ENPM605: Python Applications for Robotics

RWA #2

v1.1

Lecturer: Z. Kootbally

School: University of Maryland

Semester/Year: Spring/2025

MARYLAND APPLIED GRADUATE ENGINEERING

Table of Contents

Changelog

- ***v1.1***: Cover page logo updated.
- ***v1.0***: Original version.

Conventions

bad practice	
best practice	
code syntax	
example	
exercise	
file	
folder	
guideline	
note	
question	
task	
terminology	
warning	
web link	link
package	

ROS node	n /node
ROS topic	t /topic
ROS message	m /message
ROS parameter	p /parameter
ROS frame	f /frame

Guidelines

This assignment must be completed ***individually***. Ensure compliance with all specified guidelines, as failure to follow any part of these guidelines will lead to a grade of ***zero*** for the assignment.

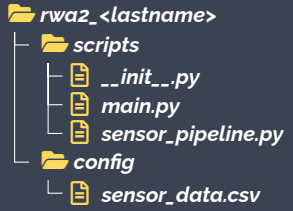
- Do not reuse a package obtained from peers.
- Keep your work confidential and refrain from sharing it with peers.
  - If you use github, you have to make your repository private.
- While discussing assignment challenges is encouraged, refrain from exchanging solutions with peers.
- Do not use code generated by AI tools.
  - The potential risks outweigh the benefits.

Objectives & Skills Practiced

- ***File I/O and Data Parsing***
  - Reading data from a CSV file using Python's `csv` module.
  - Converting each row into a dictionary with appropriate data types.
- ***Functional Programming Concepts***
  - Writing pure functions.
  - Creating data processing pipelines using higher-order functions.
  - Using lambda expressions for inline operations.
  - Applying partial function application to create specialized functions.
- ***Code Organization and Best Practices***
  - Modular function design.
  - Clear code documentation and comments.

Package

Each student uses one package with the following structure:



- ***sensor_pipeline.py***: Contains all the core functions required for this assignment.
- ***main.py***: The entry point that imports and calls functions from ***sensor_pipeline.py***
- ***sensor_data.csv***: Sample dataset containing 100 sensor readings, used by the functions in ***sensor_pipeline.py***

To get started, get this package from GitHub. Make sure to replace ***<lastname>*** with your actual last name.

Deliverable

Deliverable

Compress **rwa2_<lastname>** into ***a single file*** named **rwa2_<lastname>.<extension>**, where **<extension>** can be ***zip***, ***tar***, ***tar.gz***, etc., and **<last name>** is the student's last name. Submit it on Canvas by the due date.

Late submissions will incur a penalty according to the guidelines specified in the syllabus, with exceptions for any valid reason.

- Valid reasons include a doctor's note, proof of travel, or a note from a professor/MAGE.

Students with special circumstances may submit their assignments late without incurring any penalties. However, it is required that these students inform me in advance of their intention to submit their work past the deadline.

Tasks

Slides 9–16 outline the different tasks that must be implemented for this assignment.

Module Imports

- ***Task***: Import all necessary modules.
- ***Details***:
  - Use the `csv` module for handling CSV files.
  - Import functions from `functools` (e.g., `reduce()` and `partial()`) for later tasks.

Reading and Parsing CSV Data

- ***Task***: Create a function to read sensor data from a CSV file.
- ***Details***:
  - ***Function Name***: `read_sensor_data(filename)`
  - ***Objective***: Open the CSV file and read its contents using the `csv` module.
  - ***Requirement***: For each row, build a dictionary with exactly three keys:
    - `'sensor_id'` (convert its value to an integer)
    - `'distance'` (convert its value to a float)
    - `'timestamp'` (keep its value as a string)
  - ***Error Handling***: Handle a `FileNotFoundError` if the file is missing. Catch other exceptions that might occur during file reading.
  - ***Output***: Return a list of dictionaries containing sensor data.

Unit Conversion Function

- ***Task***: Write a pure function to convert distance measurements.
- ***Details***:
  - ***Function Name***: `inches_to_cm(inches)`
  - ***Objective***: Convert a given distance in inches to centimeters.

Processing Sensor Readings

- ***Task***: Process each sensor reading to apply the unit conversion.
- ***Details***:
  - ***Function Name***: `process_reading(reading)`
  - ***Objective***:
    - Take an individual sensor reading (which is a dictionary with exactly three keys).
    - Convert the `'distance'` from inches to centimeters.
    - ***Consideration***: Do not modify the original dictionary. Instead, create a copy of the original dictionary and update the value of `'distance'` with the newly converted value.
  - ***Implementations***:
    - Copy the reading to avoid modifying the original.
    - Perform the conversion.
    - Return the updated reading.

Filtering Sensor Readings Based on a Threshold

- ***Task***: Filter sensor readings that meet a safety threshold.
- ***Details***:
  - ***Function Name***: `threshold_filter(threshold, reading)`
  - ***Objective***: Verify whether the reading's distance (in centimeters) meets or exceeds a specified threshold (e.g., 20 cm).
  - ***Specialization***: Use `partial()` (from `functools`) to create a specialized filter (for example, `filter_above_20`) that fixes the threshold at 20 cm.
  - ***Implementations***:
    - For each reading, evaluate whether the `'distance'` value is greater than or equal to the threshold.
    - If the condition is met, append it to a list.
    - After processing all readings, return the list containing the valid distance values.

Computing the Average Distance

- ***Task***: Calculate the average distance from the filtered sensor readings.
- ***Details***:
  - ***Function Name***: `average_distance(readings)`
  - ***Objective***:
    - Use the `reduce()` function to sum all the distances (in centimeters).
    - Divide the total by the number of readings.
    - Handle cases where there are no valid readings by returning `None` or an appropriate message.

Building the Sensor Data Processing Pipeline

- ***Task***: Integrate all functions into a data processing pipeline.
- ***Details***:
  - ***Function Name***: `process_sensor_pipeline(filename)`
  - ***Objective***:
    - Read the sensor data from the CSV.
    - Process each sensor reading by converting the distance.
    - Filter the processed readings with the specialized threshold filter.
    - Print out each valid sensor reading. The output should be printed in a ***clear*** and ***formatted*** manner.
    - Compute and print the average distance of the valid readings. The output should be printed in a ***clear*** and ***formatted*** manner.
    - Include error handling if no sensor data is available.

Program Entry Point

- ***Task***: Set up the main entry point of the program in ***main.py***
- ***Details***:
  - Include the `if __name__ == '__main__':` block.
  - Call the pipeline function (e.g., `process_sensor_pipeline(<path to sensor_data.csv>)`).

Grading Rubric (1/2) 20 pts

- ***Module Imports and Setup (2 pts)***
  - ***2 pts***: Correctly import the necessary modules (e.g., `csv` and functions from `functools`).
- ***CSV Data Reading Function (4 pts)***
  - ***1 pt***: Open and read the CSV file using the correct file mode and newline settings.
  - ***1 pt***: Use the `csv` module to parse the CSV data.
  - ***1 pt***: Construct a list of dictionaries with exactly three keys: `'sensor_id'`, `'distance'`, and `'timestamp'`, with proper type conversions.
  - ***1 pt***: Implement error handling for missing files (`FileNotFoundError`) and other exceptions.
- ***Unit Conversion Function (2 pts)***
  - ***2 pts***: Correctly implement `inches_to_cm(inches)` by converting inches to centimeters (multiplying by 2.54).
- ***Processing Sensor Reading (3 pts)***
  - ***1 pt***: Implement `process_reading(reading)` to create a copy of a sensor reading without mutating the original.
  - ***1 pt***: Apply the unit conversion correctly to update the reading.
  - ***1 pt***: Ensure that the resulting dictionary maintains exactly three keys.

Grading Rubric (2/2) 20 pts

- ***Filtering Function and Partial Application (3 pts)***
  - ***1 pt***: Implement `threshold_filter(threshold, reading)` to compare the converted distance against the threshold.
  - ***1 pt***: Use `partial()` (from `functools`) to create a specialized filter (e.g., `filter_above_20`) with a fixed threshold (20 cm).
  - ***1 pt***: Apply the filtering correctly to the processed sensor data.
- ***Average Distance Computation (2 pts)***
  - ***2 pts***: Correctly implement `average_distance(readings)` using `reduce()` to compute the sum and then calculate the average, including handling the case with no valid readings.
- ***Sensor Data Pipeline and Program Entry (4 pts)***
  - ***2 pts***: Integrate all components in `process_sensor_pipeline(filename)` (reading, processing, filtering, and averaging) in a logical flow.
  - ***1 pt***: Provide clear, formatted output of valid sensor readings and the computed average.
  - ***1 pt***: Use the main guard (`if __name__ == '__main__':`) to ensure proper program entry.

Deductions

- Late submission: ***As per syllabus guidelines***.
- Use of AI-generated code: Assignment grade of ***zero***.
- Code sharing/plagiarism: Assignment grade of ***zero***.
- Non-private GitHub repository: Assignment grade of ***zero***.
- Functions lack adequate documentation (missing docstrings) or the program does not include sufficient inline comments: ***-2 pts***