Astro Pi Mission Space Lab
Guide de la phase 2

Guide pour les participants au concours Mission Space Lab Astro Pi

Étape 1 Introduction

Le challenge européen Astro Pi est un projet éducatif de l'ESA organisé en collaboration avec la fondation Raspberry Pi. Astro Pi Mission Space Lab offre aux jeunes l'incroyable opportunité de mener des recherches scientifiques dans l'espace en écrivant des programmes informatiques exécutés sur des ordinateurs Raspberry Pi, les Astro Pi, à bord de la Station spatiale internationale.

Ce guide couvre la phase 2 des 2 thèmes de Mission Space Lab : *La vie sur Terre* et *La vie dans l'espace*. Nous souhaitons que votre expérience soit réalisée en toute fiabilité à bord de l'ISS, et ce guide va vous aider à vous lancer rapidement et vous offrir les meilleures chances de mener votre programme à bien sans problème.

Collaboration durant la pandémie de COVID-19

Nous sommes conscients du fait que les restrictions imposées durant la pandémie de COVID-19 peuvent impacter négativement la capacité des équipes à travailler ensemble sur leur kit Astro Pi. Pour aider les équipes à mener ces sessions en ligne et à collaborer à distance, nous avons créé des modèles de plans de cours pour les sessions.

Première session (https://rpf.io/first-session-spacelab)
Sessions suivantes (https://rpf.io/ongoing-sessions-spacelab)

Dans le cadre de la collaboration à distance avec votre équipe, il est important de s'assurer de la mise en œuvre de politiques et procédures rigoureuses de protection des enfants, mais aussi de se conformer à la législation en vigueur.

Nous recommandons l'application des directives de protection (https://rpf.io/safeguarding) et des meilleures pratiques de la fondation Raspberry Pi.

Ce que vous allez réaliser

Ce guide fournit des informations sur l'assemblage de votre kit, sur l'écriture du code de votre expérience et sur le test de votre programme. Il contient aussi des détails essentiels sur ce que les éléments matériels et logiciels Astro Pi permettent et ne permettent pas de faire.

Nous ne tenons pas à ce que vous suiviez chaque règle, mais si vous pensez devoir faire les choses différemment, contactez-nous avant de soumettre votre projet. Un programme qui ne suit pas ce guide obtiendra un score inférieur à un programme conforme durant le processus d'évaluation, et il risque de ne pas accéder à la phase suivante s'il ne peut pas être exécuté facilement à bord de l'ISS sans modification.

La plupart des informations dans ce guide s'appliquent aux deux expériences « La vie dans l'espace » et « La vie sur Terre ». Cependant, quelques différences doivent être prises en compte en fonction du thème choisi.

Remarques concernant l'expérience « La vie dans l'espace »

- Bien que vous puissiez utiliser la caméra dans le cadre de votre expérience (par exemple pour déterminer la luminosité à bord de l'ISS), vous ne pouvez pas l'utiliser pour prendre des photos des astronautes et votre programme doit effacer toutes les images à la fin de votre expérience.
- Votre programme doit afficher un message ou une image explicite sur la matrice à LED du Sense HAT, afin que les astronautes à proximité sachent qu'une expérience est en cours. Cet élément doit évoluer régulièrement pour indiquer que tout fonctionne correctement.

Remarque concernant l'expérience « La vie sur Terre »

- Vous ne devez pas utiliser la matrice à LED pendant que votre expérience est en cours. L'Astro Pi sera recouvert pour éviter que la lumière parasite ne gâche les images prises depuis le hublot.

Même si vous avez déjà participé au concours Astro Pi auparavant, prenez le temps de lire et de suivre ce guide, car de nombreuses choses ont changé depuis les éditions précédentes.

Ce que vous allez apprendre

Vous allez apprendre à connecter le matériel requis au Raspberry Pi (Sense HAT et module de caméra) et comment convertir votre idée de la phase 1 du Mission Space Lab en une expérience fonctionnelle en écrivant un programme Python qui peut être exécuté sur les Astro Pi de l'ISS.

Ce dont vous avez besoin

Matériel

- Un Raspberry Pi
- Un Sense HAT
- Un module de caméra (s'il est nécessaire pour votre expérience)

Logiciels

Vous aurez besoin du système d'exploitation (OS) de l'Astro Pi, qui est disponible en deux versions : Desktop et Flight. Ce sont des versions personnalisées du système d'exploitation de Raspberry Pi, qui comprennent toutes les bibliothèques logicielles présentes sur les unités Astro Pi à bord de l'ISS.

Ressources supplémentaires

- Si vous le souhaitez, vous pouvez réaliser un modèle de vol imprimé en 3D (https://esero.fr/wp-content/uploads/2021/06/3D-PrintedAstroPiFlightCase_RaspberryPiProjects_FR.pdf) et l'utiliser pour simuler de façon encore plus réaliste l'environnement de l'ISS pour votre expérience. Cependant, ce n'est pas une obligation et vous pouvez participer au Mission Space Lab sans fabriquer votre réplique de flight case.

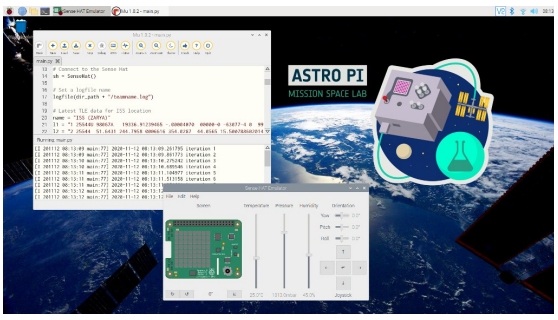
Étape 2 Mise en route

Si vous avez reçu un kit Astro Pi officiel de l'ESA, vous avez tout ce qu'il te faut pour développer et tester votre programme de la phase 2 du Mission Space Lab (MSL). Si vous le souhaitez, vous pouvez même réaliser votre propre modèle de vol Astro Pi (https://esero.fr/wp-content/uploads/2021/06/3D-PrintedAstroPiFlightCase_RaspberryPiProjects_FR.pdf) mais ne vous inquiétez pas, ce n'est pas indispensable et vous pouvez mener à bien votre programme Mission Space Lab sans avoir de modèle de vol.

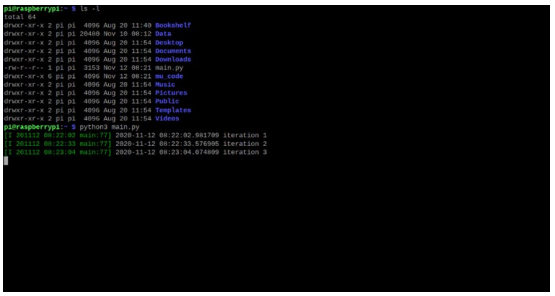
La première chose à faire consiste à configurer votre environnement de développement. Si vous avez reçu un kit de l'ESA, ce dernier contient deux cartes SD avec les deux versions personnalisées distinctes du système d'exploitation Raspberry Pi dont vous avez besoin.

Images des cartes SD

La version Desktop du système d'exploitation offre toutes les fonctionnalités d'un système d'exploitation normal et une série d'outils de programmation. Voici une présentation vidéo (https://esero.fr/tutoriels-en-ligne/decouverte-du-kit-astro-pi/) ! Utilisez cette version pour *développer* et *tester* le code de votre expérience.



La version Flight ressemble au système d'exploitation installé sur les unités Astro Pi à bord de l'ISS. Elle ne comprend aucune application X-Windows ou GUI et est « à ligne de commande uniquement », de sorte que ce n'est pas une plateforme utile pour créer votre programme. Cependant, nous recommandons fortement l'utilisation de cette version pour le *test final* de votre expérience avant de la soumettre, afin de vérifier que le programme que vous avez écrit s'exécute sans erreur. C'est le meilleur moyen de vous assurer que votre expérience passera avec succès notre procédure de test et pourra être exécutée sur les Astro Pi à bord de l'ISS sans modification.



Les paquets installés sur les versions Desktop et Flight de l'OS correspondent à ceux disponibles sur les ordinateurs Astro Pi à bord de l'ISS. Vous ne devez pas effectuer de mises à niveau ni installer de paquets ou de bibliothèques Python supplémentaires, afin de conserver la configuration disponible pour exécuter votre expérience.

Utilisation du matériel

Vous pouvez visionner de nouveau la vidéo présentant comment proposer une idée pour le projet Astro Pi (https://esero.fr/projets/astro-pi/) afin de vous rappeler les limitations du matériel Astro Pi à bord de l'ISS. Nous avons aussi les ressources suivantes pour vous initier au Sense HAT et au module de caméra :

Si vous n'avez jamais utilisé le Sense HAT auparavant, commencez par ce petit projet (https://esero.fr/wp-content/uploads/2021/06/DebuterAvecLeSenseHAT_RaspberryPi_FR.pdf) pour découvrir les utilisations de base du Sense HAT.

Notez que comme vous allez utiliser le module de caméra Raspberry Pi avec le Sense HAT, vous allez devoir faire passer le câble plat de la caméra à travers la fente du Sense HAT avant de le connecter au Raspberry Pi.

Si vous n'avez jamais utilisé le module de caméra auparavant, commencez par ce projet pour débutant (https://esero.fr/wp-content/uploads/2021/04/Premiers-pas-avec-le-module-camera-_-Raspberry-Pi-Projects_FR.pdf) afin de tester les fonctions de base de la bibliothèque Python `picamera`.

Remarque : les caméras sur les Astro Pi à bord de l'ISS correspondent au module de caméra V1 disponible en 2014. Le module de caméra suivant V2 que vous pouvez acheter aujourd'hui (et qui est inclus dans les kits de l'ESA) est doté d'un capteur mis à niveau capable de produire des images à des résolutions non disponibles sur le modèle V1. Aussi, vous devez vous assurer que la résolution spécifiée dans votre code est disponible sur le module de caméra V1. Vous trouverez des informations plus détaillées dans cette section de la documentation relative à la bibliothèque PiCamera (https://picamera.readthedocs.io/en/release-1.13/fov.html?highlight=v2#sensor-modes), mais voici des exemples de résolutions typiques que vous pouvez utiliser avec le module de caméra V1 :

2592×1944, 1920×1080, 1296×972, 1296×730 et 640×480

Performance

Les Raspberry Pi dans les kits de l'ESA pour Astro Pi 2019/20 sont les Raspberry Pi 3. Cependant, les Astro Pi actuellement à bord de l'ISS correspondent au modèle plus ancien Raspberry Pi B+. Le Raspberry Pi 3 est un modèle plus récent, et donc plus rapide et plus puissant que le B+. Vous devez garder ce point à l'esprit lorsque vous écrirez le code de votre expérience : certaines tâches nécessitant une grande puissance de calcul (impliquant, par exemple, des calculs complexes ou le traitement d'une grande quantité de données) seront effectuées plus lentement sur les Astro Pi à bord de l'ISS que sur le Raspberry Pi dans votre kit de l'ESA. En particulier, l'utilisation de bibliothèques Python, comme `OpenCV` (pour traiter les images réalisées par le module de caméra) ou `ephem` (pour déterminer quelle ville l'ISS est en train de survoler), sera significativement plus lente sur les Astro Pi.

La version Flight de l'OS contient quelques paramètres qui limitent délibérément les performances du Pi, afin de se rapprocher au plus près des capacités des Astro Pi à bord de l'ISS.

Étape 3 Écriture de votre programme

À présent, vous pouvez commencer à écrire le programme de votre expérience. Pour cela, vous devez planifier vos sessions de codage, déterminer le meilleur moyen d'écrire le programme de votre expérience et vous assurer qu'il fonctionnera sur les Astro Pi de l'ISS.

Pour vous aider dans votre planification, nous avons rédigé un guide pour la phase 2 qui propose des conseils utiles visant à faciliter le codage de votre équipe.

Quelle version de Python utiliser ?

Tous les programmes soumis dans le cadre du concours MSL doivent être écrits en Python 3.

Si vous trouvez une bibliothèque Python dont vous avez besoin pour votre expérience et qui ne peut être utilisée qu'en Python 2, contactez-nous — nous vous aiderons à trouver une approche alternative.

Bibliothèques Python

Nous avons installé une collection de bibliothèques Python sur l'OS des Astro Pi. Voici quelques informations concernant ce que vous pouvez en faire et où trouver la documentation correspondante.

Souvenez-vous que vous pouvez télécharger l'OS Astro Pi pour accéder à toutes ces bibliothèques sur votre carte SD.

Notez qu'aucune autre bibliothèque ne peut être utilisée pour votre expérience Mission Space Lab. Si votre expérience nécessite d'autres bibliothèques Python, contactez-nous et nous vous aiderons à trouver une approche alternative.

Certaines bibliothèques Python peuvent inclure des fonctions qui exécutent une requête Web pour rechercher des informations ou retourner une valeur dépendante de l'heure ou du lieu. Même si elles peuvent s'avérer très utiles, elles ne sont pas autorisées (voir la section « Mise en réseau » de ce guide).

Comment nommer vos fichiers Python Mission Space Lab

Pour soumettre le programme de votre expérience MSL, votre fichier Python principal doit être nommé `main.py`.

Idéalement, tout votre code doit être contenu dans ce fichier. Cependant, si votre expérience est très complexe, des fichiers supplémentaires sont autorisés.

Documentation de votre code

Lorsque vous créez un programme ou un logiciel très utile et que vous souhaitez le partager avec d'autres, une étape cruciale consiste à créer une documentation permettant de comprendre ce que fait le programme, comment il fonctionne et comment il peut être utilisé. Ce point est particulièrement important pour votre expérience MSL, car votre programme doit montrer clairement comment atteindre les objectifs de votre expérience.

Ce projet (https://projects.raspberrypi.org/en/projects/documenting-your-code) illustre comment ajouter des commentaires utiles à votre programme.

Toute tentative pour masquer ou compliquer la compréhension de ce que fait un code se soldera par une disqualification. Et bien évidemment, votre code ne doit pas contenir de langage grossier ni d'incivilités.

Étape 4 Collecte et enregistrement de données avec votre programme

Tout fichier créé par votre programme doit avoir un nom pertinent et informatif. Utilisez uniquement des lettres, des chiffres, des tirets (-) ou des caractères de soulignement (_) pour nommer vos fichiers. N'utilisez pas d'espaces dans les noms de fichiers, sous peine de générer des problèmes lors du transfert des fichiers entre ordinateurs.

Votre programme doit collecter et stocker les données de votre expérience. Ces mesures doivent être enregistrées dans un fichier du répertoire de travail en cours appelé `data01.csv`.

L'extension `.csv` indique qu'il s'agit d'un fichier de valeurs séparées par des virgules, dans lequel vos données seront enregistrées sous forme de tableau, chaque valeur individuelle étant séparée des autres par une virgule. Voici, par exemple, un fragment de code issu d'un fichier au format CSV qui contient des données de date, d'heure, d'humidité et de température enregistrées à intervalles d'environ une minute.

```
Date, Time, Humidity, Temperature
05/05/2018, 10:23:56, 45.60, 21.05
05/05/2018, 10:24:58, 45,62, 21,10
05/05/2018, 10:25:57, 45,68, 21,10
05/05/2018, 10:26:58, 45,72, 21,13
```

Pour créer et enregistrer facilement un fichier CSV, vous pouvez utiliser la bibliothèque `csv`, comme illustré dans l'exemple ci-dessous.

Si vous avez besoin de plusieurs fichiers de données, ils doivent être numérotés de manière séquentielle (par ex. `data02.csv`, `data03.csv`, etc.). Vous ne devez pas créer plus de cinq fichiers distincts tout au long de votre expérience.

Structure du répertoire des fichiers

Tous les fichiers journaux doivent être enregistrés au même endroit que le fichier Python lui-même lors de l'exécution des Astro Pi sur l'ISS. Vous ne devez pas utiliser un chemin spécifique dans votre code (par exemple, `/home/pi/Desktop` n'existera pas sur les Astro Pi à bord de l'ISS).

Lorsque votre code est exécuté à bord de l'ISS, il est démarré et arrêté par un système automatisé. Pour vous assurer que vos fichiers de données termineront au bon endroit, vous devez mettre en œuvre la méthode ci-dessous, qui utilise la variable `__file` spéciale contenant le chemin vers le fichier que Python est en train d'exécuter. Vous pouvez utiliser cette variable pour trouver le chemin du fichier avec la bibliothèque `pathlib`.

```
from pathlib import Path

dir_path = Path(__file__).parent.resolve()
```

Enregistrement des données dans un fichier CSV

Si vous souhaitez enregistrer les données du Sense HAT, nous recommandons d'utiliser un `csv` issu de la bibliothèque standard Python. Vous devez créer une variable chaîne contenant le chemin complet vers votre fichier CSV, ouvrir le fichier (avec droits d'écriture) et ajouter une ligne d'en-tête avant l'écriture continue de lignes de données supplémentaires.

```
import csv
from sense_hat import SenseHat
from datetime import datetime from
pathlib import Path
from time import sleep

sense = SenseHat()

dir_path = Path(_file_).parent.resolve()
data_file = dir_path/'data.csv'

with open(data_file, 'w') as f:
    writer = csv.writer(f)
    header = ("Date/time", "Temperature", "Humidity")
    writer.writerow(header)
    for i in range(10):
        row = (datetime.now(), sense.temperature, sense.humidity)
        writer.writerow(row)
        sleep(60)
```

Comment modifier le code ci-dessus pour enregistrer aussi les relevés de pression barométrique du

Sense HAT ? Votre programme peut ressembler à ceci :

```
import csv
from sense_hat import SenseHat
from datetime import datetime from
pathlib import Path
from time import sleep sense

= SenseHat()

dir_path = Path(_file_).parent.resolve()
data_file = dir_path/'data.csv'

with open(data_file, 'w') as f:
    writer = csv.writer(f)
    header = ("Date/time", "Temperature", "Humidity", "Pressure")
    writer.writerow(header)
    for i in range(10):
        row = (datetime.now(), sense.temperature, sense.humidity, sense.pressure)
        writer.writerow(row)
        sleep(60)
```

Vous pouvez préférer écrire la ligne d'en-tête en haut de votre fichier, puis ajouter chaque ligne séparément. Cette méthode est plus sûre si votre programme se termine prématurément (vous ne perdrez pas vos données) :

```
import csv
from sense_hat import SenseHat
from datetime import datetime from
pathlib import Path
from time import sleep

dir_path = Path(_file_).parent.resolve()
data_file = dir_path/'data.csv'

with open(data_file, 'w') as f:
    writer = csv.writer(f)
    header = ("Date/time", "Temperature", "Humidity")
    writer.writerow(header)

for i in range(10):
    row = (datetime.now(), sense.temperature, sense.humidity)
    with open(data_file, 'a') as f:
        writer = csv.writer(f)
        writer.writerow(row)
    sleep(60)
```

Notez que la première fois que vous écrivez dans un fichier, vous devez l'ouvrir avec `w` (mode écriture). Si vous souhaitez ajouter des données par la suite, vous devez utiliser `a` (mode ajout).

Vous pouvez utiliser deux fonctions séparées pour cela :

```
def create_csv(data_file):
    with open(data_file, 'w') as f: writer
        = csv.writer(f)
        header = ("Date/time", "Temperature", "Humidity")
        writer.writerow(header)

def add_csv_data(data_file, data): with
    open(data_file, 'a') as f:
        writer = csv.writer(f)
        writer.writerow(data)
```

Il est important d'enregistrer l'horodatage avec vos points de données, afin de savoir quand les mesures ont été effectuées, quel délai s'est écoulé entre chaque mesure et à quel moment des événements se sont produits. Vous pouvez aussi calculer la position de l'ISS a posteriori à l'aide d'ephem et d'un horodatage.

Enregistrement avec logzero

logzero simplifie l'enregistrement de remarques concernant ce qui se produit dans votre programme. Si vous accédez aux données de votre expérience et que vous découvrez que de nombreuses données sont manquantes sans aucune explication, vous ne pourrez pas déterminer ce qui s'est produit. C'est pourquoi il est important d'enregistrer autant d'informations que possible sur ce qui se produit dans votre programme. Enregistrez chaque itération de la boucle, chaque fois qu'une fonction importante est appelée et si votre programme contient des opérateurs conditionnels, chaque chemin suivi par le programme (`if` ou `else`).

Voici un exemple de l'utilisation de base de logzero pour garder une trace des itérations de la boucle :

```
from logzero import logger, logfile
from pathlib import Path
from time import sleep
from datetime import datetime

dir_path = Path(_file_).parent.resolve()

logfile(dir_path/"teamname.log")

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

Les deux principaux types d'enregistrements que vous pouvez utiliser sont `logger.info()` pour enregistrer des informations et `logger.error ()` lorsque vous rencontrez une erreur inattendue ou que vous traitez une exception. Il existe aussi `logger.warning()` et `logger.debug()`.

Par exemple, si vous disposez d'une fonction pour détecter la nuit ou la pénombre sur les photos, vous pouvez enregistrer aussi ces informations :

```
for i in range(10):
    if night_or_dark() == 'night':
        logger.info('night - wait 60 seconds')
        sleep(60)
    else:
        ...
```

Si vous voulez traiter une erreur et enregistrer le fait que vous avez procédé à ce traitement, vous pouvez utiliser `logger.error` :

```
try:
    do_something()
 except Exception as e:
    logger.error(f'{e._class_._name_}: {e})')
```

Par exemple, une division par zéro dans `do_something` génèrerait l'enregistrement suivant :

```
[E 190423 00:04:16 test:9] ZeroDivisionError: division by zero
```

Votre programme se poursuivrait sans planter, mais plutôt que de ne voir aucun enregistrement, vous verriez qu'une erreur s'est produite à ce moment-là.

Vous pouvez (et devez) utiliser la bibliothèque `csv` et la bibliothèque `logzero`.

Quota de stockage des données

Votre expérience peut produire un maximum de 3 Go de données. Vous devez donc calculer la quantité maximale d'espace de stockage occupée par les données enregistrées de votre expérience, photos comprises, et vous assurer que cette valeur n'excède pas 3 Go. Aucun fichier individuel ne doit dépasser 35 Mo.

Étape 5 Détermination de la position de l'ISS

À l'aide de la bibliothèque Python `ephem`, vous pouvez calculer la position d'objets spatiaux dans notre système solaire. Cela englobe le Soleil, la Lune, les planètes et les nombreux satellites terrestres, comme l'ISS. Vous pouvez ainsi déterminer la position actuelle de l'ISS au-dessus de la Terre, que vous pouvez ensuite utiliser pour établir si l'ISS passe au-dessus de la terre ferme ou de la mer, ou bien quel pays elle survole.

Pour des calculs précis, vous devez fournir `ephem` avec le jeu (TLE) le plus récent de données sur deux lignes pour l'ISS. TLE est un format de données utilisé pour transmettre des jeux de données orbitales qui décrivent l'orbite des satellites terrestres. Voici les dernières données TLE de l'ISS (http://www.celestrak.com/NORAD/elements/stations.txt) (ainsi que ces mêmes informations dans d'autres formats). Ces trois lignes doivent être collées dans votre code et considérées comme des arguments lorsque vous créez un objet `iss` dans votre programme.

```
from ephem import readtle, degree

name = "ISS (ZARYA)"
line1 = "1 25544U 98067A    20316,41516162  .00001589  00000+0  36499-4 0 9995"
line2 = "2 25544 51,6454 339,9628 0001882 94,8340 265,2864 15,49409479254842"

iss = readtle(name, line1, line2)

iss.compute()

print(f"{iss.sublat/degree} {iss.sublong/degree}")
```

Il existe peu de manières différentes d'exprimer la latitude et la longitude et il est important d'indiquer les unités correctes, en particulier quand on travaille avec des logiciels et des bibliothèques requérant des données dans un certain format.

Le code ci-dessus exprime la latitude et la longitude au format Degrés décimaux (DD), dans lequel les coordonnées sont indiquées en degrés (°). La latitude couvre 180° : 90° au nord et 90° au sud de l'équateur). La longitude couvre 360° : 180° à l'est et 180° à l'ouest du méridien origine (le point zéro de la longitude, défini sur un point à Greenwich, en Angleterre). Pour spécifier un lieu avec précision, chaque degré peut être exprimé sous forme d'un nombre décimal, par ex. (-28.277777, 71.5841666).

Le format degrés:minutes:secondes (DMS) constitue une autre approche dans laquelle chaque degré est divisé en 60 minutes (’) et chaque minute est divisée en 60 secondes (”). Pour une précision accrue, des fractions de seconde indiquées par un point décimal sont utilisées. La complication supplémentaire ici est qu'une valeur exprimée en degrés ne peut pas être négative. Une information supplémentaire doit être ajoutée pour chaque valeur — la référence de la latitude et la référence de la longitude. Cette information indique simplement si le point auquel les coordonnées se rapportent se trouve au nord ou au sud de l'équateur (pour la latitude) et à l'est ou à l'ouest du méridien (pour la longitude). L'exemple ci-dessus apparaîtrait donc sous la forme (28:16:40 S, 71:35:3 E).

Si vous devez utiliser le format DMS, il s'avère pratique de traiter la représentation de chaîne de `iss.sublat` et `iss.sublong` (sans division par `ephem.degree`) :

```
print(f"{iss.sublat} {iss.sublong}")
```

Vérification des coordonnées actuelles

Si vous souhaitez que votre expérience soit exécutée lorsque l'ISS survole un lieu spécifique sur Terre, vous pouvez utiliser les valeurs de latitude et de longitude pour déclencher une action. Souvenez-vous que l'orbite de l'ISS ne survole pas toute la Terre et qu'une grande partie de la surface de notre planète est de l'eau. C'est pourquoi les chances de survoler une ville ou un lieu précis durant les trois heures de votre expérience sont très faibles.

Pour déterminer en quoi cela peut être utile pour votre programme, modifiez le code ci-dessus pour qu'il imprime un message lorsque l'ISS survole l'hémisphère sud.

Votre programme ressemble alors à ceci :

```
from ephem import readtle, degree

name = "ISS (ZARYA)"
line1 = "1 25544U 98067A    20316,41516162  .00001589  00000+0  36499-4 0 9995"
line2 = "2 25544 51,6454 339,9628 0001882 94,8340 265,2864 15,49409479254842"

iss = readtle(name, line1, line2) iss.compute()
latitude = iss.sublat / degree

if latitude < 0:
  print("In Southern hemisphere") else:
  print("In Northern hemisphere")
```

Notez que lorsque votre code sera exécuté sur la station spatiale, les données de télémétrie les plus précises et à jour seront utilisées automatiquement ; aussi, vous n'avez pas besoin d'écrire des routines pour mettre à jour ces données dans le cadre de votre programme. Incluez simplement les lignes TLE les plus récentes lors de la soumission de votre code.

Étape 6 Enregistrement d'images à l'aide de la caméra

La première chose à faire, si ce n'est déjà fait, est de connecter votre module de caméra au

Raspberry Pi. Lorsque c'est fait, rallumez le Raspberry Pi et prenez quelques photos de test :

Le fragment de code ci-dessous montre comment prendre une photo avec les modules de caméra des Astro Pi à l'aide de la bibliothèque `picamera` et l'enregistrer dans le bon répertoire. La bibliothèque `picamera` est très puissante et dispose d'une super documentation (https://picamera.readthedocs.io/en/latest/).

```
from time import sleep
from picamera import PiCamera
from pathlib import Path import
os

dir_path = Path(_file_).parent.resolve()

camera = PiCamera()
camera.resolution = (1296,972)
camera.start_preview()
# Camera warm-up time
sleep(2)
camera.capture(f"{dir_path}/image.jpg")
```

Si vous utilisez la caméra à lumière visible sur Astro Pi Ed, votre programme doit supprimer toutes les images à la fin de votre expérience.

```
os.remove(dir_path/"image.jpg")
```

Si vous utilisez la caméra à infrarouges sur Astro Pi Izzy, vous obtiendrez des photos incroyables de la Terre vue de l'ISS. Même si votre programme traite ces images et n'utilise que les données extraites, nous vous recommandons de ne pas supprimer toutes les images (à moins que votre programme n'en génère tant que vous risquiez de manquer d'espace disque sur l'Astro Pi). Outre le fait qu'il s'agit d'un souvenir unique de votre mission, les images peuvent aussi vous aider à déboguer tout problème inattendu avec les résultats de votre expérience. Voici quelques exemples d'images prises avec la caméra IR sur Izzy (https://www.flickr.com/photos/raspberrypi). Si vous devez traiter les images (par ex. avec la bibliothèque OpenCV Python), vous devez tester votre code sur certaines de ces images.

Le reste des étapes concerne essentiellement les expériences sur le thème « La vie sur Terre ». Aucune image des expériences sur le thème « La vie dans l'espace » ne peut être enregistrée.

Données de localisation (« La vie sur Terre »)

Photographier la Terre depuis un hublot de l'ISS est normalement réservé aux astronautes. Nous vous recommandons d'enregistrer la position de la station spatiale pour toutes les photos que vous prenez. Vous pouvez le faire en enregistrant la latitude et la longitude dans un fichier CSV avec le nom du fichier correspondant à l'image.

Une meilleure méthode consiste à ajouter les données de position directement dans les champs EXIF du fichier de chaque image. Ces métadonnées sont « liées » au fichier de l'image et ne nécessitent pas l'accompagnement d'un fichier de données CSV.

Dans le fragment de code ci-dessous, une fonction intitulée `capture` est appelée pour prendre une image après avoir défini les données EXIF sur la latitude et la longitude actuelles. Les coordonnées dans les données EXIF sont stockées à l'aide d'une variante du format degrés:minutes:secondes (DMS), et vous pouvez voir comment la fonction `convert` récupère les données fournies par la bibliothèque `ephem` et les convertit dans un format compatible avec un stockage sous forme de données EXIF. L'utilisation de fonctions pour effectuer ces tâches permet de garder un programme propre.

```
import ephem
from picamera import PiCamera from
pathlib import Path

dir_path = Path(_file_).parent.resolve()

name = "ISS (ZARYA)"
line1 = "1 25544U 98067A    20316,41516162  .00001589  00000+0  36499-4 0 9995"
line2 = "2 25544 51,6454 339,9628 0001882 94,8340 265,2864 15,49409479254842"
iss = ephem.readtle(name, line1, line2)

cam = PiCamera()
cam.resolution = (1296,972) # Valid resolution for V1 camera
iss.compute()

def convert(angle):
    """
    Convert an ephem angle (degrees:minutes:seconds) to an
    EXIF-appropriate representation (rationals)
    e.g. '51:35:19.7' to '51/1,35/1,197/10'
    Return a tuple containing a boolean and the converted angle,
    with the boolean indicating if the angle is negative.
    """
    degrees, minutes, seconds = (float(field) for field in str(angle).split(":"))
    exif_angle = f'{abs(degrees):.0f}/1,{minutes:.0f}/1,{seconds*10:.0f}/10' return
    degrees < 0, exif_angle

def capture(camera, image):
    """Use `camera` to capture an `image` file with lat/long EXIF data."""
    iss.compute() # Get the lat/long values from ephem

    # convert the latitude and longitude to EXIF-appropriate representations
    south, exif_latitude = convert(iss.sublat)
    west, exif_longitude = convert(iss.sublong)

    # set the EXIF tags specifying the current location
    camera.exif_tags['GPS.GPSLatitude'] = exif_latitude
    camera.exif_tags['GPS.GPSLatitudeRef'] = "S" if south else "N"
    camera.exif_tags['GPS.GPSLongitude'] = exif_longitude
    camera.exif_tags['GPS.GPSLongitudeRef'] = "W" if west else "E"

    # capture the image
    camera.capture(image)

capture(cam, dir_path/"gps1.jpg")
```

Au lieu d'utiliser les données EXIF, il est possible de superposer des données de texte sur l'image visible, comme en filigrane. Toutefois, il existe toujours un risque que cela obscurcisse une zone utile de la photo et cela peut perturber le code qui détermine la luminosité des pixels de l'image. De plus, ces superpositions ne sont pas faciles à supprimer. Contrairement à la méthode EXIF, elles ne facilitent pas non plus le traitement des images sur la base de métadonnées, ni la recherche d'images sur la base de l'endroit où elles ont été prises. Aussi, nous vous recommandons de ne pas utiliser la méthode du filigrane pour enregistrer la latitude et la longitude mais de privilégier plutôt les données EXIF.

Traduction ESERO France, CNES

Numérotation des plans pour les fichiers

Parmi les projets intéressants à réaliser à l'aide d'une séquence d'images prises depuis l'ISS, vous pouvez créer un film en timelapse, comme celui présenté dans la première section de ce projet. C'est possible sur un Raspberry Pi à l'aide d'une seule commande, si les images sont enregistrées dans des fichiers nommés de manière pertinente et incluant un numéro séquentiel évident. La convention d'appellation de vos fichiers d'images est donc `image_001.jpg`, `image_002.jpg`, etc. Souvenez-vous que le nom de fichiers ne peuvent pas contenir d'espaces !

```
from time import sleep
from picamera import PiCamera from
pathlib import Path

dir_path = Path(_file_).parent.resolve()

camera = PiCamera()
camera.start_preview()
sleep(2)
for filename in camera.capture_continuous(f"{dir_path}/image_{counter:03d}.jpg"):
    print(f'Captured {filename}')
    sleep(300) # wait 5 minutes
```

Ensuite, dès que vous recevez vos images de l'ISS, vous pouvez utiliser la commande suivante pour créer un film en timelapse (vous devez d'abord installer le paquet `libav-tools`).

```
avconv -r 10 -i image%03d.jpg -r 10 -vcodec libx264 -crf 20 -g 15 timelapse.mp4
```

Il s'agit là d'une étape de traitement post-expérience. Vous ne devez pas utiliser tes trois heures d'expérience sur l'ISS pour essayer de réaliser un film en timelapse !

Photographie dans des conditions de faible luminosité et de nuit

La photographie de nuit à l'aide du module de caméra de l'Astro Pi est difficile. C'est essentiellement dû au fait qu'il y a très peu de chances que votre programme soit exécuté alors que l'ISS survole une ville lumineuse sans couverture nuageuse. La photosensibilité de la caméra est plutôt bonne mais elle doit être utilisée avec des paramètres logiciels optimaux pour une situation donnée, et il est difficile d'anticiper quels seront ces paramètres et de les intégrer dans votre programme. Adapter la caméra aux variations de lumière en temps réel est aussi complexe, en particulier lorsque la caméra se déplace par rapport à la source de lumière, ce qui est le cas des Astro Pi à bord de l'ISS.

Taille et nombre d'images

N'oubliez pas que votre expérience est limitée à la production de 3 Go de données. Assurez-vous d'avoir calculé la quantité d'espace maximale requise pour vos mesures, y compris les fichiers d'images enregistrés, et que la limite des 3 Go est respectée. Souvenez-vous que la taille d'un fichier d'image dépend de la résolution, mais aussi de la quantité de détails sur l'image : la photo d'un mur blanc prend moins de place que la photo d'un paysage.

Étape 7 Faire plusieurs choses à la fois

Dans les programmes simples, chaque ligne est exécutée dans l'ordre et les choses se produisent les unes après les autres. Le programme ne peut pas faire plusieurs choses à la fois — il doit attendre que la tâche en cours se termine avant de lancer la suivante. C'est ce que l'on appelle un processus à thread unique.

Threads multiples

Si vous devez faire plusieurs choses à la fois, vous pouvez utiliser un processus à threads multiples. Plusieurs bibliothèques Python permettent de coder ce type de processus multitâches. Toutefois, pour faire cela sur les Astro Pi, vous ne pouvez utiliser que la bibliothèque `threading`.

N'utilisez la bibliothèque `threading` que si cela s'avère absolument indispensable pour votre expérience. La gestion des threads peut être complexe et comme votre expérience sera exécutée dans le cadre d'une séquence de programmes, nous devons nous assurer que le précédent s'est terminé correctement avant de démarrer le suivant. Des threads indésirables peuvent devenir incontrôlables et monopoliser les ressources système, et doivent donc être évités. Si vous utilisez des threads dans votre code, vous devez vous assurer qu'ils sont tous soigneusement gérés et se ferment proprement à la fin de votre expérience. De plus, vous devez vous assurer que les commentaires dans votre code expliquent clairement ce qu'il en est.

Étape 8 Exécution de votre expérience pendant 3 heures

Votre expérience se verra attribuer 180 minutes de temps d'exécution sur l'ISS. Aussi, votre code doit être exécuté sur une période ne dépassant pas ces trois heures et fermer de lui-même toutes les activités (par exemple : éteindre la caméra, fermer les fichiers ouverts, effacer la matrice à LED). Au bout de trois heures, votre code sera terminé automatiquement par l'Astro Pi, mais cela peut entraîner la perte ou l'enregistrement incorrect de données ; aussi, vous ne devez pas compter sur cela pour arrêter votre programme.

Pour arrêter votre programme après un laps de temps donné, vous pouvez utiliser la bibliothèque `datetime` Python. Cette bibliothèque facilite le travail sur différentes périodes et les compare. Réaliser cette tâche sans la bibliothèque n'est pas toujours une partie de plaisir : il est facile de se tromper avec les mathématiques classiques. Par exemple, il est facile de déterminer l'écart de temps entre 10:30 et 10:50 (soustraire 30 de 50), mais un peu plus délicat entre 10:44 et 11:17 (ajouter (60 - 44) à 17). Les choses se compliquent encore si les deux heures correspondent à deux jours différents (par exemple, trouver l'écart en minutes entre 23:07 le lundi 30 mai et 11:43 le mardi 1er juin). La bibliothèque `datetime` simplifie notablement ce type d'opération en vous permettant de créer des objets `datetime` que vous pouvez simplement ajouter ou soustraire entre eux.

En enregistrant l'heure de début de votre expérience, vous pouvez vérifier de manière répétée si l'heure actuelle est supérieure à cette heure de début plus un certain nombre de minutes, de secondes ou d'heures. Cet écart s'appelle un `timedelta`.

```
import datetime
from time import sleep

# create a datetime variable to store the start time start_time
= datetime.datetime.now()
# create a datetime variable to store the current time #
(these will be almost the same at the start) now_time =
datetime.datetime.now()
# run a loop for 2 minutes
while (now_time < start_time + datetime.timedelta(minutes=2)):
    print("Doing stuff")
    sleep(1)
    # update the current time now_time
    = datetime.datetime.now()
```

Remarque : Lorsque vous décidez du temps d'exécution pour votre code, assurez-vous d'avoir bien pris en compte la durée de votre boucle pour accomplir un cycle. Si vous souhaitez profiter au maximum de votre créneau de trois heures (180 minutes) mais que chaque boucle de votre code dure 6 minutes, votre `timedelta` doit être de `180-6 = 174` minutes pour vous assurer que votre code se termine avant écoulement des trois heures.

Étape 9 Utilisation de l'affichage à LED - Expériences « La vie dans l'espace » uniquement

La matrice à LED est le seul affichage disponible pour l'ordinateur Astro Pi, qui n'est jamais connecté à un moniteur normal ou à un écran de TV à bord de l'ISS. L'équipage peut être amené à se demander si l'ordinateur Astro Pi n'a pas planté si rien ne s'affiche de temps à autre. L'équipage peut alors perdre du temps à le vérifier et/ou à appeler le contrôle au sol pour signaler un problème. Pour éviter cela, votre code doit mettre à jour la matrice à LED de manière à indiquer que votre expérience suit son cours. Si votre expérience est sur le thème « La vie sur Terre », vous ne devez pas utiliser la matrice à LED car l'Astro Pi sera masqué pour éviter que la lumière parasite ne gâche les images prises depuis le hublot de l'ISS.

La bibliothèque `sense_hat` dispose de fonctions pour écrire des messages sur la matrice à LED ou pour allumer des pixels individuels.

Il s'agit de fonctions de verrouillage. En d'autres termes, rien d'autre ne peut se produire tant que ces tâches sont en cours d'exécution. Donc si vous utilisez `show_message` pour afficher une très longue chaîne de texte, cela se fera au détriment du reste de la mission. C'est pourquoi vous devez limiter tout message d'introduction au démarrage de votre programme à moins de 15 secondes du début du programme.

Vous pouvez bien évidemment utiliser un thread pour effectuer d'autres tâches en arrière-plan pendant que des graphiques s'affichent sur la matrice à LED. Cependant, comme expliqué à la section « Faire plusieurs choses à la fois » ci-avant, vous devez éviter d'utiliser des threads sauf s'ils sont absolument essentiels pour votre expérience.

La matrice à LED peut produire des images couleur très lumineuses. Toutefois, souvenez-vous que l'ISS est un environnement de travail : vous devez éviter l'abus de lumières clignotantes ou scintillantes qui peuvent distraire les astronautes.

```
from sense_hat import SenseHat
from time import sleep
import random
sh = SenseHat()

# Define some colours - keep brightness low g
= [0,50,0]
o = [0,0,0]

# Define a simple image
img1 = [
g,g,g,g,g,g,g,g,
o,g,o,o,o,o,g,o,
o,o,g,o,o,g,o,o,
o,o,o,g,g,o,o,o,
o,o,o,g,g,o,o,o,
o,o,g,g,g,g,o,o,
o,g,g,g,g,g,g,o,
g,g,g,g,g,g,g,g,
]

# Define a function to update the LED matrix def
active_status():
    # a list with all possible rotation values
    orientation = [0,90,270,180]
    # pick one at random
    rot = random.choice(orientation)
    # set the rotation
    sh.set_rotation(rot)

# Load the image
sh.set_pixels(img1)
while True:
    # do stuff (in this case, nothing)
    sleep(2)
    # update the LED matrix
    active_status()
```

Vous devez mettre à jour l'écran au moins toutes les 15 secondes.
Si votre expérience présente une période de « veille » plus longue, vous pouvez diviser le délai d'attente :

```
sh.set_pixels(img1)
while True:
    # do stuff (in this case, nothing)
    sleep(15)
    # update the LED matrix
    active_status()
    # more doing nothing sleep(15)
    # update LEDs again
    active_status()
```

Notez qu'il est interdit de modifier le niveau de luminosité des LED. N'utilisez pas `sense.low_light`, `sense.gamma`, `sense.reset_gamma`, ou `pisense.hat.screen.gamma` dans votre programme.

Étape10 Mise en réseau et processus système

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 point de connexion, d'accéder à Internet ou d'établir une connexion réseau de quelque nature que ce soit. Cela englobe les connexions au réseau local de l'Astro Pi lui-même. Dans le cadre du test de votre programme, vous devez désactiver la connectivité sans fil et débrancher le câble Ethernet de votre Raspberry Pi pour vous assurer que votre expérience s'exécute correctement sans connexion à Internet.

De plus, votre programme n'est pas autorisé à exécuter un autre programme ou une quelconque commande que vous saisiriez normalement dans la fenêtre de terminal du Raspberry Pi.

Température de la CPU

Il est courant d'utiliser un sous-processus pour mesurer la température de la CPU. Nous recommandons toutefois d'utiliser l'interface de température de la CPU (https://gpiozero.readthedocs.io/en/stable/api_other.html#cputemperature) fournie par GPIO Zero :

```
from gpiozero import CPUTemperature

cpu = CPUTemperature()

print(cpu.temperature)
```

Étape 11 Erreurs courantes

Mission Space Lab existe depuis quelques années maintenant et a connu des expériences incroyables. Cependant, des projets fantastiques sont rejetés chaque année et n'ont pas la chance d'être exécutés sur l'ISS en raison de problèmes avec le code final.

Il y a aussi des expériences qui fonctionnent mais ne produisent pas de données pour leurs équipes en raison d'erreurs évitables.

Voici quelques problèmes courants que vous devez éviter.

Ne vous basez pas sur une intervention humaine

Votre programme ne doit pas se baser sur une intervention humaine à l'aide du joystick et des boutons. L'équipage n'aura pas le temps d'opérer les Astro Pi manuellement, votre expérience ne peut donc pas dépendre d'interventions humaines. Par exemple, si une expérience nécessite qu'un astronaute appuie sur un bouton pour commencer, cette action n'aura jamais lieu et l'expérience ne s'exécutera pas pendant trois heures. C'est aussi la raison pour laquelle les expériences sur l'équipage, comme des tests de rapidité de réaction ou de mémoire, ne sont pas compatibles avec Mission Space Lab.

Enregistrez les données

Assurez-vous que les données de votre expérience sont écrites dans un fichier dès qu'elles sont enregistrées. Évitez d'enregistrer les données dans une liste interne ou un dictionnaire au fur et à mesure et de les écrire dans un fichier à la fin de l'expérience parce que si votre expérience s'interrompt brutalement en raison d'une erreur ou si elle excède la limite de 3 heures, vous n'obtiendrez aucune donnée.

N'utilisez pas de chemins de fichiers absolus

Assurez-vous de ne pas utiliser des chemins spécifiques pour vos fichiers de données. Utilisez la variable `_file` comme décrit dans « Enregistrement des données de votre expérience ».

Vérifiez les erreurs de type « diviser par zéro »

Une cause courante de l'échec de programmes est lorsqu'une fonction mathématique essaie de diviser une valeur par zéro. Cela peut se produire si vous relevez une valeur sur un capteur et que vous l'utilisez dans un calcul. Assurez-vous toujours que votre programme peut fonctionner si l'une des valeurs retournées par un capteur (en particulier l'accéléromètre) est nulle.

Assurez-vous que votre code peut traiter les erreurs (exceptions)

Une exception est un événement qui se produit pendant l'exécution d'un programme et interrompt le déroulement normal des instructions du programme. Par exemple, si votre programme prend deux nombres et les divise entre eux, cela fonctionne dans de nombreux cas :

```
>>> a = 1
>>> b = 2
>>> c = a / b
>>> print(c)
0.5
```

Mais si le deuxième nombre est nul, l'opération de division échoue :

```
>>> a = 1
>>> b = 0
>>> c = a / b
Traceback (most recent call last): File
  "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

L'un des moyens de gérer ce problème potentiel est de l'identifier de manière anticipée :

```
if b != 0:
    c = a / b
else:
    print("b cannot be zero")
```

Un autre moyen consiste à essayer de terminer l'opération et à traiter l'exception si elle se produit :

```
try:
    c = a / b
except ZeroDivisionError: print("b
    cannot be zero")
```

Un bon exemple d'exception qui peut se produire lorsque vous utilisez le Sense HAT se présente comme suit : votre programme utilise une variable comme valeur de couleur de pixel, mais la valeur affectée à la variable n'est pas comprise dans la plage autorisée (0 à 255).

```
>>> r = a + b
>>> sense.set_pixel(x, y, r, g, b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python3.5/dist-packages/sense_hat/sense_hat.py", line 399, in set_pixel
    raise ValueError('Pixel elements must be between 0 and 255')
ValueError: Pixel elements must be between 0 and 255
```

Il est important d'anticiper tous les endroits dans votre programme où une variable peut prendre une valeur susceptible de poser problème. Par exemple, si vous utilisez la mesure de l'humidité pour déterminer à quel point les pixels sont rouges, assurez-vous que cette valeur ne peut pas sortir de la plage comprise entre 0 et 255 pas seulement durant le test, mais aussi dans toutes les situations possibles :

```
red = int(max(0, min(sh.humidity / 100 * 255, 255)))
```

Cette ligne de code signifie que si la mesure de l'humidité est égale à 0 ou inférieure, la valeur de `red` sera de 0 et que si cette mesure est égale à 100 ou supérieure, la valeur de `red` sera de 255. Pour les mesures entre 0 et 100, la valeur de `red` sera proportionnelle. De plus, la valeur de `red` sera toujours un nombre entier.

Exceptions multiples

Le moyen le plus simple de traiter les exceptions est de déterminer toutes les exceptions et de les traiter de la même manière :

```
try:
    do_something()
except:
    print("An error occurred")
```

Toutefois, cela ne vous apporte aucune information sur les exceptions. Vous devez plutôt tenir compte des

types d'exceptions qui peuvent survenir. Il est possible de traiter différentes exceptions de différentes manières :

```
try:
    divide(a, b)
except ZeroDivisionError: print("b
    cannot be zero")
except TypeError:
    print("a and b must be numbers")
```

En essayant d'éviter les exceptions et en traitant celles qui surviennent de manière optimale, vous pouvez éviter les erreurs susceptibles d'empêcher l'exécution de votre programme jusqu'au bout, et donc toute déception. Imaginez recevoir les journaux d'une expérience défaillante et constater la survenue d'une exception qui aurait pu être traitée, ou bien d'un message d'erreur qui ne révèle rien sur ce qui a mal tourné.

Étape 12 Exemple travaillé

Vous pouvez désormais combiner tous les éléments décrits dans ce document pour vous aider à coder votre expérience — l'exemple ci-dessous peut vous servir de modèle.

Imaginez la situation : l'équipe de CoderDojo Tatooine souhaite déterminer si l'environnement à bord de l'ISS est affecté par la surface de la Terre survolée. Est-ce que l'ISS est plus chaude lorsqu'elle survole un désert, ou bien plus humide lorsqu'elle survole un océan ?

- Son code prend des mesures régulières de la température et de l'humidité toutes les 30 secondes et les enregistre dans un fichier CSV.
- Il calcule aussi la latitude et la longitude de l'ISS avec la bibliothèque `ephem` et enregistre ces informations dans le fichier de données.
- Pour voir si la couverture nuageuse peut aussi être un facteur, il prend une photo à l'aide de la caméra IR sur l'Astro Pi Izzy, qui est pointé vers la Terre à travers un hublot.
- Les données de latitude et de longitude sont écrites dans les balises EXIF des images, qui ont des noms de fichiers numérotés séquentiellement. Elles sont aussi enregistrées dans le fichier CSV.
- La matrice à LED n'est pas utilisée car il s'agit d'une expérience sur le thème « La vie sur Terre ».
- Toute erreur inattendue est traitée et les détails sont enregistrés.

```
from logzero import logger, logfile
from sense_hat import SenseHat from
ephem import readtle, degree from
picamera import PiCamera
from datetime import datetime, timedelta from
time import sleep
import random
from pathlib import Path import
csv

dir_path = Path(_file_).parent.resolve()

# Set a logfile name
logfile(dir_path/"teamname.log")

# Latest TLE data for ISS location name
= "ISS (ZARYA)"
line1 = "1 25544U 98067A    20316,41516162  .00001589  00000+0  36499-4 0 9995"
line2 = "2 25544 51,6454 339,9628 0001882 94,8340 265,2864 15,49409479254842"
iss = readtle(name, line1, line2)

# Set up Sense Hat
sh = SenseHat()

# Set up camera
cam = PiCamera()
cam.resolution = (1296, 972)

def create_csv_file(data_file):
    """Create a new CSV file and add the header row""" with
    open(data_file, 'w') as f:
        writer = csv.writer(f)
        header = ("Date/time", "Temperature", "Humidity")
        writer.writerow(header)

def add_csv_data(data_file, data):
    """Add a row of data to the data_file CSV""" with
    open(data_file, 'a') as f:
        writer = csv.writer(f)
        writer.writerow(data)

def get_latlon():
    """Return the current latitude and longitude, in degrees"""
    iss.compute() # Get the lat/long values from ephem
    return (iss.sublat / degree, iss.sublong / degree)

def convert(angle):
    """
    Convert an ephem angle (degrees, minutes, seconds) to an
    EXIF-appropriate representation (rationals)
    e.g. '51:35:19.7' to '51/1,35/1,197/10'
    Return a tuple containing a boolean and the converted angle, with
    the boolean indicating if the angle is negative.
    """
    degrees, minutes, seconds = (float(field) for field in str(angle).split(":"))
    exif_angle = f'{abs(degrees):.0f}/1,{minutes:.0f}/1,{seconds*10:.0f}/10' return
    degrees < 0, exif_angle

def capture(camera, image):
    """Use `camera` to capture an `image` file with lat/long EXIF data."""
    iss.compute() # Get the lat/long values from ephem

    # convert the latitude and longitude to EXIF-appropriate representations
    south, exif_latitude = convert(iss.sublat)
    west, exif_longitude = convert(iss.sublong)

    # set the EXIF tags specifying the current location
    camera.exif_tags['GPS.GPSLatitude'] = exif_latitude
    camera.exif_tags['GPS.GPSLatitudeRef'] = "S" if south else "N"
    camera.exif_tags['GPS.GPSLongitude'] = exif_longitude
    camera.exif_tags['GPS.GPSLongitudeRef'] = "W" if west else "E"
```

```
    # capture the image
    camera.capture(image)


# initialise the CSV file
data_file = dir_path/"data.csv"
create_csv_file(data_file)
# initialise the photo counter
photo_counter = 1
# record the start and current time
start_time = datetime.now() now_time
= datetime.now()
# run a loop for (almost) three hours
while (now_time < start_time + timedelta(minutes=178)): try:
        humidity = round(sh.humidity, 4)
        temperature = round(sh.temperature, 4)
        # get latitude and longitude latitude,
        longitude = get_latlon()
        # Save the data to the file
        data = (
            datetime.now(),
            photo_counter,
            humidity,
            temperature,
            latitude,
            longitude
        )
        add_csv_data(data_file, data) #
        capture image
        image_file = f"{dir_path}/photo_{photo_counter:03d}.jpg"
        capture(cam, image_file)
        logger.info(f"iteration {photo_counter}")
        photo_counter += 1
        sleep(30)
        # update the current time
        now_time = datetime.now()
    except Exception as e:
        logger.error('{}: {})'.format(e._class_._name_, e))
```

Fragment de code du fichier `data.csv` produit :

```
2019-12-03 08:43:28,1,54.9445,31.2797,0.2812739610671997,-0,7029094696044922
2019-12-03 08:43:59,2,55.3742,31.2257,0.2812739610671997,-0,7029094696044922
2019-12-03 08:44:29,3,55.6883,31.2797,0.2812739610671997,-0,7029094696044922
2019-12-03 08:45:00,4,55.3561,31.2977,0.2812739610671997,-0,7029094696044922
```

Étape 13 Testez votre programme

C'est la dernière et plus importante partie de la phase 2.

Avant de soumettre votre programme, il est vital de le tester avec un Astro Pi exécutant la version Flight de l'OS. Il s'agit d'une version spéciale de l'OS Raspberry Pi qui ressemble au système d'exploitation installé sur les unités Astro Pi à bord de l'ISS. Elle ne comprend aucune application X-Windows ou GUI et est « à ligne de commande uniquement ». Vous devez donc absolument vérifier qu'aucune des différences entre l'OS Flight et la version de l'OS Raspberry Pi sur laquelle vous avez développé votre code n'entraînera l'échec de votre expérience.

Il y a aussi quelques paramètres dans la version Flight de l'OS qui limitent les performances du Pi, afin de se rapprocher au plus près des capacités des Astro Pi à bord de l'ISS, qui sont d'anciens modèles Raspberry Pi B+. Cela signifie que votre code sera probablement exécuté plus lentement, en particulier si vous traitez des images ou que vous identifiez des endroits sur la base de la latitude et de la longitude.

Enfin, il est aussi important pour vous de tenir compte des erreurs susceptibles de se produire pendant l'exécution du programme sur l'OS Flight des Astro Pi embarqués, comme des erreurs de chemin de fichier ou l'écrasement de fichiers. Des centaines d'équipes soumettent des programmes chaque année dans le cadre de ce concours et malheureusement, nous ne sommes pas en mesure de vérifier les problèmes et de déboguer les erreurs de code complexes : si votre programme ne s'exécute pas sans erreurs lorsque nous le testons sur l'OS Flight, votre équipe ne pourra pas accéder à la phase 3. Alors pour vous assurer que votre projet bénéficie des meilleures chances de succès, testez soigneusement votre programme, déboguez toutes les erreurs et vérifiez sa conformité aux exigences en matière de codage.

Testez votre code sur l'OS Flight

Vous devez tester que votre code s'exécutera sur les unités Astro Pi, avec les mêmes paquets installés, les mêmes versions et une configuration identique. Nous testerons votre code sur l'OS Flight et s'il ne s'exécute pas sans erreurs, vous serez disqualifié. Il est donc important que votre test soit aussi approfondi que possible. C'est pourquoi nous fournissons un OS Flight, afin que vous puissiez simuler la même configuration qu'à bord de l'ISS.

Vous pouvez utiliser la deuxième carte SD, sur laquelle une OS Flight de démonstration est installée, pour effectuer des tests. Voici une vidéo détaillant comment transférer votre programme (https://esero.fr/tutoriels-en-ligne/decouverte-du-kit-astro-pi/) de la version Desktop de l'OS vers la version Flight.

Contrôle final

Pour que votre programme s'exécute en toute sécurité et correctement sur l'ISS, certaines règles doivent être suivies. Si vous avez travaillé en suivant ce guide, votre programme doit déjà être correct. Cependant, avant de soumettre votre programme, vous devez vérifier les points suivants :

- Votre expérience ne se base pas sur une interaction avec un astronaute
- Votre programme est écrit en Python 3

Votre programme ne se base pas sur des bibliothèques autres que celles répertoriées dans ce guide

Votre programme n'utilise pas la mise en réseau de ne démarre aucun processus système

Vous avez documenté votre programme et il est facile à comprendre

Vos données sont enregistrées dans des fichiers comme décrit dans ce guide et n'utilisent pas plus de 3 Go d'espace de stockage. Aucun fichier individuel ne doit dépasser 35 Mo

Si vous avez choisi le thème « La vie dans l'espace », votre programme n'enregistre aucune photo ou vidéo

Si vous avez choisi le thème « La vie dans l'espace », votre programme affiche régulièrement des messages ou des images sur la matrice à LED pour indiquer qu'une expérience est en cours

Si vous avez choisi le thème « La vie sur Terre », votre programme n'utilise pas la matrice à LED

Votre programme s'exécute sans erreurs sur l'OS Flight et ne génère aucune exception inattendue

L'exécution de votre programme s'interrompt au bout de 3 heures

Votre programme ne contient pas de langage grossier ni d'incivilités

Exécutez votre code

Connectez le Sense HAT et le module de caméra (si nécessaire).

Démarrez votre Raspberry Pi exécutant l'OS Flight.

Copiez votre code et tout fichier requis sur le Raspberry Pi.

Le point d'entrée principal de votre expérience doit être nommé `main.py`. Exécutez-le directement avec la commande suivante exclusivement :

```
python3 main.py
```

Votre code doit s'exécuter pendant trois heures, puis s'arrêter.

Une fois l'exécution terminée, analysez les fichiers de sortie créés par votre projet. Attendez-vous des fichiers d'image de la caméra ? Des journaux de données ? Autre chose ?

Si vous voyez des erreurs ou si votre expérience ne produit pas ce que vous attendiez d'elle, vous devez résoudre ces questions avant de soumettre votre code, afin d'avoir une chance d'atteindre le tour d'évaluation final.

Remarque : pendant les tests, il est recommandé de désactiver la connexion Internet du Raspberry Pi pour vous assurer que votre expérience n'utilise pas un accès à Internet.

Traduction ESERO France, CNES