Fundamentals of Programming for AI with Python

AAE4011 – Artificial Intelligence for Unmanned Autonomous Systems (UAS)

Dr Weisong Wen
Assistant Professor, Director of PolyU TAS Lab
Department of Aeronautical and Aviation Engineering
The Hong Kong Polytechnic University
Week 3, S2, 2024/2025

Our Teaching Assistants

Zhang Ziqi

Yang Qian

Wang Xin

Qiu Shaoting

Hu Runzhi

- Role/features of TAs in this course
  - Helper in lab session
  - Expert in AI and coding with Python
  - Experts in UAS, such as drones

- Let's get to know with each other
  - Short introduction about yourself (if we have enough time)? ☺
  - Who is your Final Year Project supervisor and what is your topic? ☺
  - Why you select this course? ☺

Ground Rules

- ✓ For students:

Open mind; speak English; participate activities assigned; ask questions

- ✓ For teachers:

Arrive on time; reply emails on time; answer questions related to the subject

- ✓ Be curious, Be inspired, Be motivated, Study further by yourself.

Assessment and Basic Requirement

- Assessment:
  - Homework Assignment (Strictly no late submission) (20%)
  - Mid-Term Quiz/Test (15%, close book)
  - Group Project (Case study, several members in a group) (15%)
  - Final Exam (50%) (Open book)
- Basic requirement:
  - Mathematics on matrix and its calculation
  - Extra time for finish the coding homework based on Python
  - Assurance on the attendance
  - Basic coding skills with Python (expect to learn yourself for extra), one week lecture for basics of Python

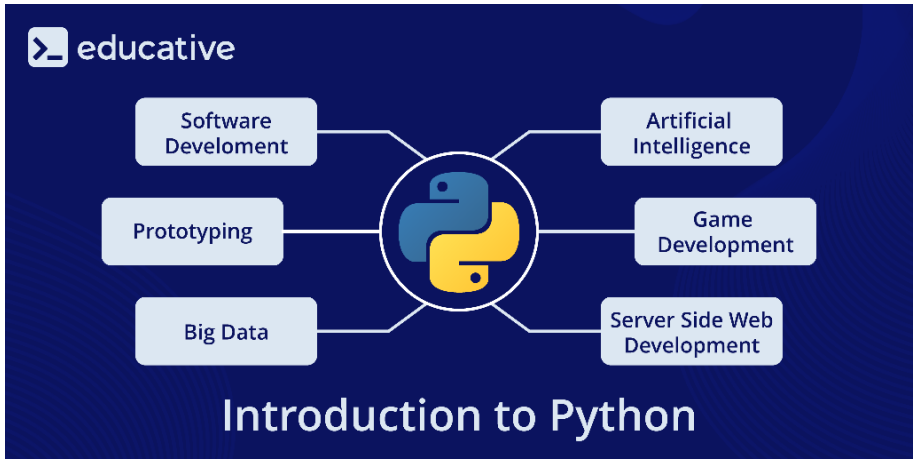
Outline for today

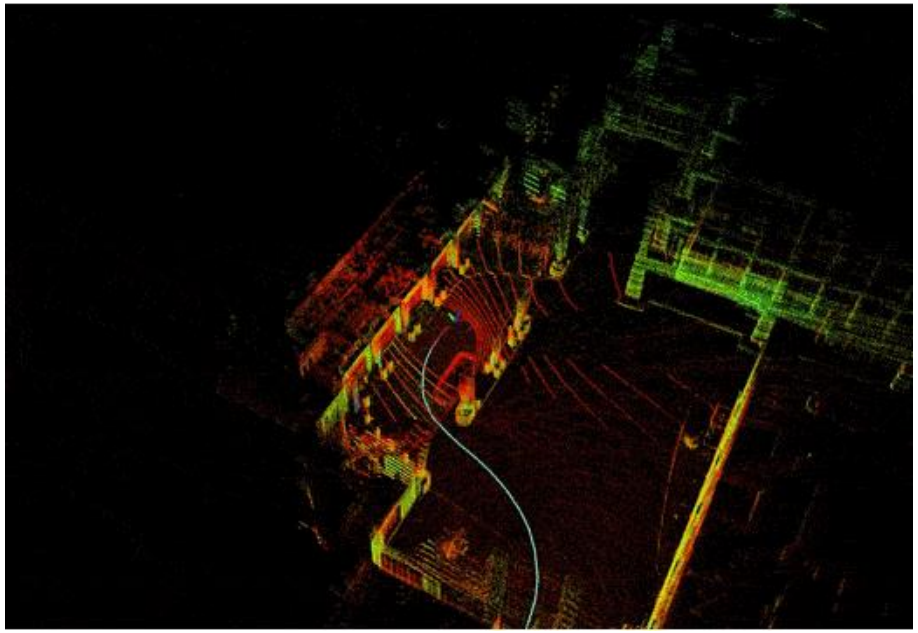
Python programming and practice (80%):

- Why learn Python for AI and UAS?
- Environment and basic Syntax
- Applications and practice
  - Functions
  - Read CSV file
  - Implement ICP for SLAM in Python

ROS for UAS and practice (20%):

- What is ROS?
- The role of ROS in UAS
- LiDAR SLAM with ROS



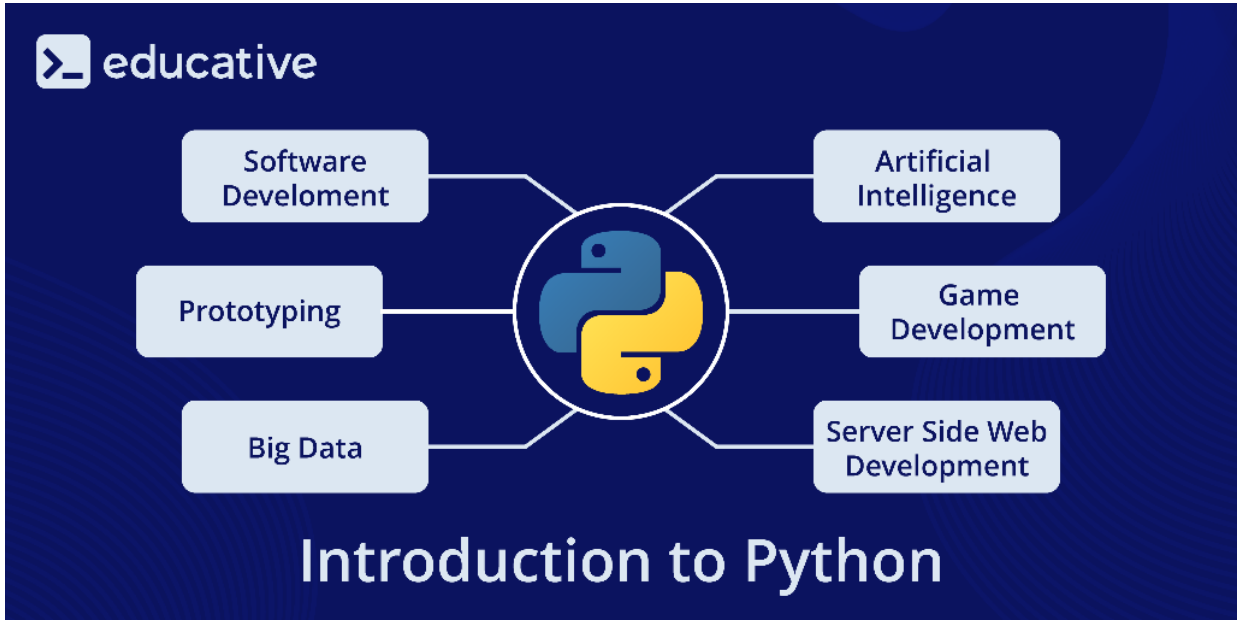


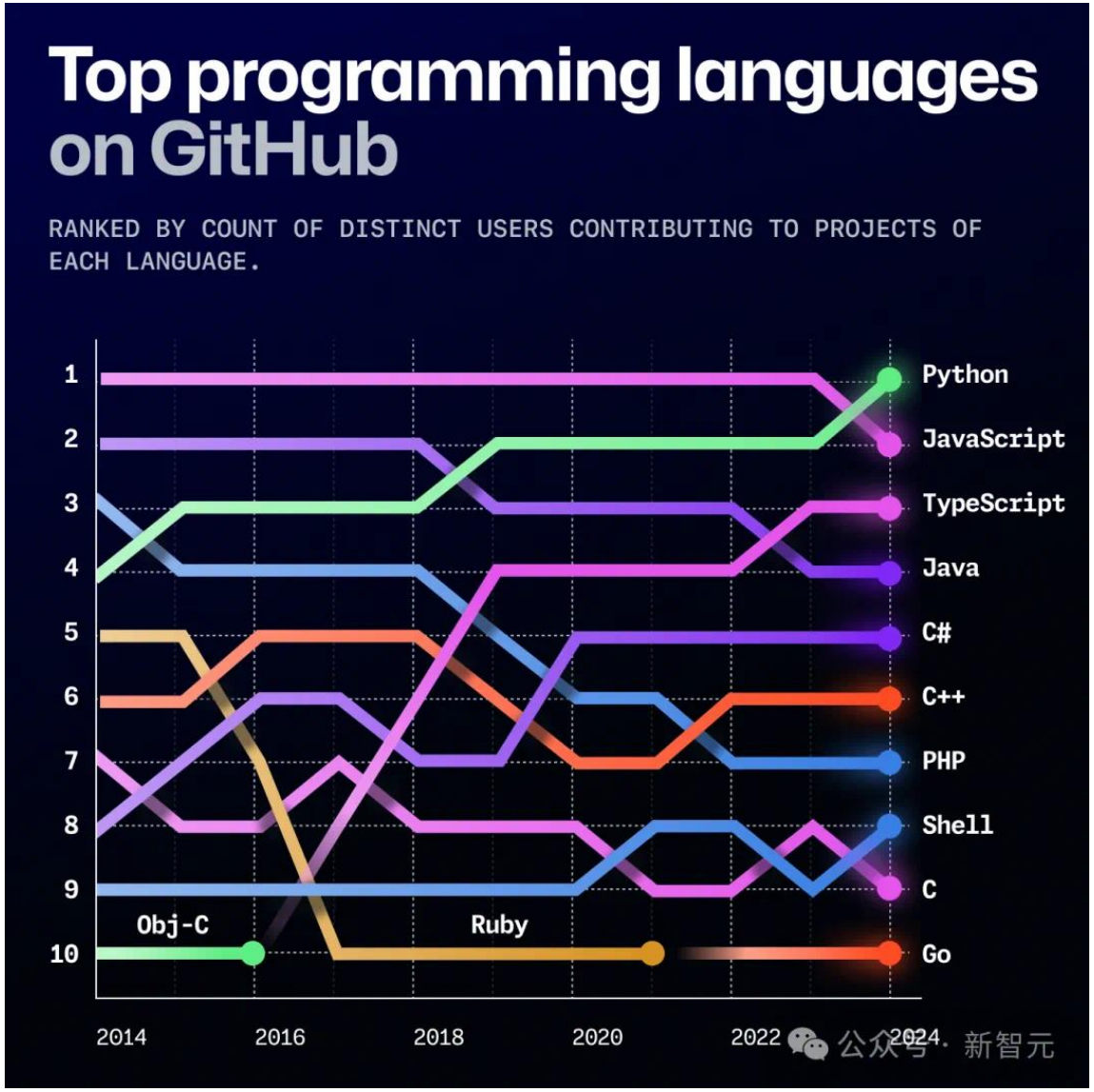
Introduction

Python

- What is Python?
  - Python is a high-level, interpreted programming language.
  - Known for its simplicity and readability, making it beginner-friendly.
  - Widely used in various fields such as web development, data analysis, artificial intelligence, and scientific computing.
- Why learn Python?
  - Versatility: It can be used for data science, AI, automation, and more.
  - Wide Application:
    - Machine Learning (e.g., TensorFlow, Scikit-learn)
    - Data Analysis and Visualization (e.g., Pandas, Matplotlib)
    - Web Development (e.g., Django, Flask)
    - Automation (e.g., Scripting, Selenium)
  - Community Support: Large community with extensive libraries and frameworks.

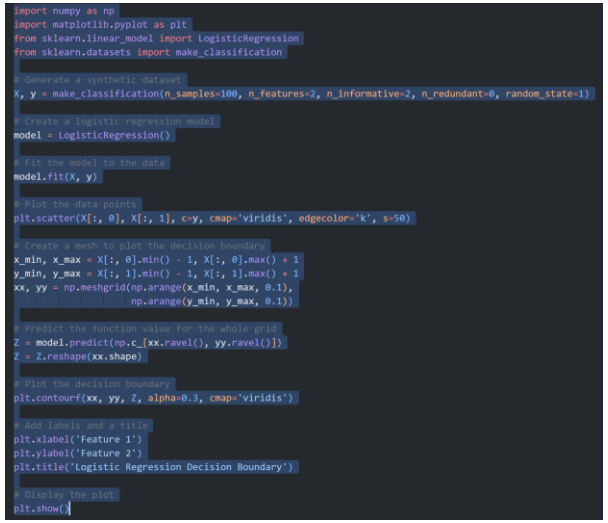
Python

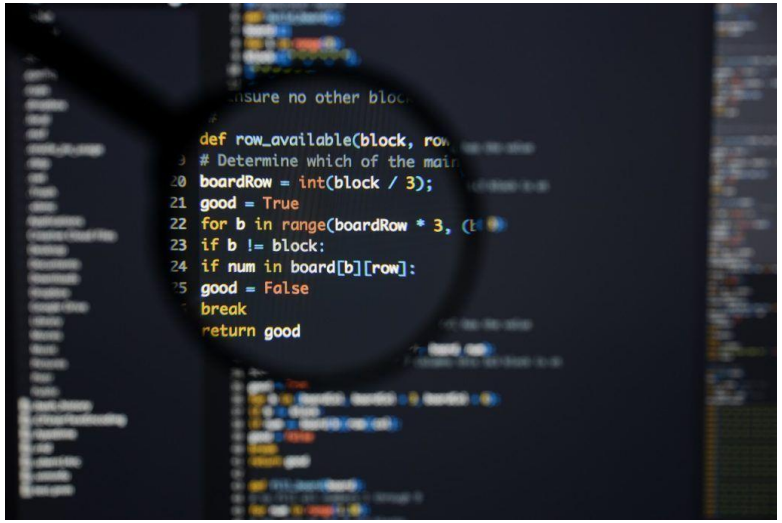


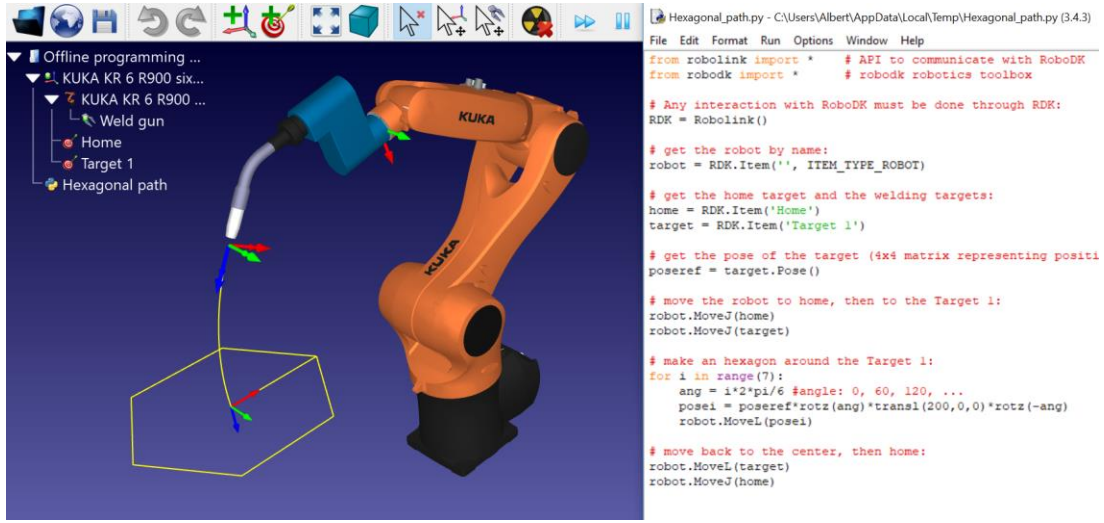


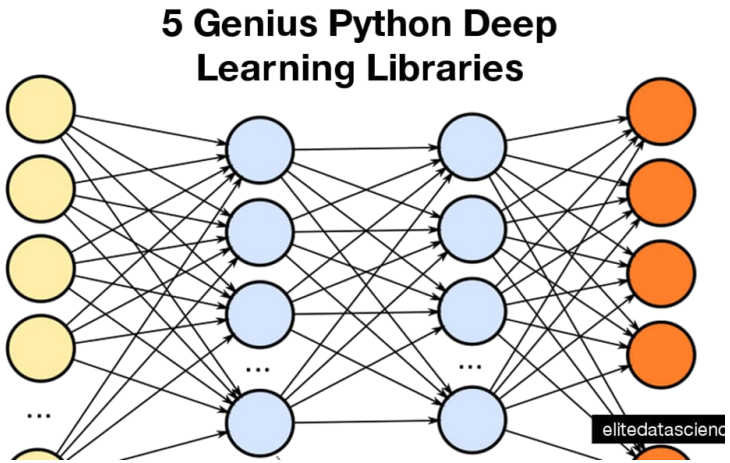
- <u>Become more and more popular:</u>
  - Super popular on GitHub community
  - Top 1 popular programming language on GitHub community, e.g. Robotics, AI, drones…

Example of Python applications









Python for Machine Learning

Python for Deep Learning

Python for Robotics

Course Overview

- Learning objectives:
  - Master the basic syntax of Python
  - Understand the core programming concepts
  - Be able to write Python programs independently
  - Learn debugging and error handling
- What to expect:
  - Interactive Learning: Concepts will be explained with practical examples
  - Hands-On Practice: Short exercises for each topic
  - Comprehensive Coverage: Basics of Python including syntax, data types, functions, and libraries

https://www.python.org

python™

Setting Up the Environment in Your Laptop

Installing Python

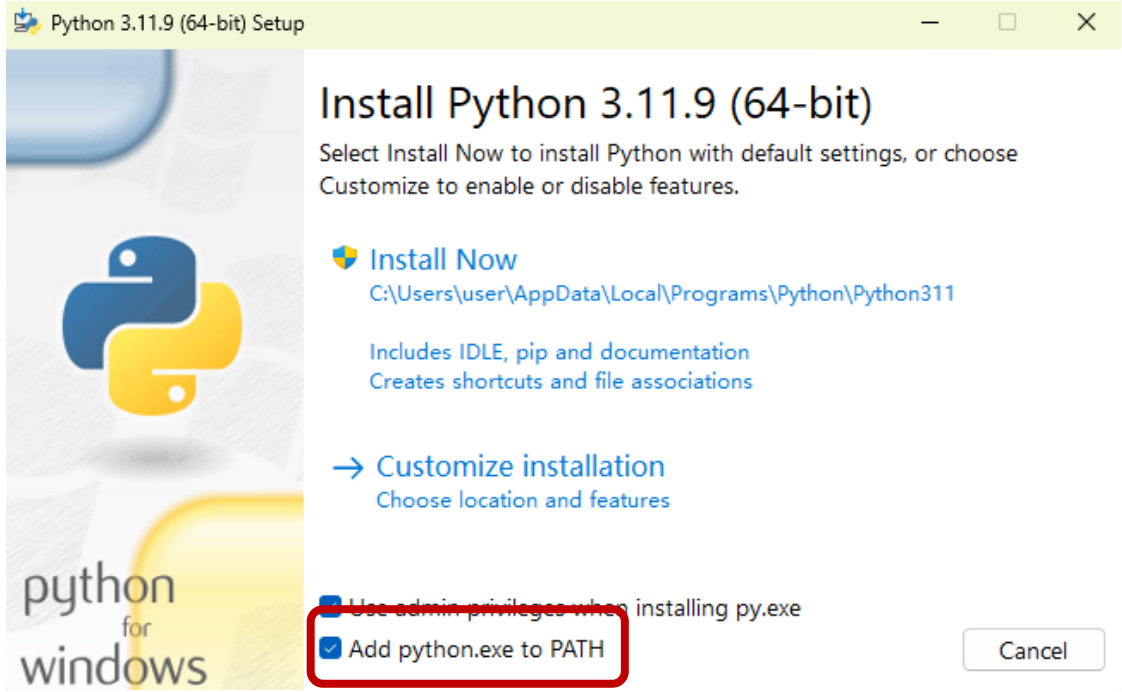
Python can be installed on Windows, macOS, and Linux operating systems. The installation process is straightforward on all three platforms.

- *On **Windows**, download the Python installer (.exe file) from the official website and run it. During installation, make sure to check the "Add Python to PATH" box.*
- *On **macOS**, Python comes pre-installed, but it's an older version. To get the latest Python, use Homebrew or download the pkg installer from the Python website.*
- *On **Linux**, use the native package manager like apt, yum, pacman, etc., to install Python and pip. For example, on Ubuntu and Debian, run **sudo apt install python3**.*
- *After installation, verify that Python is installed properly by running the **python --version** or **python3 --version** on the command line. The pip package manager is also installed with Python.*

Guide to Install Python on Windows

General Steps:

- Download the Python installer from the official Python website https://www.python.org/.
- Run the installer and select 'Add Python to PATH' before proceeding.
- Follow the on-screen instructions to complete the installation.



Considerations:

- Windows doesn't come with Python pre-installed.
- Ensure to select the 'Add Python to PATH' option for easy access from the command line.
- Python Windows installers are available for both 32-bit and 64-bit versions.

```
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user>python --version
Python 3.11.9

C:\Users\user>pip --version
pip 24.0 from C:\Users\user\AppData\Local\Programs\Python\Python311\
Lib\site-packages\pip (python 3.11)
```

Check the version of Python and pip

Guide to Install Python on macOS

General Steps:

- Python 2.x is pre-installed on older macOS versions, but it's recommended to install Python 3.x.
- Download the Python installer for macOS from the Python website.
- Open the installer and follow the instructions to install Python.

Steps to Install Python via Homebrew:

- First, install Homebrew if you don't have it already. Open Terminal and run:
  `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`
- Next, update Homebrew's package database so it has info on the latest package versions:
  `brew update`
- Now, install Python 3 by running:
  `brew install python`
- To verify Python is installed, run `python3 --version` in Terminal.
- Homebrew will also install pip for you. Check it with pip3 –version.

Guide to Install Python on Linux

General Steps:

- Most Linux distributions come with Python pre-installed.
- To install or update Python, use the package manager specific to your distribution:
  - **apt**: For Debian, Ubuntu, Pop!_OS, Linux Mint
  - **yum**: For RHEL, CentOS, Fedora, openSUSE
  - **pacman**: For Arch Linux and Manjaro
  - **apk**: For Alpine Linux

On Ubuntu (most popular version):

- Update apt's package index:
  `sudo apt update`
- Install Python 3 and pip:
  `sudo apt install python3 python3-pip`
- Verify:
  `python3 --version` and `pip3 --version`

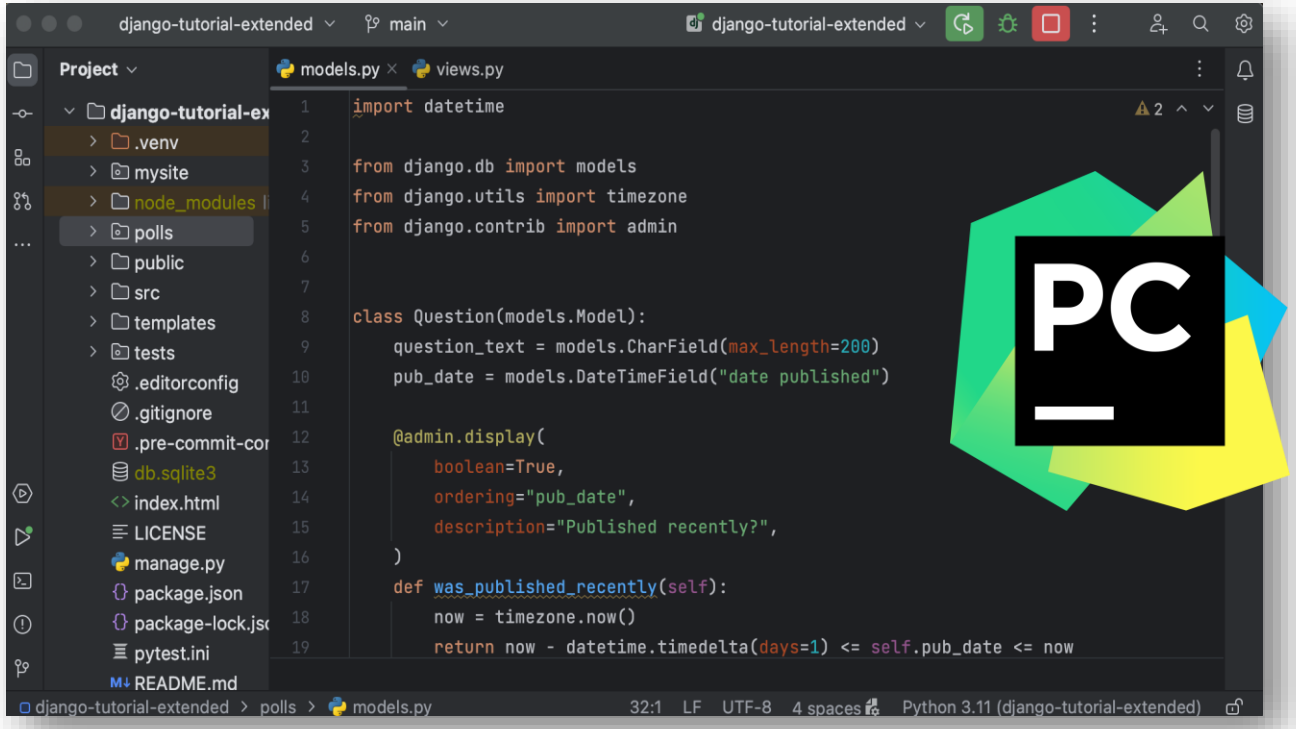
Introduction to IDEs

- <u>What is IDE?</u>
  - Definition: "An Integrated Development Environment is a software suite that consolidates basic tools for software development".
  - Key Features:
    - Code Editor
    - Debugger
    - Compiler/Interpreter
    - Intelligent Code Completion
    - Syntax Highlighting
    - Version Control Integration
  - Most popular IDEs

    PyCharm, VSCode, and Jupyter Notebook.

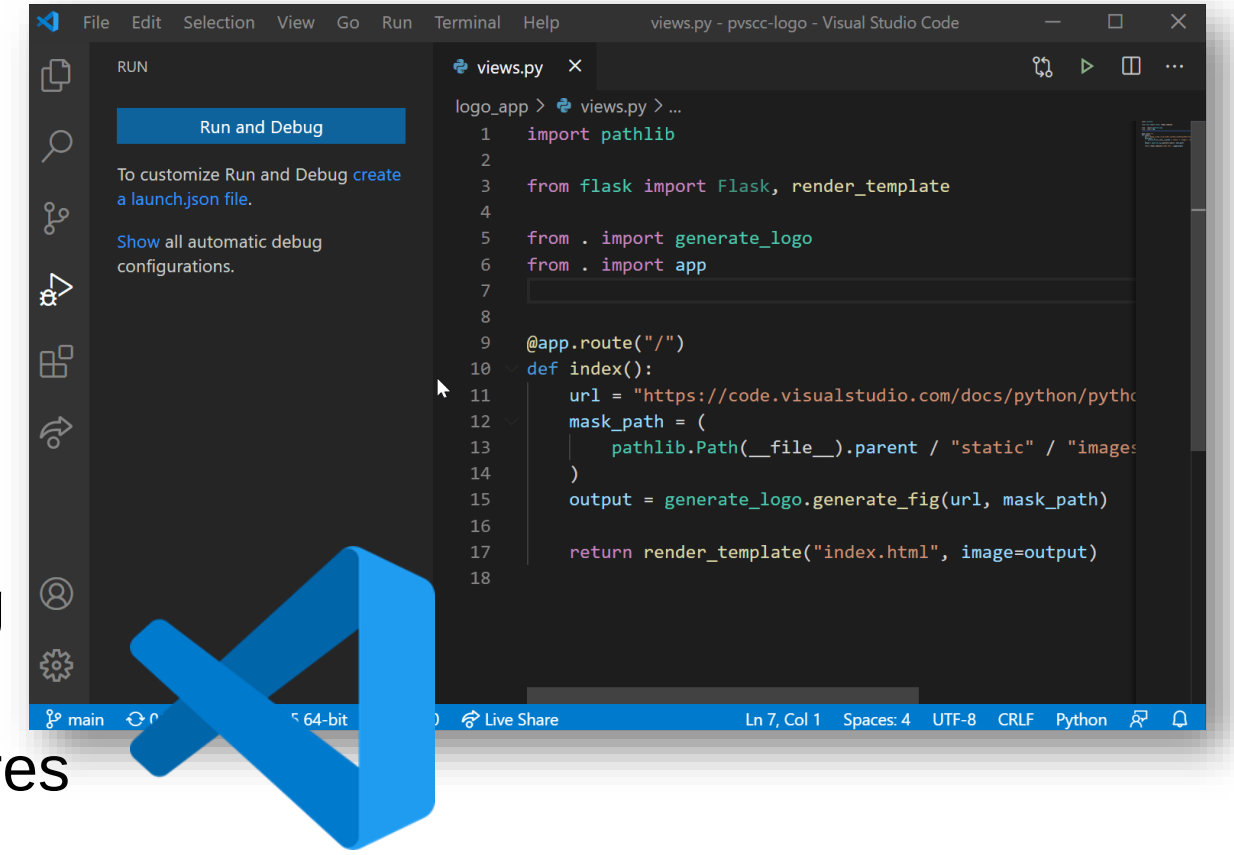
PyCharm

- PyCharm, developed by JetBrains, is a robust IDE designed specifically for Python. It is cross-platform, working on Windows, macOS, and Linux.
- Advantages:
  - ✓ Python-specific features
  - ✓ Powerful debugging tools
  - ✓ Excellent code navigation
  - ✓ Built-in project management
- Limitations:
  - × Resource intensive
  - × Steeper learning curve
  - × Be overwhelming for beginners



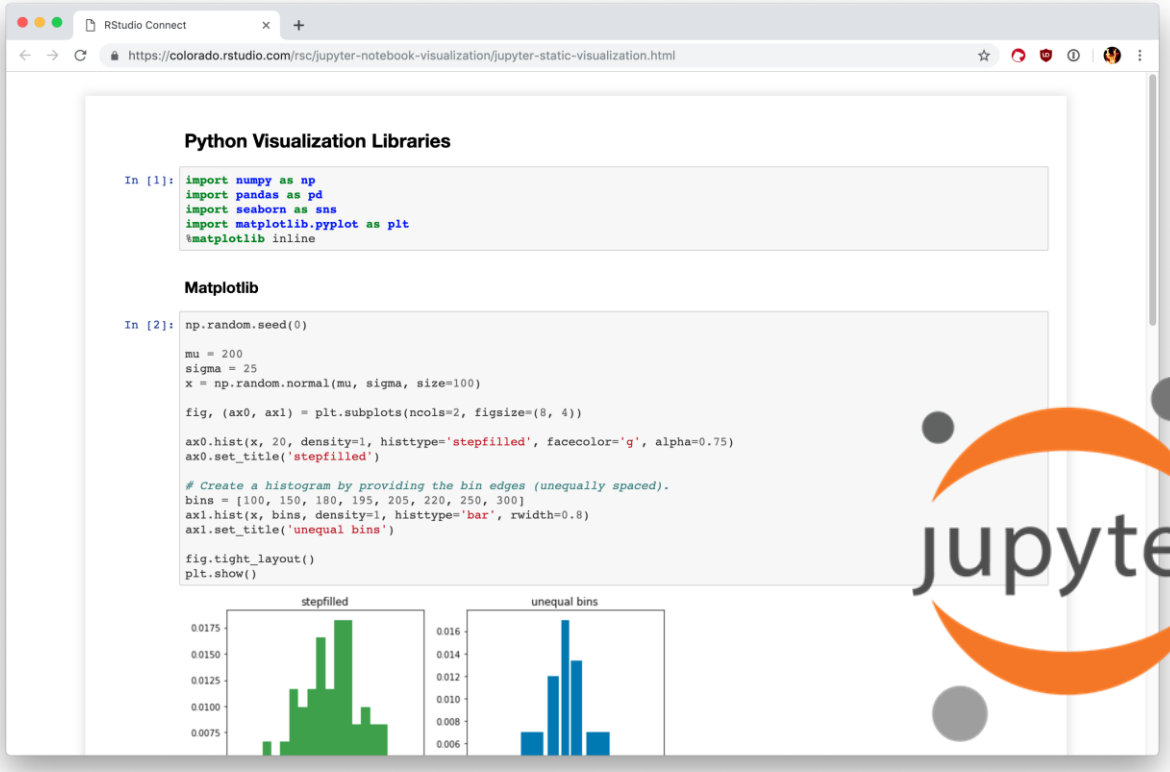
Visual Studio Code (VSCode) most popular

- Visual Studio Code is a lightweight and highly customizable code editor from Microsoft. VSCode is free and open-source, running on Windows, macOS, and Linux.
- Advantages:
  - ✓ Lightweight and fast
  - ✓ Highly customizable
  - ✓ Multi-language support
  - ✓ Active community
- Limitations:
  - × Configuration can be overwhelming
  - × Less specialized Python tools
  - × Requires setup for advanced features



Jupyter Notebook (on web browser)

- Jupyter Notebook is a popular IDE for data analysis and data science. It integrates with various data science libraries and runs in a web browser, ensuring cross-platform compatibility.
- Advantages:
  - ✓ Intuitive interface for data work
  - ✓ Cell-by-cell code execution
  - ✓ Strong community support
  - ✓ Perfect for experimentation
- Limitations:
  - × Not suitable for large projects
  - × Version control challenges
  - × Limited project structure support



Running Python Code

- In Python programming, you'll write programs in plain text files. By convention, files containing Python code use the extension, and there's no distinction between scripts or executable programs and modules. All of them will use the same extension (.py).
- First, fire up your favorite text editor and create a new file *hello.py* containing the following code:

```
print("Hello, World!")
```

- Command line execution: (python + path)

```
python .\hello.py
```

- Display of output:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\LAB\AAE4011> python .\hello.py
Hello, World!
```

Running Python Code

- Run Python scripts on Interactive Mode：
  1. Open Command Prompt or Terminal:
     - For Windows: Press Win + R; type "cmd" and press Enter
  2. Start Python interpreter:
     - Type "python" at the command prompt
     - Press Enter
     - What you'll see: (The ">>>" prompt indicates Python is ready for input)

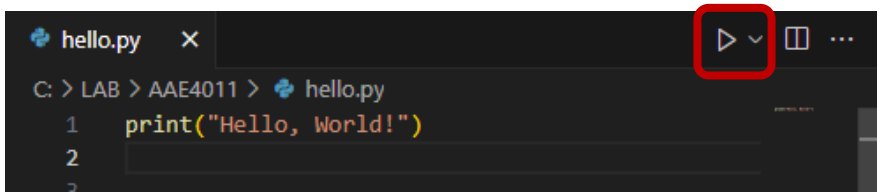
```
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user>python
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

  3. Enter Python statements:

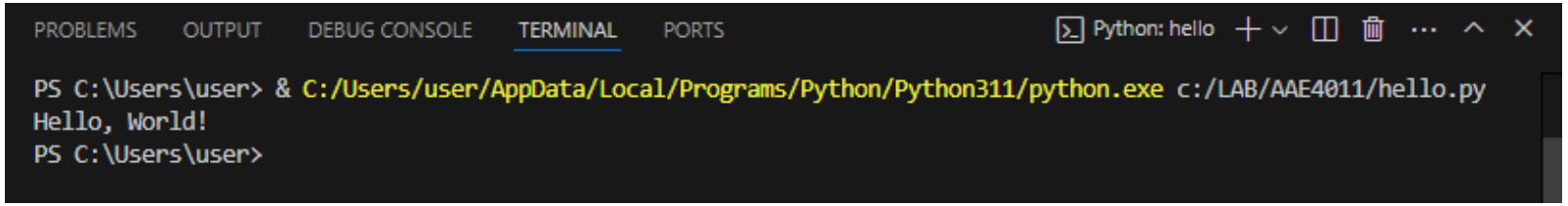
```
>>> print("Hello, World!")
Hello, World!
>>>
>>> print([x for x in range(10)])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> |
```

Running Python Code

- Run Python scripts on IDEs (take VScode for example):
  1. Select the Run Python File in Terminal play button in the top-right of the editor.

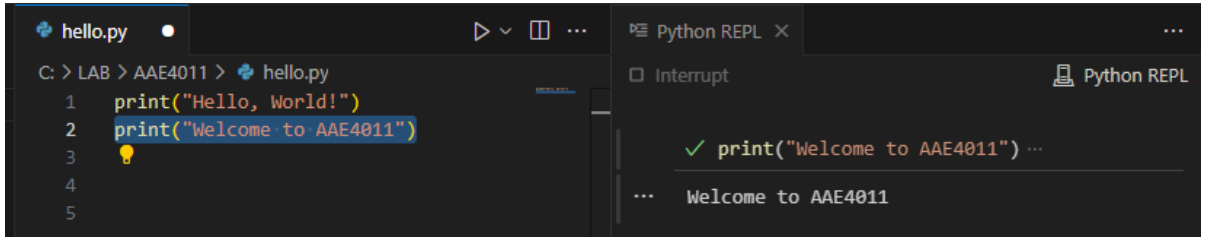


The button opens a terminal panel in which your Python interpreter is automatically activated, then runs the specified script:



2. Right-click anywhere in the editor window, and then select **Run > Python File in Terminal** (which saves the file automatically).

3. Select one or more lines, then press **Shift+Enter**, or right-click and select Run Selection/Line in Python Terminal. This option is convenient for testing just a part of a file.



Basic Syntax and Data Types

Python Syntax

- A sample code:

```
# This is a comment
print("Hello World") # Basic print statement
if True:
    print("Indented") # Indentation is important
```

- Key rules:
  - Indentation matters (use 4 spaces)
  - Case sensitive
  - Lines end without semicolons
  - Comments use #
- Execute the script (output):

```
Hello World
Indented
```

Variables in Python

- A sample code:
  - ✓ Valid Names:

```
name = "John"        # String
age = 25             # Integer
height = 1.75        # Float
is_student = True    # Boolean
```

  - × Invalid Names:

```
76_count = 76        # Can't start with number
user@name = "Tom"    # Can't use special characters
class = "Python"     # Can't use reserved words
```

- Variable rules:
  - No declaration needed
  - Dynamic typing
  - Case sensitive (age ≠ Age)
  - Must start with letter or underscore_
  - Cannot use Python reserved words

```
>>> name = "John"
>>> 76_count = 76
  File "<stdin>", line 1
    76_count = 76
      ^
SyntaxError: invalid decimal literal
>>>
```

Common Data Types

- Integers (int):

```
age = 25
population = 1_000_000 # Using _ for readability
```

- Floating Point (float):

```
height = 1.75
pi = 3.14159
```

- Strings (str):

```
name = "Alice"
message = 'Hello, World!'
address = """ 11 Yuk Choi Road,
Hung Hom, Kowloon, Hong Kong """
```

- Booleans (bool)

```
is_student = True
is_male = False
```

Python Basic Operations

```
# Addition
x = 5 + 3 # 8
x += 3     # Same as: x = x + 3

# Subtraction
y = 10 - 4 # 6
y -= 3      # Same as: y = y - 3

# Multiplication
z = 4 * 3 # 12
z *= 3     # Same as: z = z * 3

# Division
a = 15 / 3 # 5.0 (returns float)
a /= 3      # Same as: a = a / 3

# Modulus (Remainder)
b = 17 % 3 # 2

# Exponentiation
c = 2 ** 3 # 8 (2 to the power of 3)
```

```
# Using + operator
first = "Hello"
last = "World"
message = first + " " + last     # "Hello World"

# String repetition
echo = "Ha" * 3                  # "HaHaHa"

# String joining
words = ["Python", "is", "fun"]
sentence = " ".join(words)       # "Python is fun"

print(text.upper())          # "PYTHON PROGRAMMING"

print(text.endswith("ing")) # True

Text = "Python"
print("Length: {len(text)}") # 6
```

Control Structures

Conditional Statements (if, elif, else)

- A sample code:

```
# Basic grade calculator
score = 3.5 # GPA score

# Grade determination using if-elif-else
if score >= 3.7: # First check: A grade
    print("Grade: A")
elif score >= 2.7: # Second check: B grade
    print("Grade: B")
elif score >= 1.7: # Third check: C grade
    print("Grade: C")
else:              # If none above: D grade
    print("Grade: D") # Output: Grade: B
```

Always starts with if and a condition

There can be 0 or more elif clauses

The else clause has no conditionand is executed if nothing else is done

Tests are always tried in order

Since score is not greater than or equal to 3.7, the first block is skipped.
Since score is greater than or equal to 2.7, the second block is executed.
The remaining conditions are not checked, and the program ends.

Test and try yourself

Question:

Write a Python program to determine the range of a number x using the following criteria:

- If x is greater than or equal to 100, print "Large"
- If x is greater than or equal to 50 but less than 100, print "Medium"
- If x is greater than or equal to 20 but less than 50, print "Small"
- If x is less than 20, print "Tiny"

Use if-elif-else structure to complete this program.

You can test your code with x = 75.

Loops (for, while)

```
for item in sequence:
    # Code to execute for each item


# For Loop - Simple counting
print("Counting from 1 to 5:")
for num in range(1, 6):
    print(num)
```

Output:

1
2
3
4
5

```
while condition:
    # Code to execute while condition is True

# While Loop - Counter
count = 0
while count < 5:
    print(count)
    count += 1
```

While this is true

Do this

0
1
2
3
4

Loops (break, continue)

Break: Terminate the loop immediately

```
# Break example - Stop at 5
for num in range(10):
    if num == 5:
        break
    print(num)
```

Output:

0
1
2
3
4

Continue: Skip the current iteration and continue to the next iteration

```
# Continue example - Skip number 2
for num in range(5):
    if num == 2:
        continue
    print(num)
```

0
1
3
4

Test and try yourself

Question:

Please use **a for loop** to calculate the sum of numbers from 1 to 100.

Functions

Defining Functions

Functions:

- A function is a group of statements that exist within a program for the purpose of performing a specific task
- The same naming rules apply for both variables and functions
- we have been using a number of Python's built-in functions, including:
  - `print()`
  - `range()`
  - … etc

Syntax

```
def name(arg1, arg2, ... kw=val):
    # Function body
    # do stuff with args and val
    # return a value (optional)
    return result
    # Optional return statement
```

Example with optional parameter

```
def greet(name, greeting="Hello"):
    """Demonstrates function with default parameter"""
    return f"{greeting}, {name}!"

# Example calls
print(greet("Alice")) # Using default greeting
print(greet("Bob", "Good morning")) # Custom greeting

# Output:
# Hello, Alice!
# Good morning, Bob!
```

Parameters and Return Values

```python
def calculate_total(items, discount=0, tax_rate=0.1, shipping_fee=5):
 # Calculate subtotal
 subtotal = sum(items.values())
 # Apply discount if any
 if discount > 0:
     discount_amount = subtotal * (discount / 100)
     subtotal -= discount_amount

 # Calculate tax
 tax = subtotal * tax_rate

 # Calculate final total with shipping
 final_total = subtotal + tax + shipping_fee

 return subtotal, tax, final_total

# Example usage:
shopping_cart = {
    "book": 20.0,
    "laptop": 1000.0,
    "coffee": 5.0
}
```

```python
# Basic usage with default parameters
sub, tax, total = calculate_total(shopping_cart)
print(f"Basic purchase:")
print(f"Subtotal: ${sub:.2f}")
print(f"Tax: ${tax:.2f}")
print(f"Total: ${total:.2f}")

# Using keyword arguments with discount
sub, tax, total = calculate_total(
    items=shopping_cart, discount=10, # 10% discount
    tax_rate=0.08, # 8% tax
    shipping_fee=0 # Free shipping
)
print(f"\nSpecial offer purchase:")
print(f"Subtotal: ${sub:.2f}")
print(f"Tax: ${tax:.2f}")
print(f"Total: ${total:.2f}")
```

Scope and Lifetime of Variables

- **Python global variables** are those which are not defined inside any function and have a global scope.
- **Python local variables** are those which are defined inside a function and their scope is limited to that function only.

```
# Example 1: Local Variable Scope
def demonstrate_local_scope():
    """Shows how local variables are confined to function scope"""
    local_var = "I love PolyU" # local variable
    print(f"Inside function: {local_var}") # Works fine

# Try to access local variable outside function
demonstrate_local_scope() # print(local_var)
# This will raise NameError - variable not accessible
```

Scope and Lifetime of Variables

```python
# Example 2: Global vs Local Variables
global_var = "This is accessible everywhere" # global variable

def show_scope_differences():
    local_var = "Only visible inside function" # local variable
    print(f"Can access global: {global_var}") # Can access global
    print(f"Can access local: {local_var}") # Can access local

# Main program
show_scope_differences()
print(f"Can access global: {global_var}") # Works fine
# print(local_var) # Would raise NameError


# Example 3: Modifying Global Variables
counter = 0 # global variable

def modify_global():
    global counter # Must declare global to modify
    counter += 1
    print(f"Counter inside function: {counter}")

modify_global()
print(f"Counter in main program: {counter}")
```

Test and try yourself

Question:

Write a function to calculate the value of a quadratic function

$f(x) = 3x^2 + 5x + 10$.

Requirements:

- The function should take one parameter x
- Return the calculated result
- Handle both integer and decimal inputs

Data Structures

Lists

- **A container** that holds a number of other objects in a given order.
- **To create a list,** put a number of expressions in square brackets.

```
# Create an empty list
L1 = []

# Create a list with 3 integers
L2 = [6,7,8]

# Create a list with 3 strings
L3 = ['I', 'Love', 'HongKong']

# Lists do not have to be homogenous
L4 = ['AAE', 4011, 2025]
```

```
# Elements can be accessed using an integer index,
item = list[index]
print(L4[0])
#OUTPUT:
# 'AAE'

# To get a range of elements from a list, use:
L4[0:2]   # Get the first two items in a list
# ['AAE', 4011]

L4[0] = 'PolyU' # Change item in a list by the
order
print(L4[0])
# 'PolyU'
```

Tuples

- **Tuples are iterable arrays or objects** like list.
- **Tuples are immutable –** once created, unchangble.
- **Redeclare** is needed to add or remove items.

```
#  The items in tuple can be different types
tup1 = ('Google', 'Runoob', 1997, 2000)
tup2 = (1, 2, 3, 4, 5 )


#  The () can be removed if more than one items
are contained
tup3 = "a", "b", "c", "d"
print(type(tup3))


#OUTPUT:
# <class 'tuple'>
```

```
# When there is only one item in a tuple, ','
must be added after the item
tup1 = (50)
print(type(tup1))
#OUTPUT:
# <class 'int'>

tup1 = (50,)
print(type(tup1))
#OUTPUT:
# <class 'tuple'>

# Tuple is unchangecble
tup1[0] = 40
#OUTPUT:
# TypeError: 'tuple' object does not support
item assignment
```

Test and try yourself

Question:

Given 10 numbers (24, 45, 12, 67, 83, 34, 21, 59, 77, 90), please use a for loop to find the minimum number among them.

Dictionaries

- A collection of **pairs/items** where each pair has a **key** and a **value**
- Identify values by descriptive and unique **name (key)** instead of order of a list
- Key are unordered unless explicitly sorted

```python
# Use {} to creat empty dictionary
emptyDict = {}

# Print the dictionary
print(emptyDict)

# Check length of the dictionary
print("Length:", len(emptyDict))

# Check type of the variable
print(type(emptyDict))
```

```python
D = {'Chen':'A', 'Hung':'B', 'Wong':'C'}
D['Chen’]

# Update value for key 'Chen'
D['Chen'] = 'C'
D['Chen']
```

Python 3.13 (64-bit)

```
>>> D = {'Chen':'A', 'Hung':'B', 'Wong':'C'}
>>> D['
```

Test and try yourself

Given a list of names, determine the frequency of each name and identify which name appears the most frequently.

Given

```
names = ["Alice", "Bob", "Alice", "David", "Carol", "Bob", "Alice", 'Claine']
```

Modules and Libraries

Import Modules

- **Module is** a file containing Python code which may include functions, classes, or variables
- Python contains vast collection of module, including **build-in** modules and **external** ones.

```
# Use build-in Math module for mathmatic constants
and functions
import math

print(math.pi)
#OUTPUT:
#3.14159265358979

print(math.e)
#2.718281828459045

# Find absolute value using fabs() function
a = -10
print (math.fabs(a))
#10

# Find the power of exp using exp() function
test_float = 0.00
print (math.exp(test_float))
#1.0
```

```
# Use build-in Random module to create random value
import random

# Generate random floats
print(random.random())
#OUTPUT:
#0.9847936392493448

# Generate random integers
print(random.randint(0,10))
# 8

# Generate random item from a List or a String
list1 = [8, 5, 6, 2, 0, 4]
print(random.choice(list1))
# 4

# Genarate a random item from the string
string = 'ILovePolyU'
print(random.choice(string))
# 'I'
```

Introduction to Libraries

- **Libraries** are external modules that offer additional functions, classes, and methods with higher efficiency.
- Libraries cater to specific needs (data analysis, web scraping, and machine learning).

```
# Numpy for scientific computations with matrices,
arrays, or large datasets.
import numpy as np

# Type of the numpy is the same as the items
a = np.array([2, 3, 4])
print(a.dtype)
#OUTPUT:
#int64
b = np.array([1.2, 3.5, 5.1, 3.2])
print(b.dtype)
#float64

# Shape of the numpy can be changed using
reshape() function
print(b.shape)
#(4,)
b = b.reshape(2,2)
print(b.shape)
#(2, 2)
```

```
# Pandas can import data from a variety of file
formats, such as CSV, JSON, SQL, and Microsoft
Excel.
import pandas as pd

# Create a simple DataFrame
data = {'Students': ['Chen', 'Hung', 'Wong'],
'Ages': [17, 18, 17]}
df = pd.DataFrame(data)
print(df)

#OUTPUT:
#    Students  Ages
# 0     Chen    17
# 1     Hung    18
# 2     Wong    17
```

Test and try yourself

Given four linear functions, find the inverse of the four linear functions constructed matrix.

Given

$$
\begin{aligned}
f_1(x,y,z,w) &= 1x + 2y + 3z + 4w \\
f_2(x,y,z,w) &= 5x + 6y + 7z + 8w \\
f_3(x,y,z,w) &= 9x + 10y + 11z + 12w \\
f_4(x,y,z,w) &= 13x + 14y + 15z + 16w
\end{aligned}
$$

$$
\begin{bmatrix} f_1(x,y,z,w) \\ f_2(x,y,z,w) \\ f_3(x,y,z,w) \\ f_4(x,y,z,w) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}
$$

Tasks and Practice

Read a CSV file for data analysis and visualization

Time
Position x
Position y
Position z
velocity x
Velocity y
velocity z

Time	PosX	PosY	PosZ	QuatW	QuatX	QuatY	QuatZ	VelX	VelY	VelZ
160965206	0.00077	0.00047	0.00169	0.99965	0.01271	-0.02337	0.00024	-0.00173	0.00178	-0.00122
160965206	0.0008	0.00045	0.0017	0.99965	0.01277	-0.0233	0.00024	-0.00029	0.00086	-0.00103
160965206	0.00074	0.00029	0.0018	0.99964	0.0128	-0.02337	0.00027	0.00164	0.00085	-0.00032
160965206	0.00074	0.00032	0.00181	0.99964	0.01285	-0.02342	0.00027	0.00188	0.00134	-0.00127
160965207	0.00076	0.00032	0.00182	0.99964	0.0129	-0.02343	0.00026	0.00119	0.00129	-0.00106
160965207	0.00075	0.0003	0.00184	0.99964	0.0129	-0.02346	0.00028	0.00008	0.00119	-0.00132
160965207	0.0007	0.00034	0.00186	0.99964	0.01294	-0.02341	0.00027	0.00045	0.0017	-0.0012
160965207	0.00071	0.00034	0.00188	0.99964	0.01297	-0.02338	0.00027	-0.00073	0.00104	-0.00078
160965207	0.00071	0.00028	0.00189	0.99964	0.0129	-0.02337	0.00028	-0.00144	-0.00144	-0.00035
160965207	0.00066	0.0003	0.00191	0.99964	0.01288	-0.02334	0.00026	-0.00041	-0.00027	-0.00205
160965207	0.00071	0.00031	0.00191	0.99964	0.0129	-0.02335	0.00026	0.00009	0.00036	-0.00143
160965207	0.00072	0.00025	0.0019	0.99964	0.01288	-0.02337	0.00028	0.00082	0.00002	-0.00138
160965207	0.00068	0.00029	0.00191	0.99964	0.01288	-0.02337	0.00026	0.00085	-0.00012	-0.00165
160965207	0.00073	0.00033	0.00193	0.99964	0.01288	-0.0234	0.00026	0.00196	0.00048	-0.00178
160965207	0.00074	0.00027	0.00189	0.99964	0.01287	-0.02344	0.00028	0.00266	0.00071	-0.00158
160965207	0.00071	0.00031	0.00191	0.99964	0.01289	-0.02345	0.00027	0.001	0.00018	-0.00117
160965207	0.00073	0.00033	0.00195	0.99964	0.01288	-0.02346	0.00026	0.00172	0.00004	-0.00045
160965207	0.00071	0.00024	0.00195	0.99964	0.01286	-0.02346	0.00028	0.00121	0.00133	-0.00032
160965207	0.00068	0.0003	0.00195	0.99964	0.01287	-0.02345	0.00026	0.00117	0.00053	-0.00096
160965207	0.00072	0.00033	0.00197	0.99964	0.01288	-0.02345	0.00026	0.00046	0.00136	-0.00085
160965207	0.00067	0.00027	0.00199	0.99964	0.01286	-0.02344	0.00028	0.0004	0.00126	-0.00051
160965207	0.00063	0.00032	0.00197	0.99964	0.01287	-0.02344	0.00027	0.00044	0.00101	-0.00192
160965207	0.00067	0.00033	0.00198	0.99964	0.01289	-0.02343	0.00026	0.00085	0.00094	-0.00149

General Steps to read the CSV and visualize the data:

Import the head files

```
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.spatial.transform import Rotation as R
```

Setup the path of files

```
file_path1 = 'C:/Users/Weisong Wen/Dropbox/PolyU AAE Teaching-Weisong/PolyU AAE4011 Artificial Intelligence in Unmanned Autonomous Systems/AAE4011-S22425-2/Code/week3/Task1 Plot Trajectory/Solution1.csv'  # Replace with your first CSV file path
file_path2 = 'C:/Users/Weisong Wen/Dropbox/PolyU AAE Teaching-Weisong/PolyU AAE4011 Artificial Intelligence in Unmanned Autonomous Systems/AAE4011-S22425-2/Code/week3/Task1 Plot Trajectory/Solution2.csv'  # Replace with your second CSV file path
```

Read the data

```
data1 = pd.read_csv(file_path1)
data2 = pd.read_csv(file_path2)
time1 = data1['Time']
data1['Time'] = (data1['Time'] - time1.iloc[0])/(1e09)
pos1 = data1[['PosX', 'PosY', 'PosZ']]
vel1 = data1[['VelX', 'VelY', 'VelZ']]
```

Plot the results…

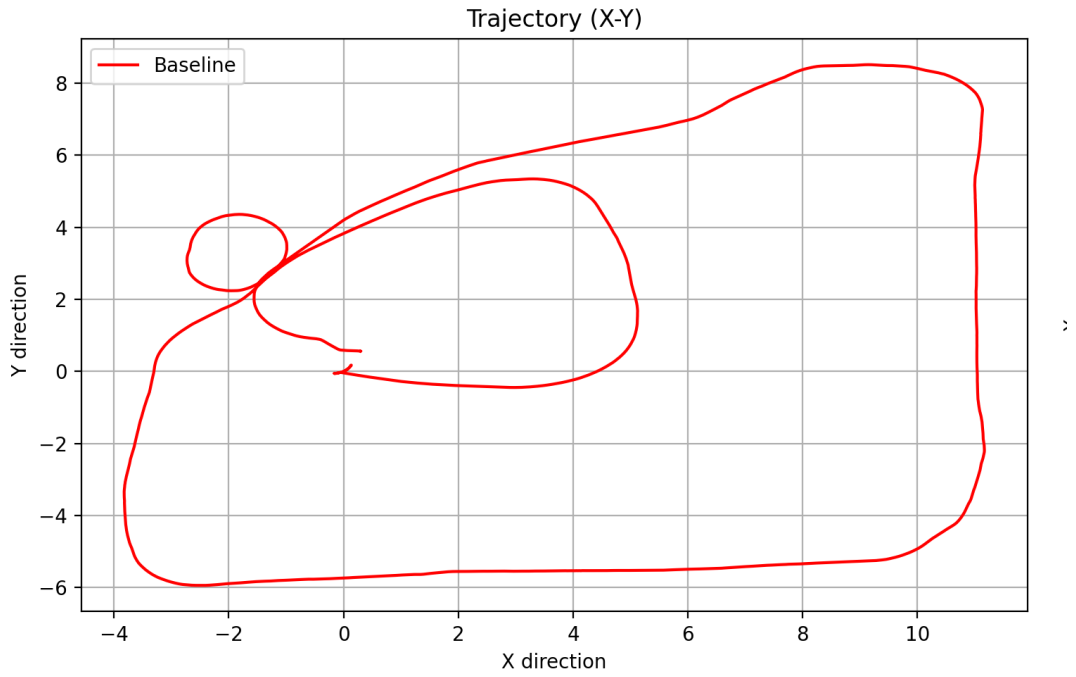
In class task 1 (plot 1 trajectory)

Example data: https://github.com/weisongwen/AAE4011-S22425/blob/main/Code/week3/Task1%20Plot%20Trajectory/Solution1.csv

(feel free to ask ChatGPT), also you could try the Deepseek at https://www.deepseek.com/

1	Time	PosX	PosY	PosZ	QuatW	QuatX	QuatY	QuatZ	VelX	VelY	VelZ
2	1609652069786850048	0.00077	0.00047	0.00169	0.99965	0.01271	-0.02337	0.00024	-0.00173	0.00178	-0.00122
3	1609652069853597696	0.00080	0.00045	0.00170	0.99965	0.01277	-0.02330	0.00024	-0.00029	0.00086	-0.00103
4	1609652069920298240	0.00074	0.00029	0.00180	0.99964	0.01280	-0.02337	0.00027	0.00164	0.00085	-0.00032
5	1609652069986794240	0.00074	0.00032	0.00181	0.99964	0.01285	-0.02342	0.00027	0.00188	0.00134	-0.00127
6	1609652070053454592	0.00076	0.00032	0.00182	0.99964	0.01290	-0.02343	0.00026	0.00119	0.00129	-0.00106
7	1609652070120257024	0.00075	0.00030	0.00184	0.99964	0.01290	-0.02346	0.00028	0.00008	0.00119	-0.00132
8	1609652070186946048	0.00070	0.00034	0.00186	0.99964	0.01294	-0.02341	0.00027	0.00045	0.00170	-0.00120
9	1609652070253766144	0.00071	0.00034	0.00188	0.99964	0.01297	-0.02338	0.00027	-0.00073	0.00104	-0.00078
10	1609652070320379392	0.00071	0.00028	0.00189	0.99964	0.01290	-0.02337	0.00028	-0.00144	-0.00144	-0.00035
11	1609652070387004416	0.00066	0.00030	0.00191	0.99964	0.01288	-0.02334	0.00026	-0.00041	-0.00027	-0.00205
12	1609652070453810432	0.00071	0.00031	0.00191	0.99964	0.01290	-0.02335	0.00026	0.00009	0.00036	-0.00143
13	1609652070520438528	0.00072	0.00025	0.00190	0.99964	0.01288	-0.02337	0.00028	0.00082	0.00002	-0.00138

The data



What you are expected to get!

In class task 2 (plot 2 trajectories)

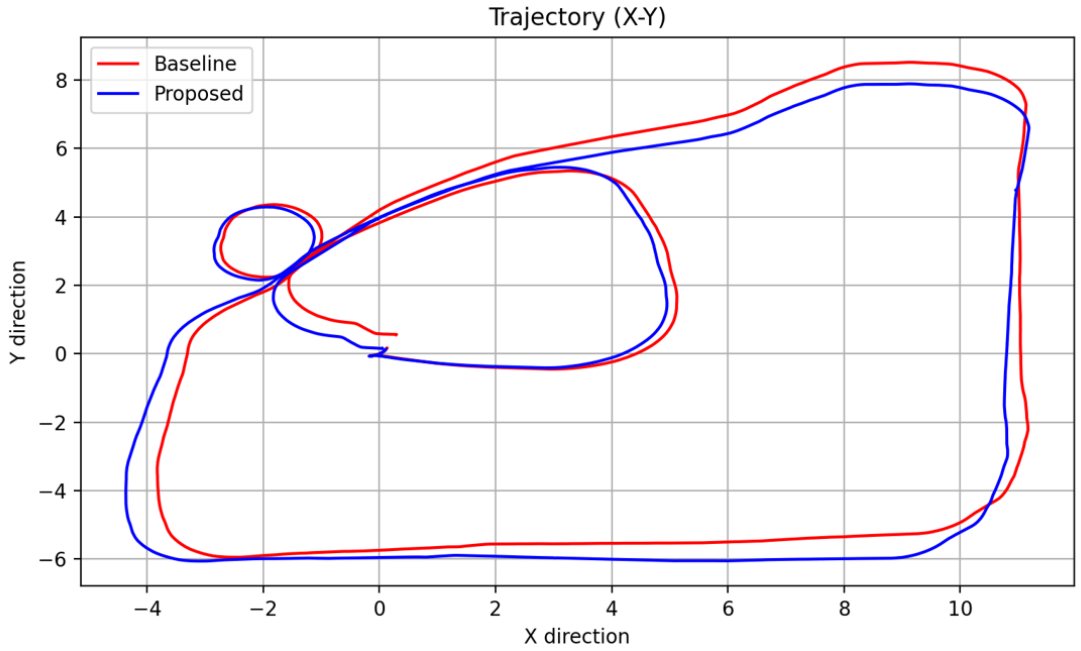
Example data 2: https://github.com/weisongwen/AAE4011-S22425/blob/main/Code/week3/Task1%20Plot%20Trajectory/Solution1.csv

Example data 2: https://github.com/weisongwen/AAE4011-S22425/blob/main/Code/week3/Task1%20Plot%20Trajectory/Solution2.csv

(feel free to ask ChatGPT), also you could try the Deepseek at https://www.deepseek.com/

1	Time	PosX	PosY	PosZ	QuatW	QuatX	QuatY	QuatZ	VelX	VelY	VelZ
2	1609652074954755072	0.03955	0.02978	0.00519	0.99687	0.01512	-0.02074	0.07476	0.07797	0.07832	0.00587
3	1609652075021330944	0.04615	0.03639	0.00428	0.99610	0.01811	-0.01676	0.08473	0.06326	0.06507	0.00196
4	1609652075088098048	0.05094	0.04132	0.00450	0.99537	0.01703	-0.01498	0.09341	0.04845	0.07084	0.00100
5	1609652075154668544	0.05520	0.04574	0.00413	0.99465	0.01316	-0.01107	0.10182	0.06214	0.07694	0.00537
6	1609652075221511936	0.05933	0.05038	0.00407	0.99377	0.01255	-0.00911	0.11039	0.05600	0.07506	0.00606
7	1609652075288313856	0.06338	0.05486	0.00355	0.99284	0.00982	-0.00605	0.11892	0.04468	0.07769	0.00125
8	1609652075355232256	0.06764	0.05990	0.00213	0.99174	0.00396	0.00126	0.12817	0.06304	0.07421	0.00310
9	1609652075421534720	0.07195	0.06537	0.00205	0.99049	0.00295	0.00077	0.13755	0.06845	0.08013	0.00365
10	1609652075488145920	0.07649	0.07113	0.00203	0.98904	0.00013	-0.00113	0.14762	0.05870	0.09311	-0.00212
11	1609652075554983936	0.08086	0.07717	0.00206	0.98746	-0.00011	-0.00113	0.15788	0.06999	0.09737	-0.00247
12	1609652075621566464	0.08502	0.08329	0.00200	0.98577	-0.00077	-0.00112	0.16809	0.06080	0.09312	0.00116
13	1609652075688190976	0.08927	0.08959	0.00209	0.98395	-0.00185	-0.00203	0.17840	0.06465	0.10245	0.00076
14	1609652075755511040	0.09347	0.09621	0.00217	0.98200	-0.00162	-0.00180	0.18885	0.05812	0.09935	0.00021
13	1609652070520438528	0.00072	0.00025	0.00190	0.99964	0.01288	-0.02337	0.00028	0.00082	0.00002	-0.00138

The data



What you are expected to get!

In class task 3 (plot 2 trajectories and velocity)

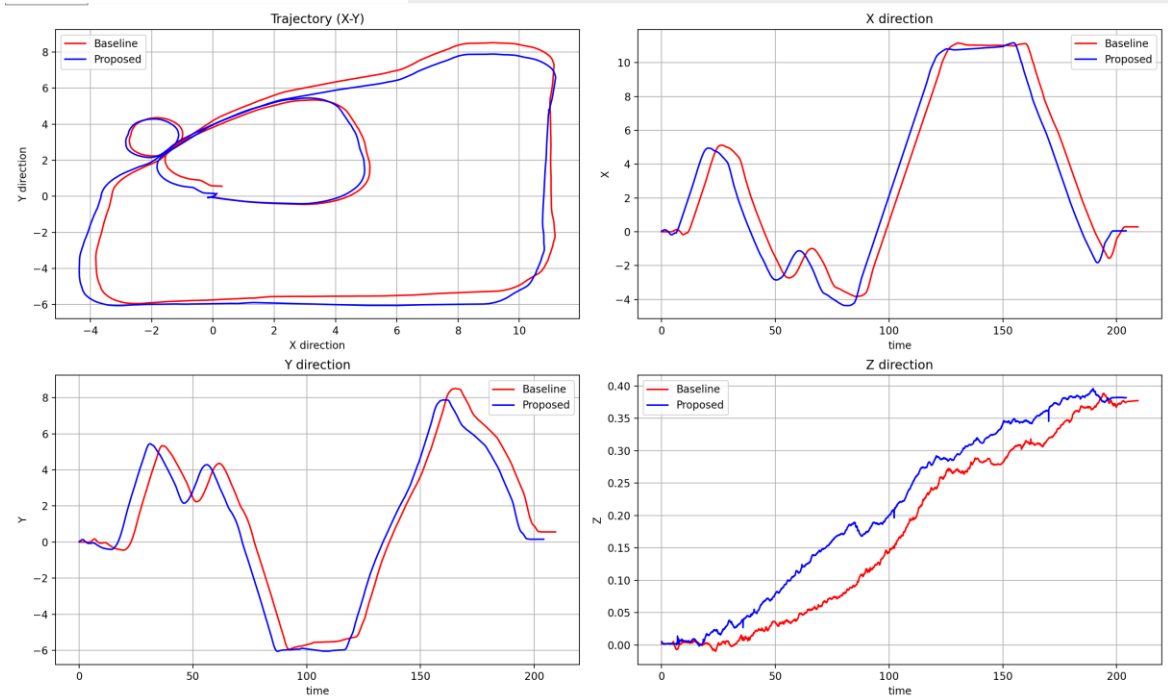
Example data 2: https://github.com/weisongwen/AAE4011-S22425/blob/main/Code/week3/Task1%20Plot%20Trajectory/Solution1.csv

Example data 2: https://github.com/weisongwen/AAE4011-S22425/blob/main/Code/week3/Task1%20Plot%20Trajectory/Solution2.csv

(feel free to ask ChatGPT), also you could try the Deepseek at https://www.deepseek.com/

1	Time	PosX	PosY	PosZ	QuatW	QuatX	QuatY	QuatZ	VelX	VelY	VelZ
2	1609652074954755072	0.03955	0.02978	0.00519	0.99687	0.01512	-0.02074	0.07476	0.07797	0.07832	0.00587
3	1609652075021330944	0.04615	0.03639	0.00428	0.99610	0.01811	-0.01676	0.08473	0.06326	0.06507	0.00196
4	1609652075088098048	0.05094	0.04132	0.00450	0.99537	0.01703	-0.01498	0.09341	0.04845	0.07084	0.00100
5	1609652075154668544	0.05520	0.04574	0.00413	0.99465	0.01316	-0.01107	0.10182	0.06214	0.07694	0.00537
6	1609652075221511936	0.05933	0.05038	0.00407	0.99377	0.01255	-0.00911	0.11039	0.05600	0.07506	0.00606
7	1609652075288313856	0.06338	0.05486	0.00355	0.99284	0.00982	-0.00605	0.11892	0.04468	0.07769	0.00125
8	1609652075355232256	0.06764	0.05990	0.00213	0.99174	0.00396	0.00126	0.12817	0.06304	0.07421	0.00310
9	1609652075421534720	0.07195	0.06537	0.00205	0.99049	0.00295	0.00077	0.13755	0.06845	0.08013	0.00365
10	1609652075488145920	0.07649	0.07113	0.00203	0.98904	0.00013	-0.00113	0.14762	0.05870	0.09311	-0.00212
11	1609652075554983936	0.08086	0.07717	0.00206	0.98746	-0.00011	-0.00113	0.15788	0.06999	0.09737	-0.00247
12	1609652075621566464	0.08502	0.08329	0.00200	0.98577	-0.00077	-0.00112	0.16809	0.06080	0.09312	0.00116
13	1609652075688190976	0.08927	0.08959	0.00209	0.98395	-0.00185	-0.00203	0.17840	0.06465	0.10245	0.00076
14	1609652075755511040	0.09347	0.09621	0.00217	0.98200	-0.00162	-0.00180	0.18885	0.05812	0.09935	0.00021
12	1609652070453810432	0.00071	0.00031	0.00191	0.99964	0.01290	-0.02335	0.00026	0.00009	0.00036	-0.00143
13	1609652070520438528	0.00072	0.00025	0.00190	0.99964	0.01288	-0.02337	0.00028	0.00082	0.00002	-0.00138

The data



What you are expected to get!

In class example 3: ICP for LiDAR SLAM

The Problem

- Given: two corresponding point sets:

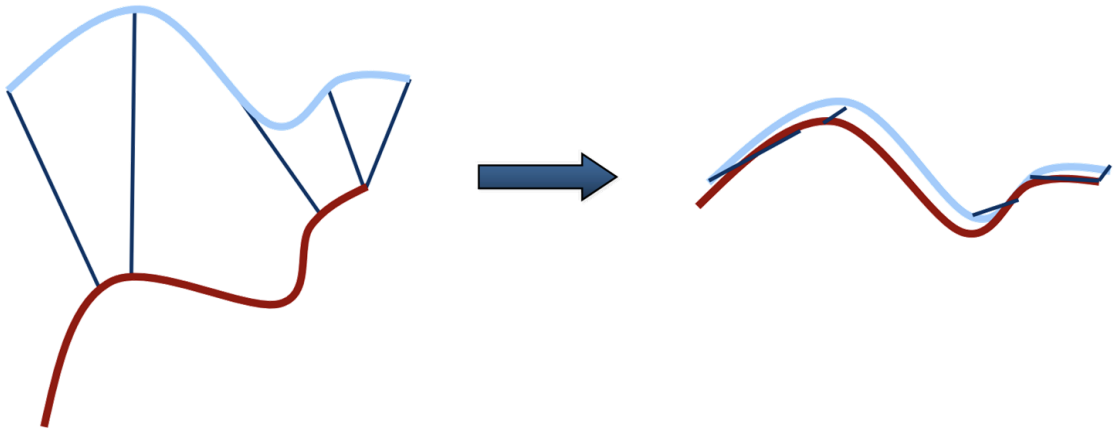
$$X = \{x_1, ..., x_n\}$$

$$P = \{p_1, ..., p_n\}$$

- Wanted: translation t and rotation R that minimizes the sum of the squared error:

$$E(R,t) = \frac{1}{N_p} \sum_{i=1}^{N_p} ||x_i - Rp_i - t||^2$$

Where x_i and p_i are corresponding points.



Iterative Closest Point (ICP) Matching: Iterative Closest Point (ICP) is a widely used algorithm in computer vision and robotics for aligning two 3D shapes or point clouds. The goal of ICP is to find the best transformation (translation and rotation) that minimizes the difference between two point clouds. Here's a brief overview of how the ICP algorithm works:

Source: https://cs.gmu.edu/~kosecka/cs685/cs685-icp.pdf

In class example 3: ICP for LiDAR SLAM

The Problem

SLAM: simultaneous localization and mapping

- Given: two corresponding point sets:

$$X = \{x_1, ..., x_n\}$$

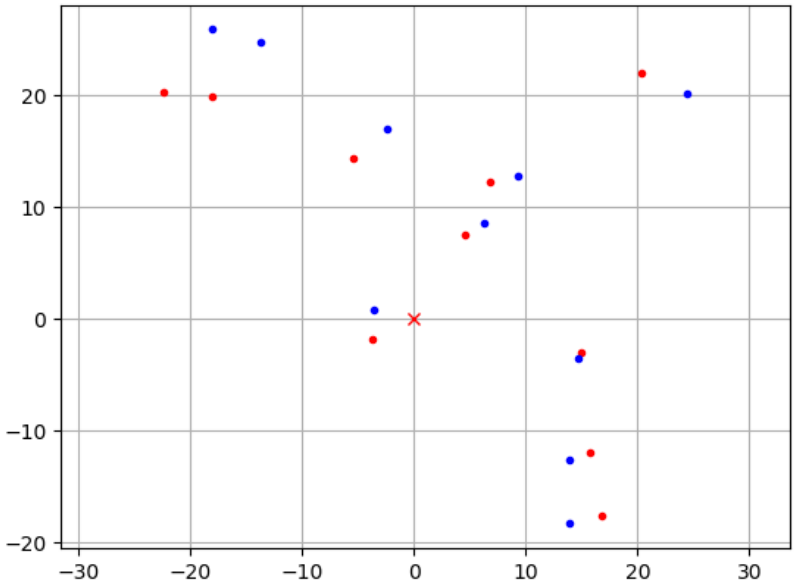
$$P = \{p_1, ..., p_n\}$$

- Wanted: translation t and rotation R that minimizes the sum of the squared error:

$$E(R,t) = \frac{1}{N_p} \sum_{i=1}^{N_p} ||x_i - Rp_i - t||^2$$

Where x_i and p_i are corresponding points.

Source: https://cs.gmu.edu/~kosecka/cs685/cs685-icp.pdf

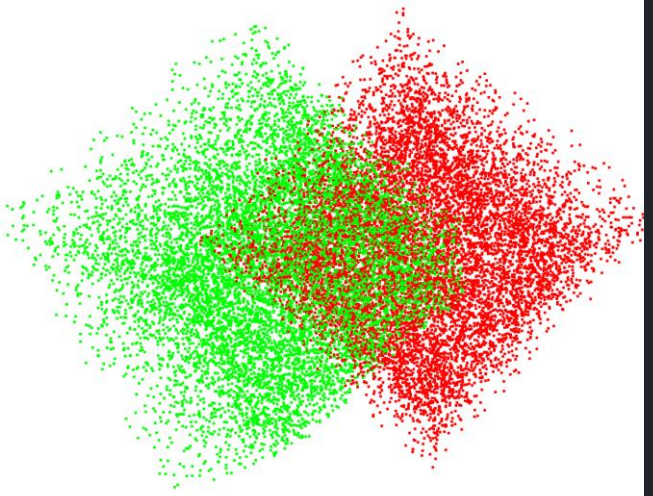


Iterative Closest Point (ICP) Matching:
Iterative Closest Point (ICP) is a widely used algorithm in computer vision and robotics for aligning two 3D shapes or point clouds. The goal of ICP is to find the best transformation (translation and rotation) that minimizes the difference between two point clouds. Here's a brief overview of how the ICP algorithm works:

In class task- replicate an example for ICP with Python

Ask ChatGPT:

python example for Iterative Closest Point (ICP) Matching, two point clouds with different colors, show the results of each iteration



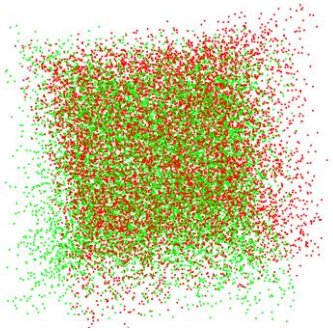
```
current transformation matrix:
[[ 0.99990402  0.00739769  0.01171414 -0.0620175 ]
 [-0.00746425  0.99995619  0.00564826  0.03496195]
 [-0.01167184 -0.00573516  0.99991543 -0.02231357]
 [ 0.          0.          0.          1.        ]]
current transformation matrix:
[[ 0.99960044  0.01360455  0.02477658 -0.1312305 ]
 [-0.01387743  0.99984456  0.01087491  0.07377986]
 [-0.02462478 -0.0112144   0.99963386 -0.04771487]
 [ 0.          0.          0.          1.        ]]
current transformation matrix:
[[ 0.99831112  0.02773721  0.05104471 -0.2692372 ]
 [-0.02893282  0.9993205   0.02283465  0.14881254]
 [-0.05037665 -0.02427295  0.99843528 -0.0954379 ]
 [ 0.          0.          0.          1.        ]]
Final transformation matrix:
[[ 0.99831112  0.02773721  0.05104471 -0.2692372 ]
 [-0.02893282  0.9993205   0.02283465  0.14881254]
 [-0.05037665 -0.02427295  0.99843528 -0.0954379 ]
 [ 0.          0.          0.          1.        ]]
```

Generate the source and target point clouds

```python
# Create a target point cloud
target = o3d.geometry.PointCloud()
target.points = o3d.utility.Vector3dVector(np.random.rand(10000, 3))

# Create a source point cloud by applying a known transformation to the target
source = o3d.geometry.PointCloud()
source.points = o3d.utility.Vector3dVector(np.random.rand(10000, 3))

# Apply a known transformation to simulate an initial misalignment
R = source.get_rotation_matrix_from_xyz((0.1, 0.1, 0.1))  # Rotation
t = np.array([0.5, -0.2, 0.3])  # Translation
transformation = np.eye(4)
transformation[:3, :3] = R
transformation[:3, 3] = t
source.transform(transformation)
```

Perform the registration

```python
# Perform ICP registration manually to visualize each iteration
current_transformation = trans_init
for i in range(max_iterations):
    reg_p2p = o3d.pipelines.registration.registration_icp(
        source, target, threshold, current_transformation,
        o3d.pipelines.registration.TransformationEstimationPointToPoint(),
        o3d.pipelines.registration.ICPConvergenceCriteria(max_iteration=1)
    )

    # Update the transformation
    current_transformation = reg_p2p.transformation @ current_transformation

    print("current transformation matrix:")
    print(current_transformation)

    # Visualize the result of the current iteration
    draw_registration_result(source, target, current_transformation, i + 1)
```



Representation of position and orientation

Pose (position and orientation):
In robotics and computer graphics, representing the position and orientation of an object in space is crucial for tasks such as navigation, manipulation, and visualization. Here are the common methods used to represent position and orientation:

Position Representation

1.Cartesian Coordinates:

1. The position of a point in 3D space is typically represented using Cartesian coordinates (x,y,z), which specify the point's location along the X, Y, and Z axes.

Orientation Representation

1.Euler Angles:

1. Euler angles represent orientation using three angles, typically denoted as (ϕ,θ,ψ) or (α,β,γ), which correspond to rotations around the principal axes (often Z, Y, X or X, Y, Z).
2. Advantages: Intuitive and easy to understand.
3. Disadvantages: Suffer from gimbal lock, where two axes become aligned, leading to a loss of one degree of freedom.

2.Rotation Matrices:

1. A 3x3 matrix that represents rotation in 3D space. Rotation matrices are orthogonal and have a determinant of 1.
2. Advantages: No gimbal lock, can be easily combined through matrix multiplication.
3. Disadvantages: Requires 9 numbers, though only 3 are independent.

Representation of position and orientation

Pose (position and orientation):
In robotics and computer graphics, representing the position and orientation of an object in space is crucial for tasks such as navigation, manipulation, and visualization. Here are the common methods used to represent position and orientation:

Position Representation

1.Cartesian Coordinates:

1. The position of a point in 3D space is typically represented using Cartesian coordinates (x,y,z), which specify the point's location along the X, Y, and Z axes.

3. Quaternions:

1. A quaternion is a four-element vector (w,x,y,z) that can represent 3D rotations without suffering from gimbal lock.
2. Advantages: Compact (4 numbers), no gimbal lock, efficient for interpolating rotations (slerp).
3. Disadvantages: Less intuitive than Euler angles.

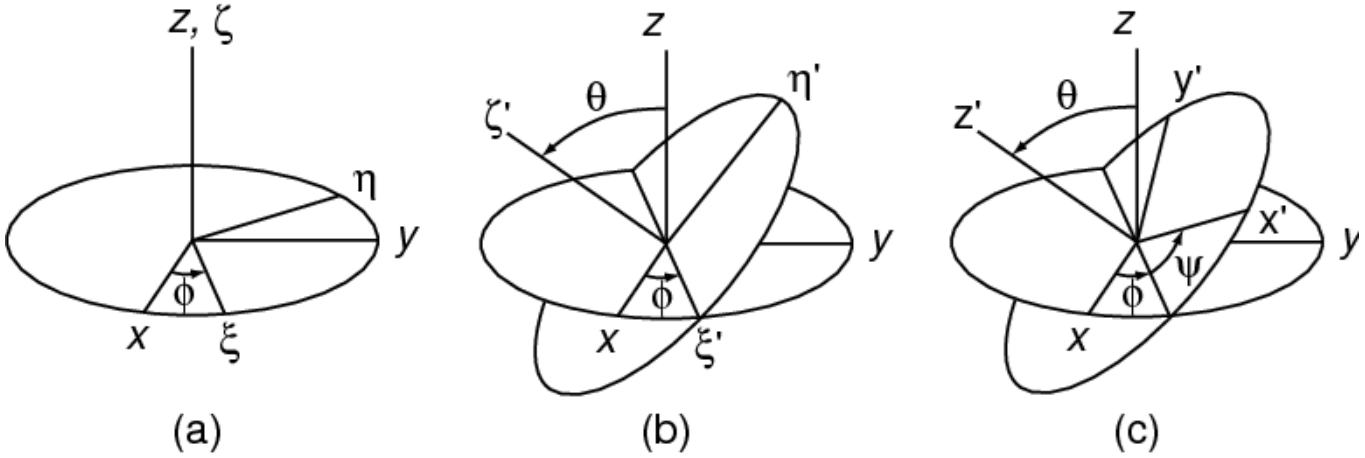
4.Homogeneous Transformation Matrices:

1. A 4x4 matrix that combines both rotation and translation, allowing for the representation of an object's position and orientation in a single matrix.
2. Advantages: Convenient for combining multiple transformations (e.g., rotation and translation) through matrix multiplication.
3. Disadvantages: Requires more storage (16 numbers) and computation.

Representation of position and orientation

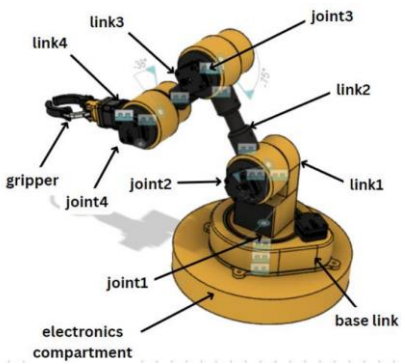
1. **Euler Angles to rotation matrix**

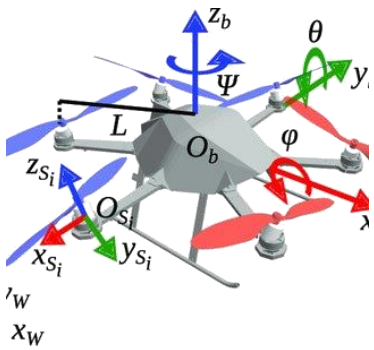
$$
\begin{aligned}
R &= R_z(\psi)R_x(\phi)R_y(\theta) \\
&= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \\
&= \begin{bmatrix} \cos\psi\cos\theta - \sin\phi\sin\psi\sin\theta & -\sin\psi\cos\phi & \sin\theta\cos\psi + \sin\phi\sin\psi\cos\theta \\ \sin\psi\cos\theta + \sin\phi\sin\theta\cos\psi & \cos\phi\cos\psi & \sin\theta\sin\psi - \sin\phi\cos\psi\cos\theta \\ -\sin\theta\cos\phi & \sin\phi & \cos\phi\cos\theta \end{bmatrix}
\end{aligned}
$$



Numpy:
Basic package for scientific computing with python. At its core, it is based on array arithmetic with ***N-dimensional data objects ndarray****.*

(ϕ,θ,ψ) is associated with yaw, pitch and roll angles.





[1] eCalc.
[2] Rituram.

Representation of position and orientation

1. **Euler Angles to Quaternions**

(α,β,γ) is associated with yaw, pitch and roll angles.

$$q_x(\alpha) = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right) \\ 0 \\ 0 \end{bmatrix}, \quad q_y(\beta) = \begin{bmatrix} \cos\left(\frac{\beta}{2}\right) \\ 0 \\ \sin\left(\frac{\beta}{2}\right) \\ 0 \end{bmatrix}, \quad q_z(\gamma) = \begin{bmatrix} \cos\left(\frac{\gamma}{2}\right) \\ 0 \\ 0 \\ \sin\left(\frac{\gamma}{2}\right) \end{bmatrix}$$

$$q_x(\alpha) \cdot q_y(\beta) = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right) \\ \cos\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right) \end{bmatrix}$$

$$q_{xyz} = q_x(\alpha) \cdot q_y(\beta) \cdot q_z(\gamma) = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right) \\ \cos\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right) \end{bmatrix} \cdot \begin{bmatrix} \cos\left(\frac{\gamma}{2}\right) \\ 0 \\ 0 \\ \sin\left(\frac{\gamma}{2}\right) \end{bmatrix}$$

$$q_{xyz} = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\cos\left(\frac{\gamma}{2}\right) - \sin\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\sin\left(\frac{\gamma}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\cos\left(\frac{\gamma}{2}\right) + \cos\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\sin\left(\frac{\gamma}{2}\right) \\ \cos\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\cos\left(\frac{\gamma}{2}\right) - \sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\sin\left(\frac{\gamma}{2}\right) \\ \cos\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\sin\left(\frac{\gamma}{2}\right) + \sin\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\cos\left(\frac{\gamma}{2}\right) \end{bmatrix}$$

```python
#! /usr/bin/env python3

# This program converts Euler angles to a quaternion.
# Author: AutomaticAddison.com

import numpy as np # Scientific computing library for Python

def get_quaternion_from_euler(roll, pitch, yaw):
  """
  Convert an Euler angle to a quaternion.

  Input
    :param roll: The roll (rotation around x-axis) angle in radians.
    :param pitch: The pitch (rotation around y-axis) angle in radians.
    :param yaw: The yaw (rotation around z-axis) angle in radians.

  Output
    :return qx, qy, qz, qw: The orientation in quaternion [x,y,z,w] format
  """
  qx = np.sin(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) - np.cos(roll/2) * np.sin(pitch/2) * np.sin(yaw/2)
  qy = np.cos(roll/2) * np.sin(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) * np.cos(pitch/2) * np.sin(yaw/2)
  qz = np.cos(roll/2) * np.cos(pitch/2) * np.sin(yaw/2) - np.sin(roll/2) * np.sin(pitch/2) * np.cos(yaw/2)
  qw = np.cos(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) * np.sin(pitch/2) * np.sin(yaw/2)

  return [qx, qy, qz, qw]

print(get_quaternion_from_euler(0.0, 0.0, 1.5708))
```

Representation of position and orientation

Ask ChatGPT:

python example code for Representation of position and orientation

1. Cartesian Coordinates for Position

```python
import numpy as np

# Position in 3D space
position = np.array([1.0, 2.0, 3.0])  # x, y, z coordinates
print("Position:", position)
```

2. Rotation Matrices

```python
# Euler angles (in radians)
euler_angles = np.array([np.pi / 4, np.pi / 6, np.pi / 3])  # Roll, Pitch, Yaw
rotation = R.from_euler('xyz', euler_angles)
print("Euler Angles (radians):", euler_angles)
print("Rotation Matrix from Euler Angles:\n", rotation.as_matrix())

# Define a rotation matrix directly
rotation_matrix = np.array([
    [0.866, -0.5, 0.0],
    [0.5, 0.866, 0.0],
    [0.0, 0.0, 1.0]
])
print("Rotation Matrix:\n", rotation_matrix)
```

Transformation

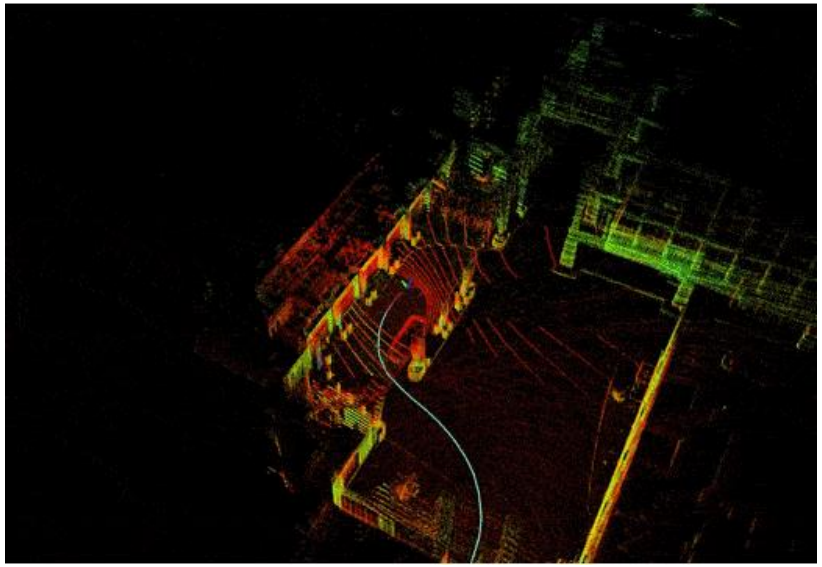
$$\mathbf{T}_{t_1}^{t_N} = \mathbf{T}_{t_1}^{t_2}\mathbf{T}_{t_2}^{t_3}\cdots\mathbf{T}_{t_{N-1}}^{t_N}$$

2. Quaternions and Homogeneous Transformation Matrices

```python
# Euler angles (in radians)
euler_angles = np.array([np.pi / 4, np.pi / 6, np.pi / 3])  # Roll, Pitch, Yaw
rotation = R.from_euler('xyz', euler_angles)
print("Euler Angles (radians):", euler_angles)
print("Rotation Matrix from Euler Angles:\n", rotation.as_matrix())

# Define a rotation matrix directly
rotation_matrix = np.array([
    [0.866, -0.5, 0.0],
    [0.5, 0.866, 0.0],
    [0.0, 0.0, 1.0]
])
print("Rotation Matrix:\n", rotation_matrix)

# Quaternion representation
quaternion = rotation.as_quat()  # Convert from Euler angles
print("Quaternion:", quaternion)

# Homogeneous transformation matrix
transformation_matrix = np.eye(4)
transformation_matrix[:3, :3] = rotation_matrix  # Rotation part
transformation_matrix[:3, 3] = position  # Translation part
print("Homogeneous Transformation Matrix:\n", transformation_matrix)
```

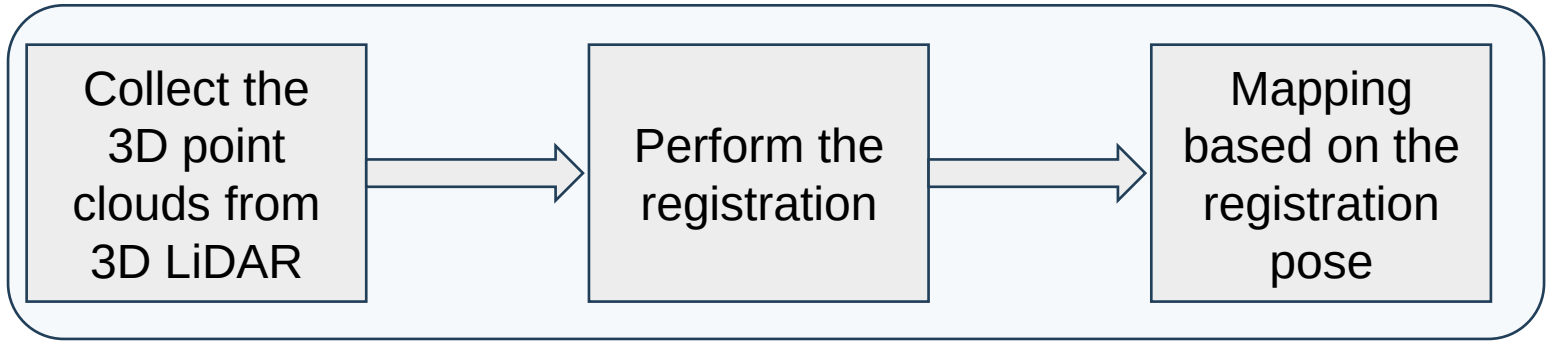
SLAM: simultaneous localization and mapping

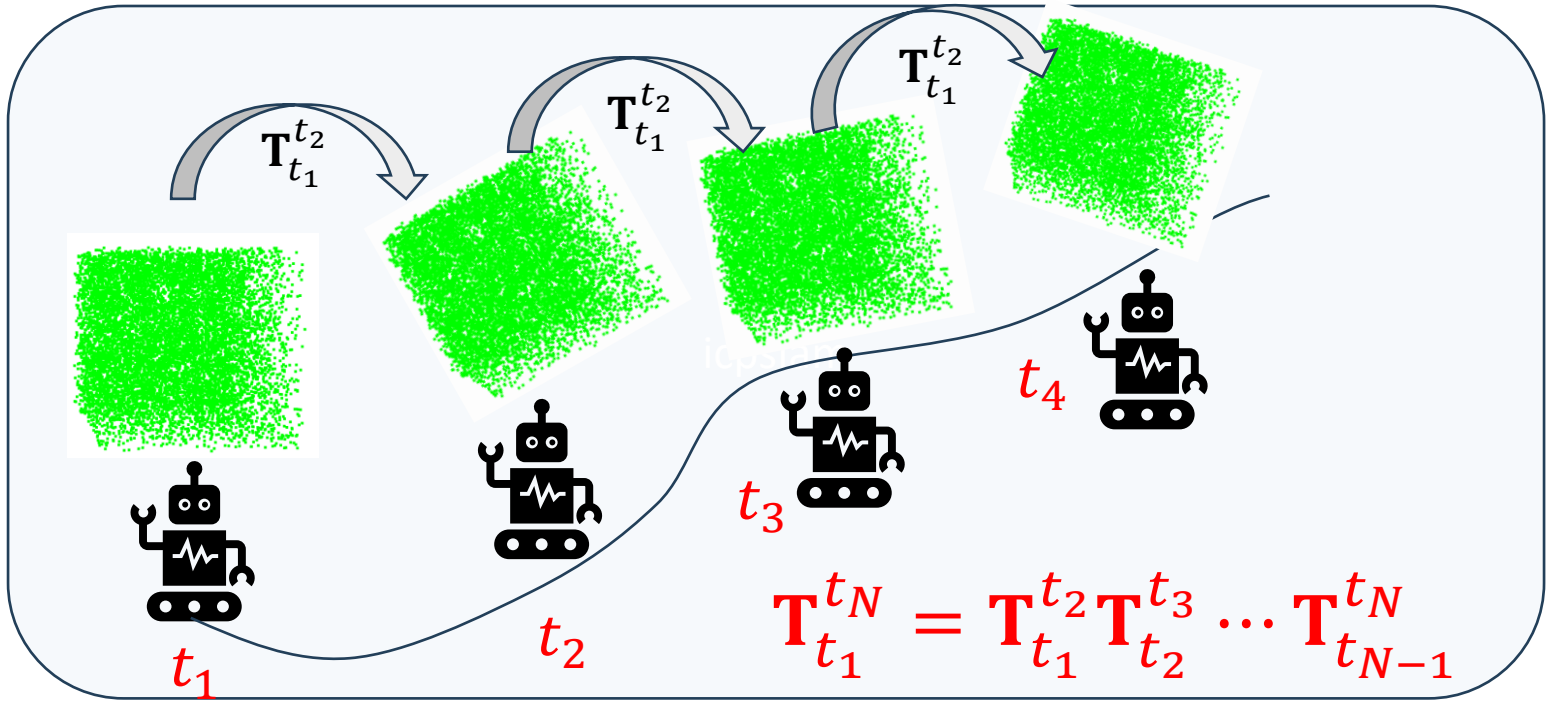
lcpslam.py



LiDAR Inertial SLAM on ROS

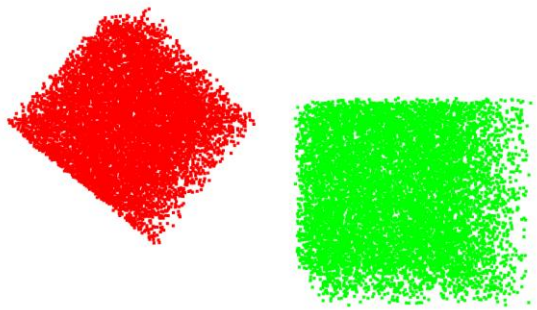
LiDAR SLAM (Simultaneous Localization and Mapping) is a technique used in robotics and autonomous vehicles to build a map of an environment while simultaneously keeping track of the device's location within that map.





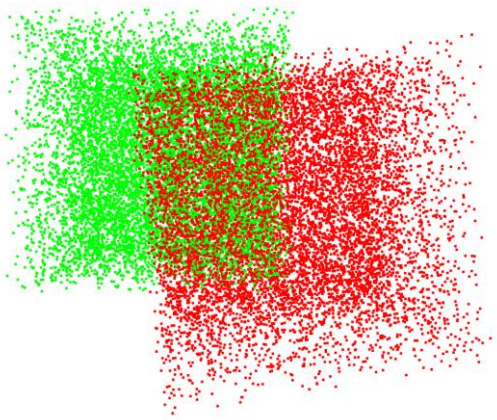
What if you set the maximum iterations to 10?

Ask ChatGPT:
python example for Iterative Closest Point (ICP) Matching, two point clouds with different colors, show the results of each iteration

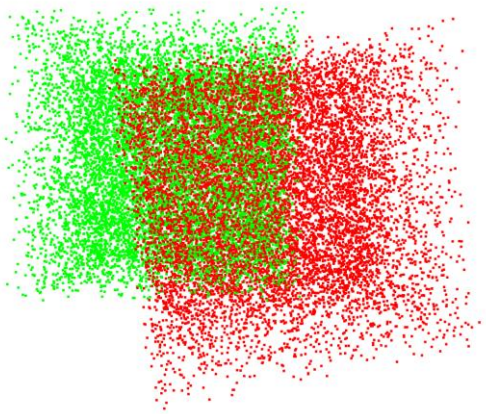


Iterative 9

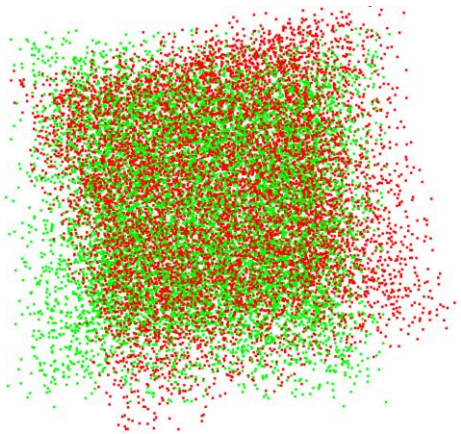
The ICP get diverge!



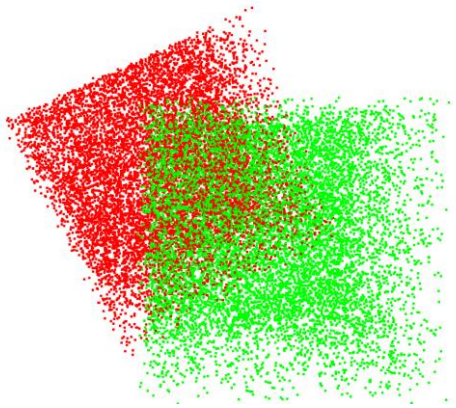
Iterative 1



Iterative 3



Iterative 5



Iterative 7

Tutorial report for AAE4011: ICP and LiDAR SLAM

Tutorial report for AAE4011: ICP and LiDAR SLAM

Weisong Wen*,[1], Shaoting Qiu[1], and Xiwei Bai*,[1]

[1] Department of Aeronautical and Aviation Engineering, The Hong Kong Polytechnic University.
welson.wen@polyu.edu.hk

*corresponding author

Keywords: Iterative Closest Point (ICP), Simultaneously localization and mapping (SLAM), LiDAR SLAM, Artificial Intelligence (AI), Unmanned autonomous systems (UAS), Robotics.

Abstract. LiDAR-based Simultaneous Localization and Mapping (SLAM) is essential for autonomous robots to navigate unknown environments by constructing maps while estimating their position. The Iterative Closest Point (ICP) algorithm plays a central role in LiDAR SLAM, aligning sequential 3D point cloud scans to refine a robot's pose and map accuracy. By iteratively minimizing distances between corresponding points, ICP computes optimal rotations and translations between scans, enabling incremental mapping. However, challenges such as computational complexity, sensitivity to initial guesses, and performance issues in dynamic or feature-poor environments limit its reliability. To address these, modern implementations integrate sensor fusion (e.g., IMUs, odometry) for robustness, probabilistic ICP variants to handle noise, and optimizations like voxel downsampling for real-time efficiency. These advancements make ICP-based LiDAR SLAM viable for applications like self-driving cars, drones, and industrial automation. For undergraduate students, understanding ICP's principles—its strengths, limitations, and integration with complementary technologies—provides foundational insight into real-world robotic navigation systems. This abstract highlights ICP's role in bridging theoretical algorithms and practical autonomy, emphasizing its balance of precision and computational feasibility in resource-constrained robotics.

2.1 Introduction

The Iterative Closest Point (ICP) algorithm is a pivotal method for aligning two 3D point clouds through iterative refinement of a rigid transformation (rotation and translation). This document formalizes its mathematical foundation, variants, and applications.

2.2 Problem Formulation

Given a source point cloud $P = \{\mathbf{p}_i\}_{i=1}^N$ and a target point cloud $Q = \{\mathbf{q}_j\}_{j=1}^M$, ICP computes a rigid transformation $T = (R, \mathbf{t})$, where $R \in \mathbb{R}^{3\times 3}$ is a rotation matrix ($R^T R = I$, $\det(R) = 1$) and $\mathbf{t} \in \mathbb{R}^3$ is a translation vector. The objective is to minimize:

$$E(R, \mathbf{t}) = \sum_{i=1}^{N} \min_j \|R\mathbf{p}_i + \mathbf{t} - \mathbf{q}_j\|^2. \tag{2}$$

2.3 Correspondence Estimation

At each iteration, correspondences are established via nearest neighbor search:

$$\mathbf{q}_i^{(k)} = \arg\min_{\mathbf{q}\in Q} \|R^{(k)}\mathbf{p}_i + \mathbf{t}^{(k)} - \mathbf{q}\|, \tag{3}$$

where $(R^{(k)}, \mathbf{t}^{(k)})$ is the current transformation estimate. Accelerated using KD-trees, this step has complexity $O(N \log M)$.

2.4 Transformation Estimation

Given correspondences $\{(\mathbf{p}_i, \mathbf{q}_i^{(k)})\}$, the optimal R and $\mathbf{t}$ minimize:

$$E(R, \mathbf{t}) = \sum_{i=1}^{N} \|R\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i^{(k)}\|^2. \tag{4}$$

2.5 Centroid Calculation

Compute centroids of matched points:

$$\boldsymbol{\mu}_p = \frac{1}{N}\sum_{i=1}^{N} \mathbf{p}_i, \quad \boldsymbol{\mu}_q = \frac{1}{N}\sum_{i=1}^{N} \mathbf{q}_i^{(k)}. \tag{5}$$

2.6 Cross-Covariance Matrix

Limitations of Doing LiDAR SLAM in Python

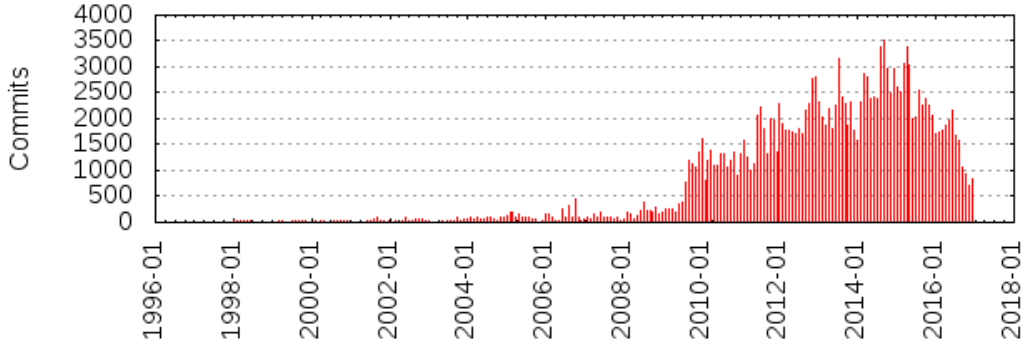
1. **Performance**:
   - **Speed**: Python is generally slower than C++ due to its interpreted nature. This can be a limitation for real-time SLAM applications that require high-speed data processing.
   - **Concurrency**: Python's Global Interpreter Lock (GIL) can be a bottleneck for multi-threaded applications, which are often needed in SLAM for handling sensor data, processing algorithms, and visualization concurrently.
2. **Libraries and Tools**:
   - While Python has libraries like open3d, numpy, and scipy for handling point clouds and numerical computations, it lacks the comprehensive SLAM-specific libraries that are available in ROS.
   - Python-based SLAM libraries may not be as mature or optimized as those available in C++.
3. **Integration**:
   - Python might not integrate as seamlessly with hardware drivers and low-level system. components as C++ does, which can be a limitation when dealing with various sensors and actuators.
4. **Community and Support**:
   - The community and support for Python-based SLAM solutions are smaller compared to ROS, which has a large community and extensive documentation.

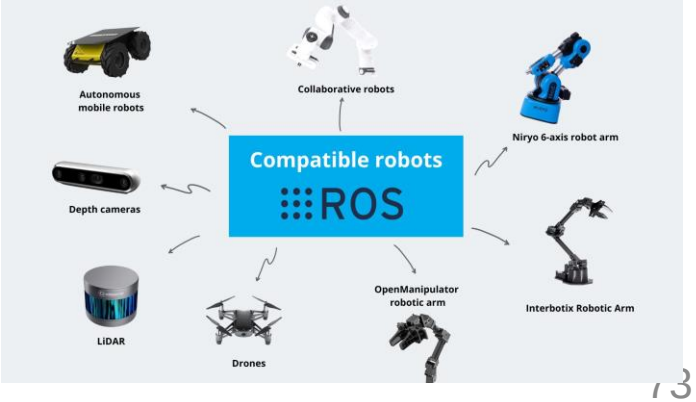
The new world for SLAM, UAS… the ROS

About ROS:

Robot Operating Systems (ROS) are frameworks that provide a collection of software libraries and tools to help developers create robot applications. ROS is designed to simplify the process of developing complex and robust robot software across a wide variety of robotic platforms. Here are some key aspects and components of ROS:

All you need is the ROS Documentation and Wiki





Key Features of ROS

1. **Modularity**: ROS is built around the concept of nodes, which are individual processes that perform computation. Nodes can be distributed across multiple machines, allowing for modular and scalable system design.
2. **Communication**: ROS provides a communication infrastructure that allows nodes to exchange information through topics, services, and actions:
   - **Topics**: Used for asynchronous communication via a publish/subscribe model. Nodes can publish messages to a topic or subscribe to receive messages from it.
   - **Services**: Provide synchronous communication, allowing nodes to send a request and receive a response.
3. **Hardware Abstraction**: ROS provides drivers and interfaces for various sensors and actuators, allowing developers to focus on higher-level functionality without worrying about hardware specifics.
4. **Tools and Libraries**: ROS includes a wide range of tools for visualization (e.g., RViz), simulation (e.g., Gazebo), data logging, and debugging. It also provides libraries for common tasks such as perception, navigation, and manipulation.
5. **Community and Ecosystem**: ROS has a large and active community, with a vast ecosystem of packages and tools contributed by developers worldwide. This makes it easier to find solutions and resources for a wide range of robotic applications.

ROS 2 is coming

ROS 2 Timeline

THE ROADMAP TO ROS 2

2014

- ROSCon 2014
- Next Generation ROS
- The Start of ROS 2 Design

2015

alpha1-alpha4 release

- C data structures, dynamic ROS Middleware implementations, initial versions of rclppNext Generation ROS

2016

alpha5-alpha8 and beta1 release

- Support for node composition
- A standard lifecycle for ROS 2 nodes
- Improved support for QoS
- ROS1Bridge

2017

beta1-beta3 and Ardent releases

- DDS_Security support (beta 1)
- Execution model in Python (beta 2)
- Distributed discovery, publish/subscribe, request/response communication
- C++ and Python 3 client libraries (rclcpp.rclpy) and message definition

2018

Bouncy and Crystal releases

- New launch system | Static remapping of topics and services | rclpy improvements | support for three RMW implementations
- Actions in C/C++, contributed packages (gazebo, navigation2, rosbag2, rqt)

2019

Dashing and Eloquent releases

- Node composition | Improve rclpy to match rclpp, support of QoS, deadline and lifespan
- Support of markup launch files (XML, YAML), ros2doctor, turtles, rqt plugins, tf tree, rtobot steering, per-node logging

2020

Foxy release

- (LTS) Release: Supported until 2023, providing stability and long-term maintenance.
- Expanded compatibility with Ubuntu 20.04 (Focal Fossa), RTOS, and embedded platforms.
- Improved ROS 2 CLI tools, RViz enhancements, better colcon build tool, and a more powerful ROS 2 launch system.
- (SROS2) tools for secure communications.

2021

Galactic release

- Better command-line interfaces and debugging capabilities.
- Continued focus on improving the performance and reliability of the ROS Middleware (RMW) interfaces.
- Quality of Service (QoS) Enhancements: More granular QoS settings for tailored communication behaviors.
- Significant updates from the ROS community, including new packages and features

2022

Humble release

- Long-Term Support (LTS): Supported until 2027.
- Security Improvements: Enhanced tools for secure communications and authentication in ROS 2 networks.
- Developer Experience: Improved usability of ROS 2 tools, making development, debugging, and deployment easier.
- Expanded support for various hardware platforms, including resource-constrained devices

2023

Iron release

- Easier access and understanding of API documentation generated from code.
- Improved debugging tools for better troubleshooting.
- New tools for managing callbacks, essential for handling asynchronous events in ROS applications.
- rclcpp Executor Updates: Optimized executors for better node execution management and reduced race conditions

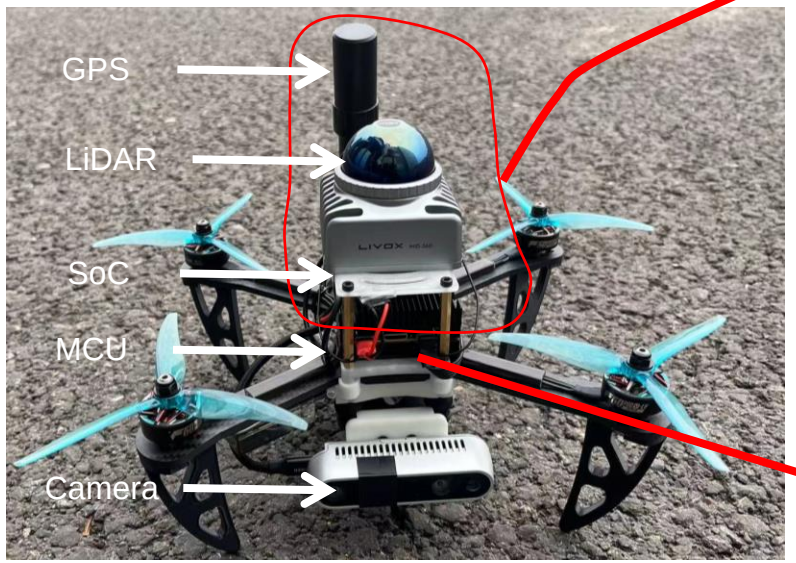
2024

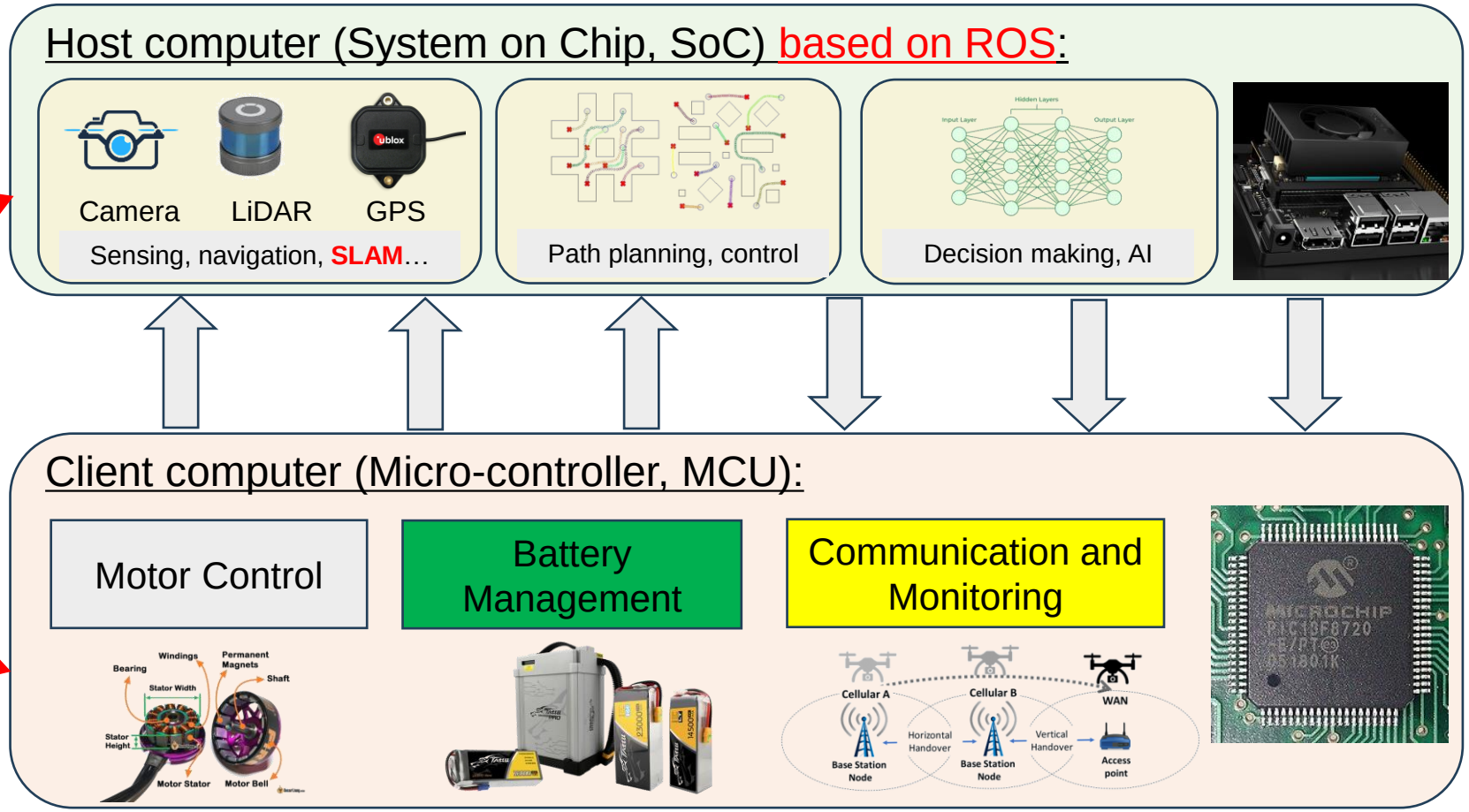
Jazzy release

- Gazebo Integration: Enhanced usability and integration with the Gazebo simulator.
- Zenoh-RMW
- Ability to record and replay service calls, better handling of ROSBag data.
- RViz2 Enhancements, and improved visualization

System of typical UAS

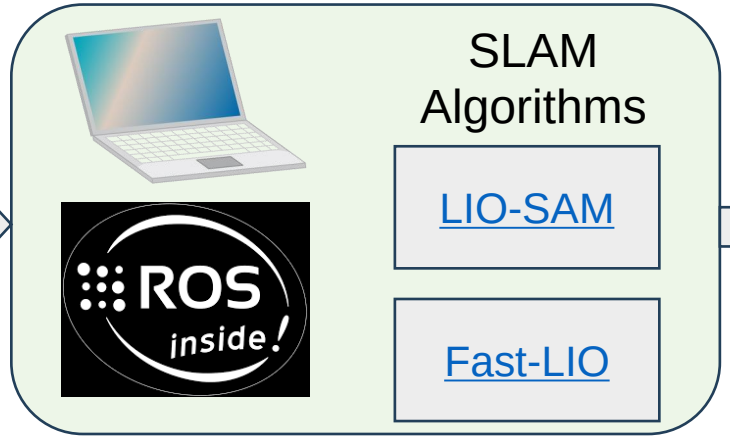
Unlike a System on Chip (SoC), which may include a full operating system and support for complex applications, an MCU is typically used for more straightforward, dedicated tasks.

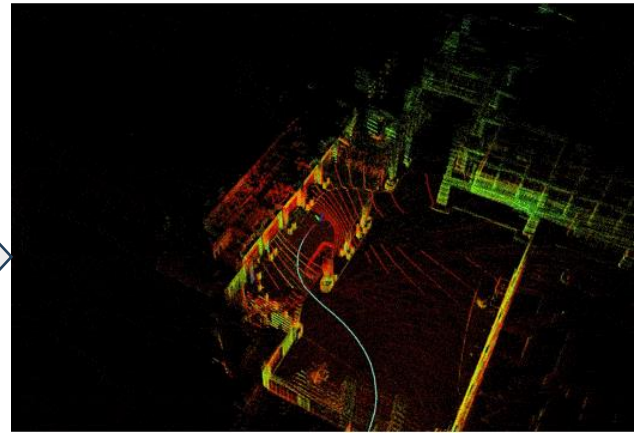




A System on Chip (SoC) for a robot is an integrated circuit that consolidates all the essential components of a computer or other electronic system onto a single chip. This can include a processor, memory, input/output ports, and other features necessary for the robot's operation. It is a computer!

In class practice: LIO-SAM based SLAM with ROS

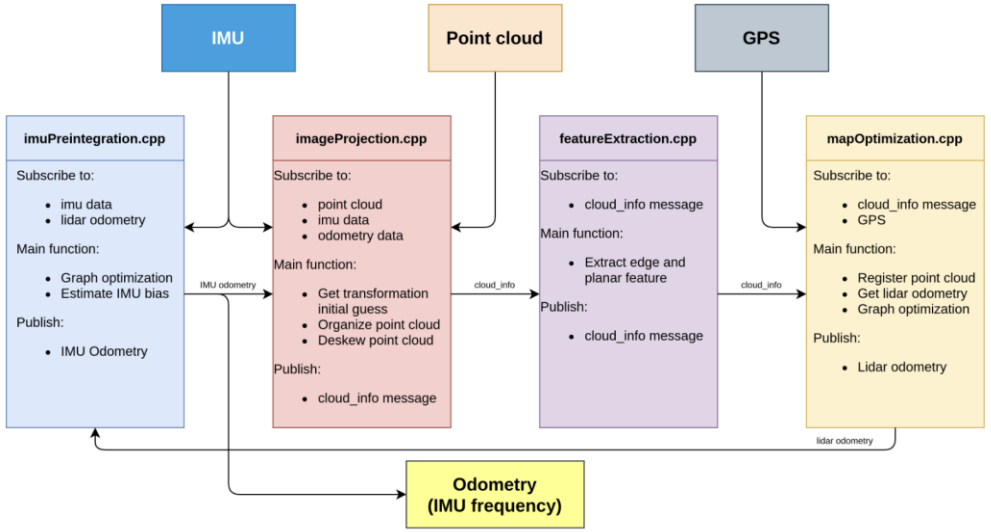




LiDAR Inertial SLAM on ROS

System architecture

LIO-SAM

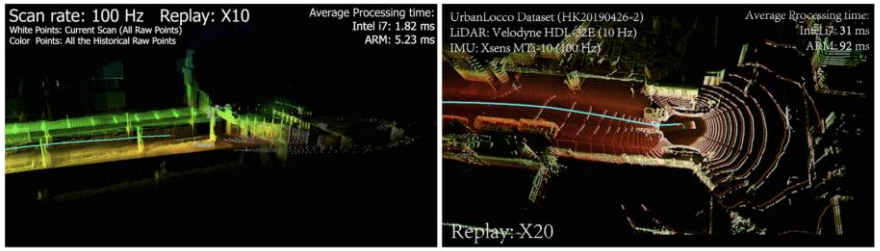


FAST-LIO

FAST-LIO (Fast LiDAR-Inertial Odometry) is a computationally efficient and robust LiDAR-inertial odometry package. It fuses LiDAR feature points with IMU data using a tightly-coupled iterated extended Kalman filter to allow robust navigation in fast-motion, noisy or cluttered environments where degeneration occurs. Our package address many key issues:

1. Fast iterated Kalman filter for odometry optimization;
2. Automaticaly initialized at most steady environments;
3. Parallel KD-Tree Search to decrease the computation;

FAST-LIO 2.0 (2021-07-05 Update)



Related video: FAST-LIO2, FAST-LIO1

In class practice: LIO-SAM based SLAM with ROS

What you need?:

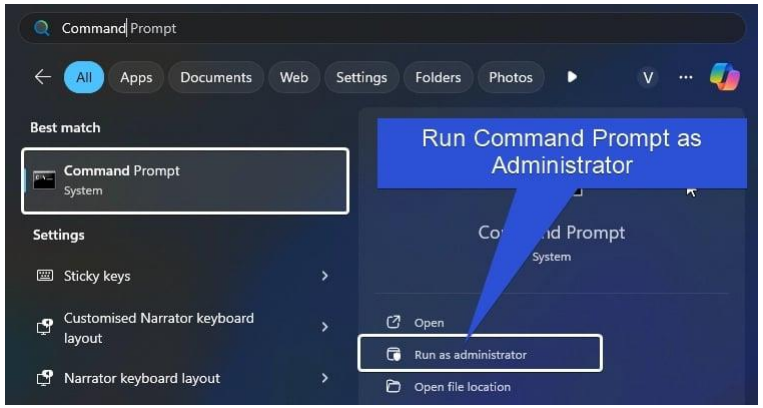
- Laptop with Windows operation system (at least Windows 11)
- 30G space at least in your laptop

Install the WSL in Windows 11 system → Install the ROS in WSL → Install the LIO-SAM in WSL → Download example dataset of 3D LiDAR → Run the LIO-SAM algorithms → Visualization of the results

Windows Subsystem for Linux (WSL) is a compatibility layer for running Linux binary executables natively on Windows 10, Windows 11, and Windows Server 2019 and later. It allows developers and system administrators to use a Linux environment directly on a Windows machine without the need for a virtual machine or dual-boot setup.

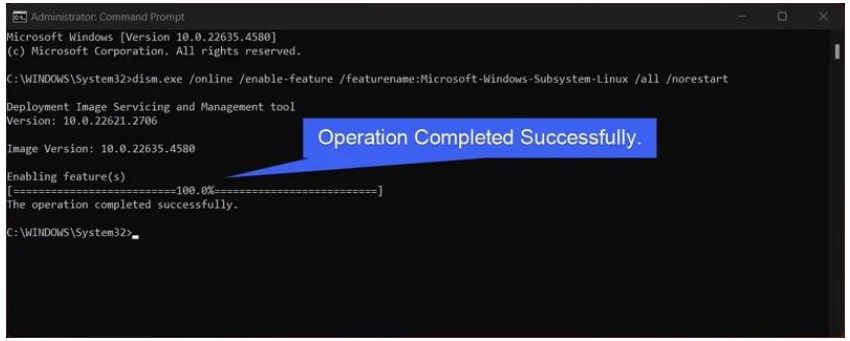
Step 1: Install WSL2

1. Open ***Command Prompt*** as Administrator



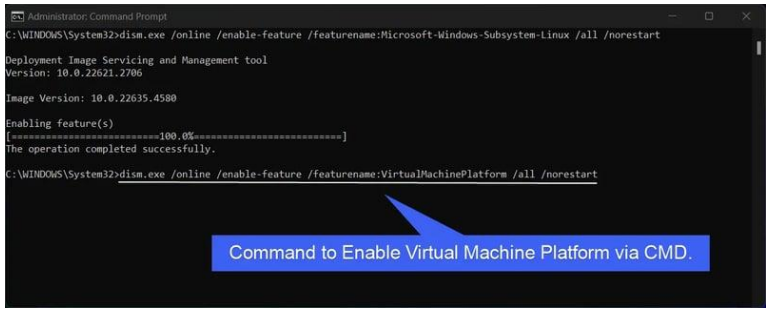
2. Enable the ***Windows Subsystem for Linux***

```
$ dism.exe /online /enable-feature
/featurename:Microsoft-Windows-Subsystem-Linux /all
/norestart
```



3. Enable the ***Virtual Machine Platform***

```
$ dism.exe /online /enable-feature
/featurename:VirtualMachinePlatform /all /norestart
```



- The **Virtual Machine Platform** is a crucial component required for running **WSL 2** on Windows 11. It provides the **underlying virtualization layer** needed to run Linux distributions in a more **efficient** and **isolated environment**.

When you get the message " **The Operation Completed Successfully** ", then WSL 2 is now enabled.

Step 1: Install WSL2

4. Check all *WSL* Distributions

```
$ wsl --list --online
```

```
PS C:\Users\print> wsl --list --online
以下是可安裝的有效分发的列表。
使用 'wsl.exe --install <Distro>' 安裝。

NAME                            FRIENDLY NAME
Ubuntu                          Ubuntu
Debian                          Debian GNU/Linux
kali-linux                      Kali Linux Rolling
Ubuntu-18.04                    Ubuntu 18.04 LTS
Ubuntu-20.04                    Ubuntu 20.04 LTS
Ubuntu-22.04                    Ubuntu 22.04 LTS
Ubuntu-24.04                    Ubuntu 24.04 LTS
OracleLinux_7_9                 Oracle Linux 7.9
OracleLinux_8_7                 Oracle Linux 8.7
OracleLinux_9_1                 Oracle Linux 9.1
openSUSE-Leap-15.6              openSUSE Leap 15.6
SUSE-Linux-Enterprise-15-SP5    SUSE Linux Enterprise 15 SP5
SUSE-Linux-Enterprise-15-SP6    SUSE Linux Enterprise 15 SP6
openSUSE-Tumbleweed             openSUSE Tumbleweed
PS C:\Users\print>
```

5. Install Certain Distributions

```
$ wsl --install -d Ubuntu-20.04
```

```
Installing: Virtual Machine Platform
Virtual Machine Platform has been installed.
Installing: Windows Subsystem for Linux
```

6. Set Your Username & Password

- Enter your username and password then click ‘Enter’.
- Then you will enter your ***Ubuntu System*** in terminal.

```
Ubuntu 24.04 LTS is already installed.
Launching Ubuntu 24.04 LTS...
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows
 username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: vik
New password:
Retype new password:
passwd: password updated successfully
Installation successful!
```

Now you are in the subsystem of Windows!

Step 1: Install Multipass (For MACOS)

1. Download and install Multipass

2. Open ***terminal*** and check available images

```
$ multipass find
```

```
andy@andys-MacBook-Air ~ % multipass find
Image                       Aliases           Version          Description
20.04                       focal             20250127         Ubuntu 20.04 LTS
22.04                       jammy             20250128         Ubuntu 22.04 LTS
24.04                       noble,lts         20250115         Ubuntu 24.04 LTS
24.10                       oracular          20250129         Ubuntu 24.10
```

3. Create ***Ubuntu20.04 VM***

```
$ multipass launch \  --name ros-vm \  --cpus 2 \  --mem 4G \  --disk 20G \  20.04
```

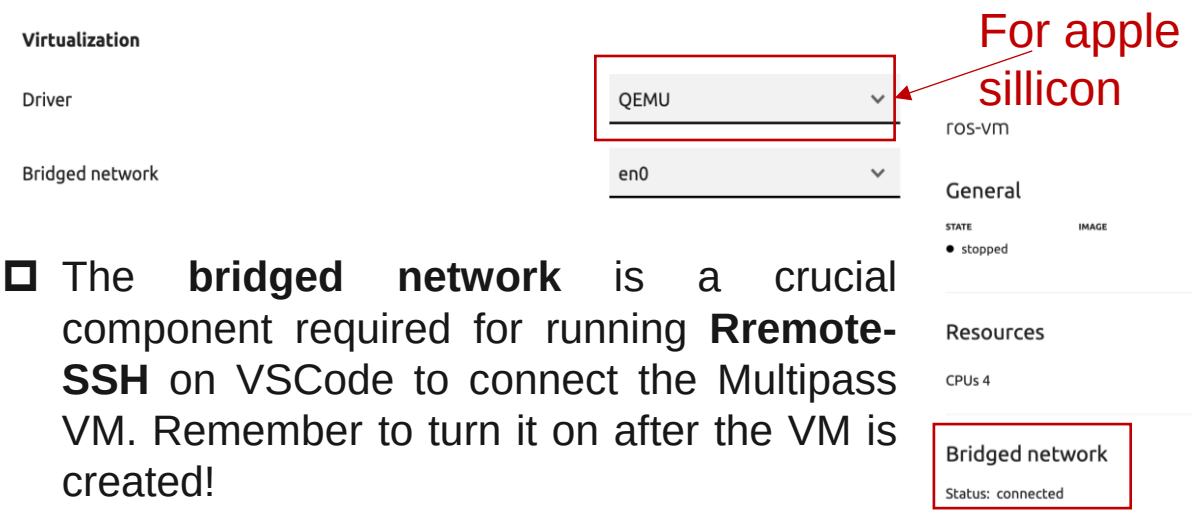
```
andy@andys-MacBook-Air ~ % multipass launch \
> --name ros-dev \
> --cpus 4 \
> --mem 8G \
> --disk 20G \
> 20.04
Warning: the "--mem" long option is deprecated in favour of "--memory". Please update any scripts, etc.
Waiting for initialization to complete |
```

NAME	STATE	CPU USAGE	MEMORY USAGE	DISK USAGE	IMAGE	PRIVATE IP
ros-dev	running		214.5MiB / 7.8GiB	1.4GiB / 19.3GiB	Ubuntu 20.04.6 ...	10.211.56.8

4. Configure Multipass

```
$ multipass networks
```

```
andy@andys-MacBook-Air ~ % multipass networks
Name    Type        Description
en0     wifi        Wi-Fi
en3     ethernet    Ethernet Adapter (en3)
en4     ethernet    Ethernet Adapter (en4)
```



- The **bridged network** is a crucial component required for running **Rremote-SSH** on VSCode to connect the Multipass VM. Remember to turn it on after the VM is created!

```
$ multipass shell ros-mv
$ sudo passwd ubuntu
```

The default username is ubuntu, configure your new password and you are good to go for the Multipass shell! Files in MACOS can be accessed by mounting the directories in your MAC!

Step 2: Install ROS 1 (MACOS/Windows)

1. Setup your ***sources.list & keys***

Setup your computer to accept software from packages.ros.org.

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

```
$ sudo apt install curl
$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

2. Installation

```
$ sudo apt update
```

```
$ sudo apt install ros-noetic-desktop-full
```

3. Environment setup

You must source this script in every bash terminal you use ROS in.

```
$ source /opt/ros/noetic/setup.bash
```

```
$ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc source ~/.bashrc
```

4. Dependencies for building packages

```
$ sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential
```

```
$ sudo apt install python3-rosdep
$ sudo rosdep init
$ rosdep update
```

https://wiki.ros.org/noetic/Installation/Ubuntu

Step 3: ROS Workspace Preparation (MACOS/Windows)

1. Introduction to ROS Package

The recommended method of working with catkin packages is using a ***catkin workspace***, but you can also build catkin packages standalone. A trivial workspace might look like this:

```
workspace_folder/          -- WORKSPACE
    src/                   -- SOURCE SPACE
        CMakeLists.txt     -- 'Toplevel' CMake file, provided by catkin
        package_1/
            CMakeLists.txt -- CMakeLists.txt file for package_1
            package.xml    -- Package manifest for package_1 ...
        package_n/
            CMakeLists.txt -- CMakeLists.txt file for package_n
            package.xml    -- Package manifest for package_n
```

2. Create a ROS Workspace

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

3. Download Some Dependencies

ROS Dependencies

```
$ sudo apt-get install -y ros-noetic-navigation
$ sudo apt-get install -y ros-noetic-robot-localization
$ sudo apt-get install -y ros-noetic-robot-state-publisher
```

gtsam (Georgia Tech Smoothing and Mapping library)

```
$ sudo add-apt-repository ppa:borglab/gtsam-release-4.0
$ sudo apt install libgtsam-dev libgtsam-unstable-dev
```

4. Download & Build LIO-SAM Package

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/TixiaoShan/LIO-SAM.git
$ cd LIO-SAM/include
$ sudo vim utility.h
```

#include<opencv/cv.h> → #include <opencv2/opencv.hpp>
#include <opencv2/imgproc.hpp>

Place these includes in the end of included items

https://wiki.ros.org/noetic/Installation/Ubuntu

Step 3: ROS Workspace Preparation (MACOS/Windows)

1. Swap space problem

```
C++: fatal error: Killed signal terminated program cc1plus
compilation terminated.
```

Input scripts below into the VM shell

```
# Create space path
$ sudo mkdir -p /var/cache/swap/
# configure space size
# bs=64M is the block size, count=64 is the block quantity, # Therefore, the swap space is bs*count=4096MB=4GB
$ sudo dd if=/dev/zero of=/var/cache/swap/swap0 bs=64M count=64
# setting the index privilege
$ sudo chmod 0600 /var/cache/swap/swap0
# create SWAP files
$ sudo mkswap /var/cache/swap/swap0
# activate SWAP files
$ sudo swapon /var/cache/swap/swap0
# check if the swap info is correct
$ sudo swapon -s
```

2. PCL problem

```
/usr/include/pcl-1.10/pcl/point_types.h:678:1: error: 'divscalar' is not a member of 'pcl::traits'
  678 | POINT_CLOUD_REGISTER_POINT_STRUCT(pcl::PointSurfel,
      | ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/usr/include/pcl-1.10/pcl/point_types.h: In function 'const pcl::GASDSignature984& pcl::common::operator/=(pcl::GASDSignature984&, const float&)':
/usr/include/pcl-1.10/pcl/point_types.h:649:1: error: 'divscalar' is not a member of 'pcl::traits'
  649 | POINT_CLOUD_REGISTER_POINT_STRUCT(pcl::GASDSignature984,
      | ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/usr/include/pcl-1.10/pcl/point_types.h:678:1: error: 'divscalar' is not a member of 'pcl::traits'
  678 | POINT_CLOUD_REGISTER_POINT_STRUCT(pcl::PointSurfel,
      | ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/usr/include/pcl-1.10/pcl/point_types.h:678:1: error: 'divscalar' is not a member of 'pcl::traits'
  678 | POINT_CLOUD_REGISTER_POINT_STRUCT(pcl::PointSurfel,
      | ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Edit the CMakeList.txt in LIO-SAM

```
# set(CMAKE_CXX_FLAGS "-std=c++11")
set(CMAKE_CXX_FLAGS "-std=c++14")
```

```
$ cd ~/catkin_ws
$ catkin_make
```

Compilation Successful!

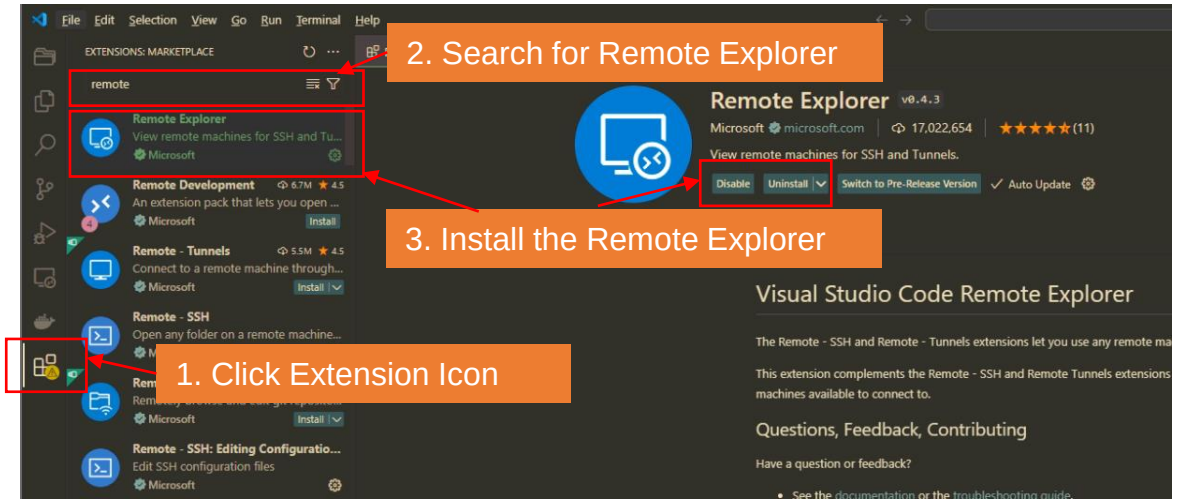
```
                 from /home/mint/catkin_ws/src/LIO-SAM/include/utility.h:27,
                 from /home/mint/catkin_ws/src/LIO-SAM/src/mapOptmization.cpp:1:
/usr/include/pcl-1.8/pcl/sample_consensus/model_types.h: In function 'void __static_initialization_and_destruction_0(int, int)':
/usr/include/pcl-1.8/pcl/sample_consensus/model_types.h:99:3: warning: 'pcl::SAC_SAMPLE_SIZE' is deprecated: This map is deprecated and is
 kept only to prevent breaking existing user code. Starting from PCL 1.8.0 model sample size is a protected member of the SampleConsensusM
odel class [-Wdeprecated-declarations]
   SAC_SAMPLE_SIZE (sample_size_pairs, sample_size_pairs + sizeof (sample_size_pairs) / sizeof (SampleSizeModel));
   ^~~~~~~~~~~~~~~
/usr/include/pcl-1.8/pcl/sample_consensus/model_types.h:99:3: note: declared here
[ 95%] Linking CXX executable /home/mint/catkin_ws/devel/lib/lio_sam/lio_sam_imuPreintegration
[ 95%] Built target lio_sam_imuPreintegration
[100%] Linking CXX executable /home/mint/catkin_ws/devel/lib/lio_sam/lio_sam_mapOptmization
[100%] Built target lio_sam_mapOptmization
```

https://blog.csdn.net/weixin_44796670/article/details/121234446
https://blog.csdn.net/qq_32761549/article/details/126640468

Step 4: Connect to IDE (VS Code_Win)

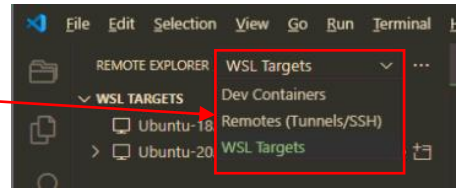
Connect Your Ubuntu to VSCode

Download VScode at:
https://code.visualstudio.com/download

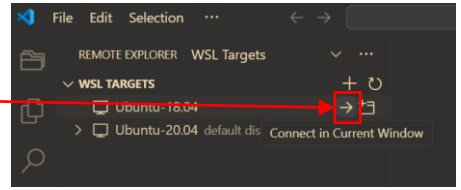


Then you can see the ***Remote Explorer*** Icon on the left bar, if not, you can add it by right click at left bar, then chose ***Remote Explorer.***

4. In the ***Remote Explorer*** extension, chose WSL Targets, then you can see all the subsystems in your environment.

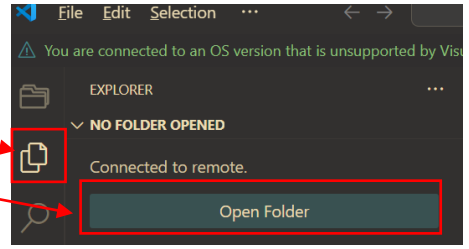


5. Chose the ***Ubuntu-20.04*** target, connect in current window by click the right arrow.

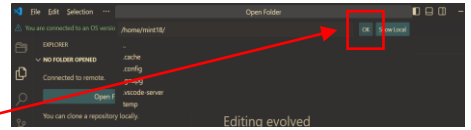


6. Click ***Allow*** to continue.

7. Click ***Explore*** icon on the left bar, select ***Open Folder***.



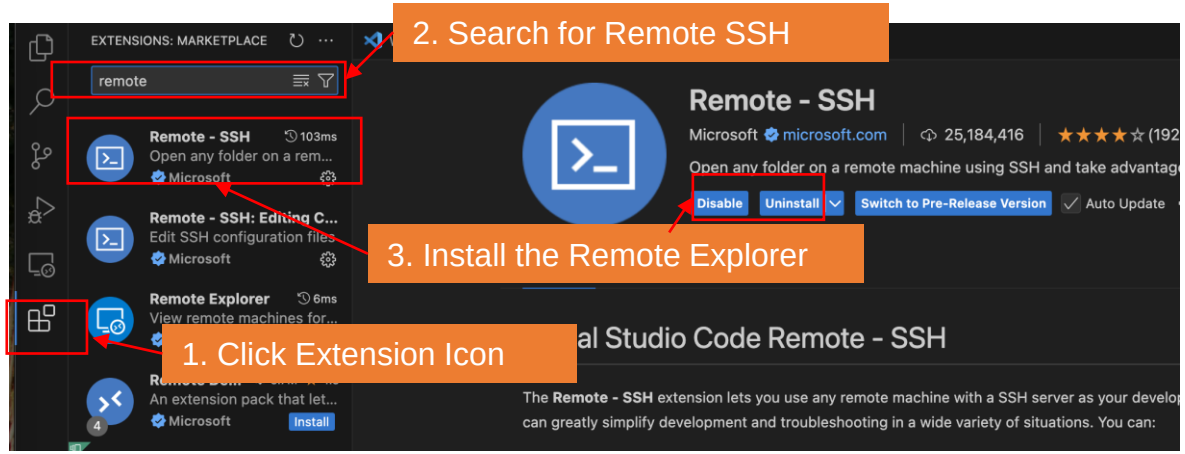
8. Click ***OK***, then the root folder in the Ubuntu18.04 subsystem is connected to VS Code, you can easily modify files in VS Code.



Step 4: Connect to IDE (VS Code_MACOS)

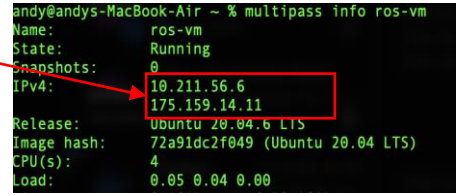
Connect Your Ubuntu to VSCode

Download VScode at:
https://code.visualstudio.com/download

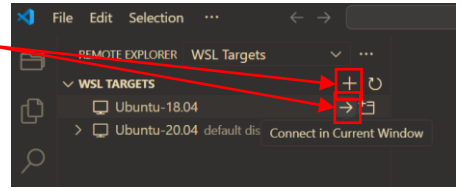


Then you can see the ***Remote Explorer*** Icon on the left bar, if not, you can add it by right click at left bar, then chose ***Remote Explorer.***

4. Check Multipass VM info using
$ multipass info ros-vm

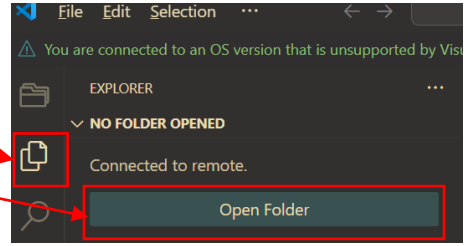


5. Add new host according to your VM IP in the prompt, connect in current window by click the right arrow.

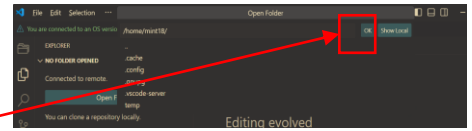


6. Adding host command is as below:
$ ssh ubuntu@<VM _IP> -p 22

7. Click ***Explore*** icon on the left bar, select ***Open Folder***.



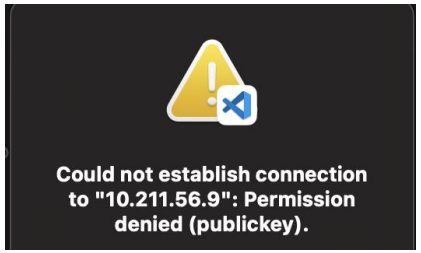
8. Click ***OK***, then the root folder in the Ubuntu20.04 subsystem is connected to VS Code, you can easily modify files in VS Code.



You can also use other alternatives to develop your project like parallel desktop or VMware Fusion, whatever fits you the best!

Step 4: Connect to IDE (VS Code_MACOS)

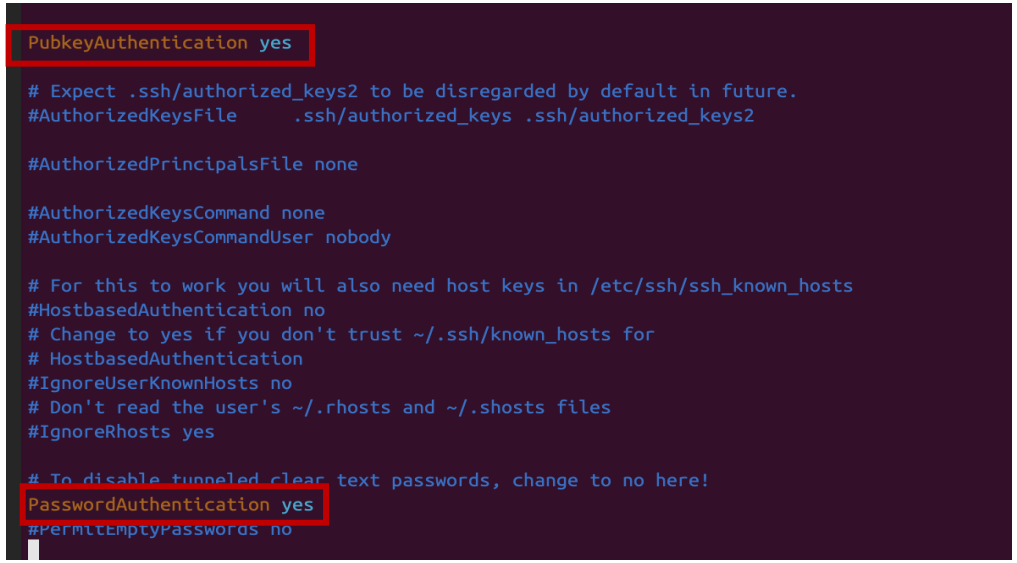
If connection permission denied…




```
# install ssh service in Multipass VM
$ sudo apt update
$ sudo apt install -y openssh-server
# reboot ssh and set up auto bootup
$ sudo systemctl enable ssh
$ sudo systemctl start ssh
```

Check in /etc/ssh/sshd_config that these two lines are not referenced. Then restart ssh by executing:

```
sudo systemctl restart ssh
```



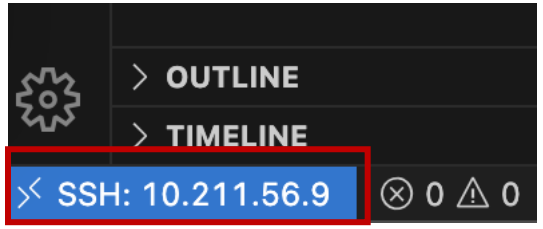
Next, In your IDE terminal:

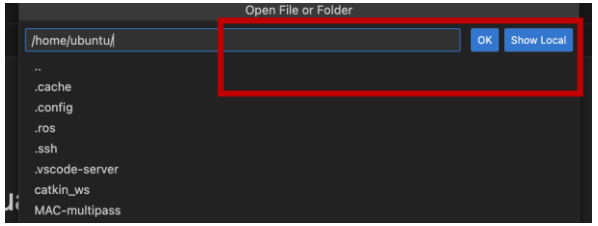
```
# generate and save the publickey
(id_ed25519.pub) and privatekey(id_ed25519)
$ ssh-keygen -t ed25519 -C "my_multipass_key"

# check publickey in code terminal
$ cat ~/.ssh/id_ed25519.pub
```

Then, In your multipass VM:

```
$ mkdir -p ~/.ssh
$ chmod 700 ~/.ssh
# Copy paste the public key line into this
file and save
$ nano ~/.ssh/authorized_keys

# Authenticate this publickey
$ chmod 600 ~/.ssh/authorized_keys
```

You are connected to your VM if you see these!





Step 5: RUN LIO-SAM!

1. Download Example Data

Example Data Link

In WSL, ubuntu's file system is connected to windows, you can find your download file at ***/mnt/***

Therefore, you can find resources under windows under the ***/mnt/*** path and use that path to access them.

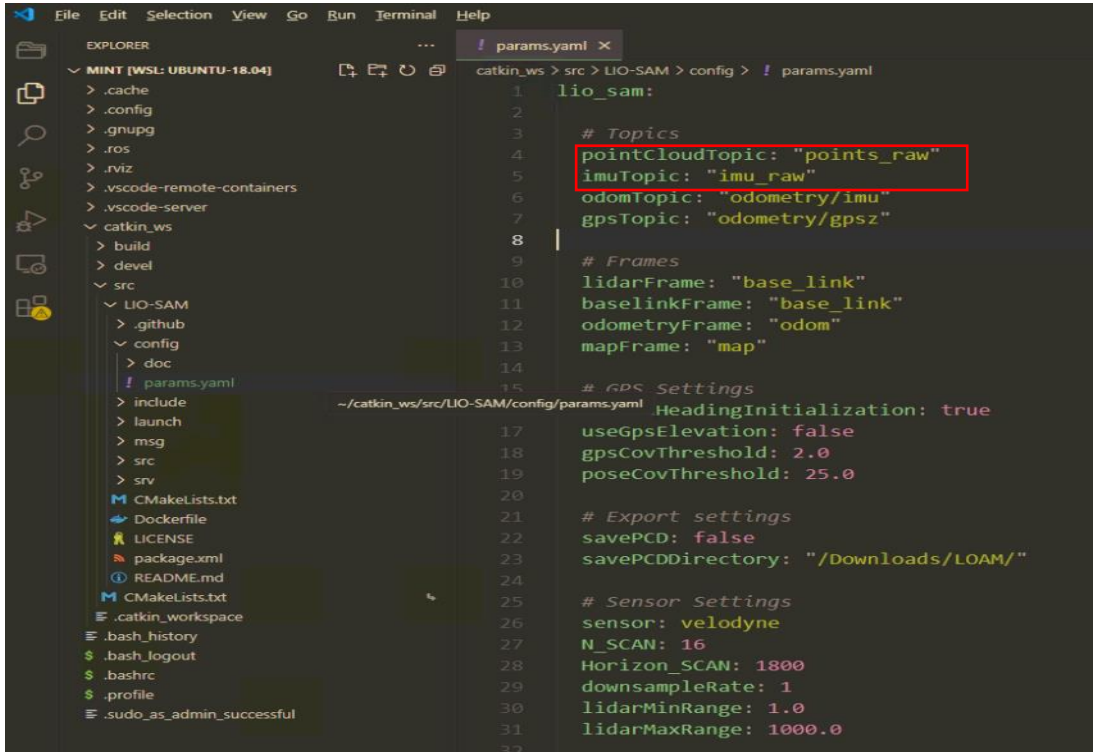
```
mint@DESKTOP-O5ET4G2:~/catkin_ws$ cd /mnt/c/Users/mint/Documents/dataset/LIOSAM/
mint@DESKTOP-O5ET4G2:/mnt/c/Users/mint/Documents/dataset/LIOSAM$ ls
campus_small_dataset.bag
mint@DESKTOP-O5ET4G2:/mnt/c/Users/mint/Documents/dataset/LIOSAM$
```

2. Check ROS Bag Topic

```
mint@DESKTOP-O5ET4G2:~$ cd /mnt/c/Users/mint/Documents/dataset/LIOSAM/
mint@DESKTOP-O5ET4G2:/mnt/c/Users/mint/Documents/dataset/LIOSAM$ ls
campus_small_dataset.bag
mint@DESKTOP-O5ET4G2:/mnt/c/Users/mint/Documents/dataset/LIOSAM$ rosbag info campus_small_dataset.bag
path:        campus_small_dataset.bag
version:     2.0
duration:    6:47s (407s)
start:       Nov 12 2019 22:00:36.64 (1573567236.64)
end:         Nov 12 2019 22:07:24.42 (1573567644.42)
size:        1.6 GB
messages:    209994
compression: none [1877/1877 chunks]
types:       sensor_msgs/Imu         [6a62c6daae103f4ff57a132d6f95cec2]
             sensor_msgs/NavSatFix   [2d3a8cd499b9b4a0249fb98fd05cfa48]
             sensor_msgs/PointCloud2 [1158d486dd51d683ce2f1be655c3c181]
topics:      /gps/fix         2039 msgs    : sensor_msgs/NavSatFix
             /imu_correct   203912 msgs    : sensor_msgs/Imu
             /points_raw      4043 msgs    : sensor_msgs/PointCloud2
```

3. Modify LIO-SAM Config File

Open the config file in the LIO SAM package in VS Code. Change the ***pointCloudTopic*** and ***imuTopic*** to corresponding topics in rosbag.



Step 5: RUN LIO-SAM!

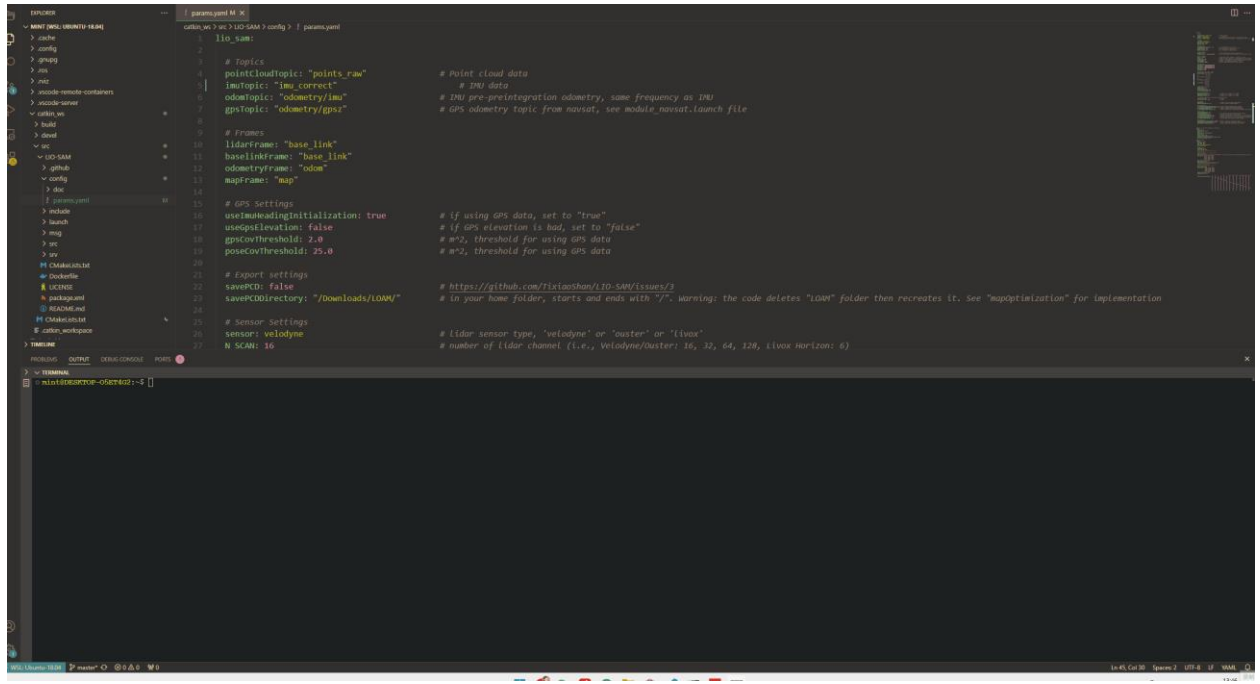
4. RUN!

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ roslaunch lio_sam run.launch
```

Split a new terminal

```
$ cd PATH_TO_BAG
$ rosbag play BAG_NAME.bag
```

Now you can see the result in ***rviz***



In class task: develop a ROS SLAM project from scratch

What you need?:

- Laptop with Windows operation system (at least Windows 11)
- 30G space at least in your laptop
- Basic understanding on the C++
- ChatGPT tools: such as DeepSeek, GPT-4o

Task 1: Given a rosbag file, read and save the 3D point clouds to PCD files:

- Step 1: Ask the Deepseek "*ros node C++ code, subscribe the 3D point cloud, save the point clouds to PCD file continuously, name the pcd file from 1*"
- Step 2: follow the procedures from Deepseek to setup the project.
- Step 3: run the code and save the PCD files.

cpp Copy

```cpp
#include <ros/ros.h>
#include <sensor_msgs/PointCloud2.h>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl_conversions/pcl_conversions.h>
#include <string>
#include <boost/filesystem.hpp>

class PointCloudSaver
{
public:
    PointCloudSaver() : nh_("~"), save_count_(0)
    {
        // Initialize parameters
        nh_.param<std::string>("save_directory", save_directory_, "~/pointclouds");
        nh_.param<std::string>("file_prefix", file_prefix_, "cloud_");
        nh_.param<std::string>("input_topic", input_topic_, "/velodyne_points");

        // Create directory if it doesn't exist
        boost::filesystem::path dir(save_directory_);
        if(!boost::filesystem::exists(dir)) {
            boost::filesystem::create_directories(dir);
            ROS_INFO("Created directory: %s", save_directory_.c_str());
        }
```

The Deepseek is more powerful than you can imagine!

In class task: develop a ROS SLAM project from scratch

1. First, create a ROS package in your workspace:
catkin_create_pkg lidar_subscriber roscpp sensor_msgs pcl_ros pcl_conversions

bash Copy

```bash
catkin_create_pkg lidar_subscriber roscpp sensor_msgs pcl_ros pcl_conversions
```

2. Create src/lidar_subscriber_node.cpp

```
class LidarSubscriber
{
public:
    LidarSubscriber() : nh_("~")
    {
        // Subscribe to the point cloud topic
        std::string topic_name = "/velodyne_points";  // Default topic name
        nh_.param<std::string>("point_cloud_topic", topic_name, "/velodyne_points");
```

In class task: develop a ROS SLAM project from scratch

3. Update `package.xml` (ensure these dependencies are present):

```xml
<build_depend>pcl_conversions</build_depend>
<build_depend>pcl_ros</build_depend>
<build_depend>roscpp</build_depend>
<build_depend>sensor_msgs</build_depend>

<exec_depend>pcl_conversions</exec_depend>
<exec_depend>pcl_ros</exec_depend>
<exec_depend>roscpp</exec_depend>
<exec_depend>sensor_msgs</exec_depend>
```

4. Update `CMakeLists.txt`:

```cmake
cmake_minimum_required(VERSION 3.0.2)
project(lidar_subscriber)

find_package(catkin REQUIRED COMPONENTS
  pcl_ros
  pcl_conversions
  roscpp
  sensor_msgs
)
```

5. Build and run the node:

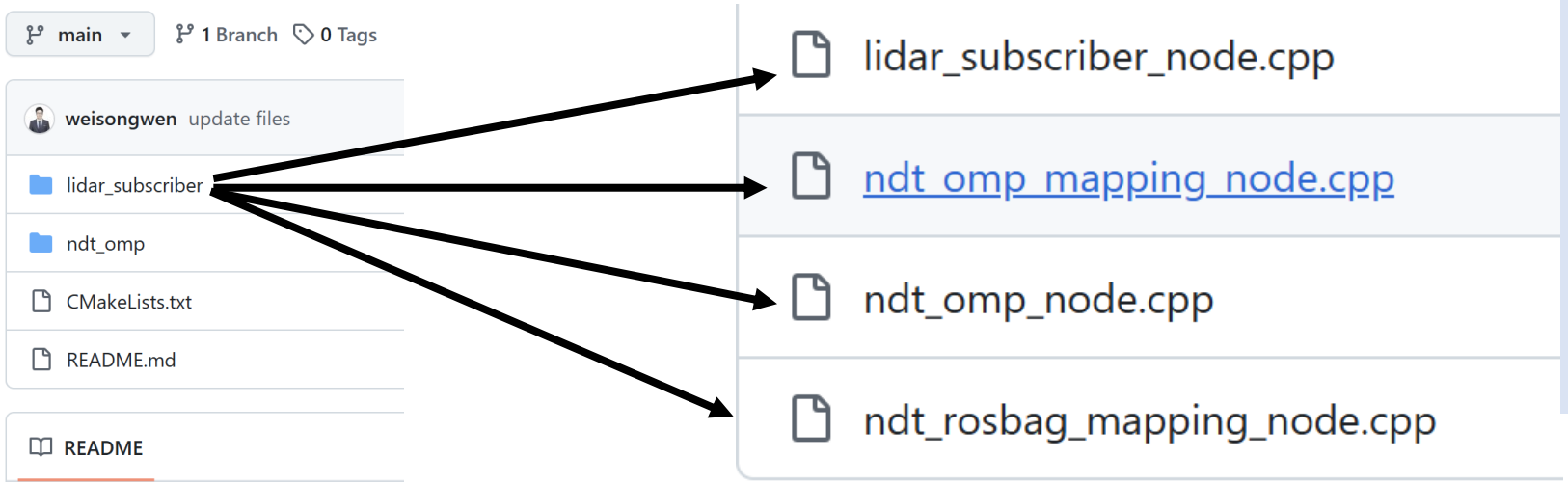
```bash
catkin build lidar_subscriber
source devel/setup.bash
rosrun lidar_subscriber lidar_subscriber_node
```

Key features:

- Subscribes to PointCloud2 messages (common format for LiDAR data)
- Converts ROS message to PCL point cloud for easier processing
- Uses parameter server for topic name configuration
- Handles XYZI points (x, y, z coordinates + intensity)
- Logs first received message confirmation

The example package can be found at designed for this course: https://github.com/weisongwen/ToySLAM

ToySLAM package



Brief introduction for ToySLAM package:

- File 3: similar to the File 2, but also plot the trajectory in the Rviz of ROS.
- File 4: Directly read the 3D point clouds from the rosbag file and perform the ICP continuously.

Brief introduction for ToySLAM package:

- 4 .cpp files associated with four different examples with certain functions.
- 4 files are associated with 4 different ros node
- All the code are generated by the AI with some slight adjustments.

Brief introduction for ToySLAM package:

- File 1: subscribe the 3D LiDAR point clouds from the rosbag file and save the point clouds to the PCD files.
- File 2: read the PCD file continuously and perform the ICP for the continuous PCD files. Output the 4x4 transformation matrix.

Questions to be asked to Deepseek for File 3

ros node C++ code, read point clouds from a rosbag, perform NDT based on ndt_omp package at https://github.com/koide3/ndt_omp to do point cloud registration and output the transformation to trajectory in RVIZ, and also show the accumulated map in Rviz

<u>Try to get the code for File 1, 3 and 4 yourselves below are some keywords for your reference:</u>

- Generate the C++ code based ros node
- Subscribe or 3D point clouds from rosbag or read the PCD files in a given folder
- Output the transformation or trajectory in the RVIZ

Tutorial report for AAE4011: NDT-Based LiDAR SLAM-Theory and Methodology

Tutorial report for AAE4011: NDT-Based LiDAR SLAM-Theory and Methodology

Weisong Wen[*,1], Shaoting Qiu[1], and Xiwei Bai[*,1]

[1] Department of Aeronautical and Aviation Engineering, The Hong Kong Polytechnic University.
welson.wen@polyu.edu.hk

[*]corresponding author

Keywords: Normal Distributions Transform (NDT), Iterative Closest Point (ICP), Simultaneously localization and mapping (SLAM), LiDAR SLAM, Artificial Intelligence (AI), Unmanned autonomous systems (UAS), Robotics.

Abstract. This paper presents a comprehensive analysis of Normal Distributions Transform (NDT)-based LiDAR SLAM, a robust approach for simultaneous localization and mapping in robotic systems. Unlike traditional Iterative Closest Point (ICP) methods, NDT employs statistical representations of point clouds to achieve superior computational efficiency and robustness in complex environments. The document details the mathematical foundations of NDT, its integration into SLAM pipelines, and emerging synergies with artificial intelligence. Key topics include probabilistic surface modeling, optimization strategies, and comparative advantages over point-to-point registration methods. Applications in autonomous vehicles, industrial robotics, and planetary exploration are discussed, along with challenges and future directions.

1 Introduction

1.1 SLAM in Robotics

Simultaneous Localization and Mapping (SLAM) forms the perceptual backbone of autonomous systems, enabling operation in unknown environments through concurrent map construction and ego-motion estimation. LiDAR sensors have become pivotal for SLAM due to their high angular resolution and immunity to lighting conditions.

2 Methodology and Theory

2.1 NDT Registration Framework

2.1.1 Voxelization Process

The target point cloud $Q = \{q_j\}_{j=1}^{M}$ is discretized into voxels with resolution δ. For each occupied voxel c:

$$\mu_c = \frac{1}{|c|} \sum_{q_j \in c} q_j \tag{2}$$

$$\Sigma_c = \frac{1}{|c| - 1} \sum_{q_j \in c} (q_j - \mu_c)(q_j - \mu_c)^T \tag{3}$$

2.1.2 Transformation Model

The source cloud $P = \{p_i\}_{i=1}^{N}$ is transformed via:

$$T(p_i; \theta) = Rp_i + t, \quad \theta = (t_x, t_y, t_z, \phi, \psi, \omega) \tag{4}$$

where $R \in SO(3)$ parametrized by Euler angles (ϕ, ψ, ω).

2.1.3 Optimization Objective

Maximize the log-likelihood of transformed points under the NDT distribution:

$$\theta^* = \arg\max_{\theta} \sum_{i=1}^{N} \log \mathcal{P}(T(p_i; \theta)) \tag{5}$$

2.2 Newton Optimization

The objective function is maximized using Newton's method with Hessian modification:

$$H = J^T \Lambda J + \lambda I \tag{6}$$

$$\Delta\theta = H^{-1} J^T \Lambda r \tag{7}$$

Conclusions and Q&A

Quick review

- Brief introduction and application of Python
- Setting up the environment
- Basic syntax and data types
- Control structures
- Functions
- Data structures
- Modules and libraries
- Error handling

Resources for further learning

- Class Python for Everybody Specialization from Prof. Charles Russell Severance at University of Michigan
- Martelli A, Ravenscroft A, Ascher D. Python cookbook[M]. " O'Reilly Media, Inc.", 2005.
- Python Documentation: https://docs.python.org/3/
- Python Tutorial: https://www.w3schools.com/python/default.asp
- GitHub projects built by Python

Error Handling

Exceptions

- Exceptions **occur during execution**, and can not be detected by basic syntax checking.
- Exceptions can cause the program stop if not handled.
- Exceptions can be handled using try and except blocks.

```
10 * (1/0)
#OUTPUT:
# Traceback (most recent call last):
#  File "<python-input-5>", line 1, in <module>
#    10 * (1/0)
#          ~^~
#ZeroDivisionError: division by zero

#Exception handling with try, except
#Try: This block will test the excepted error to
occur
#Except:  Here you can handle the error
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops!  That was no valid
number.  Try again...")
```

```
#Exception with finally: Finally block always gets
executed either exception is generated or not
def divide(x, y):
    try:
        result = x // y
        print("Yeah ! Your answer is :", result)
    except ZeroDivisionError:
        print("Sorry ! You are dividing by zero ")
    finally:
        print('This is the end of the function')

divide(3, 2)
#OUTPUT:
#Yeah ! Your answer is : 1
#This is the end of the function

divide(3, 0)
# Sorry ! You are dividing by zero
#This is the end of the function
```

Common Errors

- **Runtime Errors/Exceptions:** division by zero, name errors, type errors, index errors, key errors.
- **Syntax Errors** are common because of incorrect codes, such as missing parentheses, missing colons, and incorrect indentation.
- **Logical Errors**, off-by-one errors, wrong variable, and incorrect use of operators.

```
# Exception: index error
list1 = [1, 2, 3]
print(list1[3])
#OUTPUT:
#Traceback (most recent call last):
#  File "<python-input-7>", line 1, in <module>
#    print(list1[3])
#          ~~~~~^^^
#IndexError: list index out of range
```

```
# Logic Error: incorrect use of operators
if list1[0] = 2:
#OUTPUT:
# File "<python-input-17>", line 1
#    if list1[0] = 2:
#       ^^^^^^^^
# SyntaxError: cannot assign to subscript here.
Maybe you meant '==' instead of '='?
```

```
# Syntax Error: type error
test = list1[0] + 'int'
#OUTPUT:
#Traceback (most recent call last):
#  File "<python-input-15>", line 1, in <module>
#    list1[0]+'int'
#    ~~~~~~~~^~~~~~
#TypeError: unsupported operand type(s) for +:
'int' and 'str'
```

```
# Logic Error: off-by-one error
for i in range(5, 2):
    print(i)
#OUTPUT:
# Nothing
```

Thank you for your attention.

Dr Weisong Wen

If you have any questions or inquiries, please feel free to contact me.

Personal Page: https://polyu-taslab.github.io/team/
Email: welson.wen@polyu.edu.hk

Tools for learning AI

Development environment (computer)

- *Python is an interpreted, object-oriented, high-level programming language with dynamic data types.*

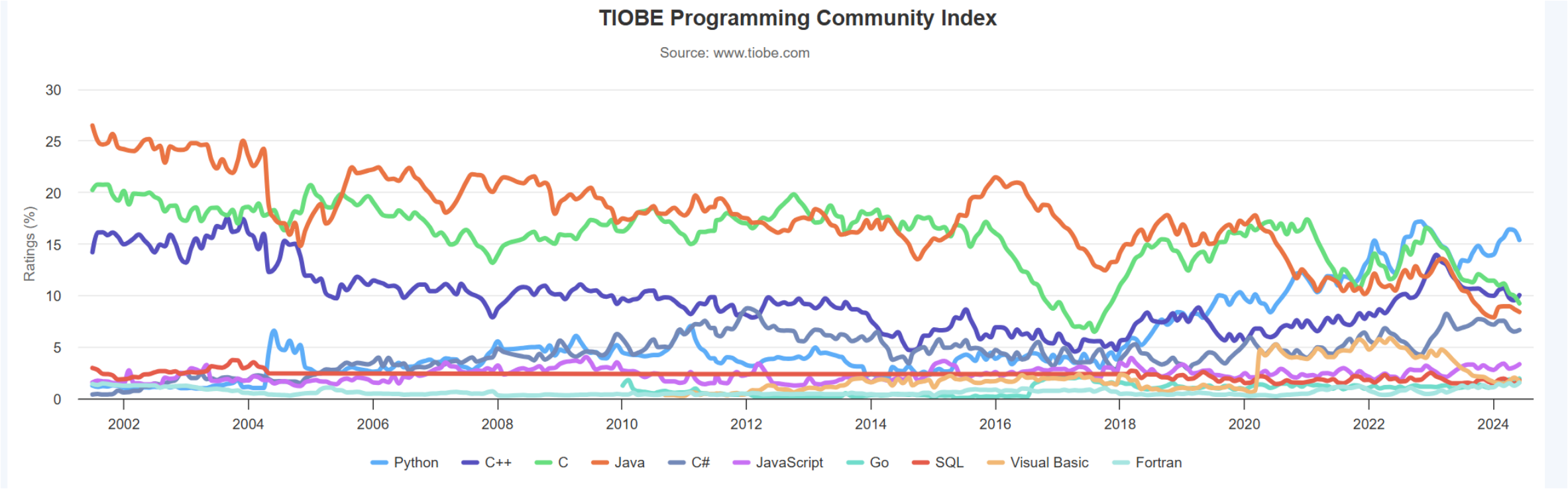
- *Open-source package and environment management system that runs on Windows, macOS, and Linux. Install, run, and update packages and their dependencies.*

- *Free software, open standards, and web services for interactive computing across all programming languages.*

Python Index Top!

TIOBE https://www.tiobe.com/tiobe-index/

Jun 2024	Jun 2023	Change	Programming Language
1	1		Python
2	3	^	C++
3	2	v	C
4	4		Java
5	5		C#



Anaconda Distribution

Millions of developers and data scientists research, innovate, and discover using Anaconda Distribution.

Repository
Over 8,000 open-source data science and machine learning packages, Anaconda-built and compiled for all major operating systems and architectures.

Learn More >

Using the Jupyter Notebook

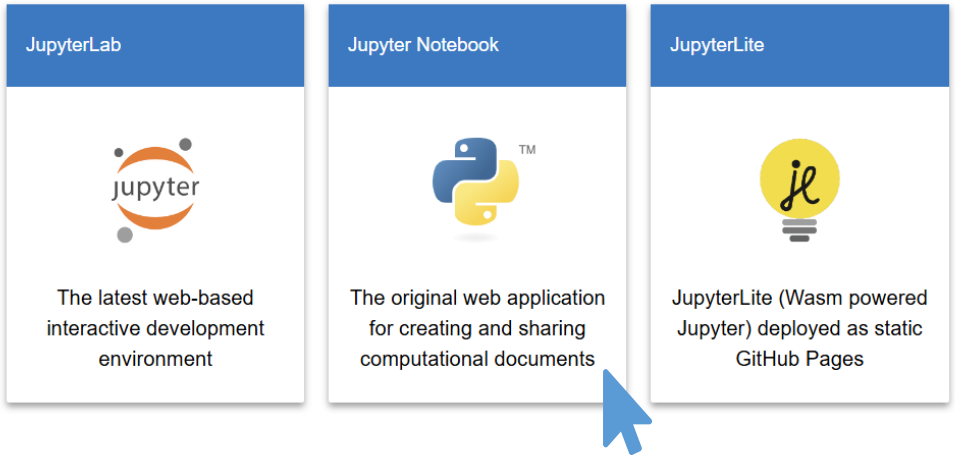
Jupyter Notebook: The Classic Notebook Interface

The Jupyter Notebook is the original web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience.

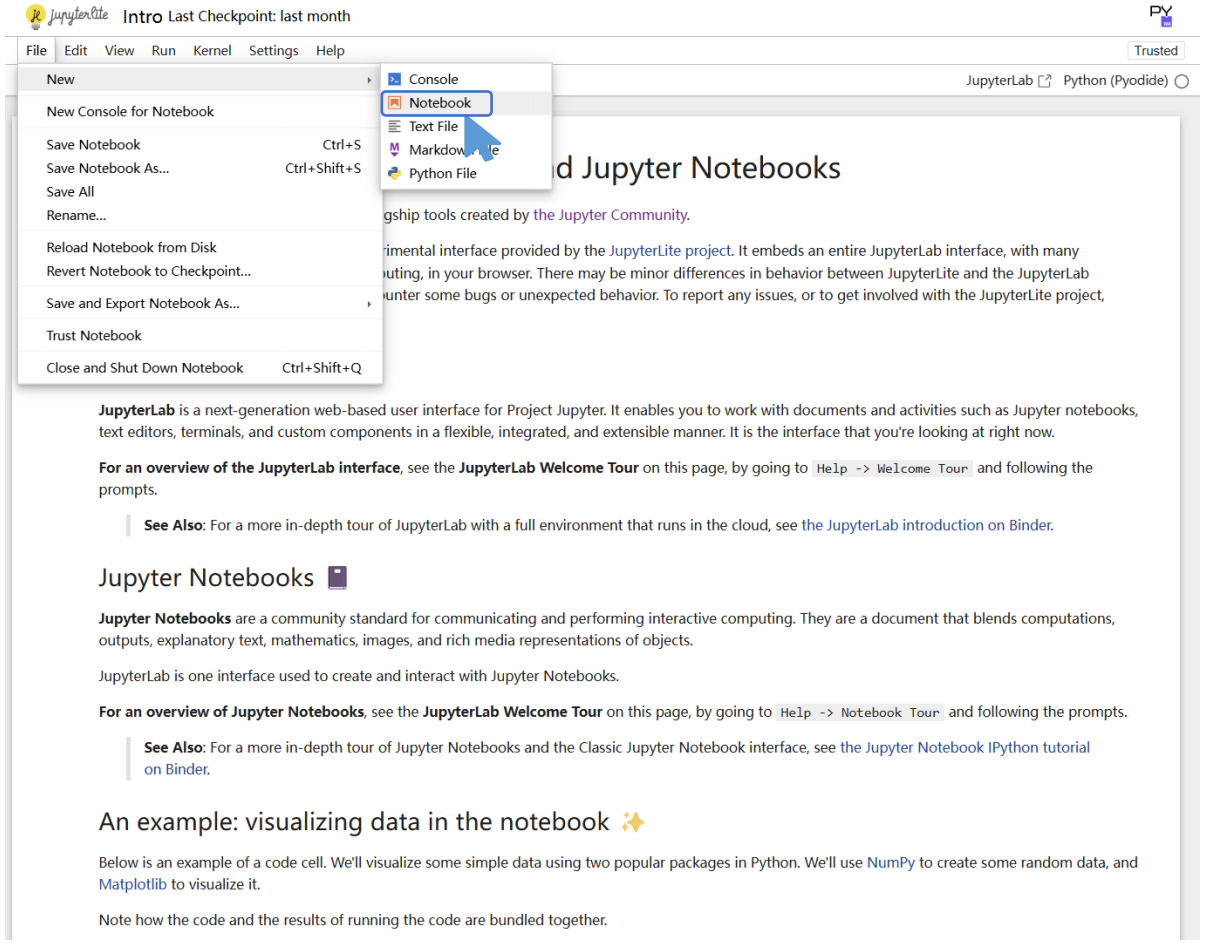
Free software, open standards, and web services for interactive computing across all programming languages

Jupyter notebook

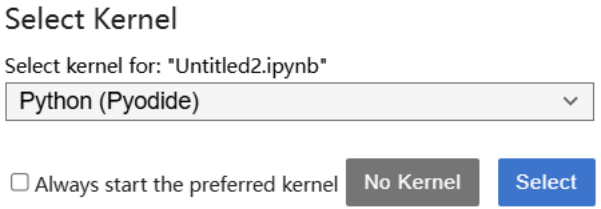
1. https://jupyter.org



2. New file Notebook (.ipynb)



3. Select python kernel

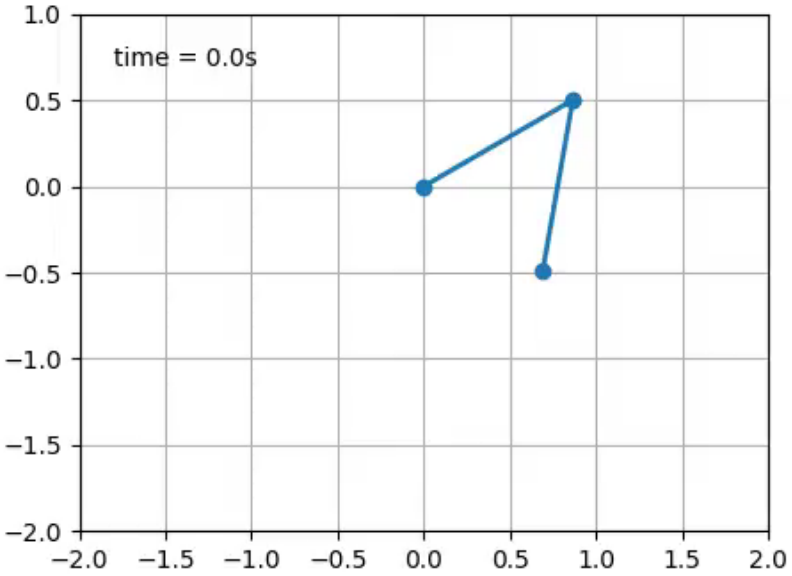


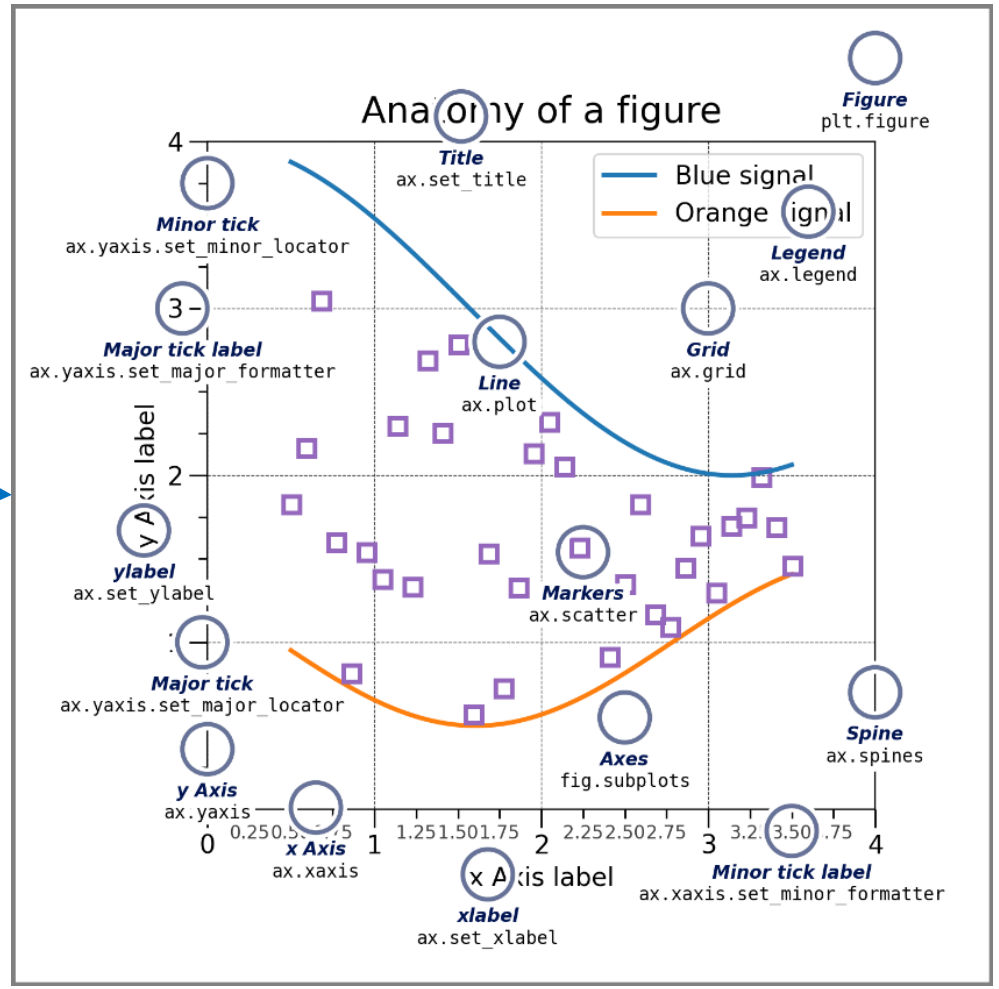
Basic Toolkit for learning AI

matplotlib

*Python **basic plotting library**, a few lines of code to generate plots, histograms, bar charts, scatter plots and more.*

Example: the double pendulum problem.

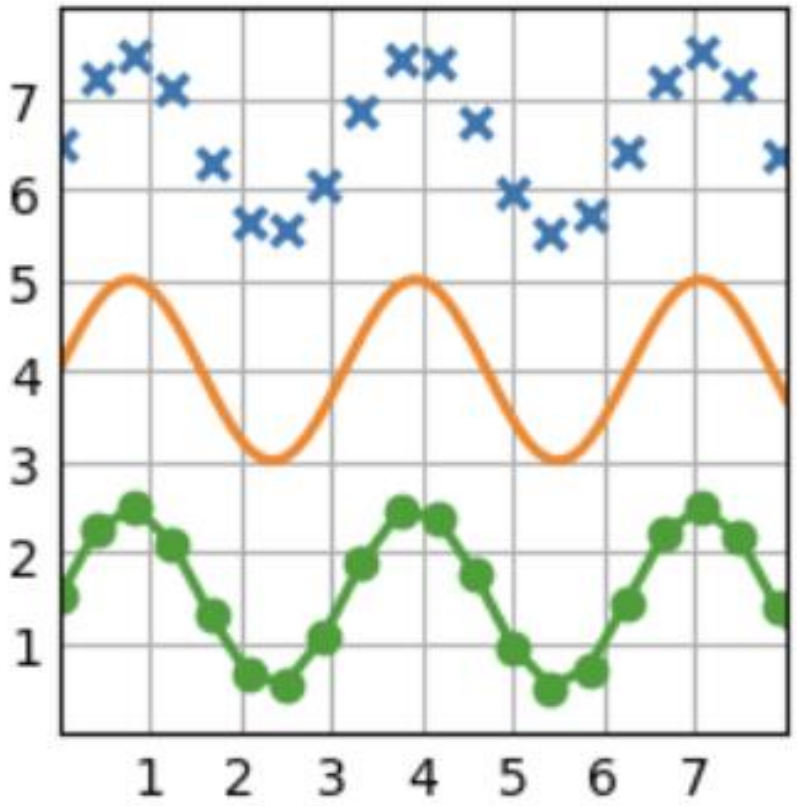




[Source]

matplotlib

```
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('_mpl-gallery')

# make data
x = np.linspace(0, 10, 100)
y = 4 + 1 * np.sin(2 * x)
x2 = np.linspace(0, 10, 25)
y2 = 4 + 1 * np.sin(2 * x2)

# plot
fig, ax = plt.subplots()

ax.plot(x2, y2 + 2.5, 'x', markeredgewidth=2)
ax.plot(x, y, linewidth=2.0)
ax.plot(x2, y2 - 2.5, 'o-', linewidth=2)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))


plt.show()
```



Test and try yourself

Given the functions of Sin, Cos, and quadratic function by matplotlib function.

Pandas

*A powerful toolset for **analyzing structured data** that can be used to quickly implement data in/out, indexing.*

Pandas provides two types of classes for handling data:

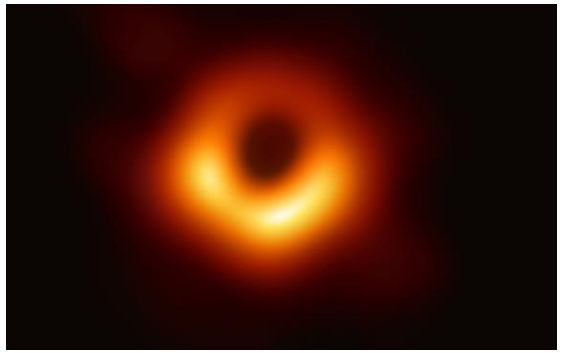
- Series: a *one-dimensional labeled array* holding data of any type such as *integers, strings, Python objects etc.*
- DataFrame: a *two-dimensional data structure* that holds data like a two-dimension *array or a table with rows and columns.*

Numpy

Basic package for scientific computing with python. At its core, it is based on array arithmetic with ***N-dimensional data objects ndarray****.*

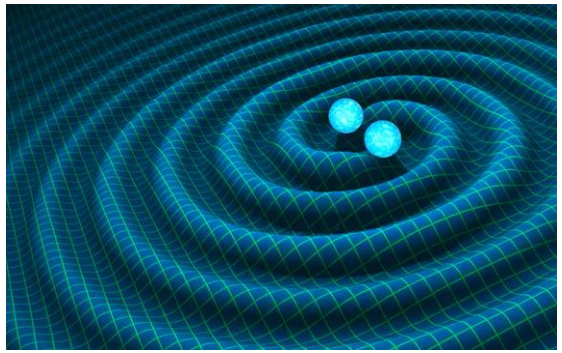
CASE STUDIES

First Image of a Black Hole



How NumPy, together with libraries like SciPy and Matplotlib that depend on NumPy, enabled the Event Horizon Telescope to produce the first ever image of a black hole.

Detection of Gravitational Waves



In 1916, Albert Einstein predicted gravitational waves; 100 years later their existence was confirmed by LIGO scientists using NumPy.

Sports Analytics

Cricket Analytics is changing the game by improving player and team performance through statistical modelling and predictive analytics.

Pose Estimation using deep learning

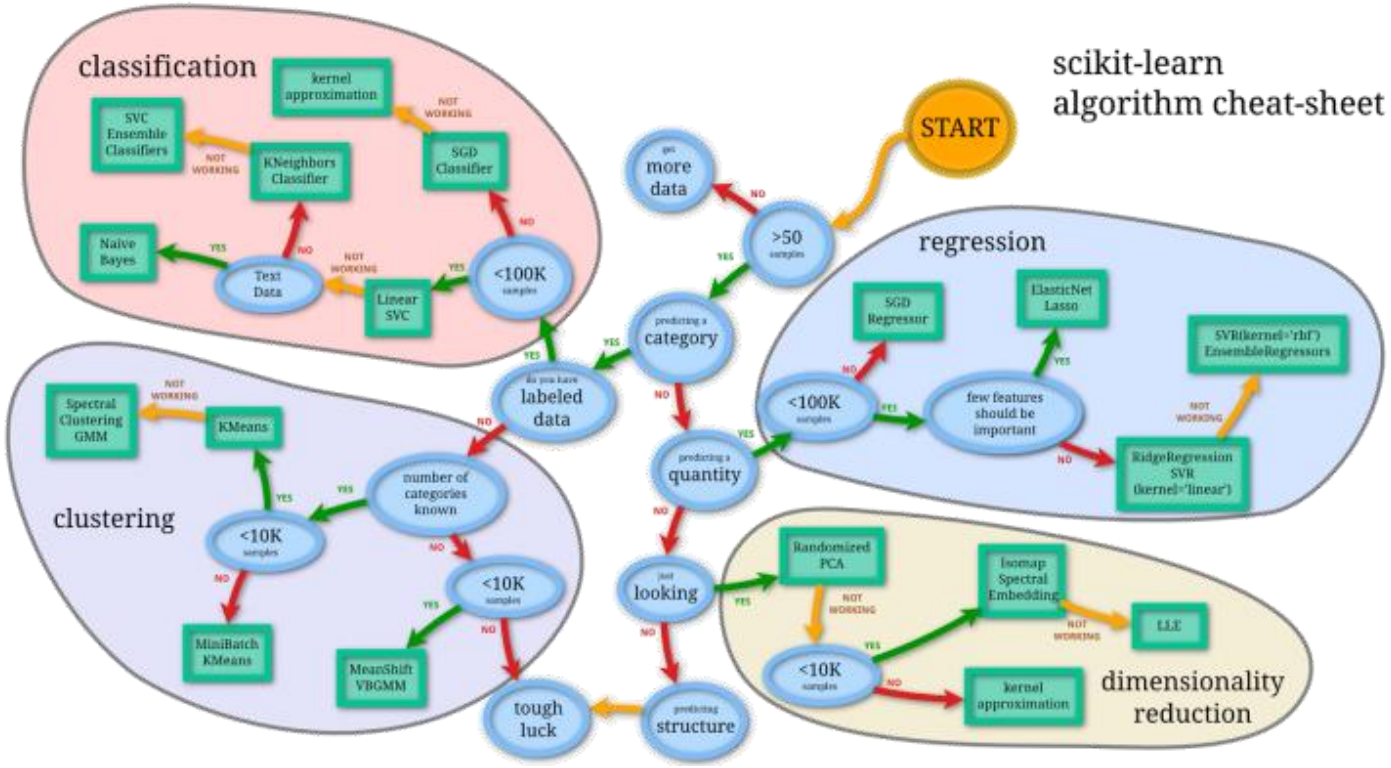
DeepLabCut uses NumPy for accelerating scientific studies that involve observing animal behavior for better understanding of motor control, across species and timescales.

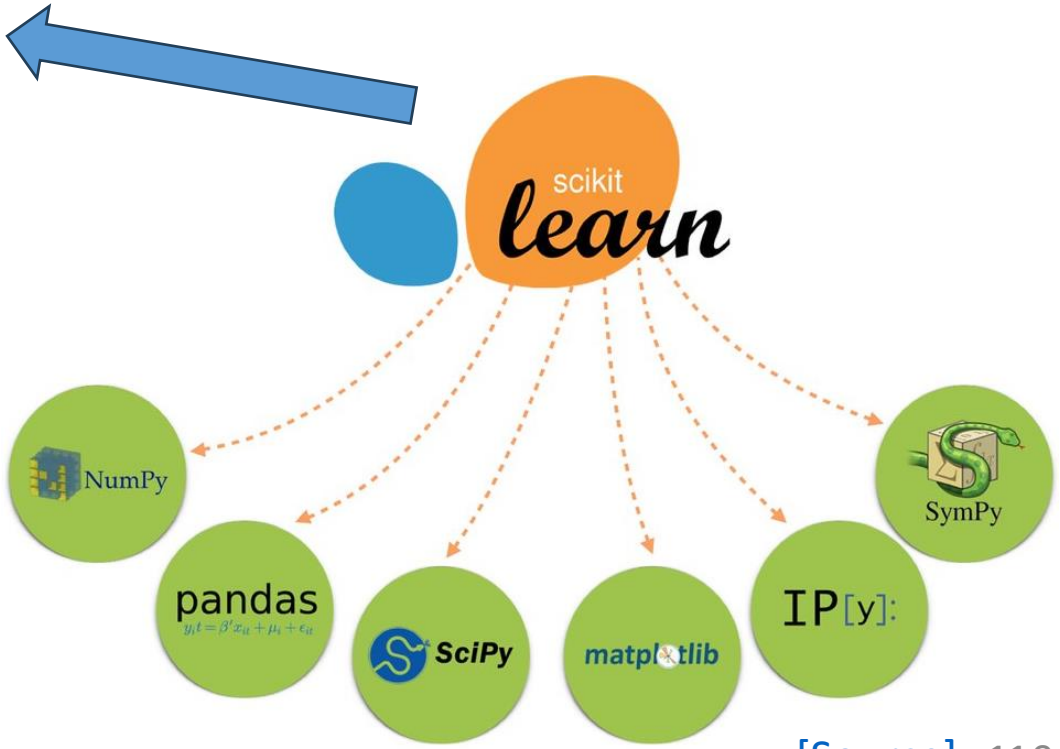
[Source]

Scikit-learn

Python's most powerful machine learning library

which is based on NumPy, SciPy and matplotlib, and supports a variety of machine learning models, including classification, regression, clustering and dimensionality reduction.





[Source]

Keras

Keras is a deep learning API written in Python and capable of running on top of either JAX, TensorFlow, or PyTorch.

First contact with Keras

The core data structures of Keras are **layers and models**. The simplest type of model is the Sequential model, a linear stack of layers.

Here is the model:Sequential

```
import keras
model = keras.Sequential()
```

Once your model looks good, configure its learning process with :.compile()

```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

Stacking layers is as easy as :.add()

```
from keras import layers

model.add(layers.Dense(units=64, activation='relu'))
model.add(layers.Dense(units=10,
activation='softmax'))
```

Common AI Framework

Pytorch

PyTorch was developed by Meta AI (formerly Facebook AI Research Lab) and is built on the Torch library. It was first released in 2016 and quickly gained attention for its flexibility, ease of use, and dynamic computational graphs.

Tensorflow

Launched in 2015, TensorFlow grew out of Google's internal research into its products and services. It evolved from an earlier framework called DistBelief and is designed to be more flexible and efficient.

Whether you favor the flexibility and user-friendliness of PyTorch or the scalability and robustness of TensorFlow, your choice will be a critical step in your AI and ML endeavors.