ENPM605 – Python Applications for Robotics

L1 – Course Overview

Version – 1.1

Lecturer: Z. Kootbally

School: University of Maryland

Semester/Year: Spring/2024

MARYLAND APPLIED
GRADUATE ENGINEERING

2024/02/03

- v1.1 (02/02): Rewrote slides on documentation generation (slides 36–39).
- v1.0 (01/30): Original version.

Conventions

- This is a *link*
- This is a  file.txt
- This is a  folder
- This is a  `terminal command`
- This is a  keyboard shortcut

About Me

- Affiliations:
  - National Institute of Standards and Technology (NIST), Gaithersburg, MD.
    - Leader of the *Automated Vehicles* Program.
    - Co-leader of the *Agility Performance of Robotic Systems* (APRS) Project.
  - University of Maryland (UMD), College Park, MD.
    - Lecturer – ENPM809Y, ENPM605, and ENPM663.
- Contact – *zeidk@umd.edu*
- Office hours – On demand.
- Interests:
  - ***Robot Agility*** – Industrial robotics, knowledge representation, robot control, task planning, and artificial intelligence (deep learning, autoencoders, procedural generation).
  - ***Automated Vehicles (AV)*** – Development of a testbed to measure the system interaction between modules of an AV system (cybersecurity, AI, communications, and perception).

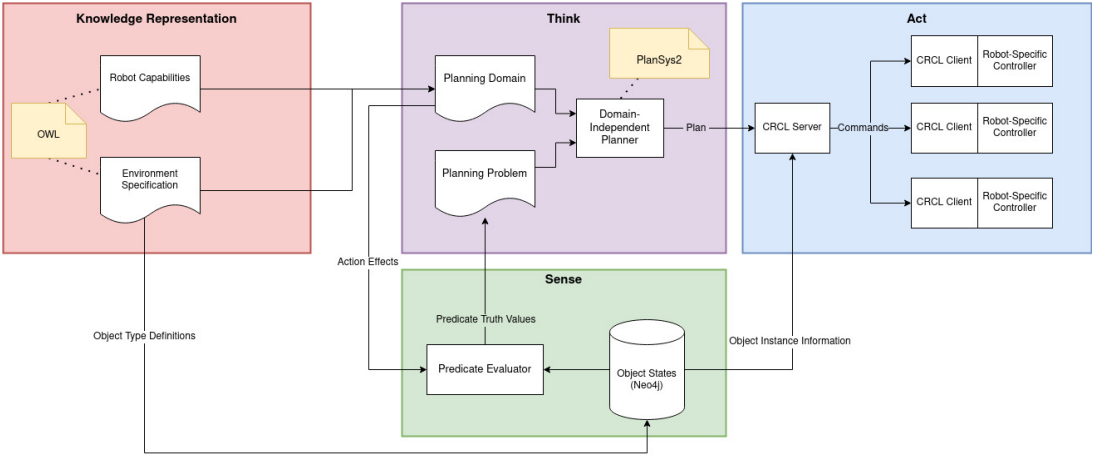
Robot Agility

- How well can a robot adapt/respond to task failures?
- How fast can a robot re-plan when a new goal is provided to it?
- How can we allow for interchangeability of robots without the need for reprogramming?
- How well can a robot respond to changing environmental conditions?

Target Audience

- Small and medium-sized manufacturers (high mix and low volume) need robots that are easier to task and retask and are more robust to execution failure during operations.
  - Able to detect and recover from failures to avoid downtime.
  - Example: *Material Handling (Trited Innovation)*
  - An agile robot: *Peg Insertion (Energid Technologies)*

Agility Performance of Robotic Systems (APRS) Project



Agile Robotics for Industrial Automation Competition (*ARIAC*)

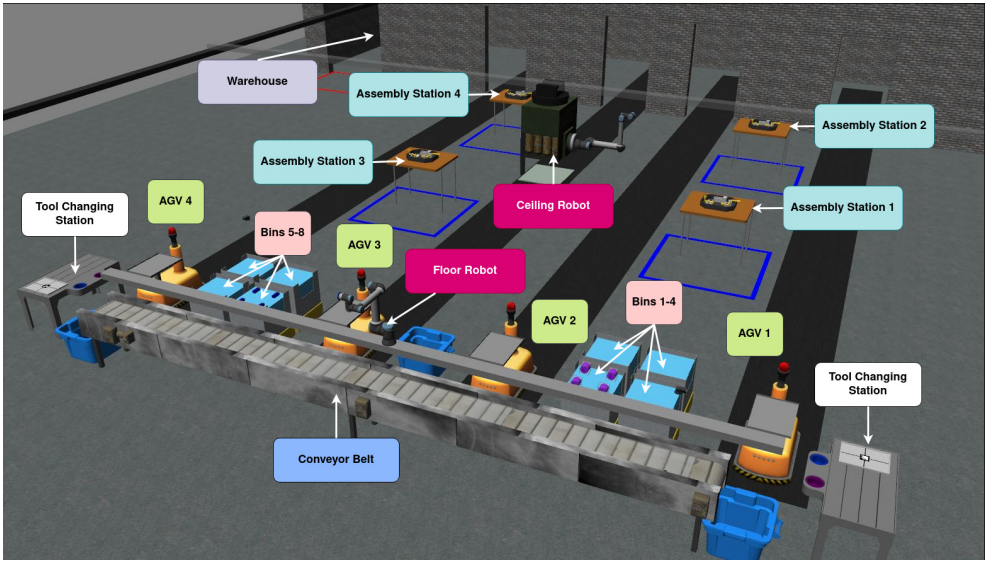
- ***Overview***:
  - Virtual competition organized by the National Institute of Standards and Technology (NIST) since 2017.
  - Cash prizes for the top three performing teams (see *ARIAC Rules*).
- ***Goal*** – Test the agility of industrial robot systems, with the goal of enabling industrial robots on the shop floors to be more productive, more autonomous, and to require less time from shop floor workers.
- ***Outcomes***:
  - Determine effective strategies pertaining to industrial applications.
    - Competitors' code is not shared or used by NIST.
  - Assess agility measurements and pinpoint deficiencies in scoring equations.

ENPM663 focuses on ARIAC.

Roles of Competitors

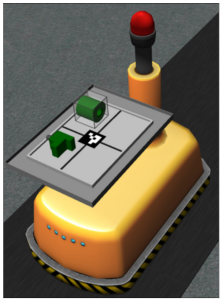
- Develop a ROS2 software package that is able to interface with the ARIAC simulation environment in order to receive and complete orders.
- Use the provided *ARIAC interfaces* for communications between competitors' control system and the ARIAC manager.
  - Multiple *tutorials* are provided by the NIST team:
    1. Start the competition.
    2. Read data from a break beam sensor.
    3. Process data from an RGBD Camera.
    4. Read an order.
    5. Move AGVs between stations.
    6. Pick a part.
    7. Pick and place a tray.
    8. Complete a kitting task.
- Develop a program to solve agility challenges.

Environment

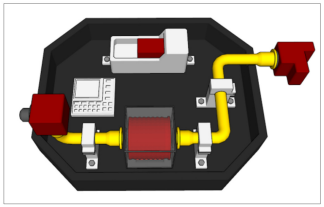


Tasks

- ***Kitting*** – Kitting is the process which groups separate but related parts as one unit. [*clip*]
- ***Assembly*** – For an assembly task, competitors are expected to use parts located on an AGV and assemble those parts at one of the four assembly stations. [*clip*]
- ***Combined*** – A combined task consists of a kitting task and an assembly task. In a combined task, only the assembly task is scored.



Completed kitting task



Completed assembly task

Agility Challenges

- ***Robot malfunction*** – One of the two robots or both robots will stop working (controllers shutdown) for a duration which is unknown to the competitors.
- ***Sensor blackout*** – Sensors of a specific type will stop reporting data for a duration unknown to the competitors.
- ***Faulty grippers*** – During pick-and-place operations, faulty grippers result in parts slipping or falling off the grippers. [*clip*]
- ***High-priority orders*** – An order of high priority interrupts (is announced) the current order.
- ***Faulty parts*** – Faulty parts are intentionally placed in bins or on the conveyor belt. [*clip*]
- ***Insufficient parts*** – In some trials, there are deliberately not enough parts provided to finish certain tasks.
- ***Part reorientation*** – In certain trials, parts must be re-oriented (turned over) prior to being placed into kitting trays. [*clip*]

Automated Vehicles Program

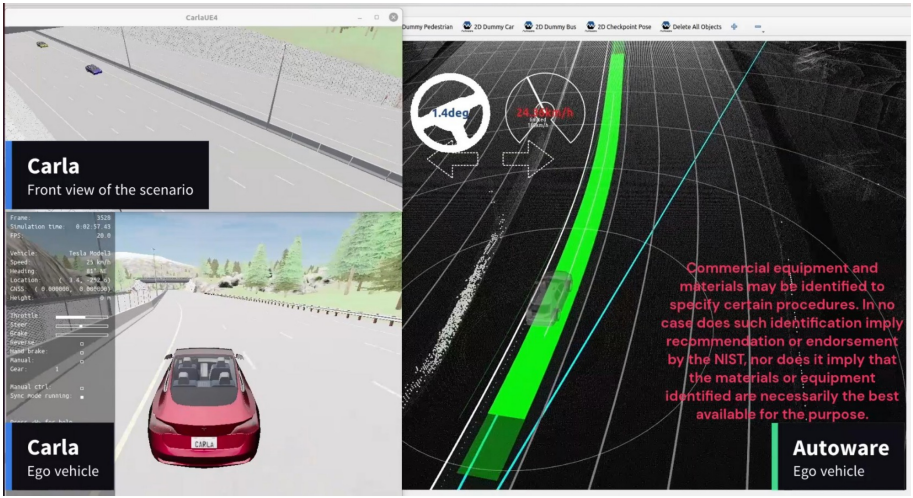
- Address system technology (Perception, AI, Cybersecurity, and Communications) performance and measurement methods.
- Design and establish a systems interaction testbed.
  - Simulation-based testbed in FY23.
  - Physical testbed in FY24.
- Provide the metrology and standards to increase the safety and the security of automated vehicles (AVs).
  - Allow industry to better understand and characterize their AV's performance.
  - Provide Government agencies the knowledge to create regulations.

Key Deliverables

- Measurement science validated through virtual and physical testbeds.

- A reproducible, vehicle-level, systems interaction testbed, with associated measurement science and standards, for on-road AVs.

Simulation



Tools: *https://github.com/usnistgov/SERI-AV*

About You

- What is your name?
- What experience, if any, do you have with robotics?
- Why are you interested in this course?
- One interesting fact about yourself.

System and Software

- Operating System (OS):
  - Ubuntu 20.04 (Focal Fossa).
  - You can use Windows or Mac OS in class...until we get to ROS lectures.
- List of software used in class:
  - *Visual Studio Code (VSC)* – Light, cross-platform IDE.
  - *The Robot Operating System (ROS)* – Galactic.
- The Linux environment may be new for you.
  - Use this course as an opportunity to learn Linux.
  - You should at least know the following shell commands (terminal):
    - `ls`, `pwd`, `cd`, `mv`, `mkdir`, `rm`, `source`, `touch`, `man`, and `grep`.

Course Overview

- Specifically designed for students who want to learn robotics with direct applications.
- Focus on Python programming and its applications with mobile robots using the Robot Operating System (ROS2).
- Students will perform small hands-on exercises in class to gain a deeper understanding of how the application of Python to ROS can be used in real-world challenges.
  - Some in-class exercises will count towards the next assignment (10% – 15%).
- The final project will be to control one or multiple mobile robots in a Gazebo environment.

- ❓ Do you need robotics knowledge for this course?
- ❓ Do you need prior programming knowledge to take this course?

Class Structure

1. Questions and discussions on the previous lecture.
2. Quiz.
3. Lecture.
   - You need to come to class with a laptop running the required software.
   - Exercises are due in class.
   - Participation is very important and this is how I will remember you at the end of the semester.

Syllabus

Refer to **ENPM605_SPRING2024_Syllabus.pdf** (uploaded on Canvas).

Quizzes

- Quizzes are taken during class hours between 4:00pm and 4:20 pm.
- Quizzes are not announced beforehand.
  - If you are not a remote/online student and you cannot come to class:
    - Do not take the quiz.
    - Take the quiz another day with a 10 % penalty.
      - No penalty if you are excused (doctor's note, supervisor's note, etc).
- No proctor needed.
- Closed-notes quizzes only (no slides and no software to test code).

Assignments

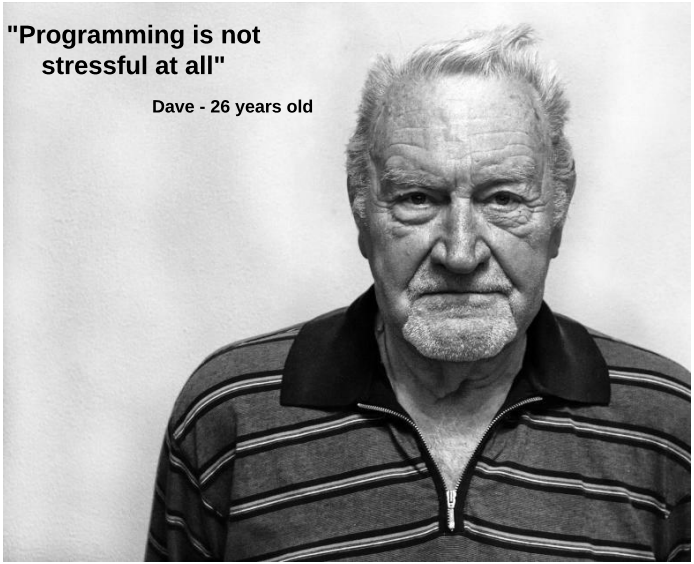
- Individual and team assignments.
- Assignments must be submitted on time on Canvas (penalty for late submission).
- Each assignment must be submitted as a zipped file (e.g., **zeid_kootbally_rwa1.zip**).
- Grading rubric and assignment description are provided in a PDF file.

Peer Reviews

- After group assignments and for the final project, you will give your teammates a score from 1-10 indicating their productive involvement.
  - You cannot rate yourself.
  - You get 1 pt by just emailing me the peer reviews for your teammates.
- 60% of your overall score will be based on your assignment/project grade and 40% will be based on your peer reviews from your group.
- Example:
  - Your group got 30/30 pts on an assignment/project.
  - Your average peer review from your teammates is 4/10 pts.
  - You emailed me your peer review for your teammates: Your average peer review becomes 5/10 pts (50%).
  - Your grade is 18 (60% of 30) + 6 (50% of 12) = 24/30 pts.

Final Project

- There is no final exam in this course but one final project.
- The final project involves developing a ROS package that assigns tasks to one or more mobile robots, which they must execute within a Gazebo simulation environment.


"Programming is not stressful at all"
Dave - 26 years old

Python is an interpreted, high-level, general-purpose programming language created by Guido van Rossum. It is dynamically typed and garbage-collected.[1]

[1] *https://en.wikipedia.org/wiki/Programming_language*

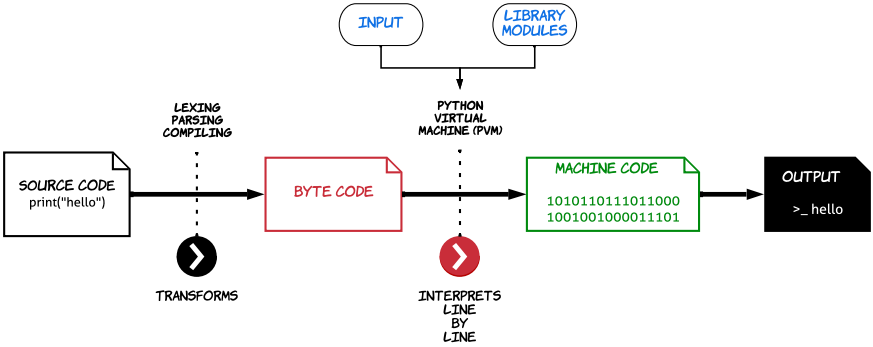
Python

- Since 2003, Python has consistently ranked in the top ten most popular programming languages as measured by the *TIOBE Programming Community Index* and by *PYPL*.
- Although Python 2.7 is still widely used, it is no longer maintained since January 2020.
- Python 3.0 (a.k.a. "Python 3000" or "Py3k") was released in 2008 and the latest version (*3.12.0*) was released in Oct, 2023.
- We need any Python 3.x for this course:
  - Check the default version of Python: `>_ python --version`
  - Check the version of Python3 on your system: `>_ python3 --version`
- If Python is not installed, install it with:

```
>_ sudo apt update && sudo apt install python3
```

Interpreted Language

- ***Source code*** – Human readable code, written in the Python language.
- ***Byte code*** – Series of instructions or a low-level program for the Python interpreter.
- ***Output*** – Byte code is parsed line by line and executed.
- ***Machine code*** – Code only the CPU can understand.



Compiled Languages

- A compiler translates the source code directly into binary machine code. After compilation, the target machine will directly run the machine code.
  - *Advantage* – Faster since the machine code has already been generated.
  - *Disadvantage* – Compile time can be long.

Interpreted Languages

- The source code is not directly run by the target machine. There is another program called the interpreter that reads and executes the byte code. The byte code cannot be run by the target machine directly.
  - *Advantage* – Portable since it does not depend on the hardware.
  - *Disadvantage* – Slower since the interpreter has to do extra work to have the byte code instruction translated into a form that can be executed on the machine.

CPython – The default implementation of the Python programming language is CPython which is written in the C programming language. CPython compiles the Python source code into the byte code, and this byte code is then executed by the CPython virtual machine.

- Visual Studio Code (VSC or VSCode) was chosen due to its versatility, complexity, the large number of extensions, and its integration with ROS.
- If you do not like VSCode, I suggest you take a look at PyCharm (student version) as it also provides a plugin to work with ROS.

VSCode Setup

- Create a workspace where you will host all Python code for this course.
- Open a terminal with Ctrl+Alt+t
- Go to your workspace: `cd <path to workspace>`
- Enter `code .`
- By starting VSCode in a folder, that folder becomes your "workspace".
- VSCode stores settings that are specific to that workspace in **.vscode/settings.json**, which are separate from user settings that are stored globally in **~/.config/Code/User/settings.json**.
- **.vscode/settings.json** is not created by default. If you modify the Workspace settings, **.vscode/settings.json** will be created.
- Any **settings.json** file in a nested folder takes precedence over other **settings.json** files.
- See **keyboard-shortcuts-linux.pdf** for VSCode shortcuts on Linux.

VSCode Extensions

- Get **extensions.json** from Canvas and place it in **.vscode**
- If you load the workspace for the firs time, you will get a prompt asking to install the extensions: Select ***Install***.
- Otherwise, Ctrl+Shift+P > ***Extensions: Show Recommended Extensions*** > Click on the little cloud icon to install all the recommended extensions.
  - You can edit this file to add new extensions if you want. You should do a backup of this file somewhere in the cloud. You can reuse this file whenever you install/reinstall VSC.

VSCode Python Interpreter

- In order to run Python code and get Python IntelliSense, you must tell VSCode which interpreter to use.
  1. Open **Command Palette** with Ctrl+Shift+P
  2. Start typing **Python: Select Interpreter**
  3. Select a Python3 interpreter.

Create Python File

- Create the folder **lecture1** in your workspace.
- Create the file **lecture1.py** inside the folder.
  - We can name this file anything we want.
- Add the following in **lecture1.py**

```
def greet(name):
    print("Hello " + name)

name = "Carl"
greet(name)
```

Execute Python Code

- Interpreted mode:
  - Use **Run Python File** (top-right triangle).
    or
  - Right-click on the Python file and select **Run Python File in a Terminal**
- Interactive mode:
  - ***REPL*** stands for Read-Eval-Print Loop, and it's a simple, interactive programming environment.
  - The REPL process is as follows:
    - ***Read*** – Reads a line of input from the user.
    - ***Evaluate*** – The inputted statement or expression is then evaluated or executed by the interpreter.
    - ***Print*** – After evaluation, the result of the execution is displayed or printed out to the user.
    - ***Loop*** – The process loops back to the reading stage, waiting for more input from the user.
  - Start REPL with Ctrl+Shift+P and start typing **REPL**.
    - Select **Python: Start REPL**.
  - If you select a portion of the code you want to run, then Shift+Enter, you will enter REPL mode where the portion of the code you selected is executed.

Debug Python Code

1. Set a breakpoint: Click in the gutter or place your cursor on a line and press F9
2. Expand top-right triangle and select **Debug Python File** or **Run** > **Start Debugging**
3. Choose **Run and Debug** or Ctrl+Shift+D
4. Since this is your first time debugging this file, a configuration menu will open from the **Command Palette** allowing you to select the type of debug configuration you would like for the opened file.
   - Select **Python File**.
   - To customize debugging, select **create a launch.json file**

- You can also work with variables in the **Debug Console**.
  - Start a debug session first and then use the > field at the bottom for your inputs.

Write Documentation

- Documenting your code is very important for third-party users and for yourself.
- Python uses docstring for code documentation.
- First, make sure the extension *Python Docstring Generator* is installed.
- Create a docstring documentation by starting typing """ (or ''') and press Enter when you see **Generate Docstring**
- Modify the default generated docstring:

```
'''
File to show how to use docstrings in Python.
'''

def greet(name):
    '''
    This function greets the user by name.

    Args:
        name (str): The name of the user.
    '''
    print(f"Hello, {name}!")

greet("Alice")
```

Sphinx

- *Sphinx* is one of the tools to generate HTML documentation from Python code.
- Install Sphinx:
  - `sudo apt install python3-sphinx`

Generate Documentation

- `>_ cd <path to workspace>`
- `>_ mkdir docs`
- `>_ cd docs`
- `>_ sphinx-quickstart`
- Modify source/conf.py

```
import os
import sys
sys.path.insert(0, os.path.abspath('../../'))
```

```
extensions = ['sphinx.ext.autodoc', 'sphinx.ext.napoleon']
```

- Modify index.rst

```
.. toctree::
   :maxdepth: 2
   :caption: Contents:

   modules
```

- Run `>_ sphinx-apidoc -o source ../lecture1`
- Run `>_ make html`
- Open build/html/index.html in a browser.
- From now on, run `>_ make html` each time you add/edit docstrings in your code.

Install `pip3`

- `pip` and `pi3` are package managers for Python, used to install and manage Python packages and libraries. `pip` is typically associated with Python 2.x, while `pip3` is used for Python 3.x. In modern Python environments, it is recommended to use `pip3` for Python 3.x.
- Install `pip3`
  - `>_ sudo apt update`
  - `>_ sudo apt install --reinstall python3-pip`
    - This command will both install `pip3` if it is not already installed and upgrade it to the latest available version if it is already installed.

New Theme for the Documentation

- Let's install the *Read The Docs* theme: `>_ pip3 install sphinx-rtd-theme`
- Change the theme in conf.py

```
html_theme = 'sphinx_rtd_theme'
```

- Regenerate the HTML documentation: `>_ make html`

Python Basics – Part I