CS 4610/5335: Robotic Science and Systems (Spring 2023) Robert Platt
Northeastern University

HW 2: Motion Planning in CSpace

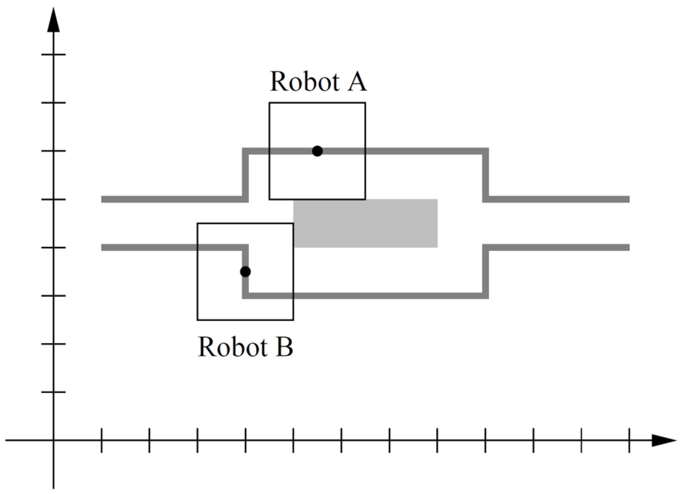
Please remember the following policies:

- Submissions should be made electronically via the Canvas. Please ensure that your solutions for both the written and/or programming parts are present and zipped into a single file.
- Solutions may be handwritten or typeset. For the former, please ensure handwriting is legible.
- You are welcome to discuss the programming questions (but *not* the written questions) with other students in the class. However, you must understand and write all code yourself. Also, you must list all students (if any) with whom you discussed your solutions to the programming questions.

We recommend that you first familiarize yourself with MATLAB and the Robotics Toolbox by following these steps:

- Install MATLAB R2020b. Visit the following article for Northeastern-specific instructions:
  `https://service.northeastern.edu/tech?id=kb_article&sys_id=68c93fd6dbf37bc0c5575e38dc961918`
  - You may have to log in (top right) and re-enter the URL to view the page's content.
  - If you have an older version of MATLAB, we recommend installing the R2019b version to ensure better support from the teaching staff.
  - If you have sufficient space on your computer, we recommend installing all of the toolboxes as well. At a minimum, you should install the Robotics System Toolbox, the toolboxes listed on the following page, and the Optimization and Symbolic Math toolboxes.
    `https://www.mathworks.com/support/requirements/robotics-system-toolbox.html`
    You can always choose to install additional toolboxes later if you are uncertain.
- If you need an introduction or refresher on MATLAB, see the "MATLAB Resources" section of the "Resources" page on Piazza for slides and practice questions (with answers) from a short but intensive introductory course.
- Install Peter Corke's Robotics Toolbox (version 10.4). This is different from the official MATLAB Robotics System Toolbox installed in the previous step.
  `http://petercorke.com/wordpress/toolboxes/robotics-toolbox`
  - The simplest way to install this is from the `.mltbx` file:
    `http://petercorke.com/wordpress/?ddownload=778`
    To install, open the downloaded file within MATLAB. Then run `rtbdemo` to check that it is working.
  - Version 10.x of the toolbox is the only version compatible with the second edition of the Robotics, Vision and Control textbook, which is the version we are using.
  - You may also want to download the toolbox manual as a code reference:
    `http://petercorke.com/wordpress/?ddownload=343`
- You will inevitably encounter errors. Here are some debugging tips:
  - The MATLAB documentation is very extensive and available both online and offline. To look up a function within the interactive session, type `help <function>` (e.g., `help SE3`). The description often contains links to further documentation and related functions.
  - Use `disp` liberally to print out variables and other debugging information.
  - If you are used to coding in MATLAB purely with matrices, note that the Robotics Toolbox defines many entities as classes and objects (e.g., `SE3`, `SerialLink`, etc.). Know the difference, and do not fret when you encounter type conversion issues – fixing them is similar to debugging dimension mismatch issues.
  - Tables 2.1 and 2.2 in the textbook (data type conversions) are extremely useful.

Configuration Space and Motion Planning



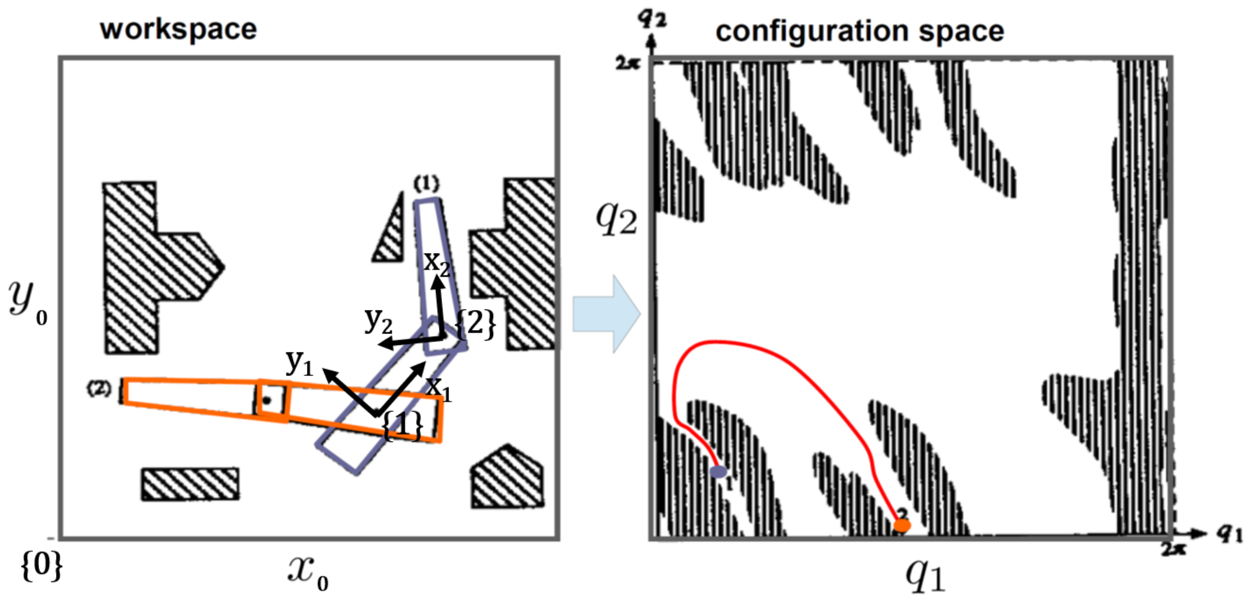
C0. **2 points.** Consider the diagram above. Two square robots A and B operate in a 2-D workspace. The two robots do not rotate, and each moves on a fixed track, so that its center remains on the solid gray line shown in the figure. The robots must move so as not to collide with each other. The diagram is to scale, with each tick denoting a distance of one unit.

(a) Even though there are two robots both moving in a 2-D workspace, we can still use a 2-D configuration space to represent the above system. Specify what the axes of our configuration space correspond to in the workspace, what the axis limits are, and what configuration the above diagram depicts.

(b) Draw the configuration space. For each configuration-space obstacle, label the coordinates of the vertices, and sketch the workspace configuration corresponding to each. Since the workspace is symmetric, only makes sketches for the left side of the workspace.

Approach: Plan in “configuration space”

Convert the original planning problem into a planning problem for a single point.



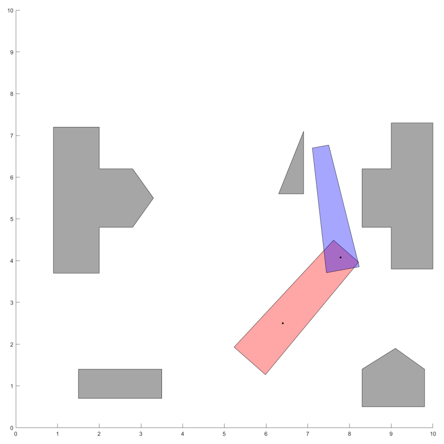
<u>Original problem</u>
– plan path for robot arm

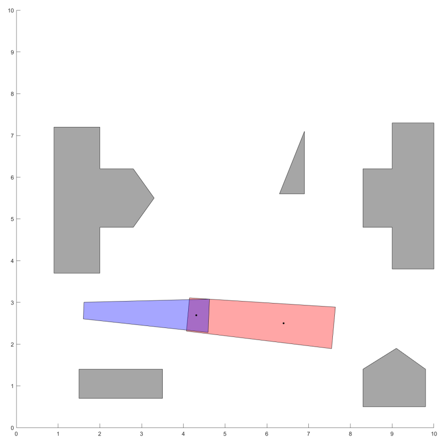
<u>Equivalent problem:</u>
– plan path for a point

In the remainder of this section, we will revisit the 2-DOF 2-link rotational planar robot shown above that we considered in lecture. The arm is attached to a table surface that we are viewing from a top-down perspective. There are obstacles on the table as shown by the shaded regions. The goal is move the arm from start configuration (1, purple) to the goal configuration (2, orange), without colliding into any obstacles. Assume that the illustrated workspace is 10 units wide and 10 units high; the origin of frame $\{0\}$ is at the bottom-left corner, with x_0 and y_0 axes as shown. The first link of the arm is attached to the table at $(6.4, 2.5)$, the origin of frame $\{1\}$. Link 2 is attached to link 1 at $(2.1, 0)$ with respect to frame $\{1\}$, the origin of frame $\{2\}$. The configuration of the arm is given by $q = (q_1, q_2)$, where q_1 is the angle between x_0 and x_1, and q_2 is the angle between x_1 and x_2. Both joints may rotate between 0 and 2π radians. For example, the start configuration (1, purple) is $q_{\text{start}} = (0.85, 0.9)$, and the goal configuration (2, orange) is $q_{\text{goal}} = (3.05, 0.05)$.

Download the starter code from Piazza (`hw2.zip`). In this file, you will find:

- `hw2_cspace.m`: Main function. Call `hw2_cspace(<questionNum>)` with an appropriate question number (1–7) to run the code for that question. Do not modify this file! (Feel free to actually make changes, but check that your code runs with an unmodified copy of `hw2_cspace.m`.)
- `C1.m` – `C7.m`: Code stubs that you will have to fill in for the respective questions.
- `q2poly.m`: Code stub that you will have to fill in, helper function for various questions.
- `plot_obstacles.m`: Helper function to draw workspace obstacles defined in `hw2_cspace.m`.


C1: Start configuration.


C1: Goal configuration.

C1. **2 points.** Plot the robot in the workspace, as shown above. The demonstration code in `C1.m` shows how to plot the robot at the zero configuration ($q = (0, 0)$). You will need to make appropriate transformations to both links' polygons and their pivot points (frame origins). Consider filling in `q2poly.m` first, which is a useful helper function for C1 and future questions. If you provide q_{start} and q_{goal} as input, you should get the figures above.

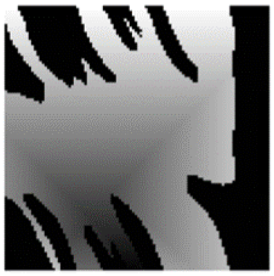
Useful functions: Read the documentation for `polyshape`, a MATLAB class for defining 2-D polygons.

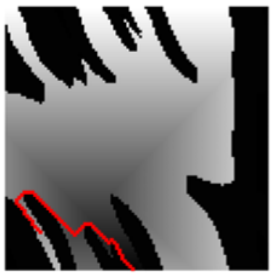
C2. **2 points.** Convert the problem into configuration space by discretizing the configuration space, and checking for collisions at each discrete grid point. Using the specified grid for each axis given in `q_grid`, compute whether the configuration at each point is in collision or not, by intersecting the links' 2-D polygon with the obstacles' 2-D polygons. Assume that the robot is never in collision with itself. The resulting matrix should look similar to the configuration space diagram shown on the slide.

Useful functions: `intersect`

Hint: Future questions rely on the output of `C2.m`, which may take a while to compute. To avoid re-computing it in future questions, we have provided functionality to save it in your MATLAB workspace, and pass it in on future calls:

```
cspace = hw2_cspace(2);
hw2_cspace(3, cspace);
```

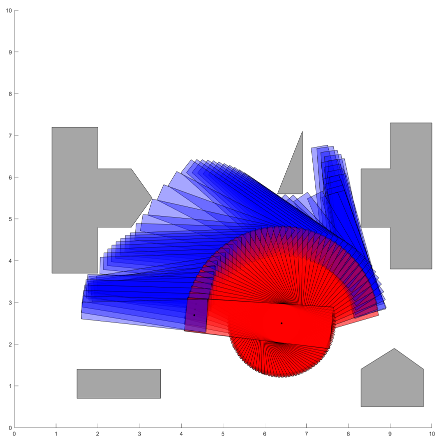

C3: Distance transform from goal configuration.

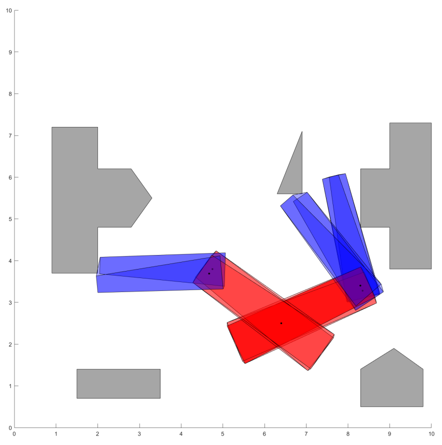

C4: Path from start to goal.

C3. **2 points.** Given a specified goal configuration and the configuration-space grid from C2, compute the distance transform from the grid point nearest to the goal.

C4. **2 points.** Using the distance transform from C3, find a path from the specified start configuration's closest grid point to the goal's grid point. Descend the distance transform in a greedy fashion, breaking ties with any strategy you wish. Diagonal neighbors are allowed.

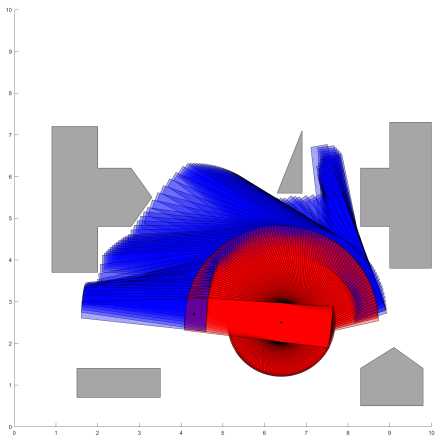
C5. **1 point.** Convert the path in grid point indices, found in C4, into a path in configurations. Remember to include the actual start and goal configurations. This should trigger a visualization similar to the one shown above.


C5: Trajectory from start to goal.


C6: Swept-volume collisions along the path.

C6. **2 points.** Unfortunately, since collisions have only been checked at discrete grid points, we cannot guarantee that the segments between those grid points are collision-free. In fact, the trajectory we found in our implementation of C5 contains three collisions, shown in the right above. These collisions can be detected by considering the *swept volume* between two configurations. The swept volume can be approximated by appropriate convex hulls of the robot links' 2-D polygons. Check if any segments of the trajectory you found in C5 are in collision, plot the violating swept volumes (similar to right diagram above), and return the number of collisions. Depending on how you found your trajectory, it may not actually have any such swept-volume collisions!
Useful functions: `convhull`

C7. **1 point.** Most of the collisions above were caused by planning a path that was too close to obstacles. One simple conservative way to avoid such collisions is to pad the obstacles by a small buffer zone. Pad the obstacles in configuration space by one grid cell (including diagonal neighbors), and verify that the resulting trajectory does not contain any swept-volume collisions.


C7: More conservative trajectory from start to goal, with no swept-volume collisions.