E.P.S.M. Project Write-up

Details:

- Paper title: 'Lazy Evaluation of Goal Specifications Guided by Motion Planning'
- Paper Authors: Juan David Hern´andez, Mark Moll, and Lydia E. Kavraki
- Students: Petre Ovidiu Adrian (343C3) & Stanciu Stefan Lucian (343C5)

I. Paper description

The scope of this paper is to highlight a motion planning algorithm that aims to enable cheap computational path planning in complex and abstract scenarios where robots and humans are expected to have frequent interactions based on ambiguous contexts.

The proposed algorithm is based on a lazy exploitation of the environment in which it takes place as to align with its specified constraints driven by a reward-penalty strategy. One iteration of such requirements can be something along the lines of '... ask a robot to "put a cup on the table," when there are multiple cups available'.

II. Algorithm description

The approach taken in the paper is that of a tree-based motion planner that aims to solve a start-to-goal motion planning problem and as such some common grounds must be defined:

- The robotic agent's motion capabilities are described through the set of configurations $\boldsymbol{q}$. The C-SPACE, C, is divided into free space (C_{free}) and the obstacle region(C_{obs}), i.e., C = $C_{free} \cup C_{obs}$. The dimensionality of the C-SPACE is given by the robot's $\boldsymbol{n}$ degrees of freedom (DOF).
- A goal region is a subset of the C-SPACE, $GR_j \subset$C. One or multiple goal regions GR_j can correspond to one specific semantic interpretation I_i. Furthermore, since the goal regions are contained in the C-SPACE, a configuration $G_{q_j} \in GR_j$ can be located either in the *free space* or the *obstacle region*.
- A separated semantic entity is used to facilitate human-robot interactions by translating a human ambiguous request into a set of valid interpretations $I = \{I_1, I_2, \dots, I_k\}$, and also provisioning a set of m goal regions $GR = \{GR_1, \dots, GR_m\}$.

Input & output data:

- The problem requires a start configuration q_s and aims to connect it to a goal configuration q_g.
- The solution to this problem is a continuous path $p : [0, 1] \rightarrow C_{free}$ such that $p(0) = q_s$ and $p(1) = q_g$, where q_g is contained in any of the provided goal regions.

General approach:

- We initially approximate the k goal regions by generating a set G of n valid goal samples, so that $G = \{g_{q_i} \in GR \wedge g_{q_i} \in C_{free}\}$, with $i = \{1,2,...n\}$.
- This reward-penalty strategy consists of identifying whether the attempts to expand towards a goal sample have been successful or not. In this paper the authors decided to validate the reward-penalty strategy by using a tree-based planner, which is based on the rapidly exploring random tree (RTT) algorithm.
- G_{heap} is a max heap, where all the goal samples g_{q_i} from GR are stored with an initial maximum weight of 1.0.
- In every tree expansion, i.e., towards a random configuration or a goal sample, the planner expands from the nearest configuration q_{near} in the generated tree (line 11), for a maximum distance ε, thus generating a new configuration q_{new} (line 12).
- In this approach, the authors also keep track of whether an expansion towards a goal sample succeeds of fails and then updates its weight according to the following formulas:

- Reward: $w(q_{g_i}) = w(q_{g_i})/(1.0 - w(q_{g_i}))$, if $w(q_{g_i}) < 1.0$
- Penalty: $w(q_{g_i}) = w(q_{g_i})/(w(q_{g_i}) + 1.0)$

Pseudo-code:

```
begin
    V = {q_start}, E = { }
    G_heap = goalRegionsSampler(GR)
    while not stopCondition() do
        if biasToGoal() then
            q_towards = G_heap.top()
            goal_biased = True
        else
            q_towards = C.genRandomConf()
            goal_biased = False
        q_near ← T.findNearNeighbor(q_towards)
        q_new, success ← calcNewState(q_near, ε)
        if success then
            V.addNewNode(q_new)
            E.addNewEdge(q_near, q_new)
            if goal_biased then
                G_heap.rewardGoalSample(q_towards)
        else if not success & goal_biased then
            G_heap.penalizeGoalSample(q_towards)
```

III. Implementation, testing, and demonstration plan

- Implement the algorithm described above in python.
- Create test data sets for a scenario such as an automated valet parking (AVP) disregarding 3D data and only focusing on the path planning. (This should be treated as a placeholder for the semantic entity)
- Have animated graphs as a demo for the next project stage.
- Try to benchmark the implementation by introducing fictional robot's data with higher DOFs.