Applied Reinforcement Learning
SS 2019

Project Proposal — Obstacle Avoidance

Akbar, Uzair	Gündogan, Alperen	Ellouze, Rachid
3697290	3694565	3671114

May 16, 2019

Abstract

This report presents our project proposal for the TUM Applied Reinforcement Learning class (summer semester 2019). Our proposed application is that of obstacle avoidance, which is a fundamental requirement for autonomous robots which operate in, and interact with, the real world. Conventional path planners for obstacle avoidance require tuning a number of parameters and do not have the ability to scale well with large datasets and continuous use. Therefore, we propose the use of Q-learning and/or SARSA on ϵ-greedy policy with Linear Value Function Approximation (LVFA) for obstacle avoidance, where LIDAR sensor information is provided. We will first train the agent in a simulation environment, and to bridge the reality-gap we shall consider using simulation $\rightarrow$ reality 'curriculum learning' and additive LIDAR simulation noise if necessary.

Contents

1 Introduction & Problem Definition

In order for mobile robots to navigate the real world they need to be able avoid obstacles along their trajectories. Obstacle avoidance is therefore a very established problem in robotics. Standard approaches involve a two step procedure — first use sensor data with methods like Simultaneous Localization and Mapping (SLAM) to infer traversable spaces, and secondly follow it up with path-planning algorithms, such as RRT*, to traverse those spaces while avoiding obstacles.

However, with this 2-step, computation heavy approach, it can be difficult to do path-planning on-the-fly with SLAM. Additionally, it is challenging for these approaches to automatically adapt with non-stationary environments. Additionally, we would like to explore an approach where the solution scales well with continuous use.

Because of these reasons, we propose the use of Reinforcement Learning (RL) based approach to tackle the obstacle avoidance problem. In the following report we formalize the problem in the context of turtlebot2 with a LIDAR sensor. Although the proposed approach is specific to the settings described in this report, it is only a proof-of-concept can be extended to any other setting, including dynamic environments, by e.g., incorporating optical-flow information in addition to LIDAR.

1.1 Goals

Our goals can be explicitly defined as follows:

- The turtlebot covers maximum ground in an episode.
- The turtlebot avoids collisions.

1.2 Problem Definition

The problem of obstacle avoidance can be formulated as a *Markov Decision Process* (MDP) where the turtlebot is interacting with environment with a LIDAR sensor and its actuators. At each time step $t \in [0, T]$, the turtlebot chooses an action $a_t \in \mathcal{A}$ based on the current state s_t, receives a feedback from the environment as a reward signal r_t and observe the next state s_t. The goal is to maximize the cumulative reward $R_t = \sum_{t=0}^{T} \gamma^t r_t$, where γ is the *discount factor*. In the following section we shall define each one of these quantities in the context of our goals in section 1.1.

2 Solution Approach

2.1 Environment Setup

2.1.1 Sensors

We will use LIDAR which is divided into N beams over the coverage of $[\theta_{min}, \theta_{max}]$. The appropriate parameters will be decided through the experiments. Each laser returns x_n if it detects an obstacle which denotes the (discretized) distance to that obstacle, otherwise d_{max}. These measurements are stored into a 'depth' vector as $d^T = (x_1, ..., x_N)$ and is exploited in our state representation.

2.1.2 State Space Definition

We define the state-space as $\mathcal{S} := \left\{ s \,\middle|\, s = [\, d^T \,,\, v \,,\, w \,]^T \right\}$, where d is the depth vector observed from the on-board LIDAR sensor. To leverage temporal information, v and w are linear and angular velocities of the robot respectively and shall be discretized as

$$v \in V := \left\{ 0.4, 0.2\,\mathrm{m\,s^{-1}} \right\}, \quad w \in W := \left\{ \frac{\pi}{6}, \frac{\pi}{12}, 0, -\frac{\pi}{12}, -\frac{\pi}{6}\,\mathrm{rad\,s^{-1}} \right\}.$$

2.1.3 Action Space Definition

The control space $\mathcal{A}$ shall determine the permissible linear and angular velocities for the turtlebot. We aim to discretize the action space in one of two ways.

1. $\mathcal{A}_1 := \Big\{ a \;\Big|\; a = \big(v', w'\big),\ v' \in V,\ w' \in W \Big\}$, where v' and w' are next state linear and angular velocities of the robot respectively. Hence, $|\mathcal{A}_1| = 10$.
2. A much simpler/smaller action space where v' and w' are "decoupled," e.g. $\mathcal{A}_2 := \big\{ a \;\big|\; a \in \mathcal{A}_1\,,\ v' = v \ \vee\ w' = w \big\}$, with $|\mathcal{A}_2| = 7$.

Due to a smaller action space, a good policy for $\mathcal{A}_2$ should be easier to learn than $\mathcal{A}_1$, however it might be biased given the additional constraints.

2.1.4 The Reward Function

Our task for the turtlebot is for it to go as fast as possible without any collisions. For this, we can employ a simple, action agnostic stage-wise reward function

$$r(s) = \begin{cases} v \cdot \cos(w) & \text{if episode running,} \\ c & \text{if collision.} \end{cases}$$

The cosine term penalizes meaningless rotation and c is a large negative reward.

2.2 The Reinforcement Learning Agent

We aim to explore a number of RL approaches for the problem that is formalized in the preceding sections. The approaches that we shall begin with are

1. **Q-Learning.** We want to use a model-free approach which could be implemented quickly. Therefore, the obvious candidate is simple *Q-learning*, which shall be our first approach and a baseline.
2. **SARSA with LVFA.** We also wanted to choose a method which is provably convergent (within some bound of the optimal) with LVFA. Hence, Q-learning will not suffice. Therefore we also intend to explore *SARSA* for an *ϵ-greedy policy.*
3. **TD with LVFA.** For completeness purposes and for a good comparative study of RL algorithms, we shall also consider the implementation of *Temporal Difference* (TD) based *Policy Iteration* (PI) with LVFA should the time allow.

Note that both approaches shall incorporate *eligibility traces*, since it would be difficult to train otherwise.

Although non-linear, *neural-network* based value function approximations have been popular recently in RL literature, we shall be using LVFA not only because it is a requirement of this class, but has theoretical lower bounds on the performance (with SARSA) and hence the results are reproducible. This is generally not true for non-LVFA approaches.

2.3 Feature Extraction & State Representations

Given the definition of our state-space in sections 2.1 and 2.2.1, our initial approach shall not include any further state-space dimensionality reduction, rather we shall begin with a sufficiently small 'depth' vector d so that our state space is small enough by design. Should we later need to increase the size of the state-space for some reason, adequate state representation techniques shall be explored accordingly.

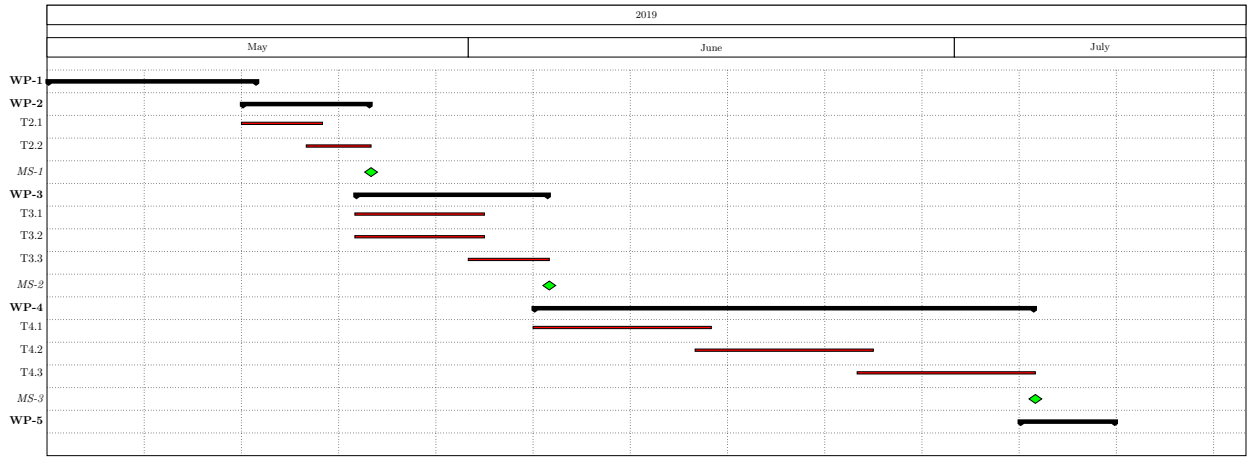

Figure 1: Tentative project time-line.

2.4 Training Environment & Reality-gap

We shall be training the RL agent in a simulation environment using either Gazebo, Stage or Box2D. An episode shall end on either a collision, or after N number of steps. For real-world training purposes, collisions can be detected using the turtlebot bumper values. It is expected that due to the nature of depth-vectors d obtained from LIDAR, their distribution should be similar enough in both a simulation and the real world that direct translation of the learned LVFA weights should still work. However, if that is not the case, we shall explore the following two solutions:

1. **Additive Simulation Noise.** We might consider adding noise to the sensor values (specifically depth vector d) in order to generalize better to the real work. However, we might need to explore a variety of different noise distributions should a simple Gaussian case fail to solve the problem.
2. **Curriculum Learning.** If the first approach fail to generalize well, we might consider re-training the turtlebot in the real world environment with the simulation weights as a good initialization for LVFA.

3 Project Organization

Figure 1 represents the time-line for the project. A list of tentative task distribution is also given below.

- Work Package 1: Required tutorials.(Uzair, Alperen, Rachid)
- Work Package 2:
  1. Setup environment and simulation framework. (Uzair, Alperen, Rachid)
  2. Implement basic functionalities.(*Milestone 1*).
- Work Package 3: Apply a draft algorithm in the simulation environment.
  1. Feature extraction design i.e. N, $[\theta_{min}, \theta_{max}]$. (Uzair)
  2. Implement RL algorithm. (Alperen)
  3. Tests with the simulation, tune the parameters to obtain a working agent.(Rachid) (*Milestone 2*)
- Work Package 4: Test with the real agent and improve the results.
  1. Apply different approaches on simulation. (Uzair)
  2. Transfer the simulated policy to the real agent and test. (Alperen)
  3. Decide and validate the final approach. (Rachid) *Milestone 3*)
- Work Package 5: Presentation and final report. (Uzair, Alperen, Rachid)