EEL 4930/5934: Autonomous Robots
HW4: Robot Perception and Motion Planning (Spring 2025)

Tasks Overview:

A. **Augmented Visuals by Homography Estimation**
   - Section-5934: 25% + 5%
   - Section-4930: 30% + (Bonus 5%)

B. **Camera calibration of your cell phone or any other camera (20%)**

C. **Perspective geometry: analytical exercise**
   - Find line-of-sight direction (5%)
   - Find translation from Essential matrix (5%)

D. **Implement 2D motion planners**
   - Bug0 algorithm (20%)
   - Bug1 algorithm is implemented for you
   - Bug2 algorithm (20%)
   - Bonus (up to +10%)

Submission: Canvas only; Due: Monday March 31, 2024 by 11.59pm

- A single zip file with a folder that should contain
  - **Part_A** sub-folder with completed code
  - **Part_D** sub-folder with completed code
  - A single PDF report
  - Optional Readme.txt with any additional information

- **Part A**
  - Download the **HH4_Blank** folder from Canvas. Check the **Part_A** sub-folder.
  - Do not add any more Python files; just complete the functions.
  - In your report, only show the input/output images as shown below

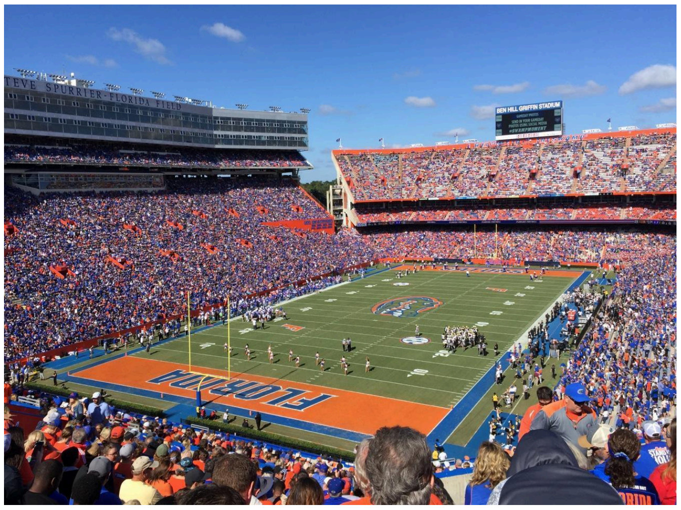
- **Part B-C**:
  - No code is needed; present results in the report

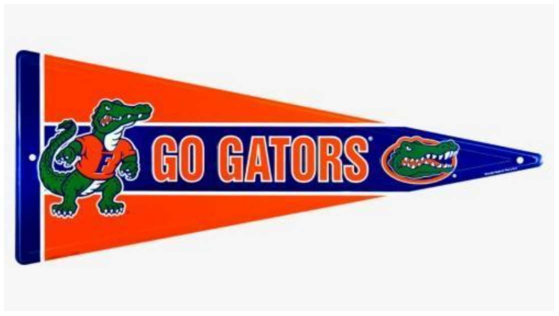
- **Part D**
  - Download the **HH4_Blank** folder from Canvas. Check the **Part_D** sub-folder.
  - Do not add any more Python files; just complete the functions.
  - In your report, only show the input/output images as shown below
- Assignments over 30 MB will get a penalty (up to -25%)
- **References:** Lecture 5,7 contents

Part A: Augmented Visuals by Homography Estimation

Refer to the following files in the HH3_Blank

- Driver script: `AR_Homography.py`
- Library: `libs_hh3/homography.py`
- Input data (see below): `data/game_fl.jpg` and `data/logo_gators.png`





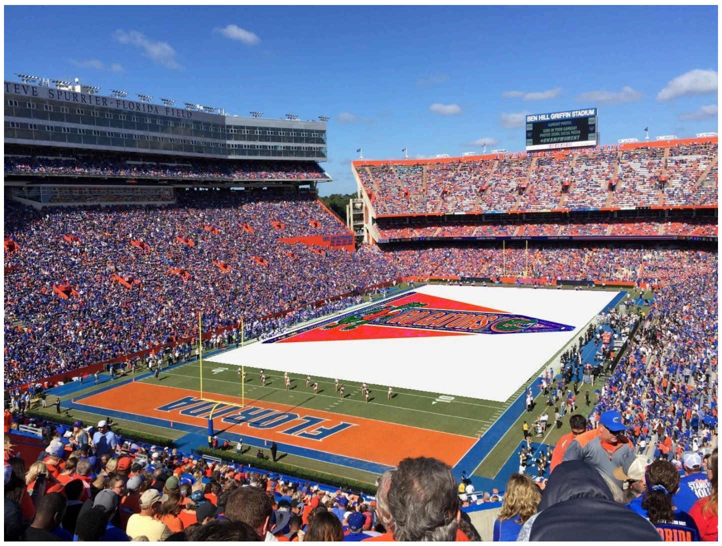
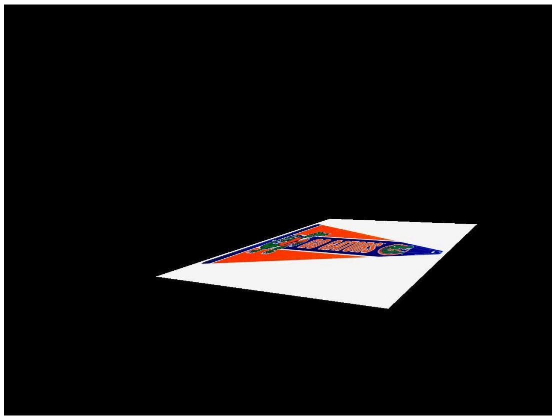
Your target is to augment the logo into the image by homography perspective transformation.

Part A.1: To achieve this, please follow these steps

- Get the four points on the destination image using the `Get_mouse_clicks.py`
- The driver script `AR_Homography.py` is already completed for you
- You need to complete the following library function (in `libs_hh3/homography.py`)

```
def computeHomography(Us, Vs)
```

- You can check your outputs compared to using the cv2.findHomography function
- There will be two outputs, as shown below; you should generate AR_out and AR_warped figures and show them in your report.

AR_out	AR_warped
	

Part A.2: Required for Section-5934 (Bonus +5 for Section-4934)

There is a basic warping function implemented for you (in `libs_hh3/Homography.py`)

```
def warp_and_augment(im_logo, im_dst, H)
```

You will see that it generates somewhat noisy output in some pixels. There are a few better ways to implement this function for much more accurate warping.

Implement a better warping function! Write the main ideas/intuitions of your algorithm and show the comparative results in your report.

Part B: Camera Calibration

In this part, you will calibrate a camera: cell-phone or any other camera that you have access to. Refer to Lecture-5 and find the **intrinsic camera parameters** as follows:

- Print a checkerboard and place it on a wall (a sample pattern.pdf is provided for you)
- Use any of the following libraries to calibrate your camera:
  - **ROS**, **OpenCV**, **CalTech Matlab code**
  - Or any other library of your choice
- Provide the intrinsic matrix **K** and your camera model in your report.
- Provide information about your camera (model, configurations, etc).
- You can check the lens focal length and other information in the 'properties' of captured images and/or the manual/datasheet.

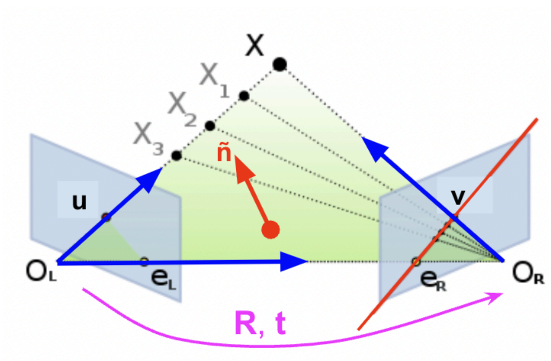
Part C: Perspective Geometry

Refer to the Epipolar geometry concepts in Lecture-5

Essential Matrix: $E = t \times R$

Fundamental Matrix: $F = K_R^{-T} E K_L^{-1} \equiv K^{-T} E K^{-1}$

- Relates **u** (left image point) and **v** (right image point)
- The constraint: $v^T F u = 0$



$$P_L = \mathbf{K}_L [\mathbf{I} \;\; \mathbf{0}] \quad \text{and} \quad P_R = \mathbf{K}_R [\mathbf{R} \;\; \mathbf{t}]$$

$$e_L = K_L [\mathbf{I} \;\; \mathbf{0}] \begin{bmatrix} -R^T t \\ 1 \end{bmatrix} = -K_L R^T t$$

$$e_R = K_R [R \;\; t] \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} = K_R t$$

Part C.1: As you know, a 2D point **u** (in the left image) maps to a line ***L*** on the right image. What is the equation for line ***L***?

Part C.2: The essential matrix is defined as **E = t x R**. Prove that given the 3x3 matrix **E**, we can recover the translation from the nullspace of $\mathbf{E}^T$.

That is, prove that `t = ± nullspace(E`^{T}`)`

For both the C.1 and C.2 parts, show your derivations or calculation steps in the report.

Part D: Implement 2D Motion Planning Algorithms (Python)

Refer to the path planning and motion planning concepts in Lecture-7. We discussed the family of Bug algorithms for source-to-destination 2D navigation. Since most of you will be using the TurtleBot or the PuppyPI dogs for 2D navigation, learning to implement these algorithms will benefit you significantly. We will consider the most popular algorithms- Bug0, Bug1, and Bug2 - in Python.

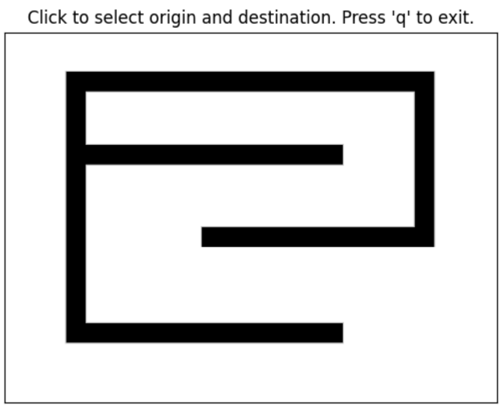
Check out the Part_D sub-folder, where you will find the files **bug0.py**, **bug1.py**, and **bug2.py**. The relevant libraries and data folders are there as well. Everything related to the **Bug1 algorithm** is implemented for you so that you can track down the functions and relevant notations used in the code base. For a start, just run

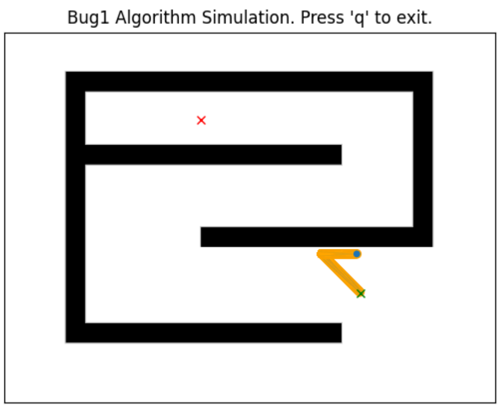
```
python3 bug1.py --map_path=data/maze3.png
```

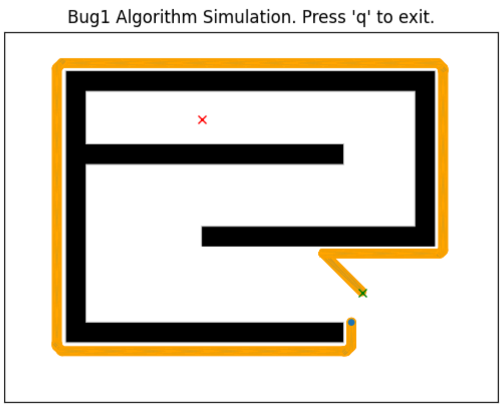
It will load the maze3 map for you and ask you to choose a source and destination pair; you click on the map to choose two points, and it will start executing the motion planner.

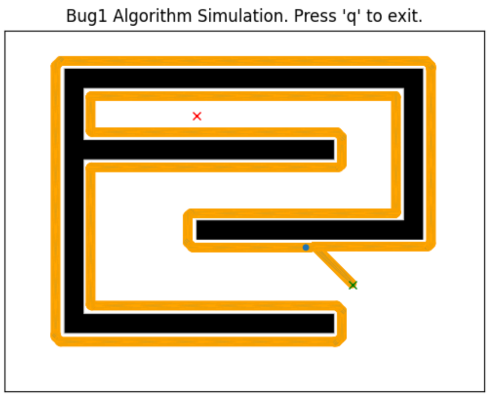
Some snapshots are shown below; once the simulation window appears, you will click on it to define the **source** and **destination**, then the *point robot* will plan its path according to the Bug1 algorithm and execute it. Remember the steps of Bug1, covered in Lecture-7: (1) head towards the goal in a straight line; (2) circumnavigate around obstacles along the way to find the closest leave point; (3) go to the closest leave point and repeat this process to reach the destination.

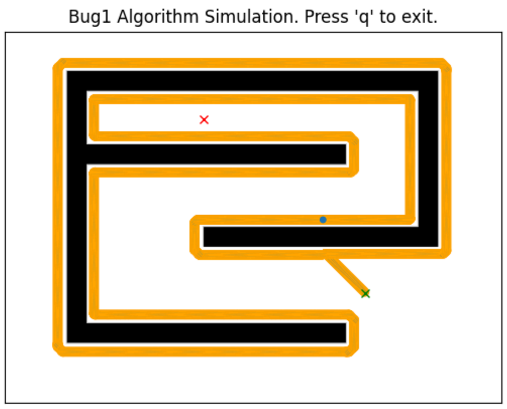
A particular sequence would look like the following: (note that if the maze is complex, it will take a long time to traverse multiple obstacles, so do your initial testing on easier mazes (like maze #1).

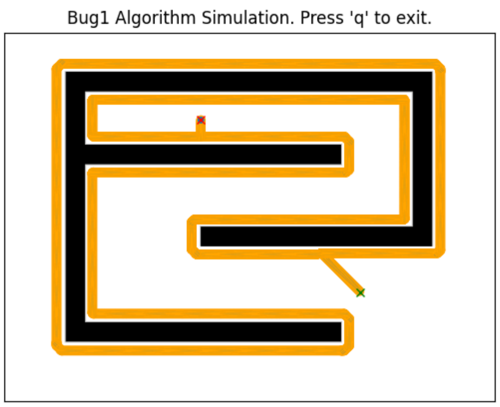












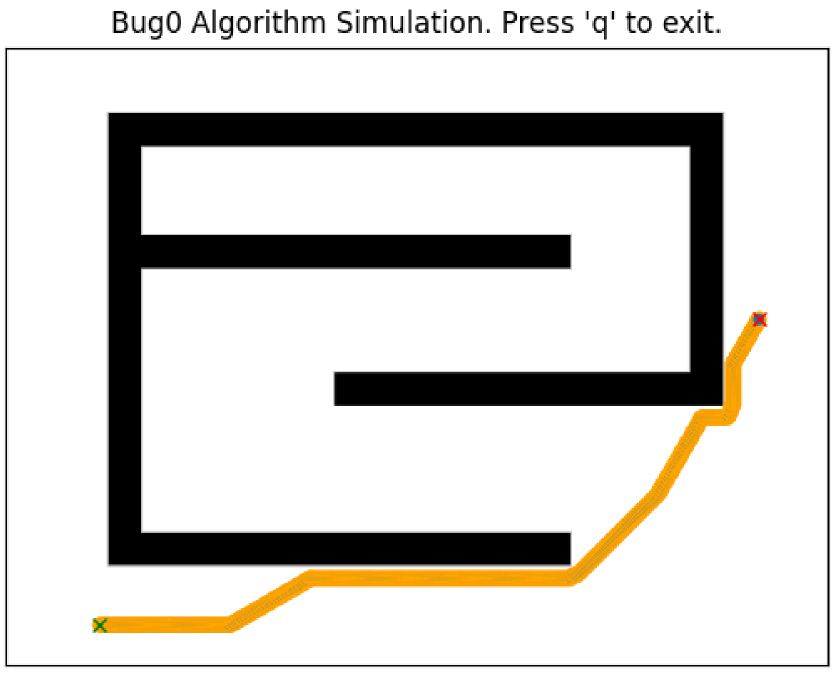
Your task for this homework includes:

1. **Understand the three states of the Bug1 algorithm and how it was programmed**
    - Follow the functions calls in the Bug1.py - which implements the simplest form of if-else state transition, should be easy to follow.
    - Check the Simulation library function that it calls, namely
        - `sim.set_heading`
        - `sim.navigability`
        - `sim.move_forward`
        - `sim.distance`
        - `sim.follow_boundary`

2. **Implement Bug0 algorithm**
    - Not that it is an even simpler algorithm than Bug1. It has two states:

```
state_dict = {
        0: 'go to destination',
        1: 'follow wall',
    }
```

    - Complete the rest of Bug0.py code
    - Hint: You will need to implement one more function in the simulator
        - `sim.rotate`
    - A sample output for a given source-destination pair would look like this:



3. Implement Bug2 algorithm

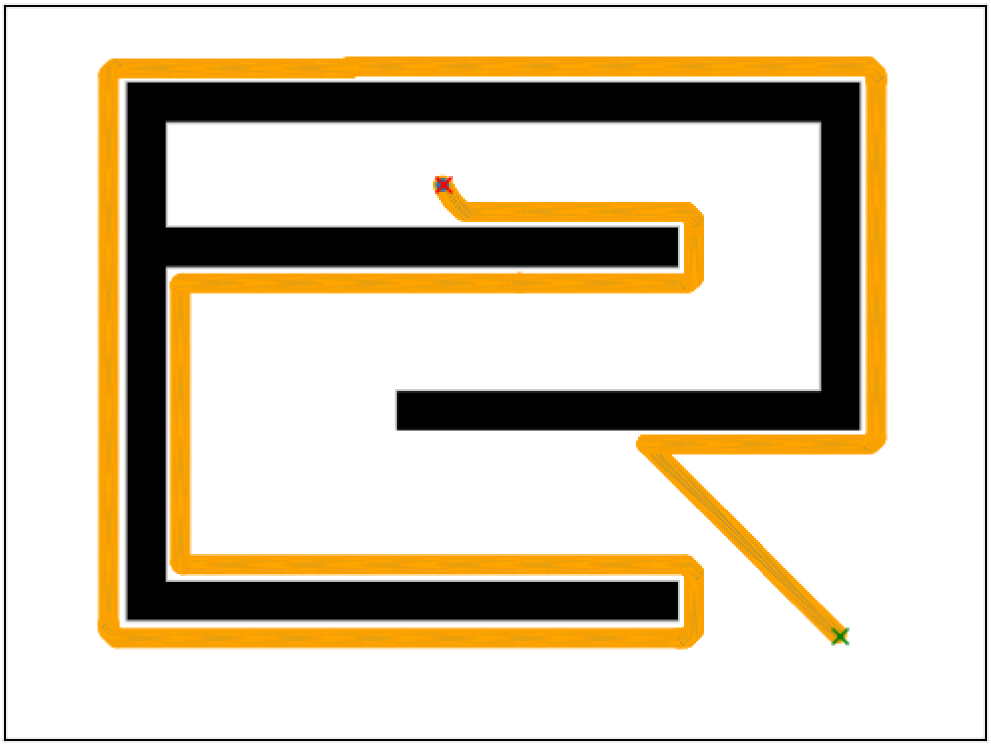
- Bug2 also has two states:

```
        state_dict = {
                0: 'go to destination',
                1: 'follow wall',
            }
```

- Complete the rest of Bug2.py code
- Hint: You will need to implement one more function in the simulator
  - `sim.mline_crossing`

- A sample output for a given source-destination pair would look like this:

Bug2 Algorithm Simulation. Press 'q' to exit.



Other instructions:

- *Feel free to implement any other supporting functions if you need, but do not create any more Python files. Just add your new defines to the existing library file.*
- ***Bonus point (up to +10):*** *if you can implement this Part_D in a ROS2 package*