**Environment :** Open AI Gym environment customized by F1 Tenth

**Observations :** 1D LiDAR with 2700 field of view. Each 1D vector has 1080 beams and each beam is a real value indicating the distance to wall or obstacle.

**Actions :** Speed and Steering angle.

**State Space Representation :**

To understand where we are in a racetrack we represent our current observations obtained from the environment as states. Observations (LiDAR readings) are continuous values and cannot be directly used in constrained environments as we ideally have infinite possibilities of representing different states. Thus one option to deal with continuous state space is to discretize it. Multiple ways of discretization is possible and here we intend to represent the LiDAR information in a binary format. Particularly 11-bit binary representation is observed to work best which provides a decent trade-off by having finite 211 states as well as having enough states to represent the entire range of LiDAR readings, however this is arbitrary and can be tweaked as needed.

The process of discretizing the observations is as follows

1. The observations (vector of 1080 real values) is first stripped off the first and last 100 values leaving 880 real values because we are interested in having 2200 field of view instead of 2700 .
2. Then the observations are grouped into p bins (a bin is a set of points). No. of bins (p) is user defined and determines the no. of steering angles that we are going to use in the action space.

It must be noted that we have a trade-off in the choice of no. of bins. If the no. of bins is large we have more control over the steering angle but at the same time we increase the action space linearly. Having less no. of bins leads to manageable action space but we loose the granularity in steering angles.

1. Each bin is processed to extract the median of the bin which indicate the median distance to the obstacle/wall in the bin.

It is observed that median preserves the distribution of the LiDAR data over mean and is a better choice for discretization.

1. At this step we end up with a p-dimensional vector which is projected using random binary projection into an 11-dimensional space. The projection matrix is generated randomly with 4:1 ratio of 0 and 1.

There are many different projection schemes that perform the same and one might want to explore them that works the best.

1. This projection is then converted to binary values using a simple threshold.

It should be noted that the projection should preserve the similarity of the data points and similar LiDAR readings should have identical binary representations at the same time the projections should provide wider coverage among 211 states.

**Action Space** :

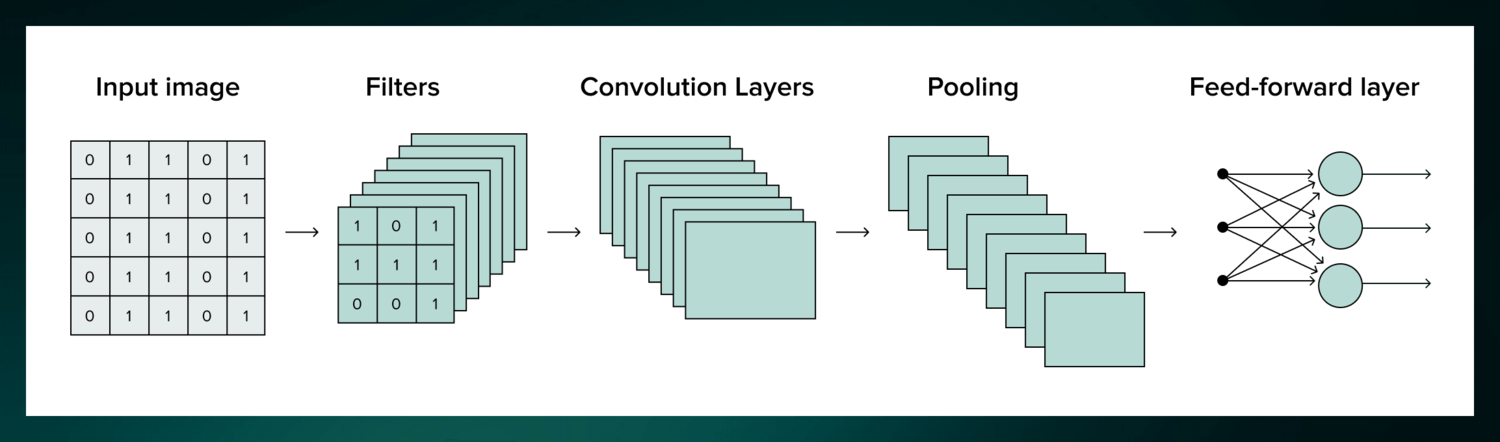
1. **Steering Angle** : As the LiDAR information is down sampled to p-beams where the angle between two beams is ~(220/p)0. Each of the p-beams corresponds to a specific angle between -1100 and 1100.

A vector of angle in radians is precomputed for entire field of view which represent the candidate pool for the selection of steering angle at any time instant.

1. **Speed :** To allow the agent to learn optimal speed at each time instant, speed is also introduced as real value uniformly sampled between 0.8 and 1.8. In this case ‘s’ different speed levels are considered.

The number of steering angles and speeds are arbitrary and can be changed as needed to have more granularity.

**Weight Representation :** As any algorithm works by adjusting its weights, we need to represent the weights in a specific format keeping in mind that every state has a set of actions (steering angle, speed) . Thus every state has synapses connected to each of p-angles and s-speeds. An easy way of visualizing this is a matrix of weights of shape (p,s) for each state. This matrix is needed for each of the 211 states.



State 1

State 2048

Speed

Steering Angle

Thus we have 3D weight representation.

**Reward:**

Choice of reward is crucial for the agent to learn. Reward needs to be positive or negative depending on the choice of action selected. The current reward consideration is

1. **Progress reward** : This reward is a quantitative measure reinforcing agent to indicate the progress made from the starting point. This value is normalized to prevent biasing the agent with huge positive rewards.

As the track properties like track length and track width are known. The progress can be easily calculated as

1. **Centerline Reward** : This reward penalizes the agent for deviating against the track centerline. Although it is a naïve approach, it is beneficial here as there are no obstacles and traversing along the centerline is the optimal trajectory. To calculate the centerline reward we make use of the track centers provided by the environment.
   1. The current position (x1,y1) of the agent is known, then we calculate the track center (c\_x1,c\_y1) which is closest to (x1,y1) by computing the distance between (x1,y1) and all track centers.
   2. As the agent progresses, we get the next position (x2,y2). However, the agent’s progress is very minimal compared to spacing between two track centers. Thus, we find the nearest center (c\_x2,c\_y2) which is the center either forward or backward from (c\_x1,c\_y1) i.e. we find the center from the subset of centers around the (c\_x1,c\_y1) that is close to the new co-ordinates (x2,y2). Because the agent can move in either direction we considers the centers around the (c\_x1,c\_y1). We need to ensure that both (c\_x1,c\_y1) and (c\_x2,c\_y2) are not same which leads to NaN values during the angle computation.
   3. Using the set of points we construct two-line equations
   4. The angle ϴ between the two lines is given by
   5. The following reward function determines the reward for every deviation. If the deviation is greater than 900 we give negative -1 as reward and if the deviation is between 0 and 900 we have exponential decay as shown below.

A diagram of a line with arrows and lines

AI-generated content may be incorrect.A graph of a line

Description automatically generated

1. **Milestone reward** : To reinforce the agent that its actions are correct we introduce a small positive reward of 5 for every new center it has reached. It is calculated as, the distance traversed from starting point (s\_x1,s\_y1) such that the distance is almost equal to the distance between two centers. Note that once the distance limit is reached the point (s\_x1,s\_y1) is updated to the new current location for further calculations.
2. **Collision reward** : A large negative reward of -50 is used to indicate collision.

The sum of all the above rewards is used as final reward for every iteration. The reward function can be engineered to include the speed reward as well.

**SARSA weight updates:**

Sarsa λ is a reinforcement learning algorithm used for making decisions in uncertain environments.

1. **Learning optimal policies:** Sarsa λ helps an agent learn the best way to act in an environment to maximize rewards.
2. **Temporal Difference (TD) learning:** It's based on TD learning, which updates value estimates based on the difference between predicted and actual rewards.
3. **Eligibility traces:** It uses "eligibility traces" to keep track of which state-action pairs were visited most recently, giving more weight to recent experiences.

A white background with black text

AI-generated content may be incorrect.The algorithm is given as

where,

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Q – weight matrix | | E–Eligibility Trace | | R – reward | | S – current state | A – current action | |
| S’ – Next state | A’ – Next action | | λ – Trace decay factor | | ϒ – Discount factor | | | α – Learning rate |

**BTSP weight updates:**

In BTSP synaptic changes are modulated by two signals namely Eligibility Trace (ET) and Instructive Signal (IS).

1. **Eligibility Trace :** ET is a decaying signal that is generated due to a visual/sensory cue. In our agent navigation problem this can be attributed to the LiDAR information which determines the current state of the agent.

Let us define ET as a 1D vector of same shape as no. of states (1,211) whose magnitude indicate the change in synaptic weight for each state. The decaying values over time indicate lower change in synaptic values for past states when compared with current states.

The state index deduced from the LiDAR is used to update the ET values. The value is set to 1 at the state index for current timestep and successively the values decay over time.

**Ex:** Consider a sample ET [0,0,0.81,0,0,0,0.9,0,0,……1] which is a vector observed at timestep 3. The values in the ET indicate that state 3 (observed at timestep 0) is eligible for synaptic modifications with a factor of 0.81, likewise state 7 (observed at timestep 1) is eligible for synaptic modifications with a factor of 0.9 and state 2048 is the current state at timestep 3 allowing the synaptic weights associated with this state to undergo maximum change.

1. **Instructive Signal :** IS, a supervisory signal arising from dendritic calcium spikes is responsible for synaptic modulation. Here, the agent is supervised using the LiDAR information particularly the distances to the obstacles.

We hypothesize that the down-sampled LiDAR information containing distances to the walls or obstacles can provide more information about the possible trajectory and an idea of the changes in the distances from the past. To match the dimensions, IS is considered a 2D vector whose shape is same as action space i.e.. (steering angles, speeds).

Steering Angle

Speed

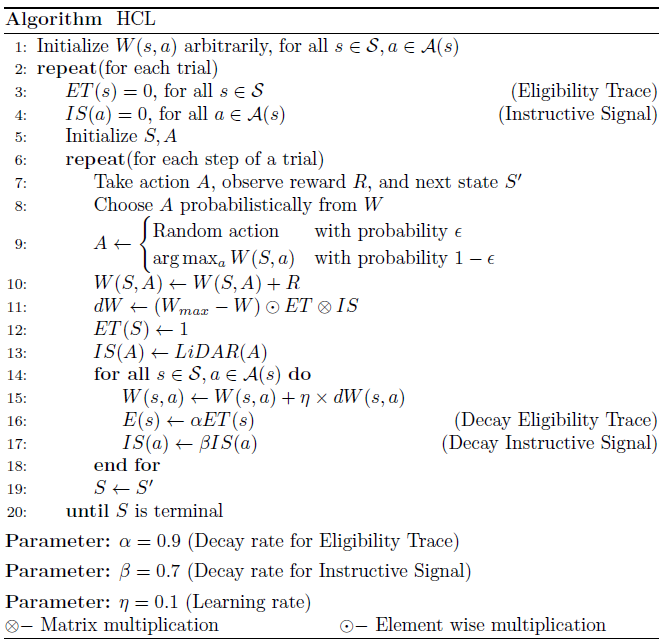
**IS**

As we have the down-sampled LiDAR information with median values. We use these values to represent the distance information. Instead of updating all the( state, action) pairs with the distances we update the distance information only at the specific (state, action) pair chosen by the agent. Additionally to ensure we are tracking the history of the agent; each time we update the current pair with the current distance and all the old pairs are decayed with a decay rate of 0.7.

1. **BTSP weight Update:**

Following the original BTSP synaptic update rule, we perform two simple operations to modify the weights.

1. Assign reward to the weight of the specific (state, action) pair.
2. Multiply the Eligibility Trace and Instructive Signal to determine the rate of change of the weights.



A diagram of a robot

AI-generated content may be incorrect.The following figure illustrates the flow of navigation within the environment.