Last Update: 12 March 2019

AC3R Documentation

I. Introduction

This document aims to briefly describe the process of generating virtual crash scenarios from crash reports, and the system architecture of AC3R.

II. Overall Process

In general, AC3R takes 6 steps to generate 3D scenarios from crash reports. Figure 1 briefly illustrates this scenario generation process.

1. Reads and Preprocesses Crash Reports

User selects crash report(s) to be analyzed, then AC3R normalizes the content (See section 3.2.2 in my Thesis) and identifies the properties of involved vehicles.

2. Extracts Environment Properties

AC3R extracts the properties of road, weather and lighting in the crash description. To do this, AC3R uses an Ontology, which stores all the concepts relevant to traffic accident, and leverages Dependency Parsing, an NLP technique, to identify useful dependencies between words in the crash description. AC3R checks if these words matches any concept in the Ontology. If the word is matched, AC3R retrieves the domain of the concept to learn whether it describes a road, weather or lighting property.

For a detailed description of this step, please refer to Section 3.2.1 and 3.2.3 in my Thesis.

3. Reconstructs the Accident Development

By leveraging Dependency Parser, AC3R extracts the sequence of actions for each vehicle in the crash reports, and their damaged position (left front, right rear, etc.) after the first impact.

Please see section 3.2.4 in my Thesis for further information.

4. Reconstructs Vehicle Actions and Environment using BeamNG syntax

From the sequence of actions, AC3R creates waypoints for each vehicle according to the characteristic of the vehicles's action. For example, if a vehicle stops throughout the scenario, then no trajectory is constructed, but if a vehicle travels on the street, then waypoints are created along the traveling direction of the vehicle to form a trajectory. Once the scenario starts, the vehicle travels along its waypoints to reach the crash point and hit the other vehicle.

After the waypoints are computed, AC3R creates a Lua file to store the dynamic elements of the scenario, such as how a vehicle travels along its trajectory, when the victim leaves its position to hit the striker, and how the scenario is considered passed/failed.

Additionally, a prefab file is created to store the environment and road properties of the scenario. AC3R also generates a JSON file to store metadata needed to run the scenario.

Section 3.2.4 and 3.2.5 provide a detailed description of this step.

5. Runs the Scenario in BeamNG

After the Lua, prefab and JSON files are generated, AC3R automatically calls a shell command to load BeamNG. Next, the scenario is automatically loaded and started. A scenario should simulate a crash within 30 seconds after its start, otherwise, it is deemed as “No Crash”, meaning the scenario does not match with the crash report.

When a crash happens, BeamNG records the damaged area(s) (Right Rear, Left Front, Right Middle, etc.) of each vehicle into a file for AC3R to analyze.

6. Checks if the Scenario Matches the Crash Description

From the vehicle's damage file, AC3R extracts the damaged area(s) of the vehicle, and compares them to the impact position described in the crash reports. If the damaged areas of BOTH vehicles match the crash description, then the scenario is regarded as totally match the crash description. If the damage areas of ONE vehicle match the crash description, the scenario is considered partially match the crash description. Otherwise, the scenario is considered unmatched with the crash report.

Section 4.1.2 specifies the evaluation metrics for damage comparison between the scenario and crash report.

'

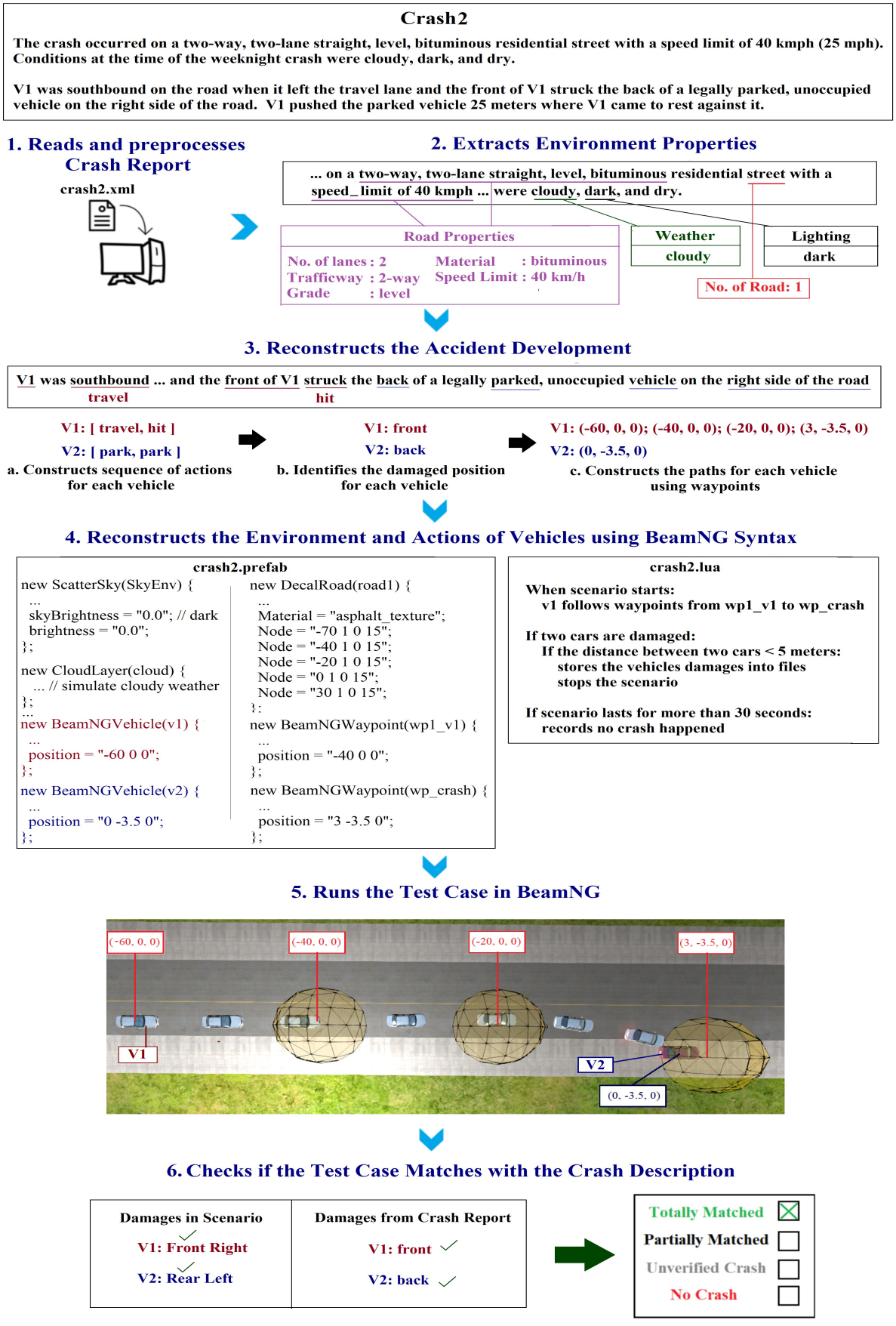

Figure 1: Procedure of generating scenarios from crash reports, and evaluating their accuracy.

III. System Architecture

AC3R comprises of the following component groups:

1. **Text Preprocessor** extracts vehicles' properties from XML tags in crash reports, and normalize crash descriptions.
2. **NLP Libraries** stems words and extract important grammatical relations from crash descriptions.
3. **Ontology Handler** searches and retrieves the name, attribute, and knowledge domain (environment or accident development) of existing concepts in the Ontology.
4. **Environment Analysis** extracts the properties of road, weather, and lighting from the crash description.
5. **Accident Development Constructor** builds the sequences of actions for each vehicle from the crash description.
6. **Trajectory Planner** constructs the trajectory of vehicles from the sequence of actions by creating an array of waypoints. Each waypoint is a 3D coordinate that a vehicle needs to pass through when it moves in the scenario.
7. **BeamNG Scenario Constructor** creates files containing the environment, accident development, and metadata of the generated scenarios. These files conform to the syntax of creating Lua, prefab, and JSON files in BeamNG.
8. **Scenario Runner** executes a shell command to automatically run the generated scenario in BeamNG.
9. **Crash Accuracy Evaluator** compares the damaged area of vehicles in the scenario with the impacted position described in the crash report, and determines how accurate the generated scenarios are.
10. **Utilities** contains the shared NLP and miscellaneous functions being used by multiple classes.

Figure 2 illustrates the modules in AC3R system.

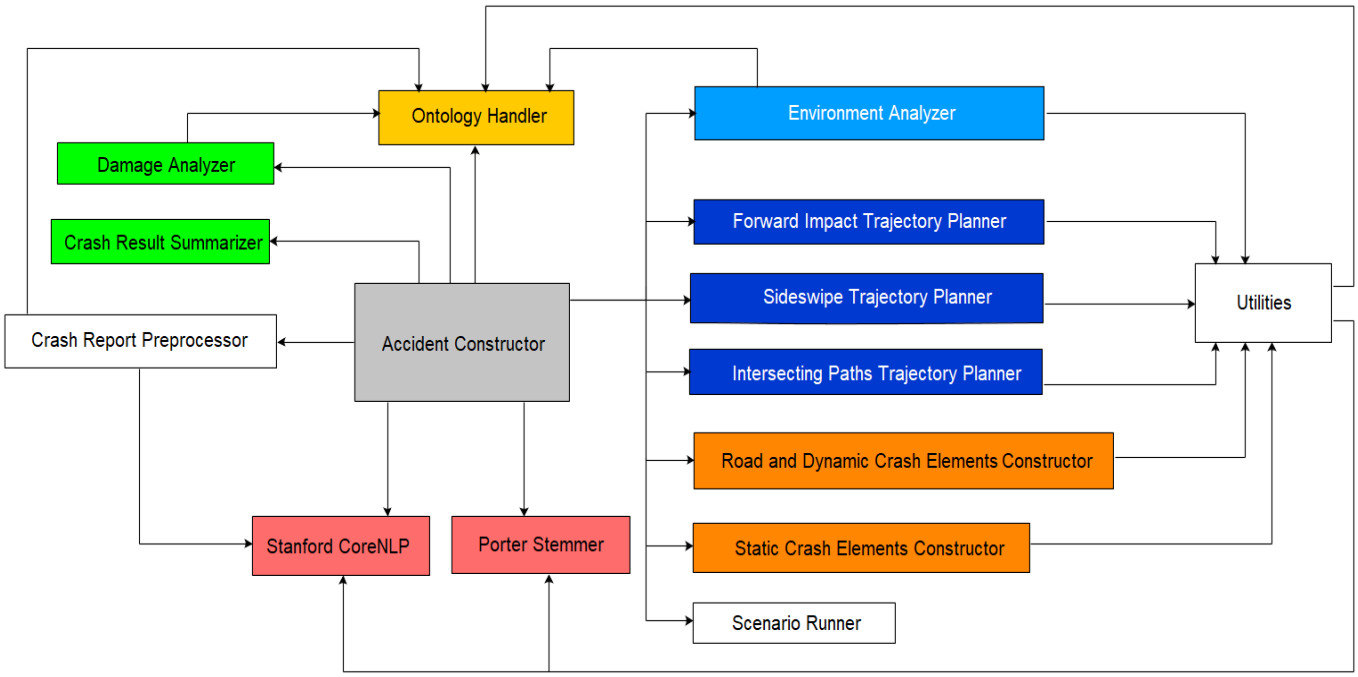

Figure 2: System Architecture of AC3R