1 Introduction

In this assignment you will be writing a program to help guide the two link robot arm shown in the figure below from one configuration to another while avoiding the objects in the workspace. This assignment will give you some experience working with representations of configuration space. In this example the configuration of the robot is captured by the two joint angles, θ_1 and θ_2, as shown in Figure 1. In the code these are represented in degrees with values between 0 and 360.

The graph on the left of Figure 2 shows a depiction of the robot and the workspace obstacles while the graph on the right shows a plot of the corresponding configuration space and configuration space obstacles. The horizontal and vertical axes in this second figure correspond to θ_1 and θ_2 respectively. If you look carefully at the figure on the left you will note that both the robot and the obstacle are represented by a collection of triangles. This decomposition helps us to decide whether a particular robot configuration would lead to a collision by checking whether any of the triangles in the robot intersect any of the obstacle triangles.

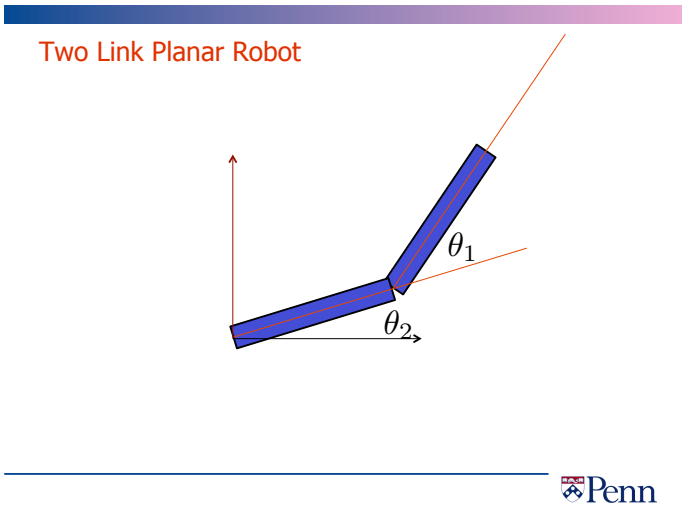

Figure 1: Two Link Planar Arm showing the two joint angles, θ_1 and θ_2, that serve as configuration space coordinates.

2 Assignment: Configuration Space

2.1 Files in this Assignment

`triangle_intersection.m` [*] - The function used to check if the given triangles overlap

`DijkstraTorus.m` [*] - The function to find the shortest path through a given input map using Dijkstra's algorithm

`CollisionCheck.m` - The function to determine if two sets of triangular faces overlap

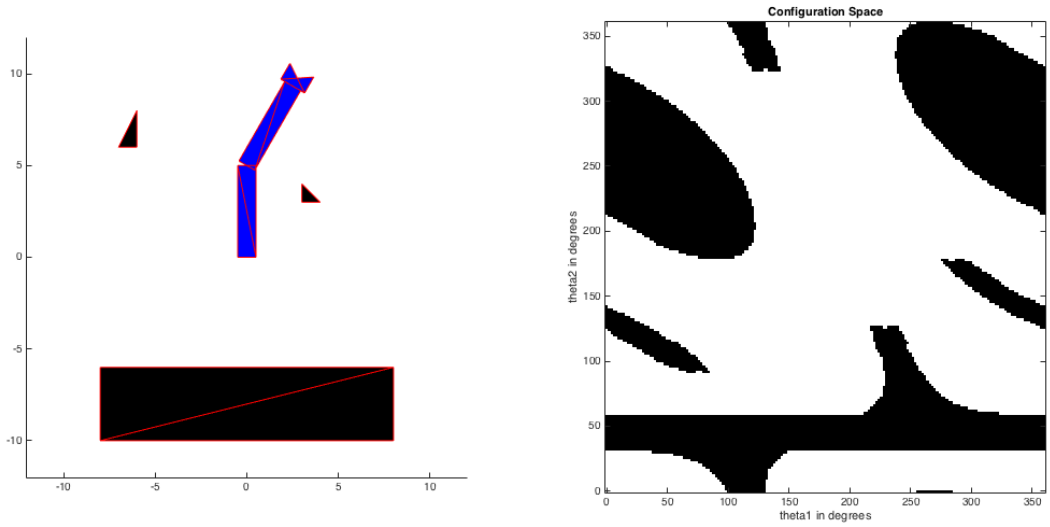

Figure 2: Figure showing the two link robot in an environment with obstacles and the corresponding configuration space.

`TwoLinkRobot.m` - The function to compute the layout of the robot based on the joint angles, θ_1 and θ_2.

`TwoLinkCSpace.m` - The script used to run the simulation

`appendFV.m`, `transformFV.m` - Helper functions for constructing the simulation environment

`evaluate.p` - Code that evaluates your implemented functions

`submit.m` - Script which calls the `evaluate` function for generating submission result

* indicates the files you will need to complete

2.2 Tasks

This assignment is split into two parts. Please finish the first part before moving to the second part.

Part 1 : [15 points] Triangle Intersection

In the code that we provide you will notice a function called `CollisionCheck` which is used to decide whether the obstacle and robot triangles intersect. This function in turn depends upon a function called `triangle_intersection` which you will write. This function should have the following function signature.

```
function flag = triangle_intersection(P1, P2)
% triangle_test : returns true if the triangles overlap and false otherwise
```

The input and output arguments for this function are explained below:

`P1, P2`: a 3 by 2 array (each), describing the vertices of a triangle, the first column corresponds to the x coordinates while the second column corresponds to the y coordinates

`flag`: Return value for the function, set to true if it determines that the triangles do intersect and false otherwise.

One approach to writing this function is to consider all 6 edges, 3 for each triangle, for each edge determine whether it acts as a separating line where all of the vertices of one triangle lie on one side and all the vertices for the other triangle on the other side.

Once you have written this function you can test if by executing the first three cells in the supplied script file `TwoLinkCSpace`. This code samples a grid of evenly spaced points in the 2 dimensional configuration space and produces a logical array `cspace` where the true entries correspond to points in configuration space obstacles and the false entries correspond to points in freespace.

Part 2 : [10 points] Dijkstra on a Torus

In the first homework assignment you were asked to complete a function called `DijkstraGrid` which we used to plan paths through simple grid based environments which were modeled using 2D logical arrays. In this homework you are going to extend that idea a bit by completing a new function called `DijkstraTorus`.

We have provided a skeleton of the code which looks and acts much like `DijkstaGrid`. You should be able to adapt the code that you wrote for homework 1 to implement this new function. The only real difference is that you need to keep in mind the fact that in this configuration space the two parameters, θ_1 and θ_2, correspond to angles so when you consider the neighbors of each cell you need to keep in mind the fact that the angles wraparound in both dimensions giving the configuration space the topology of a torus.

Once you have written this function you can test it by running the last two cells in the supplied script file `TwoLinkCSpace` which plots a path through the configuration space using `DijkstraTorus` and then animates this path. You should experiment with choosing different start and end coordinates. Just be sure to select coordinates that correspond to grid cells that are in freespace. That is if you choose coordinates `[i, j]` the corresponding entry in the cspace array `cspace(i,j)` should be false.

2.3 Submission and Grading

To submit your result to our server, you need to run the command `submit` in your MATLAB command window. A script will then evaluate your `triangle_intersection` and `DijkstraTorus` and generate two output files (`triangle.mat`) and (`DijkstraTorus.mat`) to be uploaded to the Coursera web UI. If you pass our test cases, you will get the full score in this assignment. You may submit your result multiple times, and we will count only the highest score towards your grade.

Part	Submitted File	Points
Triangle Intersection	`Triangle.mat`	15
Dijkstra on a Torus	`DijkstraTorus.mat`	15

3 Stretch Goal

For those of you that are looking for a bit more of a challenge you will notice that when you run this code that the Dijkstra's procedure can take a fairly long time to find a solution. See if you can modify the AStar code that you wrote for Homework 1 to handle this case. Remember that you need to account for the new toroidal topology when you construct your heuristic function. Experiment to see which planner reaches the destination in the smallest number of iterations.