Journal of Korea Robotics Society (2019) 14(4):340-347
https://doi.org/10.7746/jkros.2019.14.4.340 ISSN: 1975-6291 / eISSN: 2287-3961

자율 기기를 위한 속도가 제어된 데이터 기반 실시간 스트림 프로세싱

Rate-Controlled Data-Driven Real-Time Stream Processing for an Autonomous Machine

노 순 현[1] · 홍 성 수[2] · 김 명 선†
Soonhyun Noh[1], Seongsoo Hong[2], Myungsun Kim†

Abstract: Due to advances in machine intelligence and increased demands for autonomous machines, the complexity of the underlying software platform is increasing at a rapid pace, overwhelming the developers with implementation details. We attempt to ease the burden that falls onto the developers by creating a graphical programming framework we named Splash. Splash is designed to provide an effective programming abstraction for autonomous machines that require stream processing. It also enables programmers to specify genuine, end-to-end timing constraints, which the Splash framework automatically monitors for violation. By utilizing the timing constraints, Splash provides three key language semantics: timing semantics, in-order delivery semantics, and rate-controlled data-driven stream processing semantics. These three semantics together collectively serve as a conceptual tool that can hide low-level details from programmers, allowing developers to focus on the main logic of their applications. In this paper, we introduce the three-language semantics in detail and explain their function in association with Splash's language constructs. Furthermore, we present the internal workings of the Splash programming framework and validate its effectiveness via a lane keeping assist system.

Keywords: Autonomous Machine, Language Semantics, Programming Framework

1. 서 론

최근 기계 학습 기술의 진보로 인한 AI의 발전으로 인해 로봇, 드론 및 자율 주행 자동차와 같은 자율 기기(autonomous machine)들의 사용이 확대되고 있다. 자율 기기에서는 AI 컴퓨팅을 효과적으로 지원하기 위해 다양한 전자 기계 센서, 고성능 이기종 멀티 코어 프로세서, 분산 컴퓨팅 머신 구조 및 정교한 센서 융합 알고리즘이 장착 되어야 한다. 이에 따라 사용되는 소프트웨어 플랫폼의 복잡성이 자율 기기 응용 개발자가 직접 다루기 어려울 정도로 빠르게 증가하고 있다.

이런 문제를 극복하기 위해서는 자율 기기의 복잡한 소프트웨어 아키텍처의 설계, 개발 및 검증을 지원할 수 있는 효과적인 프로그래밍 프레임워크가 필요하다. 이 프레임워크는 응용 개발자에게 구현 세부사항을 숨기고 모델 기반 코드 생성 기능을 지원하기 위한 프로그래밍 추상화를 제공해야한다. 또한 자율 기기 제어 과정에서 실시간 스트림 처리를 지원하기 위해서는 응용 개발자가 End-to-End 타이밍 제약 조건을 명시할 수 있어야 하고, 런타임에 명시된 제약 조건이 위반될 경우 이를 감지하고 처리할 수 있어야 한다.

이러한 요구를 해결하기 위해 우리는 Splash라는 그래픽 프로그래밍 프레임워크를 개발하였다[1]. Splash의 디자인 목표는 네 가지이다. 첫째, Splash는 자율 기기의 스트림 처리를 지원하는 효과적인 프로그래밍 추상화를 제공해야한다. 둘째, 프로그래머

Received : Oct. 2. 2019; Revised : Oct. 25. 2019; Accepted : Oct. 29. 2019
※ This research was supported by a grant of the Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. R7117-16-0164, Development of wide area driving environment awareness and cooperative driving technology which are based on V2X wireless communication) for Seongsoo Hong and Soonhyun Noh. Also, this work was financially supported by Hansung University for Myungsun Kim.
1. Students, Electrical and Computer Engineering Department, Seoul National University, Seoul, Korea (shnoh@redwood.snu.ac.kr)
2. Professor, Electrical and Computer Engineering Department, Seoul National University, Seoul, Korea (sshong@redwood.snu.ac.kr)
† Assistant Professor, Corresponding author: IT Convergence Engineering Department, Hansung University, Seoul, Korea (kmsjames@hansung.ac.kr)

가 End-to-End 타이밍 제약 조건을 지정하고 런타임에 제약 조건의 위반을 모니터링 할 수 있어야한다. 셋째, 예외 처리(exception handling), 모드 변경(mode change) 및 센서 융합(sensor fusion)과 같은 핵심 유틸리티를 제공해야한다. 마지막으로, 시스템을 구현하는 동안 성능 최적화 기능을 프로그래머들에게 지원해야한다.

Splash는 프로그래머들에게 세 가지 핵심 언어 시맨틱을 지원한다. 이는 (1) 타이밍 시맨틱, (2) 순서 기반 전달 시맨틱, (3) 속도가 제어된 데이터 기반 스트림 처리 시맨틱이다. 이 세 가지 시맨틱을 통해 Splash는 프로그래머들에게 세부 구현 정보를 숨기는 고급 프로그래밍 추상화 기능을 제공할 수 있다. Splash는 응용 개발자가 직접 세부 구현을 하지 않아도 이 세 가지 시맨틱을 자동적으로 지원한다. 또한 런타임에 정의된 시맨틱이 잘 지켜지고 있는지를 모니터링하여 사용자가 시간 동기화 이슈 및 오류를 편리하게 처리 할 수 있도록 한다.

본 논문에서는 Splash의 가장 중요한 프로그래밍 추상화로서, 속도가 제어된 데이터 기반 스트림 처리 시맨틱에 중점을 두고 그 메커니즘을 제시한다. 먼저 Splash 프로그래밍 프레임워크 실현을 위한 개발 도구와 런타임 시스템 구현 내용을 설명하고, Splash로 프로그래밍 된 LKAS (lane keep assist system)를 사용하여 제안된 프로그래밍 프레임워크의 효용성을 입증한다.

2. Splash 프로그래밍 언어

본 절에서는 프로그래밍 추상화를 이해하는 데 도움이 되도록 기본 타이밍 시맨틱과 Splash 프로그래밍 언어에 대한 간략한 소개를 한다.

2.1 타이밍 시맨틱과 End-to-End 타이밍 제약 조건

Splash 상에서 처리되는 모든 데이터 아이템들은 자신이 생성된 시점을 타임 스탬프로 가지고 있다. 우리는 이를 birthmark라고 정의한다. 센서에서 생성되는 데이터들은 생성 시점이 birthmark로 기록되고, 다른 컴포넌트에서 출력되는 데이터들은 자신을 출력할 때 사용된 입력 데이터의 birthmark를 그대로 사용한다. Splash는 각 데이터에 기록되어 있는 birthmark과 개발자가 명시한 타이밍 제약 조건을 비교하여 각 조건의 위반 여부를 자동으로 감시하고 처리할 수 있다.

Splash는 세 가지 유형의 End-to-End 타이밍 제약 조건을 지원한다[2].

1) 단일 센서 값에 대한 최신성(freshness) 제약 조건: 센서 값이 시스템 내부에서 사용될 수 있는 유효 시간을 제한한다. 명시된 유효 시간이 경과되면 해당 센서 값은 버려진다.
2) 다중 센서 값에 대한 상관(correlation) 제약조건: 센서 융합에 사용되는 입력 센서 데이터 간의 birthmark의 최대 차이를 제한한다.
3) 프로세스의 출력 포트에 대한 속도 제한 조건: 초당 생성되는 출력 데이터 항목 수를 제한한다. 속도 제한은 Splash 런타임이 동일한 채널에서 연속적인 데이터 항목 사이의 지터(jitter)를 최소화하기 위해 최선을 다하지만 스트림 출력 포트에 지터가 전혀 없는 것을 보장하지 않는다는 점에서 소프트 실시간 제한 조건이다.

프로그래머는 Splash의 언어 구성을 통해 응용 프로그램 개발 중에 이러한 세 가지 유형의 타이밍 제약 조건을 명시적으로 나타낼 수 있다. 런타임 시 주석에 명시된 타이밍 제약 조건을 위반한 사항이 감지되면 Splash 런타임에서 예외가 발생한다.

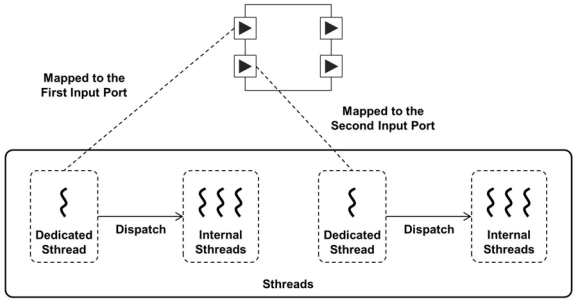
2.2 Splash의 언어 구성 요소

Splash 프로그램은 본질적으로 노드(node)와 두 노드 사이의 간선(edge)로 구성된 방향성 그래프(directed graph)이다. Splash 용어에서는 노드와 간선을 각각 컴포넌트와 채널로 정의한다. Splash 프로그램의 구성 요소는 원자적 컴포넌트와 복합 컴포넌트가 있다. 복합 컴포넌트를 팩토리라고도 한다. 원자적 컴포넌트는 (1) 프로세싱 컴포넌트, (2) 소스 컴포넌트, (3) 싱크 컴포넌트, (4) 융합 연산자의 네 가지 유형으로 더 분류된다.

컴포넌트는 데이터의 입출력을 하기 위해 스트림 입력 포트와 스트림 출력 포트를 가질 수 있다. 이전 컴포넌트의 스트림 출력 포트는 다음 컴포넌트의 스트림 입력 포트에 연결되며 이러한 연결은 하나의 채널을 생성한다.

프로세싱 컴포넌트는 입력 데이터를 사용하여 프로그래머가 정의한 연산을 수행하고 결과를 출력하는 컴포넌트로, Splash에서 가장 중요한 언어 구성 요소 이다. [Fig. 1]은 2 개의 스트림 입력 포트와 2 개의 스트림 출력 포트가 있는 프로세싱 컴포넌트를 나타낸다.

기본 운영 체제 및 컴퓨팅 플랫폼에서 명시 적으로 병렬 처리를 활용하기 위해 Splash는 프로세싱 컴포넌트에 대한 다중 스레드 프로세스 모델을 제공한다. 이 모델에서 프로세싱 컴포넌트는 sthread 그룹으로 구성된다. Sthread는 프로세싱 컴포넌트 내에서 실행되는 독립적인 논리적 엔터티(entity)이다.



[Fig. 1] Process and its sthreads

[Fig. 1]은 동시 서버 설계 패턴과 유사한 프로세싱 컴포넌트를 보여주고 있다[3]. 각 포트에 대한 전용 sthread와 작업자 sthread 역할을 하는 여러 개의 내부 스레드가 존재한다.

Splash는 스트림 데이터를 송수신하기 위한 스트림 입력/출력 포트 이외에도 단발성 이벤트를 전달하기 위한 이벤트 입력/출력 포트와 모드 변경 신호를 전달하기 위한 모드 변경 입력/출력 포트를 지원한다.

채널은 스트림 데이터의 전달 경로이며 스트림 출력 포트에서 스트림 입력 포트까지 실선으로 표시된다. Splash 언어는 스트림 데이터 아이템이 항상 birthmark 순서에 따라서 채널을 통과하도록 보장한다. 우리는 이를 순서 기반 전달 시맨틱이라고 부른다. 컴포넌트는 스트림 입력을 소비할 때까지 저장할 필요가 있는데, 이를 위해 FIFO 큐가 사용된다. 이 FIFO 큐는 컴포넌트의 입력 스트림 데이터를 수신하는 스트림 입력 포트에 각각 존재한다. 채널이 여러 입력 포트에 연결될 때 송신부 컴포넌트 출력 포트에서 생성 된 모든 데이터 항목들은 수신부 컴포넌트들의 입력 포트 상의 각 FIFO 큐에 복제된다.

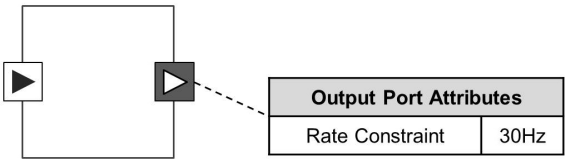
3. 속도가 제어된 데이터 기반 스트림 프로세싱

자율주행 기기의 프로그래밍은 제어 엔지니어, 소프트웨어 프로그래머 및 AI 엔지니어와 같은 다양한 기술 배경을 가진 개발자들에 의하여 수행된다. 제어 엔지니어는 제어 알고리즘을 주기적으로 호출하는 형태의 타이밍 중심적인 처리를 선호하고, AI 엔지니어는 채널에 데이터가 유입될 때마다 사전에 정의된 핸들러를 수행하는 데이터 중심 처리를 선호한다. 반면, 소프트웨어 프로그래머는 종종 이벤트 중심 처리에 의존한다. Splash는 이 세 가지 스타일을 모두 지원한다.

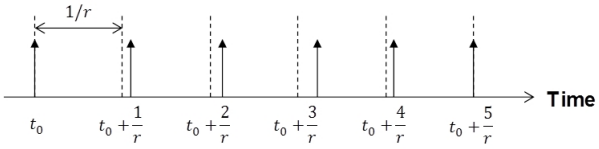
위 세 가지 중에서 Splash는 프로그램에서 달리 지정하지 않는 한 데이터 기반 처리를 기본으로 사용한다. 그러나 데이터 기반 처리는 지터를 발생시킬 수 있고 포트 상의 FIFO 큐에 존재하는 데이터 항목 개수가 급격히 증가할 수 있다는 부작용이 존재한다. 시스템 내부에서 통신 지연 및 실행 시간의 변동은 데이터 트래픽이 갑자기 집중적으로 한번씩 발송되는 현상을 나타내어 전체적인 제어 품질을 크게 저하시킬 수 있다. 이 문제를 해결하기 위해 Splash는 속도가 제어된 데이터 기반 스트림 처리 방식을 제공한다.

3.1 시맨틱스

프로그래머는 Splash 프로그램에서 프로세싱 컴포넌트의 스트림 출력 포트에서 원하는 데이터 생성 속도를 선택적으로 지정할 수 있다. 이러한 기능을 수행하는 모듈을 속도 제어기(rate controller)라고 하며, 데이터 항목의 지연을 최소화하는



[Fig. 2] Stream output port with a rate controller



[Fig. 3] Behavior of a rate controlled output stream port

범위에서 지터를 줄이고 FIFO 큐의 최대 크기를 제한한다. 속도 제어기의 존재는 프로그래머에게 숨겨져 있다. [Fig. 2]는 원하는 데이터 생성 속도 주석에 적힌 스트림 출력 포트를 나타낸다. 속도 제어기로 기능이 보강 된 스트림 출력 포트에는 다른 포트와 구별하기 위해 다른 기호가 사용된다.

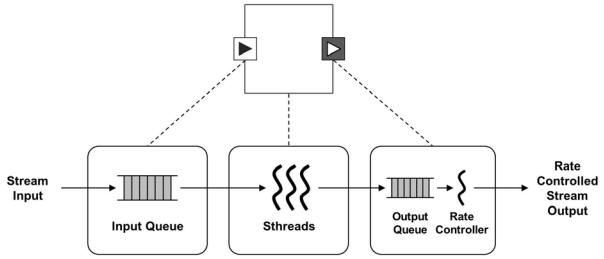
속도 제한 조건이 r인 스트림 출력 포트에는 크기 $1/r$의 타이밍 윈도우가 할당된다. 속도 제어 스트림 출력 포트의 의미는 각 타이밍 윈도우에서 정확히 하나의 데이터 항목을 생성 할 수 있음을 의미한다. 즉, 포트는 시간 간격 $[t_0+n/r, t_0+(n+1)/r]$마다 하나의 데이터 항목을 생성한다. 여기서 t_0은 첫 번째 데이터 항목이 생성 된 시간이고 n은 인덱스를 나타낸다. [Fig. 3]은 속도 제어 출력 스트림 포트의 올바른 동작을 보여준다.

속도가 제어된 스트림 출력 포트 에서는 두 가지 유형의 출력을 생성한다. 첫 번째 출력은 실제 데이터 항목이고 두 번째는 외삽(extrapolation) 명령이다. 속도가 제어된 스트림 출력 포트는 현재 타이밍 윈도우 내에 전송할 데이터 항목이 있는 경우 실제 데이터 시간을 생성하고 그렇지 않으면 외삽 명령을 출력한다. 프로세싱 컴포넌트가 스트림 입력 포트에서 외삽 명령을 수신하면 반드시 외삽 태스크를 수행하는 함수를 호출하여야 한다.

3.2 런타임 메커니즘

속도 제어 스트림 출력 포트의 런타임 메커니즘은 [Fig. 4]와 같이 출력 큐와 속도 제어기로 구성된다. 프로세싱 컴포넌트 내부의 스레드는 데이터 항목을 출력 큐에 삽입 후 큐에 저장된 데이터 항목의 최신 제약 조건을 통해 출력 큐의 크기를 조절한다. 출력 큐의 크기는 아래와 같다.

$$S_{\max} = \lfloor r \times f \rfloor \quad (1)$$



[Fig. 4] Runtime mechanism of a rate controller

위 식의 f는 데이터 항목의 최신성(freshness) 제약 조건이다

어떤 스레드가 데이터 항목을 삽입하려고 시도 할 때 출력 큐가 가득 찬 경우, 출력 큐의 맨 앞쪽에 있는 데이터 항목은 삭제되고 새로운 데이터 항목이 큐의 맨 뒤쪽에 삽입된다.

속도 제한 조건이 r인 속도 제어기는 $1/r$ 간격마다 호출된다. b를 마지막으로 전송 된 데이터 항목의 birthmark로 정의하면, 속도 제어기는 주기적으로 호출되면서 출력 큐의 헤드에서부터 birthmark가 b보다 큰 첫 번째 데이터 항목 d를 검출한다. 그러한 d가 있으면, 출력 큐에서 d 이전의 모든 데이터 항목을 버리고 를 출력한다. 그렇지 않으면 외삽 명령이 생성되고 외삽된 데이터 항목은 $b+1/r$인 birthmark를 갖는다. 이를 통해 순서 기반 전달 시맨틱을 보장할 수 있다.

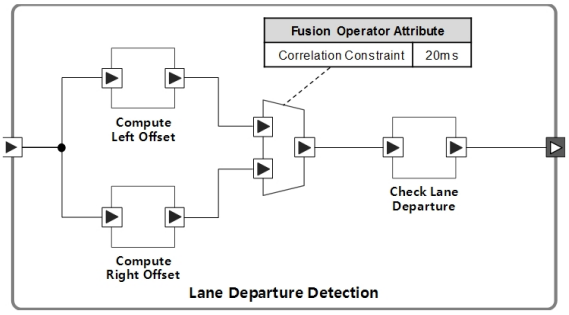
4. Splash의 도구와 런타임

본 절에서는 Splash의 도구 및 런타임 작동 방식에 대해 기술한다. Splash를 사용하여 개발 한 LKAS (Lane Keep Assistant System) 응용 예를 사용하여 도구 및 런타임의 메커니즘을 설명한다.

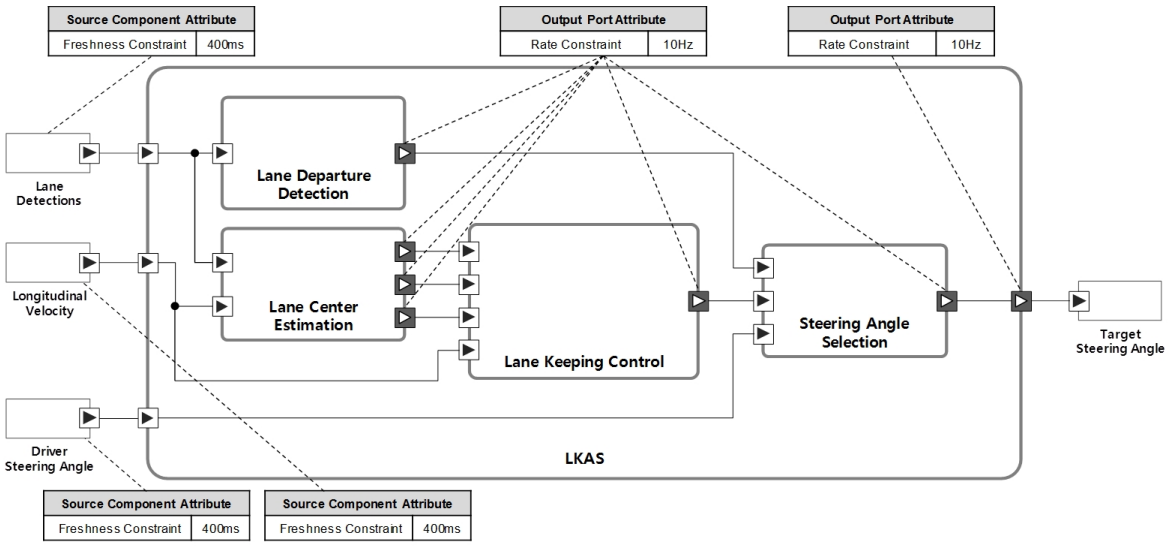
4.1 Splash를 사용한 LKAS

본 연구에서는 [4]에 주어진 알고리즘을 기반으로 Splash를 사용하여 LKAS를 설계하였다. 이 응용 프로그램은 자율주행 차량이 감지 된 차선 내부에 들어 올 수 있도록 조향 각도를 자동으로 조정한다. [Fig. 5]는 LKAS의 최상위 팩토리를 보여준다. 입력에는 차선 센서의 차선 감지, 차량의 종 방향 속도 및 운전자 조향 각도가 사용되며 출력은 차량의 목표 조향 각도이다. 최상위 팩토리는 4 개의 하위 팩토리로 구성된다: (1) 차선 이탈 감지(lane departure detection), (2) 차선 중심값 추정(lane center estimation), (3) 차선 유지 관리(lane keeping control), (4) 조향각 선택(steering angle selection).

[Fig. 6]은 차선 이탈 감지 팩토리를 나타낸다. 왼쪽 차선과 오른쪽 차선 경계에서 자율주행 차량의 오프셋 거리를 계산하여 차량이 차선 경계에 너무 가까운 지 확인하고, 융합 연산자를 사용하여 두 오프셋을 병합한 후 오프셋이 사전 정의 된 임계 값보다 낮은 지 확인한다. 차선 중심값 추정 팩토리는 차선의



[Fig. 6] Lane departure detection factory



[Fig. 5] Lane keeping assist system factory

```
1:  {
2:    "name": "LaneDepartureDetection",
3:    "processing_block": [
4:      {
5:        "name": "ComputeLeftOffset",
6:        "stream_input_port": {"source_channel": "//@top_level_factory/@factory.0/@channel.0"},
7:        "stream_output_port": {"target_channel": //@top_level_factory/@factory.0/@channel.1"}
8:       },
9:          …
10:   ],
11:   "fusion_operator": {
12:         …
13:   },
14:   "channel": [
15:       …
16:   ],
17:   …
18: }
```

[Fig. 7] JSON output of schematic editor for lane departure detection factory

중심 곡선과 차량의 측면 편차 및 진행 각도를 계산한다. 차선 유지 관리 팩토리는 차선 중심값 추정 팩토리의 출력과 차량의 종 방향 속도를 입력으로 사용한다. 그런 다음 차선을 유지하기 위해 필요한 조향 각도를 출력한다. 조향각 선택 팩토리는 차선 이탈 감지 팩토리의 출력에 기초하여 LKAS 또는 운전자가 차량을 제어 할 것인지를 결정한다.

이 응용 프로그램에는 세 종류의 타이밍 제약 조건이 명시되어 있다. 세 가지 소스 컴포넌트 각각은 최신성 제약 조건으로 400ms를 설정하고 있으며 LKAS 팩토리의 스트림 출력 포트는 속도 제한으로 10HZ를 설정한다. 이러한 제약 조건은 [Fig. 5]에 나타나있다. 또한 [Fig. 6]처럼 차선 이탈 감지 팩토리에 사용되는 융합 연산자를 20ms의 상관 제약 조건으로 설정하였다.

4.2 Splash 도구

Splash는 프로그래머에게 스키매틱 편집기(schematic editor)와 코드 생성기(code generator)의 두 가지 개발 도구를 제공한다. Splash 프로그래밍 언어는 컴포넌트 간의 상호 작용을 정의하고, 컴포넌트 내부의 서브 프로그램의 정의에는 C++와 같은 호스트 언어[5]가 사용된다. 프로그래머는 스키매틱 편집기를 사용하여 프로그램을 작성한다. 프로그래머가 프로그램 작성을 완료 한 후 스키매틱 편집기는 코드 생성기에 대한 입력으로 사용할 JSON 파일을 생성한다. [Fig. 7]은 차선 이탈 감지 팩토리의 JSON 파일 출력을 나타내고 있다. 여기에는 팩토리 및 처리 블록, 융합 연산자, 채널 및 스트림 입력/출력 포트와 같은 내부 언어 구성에 대한 정보가 포함된다.

코드 생성기는 스키매틱 편집기에서 생성 된 JSON 파일을 입력으로 사용하여 IDL(인터페이스 정의 언어) 파일[6]과 C++로 작성된 템플릿 소스 코드 파일의 두 가지 유형의 출력을 생성한다. IDL 파일은 각 포트 인터페이스의 데이터 유형을 설명하며 [Fig. 8]은 차선 이탈 감지 팩토리의 IDL 출력을 보여주고 있다.

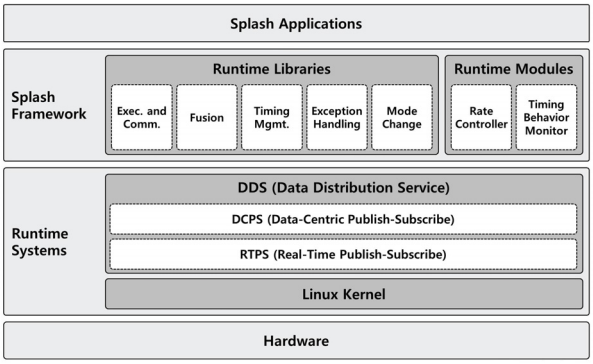
```
1:  module LaneDepartureDetection
2:  {
3:      struct data0
4:      {
5:          double left_curvature;
6:          double left_curvature_derivative;
7:          // …
8:      };
9:      struct data1
10:     {
11:         bool detected;
12:     };
13:     // …
14: };
```

[Fig. 8] IDL output for lane departure detection factory

```
1:  int main(void)
2:  {
3:      ProcessingComponent pc;
4:      StreamInputPort
5:      <LaneDepartureDetection::data0> sin0;
6:      StreamOutputPort
7:      <LaneDepartureDetection::data1> sout0;
8:      pc.initialize("Compute Left Offset");
9:      sin0.initialize();
10:     sout0.initialize(30);
11:     sin0.attach(&pc, "topic1");
12:     sout0.attach(&pc, "topic2");
13:     pc.run();
14: }
15:
16: template<typename t> void
17: ProcessingComponent<t>::usr_func(t input)
18: {
19:     LaneDepartureDetection::data1 output
20:     // Write user logic here
21:     this->write(&output, "topic1");
22: }
```

[Fig. 9] Source code for compute left offset processing component

[Fig. 9]는 차선 이탈 감지 팩토리의 왼쪽 오프셋 계산 프로세싱 컴포넌트에 대한 코드 생성기에서 작성된 템플릿 코드의 출력을 나타낸다. 그림에 표시된 것처럼 템플릿 코드는 구성부(3-12 행)과 실행부(13 행)의 두 세그먼트로 이루어진다. 구성부에서는 프로세싱 컴포넌트 및 해당 내부 스트림 입/출력 포트 오브젝트가 선언(3-7 행)되고 초기화된다(8-10 행). 그리고 나서, 스트림 입/출력 포트 객체는 프로세싱 컴포넌트 객체(11-12 행)에 결합된다. 실행이 되면 데이터 항목이 입력 포트(13 행)에 들어 오기를 기다린다. 데이터 항목이 도착하면 왼쪽 오프셋 계산 프로세싱 컴포넌트의 사용자가 작성한 함수가 호출된다(16-22 행). 프로그래머는 해당 함수를 20 행에 작성한다.



[Fig. 10] Splash runtime architecture

템플릿 소스 코드를 작성한 후 프로그래머는 호스트 언어의 컴파일러와 IDL 프리 프로세서를 사용하여 실행 가능 이미지를 컴파일하고 빌드한다[6].

4.3 Splash 런타임

Splash 런타임은 [Fig. 10]과 같이 두 개의 소프트웨어 계층으로 구성된다. 상위 계층은 호스트 언어로 작성된 런타임 라이브러리 및 모듈로 구성된 Splash 프레임 워크이다. [Fig. 9]처럼 프로그래머가 보강한 템플릿 코드는 Splash 프레임 워크에서 제공하는 라이브러리를 사용한다. 런타임 라이브러리는 기능에 따라 5가지 유형이 존재한다: (1) 실행 및 통신, (2) 융합, (3) 타이밍 관리, (4) 익셉션 처리, (5) 모드 변경. Splash 프레임 워크에는 (1) 속도 제어기와 (2) 타이밍 동작 모니터의 두 가지 런타임 모듈이 있다.

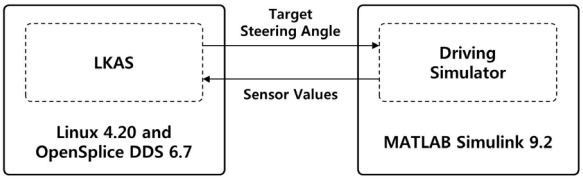
하위 계층에는 DDS(데이터 배포 서비스) 및 Linux 기반의 런타임 시스템이 있다. DDS는 실시간 발행-구독(Publish-Subscribe) 통신 방식[7]이며 오픈 소스인 OpenSplice DDS를 사용한다[8].

5. 실 험

본 절에서는 제안된 프로그래밍 프레임워크로 LKAS를 구현하고 성능을 측정하여 Splash의 유효성을 검증한다. 먼저 구현의 실행 환경과 실험에 사용된 메트릭(Metric)을 설명한 후 실험 결과와 분석을 기술한다.

5.1 검증을 위한 실험 환경과 성능 메트릭

본 연구에서는 LKAS 구현을 검증하기 위해 MathWorks의 Simulink를 사용하여 주행 환경을 시뮬레이션 하였다[4]. [Fig. 11]은 전체 실험 환경을 나타낸다. LKAS는 시뮬레이터로부터 센서 값을 수신하고 목표 스티어링 휠 각도를 출력한다. 그 후 시뮬레이터는 스티어링 휠 각도를 입력으로 받는다. 즉, 전체적인 소프트웨어 구성은 폐루프(closed loop) 형태로 이루어진다.



[Fig. 11] Software components of the validation platform

Splash의 속도 제어기의 효과를 검증하기 위해 LKAS의 세 가지 다른 메트릭을 사용하였다. 먼저 속도 제어기의 속도 조절 능력을 평가하기 위해 LKAS의 출력 지터를 측정하였다. 둘째, 속도 제어기로 인한 오버헤드를 검출하기 위하여 End-to-End 지연시간을 측정하였다. 마지막으로 속도 제어기의 출력 큐에 있는 데이터 항목 수의 변화를 측정하여 속도 제어기가 출력 큐에 있는 데이터 항목 수를 제한하는 기능을 확인하였다.

5.2 실험 결과

[Fig. 5]에 나타낸 것처럼 10Hz 속도 제한 조건을 갖는 속도 제어기를 각 팩토리의 스트림 출력 포트에 연결하였다. 그런 다음 속도 제어기가 있는 경우와 없는 경우 각각의 출력 지터를 측정하였다. 출력 지터는[9]에 사용된 정의를 사용하였으며 아래 식 (2)와 같다.

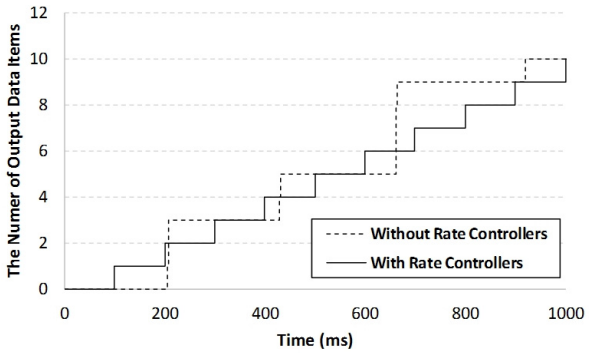
$$J = \sqrt{\frac{\Sigma_{j=1}^{h}(E(O)-o_j)^2}{h}} \tag{2}$$

식 (2)에서 h는 측정 횟수를 나타내며, O는 h개의 연속된 출력들 간의 시간 차이들을 원소로 하는 집합이다. $o_j \in O$를 만족하고 $E(O)$는 원소들의 평균을 뜻한다.

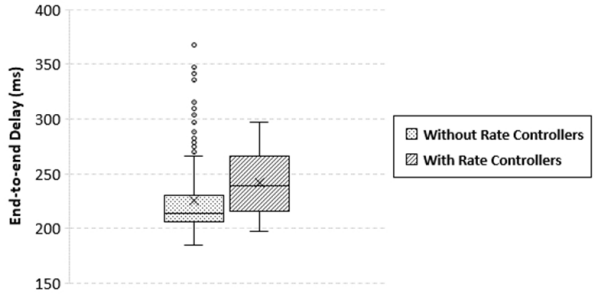
식 (2)에서 정의한 메트릭을 사용했을 때, 속도 제어기가 있는 경우 출력 지터가 0.14 ms인 반면 속도 제어기가 없는 경우에는 출력 지터가 122.36 ms를 나타내었다.

속도 제어기의 효과를 더 잘 설명하기 위해 [Fig. 12]에 LKAS에서 출력되는 누적 된 데이터 항목 수를 나타내었다. 예상한 대로, 속도 제어기가 있는 LKAS는 100ms마다 출력 데이터 항목을 생성하는 반면 속도 컨트롤러가 없는 LKAS는 출력이 불규칙적으로 생성되었다.

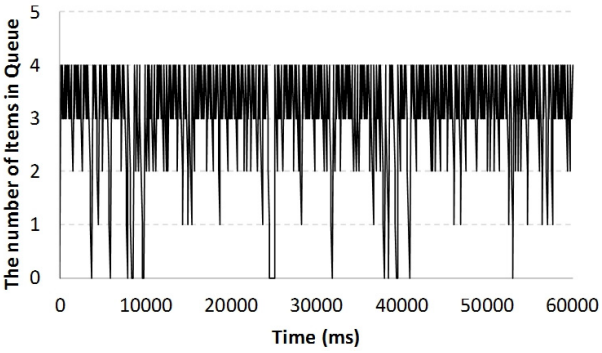
두 번째 실험에서는 제안된 프레임워크의 오버 헤드를 알아보기 위하여 각 데이터 항목에 대한 LKAS의 전체 지연 시간을 측정하였다. 이때 입력 데이터가 소스 컴포넌트에서 싱크 컴포넌트에 도달하는 데 걸리는 시간을 End-to-End 지연 시



[Fig. 12] Comparison of the number of the accumulated output data items



[Fig. 13] Comparison of the end-to-end latency



[Fig. 14] The number of data items in the output queue

간으로 정의한다. [Fig. 13]은 그 결과를 box-plot을 통하여 보여준다. 평균 End-to-End 지연 시간은 226 밀리 초에서 242 ms로 6.61 % 증가한 반면 최대 End-to-End 지연 시간은 368ms에서 297ms로 23.91 % 감소하였다. 평균 End-to-End 지연 시간의 증가는 출력 큐에서의 지연으로 발행한다. 하지만 속도 제어기가 처리하기에는 너무 오래된 데이터 항목을 효과적으로 필터링했기 때문에 최대 End-to-End 지연 시간은 단축되었다.

마지막 실험에서는 속도 제어기의 출력 큐에 있는 데이터 항목 수를 측정하고 그 결과를 [Fig. 14]에 나타내었다. 그림을 통하여 속도 제어기가 출력 큐에 항목 수를 성공적으로 제한시켰다는 것을 확인할 수 있다.

6. 관련 연구

학계와 산업계에는 다양한 스트림 처리 프레임워크들이 자율기기의 응용 개발에 사용되고 있다. 대표적인 예로는 MathWorks사의 Simulink[10], Intempora 사의 RTMaps[11], UC Berkeley의 Ptolemy II[12]가 있다. 이들은 Splash와 마찬가지로 방향성 그래프를 기반으로 스트림 처리 과정을 표현한다. 본 장에서는 이 프레임워크들과 Splash 간에 어떤 차이점이 있는지를 비교한다.

Simulink는 제어와 신호 처리 분야에서 널리 쓰이는 상용화된 모델링 프레임워크이다. Simulink는 주기적인 샘플링 시점마다 개발자가 정의한 연산을 수행하는 타이밍 중심적인 프로그래밍 추상화를 제공한다. 또한 Simulink는 이벤트 중심적인 프로그래밍도 같이 지원한다. 하지만 Simulink는 실시간 스트림 처리 응용의 개발을 지원한다는 측면에서 네 가지 한계가 존재한다. 첫째, 시간 제약의 명시와 처리를 지원하지 않는다. 둘째, 순서 기반 전달 시맨틱을 지원하지 않는다. 셋째, 데이터 중심적인 프로그래밍을 지원하지 않는다. 넷째, 모드 전환이나 예외 처리와 같은 프로그래밍 기법을 지원하지 않는다.

RTMaps는 여러 개의 센서들과 액츄에이터들이 탑재된 자율기기 상에서 응용을 개발하기 효과적인 프로그래밍 프레임워크이다. RTMaps는 Splash와 마찬가지로 각 스트림 데이터 아이템에 생성 시간을 타임 스탬프로 찍고, 이에 기반하여 End-to-End 타이밍 제약 조건 중 최신성 제약과 상관 제약을 명시하고 자동적으로 처리하는 기능을 제공한다. 하지만 RTMaps는 속도 제한 조건은 명시적으로 다루지 않기 때문에 이를 개발자가 직접 다뤄야 한다는 단점이 있다. 또한 RT Maps는 순서 기반 전달 시맨틱을 벡터화 연산자(vectorizer operator)와 같은 일부 연산자에서만 지원한다.

Ptolemy II는 다양한 프로세스 네트워크 모델을 지원하는 학술적인 프로그래밍 프레임워크이다. PtolemyII는 프로그래머가 동시에 두 종류 이상의 모델을 동시에 다룰 수 있도록 해주고, 모드 전환이나 예외 처리와 같은 프로그래밍 기법을 효과적으로 지원한다. 하지만 Ptolemy II 상의 대부분의 프로세스 네트워크 모델들은 실시간 스트림 처리를 위한 시간 제약의 명시와 처리를 지원하지 않는다. 또한 Ptolemy II는 동시성 프로그래밍을 위한 지원을 하지 않는다는 한계점이 존재한다.

7. 결 론

본 논문에서는 자율주행 기기 개발을 위한 Splash라는 그래픽 프로그래밍 프레임 워크를 제시하였다. 우리는 세 가지 핵심 언어 시맨틱, 즉 타이밍 시맨틱, 순서를 보장한 전달 시맨틱 및 데이터 기반 속도를 제어하는 스트림 프로세싱 시맨틱을 소개했다. 이 중 데이터 기반 속도를 제어하는 스트림 프로세

싱 시맨틱을 강조하고 그 메커니즘을 제시하였다. 또한 Splash 프로그래밍 프레임 워크를 실현하기 위해 구현 한 개발 도구와 런타임 시스템에 대해서도 기술하였다. 마지막으로 LKAS를 통해 Splash 프로그래밍 프레임 워크의 효과를 검증하였다. 향후 Splash를 사용하여 자율 주행 차량을 위한 다양한 실시간 애플리케이션을 설계하고 구현할 계획이다. 이를 위해 Splash는 성능과 안정성 측면에서 더욱 최적화 될 것으로 예상된다.

References

[1] S. Noh and S. Hong, "Splash: a graphical programming framework for an autonomous machine," *2019 16th International Conference on Ubiquitous Robots (UR)*, Jeju, Korea, DOI: 10.1109/URAI.2019.8768652.

[2] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing real-time requirements with resource-based calibration of periodic processes," *IEEE Transactions on Software Engineering*, vol. 21, no. 7, Jul., 1955, DOI: 10.1109/32.392979.

[3] C. Breshears, *The art of Concurrency*, 1st, O'Reilly Media, Inc., 2009.

[4] *Lane Keeping Assist with Lane Detection*, [Online], https://www.mathworks.com/help/mpc/ug/lane-keeping-assist-with-lane-detection.html, Accessed: September 20, 2019.

[5] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773-801, May, 1995.

[6] *Vortex OpenSplice documentation*, [Online], https://istkb.adlinktech.com/article/vortex-opensplice-documentation, Accessed: September 20, 2019.

[7] Data Distribution Service (DDS) Version 1.4, formal/2015-04-10, Apr., 2015.

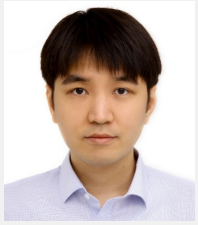
[8] *Vortex OpenSplice*, [Online], https://github.com/ADLINK-IST/opensplice, Accessed: September 20, 2019.

[9] N. Kim, M. Ryu, S. Hong, M. Saksena, C.-H. Choi, and H. Shin, "Visual Assessment of a Real-Time System Design: A Case Study on a CNC Controller," *17th IEEE Real-Time Systems Symposium*, Washington, DC, USA 1996, DOI: 10.1109/REAL.1996.563726.

[10] *Simulink*, [Online], https://www.mathworks.com/help/simulink/index.html, Accessed: September 20, 2019.

[11] N. d. Lac, C. Delaunay, and G. Michel, "RTMaps: real time, multisensor, advanced prototyping software," *3rd National Conference on Control Architectures of Robots*, Bourges, France, 2008.

[12] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity - the Ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127-144, Jan., 2003.



노 순 현

2012 서울대학교 전기정보공학부(학사)
2013~현재 서울대학교 박사과정

관심분야: 자율주행 프레임워크, 리눅스 커널 자원 관리 최적화(CPU/메모리)

홍 성 수

1986 서울대학교 컴퓨터공학부(학사)
1988 서울대학교 컴퓨터공학부(석사)
1994 Computer Science, University of Maryland (공학박사)
1995~현재 서울대학교 교수

관심분야: 실시간 시스템 설계, 실시간 운영체제, 임베디드 시스템, 교차 계층 최적화

김 명 선

2002 LG전자 전송 연구소
2011 삼성전자 DMC 연구소
2016 서울대학교 전기컴퓨터공학부(공학박사)
2019 삼성전자 SR 연구소
2019~현재 한성대학교 IT융합공학부 조교수

관심분야: 인공지능 가속기, 임베디드 시스템, HW/SW Co-design