Decision Making Under Uncertainty for Urban Driving

Philippe Weingertner, Arnaud Autef, Simon Le Cleac'h

***Abstract*—In this work we examine the problem of Motion Planning for Autonomous Driving - AD. We must plan in a stochastic environment with several sources of uncertainties: imperfect sensors, occlusions, unpredictable behaviour of external agents. We focus on sensor uncertainties and model the AD problem as a Partially Observable Markov Decision Process - POMDP - with a discrete action space and continuous state and observation spaces. We propose improvements to a traditional *online* algorithm used for such large POMDPs: the POMCP algorithm. The goal is to make our motion planner a *safer* autonomous driver. Our proposed solutions include: a "safe" discretization of the observation space, importance sampling techniques and an *offline* preprocessing of the state space to limit the planner's behaviour to "safe" actions. We implemented and tested our methods on two AD environments: a Custom Anti-Collision Test Environment - CACTE - and the Urban Driving Environment - UDE - from Stanford's Intelligent Systems Laboratory - SISL.**

I. Problem Statement

We consider the problem of determining a sequence of optimal actions for a Urban Driving Motion Planner. The autonomous driving pipeline consists of a two modules with distinct roles: a **sensors fusion and localization** to produce a probabilistic model of the environment's dynamics, **decision making** module in charge of defining a driving policy. The decision making module can be further subdivided into three modules:

1) **Route planner**: to define a long term driving goal.
2) **Behavioral planner**: to define a list of short term objectives, typically going from A to B with a mixture of *efficiency* (time taken) *comfort* (minimizing jerk) and *safety* (avoiding collisions and keeping safety distances) objectives.
3) **Motion planner**: to complete the motion tasks submitted by the behavioral planner.

Fig. 1: Autonomous Driving Pipeline

We focus on the **Motion planner**, which can be thought as a module taking sequential decisions, in the form of a discrete set of actions (acceleration or deceleration) that will then be converted to a sequence of continuous actions by a command and control module (usually a simple Model Progressive Controller - MPC).

The motion planner has to deal with several sources of uncertainties that are not directly observable: sensors uncertainties, occlusions and drivers intentions. Linking driving decisions to a proper handling of those sources of uncertainties is of paramount importance.

In this paper we study how POMDP models can be applied to an AD motion planner, and emphasize on the *safety* objective of the Motion Planner: our proposed algorithm should succeed in their objective of going from position A to B with the strict requirement of avoiding any collision with other agents

When dealing with huge states spaces, as it is the case in our urban driving setting, online methods are usually preferred over offline methods and our work starts from there. "A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles" [9], reveals that most motion planning techniques somehow boil down to an online graph search. In the paper, we study how uncertainty can be properly modeled and handled during this online graph search process to improve safety, and also consider offline methods to guide this graph search and improve safety.

II. Related Work

The *Partially Observable Monte Carlo Planning* - POMCP - algorithm [10] was proposed in 2010 to deal with large POMDPs, it is the starting point of our work. It was initially limited to discrete actions and observations space. In [1], an AD task is solved using a POMDP modeling of the environment along with an online POMCP solver. In order to deal with continuous state and continuous observation spaces a Progressive Widening technique was used. However, their work does not emphasize on safety in AD. A recent improvement to POMCP, POMCPOW [11] lets POMCP deal with continuous actions and observations. The proposed method is strongly tied to particle filters within the search tree. According to its author, the particle filters inside of the tree could be customized, but it's not immediately clear how to integrate a Kalman Filter. The Kalman Filter is the filter we decided to use because it is simple, fast and has no particle deprivation issue. In [5] the idea of a lossless partitioning, with respect to the policy, of the continuous observation space is introduced but the proposed approach is limited to discrete states and relies on point-based backups (PBVI, Perseus). The underlying idea is appealing to us and we decided to explore how to take advantage of the structure of our problem to come up with such a lossless partitioning of the continuous observation space. In addition to POMCP, DESPOT is another online POMDP solver that is providing

state of the art results: it has been used in conjunction with Importance Sampling in [7]. Dealing with huge states spaces, as it is the case in urban autonomous driving, online methods are usually preferred over offline methods. But using an online method based on sparse sampling may lead to safety issues. Rare events with critical consequences may not be sampled leading to sub-optimal and potentially dangerous decisions. That is why using Importance Sampling in conjunction with POMCP or DESPOT is important. In addition to the sampling of observations, we bring another contribution to POMCP. We modify the action selection phase to add a safety criterion so that the car can only take actions that will not result in a collision, with a chosen level of confidence. This approach has been developed and tested in the AD setting in [2]. We use the same formulation for the offline computation of the safe actions, and we extend this approach to the POMDP setting, while [2] applied it to a fully observable MDP. We build upon the work referred to previously and we will be using the Julia POMDP framework from [4] for experimentation.

III. Proposed Approach

First, we describe the modelling of our two simulation environments by a POMDP

A. POMDP Models

1) Custom Anti-Collision test environment: **State space** $\mathcal{S}$ and **observation** space $\mathcal{O}$ are continuous, they correspond to position and speed information for each car:

$$\{(x, y, v_x, v_y)_{ego}, (x, y, v_x, v_y)_{obj\,1..n}\}$$

The **action space** $\mathcal{A}$ is discrete with 4 actions corresponding to acceleration levels along a straight line:

$$a \in \{-4\ ms^{-2}, -2\ ms^{-2}, 0\ ms^{-2}, 2\ ms^{-2}\}$$

The **transition** and **observation** models are linear Gaussian with $Pr(P_i^{t+1} \mid P_i^t) = \mathcal{N}(TP_i^t, Q)$ and $Pr(O_i^t \mid P_i^t) = \mathcal{N}(HO_i^t, R)$.
The **belief updater** is a Kalman Filter.
The **reward model** accounts for efficiency, comfort and safety objectives. Total reward is the sum of the following terms:

- r_efficiency $= a$ as we target full speed
- r_comfort $= -1$ for hard breaking $a = -4\ ms^{-2}$ 0 otherwise
- r_safety_ttc $= (10 -$ smallest Time To Collision$) * 10$ if $ttc < 10$
- r_safety_collision$= -1000$ in case of a collision

2) Urban Driving Environment: The UDE is a *2D representation* of an AD task: the ego vehicle has to perform a left turn at a T-shaped intersection while avoiding two other cars driving on the main road. It is described in more details in part IV.
The **state space** $\mathcal{S}$ includes, for each car:

- The Cartesian coordinates of the car in the environment x, y, θ. θ being the orientation of the car around its center.
- The Frenet coordinates of the car in the environment s, t, ϕ, where s and t are the tangent and the normal coordinates of the car on its *current* lane.
- The norm of the car's speed vector v.
- The dimensions $(width, length)$ of the car.
- The car's *driver model*, determining the car's driving behaviour (for the ego it is instead the motion planning algorithm selected).

The **action space** $\mathcal{A}$ is discretized and limited to 4 actions. The ego vehicule is following a *predetermined* curve on the T-shaped intersection of the UDE, it is not allowed to drift from this curve and its actions are *acceleration* levels of the vehicle along the curve at every time-step. The 4 actions correspond to the following acceleration levels:

$$\{-4m/s^{-2},\ -2m/s^{-2},\ 0m/s^{-2},\ 2m/s^{-2}\}$$

The **observation space** $\mathcal{O}$ choice has been crucial, as we selected what our agent would be able to "see" during simulations. Observations are vectors of size $4 \times n_{cars}$. They are obtained by concatenating vectors (d_{ego}, l, s, v) for each car, where d_{ego} is the Euclidian distance (in meters) between the car and the ego vehicle, l is the lane of the car, s its tangent coordinate on the lane and v its speed (in meters / second).
The $state \rightarrow state'$ **transition** model is based on simple point mass dynamics. We added a *transition noise* to transition dynamics of (s, v) for each car, in the form of a gaussian distribution with mean 0 and variance $\sigma_s^2 = 0.25\ m^2, \sigma_v^2 = 0.25\ m^2.s^{-2}$.
The $state \rightarrow observation$ **observation** model correspond to picking up the (d_{ego}, l, s, v) values from the state and adding an *observation noise* to (s, v), in the form of a gaussian distribution with mean 0 and variance $\sigma_s^2 = 1.0\ m^2, \sigma_v^2 = 1.0\ m^2.s^{-2}$.
The **belief updater** is a Kalman Filter.
The **reward model** if simple: a reward of $+1.0$ is granted when the agent reached the end of the intersection, any collision ends the simulation and yields a reward of -1.0. The discount factor for rewards is set at $\gamma = 0.95$.

B. POMCP algorithm

The POMCP algorithm [10] is an extension of the traditional *Monte Carlo Tree Search* algorithm to *Partially Observable* Markov Decision Processes. In our attempt to solve the two POMDP models specified previously, we start from a variant of this algorithm, which maintains a *belief b* in observation nodes of the search tree. The core of the algorithm is presented in Algorithm 1 and the rollout routine in Algorithm 2.

C. Safe variants of POMCP

Then, we make some modifications to this basic POMCP algorithm to improve performances of our policies in terms of safety, where safety corresponds to the collision rate of our ego vehicle with other agents and should be as close as possible to 0.

1) Safe discretization of observations: In the CACTE and UDE, the observation space is *continuous*, and very hard to discretize. In CACTE for instance, when considering a scene with 10 objects, observations are vectors of 40 real coordinates. Therefore, With POMCP, for every tree query, a previously unseen observation will be sampled. If this

Algorithm 1 POMCP algorithm

function SELECTACTION(b, d)
  $h \leftarrow \emptyset$
  loop
    SIMULATE(b, h, d)
  return arg max$_a$ $Q(h, a)$
function SIMULATE(b, h, d)
  if $d = 0$ **then**
    return 0
  $a \leftarrow$ arg max$_a$ $Q(h, a) + c\sqrt{\frac{logN(h)}{N(h,a)}}$
  $s \sim b$
  $(s', o, r) \sim G(s, a)$
  if $hao \notin T$ **then**
    for $a \in A(s)$ **do**
      $(N(h, a), Q(h, a)) \leftarrow (N_0(h, a), Q_0(h, a))$
    $T = T \cup \{hao\}$
    return ROLLOUT(b, d, π_0)
  $b' \leftarrow$ UPDATEBELIEF(b,a,o)
  $q \leftarrow r + \lambda$ SIMULATE($b', hao, d - 1$)
  $N(h) \leftarrow N(h) + 1$
  $N(h, a) \leftarrow N(h, a) + 1$
  $Q(h, a) \leftarrow Q(h, a) + \frac{q - Q(h,a)}{N(h,a)}$
  return q

Algorithm 2 Rollout evaluation

function ROLLOUT(b, d, π_0)
  if $d = 0$ **then**
    return 0
  $a \sim \pi_0(b)$
  $s \sim b$
  $(s', o, r) \sim G(s, a)$
  $b' \leftarrow$ UPDATEBELIEF(b,a,o)
  return $r + \lambda$ ROLLOUT($b', d - 1, \pi_0$)

observation does not belong to the tree, a new node will added. By using POMCP with continuous observations, the probability of sampling identical observations is zero, and the end result is a shallow tree, see figure 2. Technically POMCP will execute without error but will return a significantly sub-optimal solution.

After considering potential use of POMCP with DPW or POMCPOW to deal with continuous observations, we developed our own POMCP variant by taking advantage of the structure of our AD problem. Ideally we would like to be able to limit the number of observation nodes in the tree so that we can explore the tree in depth and keep the capability to do belief updates during the rollout evaluation with continuous observations.

When aggregating observations, we would like grouped observations to correspond to similar utilities. The utility function is highly safety-dependent. In the CACTE, it also depends on efficiency and comfort, but both are action rather than observation dependent. Therefore, grouping together observations according a safety criteria looked reasonable, and a good way to improve safety, let us describe the procedure with both environments:

- In **CACTE**, for every single observation, we evaluate the smallest Time To Collision to the 10 objects observed and create 11 observations classes corresponding to a TTC ranging from less than 1, to less than 10, with any other smallest TTC value above 10 being aggregated into the lowest collision risk category. We create observations classes that should correspond to same safety utility or a somewhat more pessimistic safety value.
- In the **UDE**, we create classes of observations based on the distance between the ego car and the closest car in the scene. We define 20 observations classes corresponding to distances between 0 and 10 meters (the last class containing all distances above 10m).

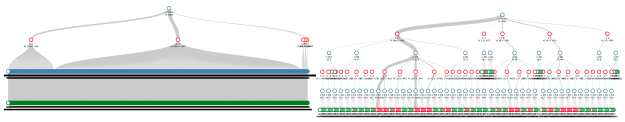

Fig. 2: With safe clusterization of observations (right)

In both settings, for every observation class node, we keep track of the most recent raw observation. This raw observation value is used to update the belief node with a Kalman Filter.

While this solution is taking advantage of the structure of our problem it highlights some more general ideas that could be re-used to apply POMCP algorithm to problem with continuous observations:

- Observations nodes are aggregated per Utility class
- We have a dual capability to use a discretized version of the observation when handling MCTS tree expansion while still using a continuous observation for BeliefUpdate in a way that would be adapted to elaborate Importance Sampling approach, where we want to prioritize the sampling of collision scenarios in the tree to efficiently avoid them.

The proposed modification of POMCP algorithm is presented in Algorithm 3 and Algorithm 4.

For rollout evaluation we keep on using continuous observations to update our belief with a Kalman Filter and we sample continuous states. The discretization of observations was used only during the tree expansion phase: to limit the tree width extension so that we can explore the tree in depth.

2) Reward shaping: The base reward used to guide our policies is a sparse reward where we get a large penalty in case of collision. To improve safety, we propose to shape this reward according to safety metrics. In the CACTE, reward shaping takes the form of reward that is proportional to the smallest computed Time To Collision w.r.t. any other cars detected. In the UDE, we explore increased reward penalties for collisions.

3) Safe exploration with constrained actions: The last technique we propose to enforce safety is to adapt the work of Bouton and al. in [2] to the partially observable setting. In their work, they propose an interesting offline method to

Algorithm 3 Modified POMCP Tree Search

function SELECTACTION(b, d)
  $h \leftarrow \emptyset$
  loop
    SIMULATE(b, h, d)
  return arg max$_a$ $Q(h, a)$
function SIMULATE(b, h, d)
  if $d = 0$ **then**
    return 0
  $a \leftarrow$ arg max$_a$ $Q(h, a) + c\sqrt{\frac{logN(h)}{N(h,a)}}$
  $s \sim b$
  $(s', o, r) \sim G(s, a)$
  $o_{class}, ttc \leftarrow$ CLUSTERIZEOBSERVATION(s',o)
  if $hao_{class} \notin T$ **then**
    for $a \in A(s)$ **do**
      $(N(h, a), Q(h, a)) \leftarrow (N_0(h, a), Q_0(h, a))$
    $o^{id}_{class} = o_{class}, o^{ttc}_{class} = ttc, o^{raw}_{class} = o$
    $T = T \cup \{hao_{class}\}$
    return ROLLOUT(b, d, π_0)
  else if ttc $< o^{ttc}_{class}$ **then**
    $o^{ttc}_{class} = ttc, o^{raw}_{class} = o$
  $b' \leftarrow$ KALMANUPDATER(b,a,o)
  $q \leftarrow r + \lambda$ SIMULATE($b', hao_{class}, d - 1$)
  $N(h) \leftarrow N(h) + 1$
  $N(h, a) \leftarrow N(h, a) + 1$
  $Q(h, a) \leftarrow Q(h, a) + \frac{q - Q(h,a)}{N(h,a)}$
  return q

Algorithm 4 Clusterize observation

function CLUSTERIZEOBSERVATION(s', o)
  $ttc \leftarrow$ SMALLESTTIMETOCOLLISION(s',o)
  $o_{class} = floor(min(ttc, 11))$
  return o_{class}, ttc

ensure safety in an AD setting, they use a fully observable model of the AD task, discretize the state space, and apply a *Value Iteration* algorithm to compute:

$$\mathbb{P}_{collision}(s, a),\ \forall s, a \in \mathcal{S} \times \mathcal{A}$$

Where those collision probabilities are obtained performing "safe Bellman updates" until convergence:

$$\mathbb{P}^{k+1}_{collision}(s, a) = \sum_{s'} T(s'|s, a) \min_{a'} \mathbb{P}^{k+1}_{collision}(s', a')$$

$\forall s, a \in \mathcal{S} \times \mathcal{A}$, with $\mathbb{P}_{collision}(s) = 1$ for collision states, $\mathbb{P}_{collision}(s) = 0$ for terminal and non-colliding states, $\mathbb{P}_{collision}(s)$ initialized to 0 for other states. Then, a threshold for safety $0 < t < 1$ is selected, and the AD car is restricted in the actions a it can take by the condition $\mathbb{P}_{collision}(s, a) < 1 - t$. For instance, with a safety level $t = 0.999$, $\mathbb{P}_{collision}$ must be below 0.1%. If $\min_a \mathbb{P}_{collision}(s, a) \geq 1 - t$, agent must select $a^*(s) = \arg\min_a \mathbb{P}_{collision}(s, a)$
In our work, we adapted this work to the *Partially Observable* setting, where we work with *beliefs b* rather than states. Our approach is the following:

1) Compute the same offline safety preprocessing on a discretized version of $\mathcal{S}$ and get $\mathbb{P}_{collision}(s, a),\ \forall s, a \in \mathcal{S} \times \mathcal{A}$
2) Set a safety threshold t
3) When agent is in belief b, make the approximation:

$$\mathbb{P}_{collision}(b, a) \approx \int_s b(s)\mathbb{P}_{collision}(s, a)ds$$

This approach allowed us to transfer this safety improvement technique to the partially observable setting, and interesting remarks can be made:

- The approximation above is similar to the one made with the **QMDP** approach to POMDPs: for the approximation to be valid, the environment should fully observable after an intial sampling from the belief b.
- We have two external cars in our environment, which would have made value iteration two expensive on a discretized version of the state space with two cars. We solve value iteration on a state space with a single external car, and take the following **utility decomposition** approach:

  $$Act(b) = Act(b)_{first\_car} \cap Act(b)_{second\_car}$$

  Where $Act(b)_{car}$ is the set of available actions using our above approximation when looking at potential collisions only with this car. If $Act(b) = \emptyset$, we compute $\min_a \mathbb{P}_{collision}(b_{car}, a)$ for the two cars, take the maximum (i.e look at the **most dangerous** car), and select the according **safest** action $\arg\min_a \mathbb{P}_{collision}(b_{car}, a)$.

D. Other possible approaches

a) Exhaustive online forward search: We also consider what it would take to perform an exhaustive online forward search. By using the observation discretization scheme proposed previously, the complexity of an exhaustive online Forward Search would be: $O(|A|^d|O|^d)$.With $|A| = 5$ and $|O|$ being reduced to 10, we would be able to consider a forward search up to a depth of 5. With a time step of 250 ms, this corresponds to a 1.250 seconds horizon.

b) Exact solution with Linear Gaussian dynamics and quadratic reward: One very interesting point to consider here, is that we have a linear-Gaussian dynamics model: $T(z \mid s, a) = \mathcal{N}(z \mid T_s s + T_a a, \Sigma)$. As long as we consider a state vector of the form $s = [x, y, v_x, v_y]$ and a discrete action that corresponds to an acceleration or deceleration command this condition is fulfilled. If we could express the reward function with a quadratic form, then it can be shown that the optimal policy can be computed exactly offline. We refer to the "Decision Making under Uncertainty" book in [6] sections 6.4.6 and 6.2.2 for more details. The reward should be expressed in a form $R(s, a) = s^T R_s s + a^T R_a a$ where $R_s = R_s^T \leq 0, R_a = R_a^T < 0$. This is a very interesting property that we should take advantage of. This would require further investigations.

IV. Experimental Setup

A. Custom Anti-Collision Test Environment

While the SISL Urban Driving environment is great, building the CACTE was a good way to learn and better understand some details in the process of using and interfacing the juliaPOMDP framework. This environment is useful for tests in unstructured environments without lanes and roads constraining the traffic of other cars, see figure 3.

This environment can be downloaded from: Anti-collision Tests Link. It depends on a very limited number of packages and allows to experiment with a complex high dimensional POMDP with 1 egovehicle and 10 moving objects in the scene (more can be easily added). It enables to benchmark performances both in terms of runtime and various performance metrics: efficiency, comfort and safety. It is lightweight tool to create challenging anti-collision tasks.

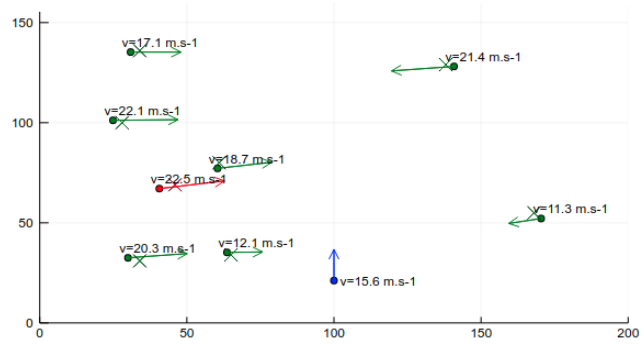

Fig. 3: The anti-collision test environment, the ego car (blue dot) has to avoid 10 other vehicles (red dots).

B. SISL Urban Driving environment

To evaluate our safe variants of POMCP, we rely on the SISL Urban Driving environment that is used in [2]. We focus on the intersection setting shown in figure 4. In this environment there are always two cars in the scene in addition to the ego car. The ego car has to make a left turn to reach the goal while avoiding any collision. All three cars are following a predetermined route and always stays in the middle of their lanes. For the ego car, we are controlling the acceleration a of the vehicle along the predefined trajectory. We restrict our action space $\mathcal{A}$ to $a \in [-4, -2, 0, 2]$. The two other cars follow a rule-based policy: they give the priority to other vehicles when turning left and use the time to collision to make a crossing decision. Otherwise they are following the Intelligent Driver Model [12]. In this environment, we test the POMCP algorithm with observation clustering and action selection safety criterion (Safe RL). For comparison we also test the version without the action selection safety criterion (Regular RL).

To apply Safe RL, we discretize the state space containing all the possible configurations of the ego car and another car in the scene. We discretize the longitudinal velocities of both cars and their positions along the lanes present in the scene. We obtain a discretized state space of 330.000 states and apply the value iteration algorithm described in III-C3 to get the safety criterion. We set the safety threshold to $t = 0.9999$.

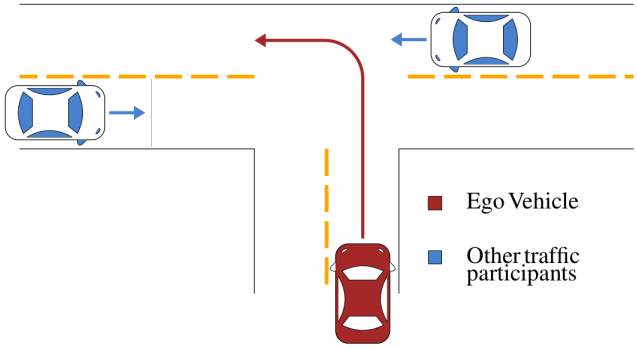

Fig. 4: The ego vehicle have to perform a left turn at an unsignalized intersection while avoiding the other vehicles. Figure taken from [2].

V. Results

A. Results on the Custom Anti-Collision Test Environment

1) Processing time and exploration depth results: In the below tests, the max_depth is set to 20. The tree_queries parameter is increased and we check the maximum exploration depth i.e. how deep we can explore and exploit the search tree. We use a time step of 200 ms in our Transition Model. So an exploration depth of 10 corresponds to an exploration up to a 2 seconds time horizon.

As a reminder without our POMCP modification, as observations are continuous, the exploration depth is 1.

Results have been obtained on a 7th generation intel core i7 processor, without any specific optimization or multithreading.

TABLE I: POMCP results

tree_queries	runtime	max exploration depth
100	35 ms	5
500	160 ms	10
1000	320 ms	12
2000	640 ms	14

2) Accuracy results: The scenario is quite challenging and default policies generally fail. The *Time To Goal* and *Number of hard breaking decisions* performance metrics are averaged over the scenarios that did not result in a collision.

We run exactly the same set of tests with the different policies.

- policy_v0 is the default one (pomcp legacy and sparse reward).
- policy_v1 is based on pure predicted time to collision. This is our real baseline. Not making use of the POMDP model.
- policy_v2 is based on reward reshaping based on time to collision with legacy POMCP (pomcp legacy + reshaped reward)

- policy_v3 is with the modified pomcp. It is the policy that is POMDP based (with exploration depth > 1) with a modified POMCP.

Policy_v1 and policy_v2 yield exactly the same results, which is expected as using POMCP with policy_v2 does not give improvements: observations are continuous and we do not explore the tree at all.

TABLE II: Benchmark results

	% Collisions	Time To Goal	Hard Breaking
policy_v0	80%	11 s	8
Baseline policy_v1	60%	11.8 s	10
policy_v2	60%	11.8 s	10
policy_v3	0%	12.8 s	10

As expected, we get clearly better safety results with policy_v3.

B. Results on SISL Urban Driving environment

1) Offline Computation of the safe actions: To determine the set of safe actions for a given a belief, we compute the value iteration described in III-C3 with a discretized state space of size 330.000. This value iteration takes 2 hours to run on a i-core7 platform without any specific optimization or multithreading. At the end of the optimization, we get a Bellman Residual of 10^{-5}. Which means that our computation of $\mathbb{P}_{collision}(s, a)$ through value iteration is precise up to the fourth digits. This is consistent with the threshold value that we selected to ensure safety: $t = 0.9999$. We visualize in figure 5 the repartition of the tuples of discretized states and actions (s, a) according to their safeties defined by $\mathbb{P}_{collision}(s, a)$.

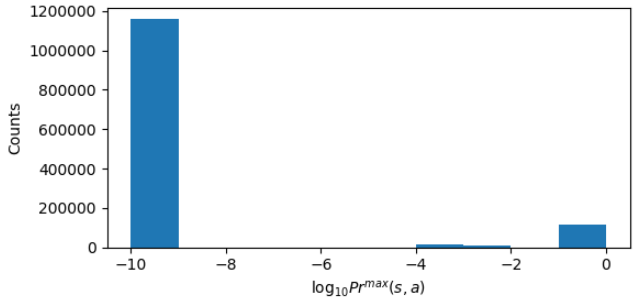

Fig. 5: Histogram presenting the repartition of the state-action pairs (s, a) according to their safety values. The safest states are on the left of the chart.

2) Exploration with constrained actions: This problem has two conflicting objectives: efficiency which we define as the time required to reach the objective, and safety. We compare the Regular RL and Safe RL algorithms for different reward settings. We vary the collision cost between 0.5 and 8 for both solution methods and estimate their performance. The performance is defined by computing the mean number of timesteps needed to reach the goal (efficiency criterion) and the collision rate (safety criterion). We perform 20 rollouts of each algorithm for each value of the collision cost. The tree exploration is parameterized by the number of tree queries of 100 at each time step, ad the exploration parameter $c = 0.1$. Based on these simulations, we draw the Pareto frontiers shown in figure 6.

On this chart the optimal region is in the bottom-left corner. We can see that Regular RL can reach points that are more efficient than those obtained with Safe RL. However, the chart shows that Safe RL can outperform Regular RL on the safety criteria while having an efficient policy. This indicates that the Safe RL algorithm can generate policies with performances that could not be reached by applying reward shaping to Regular RL.

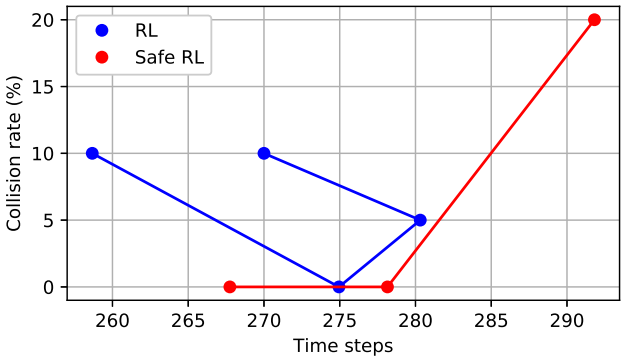

Fig. 6: Pareto frontiers obtained using the Safe RL and regular RL algorithms. These frontiers corresponds to different behaviors in terms of safety and efficiency induced by variations in the collision cost in the reward function.

VI. Conclusion

We came up with a detailed POMDP formulation of the problem of Motion Planning for an Autonomous Driving car dealing with sensors uncertainty and proposed several improvements to the legacy POMCP solver to enforce safety. We handled the issue of continuous observations in a high dimensional POMDP and proposed several safety improvement techniques. We provided reference implementations and benchmark results on a set of tests to validate the proposed ideas. Further work could involve:

- The ego vehicle is currently constrained to longitudinal accelarations on a curve, we could explore more complex 2d scenarios, and include pedestrians in the environment.
- Using value iteration on a discretized *belief* rather than *state* space to compute the safe actions. It would improve safety and let us avoid a "QMDP-like" assumption. Belief space discretization could be driven by safety criteria, in the same line than our modification of POMCP to handle continuous observations.
- Pure offline methods similar to the ones used for aircraft collision avoidance [3] could be considered as well.
- Finally, it would be interesting to explore *online* safety methods to constrain actions compared to an *offline* preprocessing of the discretized state space.

REFERENCES

[1] Maxime Bouton, Akansel Cosgun, and Mykel J. Kochenderfer. Belief state planning for autonomously navigating urban intersections. 2017.

[2] Maxime Bouton, Jesper Karlsson, Alireza Nakhaei, Kikuo Fujimura, Mykel J. Kochenderfer, and Jana Tumova. Reinforcement learning with probabilistic guarantees for autonomous driving. In *Workshop on Safety Risk and Uncertainty in Reinforcement Learning,*, 2018.

[3] Hugh Durrant-Whyte, Nicholas Roy, and Pieter Abbeel. *Unmanned Aircraft Collision Avoidance Using Continuous-State POMDPs*. 2012.

[4] Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer. POMDPs.jl: A framework for sequential decision making under uncertainty. 18(26):1–5, 2017.

[5] Jesse Hoey and Pascal Poupart. Solving pomdps with continuous or large discrete observation spaces. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 1332–1338, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[6] Mykel J. Kochenderfer. *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[7] Yuanfu Luo, Haoyu Bai, David Hsu, and Wee Sun Lee. Importance sampling for online planning under uncertainty. 2017.

[8] Yuanfu Luo, Panpan Cai, Aniket Bera, David Hsu, Wee Sun Lee, and Dinesh Manocha. Autonomous driving among many pedestrians: Models and algorithms. *CoRR*, abs/1805.11833, 2018.

[9] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, pages 33–55, March 2016.

[10] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2164–2172. Curran Associates, Inc., 2010.

[11] Zachary N. Sunberg and Mykel J. Kochenderfer. Online algorithms for POMDPs with continuous state, action, and observation spaces. 2018.

[12] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62:1805–1824, 02 2000.