Assignment III: Probabilistic Reasoning

Introduction

This assignment concerns probabilistic reasoning to estimate a quantity of interest using noisy observations acquired over time. We consider the example of an intelligent road vehicle equipped with a (noisy) sensor that needs to locate other vehicles in order to plan a safe path to its goal.

Domain Description

Consider a grid world with an autonomous car (with your software in its computer) moving in the presence of other cars on the road. The autonomous car (now referred to as AutoCar) needs to know the location of other cars (referred to as StdCar) in order to plan its path safely without collisions with other cars. The domain is described as follows:

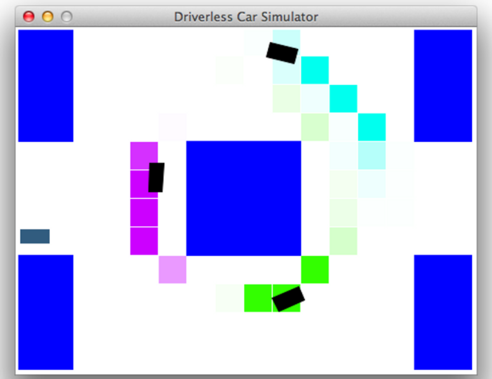

Figure: The simulated 2D environment used in this assignment. The AutoCar (leftmost car) estimates a belief (multiple colours) over the positions of other 3 StdCars.

Environment

The environment is a 2D grid world with discrete grid cells. The grid can have cells which are occupied with obstacles. Along with the AutoCar, there can be K StdCars present in the environment. The cars can move from one grid cell to another grid cell that is unoccupied. Some cars may simply be static the whole time.

Noisy Sensor

The AutoCar is equipped with a microphone-like sensor that measures the *loudness* of another car in the vicinity. The loudness recorded in the microphone can be interpreted as a measurement of the relative distance between the AutoCar and a StdCar.

The sensor measurements are noisy. Let z_t denote the measured distance. The distance is normally distributed as $z_t = \mathcal{N}(||y_t - x_t||_2, \sigma^2)$, where y_t and x_t denote the true positions of the AutoCar and the StdCar in the grid respectively and σ denotes the standard deviation. For example, if the relative distance is 5 grid cells then the sensor may measure the distance as 5.1, 4.0, 8.0 with decreasing probabilities.

Transitions

The motion of a StdCar(s) is also uncertain. When the car takes an action (i.e., movement from one grid cell to another cell) the car may not arrive at the intended grid cell. The transition probabilities for car movement are assumed to be known.

Assume that the movement of the StdCar(s) is *independent* of each other. Note that in real road driving this assumption may *not* be true, two cars may be moving while taking the motion of each other into consideration, but such correlations can be ignored for now. The noise in the sensor measurement for each car is also assumed independent.

Your Implementation

Part A: Estimation

Our goal is to implement a probabilistic tracker (running on the AutoCar) that estimates a probability distribution (a *belief*) over the possible location of StdCar(s) from sensor data collected over time. Formally, we want to estimate the belief as $p(x_t^k|z_0^k, z_1^k, \dots z_t^k)$, where x_t^k denotes the 2D location for the k^{th} StdCar from the sequence of noisy observations $\{z_0^k, z_1^k, \dots z_t^k\}$ of the k^{th} StdCar as received by the AutoCar's sensor up to time t. Since, the estimated beliefs are probability distributions, they must be *normalised*.

Please implement an algorithm (called *Estimator*) to estimate the belief over the position of each StdCar(s) from the collected measurement corresponding to that car. Analyse the state space, the transition and the observation models.

- One approach is to use Exact Inference (standard Filtering in a time series model) to estimate the current position of the car. However, this approach may be too slow for a larger grids to be computed in a reasonable time. An alternate approach is Particle Filtering. This approach will now only approximately represent the belief but has the advantage of better scalability.
- Please base your implementation on a Particle filter. Experiment with this approach on a few settings. You may notice that the particle filter may not perform well in some cases (e.g., when the StdCar is not moving). Based on your insights, improve your particle filter implementation for the given problem to arrive at your best solution.

Since the focus of this part is on tracking, assume that the code for driving the AutoCar is given. The code provided will drive the car collecting measurements of other cars. Your task will be to estimate the belief given the observations collected. Note that the driving code is not sophisticated and crashes (with StdCars or obstacles) may occur.

Part B: Planning

We now turn our attention to driving the AutoCar making use of the tracking ability you developed in Part A. Assume that the AutoCar is provided a sequence of goal locations (grid cells) that must be visited (in order). Given the tracker developed in Part A, the AutoCar should have a good estimate of the positions of other cars. Our goal now is to use this information to plan a safe path without colliding with any of the other cars in the environment or the walls.

- Please implement a planner (called *Intelligent Driver*) for safely driving the AutoCar. The planner will determine which grid cell the AutoCar should move into next. Please think about how the car can be kept safe while determining paths to the prescribed goal locations in the presence of uncertainty over the other cars. You may need to forecast the positions of other cars into the future as you decide the next action for the AutoCar. The AutoCar simply needs to decide the next grid cell it will move into. Once the car decides the grid cell it is aiming for, the car will turn and start moving towards the grid cell (motion will be simulated).

- A simple planner for driving the car is already provided as reference (See AutoDriver). The approach extracts a graph from the simulator and then determines a path that avoids collision by using the beliefs to detect the presence of other cars on its path. This driver is rather naive and does not necessarily attempt to visit all the goal locations. As you implement your Intelligent Driver you may refer to the provided code as a reference.

Implementation Guidelines

Starter Code

Conda Environment. Please configure a conda environment with the permitted assignment dependencies using the commands provided below.

```
# Create the conda environment conda env create -n assign3 --file environment.yml # For Linux conda env create -n assign3 --file environment_win.yml # For Windows conda env create -n assign3 --file environment_mac.yml # For Mac conda activate assign3 # Run the environment
```

Code Package. Please download the starter code from Moodle. The downloaded zip has the following structure. Note that your implementation will only be modifying the files listed in red below. Do not modify any other file(s).

```
A3 ├── layouts # The layouts of different environments. Prefix 'm_' implies a 'multiple goals' version of the corresponding layout. │   ├── small.json │   ├── lombard.json │   ├── m_small.json │   └── m_lombard.json ├── learned # The transition probabilities for each layout. Layout 'x' and 'm_x' have the same transition probabilites. │   ├── smallTransProb.p │   └── lombardTransProb.p ├── engine # You may want to explore this directory. ├── environment.yml # Conda environment file for Linux. ├── environment_win.yml # Conda environment file for Windows. ├── environment_mac.yml # Conda environment file for Mac. ├── drive.py # Driver code to run and visualize your implementations. ├── none.py # Code for no inference. ├── autoDriver.py # A baseline naive driver. ├── estimator.py # The file where you place your estimation (or tracking) implementation. ├── intelligentDriver.py # The file where you have to implement your planning approach. └── util.py # Code for utilities needed in estimator.py
```

Simulator

The simulator can be started as follows. The functionality of the simulator can be adjusted with the flags listed below.

```
python3 drive.py
```

-a <autonomous or not>	Enabling autonomous driving (instead of manua 'd' keys.
-i <inference-method>	Use { "none", "estimator" } to estimate the belief
-l <map>	The layout/map can be "small" or "lombard".
-k	The number of StdCar(s) in the environment.
-m <multiple goals>	Multiple-goal version of the layout
-d <debug>	Debug mode where all cars are displayed on the
-p <parked>	All StdCars remain parked (so that they don't mc
-j	To invoke your intelligent driver.

Invoke the environment (without estimation) in the 'small' layout with 2 StdCars as follows:

```
python3 drive.py -d -k 2 -l small -a -i none
```

The estimator implementation can be tested with the default driver 'AutoDriver' on the above environment as follows:

```
python3 drive.py -d -k 2 -l small -a -i estimator
```

The intelligent driver implementation can be tested on layouts with multiple goals as follows:

```
python3 drive.py -d -k 2 -m -l small -a -i estimator -j
```

Note that the simulation automatically stops when the AutoCar *crashes* i.e., collides with a StdCar, hits the obstacles, or hits the boundary of the layout. After a crash, you can close the simulation window using the GUI or by pressing the 'q' key.

Where to code?

- **Estimation.** Please place your estimation code in estimator.py file. Please implement the function `def estimate(self, posX, posY, observedDist, isParked)` in `Estimator` class. Your implementation should modify the `self.belief` variable in place.
- **Planning.** Please place your planner code in intelligentDriver.py file. Please see `class IntelligentDriver` in `intelligentDriver.py`. Please implement the function `def getNextGoalPos(self, beliefOfOtherCars, parkedCars, chkPtsSoFar)`. The `createWorldGraph` method given in `intelligentDriver.py` provides a possible graph representation of the 2D grid world (extracted from the simulator state) that can be used by your planner. You can modify the method to create your own representation which suits your planning algorithm. A simple planning approach is available as a reference in the AutoDriver class of autoDriver.py.
- **Others.** Helper methods and classes defined in `util.py` file can be used in your implementation. You can find additional details in the descriptions of the functions provided in the corresponding python files. Note that the standard deviation of the microphone sensor, σ is Const.SONAR_STD. See `const.py`.

Evaluation

- **Parameters.** Evaluation may vary parameters such as the simulation layout settings, transition model, noise model, etc. You can assume a maximum of **15 cars** and a maximum grid size of **50x50**.
- grEstimation will be evaluated based on metrics such as the *accuracy of tracking* (with respect to the true position of the car in the simulator) and the *quality of approximation* (comparison of the estimated belief with exact inference). The belief can be recorded at any wall clock time upto a maximum of about **2 minutes**.
- Planning will be evaluated based on the fraction of goal locations/checkpoints visited successfully (without colliding with a StdCar or walls) in a given time interval (typically not exceeding **10 minutes** of runtime).

Submission Guidelines

- **This assignment is to be done individually or in pairs. The choice is entirely yours.**
- The assignment is intended for both COL333 and COL671 students. If the assignment is being done in pairs then the partner is to be selected only within your class. That is COL333 students should be paired within COL333. Similarly, masters students in COL671 should pair within their class of COL671.
- Please submit two files namely **estimator.py** with your best tracking algorithm and **intelligentDriver.py** with your best driver implementation. Please describe the core ideas of your algorithm in the text file called **report.txt** (max. 2 pages in standard 11 point Arial font). The first line of the report should list the name and entry number of student(s) who submit the assignment.
- Please create your **submission package** as follows. Submit a single zip file named *<A3-EntryNumber1-EntryNumber2>.zip or <A3-EntryNumber>.zip*. Please use the *"2019CS———"* style format while writing your entry number. Upon unzipping, this should yield a single folder with the same name as the zip file. Inside the folder, there should be three files: estimator.py, intelligentDriver.py, and report.txt. Please do not submit any other files.
- The assignment is to be submitted on Moodle. Exactly **ONE** of the team members needs to submit the zip file.
- **The submission is due at 7:30 pm on 15.11.2022.**
- **This assignment will carry 12% ($\pm$1%) of the grade.**

Other Guidelines

- Late submission deduction of (10% per day) will be awarded. Late submissions will be accepted till 2 days after the submission deadline. There are no buffer days. Hence, please submit by the submission date.
- Please strictly **adhere** to the input/output syntax specified in the assignment as the submissions are processed using scripts. Please do not modify files beyond the files you are supposed to modify. Failure to do so would cause exclusion from assessment/reduction.
- Failure to comply with the assignment instructions, will lead to exclusion from assessment and award of a reduction.
- Queries (if any) should be raised on **Piazza**. Please follow Piazza for any updates to the assignment statement. Do not use email or message on Teams for assignment queries.

- **Please only submit work from your own efforts.** Do not look at or refer to code written by anyone else. You may discuss the problem, however the code implementation must be original. Discussion will not be grounds to justify software plagiarism. Please do not copy existing assignment solutions from the internet, your submission will be compared against them using plagiarism detection software. Copying and cheating will result in a penalty of at least -10 (absolute). The department and institute guidelines will apply. More severe penalties such as F-grade will follow.
- This assignment builds on the Driverless Car framework implemented by Chris Piech and later