```python
from exif import Image
from datetime import datetime
from pathlib import Path
from picamzero import Camera
from datetime import timedelta
from time import sleep  # Import the sleep function
import cv2
import math

#import datetime
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() # Update the current time

#define the base folder of the program
base_folder =Path(__file__).parent.resolve()
# define the output file as result.txt
data_file = base_folder / "result.txt"

#Directory path where the images are located
image_directory = base_folder

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

def get_time_difference(image1, image2):
    time_1 = get_time(image1)
    time_2 = get_time(image2)
    time_difference = time_2 - time_1
    return time_difference.seconds

def convert_to_cv(image1, image2):
    image1_cv = cv2.imread(str(image1), 0)
    image2_cv = cv2.imread(str(image2), 0)
    return image1_cv, image2_cv

def calculate_features(image1, image2, feature_number):
    orb = cv2.ORB_create(nfeatures = feature_number)
    keypoints_1, descriptors_1 = orb.detectAndCompute(image1_cv, None)
    keypoints_2, descriptors_2 = orb.detectAndCompute(image2_cv, None)
    return keypoints_1, keypoints_2, descriptors_1, descriptors_2

```

```
48 def calculate_matches(descriptors_1, descriptors_2):
49     brute_force = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
50     matches = brute_force.match(descriptors_1, descriptors_2)
51     matches = sorted(matches, key=lambda x: x.distance)
52     return matches
53 
54 #Create an instance of the PiCamera class
55 cam = Camera() #the camera is opened at the beginning of the program and remains open until all images are captured and the camera preview is stopped.
56 try:
57     # cam.start_preview()
58 # Camera warm-up time
59     sleep(2) #sleep two seconds before capturing an image, because this gives the camera’s sensor time to sense the light levels.
60 
61     # Define the maximum duration for capturing images (8 minutes)
62     max_capture_duration = timedelta(minutes=8)
63     start_time = datetime.now()
64 
65 # Define the maximum number of images to capture
66     max_images = 30
67     num_images = 0
68 # Capture images until the maximum duration is reached or the maximum number of images is captured
69     while num_images < max_images and datetime.now() - start_time < max_capture_duration:
70         cam.capture_image(str(base_folder / f"image{num_images}.jpg"))
71         num_images += 1
72         sleep(2)
73 
74 finally:
75     del cam
76 
77 def display_matches(image1_cv, keypoints_1, image2_cv, keypoints_2, matches):
78     match_img = cv2.drawMatches(image1_cv, keypoints_1, image2_cv, keypoints_2, matches[:100], None)
79     resize = cv2.resize(match_img, (1600,600), interpolation = cv2.INTER_AREA)
80     cv2.imshow('matches', resize)
81     cv2.waitKey(0)
82     cv2.destroyWindow('matches')
83 
84 def find_matching_coordinates(keypoints_1, keypoints_2, matches):
85     coordinates_1 = []
86     coordinates_2 = []
87     for match in matches:
88         image1_idx = match.queryIdx
89         image2_idx = match.trainIdx
90         (x1,y1) = keypoints_1[image1_idx].pt
91         (x2,y2) = keypoints_2[image2_idx].pt
92         coordinates_1.append((x1,y1))
93         coordinates_2.append((x2,y2))
```

```
    return coordinates_1, coordinates_2

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

def calculate_speed_in_kmps(feature_distance, GSD, time_difference):
    distance = feature_distance * GSD / 100000
    speed = distance / time_difference
    return speed

################## Initialize an empty list to store speeds###################
speeds = []

##################define the pair of consecutive images  ######################

for i in range(num_images - 1):
    image1 = image_directory / f"image{i}.jpg"
    image2 = image_directory / f"image{i + 1}.jpg" #Calculate time differences for every pair of consecutive images

    time_difference = get_time_difference(image1, image2)   # Get time difference between images
    image1_cv, image2_cv = convert_to_cv(image1, image2)    # Create OpenCV image objects
    keypoints_1, keypoints_2, descriptors_1, descriptors_2 = calculate_features(image1_cv, image2_cv, 1000)  # Get keypoints and descriptors
    try:

        matches = calculate_matches(descriptors_1, descriptors_2)  # Match descriptors  # Display matches
        coordinates_1, coordinates_2 = find_matching_coordinates(keypoints_1, keypoints_2, matches)  #print(coordinates_1[0],
coordinates_2[0])
        average_feature_distance = calculate_mean_distance(coordinates_1, coordinates_2)
        speed = calculate_speed_in_kmps(average_feature_distance, 12648, time_difference) #calculate Average Linear Speed of ISS in kmps
        estimate_kmps = speed  # Replace with your estimate

        if 5 < estimate_kmps < 10:  #Reject speeds that are under 5 or exceed/equal 10  for more accurate values
            speeds.append(estimate_kmps)  # Append the speed to the list
    except (cv2.error, ZeroDivisionError): pass

############################## Calculate the mean of all the speeds ###################################
if len(speeds) > 0:
    mean_speed = sum(speeds) / len(speeds)
else:
    print("speeds is empty")
```

```
    mean_speed = 0

# Format the mean_speed to have a precision of 5 significant figures # That is the Final Average linear Speed of ISS
mean_speed_formatted = "{:.4f}".format(mean_speed)

##############Write the average (mean) linear speed of the ISS to the output file (result.txt) #############
with open(data_file, "w") as f:
    f.write(mean_speed_formatted)

file = open(data_file)
 ###################### close all files and camera ##############################
file.close()


```