Assignment 1

(100 pts is the perfect score)

Deadline: Tuesday, Sept 24, 11:59 PM

Submission Process: You are asked to upload a zip file on Canvas containing all necessary files of your submission. Only 1 student per team needs to upload a submission.

You should create a top-level folder named with your NETID, (e.g., xyz007-abc001 for teams of 2 students or just xyz007 for students working independently). Include at least the following files by following this folder structure:

- xyz007-abc001/component_1.py
- (...)
- xyz007-abc001/component_n.py
- xyz007-abc001/component_i.{png|gif|mp4}
- xyz007-abc001/report.pdf

That is, for each *i* component of this assignment, submit the related source code and any media that is specified. A report, in pdf format, detailing your implementation is also needed. You can optionally create a *utils.py* file that contains helper functions under the xyz007-abc001 folder.

Please zip this xyz007-abc001 folder and submit the xyz007-abc001.zip on Canvas.

On the first page of the report's PDF indicate the name and Net ID of the students in the team (up to 2 students).

Extra Credit for LaTeX: You will receive 6% extra credit points if you submit your answers as a typeset PDF, i.e., one generated using LaTeX. For typeset reports in LaTeX, please also submit your .tex source file together with the report, e.g., by adding a report.tex file under the xyz007-abc001 folder. There will be a 3% bonus for electronically prepared answers (e.g., using MS Word, etc.) that are not typeset. Remember that these bonuses are computed as a percentage of your original grade, i.e., if you were to receive 50 points and you have typeset your report using LaTeX, then you get a 3-point bonus. If you want to submit a handwritten report, scan it or take photos of it and submit the corresponding PDF as part of the zip file on Canvas. You will not receive any bonus in this case. For your reports, do not submit Word documents, raw ASCII text files, or hardcopies etc. If you choose to submit scanned handwritten answers and we cannot read them, you will not be awarded any points for the unreadable part of the solution.

Failure to follow the above rules may result in a lower grade on the assignment.

Programming Languages:

You can use any programming language but we encourage you to use either Python or C++. In general, you can use any library **as long as it does not implement components of the assignment**. For example, in this first assignment, for vector operations you can use Numpy (Python) or Eigen (C++) but cannot use rotation-specific operations.

Your assignment has to run on iLab and if there is any specific instruction to run your assignment, add it to your report. If we cannot run your code on iLab, 0 credits will be awarded for the programming tasks.

If you are using python, please use the following conda environment:

Setup python environment

```
#!/bin/bash
conda create -n cs560 python=3.10 numpy=1.25 matplotlib=3.7 -y
```

```
conda activate cs560
# run your code from here
```

It should be possible for the TAs to check whether your software returns a correct result for the implemented functions. Be sure to use the function signature given.

Note that the following two assignments will build on top of the infrastructure of the first one. So take some care with the code you develop. It should be clean, modular and reusable. For all of your assignments, consider that the units are meters and radians.

Grading: Each component contributes a different percentage to the grade of your assignment.

CS460 The *extra* portions are optional, a correct implementation would give you an extra 10%.

CS560 The *extra* portions are necessary to get the maximum grade of each component.

1 Validating Rotations (15%)

Implement the following functions:

```
def check_SOn(matrix: m, float: epsilon=0.01) -> bool:
    (...)
def check_quaternion(vector: v, float: epsilon=0.01) -> bool:
    (...)
def check_SEn(matrix: m, float: epsilon=0.01) -> bool:
    (...)
```

Note: In most libraries, you may find a rotation or quaternion implemented as classes. For your implementation, use *standard* matrices (Numpy's n-darray) and consider a vector to be a 1-column matrix.

- **check_SOn** - Input: matrix $\in \times$. Return: True if $m \in SO(n)$ (within epsilon numerical precision), false otherwise. (Consider n=2 or n=3 only)
- **check_quaternion** - Input: vector $\in \times$. Return: True if $v \in S^3$ (within epsilon numerical precision), false otherwise.
- **check_SEn** - Input: matrix $\in \times$. Return: True if $v \in SE(n)$ (within epsilon numerical precision), false otherwise. (Consider n=2 or n=3 only)

It is up to you how use the *epsilon* parameter.

Report your implementation.

1.1 Extra

Implement the following functions:

```
def correct_SOn(matrix: m, float: epsilon=0.01) -> matrix:
    (...)
def correct_quaternion(vector: v, float: epsilon=0.01) -> vector:
    (...)
def correct_SEn(matrix: m, float: epsilon=0.01) -> matrix:
    (...)
```

Each function *corrects* the given input if the element is not part of the group within an epsilon distance. The corrected matrix has to be *close* to the input matrix (always returning the identity is an invalid implementation). The correction implies that you can compute a *distance* between the given matrix (which may not be part of the group) and a member of the group.

Test your function with multiple random rotations and report your implementation and the results in your report.

2 Uniform Random Rotations(25 pts)

Implement the following functions and visualize the output.

```
def random_rotation_matrix(bool: naive) -> matrix:
    (...)
```

- **random_rotation_matrix** - Input: A boolean that defines how the rotation is generated. If *naive* is true, implement a naive solution (for example, random euler angles and convert to rotation matrix). If *naive* is false, implement the function as defined in Figure 1 in [1]. Return: A randomly generated element $R \in SO(3)$.

Your function should always return valid rotations (calling your function *check_SOn* should always return True). This does not mean that the generated rotation is uniform random. A visual way to check the distribution of the rotations is to visualize a the rotation in a sphere as described in algorithm 1.

Algorithm 1 Rotation visualization

Require: R is a rotation ($SO(3)$ or S^3)
$v_0 \leftarrow (0, 0, 1)$ ▷ At north pole
$v_1 \leftarrow (0, \epsilon, 0) + v_0$
$v_0' \leftarrow R * v_0$
$v_1' \leftarrow R * v_1 - v_0$
Plot vector from v0p to v1p

In your report include and analyze any design choice of your implementation. Include visualizations of the random samples (the spheres).

2.1 Extra

```
def random_quaternion(bool: naive) -> vector:
    (...)
```

- **random_quaternion** - Input: A boolean that defines how the rotation is generated. If *naive* is true, implement a naive solution (for example, random euler angles and convert to rotation matrix). If *naive* is false, implement the function as defined in Algorithm 2 in [2]. Return: A randomly generated element $q \in S^3$.

3 Rigid body in motion (25%)

Define a planar environment of dimensions 20x20, with bounds [-10,10] for x and y. A rectangular robot of dimensions 0.5x0.3. This robot is controlled via a velocity vector $V = (v_x, v_y, v_\theta)$. There are no obstacles in this environment.

Implement the following functions:

```
def interpolate_rigid_body(vector: start_pose, vector: goal_pose) -> path:
    (...)
def forward_propagate_rigid_body(vector: start_pose, Plan: plan) -> path:
    (...)
def visualize_path(path):
    (...)
```

- **interpolate_rigid_body** - Input: start pose $x_0 \in SE(2)$ and goal position $x_G in SE(2)$. Output a path (sequence of poses) that start at x_0 and ends in x_G.
- **forward_propagate_rigid_body** - Input: start pose $x_0 \in SE(2)$ and a Plan (a sequence of N tuples (velocity, duration)) that is applied to the start pose and results in a path of $N+1$ states.
- **visualize_path** - Input: A path to visualize. Your visualization must include the path and an animation of the robot's movement.

In your report include and analyze any design choice of your implementation. Include visualizations of the paths generated by both methods. Submit an example animation (gif is prefered but can also be a small mp4).

4 Movement of a Arm (35%)

Using the same environment as before, implement the 2-joint, 2-link arm in figure 1 arm using boxes as the geometries of individual links. Here are more details regarding your robotic arm:

- The first link has length 2 and the second has lenght 1.5.
- All frames associated with links $\{L_i\}$ are at the *center* of the boxes. The frames associated with joints $\{J_i\}$ are located at the box's bottom.

To implement your arm, define the coordinate frame of each link and the relative poses with each other. Start with only the first link and its transformations before moving to the second link.

```
def interpolate_arm(vector: start, vector: goal) -> path:
    (...)
def forward_propagate_arm(vector: start_pose, Plan: plan) -> path:
    (...)
def visualize_arm_path(path):
    (...)
```

- **interpolate_arm** - Input: start configuration $q_0 = (\theta_0, \theta_1)$ and goal configuration q_G. Output a path (sequence of poses) that start at q_0 and ends in q_G.
- **forward_propagate_arm** - Input: start pose $q_0 \in SE(2)$ and a Plan (a sequence of N tuples (velocity, duration)) that is applied to the start pose and results in a path of $N+1$ states.
- **visualize_arm_path** - Input: A path to visualize. Your visualization must include the path and an animation of the robot's movement.

In your report include and analyze any design choice of your implementation. Include visualizations of the paths generated by both methods. Submit an example animation (gif is prefered but can also be a small mp4).

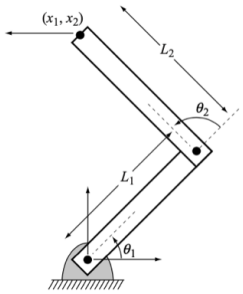

Figure 1: Robot Diagram: 2-link planar manipulator

Collusion, Plagiarism, etc.: Each team must prepare their solutions independently from others, i.e., without using common code, notes or worksheets with other students. You can discuss material about the class that relates to the assignment but you should not be provided the answers by other students and you must code your own solutions as well as prepare your own reports separately.

Furthermore, you must indicate at the end of your report any external sources you have used in the preparation of your solution. This includes both discussions with other students and online sources. Unless explicitly allowed by the assignment, do not plagiarize online sources and in general make sure you do not violate any of the academic standards of the course, the department or the university.

Online sources include the use of ChatGPT or other Large Language Models and Generative AI tools. While the use of such tools is allowed as supportive instruments for programming, the teams need to report the level of their use and the part of the code that was generated by them. Students have to be careful that blindly copying significant pieces of code from such tools - which falls under the definition of plagiarism - may result on submissions that do not accurately answer the assignment.

Failure to follow these rules may result in failure in the course.

References

[1] J. Arvo, “Fast random rotation matrices,” in *Graphics gems III (IBM version)*. Elsevier, 1992, pp. 117–120.

[2] J. J. Kuffner, “Effective sampling and distance metrics for 3d rigid body path planning,” in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA’04. 2004*, vol. 4. IEEE, 2004, pp. 3993–3998.