# CS303 Project 3: Trajectory Planning

December 2022

## 1 Introduction

In this project, you are tasked with planning the trajectory of an agent to navigate on a 2-D plane while trying to collect valuable targets.

The agent's movement follows a B-Spline trajectory. A B-Spline is a piece-wise defined curve that is constructed by specifying a set of control points $C$, knots (where the pieces meet), and the spline degree $P$. It is commonly used as a trajectory parameterization in robotics, where the control points can be thought of as waypoints that the robot should follow. Specifically, the position of the agent at time $t$ is $B(t), 0 \leq t \leq |C| - P$.

There are $N$ targets from $K$ classes scattered on the x-y plane, each described by a feature vector of dimension $D$ and offers $r_k$ points when collected. Note that some target classes may have $r_k < 0$ and you should try to avoid them.

As shown in 1, the agent goes from $B(t_0) = (0, 0)$ to $(|C|, 0)$. Since a clamped B-Spline is constructed such that it always passes through the first and the last control points (but not necessarily others), you'll only have to determine the remaining $|C| - 2$ control points. A target is considered to be collected if the trajectory ever goes near it within a radius of $R$, i.e., $||B(t) - x_i||_2 < R$ for some $t$, where $x_i$ is the position of the target.

This process repeats for $E$ times and the score you get each time is calculated as the sum of points you get from collecting the targets along the way and the goal of this project is to design an agent that maximizes this score.

We will refer each trial as a "game" in the following discussion.

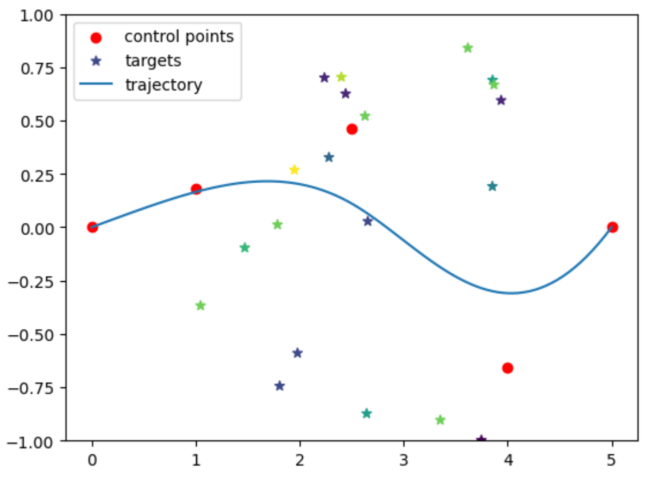

Figure 1: Illustration of the agent's movement in one game.

## 2 Instruction

### 2.1 Parameters and Assumptions

In this project, we let

- spline degree $P = 3$ (which controls the smoothness of the spline),
- number of control points $|C| = 5$ (which means you are responsible for determining the remaining 3),
- feature dimension $D = 256$,

- collecting radius $R = 0.3$.

In each game, $N$ targets are sampled from a dataset $\{(\texttt{feature}, \texttt{class})_i\}$ and then uniformly distributed in the square $0 < x < |C|, -1 < y < 1$. $r_k$ is sampled as a random integer between $-K$ and $K$ for each $k$ independently. See `generate_game` in `src.py` for details.

For all $E > 100$ games, $10 \leq N \leq 100$. A fraction of the dataset (from which the targets are sampled) will be provided while a separate held-out dataset will be used in the final evaluation.

## 2.2 Implementing the Agent

Essentially you are require to implement the `Agent` class. In each game, `get_action` is called to compute the required control points given the configuration of a game:

1. `target_pos`: a `Tensor` of shape $(N, 2)$ indicating the positions of the targets.
2. `target_features`: a `Tensor` of shape $(N, D)$ describing the targets.
3. `class_scores`: a `Tensor` of shape $(K,)$ specifying $r_k$.

and the return value `ctps_inter` is a `Tensor` of shape $(3, 2)$ specifying the 3 control points.

## 2.3 Evaluation and Grading

Your agent will be graded according to the average score it gets out of $E$ games generated as described above. The time limit for each game is $0.3s$. You get a score of 0 for the games where `get_action` fails to return a valid answer within the time limit.

As the evaluation involves randomness, a set of fixed random seeds will be used for generating the test cases with a large enough $E$.

## 3 Files & Usage

Here we introduce the files provided.

1. `demo.ipynb` visually demonstrates what a game looks like, particularly how a B-Spline trajectory is generated and evaluated.
2. `src.py` defines the constants and provides some basic utilities, e.g., computing a discretized B-Spline trajectory given its parameters. It also provides the example code to generate game configurations.
3. `agent.py` gives the starter code for an agent which returns random control points. Implement `get_action` with your own algorithm.
4. `eval.py` is the example script for evaluating your agent.
5. `data.pth` is the dataset from which the targets' features and labels are sampled. Note that in the final evaluation, a different dataset (but still following the same distribution) will be used. It can be loaded by `torch.load("data.pth")`.

## 4 Requirement & Submission

In this project you are required to use *Python* with *PyTorch*. However, installing a CUDA-enabled version is optional since not all students have a capable GPU.

Your submission should be a **Python Package** named `project3` as a single `.zip` file, including a module defining the `Agent` class implementing `get_action(target_pos, target_features, cls_scores)`. Being a **Python Package** means the files should be structured (when unzipped) like

```
project3/ # the whole package should be less than 1MB in size
    __init__.py
    agent.py # which implements the Agent class
    maybe_some_helpers.py
    maybe_some_parameters.pth
    ... # additional files you would like to use
```

such that it could be imported as a package from outside:

```
# in the evaluation script
from project3 import Agent

agent = Agent()
...
ctps_inter = agent.get_action(target_pos, target_features, class_scores)
```

The final evaluation environment will be on *Ubuntu* with *Python* 3.8 and *PyTorch* 1.13.0. Although you are free to include any additional files you would like to use, it is your responsibility to ensure they work properly outside your local environment.

## 5 Hints

- How to identify a target (i.e., determine the class it belongs) by its feature?
- As the games involve randomness, how to reliably evaluate the performance of your algorithm?
- Automatic differentiation with PyTorch.
- Automatic batching operations with *functorch*.