Starter code for this problem set has been made available online through Github; to get started download the code by running `git clone https://github.com/PrinciplesofRobotAutonomy/CS237B_HW2.git` in a terminal window.

You will submit your homework to Gradescope. Your submission will consist of (1) a single pdf with your answers for written questions (denoted by the \( \text{\#} \) symbol) and (2) a zip folder containing your code for the programming questions (denoted by the \( \text{\#} \) symbol).

**Important:** Remember, your written part must be typeset (e.g., \LaTeX\ or Word).

**More Important:** In this problem set, we will work with the HighD dataset. The dataset is free for non-commercial use, but you have to request access to download the dataset, which can take about 24 hours to be approved. Therefore, apply for the non-commercial license for HighD dataset [here](#) as soon as possible. In your request, mention that you’re a Stanford student taking CS237B student to expedite the process.

**Install Additional Software Dependencies**

Homework 2 continues to use the virtual environment from Homework 1 with added dependencies. To maintain consistency within this class, we already listed all dependencies on `requirements.txt` in the git repository you cloned. Navigate to the repository, and activate your virtual environment. For Anaconda, this will be:

```
$ conda activate cs237b
```

Once you’re in the virtual environment, run:

```
$ pip install -r requirements.txt
```

Now we’re ready to go!
Problem 1: Do elephants play chess?

(i) Read the paper by Brooks [1] on why elephants don’t play chess! Review the paper by providing

- a summary of the main points in your own words,
- a discussion of the advantages and disadvantages of the overall approach from the viewpoint of designing system architectures!
Problem 2: Model Checking

(i) Consider again the Kripke structure from the lecture, whose state transition graph and computation tree are shown above. Which of the following formulas hold in the initial (red) state?

(a) $AXX(A)$
(b) $EXX(A)$
(c) $EG((A \land X(B)) \lor (B \land X(C)))$
(d) $AF(AG(C))$
(e) $AG(EX(AG(C)))$

(ii) Suppose $a^*$ (the set of atomic propositions) is $\{A, B, C, D\}$. For each of the following formulas, draw a state transition graph which makes the formula true. Use the fewest states possible.

(a) $\neg(B) \land E(A \lor B) \land EG(\neg B)$
(b) $AF(A \land X(B) \land XX(C) \land XXX(D)) \land AG(\neg A \lor \neg B \lor \neg C \lor \neg D)$
Problem 3: Neural Network Verification

Neural network verification is an up-and-coming field of research. This question will give you a small taste of the types of tools this community is currently developing.

The verification of neural networks is still an open research question. What do we mean my verification of neural networks, and why do we need this? In homework 1, you might have thought that your Actor-Critic network is ‘complete’ because your cart-pole reaches the maximum reward most of time. However, it is difficult to evaluate, let alone predict, what would happen if the environment changes unexpectedly. What happens if our pole gets really, really long? What happens we have more gravity then usual? What happens if we suddenly can’t choose one of the actions? In other words, it has been observed that neural networks can react unexpectedly to even slight perturbations of their inputs, which raises safety concerns. Therefore, active research is being done to provide formal guarantee of performance of the neural net. Marabou is one of the latest such tools, and we will take a look at how Marabou can evaluate safety of a neural network.

Install the Marabou neural network verification package. You will have to install the package from source in order to get the python API installed. Note that we only tested building Marabou from source on Linux, so you should complete this assignment in Anaconda or Virtualenv environment.

- First, clone Marabou in Problem_3 folder: https://github.com/NeuralNetworkVerification/Marabou
- Next, build Marabou by running the following commands:

  ```
  $ cd path/to/marabou/repo/folder
  $ mkdir build
  $ cd build
  $ cmake .. -DBUILD_PYTHON=ON
  $ cmake --build .
  ```

- Make sure to add Marabou to your PYTHONPATH by running the following commands. Your path to Marabou folder should end with Marabou/build/bin/:

  ```
  $ export PYTHONPATH="$PYTHONPATH:/path/to/marabou/folder"
  $ export JUPYTER_PATH= "$JUPYTER_PATH:/path/to/marabou/folder"
  ```

- Afterwards, move mdl folder and Verification.py to Marabou.
- Verify your install by running the following command:

  ```
  $ ./build/Marabou resources/nnet/acasxu/ACASXU_experimental_v2a_2_7.nnet resources/properties/acas_property_3.txt
  ```

  Your command should run and end with either ‘SAT’ or ‘UNSAT’ conditions.

The verification of neural networks is still an open research question. As discussed in class, Marabou is able to use a satisfiability modulo theories (SMT) solver, which is a special flavor of a SAT solver. The solver answers the reachability queries about neural networks. Note, however, that Marabou also has limitations, but we won’t go in too much detail.\(^1\) Actor network was designed with the explicit goal of supporting this verification step, and this might not work with your favorite neural network.

To see what Marabou can do for us, we will take another look at the Actor network and verify that we trained a sufficiently good network using Marabou’s interval propagation capability. To get started, let’s copy over your frozen Actor model (frozen_actor.pb) from your Problem_3 folder of Homework 1 to mdl folder for this problem.

\(^1\)For more details, see tool paper.
(i) In `Verification.py`, take a look at `check_actor_correct`. We will check the following properties: if our system is in states between `state_low` and `state_high`, then the actor MUST produce a probability distribution that favors `correct_action` over the other one (remember that our actor can choose between two action only). Add upper and lower bound constraints in `check_actor_correct`.

(ii) Time to test your code! in Marabou folder, run

```
$ python Verification.py
```

What results are you getting? Does this mean your Neural network is safe or not? In your writeup, include your brief answer with justifications.
Problem 4: System-level verification via stress testing

With first autonomous vehicles prototypes, plenty of data collectors (cars with full sensor setup driven by a human), and other data collection methods (e.g. stationary), the problem shifts from getting enough data to getting the right data. Once enough data is collected, however, it becomes hard to marginally improve the dataset. The majority of newly collected data contains redundant scenarios which helps very little in assessing the performance of the system within the full problem space. For companies with access to a fleet of data generating vehicles, it is crucial to assess what type of data is collected and to measure how valuable the collected data is. Such data distribution analysis serves as the guideline for what type of data is collected as well as how to verify their system.

For this homework, you will work with an autonomous driving highway dataset obtained by a drone. Our goal is to identify scenarios within the dataset which are particularly hard (stressful) for the system. We want to evaluate those scenarios using different risk measures. In particular, you will investigate the following:

1. Searching for defined maneuver patterns in the dataset.
2. Determining relevant information for these maneuvers and extract them in the form of scenarios.
3. Evaluate the generated scenarios based on risk measures.

We strongly encourage you to take a look at all the code, even sections that you will not be directly working on.

Working with big datasets often yields the problem to remove irrelevant or redundant information as well as detecting particular relevant information. Scenarios are rated through there complexity, severity, and occurrence probability. In terms of risk assessment for automated driving the quality of a gathered dataset does not directly correlate to the number of miles driven. Complex scenarios, for example, passing construction sites or accidents are less frequently observed, and therefore more valuable to the verification and validation process. For example, driving several hours on empty highways might contribute less to the overall data than a few minutes on a busy and uniquely constructed road.

In our case, we use the HighD dataset \cite{2} to extract and assess lane change scenarios. Using human-based trajectories for certain maneuvers for example lane changing or exiting highways is common to meet minimum specifications for complicated driving tasks. Unfortunately for us, lane change events are significantly less frequent than lane following events. For accident analysis, lane change events in $\approx 150$ hours of driving data are too sparse, making it difficult to generate datasets representing the real human traffic space.

Let us start by having a look at HighD dataset itself: Once you obtain access to HighD dataset, copy all files under highd. \cite{2} It consists of 60 recordings which were measured among 6 different locations distributed along the highway. Figure 1 depicts a single frame from the dataset.

![Figure 1: An example of the HighD dataset \cite{3}.](image)

HighD data is structured with recordings. Each recording has one Traffic Object and several Traffic Participants. You can consider each recording happened in the Traffic Object, and Traffic Participants are

\footnote{For more detailed information on the HighD dataset make sure to read \cite{2}.}
surrounding, or pass by, Traffic Object. Each recording has three different files:

- **xx_recordingMeta.csv**: a meta file that describes recording (frame rate, location, time, etc.)
- **xx_tracksMeta.csv**: a meta file that describes each object detected during the recording (type, direction, some pre-calculated risk measures, etc.)
- **xx_tracks.csv**: the recorded data of Traffic Object listed for each frame (position, velocity, lane id, etc.)

In this exercise we want:

1. to identify all relevant other Traffic Participants for each Traffic Object.
2. to extract Traffic Participants that perform a lane change out of the identified Traffic Participants.

**Note:** Luckily for us, the lane in which Traffic Objects are driving is already labeled in our dataset. Therefore, lane change can be ‘detected’ when the lane ID changes.

**Part 1: Extraction of Relevant Scenarios**

The notebook `extraction.ipynb` will guide you through this part of the exercise. You will not have to adapt this file.

```bash
$ jupyter notebook extraction/extraction.ipynb
```

Take a look at `traffic_object.py` that defines the central class for this exercise (TrafficObject). We will implement some of the functions in this problem.

Extraction script will extract one instance for each Traffic Object (cell [7] in extraction.ipynb). Unique ID is added to Traffic Object, which will be added to a dictionary. After extraction, this dictionary is passed onto cell [8] so that each object has access to its environment.

(i) 📝 Take a look at one of the `xx_tracks.csv` files. Which column would you extract data from to detect a lane change of a traffic object?

(ii) 📝 Could you think of any other features which are not included in the dataset which might be also suited for a lane change detection?

(iii) 📝 We define our relevant Traffic Participants based on the driving direction and position overlap. For more detail, take a look at `determine_possible_relevant_others` function. Filter out all IDs which either have a different driving direction or do not overlap in existence time with the Traffic Object we are looking at.

(iv) 📝 After the initial filtering of Traffic Participants, we want to weed out more Traffic Participants based on the distance to each Traffic Participants. In `determine_relevant_others_distance`, filter out the IDs of Traffic Participants that are too far ahead or behind of the Traffic Object. The distance threshold is defined in (`config`).

(v) 📝 Finally, we use the 8 car model (Figure 3) to strip down the remaining IDs to maximum of 8 around the Traffic Object. Fill in the missing code in `determine_relevant_others_8car`. Watch out – this contains many edge cases!
Now we’re done filtering relevant traffic participants, we can extract only the Traffic Participants that contains lane changing scenario. Lane changing scenarios are extracted only from the beginning and the end of the lane changing event. The start time should be when the vehicle first crosses the lane, and the end time is when lateral velocity of Traffic Object reaches the threshold. Implement lane changing scenarios in `extract_maneuvers`.

Out of all the scenarios, find a scenario in which the 8 car filter makes a difference, and include two plots on the scenario.

**Part 2: Scenario Assessment using Risk Measures**
Now that we have extracted our data, we’re ready to analyze our data. In particular, we will assess each scenario using risk measures so that we can test our controllers on high-risk scenarios afterwards.

The two most common and frequently used risk measures are Time-Headway (thw) and Time-to-Collision (ttc). While many variations are used in the industry, we will use the simplest form, (1) and (2). Time-Headway expresses time $\tau$ that it takes for a following vehicle with velocity $v_{\text{follow}}$ to pass the vehicle with leading distance $d_t$. Time-to-Collision is similar to Time-Headway, only it takes the current velocity $v_{\text{lead}}$ of the leading vehicle into account.

$$\tau_{\text{thw}}(t) = \frac{d(t)}{v_{\text{follow}}(t)}$$  \hspace{1cm} (1)

$$\tau_{\text{ttc}}(t) = \frac{d(t)}{v_{\text{follow}}(t) - v_{\text{lead}}(t)} = \frac{d(t)}{v_{\text{rel}}(t)}$$  \hspace{1cm} (2)

Note that both measures are not taking into account of vehicle acceleration. This simplification could lead to misrepresentation of a situation, especially in accidents during which emergency braking tends to occur more often. Enhanced-Time-to-Collision (ettc) is similar to Time-to-Collision, only with relative acceleration $a_{\text{rel}}$ between the following and leading vehicle is now part of the metric.  

$$\tau_{\text{ettc}}(t) = \frac{v_{\text{rel}}(t)}{a_{\text{rel}}(t)} \left( 1 - \sqrt{1 - 2a_{\text{rel}}(t)\tau_{\text{ttc}}(t)/v_{\text{rel}}(t)} \right) ; \quad v_{\text{rel}}(t) \geq 2a_{\text{rel}}(t)\tau_{\text{ttc}}(t)$$  \hspace{1cm} (3)

Based on the definition of risk metrics, do we want high-risk or low-risk scenarios to stress test our controller?

Suppose your manager approved additional drone time to collect more data. As a thoughtful engineer, you would want to consider the value of newly collected data (‘marginal relevance’). For your new data, would you want a high-risk or low-risk scenario? (Hint: This is a trick question.)

For part 2, assessment.ipynb walks you through risk assessment of extracted driving scenarios.

$\ jupyter notebook assessment/assessment.ipynb$

---

For more information on these metrics can be found in (3) [5].
In `risks.py` implement the following risk measures:

- Time-Headway: \( \text{thw} \)
- Time-To-Collision: \( \text{ttc} \)
- Enhanced-Time-To-Collision: \( \text{ettc} \)

Fig 4 shows the risk distributions over a large dataset we collected in the past year. Compare risk distributions of recording 18 with this overall distribution (Figure 4). What is the marginal value of the newly acquired recording 18? In other words, how much value will recording 18 add to your entire dataset?
References


