Assignment 1

(100 pts is the perfect score - 10% of the course credit)

Deadline: Sunday, October 8, 23:59 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/create_scene.py (p1)
- xyz007-abc001/collision_checking.py (p2)
- xyz007-abc001/2d_rigid_body.py (p3)
- xyz007-abc001/2d_rigid_body_animation.gif/mp4 (p3)
- xyz007-abc001/planar_arm.py (p4)
- xyz007-abc001/planar_arm_animation.gif/mp4 (p4)
- xyz007-abc001/report.pdf

Please zip this xyz007-abc001 folder and submit the xyz007-abc001.zip on Canvas. You can also create a utils.py file that contains helper functions under the xyz007-abc001 folder.

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.

Requirements: Standard Python libraries are allowed. Additionally, Numpy, Scipy and Matplotlib can be used. Please use the following script to build your Python environment under conda.

Setup python environment

```
#!/bin/bash
conda create -n cs460 python=3.10.12 numpy=1.25.2 matplotlib=3.7.2 scipy=1.11.2 -y
conda activate cs460
# run your code from here
```

This python environment will be used for all assignments. We expect to run your code on ilab. If we cannot run your code in the python environment as described above, then 0 credits will be awarded for the programming tasks.

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.

1 Generate, visualize & store 2D polygonal scenes (20 pts)

Your software should be able to generate scenes with sets of convex polygons for testing. It should also be possible to visualize them as well as load/store them via files.

Build a script that can generate files that are similar to the example provided to you (e.g., "collision_checking_polygons.npy"). Make sure that your script has the ability to receive as input: i) the total number of polygons in the scene P, ii) the minimum number of vertices N_{min}, iii) the maximum number of vertices N_{max}, iv) the minimum "radius" r_{min}, and v) the maximum "radius" of the polygon r_{max}.

Here is a possible strategy for generating a convex polygon but you can also follow your own strategy:

- Sample uniformly at random coordinates (x_{center}, y_{center}), which will correspond to the center of the polygon.
- Sample the number N of vertices of the polygon by uniformly sampling in the range $[N_{min}, N_{max}]$.
- For each vertex n of the N vertices, sample uniformly at random an angle α_n in the range of $[0, \ldots, 360]$ degrees. Then, generate the coordinates of the N vertices by placing them on a circle centered at (x_{center}, y_{center}) with radius r_{min} and at the angle α_n.
- For each vertex, also uniformly sample a number r in the range $[0, r_{max} - r_{min}]$. Extrude each vertex outward from the center (x_{center}, y_{center}) by moving it away by the value r.
- For each vertex, use the sign of the cross product between each pair of adjacent vertices and the origin to determine convexity. If a vertex makes your polygon non-convex, then remove it. Iterate over all of your vertices until the polygon is convex or you are down to 3 vertices, where convexity is guaranteed.
- Alternatively for the last step you could also compute the convex hull of the N sampled vertices by using the "convexHull" function from scipy (scipy.spatial.ConvexHull).

Consider that your polygonal map is defined in a continuous space of dimensions $2m \times 2m$, i.e., the coordinates of the vertices should be represented in this continuous space. You should be able to visualize the corresponding map. For this you will have to pick a resolution and select the size of your visualization, e.g., 800pixels x 800pixels that will represent the $2m \times 2m$ polygonal map.

Polygons can be represented using numpy arrays, which will allow you to store the vertices of the polygons. You should then be able to store the corresponding files as well as load them.

In your report provide the following:

- 4 examples of maps that you have generated and the corresponding parameters you used for the generation of these maps. Use different parameters for the 4 different maps and discuss the choice of your parameters in the resulting environments you generated, i.e., try to generate some maps that are dense with obstacles and some that are sparse.
- A visualization of one additional map, which you have manually constructed, where you use convex polygons to form two letters inside the space. These two letters should correspond to the first letter of the first name of each team member (or the first letters of the first and last name for students working individually). You do not have to spend much time here to make

the letters accurately represented and the convex polygons perfectly aligned. As long as the letters are recognizable when visualized, this is sufficient. See example Fig. 1 for letter “E”.

- Describe the exact approach you followed in order to generate the convex polygons. If you followed the above mentioned strategy describe the operations you performed so as to i) rotate each polygon, ii) extrude vertices and iii) how you determine/enforce the convexity of each polygon. If you deviated from the above strategy indicate so and explain your choices.

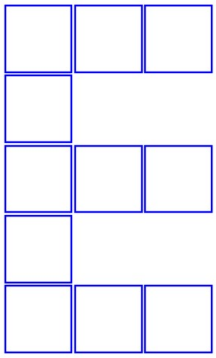

Figure 1: Letter example.

2 2D collision checking for convex polygons (20 pts + 10 points for efficient implementation)

You should implement an approach in your software for detecting whether two polygons collide with each other. For this component of this assignment attach a figure like the one shown in Fig. 2 in your report for all the maps generated as part of component 1 of the assignment. Fig. 2 gives you an example of how your polygonal scenes can look like in general. Plot polygons to be filled if there is a collision with other polygons or boundary unfilled otherwise. Save the figure and put it in the report. When you plot figures, please use

```
fig, ax = plt.subplots(dpi=100)
ax.set_aspect('equal')
```

Save all your code in collision_checking.py. You have two functions.

```
collides(poly1: np.ndarray, poly2: np.ndarray)
plot(polys: np.ndarray)
```

Your report should also describe how you implemented your collision checking approach. It is ok to execute an exhaustive approach where you: a) check all pairs of line segments between the two polygons for collisions and b) check whether any of the vertices of one polygon are inside the other. Discuss any attempts you made in order to speed up the polygonal collision checking code relative to the exhaustive approach (the 10 points for efficient implementation). Report how much faster is the code due to your changes by evaluating the code on the 5 scenes from part 1.

It should be possible for the TAs to check whether your software returns a correct result for a polygonal scene of their choice. For this reason, make sure that your software uses the following code in order to load an input file for collision checking:

```
polygons = np.load('collision_checking_polygons.npy', allow_pickle=True)
```

The TAs will store a different set of polygons during grading inside the "collision_checking_polygons.npy" file than the one shown in Fig. 2 to evaluate the correctness of the software.

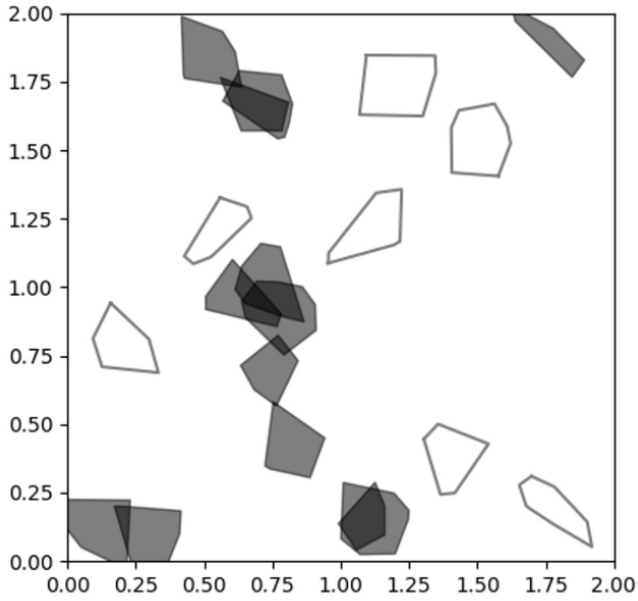

Figure 2: Collision checking example.

3 Collision-free Navigation for a 2D rigid body (30 pts)

Implement an environment in where you can move a rectangular object among a set of convex polygons without collisions. This rectangle will be controlled as a car in assignment 2, so make sure that your code is clean and reusable.

- Load a map ($2 \times 2m^2$) using the following code inside your software:

  ```
  polygons = np.load('2d_rigid_body.npy', allow_pickle=True)
  ```

  You can test different scenes by renaming them as "2d_rigid_body.npy" (the TAs will use a different scene other than the one shown in Fig. 3). Create a rectangular polygon (with dimensions: $0.2 \times 0.1m^2$) as a 2D rigid body and initialize it in a random collision-free position and orientation. Treat the boundary of the $2m \times 2m$ region as obstacles.

- Use the keyboard to control the rectangle: (i) forward/backward motion along the long axis of the rectangle, and (ii) clockwise and counterclockwise rotation about the center. Include in the report which 4 keys you have assigned to the corresponding control (e.g., up/down, left/right arrows).

- Make use of your collision checking code from the previous step so as to avoid the rectangular object penetrating the convex polygonal obstacles. In particular, every time that the keyboard command would result in a motion that leads to a collision, the rectangle should be returned to its previous collision-free position and orientation. The example video didn't have the "avoid penetrating" function but gives a general example of when collision will be detected.

- Generate animations on your maps that show that you can move the rectangle from a random collision-free configuration to another collision-free part of the scene using keyboard control in a collision-free manner. Save your animations in .gif or .mp4 format (you can use an external recording tool, keep file size less than 1MB, for example, ScreenToGif).

- Compute and visualize the collision-free configuration space of the rectangular object for the 5 different scenes you generated in part 1 of this assignment. Fig. 3 (right) provides an

example of this computation for the scene shown in Fig. 3 (left). Do so for three different orientations of the rectangle, 0^o as shown in Fig. 3, 45^o and 90^o. Use for computation the Minkowski sum operation. Include the corresponding visualizations in your report.

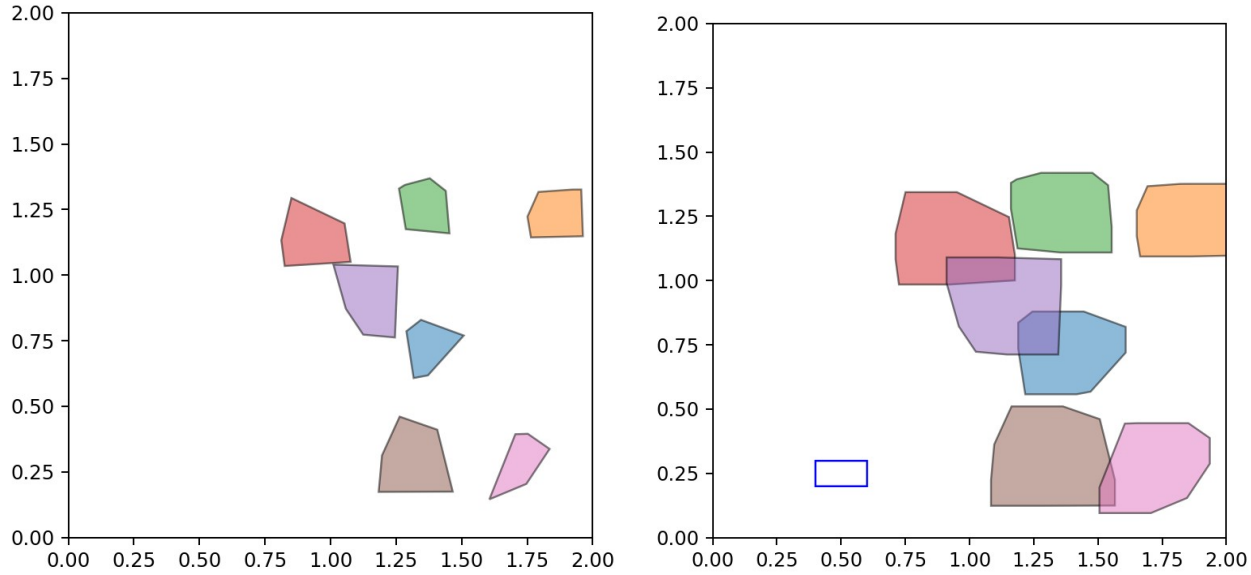

Figure 3: Rectangular object collision-free example. You don't need to draw the rectangle object in the right figure; this is just to illustrate that obstacles have been "expanded" based on the rectangle.

4 Collision-free Movement of a Planar Arm (30 pts)

Implement an environment in where you can move a planar 2-link arm among a set of convex polygons without collisions.

- You should be able to load your polygonal scenes and create an arm with two rectangular polygons and two joints inside them. The length between joint 1 and joint 2 should be 0.4m, and the length between joint 1 and joint 2 should be 0.25m. The width of the rectangles should be 0.1m. The joints correspond to circles of radius 0.05m. The first joint is fixed at coordinates [1,1]. Initialize your arm so that both joint angles are at 0 degrees, i.e., the arm points to the right. At this configuration detect all of the polygons that the arm is colliding with and remove them from the scene. Plot the rest of the convex polygons and the arm using matplotlib. The links should touch the joints, see Figure 4 for an example. Provide visualizations of the corresponding scenes in your report.
- Use the keyboard to control the two joints individually (you can achieve unlimited rotation with each joint). Indicate in your report what keys you used in order to control the arm.
- Make use of your collision checking code from step 2 of the assignment so as to avoid the robotic arm penetrating the convex polygonal obstacles remaining in the scene. In particular, every time that the keyboard command would result in a motion that leads to a collision, the arm should be returned to its previous collision-free position and orientation. The example video didn't have the "avoid penetrating" function but gives a general example of when a collision will be detected, and the workspace in this example is different from this assignment.
- Generate animations on your maps that show that you can move the planar arm from the initial collision-free configuration to other collision-free configuration using keyboard control

in a collision-free manner. Save your animations in .gif or .mp4 format (you can use an external recording tool, keep file size less than 1MB, for example, ScreenToGif).

- Compute and visualize the collision-free configuration space for the planar arm by discretizing each joint angle with a resolution of $2\pi/100$ in the range of $[-\pi, \pi]$. See Fig. 4 (right) for an example. A 100x100 occupancy grid is generated where each cell corresponds to values for the two joint angles. The x-axis is the first joint angle and the y-axis is the second one. The yellow color corresponds to collisions with polygonal obstacles in the scene. You do not need to consider self-collision of arm for this part of the assignment. For this plot, please use provided map:

```
polygons = np.load('arm_polygons.npy', allow_pickle=True)
```

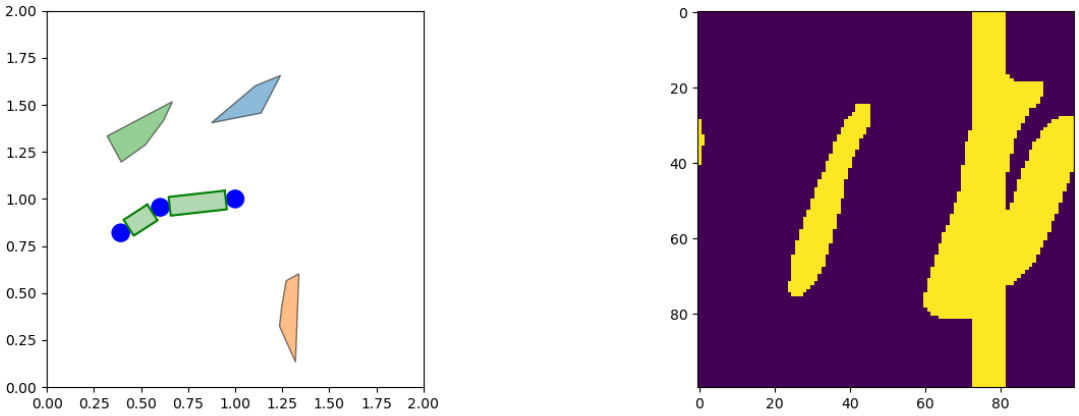

Figure 4: Arm collision-free example.

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.