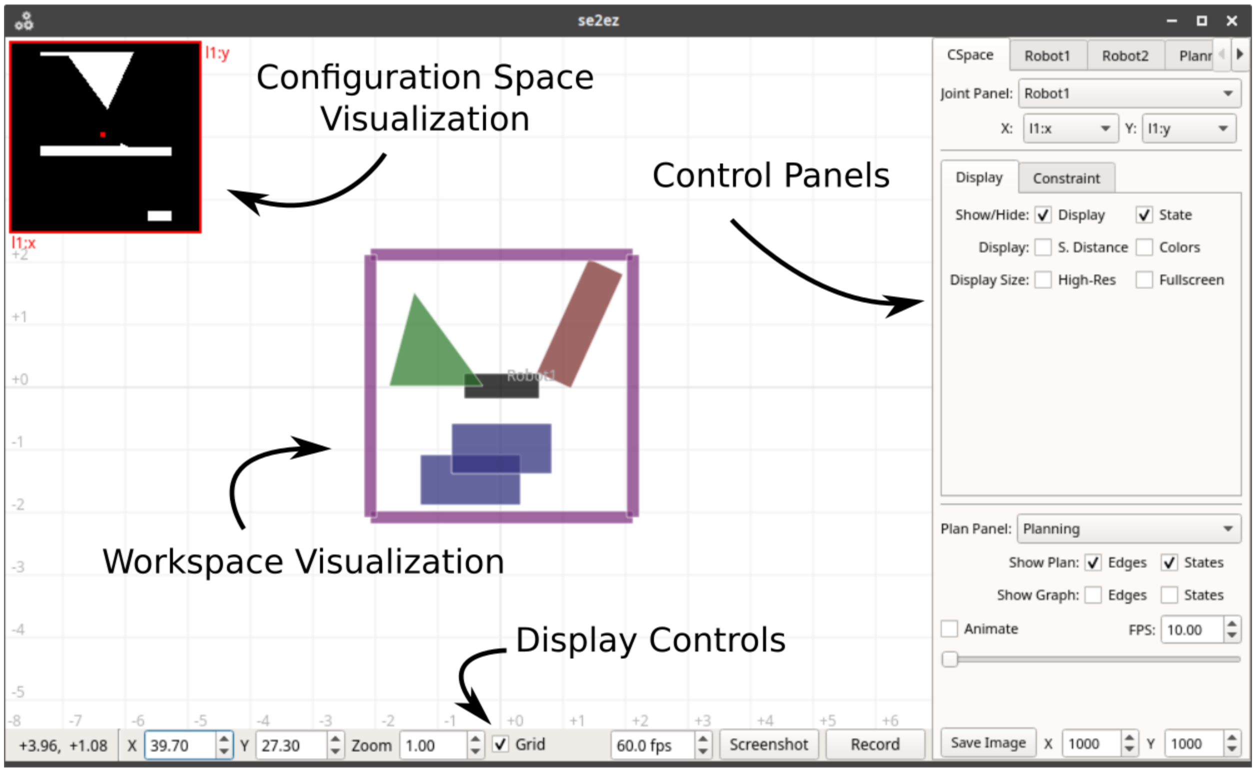
Algorithmic Robotics
COMP/ELEC/MECH 450/550
Project 2: Configuration Spaces

The goal of this project is to provide a deeper understanding of the relation between a robot and its configuration space (or “C-Space”). The mapping from the robot and environment (a collection of rigid bodies in either 2D or 3D) to the abstract configuration space (possibly high-dimensional, non-Euclidean) can be unintuitive—how do the obstacles in the workspace result in obstacles in the configuration space? We are providing a script, `se2ez-gui`, available on the virtual machine that provides live visualization of the configuration space for planar robots. You will be editing robot description files and viewing their resulting configuration spaces, as well as profiling the performance of a few motion planners in the scenes you create.

The `se2ez-gui` Program

First, extract the provided Project 2 files. Open a terminal into the Docker container, navigate to the now extracted `Project 2` folder, and type `se2ez-gui class/planar.yml`. You should see the following GUI appear:



Joint Panel: Robot1
X: l1:x Y: l1:y
Show/Hide: ✓ Display ✓ State
Display: ☐ S. Distance ☐ Colors
Display Size: ☐ High-Res ☐ Fullscreen
Plan Panel: Planning
Show Plan: ✓ Edges ✓ States
Show Graph: ☐ Edges ☐ States
☐ Animate FPS: 10.00
+3.96, +1.08 X 39.70 Y 27.30 Zoom 1.00 ✓ Grid 60.0 fps
Save Image X 1000 Y 1000

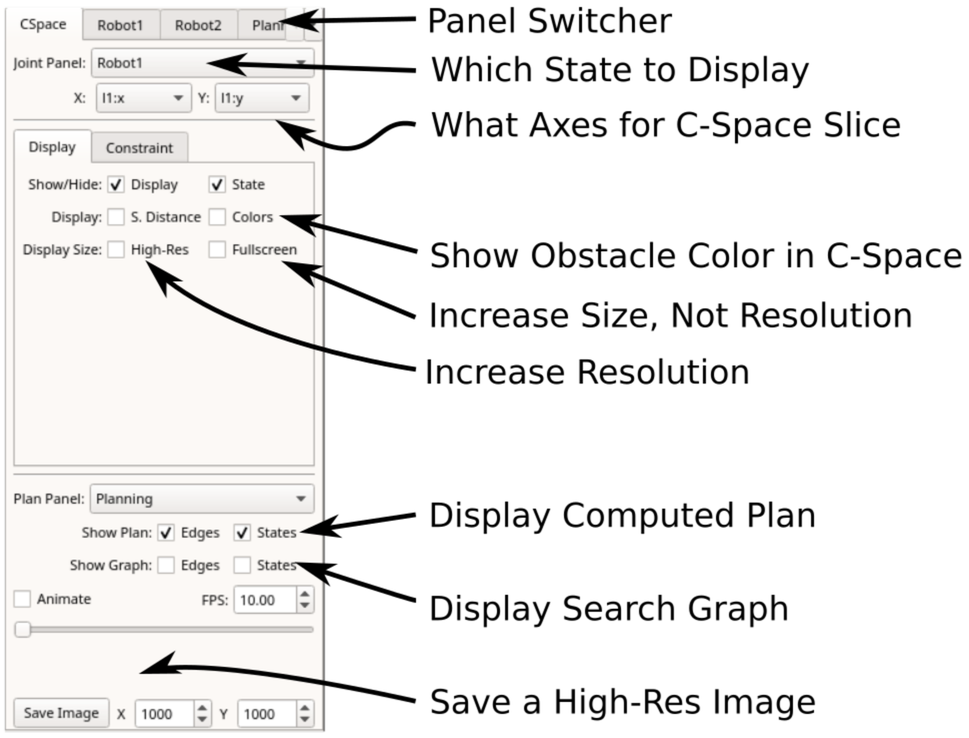
You will be using this GUI to explore various robot and environment configurations to get a better understanding of the configuration space. The GUI has components to visualize the configuration space of a robot, visualize two states of the robot, and visualize motion planning results. There are four main components in the GUI:

1. *Control Panels*: To the right are a variety of options to control visualization, as well as execute subroutines such as planning. Notice at the top there are four tabs: “CSpace” for configuration space visualization options, “Robot1” for options for one robot state and “Robot2” for the other, and “Planning” for motion planning. We call the panel currently selected the “active” panel. The specifics of each panel will be discussed later.
2. *Workspace Visualization*: This is the main visualization component which displays all rigid bodies in the workspace. In this program, there are two robot states visualized—“Robot1” and “Robot2”, denoted by the floating text over the robot. If a motion plan is executed, the motion plan will also be visualized here. “Mouse 1”, or left clicks, pans the workspace around. “Mouse 2”, or right clicks, controls the active panel. Scrolling zooms in and out. If the “CSpace” panel is selected, the robot’s state can be set via right-clicking in the configuration space visual. If the “Robot1” or “Robot2” panels are selected, the robot’s state is set via inverse kinematics to wherever the mouse is in workspace.
3. *Configuration Space Visualization*: This is a visualization of the robot’s configuration space. The red dot is the current state of the robot. The X- and Y-axes are two configuration dimensions of the robot, set by the configuration space panel.
4. *Display Controls*: Controls for the current display, including current workspace X, Y, and zoom. The screenshot button will take a screenshot of the current display and save it to the directory where `se2ez-gui` was run.

Try right-clicking in the workspace visualization, specifically within the purple box: you should see the robot, the grey rectangle, moves where you click. Notice that this moves “Robot1,” and not the “Robot2” panel. If you want to move “Robot2,” select the “Robot2” panel, then right click in the workspace. Moreover, note how the configuration space changes with the rotation—the configuration space shown is just a 2D “slice” of the 3D configuration space on the X- and Y-axes of the robot (this can be changed, discussed later).

You should also try out the other planar arm robots included with the project files: `se2ez-gui class/2-link.yml` and `se2ez-gui class/arm.yml`

Configuration Space Panel

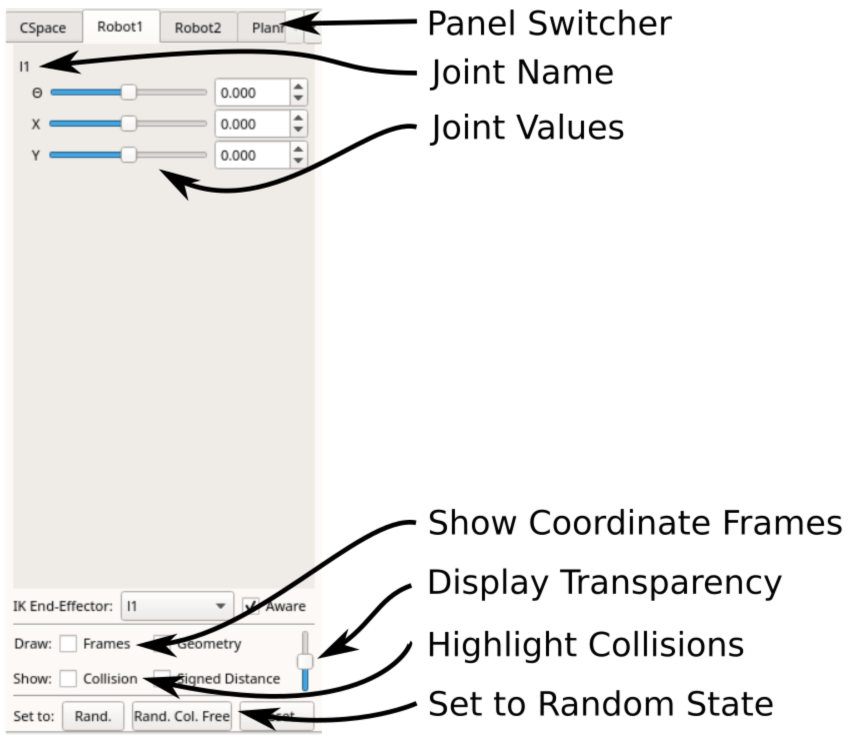


Joint Panel: Robot1
X: l1:x Y: l1:y
Show/Hide: [✓ Display] [✓ State]
Display: S. Distance Colors
Display Size: High-Res Fullscreen
Plan Panel: Planning
Show Plan: [✓ Edges] [✓ States]
Show Graph: Edges States
Animate FPS: 10.00
X 1000 Y 1000

The first panel visible is the configuration space panel, labeled as "CSpace." This controls the display settings for the configuration space visualization in the upper left. The important settings are as follows, from top to bottom as indicated in the figure:

1. *State Display*: Chooses which robot state panel to show as the red dot in the configuration space.
2. *Axes*: Selects which two configuration space dimensions to show. Displays a "slice" of the configuration space along these two dimensions. The rest of the robot's configuration variables are kept the same in this slice.
3. *Obstacle Color*: Colors the configuration space obstacles identical to their workspace counterparts if true. Useful for understanding the mapping between workspace and configuration space. We **highly** recommend using this feature.
4. *High-Res*: Generates a higher resolution image of the configuration space slice. Takes longer to update due to needing more computation.
5. *Fullscreen*: Increases the size of the configuration space display without increasing the resolution.
6. *Show Plan*: If a motion plan is computed with the "Planning" panel, result will also be displayed in configuration space as an orange path. Note that plans are shown "projected" onto the current configuration space slice—if you have a robot with more than 2 degrees of freedom, it will appear as if the path goes through an obstacle as some "hidden" variable is changing.
7. *Show Graph*: Shows the search graph generated by the planner. Both edges and states can be shown.
8. *Save Image*: Saves a high-resolution image of the current configuration space slice.

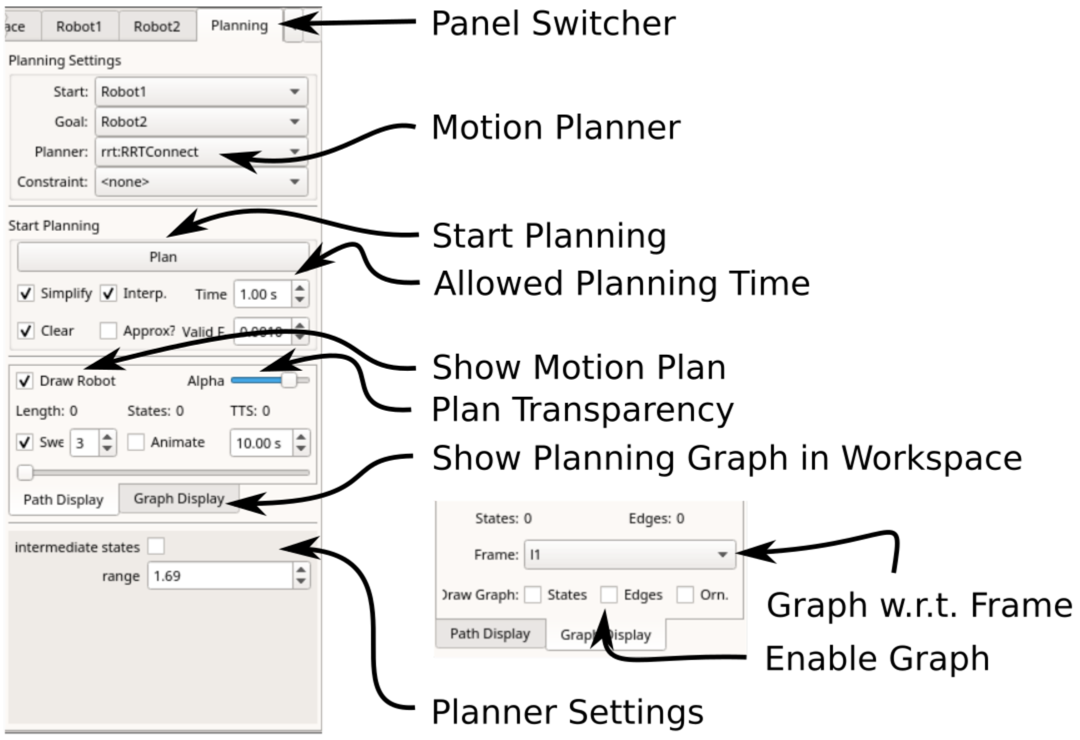
Robot State Panels



The second and third panels control two robot states (configurations) of the robot. These are “Robot1” and “Robot2.” As stated before, right clicking in the workspace while one of these panels is active will use *inverse kinematics* to set the location of the robot to the clicked point. If the robot does not move where you clicked, it means it could not find a valid solution that placed the robot where you clicked—try somewhere else. These panels have the following important options, from top to bottom as indicated in the figure:

1. *Joint Name*: Each joint on the robot has a name, and this name is displayed before that joint’s value controls.
2. *Joint Values*: The values for the controllable elements of the joint, displayed as a slider and an input box.
3. *Transparency*: Sets the transparency of the robot state display.
4. *Coordinate Frames*: Displays the local reference frames in the robot and environment for the current configuration.
5. *Highlight Collisions*: Highlights the colliding elements of the robot and environment red when a collision occurs.
6. *Set to Random State*: Uniformly samples a random robot state.

Planning Panel



The final panel has controls for computing and displaying motion plans. To start out, simply maneuver "Robot1" to a different configuration than "Robot2" by right-clicking somewhere in the workspace. Do the same by then selecting the "Robot2" panel and manuevering "Robot2" to somewhere else. Then, switch back to the planning panel and click the large "Plan" button (pointed out by "Start Planning"). You should see the swept volume of a plan that carries the robot from "Robot1" to "Robot2" now displayed in the workspace and configuration space. This panel has the following important options, from top to bottom as indicated in the figure:

1. *Motion Planner*: Select which motion planner to use here. This is similar to OMPL.app.
2. *Start Planning*: Press this button to start motion planning! You should see planner output in the terminal you launched `se2ez-gui` from.
3. *Allowed Planning Time*: Configure the amount of time given to the planner here.
4. *Show Motion Plan*: Deselect this to stop displaying the found motion plan. Note the "Swept" and "Animate" boxes at the bottom. If "Animate" is checked, uncheck "Swept" to display an animated version of the motion plan.
5. *Plan Transparency*: Change the display transparency of the states shown in the path display.
6. *Show Planning Graph in Workspace*: Switch to this tab to show options for displaying the motion planning graph in the workspace, with respect to some frame of the robot.
7. *Planner Settings*: Same as in OMPL.app, the various setting each planner has.

Robot Description Format

se2ez-gui loads robots specified by a simple YAML file. All files are specified as a list of coordinate frames, each of which has some set of rigid bodies attached. Before more specific documentation, use the following example files for a rigid body that can translate and rotate in the plane (similar to the provided class/planar.yml file) and a planar manipulator with two links (similar to the provided class/2-link.yml file).

Listing 1: Example Rigid Body YAML File

```yaml
robot:                 # All files must begin with the "robot" field.
  # Below is a list of all coordinate frames on the robot.
  - name: "l1"         # Every coordinate frame needs a unique name.
    joint: "flying"    # The joint type that controls the frame (more later)
    limits:            # Some joint types need position limits specified
      upper: [ 2,  2]    # For "flying" joints, need X- and Y- limits
      lower: [-2, -2]
    geometry:          # Specify rigid body geometry attached to frame
      - type: "box"    # Type of geometry. Types are specified below
        dimensions: [0.6, 0.2] # X- and Y- dimensions of the box
  # An example specification of an obstacle
  # Note that by default the joint type is "fixed"
  - name: "obstacle_1"
    tip: [1.25, 1., -2]
    geometry:
    - type: "box"
      dimensions: [1.0, 0.3]
      color: [0.5, 0.2, 0.2, 1]
```

Listing 2: Example Two-Link Planar Manipulator YAML File

```yaml
# An example two-link manipulator robot
robot:
  - name: "base"   # Unique name
    joint: "fixed" # By default, the joint type is "fixed" (a fixed transform)
    parent: "root" # There is an implicit frame "root" that is the world frame.
                   # If not specified, assumes "root"
    tip: [0, 0, 0] # Transform applied _after_ applying the joint transform.
                   # The format is [X, Y, theta], where theta is in radians.
                   # Transform is applied in local reference frame.
    geometry:
      - type: "circle"
        dimensions: [0.1]
  - name: "link_1"
    joint: "continuous"
    parent: "base"      # Parent is the prior defined frame.
    tip: [1, 0, 0]
    allowed: ["base"]   # Frames that this frame is allowed to collide with.
                        # Usually necessary for chain manipulators
    # Multiple rigid bodies can be associated with a frame
    geometry:
      - type: "box"
        dimensions: [0.5, 0.1]
        offset: [-0.5, 0, 0]      # Geometry can positioned at a fixed offset away
    .
      - type: "circle"
        dimensions: [0.1]
  - name: "link_2"
```

```
    joint: "continuous"
    parent: "link_1"
    tip: [1, 0, 0]
    allowed: ["link_1"]
    geometry:
      - type: "box"
        dimensions: [0.5, 0.1]
        offset: [-0.5, 0, 0]
```

There are many joint types available to choose from:

Listing 3: Joint Types

```
# Fixed transform, cannot move.
# If not specified, joint is assumed fixed.
joint: "fixed"
---
# SE(2) (rotation and translation in the plane) with rotation limits
joint: "float"
limits:
  upper: [1, 1, 3.14]
  lower: [-1, -1, -3.14]
---
# SE(2) (rotation and translation in the plane) without rotation limits
joint: "flying"
limits:
  upper: [1, 1]
  lower: [-1, -1]
---
# R^2 (translation in the plane)
joint: "translate"
limits:
  upper: [1, 1]
  lower: [-1, -1]
---
# Prismatic joints. Extend in the X-axis.
joint: "prismatic"
limits:
  upper: 1
  lower: -1
---
# Revolute joints without rotation limits
joint: "continuous"
---
# Revolute joints with rotation limits
joint: "revolute"
limits:
  upper: 3.14
  lower: -3.14
```

There are also a variety of rigid body geometries you can use, with placement and display options:

Listing 4: Geometry Types

```
geometry:
  - type: "circle"     # Circle
    dimensions: [1]    # Radius
  - type: "box"        # Rectangular Prism
```

```
    dimensions: [1, 1] # X- and Y- radii
  - type: "convex"     # Convex polygons
    points:
      - [3.91, 3.74]   # Vertices
      - [1.61, 1.07]
      - [2.08, 1.48]
  - type: "simple"     # Simple (no holes) polygons
    points:
      - [3.91, 3.74]   # Vertices
      - [1.61, 1.07]
      - [2.08, 1.48]
      - [2.95, 1.60]
      - [4.41, 3.03]
---
geometry:
  - type: "circle"
    dimensions: [1]
    offset: [0, 0, 0]   # Offset transfrom from frame's tip (optional)
    color: [0, 0, 0, 1] # Geometry color in (r, g, b, alpha)
```

Finally, for the final exercise of this project you will need to create "named" configurations of the robot, which is another top-level YAML entry, placed underneath "robot":

Listing 5: Named Configurations

```
robot:
  ...
---
states: # A list of name states
  - name: "start"   # Name of the state
    configuration:  # List of all configuration variables
      - frame: "l1" # Name of the frame (joint) specified
        value: [-1.68, -1.16, 1.571] # Joint value (X, Y, theta)

  - name: "goal"    # "start" and "goal" are used for benchmarking
    configuration:
      - frame: "l1"
        value: [0, 1.44, 0]
---
states: # An example for a two-link revolute manipulator
  - name: "start"
    configuration:
      - frame: "link_1"
        value: [1.257]
      - frame: "link_2"
        value: [-0.251]

  - name: "goal"
    configuration:
      - frame: "link_1"
        value: [1.885]
      - frame: "link_2"
        value: [-1.571]
```

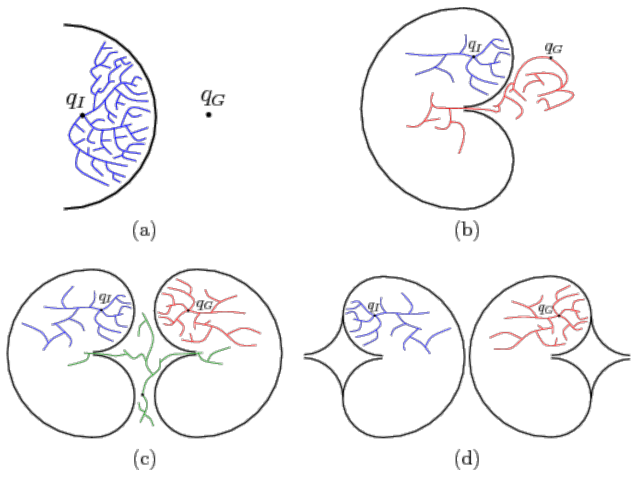


Figure 1: An example of hard planning environments, specifically "bug traps," named for the difficulty that the bug algorithms would have. Notable features include an expansive starting area that requires navigating a very narrow passage to escape. **b)** shows the classic single bug-trap.

Provided Programs

Compile and execute the C++ programs in the `src` folder. To do this, simply extract the provided Project 2 files, open a terminal into the Docker container, navigate to the `Project 2` folder, and type `make`. You should now have two programs: `se2ez-plan` and `se2ez-benchmark`.

- `se2ez-plan`: This program executes a motion planning problem specified in a provided YAML file. The program looks for the named configurations "start" and "goal" in the YAML file (see above). You can run this program by executing `./se2ez-plan <filename.yml>`. You should use this script to validate that your YAML file's planning problem is correct before benchmarking.
- `se2ez-benchmark`: This program benchmarks a motion planning problem specified in a provided YAML file. The program looks for the named configurations "start" and "goal" in the YAML file (see above). You can run this program by executing `./se2ez-benchmark -f <filename.yml>`. You can specify the number of runs by using the `-r <runs>`, and the amount of time for planning by using `-t <time in seconds>`.

Deliverables

This project must be completed in pairs. You and your partner need only provide one submission. **Submissions are due Thursday Sept. 29 at 1pm**.

To submit your project, create a compressed archive of all your YAML files with the following naming scheme: `Project2_<your NetID>_<partner's NetID>.zip`. Submit this archive to Canvas. We provide the script `submit.sh` that will automatically create the archive for you, and check if all expected files are present. You can run the script by typing `./submit.sh <your netID> <your partner's netID>`.

In addition to the archive, you are required to submit a report. Make sure to include both you and your partner's name on the report. For *all* exercises, you must include a figure that shows the robot's workspace. This report should be a separate file from the archive. The report should be no longer than 6 pages in PDF format, including figures. Try to optimize space in your document and also make an effort to be concise and accurate in your writing. Your report should contain the following information:

1. **(5 points)** Design an environment for a robot that contains an "unintuitive" transformation from workspace to configuration space. This is based on your definition of "unintuitive"—what is surprising about the configuration space mapping, and why? Include an image of the robot's configuration space. Name your YAML file `unintuitive.yml`.
2. **(5 points)** Design an environment for a two-link manipulator with 1 revolute joint and 1 prismatic joint (either order works). The environment must has two *disconnected components* in its configuration space. Use at most two circular obstacles. Include an image of the robot's configuration space and identify the two components. Name your YAML file `disconnected1.yml`.
3. **(5 points)** Design an environment for a two-link revolute manipulator that has two *disconnected components* in its configuration space. You can use no workspace obstacles, all geometry must be on the robot. Recall that each frame on the robot can have multiple rigid bodies attached. Include an image of the robot's configuration space and identify the two components. Name your YAML file `disconnected2.yml`.
4. Design a series of "bug-trap" environments (see Figure 1). Include a figure of both robot's configuration spaces and identify the bug-trap.
   (a) **(10 points)** A bug-trap for a robot that can translate and rotate in the plane. Name your YAML file `bugtrap1.yml`.
   (b) **(15 points)** A bug-trap for a two-link manipulator robot with 1 revolute joint and 1 prismatic joint (either order works). Name your YAML file `bugtrap2.yml`.
5. Design a series of environments that have at least two path homotopy classes. One of these two classes must be a "short" path, but is significantly harder for a planner to find. The other class must be a "longer way around," but is significantly easier for the planner to find. For both, include a figure of this robot's configuration space and identify the two homotopy classes.
   Include benchmarking results to support your claims. We recommend using the "simplified solution length" value—use the advanced options to display the plot as a cumulative distribution function. You should see an earlier bump in the curve corresponding to the few number of solutions that return this more difficult homotopy class. This value corresponds to the path length *after* simplification routines have been applied, and thus the output path from the planner will be closer to a local minimum for the homotopy group.
   (a) **(10 points)** An example for a robot that can translate and rotate in the plane. Name your YAML file `homotopy1.yml`.

(b) **(15 points)** An example for a two-link revolute manipulator. Name your YAML file `homotopy2.yml`.

6. **(15 points)** Design an environment that contains a significantly hard (but still solvable) planning problem for a 2-D mobile manipulator robot. The robot has a two-link revolute arm mounted on its base which can rotate and translate in the plane. Name your YAML file `difficult.yml`. Include benchmarking results to support your claim of difficulty. Identify what portions of this robot's configuration space make this problem difficult—include a figure of the configuration space with the difficult areas identified.
7. **(15 points)** Design an environment that contains a significantly hard (but still solvable) planning problem for a robot with many (greater than two) degrees-of-freedom. You can only use revolute and prismatic joints to build this robot. Name your YAML file `highdof.yml`. Include benchmarking results to support your claim of difficulty. Identify a portion of this robot's configuration space make this problem difficult—include a figure of a slice of the configuration space with the difficult areas identified.
8. **(5 points)** Rate the difficulty of each exercise on a scale of 1–10 (1 being trivial, 10 being impossible). Give an estimate of how many hours you spent on each exercise, and detail what was the hardest part of the assignment. Also, rate the difficulty of using the `se2ez-gui` application and description format. If you have any thoughts on what could be made easier or features that would be useful, please include them here. **Additionally**, as you completed the project in pairs, describe your individual contribution to the project.

Take time to complete your write-up. It is important to proofread and iterate over your thoughts. Reports will be evaluated for not only the raw content, but also the quality and clarity of the presentation.