org/skyfield](https://rhodesmil.org/skyfield)

astro_pi_orbit

Utilisation

La bibliothèque `astro_pi_orbit` offre des fonctionnalités destinées à aider les participants à Mission Space Lab à exploiter les données orbitales. Elle peut être utilisée pour :

- Déterminer la position en direct de l'ISS
- Accéder aux fichiers d'éphémérides de421 ou de440s (ces fichiers sont trop volumineux pour que vous puissiez les télécharger vous-même)
- Obtenir la trajectoire de l'ISS

Documentation

- [astro_pi_orbit](#)

picamzero

La bibliothèque Python permettant de contrôler le module caméra du Raspberry Pi sur les ordinateurs Astro Pi s'appelle picamzero. Pour commencer, consultez ce guide qui propose un tutoriel pratique sur son utilisation.

Utilisation

```
from picamzero import Camera
from time import sleep

camera = Camera()

# Take a picture every minute for 3 hours
for i in range(3*60):
    camera.take_photo(f'image_{i:03d}.jpg')
    sleep(60)
```

Documentation

- [picamzero](#)

Information importante

La bibliothèque picamzero figure dans Thonny, mais elle ne s'installe pas sous Windows ni sous macOS, car elle contrôle le matériel du Raspberry Pi et repose sur des composants exclusifs à Linux. Toutefois, cela ne vous empêchera pas de mener à bien votre expérience : vous pouvez toujours exécuter votre code à l'aide de l'outil Astro Pi Replay, que ce soit en ligne ou hors ligne. Pour installer picamzero sur un Raspberry Pi, ouvrez le terminal dans Thonny et exécutez la commande suivante : `pip install picamzero`.

GPIO Zero

GPIO Zero est une bibliothèque GPIO (General-Purpose Input/Output, c'est-à-dire Entrée-sortie à usage général) simple mais puissante. La plupart de ses fonctions sont limitées à bord de l'ISS. Par exemple, la seule broche à laquelle vous pouvez accéder est la broche GPIO 12, où le capteur de mouvement est connecté. Cependant, certaines de ses autres fonctionnalités peuvent être utiles pour votre expérience, comme le dispositif interne CPUtemperature.

Utilisation

Comparer les relevés de température du processeur du Raspberry Pi aux relevés de température du Sense HAT.

```

from sense_hat import SenseHat
from gpiozero import CPUTemperature

sense = SenseHat()
cpu = CPUTemperature()

while True:
    print(f'CPU: {cpu.temperature}')
    print(f'Sense HAT: {sense.temperature}')

```

Documentation

- gpiozero.readthedocs.io

NumPy

NumPy est une bibliothèque de traitement de tableau à usage général conçu pour manipuler efficacement de grands tableaux multidimensionnels (par exemple, des matrices) d'enregistrements arbitraires sans sacrifier trop de vitesse pour de petits tableaux multidimensionnels.

Utilisation

Vous pouvez utiliser NumPy pour traiter les pixels individuels de chaque image prise par la caméra à l'aide de la fonction `capture\_array` de picamzero. Par exemple, vous pouvez filtrer l'image pour n'afficher que le canal rouge :

```

from picamzero import Camera
import numpy as np
import PIL

camera = Camera()
image_as_array = camera.capture_array()
red_channel = image_as_array[:, :, 0]
Image.fromarray(red_channel).save("red-channel.jpg")

```

Documentation

- [NumPy](https://numpy.org)

SciPy

SciPy est une librairie Python libre et open source utilisée pour l'informatique scientifique et technique. SciPy contient des modules pour l'optimisation, l'algèbre linéaire, l'intégration, l'interpolation, les fonctions spéciales, la FFT (transformation de Fourier rapide), le traitement de signaux et d'images, les solveurs d'EDO (équations différentielles ordinaires) et d'autres tâches fréquentes en science et en ingénierie. Vous devrez peut-être utiliser cette bibliothèque pour résoudre une équation particulière.

Documentation

- [SciPy](https://scipy.org)

Pandas

Pandas est une bibliothèque open source proposant des outils d'analyse de données et des systèmes de données performants et faciles à utiliser.

Utilisation

```
import pandas as pd

df = pd.read_csv("my_test_data.csv")
df.describe()
```

Documentation

- [Pandas](#)

logzero

logzero est une bibliothèque utilisée pour faciliter la journalisation. Les journaux sont les enregistrements de ce qui s'est passé pendant qu'un programme était en cours d'exécution, et peuvent être vraiment utiles pour le débogage.

Utilisation

Les journaux sont classés en différents niveaux en fonction de leur gravité. En utilisant les différents niveaux de manière appropriée, vous pourrez ajuster le nombre d'informations que vous obtenez sur votre programme en fonction de vos besoins de débogage.

```
from logzero import logger

logger.debug("hello")
logger.info("info")
logger.warning("warning")
logger.error("error")
```

Documentation

- [logzero](#)

Matplotlib

Matplotlib est une bibliothèque de traçage 2D qui produit des tracés de qualité publication dans une variété de formats papier et d'environnements interactifs. Vous pouvez l'utiliser pour analyser les résultats de vos essais.

Utilisation

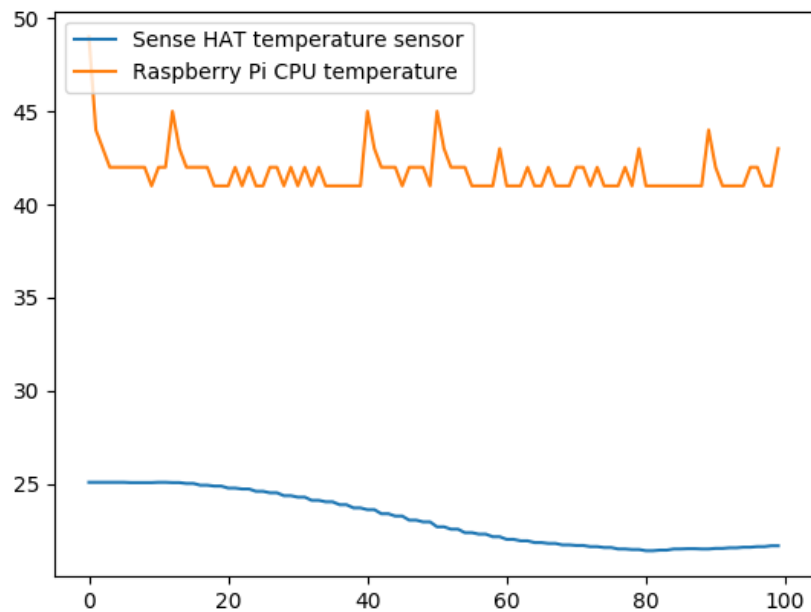
```
from sense_hat import SenseHat
from gpiozero import CPUTemperature
import matplotlib.pyplot as plt
from time import sleep

sense = SenseHat()
cpu = CPUTemperature()

st, ct = [], []
for i in range(100):
    st.append(sense.temperature)
    ct.append(cpu.temperature)
    sleep(1)

plt.plot(st)
plt.plot(ct)
plt.legend(['Sense HAT temperature sensor', 'Raspberry Pi CPU temperature'], loc='upper left')
plt.show()
```

Le résultat du programme est une courbe de température générée avec Matplotlib.



Documentation

- [Matplotlib](#)

Pillow

Pillow est une bibliothèque de traitement d'images. Elle offre une prise en charge étendue des formats de fichier, une représentation interne efficace et des capacités de traitement d'image assez puissantes.

La bibliothèque d'images de base est conçue pour offrir un accès rapide aux données stockées dans quelques formats de pixel de base. Elle peut fournir une base solide pour un outil de traitement d'image général.

Documentation

- [Pillow](#)

OpenCV

OpenCV est une bibliothèque open source dédiée à la reconnaissance visuelle. Vous pouvez par exemple utiliser OpenCV pour la détection des contours.

Documentation

- [OpenCV](#)

exif

exif vous permet de lire et de modifier les métadonnées Exif des images à l'aide de Python. Vous pouvez l'utiliser pour intégrer les données GPS sur toutes les images que vous prenez, ou [pour analyser les photos prises à bord de l'ISS](#).

Documentation

- [exif](#)

scikit-learn

scikit-learn est un ensemble d'outils simples et efficaces pour l'extraction et l'analyse des données accessibles à tous et réutilisables dans différents contextes. Il a été conçu pour interagir avec numpy, scipy et matplotlib.

Documentation

- [scikit-learn](#)

scikit-image

scikit-image est une bibliothèque de traitement d'images open source. Elle comprend des algorithmes pour la segmentation, les transformations géométriques, la manipulation de l'espace colorimétrique, l'analyse, le filtrage, la morphologie, la détection de caractéristiques, et bien plus encore.

Documentation

- [scikit-image](#)

reverse-geocoder

reverse-geocoder prend une coordonnée latitude/longitude et donne le village ou la ville la plus proche.

Utilisation

Lorsqu'il est utilisé avec skyfield, reverse-geocoder peut déterminer la position actuelle de l'ISS.

```
import reverse_geocoder
from skyfield.api import Loader
from pathlib import Path

tle_file = Path.home() / "iss.tle"
load = Loader(tle_file.parent)
if not tle_file.exists():
    load.download("https://celestrak.com/NORAD/elements/stations.txt", filename=tle_file.name)
satellites = load.tle_file(tle_file.name)
iss = satellites[0]
ts = load.timescale()

coordinates = iss.at(ts.now()).subpoint()
coordinate_pair = (
    coordinates.latitude.degrees,
    coordinates.longitude.degrees)

location = reverse_geocoder.search(coordinate_pair)
print(location)
```

Selon ce résultat, l'ISS se trouve actuellement au-dessus de Hamilton, dans l'état de New York :

```
[OrderedDict([
  ('lat', '42.82701'),
  ('lon', '-75.54462'),
  ('name', 'Hamilton'),
  ('admin1', 'New York'),
  ('admin2', 'Madison County'),
  ('cc', 'US')
]])
```

Remarque : la bibliothèque reverse-geocoder ne peut pas être exécutée avec l'outil Replay en ligne, car elle utilise le multitraitement, ce qui est incompatible avec l'environnement de cet outil. Si vous souhaitez utiliser cette bibliothèque, vous devrez tester les sections concernées de votre code localement dans votre éditeur de code, ou à l'aide de la version du Replay Tool disponible sous forme de plug-in pour Thonny.

Documentation

- [reverse-geocoder](#)

sense_hat

La bibliothèque sense_hat est la bibliothèque principale utilisée pour collecter des données à l'aide du Sense HAT Astro Pi. Consultez [ce guide](#) pour commencer.

Utilisation

Vous pouvez consigner l'humidité par rapport à l'affichage en utilisant le code ci-dessous :

```
from sense_hat import SenseHat
sense = SenseHat()
print(str(sense.get_humidity()))
```

Documentation

- [sense-hat](#)
- [Documentation supplémentaire pour le capteur de couleur](#)

La bibliothèque ai_edge_litert permet d'exécuter des tâches de machine learning et d'inférence IA sur les processeurs des ordinateurs Astro Pi. Elle succède aux bibliothèques tflite et pycoral, qui ne sont plus prises en charge par Google.

Utilisation

L'exemple ci-dessous montre comment charger et exécuter le modèle MobileNet v1 :

```
from ai_edge_litert.interpreter import Interpreter
from PIL import Image

interpreter = Interpreter("mobilenet_v1_0.25_224_quant.tflite")
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

height = input_details[0]["shape"][1]
width = input_details[0]["shape"][2]
img = Image.open("black.jpg").resize((width, height))
input_data = np.expand_dims(img, axis=0)
interpreter.set_tensor(input_details[0]["index"], input_data)
interpreter.invoke()
output_data = interpreter.get_tensor(output_details[0]["index"])
print(f"tflite: {np.squeeze(output_data)}")
```

Documentation

- [migration](#)
- [pretained models](#)

Information importante

En raison des exigences strictes en matière de sécurité applicables aux logiciels exécutés à bord de l'ISS, seules les bibliothèques tierces répertoriées ci-dessus peuvent être utilisées dans votre programme. Si vous pensez qu'il manque un élément important ou si vous avez des suggestions concernant d'autres bibliothèques, n'hésitez pas à nous en faire part.

Configuration de votre environnement de programmation

Nous vous recommandons d'utiliser Thonny pour écrire votre programme.

Installer Thonny

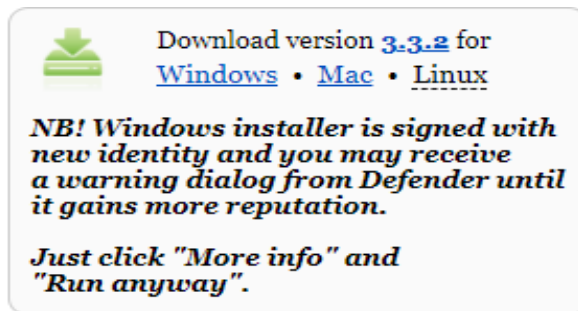
Installer Thonny sur un Raspberry Pi

- 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
- 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

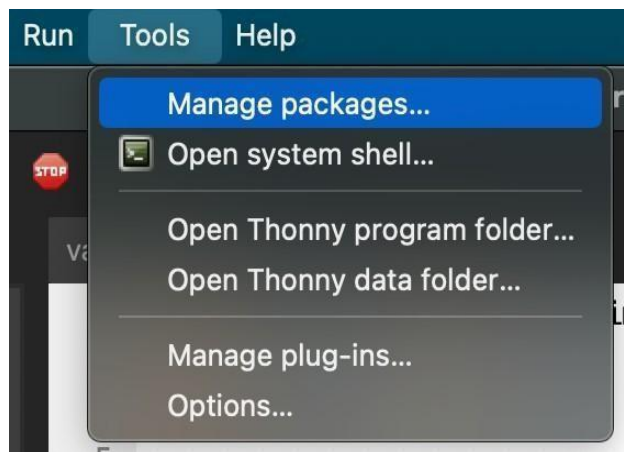
Ouvrez Thonny depuis votre lanceur d'application. Vous devriez obtenir une fenêtre qui ressemble à ceci :



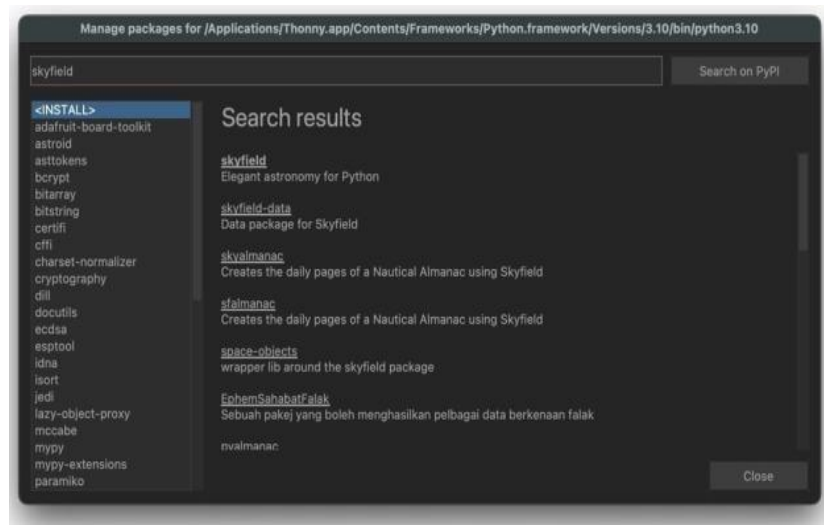
Vous pouvez utiliser Thonny pour écrire du code Python standard. Saisissez ce qui suit dans la fenêtre principale, puis cliquez sur le bouton **Run** (Exécuter) (il vous sera demandé d'enregistrer le fichier).

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

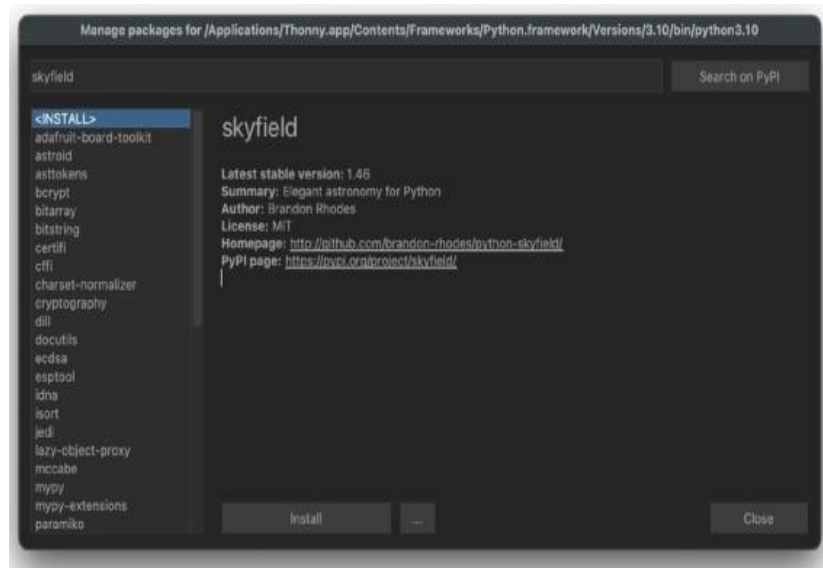
Pour installer l'une des bibliothèques Python, ouvrez Thonny et cliquez sur **Tools > Manage packages...**



Recherchez la bibliothèque qui vous intéresse en saisissant son nom dans la barre de recherche.



Sélectionnez le fichier approprié dans les résultats de recherche, puis cliquez sur **Install** (Installer).



Si vous utilisez un autre IDE pour écrire votre code, vous devrez suivre les instructions spécifiques à votre environnement pour télécharger les bibliothèques souhaitées depuis [PyPi](https://pypi.org/).

Planifier votre projet

Maintenant que vous avez configuré votre environnement de programmation, vous pouvez commencer à planifier votre projet Mission Space Lab. Nous estimons que la réalisation de la mission dans son intégralité prendra entre 2 et 6 heures.

Discutez avec votre équipe pour déterminer le sujet de votre projet, la manière dont vous allez collecter et analyser vos données, ainsi que la répartition des tâches. Établir un plan avant de commencer à coder vous aidera à rester organisé et à progresser de manière régulière.

Partagez vos idées, vos progrès et les éventuelles difficultés rencontrées avec le responsable d'équipe. Il ou elle pourra vous aider à affiner votre projet, à résoudre les problèmes et à faire avancer votre programme.

Écriture de votre programme

Nous vous recommandons d'écrire votre programme Python par petites étapes faciles à gérer, plutôt que d'essayer de tout rédiger d'un seul coup.

En suivant les étapes décrites ci-dessous, vous créerez les éléments essentiels de votre projet Mission Space Lab et vous testerez chaque élément au fur et à mesure.

Utiliser de l'IA dans votre code

Si votre équipe utilise des outils d'IA (tels que ChatGPT ou Gemini) pour faciliter l'écriture de code ou le débannage, vous devez expliquer clairement comment vous les avez utilisés.

Pour ce faire, vous pouvez :

- Ajouter des commentaires explicites dans votre code Python à côté des sections générées par IA.
- Décrivez votre utilisation de l'IA dans le formulaire de soumission du code.

1) Créer votre fichier main.py

Chaque soumission doit inclure un fichier nommé « main.py ». Ce fichier sert de point de départ à votre programme. C'est lui que le système automatisé de l'ISS recherchera et exécutera.

Vous pouvez inclure des fichiers Python supplémentaires dans votre soumission, mais votre programme doit démarrer à partir du fichier main.py. Si vous débutez sur Python ou si vous n'avez jamais travaillé avec plusieurs fichiers auparavant, nous vous recommandons de conserver l'intégralité de votre code dans le fichier main.py. Cela facilitera le développement, les tests et la soumission de votre programme.

- Créez un nouveau fichier nommé **main.py** dans le dossier de votre projet, puis enregistrez-le.

2) Collecter les données des capteurs

Votre programme doit utiliser au moins l'un des capteurs de l'Astro Pi ou la caméra pour collecter des données. L'utilisation de bibliothèques externes pour calculer ou prédire des valeurs n'est pas considérée comme une collecte de données provenant des capteurs ou de la caméra. Par exemple, un programme qui utilise uniquement la bibliothèque « skyfield » pour déterminer la position de l'ISS ne répondrait pas à cette exigence, car il utilise un modèle plutôt que des mesures provenant des capteurs de l'Astro Pi.

Prendre des mesures avec le Sense HAT

L'un des moyens de collecter des données sur l'ISS consiste à utiliser les capteurs du Sense HAT. Ces capteurs permettent de mesurer différentes variables, telles que la luminosité, la couleur, la température, l'humidité et le mouvement.

L'exemple ci-dessous présente un programme simple qui récupère une valeur de couleur et l'affiche sous forme de code RVB :

```
from sense_hat import SenseHat
sense_hat = SenseHat()
rgb = sense_hat.colour.colour
print(rgb)
```

Consultez l'activité « [Comment collecter les données de l'astro pi ?](#) » pour en savoir plus sur la manière de relever les données des capteurs à l'aide du Sense HAT.

Prendre des photos avec la camera

Une autre façon de collecter des données depuis l'ISS consiste à utiliser la caméra de l'Astro Pi. Cette caméra permet de prendre des photos de la Terre, qui pourront ensuite être analysées dans le cadre de votre étude.

L'exemple ci-dessous présente un programme simple pour prendre une photo :

```
from picamzero import Camera

camera = Camera()
camera.take_photo("image.jpg")
```

Consultez l'activité « [Prise en main des caméras astro pi](#) » pour en savoir plus sur l'utilisation de la caméra.

3) Enregistrer les données dans un fichier

Votre programme doit enregistrer les données qu'il recueille afin qu'elles puissent être transmises pour être analysées. Ces données peuvent être des relevés de capteurs enregistrés dans un fichier ou des images capturées par la caméra.

L'exemple ci-dessous montre comment enregistrer des données dans un fichier CSV :

```
import csv

data = 42

with open("data.csv", "w") as csvfile:
    writer = csv.writer(csvfile),
    writer.writerow([str(data)])
```

Information importante

Il existe des règles strictes concernant les types de fichiers que vous êtes autorisé à enregistrer. Veuillez à consulter [le règlement](#) pour connaître les types de fichiers autorisés.

4) Terminer dans le délai imparti de 10 minutes

Chaque programme de Mission Space Lab dispose d'exactly 10 minutes pour être exécuté à bord de l'ISS. Votre programme doit comptabiliser le temps écoulé depuis son lancement et s'arrêter automatiquement avant la fin des 10 minutes. Cela permettra de garantir que votre programme se termine correctement et que toutes les données que vous avez collectées soient enregistrées correctement.

Une façon d'y parvenir consiste à utiliser la bibliothèque **datetime** de Python :

- Noter l'heure à laquelle votre expérience commence.
- Vérifier régulièrement l'heure pendant que votre programme est en cours d'exécution.
- Arrêter le programme au bout de 10 minutes.

L'exemple ci-dessous utilise cette méthode pour afficher « Bonjour depuis l'ISS » toutes les secondes pendant 1 minute :

```
from datetime import datetime, timedelta
from time import sleep

# Create a variable to store the start time
start_time = datetime.now()
# Create a variable to store the current time
# (these will be almost the same at the start)
now_time = datetime.now()
# Run a loop for 1 minute
while (now_time < start_time + timedelta(minutes=1)):
    print("Hello from the ISS")
    sleep(1)
    # Update the current time
    now_time = datetime.now()
# Out of the loop - stopping
```

- Modifiez votre fichier **main.py** afin d'utiliser la bibliothèque **datetime** qui vous permettra d'arrêter votre programme avant la fin du créneau de 10 minutes.

Remarque : lorsque vous déterminez la durée d'exécution de votre programme, n'oubliez pas de tenir compte du temps nécessaire pour effectuer une itération.

Par exemple, si chaque boucle de votre programme dure environ 2 minutes, vous ne devez pas définir la durée d'arrêt à exactement 10 minutes. Vous devez plutôt arrêter la boucle au bout d'environ 8 minutes. Cela laissera suffisamment de temps à la dernière itération pour se terminer et garantira que votre programme s'arrête avant que la limite des 10 minutes ne soit atteinte.

5) Utilisez la bonne structure de répertoires pour vos fichiers de données

Lorsque votre code s'exécutera à bord de l'ISS, il sera lancé automatiquement par le système d'exploitation de vol de l'Astro Pi. Cela signifie que vous ne devez pas utiliser de chemins d'accès aux fichiers codés en dur dans votre code, tels que **/home/pi/Desktop**, car ces emplacements pourraient ne pas exister sur l'ordinateur de bord.

Utilisez plutôt la variable spéciale **__file__** pour déterminer l'emplacement de votre fichier **main.py** et enregistrez tous vos fichiers de données ou photos par rapport à cet emplacement. Cela garantira que vos fichiers sont stockés au bon endroit, quel que soit l'endroit où votre programme est exécuté.

L'exemple ci-dessous utilise la bibliothèque **pathlib** et la variable **__file__** pour enregistrer des fichiers dans le même répertoire que **main.py** :

```
from pathlib import Path

# Resolve the directory where main.py is currently running
dir_path = Path(__file__).parent.resolve()

# Create a safe file path inside your project directory
data_file = dir_path / "data01.csv"
```

- Vérifiez que votre code n'utilise pas de chemins d'accès absolus aux fichiers..

Exemple complet et fonctionnel

Cet exemple de code permet de prendre une photo et d'enregistrer cinq mesures du magnétomètre, puis d'enregistrer ces données dans un fichier.

Vous pouvez adapter le code pour capturer davantage d'images et de données, afin de profiter au maximum des 10 minutes dont dispose votre équipe.

```
from picamzero import Camera
from sense_hat import SenseHat
import csv

camera = Camera()
sense_hat = SenseHat()

# take a photo and save it in a file called image.jpg
camera.take_photo("image.jpg")

# create a variable to store the sensor readings
data = []

# repeat 5 times
for i in range(5):
    # read the raw x,y,z data from the magnetometer.
    # The contents of magnetometer_reading will be a dictionary
    # that looks like this:
    #
    # {
    #     "x": 7.907470226287842,
    #     "y": 5.466001033782959,
    #     "z": 12.868783950805664
    # }
    magnetometer_reading = sense_hat.get_compass_raw()

    # Take the numeric values from the magnetometer_reading
    # and put them in a list called magnetometer_values
    magnetometer_values = list(magnetometer_reading.values())
    # append the values to the data list to save later
    data.append(magnetometer_values)

# open a file called data.csv for writing
with open("data.csv", "w") as csvfile:
    writer = csv.writer(csvfile)

    # write a header to explain what the columns mean
    writer.writerow(["magnetometer_x", "magnetometer_y", "magnetometer_z"])
    # write all 5 readings from the data list to data.csv
    writer.writerows(data)
```

Améliorer votre programme

Cette partie propose des idées pour améliorer votre programme, ainsi que des liens vers d'autres guides de projet qui montrent comment les données et les images recueillies à bord de l'ISS peuvent être utilisées pour étudier différents phénomènes scientifiques.

Ces guides ont pour but de vous inspirer, mais vous n'êtes pas limité aux projets décrits ici. Vous êtes libre de concevoir et de mener votre propre projet scientifique, à condition que votre programme respecte [le règlement Mission Space Lab](#).

Prédire la position de l'ISS

Vous pourrez télécharger jusqu'à 42 photos prises depuis l'ISS. Il peut être intéressant de connaître l'endroit précis de l'orbite où une image a été prise. C'est quelque chose que vous pouvez faire facilement grâce aux bibliothèques « **astro_pi_orbit** » et « **exif** » disponibles sur les appareils Astro Pi.

Vous trouverez ci-dessous un exemple de programme permettant de créer une nouvelle image nommée **image_with_coordinates.jpg**. La bibliothèque **pica**zéro définira les métadonnées **Exif** de l'image afin d'y inclure la latitude et la longitude actuelles de l'ISS.

```
from astro_pi_orbit import ISS
from picamzero import Camera

iss = ISS()

def get_predicted_latlon_coordinates(iss):
    """
    Returns a tuple of latitude and longitude coordinates expressed
    in signed degrees minutes seconds.
    """
    point = iss.coordinates()
    return (point.latitude.signed_dms(), point.longitude.signed_dms())

cam = Camera()
cam.take_photo("image_with_coordinates.jpg", latlon_coordinates=get_predicted_latlon_coordinates(iss))
```

Vous devrez utiliser l'outil [Astro Pi Replay](#) pour exécuter cet extrait de code.

Capture de séquences d'images

Avec **pica**zéro, il est très simple de prendre une série de photos en appelant la fonction «**capture_sequence**». L'exemple ci-dessous permet de prendre trois photos à la suite, avec un intervalle de 3 secondes entre chacune d'elles :

```
# Import the Camera class from the picamzero (picamzero) module
from picamzero import Camera

# Create an instance of the Camera class
cam = Camera()

cam.capture_sequence("sequence", num_images=3, interval=3)
```

Exécutez ce code à l'aide de l'outil [Astro Pi Replay](#). Vous devriez alors constater qu'il prend 3 photos.

Numérotation des images et des fichiers

Lorsque vous traitez de nombreux fichiers d'un même type, il est conseillé de respecter une convention de nommage. Nous vous recommandons d'utiliser un ordre numérique clair, complété par des zéros en tête (**image01.png**, **image02.png**, etc.) afin de garder vos fichiers bien organisés. Vous pouvez utiliser cet extrait de code pour ajouter des zéros en tête des nombres :

```
file_number = 1
filename = f"image_{file_number:03d}.jpg"
print(filename)
```

Mise en mémoire des fichiers

Lorsque vous écrivez dans un fichier à l'aide de la fonction «**open**», Python n'enregistre généralement pas immédiatement le fichier sur le disque. Il conserve plutôt le contenu à enregistrer dans une zone de stockage temporaire de la mémoire de l'ordinateur, appelée «**tampon**». Python procède ainsi afin de pouvoir choisir le moment le plus opportun pour écrire sur le disque, ce qui n'a généralement aucune importance pour nous. Toutefois, tant que les données se trouvent dans le tampon et ne sont pas encore enregistrées sur le disque, elles risquent d'être perdues en cas d'erreur. Pour éviter cela, vous pouvez demander à Python d'enregistrer le contenu du tampon sur le disque à la fin de chaque ligne de texte en définissant l'argument **buffering** à 1 :

```
with open("some_file.txt", "w", buffering=1) as f:
    f.write("example data")
```

Information importante

si vous écrivez des données dans un fichier (avec l'argument «**wb**»), vous devez indiquer à Python de ne pas utiliser de tampon et d'écrire immédiatement les données sur le disque. Pour ce faire, définissez l'argument de mise en tampon sur 0.

Utilisation de vos données

Si vous le souhaitez, vous pouvez utiliser les données et les images que vous avez recueillies pour mener des recherches scientifiques ! Vous pouvez réaliser les expériences scientifiques de votre choix, à condition que votre projet respecte [le règlement](#). Si vous préférez suivre un guide tout prêt, vous pouvez choisir l'un des projets ci-dessous.

- **Using NDVI (normalised difference vegetation index)**

Découvrez comment traiter les données de lumière visible pour analyser la santé des plantes, la densité de la végétation et les caractéristiques environnementales à l'échelle de la Terre grâce à notre guide de projet intitulé « [Évaluer la santé des plantes avec le NDVI et le Raspberry Pi](#) ».

- **Calculating the speed of the ISS**

Vous pouvez commencer par apprendre à écrire un programme à partir de photos d'archives prises par des équipes les années précédentes, grâce à notre guide de projet « [Calculer la vitesse de l'ISS à l'aide de photos](#) ». Une fois votre programme écrit, vous pourrez le tester avec différentes images ou différents jeux de données afin d'améliorer la précision de votre estimation. Voici quelques exemples d'images et de données que vous pouvez utiliser :

- [Astro Pi Mission Space Lab 2022/23 photos](#)
- [Astro Pi Mission Space Lab 2022/23 data](#)

Tester votre programme

Check-list du programme

Que vous veniez tout juste de commencer à écrire votre programme ou que vous soyez sur le point de le terminer, il est essentiel de procéder à des tests réguliers. Tester au fur et à mesure vous aide à détecter les problèmes dès leur apparition et vous donne l'assurance que votre code s'exécutera correctement à bord de la Station spatiale internationale (ISS).

Assurez-vous que votre programme dispose des fonctions de base suivantes :

- ✓ Un fichier nommé main.py contenant votre code principal.
 - ✓ Lire les données d'un capteur Sense HAT ou prendre une photo avec l'appareil photo de l'ordinateur Astro Pi.
 - ✓ Enregistrer une photo ou des données de capteur dans un fichier.
 - ✓ Arrêtez avant 10 minutes.
- Vérifiez que votre programme respecte [le règlement du Mission Space Lab](#).

L'outil Astro Pi Replay

Tester un programme conçu pour fonctionner sur l'ISS peut sembler difficile lorsque l'on n'a pas accès à un Astro Pi. C'est pourquoi nous avons créé l'outil [Astro Pi Replay](#). Grâce à cet outil, vous pouvez tester votre programme sans avoir besoin de matériel spécifique.

Information importante

L'outil Astro Pi Replay simule l'environnement Astro Pi à l'aide des données recueillies lors des précédentes missions Astro Pi. Par exemple, lorsque votre programme prend une photo, l'outil Astro Pi Replay fournit l'une des images archivées capturées à bord de l'ISS au lieu d'accéder à une véritable caméra. Pour votre code, l'image apparaît comme si elle venait d'être prise, ce qui vous permet de développer et de tester vos programmes sans avoir besoin d'accéder au matériel Astro Pi.

L'outil Astro Pi Replay est disponible en deux versions :

- Une version en ligne, accessible depuis votre navigateur web.
- Une version hors ligne, disponible sous forme de plug-in Thonny.

Nous vous recommandons d'utiliser la version en ligne, car elle ne nécessite aucune installation et constitue le moyen le plus rapide de tester votre programme.

Accéder à l'outil Astro Pi Replay en ligne

Le moyen le plus simple de vérifier si votre programme fonctionnera à bord de l'ISS est de télécharger votre fichier **main.py** sur l'outil en ligne [Astro Pi Replay](#).

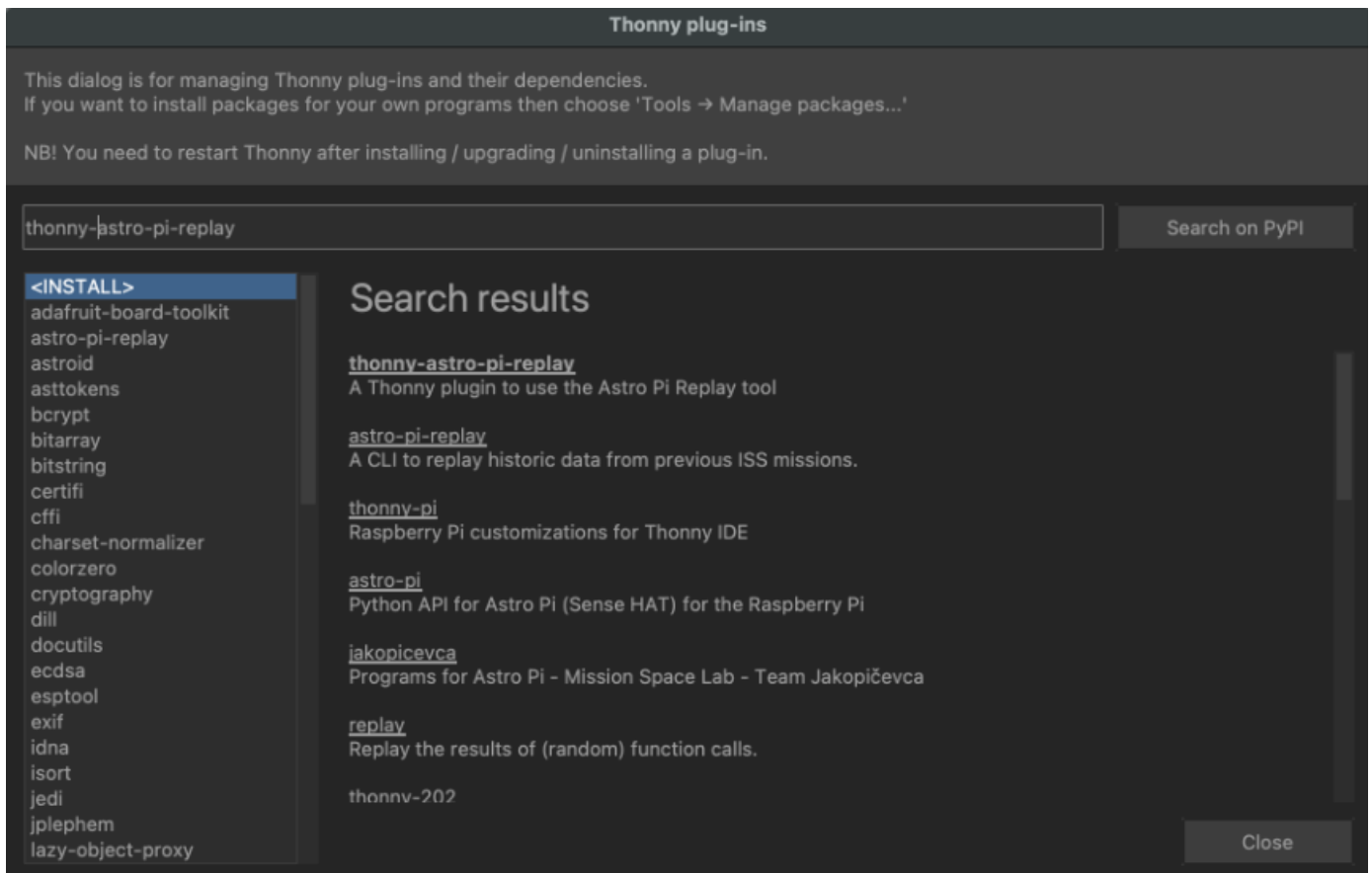
Pour exécuter votre programme, ouvrez le lien, puis glissez-déposez ou sélectionnez votre fichier **main.py** et cliquez sur « **Exécuter** ». L'outil Replay Tool exécutera votre programme dans son intégralité et vous montrera les images et les données que vous avez capturées, ainsi que tous les fichiers générés par votre programme.

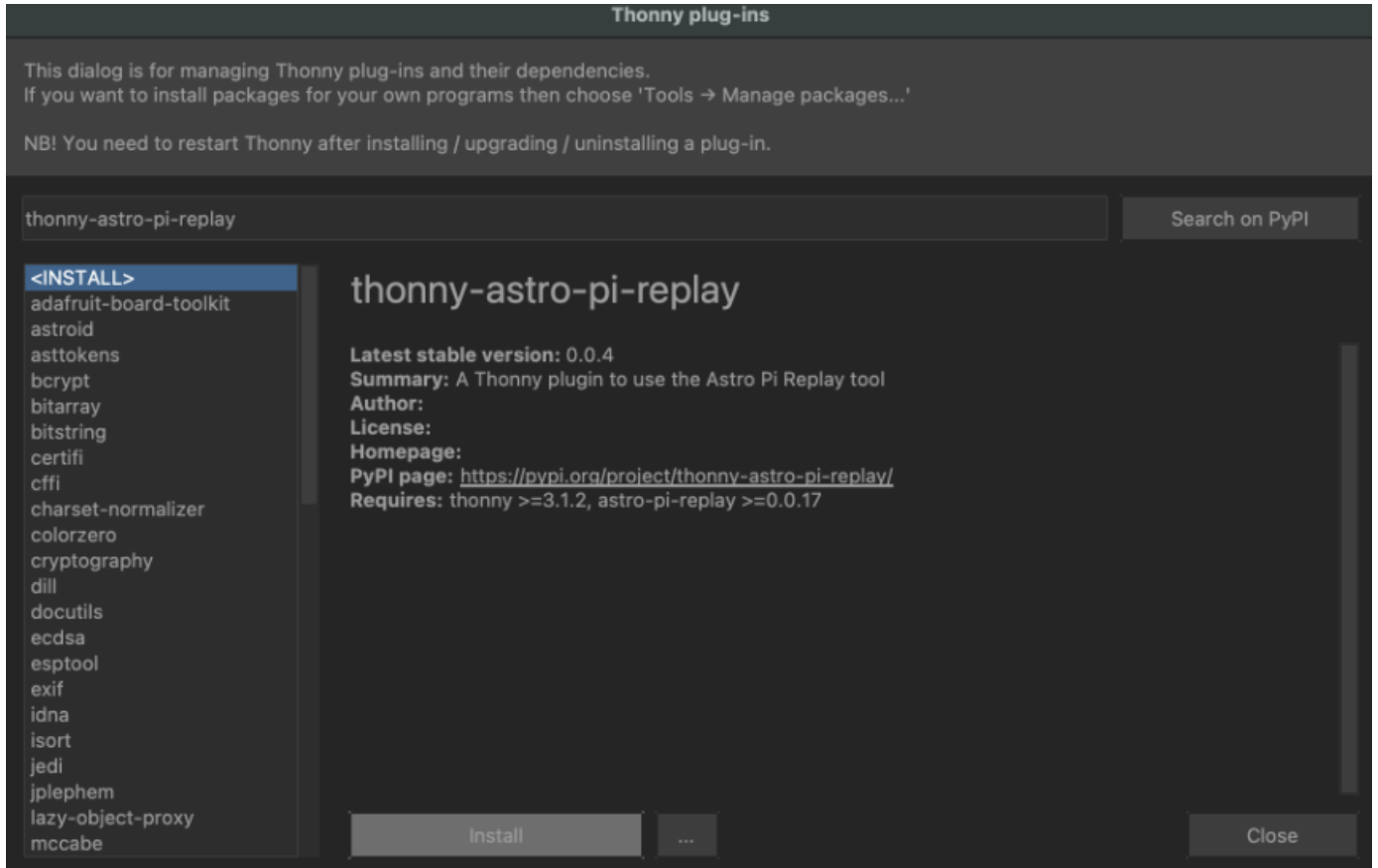
Installation du plug-in Astro Pi Replay pour Thonny

Si vous utilisez Raspberry Pi OS, vous devrez suivre les instructions relatives à la configuration de Thonny pour utiliser un environnement virtuel, disponibles sur [le site web de Raspberry Pi](#), avant de passer aux instructions ci-dessous.

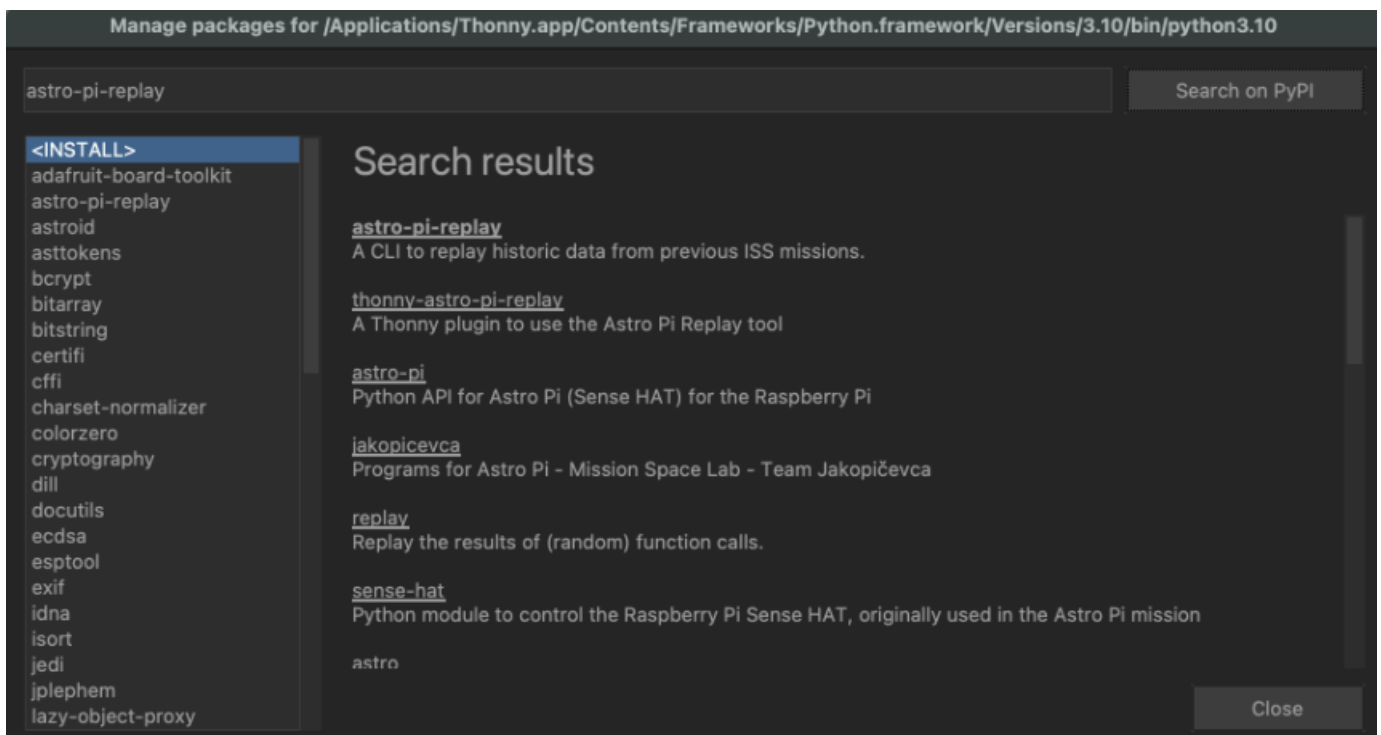
Si vous avez déjà participé au projet Mission Space Lab et que vous avez déjà installé l'outil Astro Pi Replay, vous devez réinstaller la **bibliothèque astro-pi-replay** afin de vous assurer que vous disposez bien de la dernière version. Pour ce faire, **supprimez le répertoire `~/.astro_pi_replay`** situé dans votre dossier personnel (par exemple, à l'aide de la **commande `rm -rf ~/.astro_pi_replay`** dans une fenêtre de Terminal), puis suivez les instructions ci-dessous.

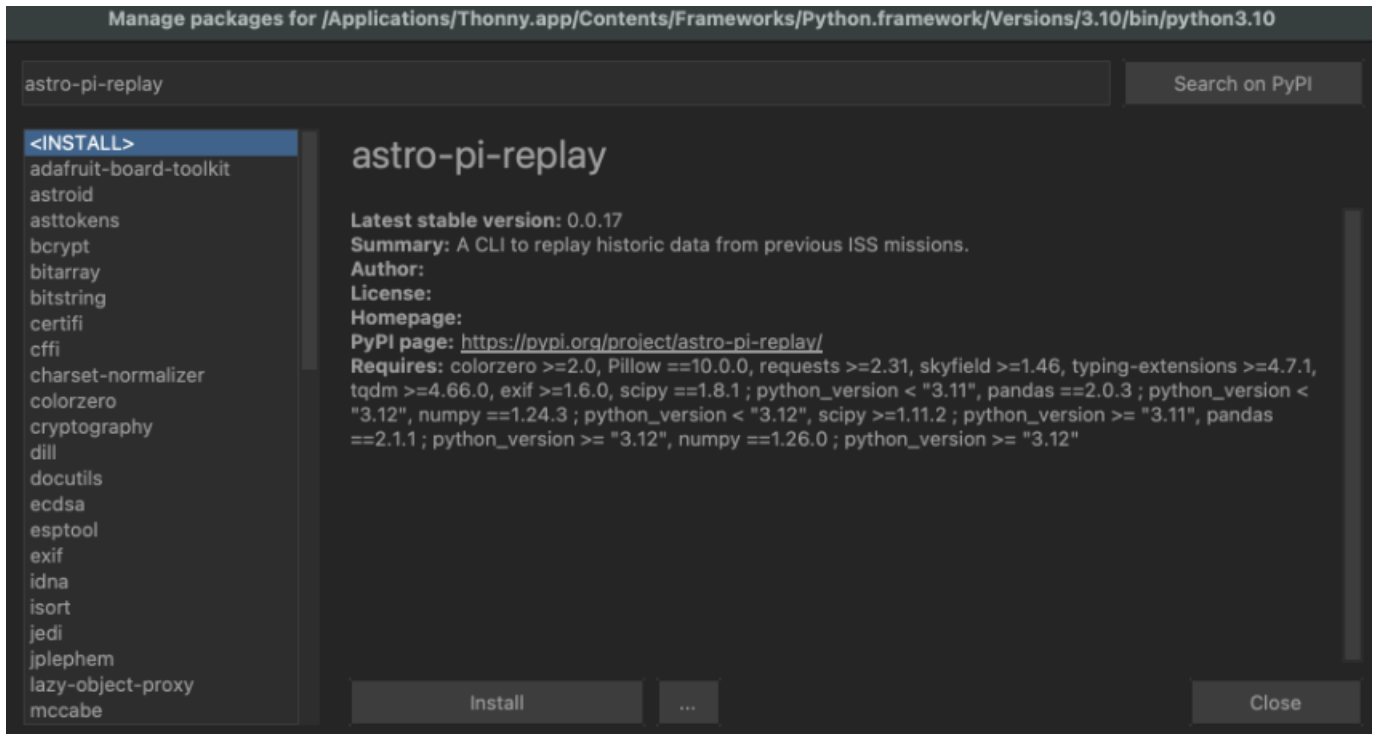
Pour installer l'outil Astro Pi Replay, ouvrez Thonny, cliquez sur **Outils > Gérer les plug-ins...**, puis recherchez « **thonny-astro-pi-replay** ». Sélectionnez le plug-in approprié, puis cliquez sur Installer.





Ensuite, cliquez sur **Outils > Gérer les paquets...**, puis recherchez « **astro-pi-replay** ». Sélectionnez le bon package, puis cliquez sur Installer.





Vous devrez ensuite fermer puis relancer Thonny pour que l'installation soit terminée.

Information importante

Bien que toutes les fonctions de la bibliothèque picamzero soient disponibles, de nombreux réglages et paramètres de picamzero qui devraient normalement permettre d'obtenir une image différente sont ignorés sans avertissement lorsque le code est exécuté à l'aide d'Astro Pi Replay. De plus, la plupart des attributs de l'objet Camera sont ignorés. Par exemple, définir l'attribut de résolution sur une valeur autre que (4056,3040) n'a aucun effet lors d'une simulation sur Astro Pi Replay, mais modifierait la résolution si le code était exécuté sur un Astro Pi en orbite.

Tester votre programme

Pour tester votre programme et simuler son exécution à bord de l'ISS, rendez-vous sur l'outil [Astro Pi Replay](#) et envoyez votre fichier de programme. Si vous utilisez le plug-in Astro Pi Replay avec Thonny, exécutez votre code **main.py** via ce plug-in en ouvrant le menu « **Run** » et en cliquant sur « **Astro-Pi-Replay** ».

Votre code doit être exécuté en moins de 10 minutes.

Si vous utilisez la version en ligne de l'outil Astro Pi Replay Tool, vous pouvez télécharger un fichier ZIP contenant les résultats de votre programme.

Une fois votre programme terminé, examinez les résultats et vérifiez les points suivants :

- Votre programme a-t-il créé le ou les fichiers de données attendus dans le dossier de votre projet ?
- D'autres fichiers de sortie ont-ils été créés ?
- Vos fichiers enregistrés sont-ils conformes au règlement, notamment en ce qui concerne la limite de stockage de 250 Mo et les types de fichiers autorisés ?
- Vos fichiers contiennent-ils des avertissements ou des erreurs ?

Si vous constatez des erreurs, des avertissements ou un comportement inattendu, modifiez votre code et testez-le à nouveau. L'outil Astro Pi Replay peut être utilisé autant de fois que vous le souhaitez. Vous pouvez donc continuer à peaufiner votre programme jusqu'à ce que vous soyez certain qu'il fonctionne de manière fiable et qu'il soit prêt à être soumis.

Des tests approfondis donneront à votre équipe les meilleures chances d'obtenir le statut « prêt pour le vol » et de voir votre programme exécuté à bord de l'ISS.

- **Testez votre programme à l'aide de l'outil Astro Pi Replay et vérifiez les résultats pour détecter d'éventuels problèmes ou comportements inattendus.**

Dépannage

Si votre programme rencontre des erreurs et ne fonctionne pas lors de votre premier test, voici quelques conseils de programmation utiles qui peuvent vous aider à les identifier et à les corriger.

Gestion des erreurs et des exceptions

Lorsque Python rencontre un problème, il peut générer une erreur ou une exception. Si votre programme ne gère pas cette situation, il risque de s'arrêter de manière inattendue.

En anticipant les problèmes potentiels et en traitant les exceptions, vous pouvez améliorer la fiabilité de votre programme. Au lieu de planter, votre code pourra ainsi réagir au problème et continuer à s'exécuter, ce qui réduira le risque de perdre des données ou des images précieuses pendant son séjour à bord de l'ISS.

Rendez-vous sur [le site Ada Computer Science](https://adafruit.com/learn/python/exceptions/) pour en savoir plus sur la gestion des exceptions.

- **Vérifiez votre programme et demandez-vous s'il est nécessaire de définir le mode de mise en mémoire tampon lors de l'écriture dans un fichier.**

Enregistrement

Si votre programme rencontre un problème, il peut être utile de disposer d'un historique des événements. L'enregistrement vous permet de suivre le comportement de votre programme et peut vous aider à identifier et à résoudre les problèmes lors de l'analyse de vos résultats après la mission.

[La bibliothèque logzero](https://projects.raspberrypi.org/en/projects/mission-space-lab-creator-guide/print) offre un moyen simple d'enregistrer des informations sur le fonctionnement de votre programme. Par exemple, vous pourriez consigner :

- Chaque itération d'une boucle
- Lorsqu'une fonction importante est appelée
- Les valeurs relevées par les capteurs ou des calculs clés
- Quelle partie d'une condition if/else a été exécutée ?

Veillez à ne pas enregistrer plus d'informations que nécessaire. Toutes les données enregistrées sont comptabilisées dans votre quota de téléchargement, et **les équipes peuvent télécharger au maximum 250 Mo de données** depuis la Station spatiale internationale (ISS).

L'exemple ci-dessous utilise **logzero** pour enregistrer chaque itération d'une boucle :

```
from logzero import logger, logfile
from time import sleep

logfile("events.log")

for i in range(10):
    logger.info(f"Loop number {i+1} started")
    ...
    sleep(60)
```

Les deux principaux types d'entrées de journal que vous pouvez utiliser sont «**logger.info()**» pour consigner des informations, et «**logger.error()**» lorsque votre programme rencontre une erreur inattendue ou gère une exception. Il existe également «**logger.warning()**» et «**logger.debug()**».

Information importante

Nous vous recommandons d'utiliser la bibliothèque logzero dans tous vos projets Mission Space Lab. L'enregistrement des événements importants peut vous aider à comprendre ce qui s'est passé au cours de votre expérience et à diagnostiquer les problèmes, même si vous enregistrez déjà les données des capteurs ailleurs.

La fermeture des ressources

Une fois votre programme terminé, il est recommandé de fermer toutes les ressources encore ouvertes. Cela inclut les fichiers, les caméras et tout autre élément utilisant les ressources du système.

La fermeture des ressources permet de s'assurer que toutes les données sont correctement enregistrées et que votre programme se ferme correctement.

Par exemple, si vous avez ouvert un fichier, vous devez le fermer une fois que vous avez fini d'écrire dedans :

```
file = open(file)
file.close()
```

- **Vérifiez votre fichier main.py et modifiez-le afin qu'il ferme correctement toutes les ressources.**

Erreurs communes

Certaines équipes de Mission Space Lab ont rencontré des problèmes qui ont empêché leurs programmes d'atteindre le statut « prêt au vol » ou de fonctionner correctement à bord de l'ISS.

La liste ci-dessous met en évidence certaines des erreurs les plus courantes et explique pourquoi elles peuvent poser des problèmes dans l'environnement de vol. Tous les points ne s'appliqueront pas nécessairement à votre projet, mais un examen attentif de cette liste pourra vous aider à identifier d'éventuels problèmes avant de soumettre votre code.

Enregistrer les coordonnées de l'ISS à partir de «skyfield» ou «astro_pi_orbit» au lieu d'utiliser un capteur

Votre code doit enregistrer des données provenant d'un capteur ou sauvegarder une photo dans un fichier. L'utilisation de « skyfield » ou « astro_pi_orbit » pour enregistrer les coordonnées actuelles de l'ISS n'est pas prise en compte, car cette méthode consiste à rechercher la position de l'ISS dans un tableau de positions prévues et n'utilise pas réellement de capteur ni d'appareil photo.

Ouvrir et fermer l'appareil photo à plusieurs reprises

Si vous créez plusieurs objets « Camera », par exemple dans une boucle, vous risquez de saturer la mémoire du Raspberry Pi et que votre programme ne soit pas accepté par Astro Pi Mission Control.

Enregistrer plus de 42 images

Votre programme ne doit pas conserver plus de 42 images à l'issue des 10 minutes, même s'il peut en stocker davantage pendant son exécution.

Données utilisateur

Votre programme ne peut pas fonctionner s'il repose sur une interaction avec un astronaute.

Utilisation de la matrice LED

Votre programme n'est pas autorisé à utiliser la matrice LED.

Utilisation des chemins d'accès absolus aux fichiers

Veillez à ne pas utiliser de chemins d'accès spécifiques pour vos fichiers de données. Utilisez la **variable `__file__`**.

Ne pas enregistrer les données immédiatement

Veillez à ce que toutes les données expérimentales soient enregistrées dans un fichier dès leur collecte. Évitez d'enregistrer les données au fur et à mesure dans une liste ou un dictionnaire interne, puis de les transférer dans un fichier à la fin de l'expérience, car si celle-ci s'interrompt brusquement en raison d'une erreur ou du dépassement de la limite de 10 minutes, vous ne disposerez d'aucune donnée.

Manque d'espace

Vous pouvez générer jusqu'à 250 Mo de données. N'oubliez pas que la taille d'un fichier image dépend non seulement de la résolution, mais aussi du niveau de détail de la photo : une photo d'un mur blanc uni sera moins volumineuse qu'une photo de paysage. L'utilisation d'Astro Pi Replay vous donnera une bonne idée du nombre de photos que vous pourrez prendre.

Oublier d'appeler votre fonction

Dans certains cas, des équipes ont écrit une fonction mais ont oublié de l'appeler dans leur programme `main.py`. Soyez vigilants !

Enregistrement dans des répertoires qui n'existent pas

De nombreuses équipes souhaitent organiser leurs données dans des répertoires tels que «data», «images», etc. C'est en soi une très bonne chose, mais on oublie facilement de créer ces répertoires avant d'y enregistrer des données.

Mise en réseau

Pour des raisons de sécurité, votre programme n'est pas autorisé à accéder au réseau de l'ISS. Il ne doit pas tenter d'ouvrir un serveur, d'accéder à Internet ni d'établir une connexion réseau de quelque nature que ce soit. Cela inclut les connexions au réseau local vers l'Astro Pi lui-même.

Essayer de lancer un autre programme

En plus de ne pas être autorisé à utiliser le réseau, votre programme n'est pas autorisé à exécuter un autre programme ni aucune commande que vous taperiez normalement dans la fenêtre de terminal du Raspberry Pi, telle que « `vcgencmd` ».

Plusieurs tâches en parallèle

Si vous devez effectuer plusieurs tâches simultanément, vous pouvez utiliser un processus multithread. Il existe plusieurs bibliothèques Python qui permettent d'intégrer ce type de multitâche dans votre code. Cependant, pour ce faire sur les Astro Pi, **vous n'êtes autorisé à utiliser que la bibliothèque ``threading``**.

N'utilisez la bibliothèque de threads qu'en cas d'absolue nécessité. La gestion des threads peut s'avérer délicate, et comme votre programme s'exécutera au sein d'une séquence comprenant de nombreux autres programmes, nous devons nous assurer que le programme précédent s'est terminé sans problème avant de lancer le suivant. Les threads indésirables peuvent se comporter de manière imprévisible et mobiliser une part excessive des ressources système. Si vous utilisez des threads dans votre code, vous devez vous assurer qu'ils sont tous gérés avec soin et fermés correctement à la fin de votre programme. Vous devez également veiller à ce que les commentaires de votre code expliquent clairement comment cela est réalisé.

Temps d'exécution du programme trop court

Certaines équipes règlent la durée d'exécution de leur programme sur une valeur faible (par exemple, 1 minute) à des fins de test, puis oublient de la rétablir à une valeur appropriée. Veuillez à utiliser autant que possible le créneau horaire qui vous a été attribué.

Erreur de division par zéro

Si votre programme tente de calculer la vitesse de l'ISS, vous devrez vous assurer qu'il n'essaie pas de diviser par zéro lors du calcul de la vitesse. Cela peut se produire lorsque les plages de temps des images sont arrondies à la seconde près (par exemple, lorsque vous utilisez la commande **datetime_digitized**). Si votre code prend deux photos en moins d'une seconde, celles-ci peuvent sembler avoir le même horodatage, ce qui provoquera une erreur « **ZeroDivisionError** » et entraînera le plantage de votre programme.

Pour éviter cela, vous pouvez ajouter une commande « **sleep** » à votre code. Cela garantira un intervalle d'au moins 1 seconde entre chaque photo :

```
from time import sleep
from picamzero import Camera

camera = Camera()
camera.take_photo("image1.jpg")
sleep(1)
camera.take_photo("image2.jpg")
```

Ne pas utiliser d'images en pleine résolution pour calculer la vitesse de l'ISS

Si vous utilisez un code issu du projet « [Calculer la vitesse de l'ISS à l'aide de photos](https://projects.raspberrypi.org/en/projects/mission-space-lab-creator-guide/print) », sachez que la distance d'échantillonnage au sol (GSD) varie en fonction de la résolution de l'image.

La valeur indiquée dans le guide du projet « Calculer la vitesse de l'ISS à l'aide de photos » n'est valable que pour les images en pleine résolution (4056 × 3040), mais pas nécessairement pour les résolutions inférieures. C'est pourquoi nous vous recommandons de prendre des photos en pleine résolution.



Cv2.Error lors du calcul des caractéristiques de l'image

Si vous suivez le guide « [Calculer la vitesse de l'ISS à l'aide de photos](#) » et que les photos que vous comparez manquent de contraste, la fonction «**calculate_features**» pourrait renvoyer «**None**» au lieu des données de description de l'image. Il s'agit d'une cause fréquente de bugs, car les fonctions suivantes, telles que «**calculate_matches**», attendent un tableau NumPy et se bloqueront si elles reçoivent «**None**» à la place.

Pour éviter que votre programme ne plante lorsque cela se produit, vous pouvez encapsuler votre code dans une instruction «**try-except**», comme ceci :

```
cv_error_count = 0
max_cv_errors = 5

for i in range(10):
    try:
        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)

        # successfully calculated the speed - reset counts
        cv_error_count = 0
    except cv2.error as e:
        cv_error_count += 1
        if cv_error_count >= max_cv_errors:
            raise e
```

Cet extrait de code tente de calculer les caractéristiques de vos images, mais au lieu de se bloquer lorsqu'il rencontre une erreur cv2.error, il passe à l'étape suivante et réessaie. Étant donné que le paysage et l'éclairage changent constamment sous la Station spatiale internationale (ISS), le problème se résout souvent de lui-même lors de l'itération suivante — mais cela n'est malheureusement pas garanti à 100 %. C'est pourquoi cet extrait de code comptabilise le nombre d'erreurs rencontrées et déclenche à nouveau l'exception cv2.error si plus de 4 erreurs se produisent d'affilée. Si cela se produit, Astro Pi Mission Control fera de son mieux pour réexécuter votre code dans de meilleures conditions.

- **Relisez votre programme. Identifiez-vous une des erreurs courantes dans votre programme ?**

Soumettre votre programme

Check-list du programme

Avant de soumettre votre projet, vérifiez de nouveau que votre programme répond aux exigences suivantes :

- ✓ Un fichier nommé main.py contenant votre code principal.
- ✓ Lire les données d'un capteur Sense HAT ou prendre une photo avec l'appareil photo de l'ordinateur Astro Pi.
- ✓ Enregistrer une photo ou des données de capteur dans un fichier.
- ✓ Arrêtez avant 10 minutes.
- ✓ Le code fonctionne correctement, sans erreur, lors des tests

- **Vérifiez que votre programme respecte [le règlement du Mission Space Lab](#).**

Préparation de votre soumission

Lorsque vous soumettez votre projet, vous devrez fournir un fichier zip contenant les fichiers de votre programme, y compris votre fichier **main.py**.

Si vous ne savez pas comment créer un fichier zip, adressez-vous à votre responsable d'équipe.

Si vous utilisez la version en ligne d'Astro Pi Replay, vous pouvez télécharger un fichier zip contenant les résultats générés lors de vos essais. **Il ne s'agit pas du fichier zip que vous devez envoyer pour votre soumission.**

- **Créez un fichier ZIP contenant votre projet et demandez à votre responsable d'équipe de nous l'envoyer avant la date limite.**

Soumettre votre projet

Si vous êtes satisfait de votre programme, que vous l'avez testé à l'aide d'Astro Pi Replay, que vous avez lu le règlement de Mission Space Lab et que vous avez rempli la checklist ci-dessus, votre projet est prêt à être soumis.

Demandez au responsable d'équipe de soumettre votre fichier zip au nom de votre équipe.

Ressources supplémentaires

Retrouvez ci-dessous une multitude de ressources pour vous aider à réussir à chaque étape de votre parcours Astro Pi :

- [Prise en main des caméras](#)
- [Prise en main de Sense Hat](#)
- [Collecter les données de l'Astro Pi](#)
- [Mesurer la santé des plantes avec NDVI et Raspberry Pi](#)
- [Calculer la vitesse de l'ISS à partir d'images](#)
- [Boîtier de vol Mark II imprimé en 3D](#)
- [Construire son propre banc d'essais Astro Pi](#)
- [Les tutoriels Python d'OpenCV](#) expliquent comment réaliser toutes sortes de choses intéressantes, comme l'apprentissage automatique, la détection des contours et le suivi d'objets.
- [La galerie d'exemples d'EarthPy](#) vous explique étape par étape comment traiter des images satellites.
- [Cette page GitHub](#) contient une sélection de ressources consacrées à l'espace.
- [Cette liste de jeux de données satellitaires annotés](#) peut être utilisée pour entraîner n'importe quel modèle de machine learning.
- Les équipes à la recherche d'un défi plus ambitieux trouveront ici une ressource complète sur [l'utilisation du machine learning avec des images aériennes et satellitaires](#). Attention toutefois : ce sujet est assez pointu.