공학석사 학위논문

Splash 빌드 유닛을 이용한 분산 시스템으로의 Splash 응용 프로그램 자동화 배포

Automated Deployment of a Splash Application to the Distributed System via Splash Build Unit

2021 년 8 월

서울대학교 대학원

전기·정보공학부

최 우 영

Splash 빌드 유닛을 이용한 분산 시스템으로의 Splash 응용 프로그램 자동화 배포

지도 교수 홍 성 수

이 논문을 공학석사 학위논문으로 제출함

2021 년 05 월

서울대학교 대학원

전기·정보공학부

최 우 영

최우영의 공학석사 학위논문을 인준함

2021 년 06 월

위 원 장 _____ 김태환

부위원장 _____ 홍성수

위 원 _____ 심규석

초 록

Splash는 점점 복잡해지는 자율기기(autonomous machine)를 위한 응용 프로그램 개발을 효과적으로 할 수 있는 프로그래밍 프레임워크이다. Splash는 그래픽 사용자 인터페이스를 이용한 프로그래밍 추상화를 제공할 뿐만 아니라 실시간 제어 시스템에 필수적인 실시간 스트림 처리, 센서 퓨전, 모드 변경, 예외 처리 등과 같은 기능들을 제공한다. Splash는 ROS 2에 이미 존재하는 기능의 이점을 이용하기 위해서 ROS 2 기반으로 개발되었다. 따라서, Splash를 사용하여 작성된 응용은 Splash 런타임 라이브러리와 ROS 2 런타임 라이브러리가 설치된 단일 혹은 분산 컴퓨팅 환경에서 구동된다. 하지만 분산 컴퓨팅 환경에서 Splash 응용을 사용하기 위해선 각 머신의 환경에 맞게 응용과 런타임을 빌드 및 설치해야 하고, 이는 소프트웨어 업데이트 및 유지보수를 하는데 시간과 비용이 많이 들도록 한다. 이런 불편함은 분산 컴퓨팅 머신에 확장성(scalability), 가용성(availability), 관리성(manageability)을 고려한 소프트웨어 배포(deployment)의 필요성을 제기한다. 본 논문에서는 Splash 응용 프로그램의 자동화 배포 및 관리를 위한 도커(Docker) 기반 Splash 빌드 유닛을 구현한다. Splash 빌드 유닛을 통해서 개발자는 소프트웨어 빌드 및 배포 프로세스를 자동화하고 프로그램의 병렬성(parallelism) 및 동시성(concurrency)을 최적화할 수 있다. Splash 빌드 유닛을 이용한 소프트웨어 자동화 배포에 대한 유용성을 검증하기 위해서 분산 시스템에서 Splash 빌드 유닛을 이용하여 딥러닝 기반 자율 주행 응용 프로그램의 예제를 진행하였다.

주요어 : Splash, 프로그래밍 프레임워크, 멀티코어 분산 시스템, 자동화 소프트웨어 배포, 자율 기기, ROS 2
학 번 : 2019-26977

목 차

표 목차

그림 목차

제 1 장 서 론

최근 딥러닝을 포함한 기계 학습의 폭발적인 발전으로 자율주행차, 로봇, 드론, 등과 같은 자율 기기의 연구 개발과 상용화가 활발하다. 폭스바겐, 현대자동차와 같은 자동차 업계 뿐만 아니라 구글, 애플과 같은 IT 기업도 딥러닝 기술을 탑재한 자율 주행 차량을 개발 하고 있다[1]. 점점 복잡해지는 딥러닝 기반 응용 프로그램이 자율 기기에 사용됨에 따라 다양한 센서, GPU, 인공신경망 연산 가속기 등을 포함한 고성능 멀티코어 컴퓨팅 자원들이 자율 기기에 탑재되어 하드웨어의 복잡성이 증가하였다[2][3]. 따라서 자율 기기 소프트웨어 개발을 위해 이런 복잡한 자율 기기 하드웨어를 효과적으로 다룰 수 있는 다양한 프로그래밍 추상화를 제공하는 프로그래밍 프레임워크의 필요성이 대두되었다.

그래픽 프로그래밍 프레임워크인 Splash는 자율 기기를 위한 응용 프로그램의 다양한 요구사항을 만족시키기 위해 개발되었다[2][3][4]. Splash의 네 가지 디자인 목표는 다음과 같다: (1) 사용하기 쉬우면서 효율적인 programming abstraction을 제공한다 (2) 실시간 스트림 처리를 지원한다 (3) 실시간 시스템에서 필수적인 모드 변경, 센서 퓨전 등의 기능을 제공한다 (4) 멀티코어 분산 컴퓨팅 시스템에서의 성능 최적화를 지원한다. 이와 같은 디자인 목표를 가진 Splash는 그래픽 사용자 인터페이스를 이용한 프로그래밍 추상화를 제공할 뿐만 아니라 실시간 시스템에 필수적인 실시간 스트림 처리, 센서 퓨전, 모드 변경, 예외 처리 등을 지원한다[2][3]. 사용자는 Splash가 제공하는 schematic editor에서 그래픽 기반으로 알고리즘을 설계하고 코드 생성기를 통해서 생성된 skeleton code에 필요한 유저코드를 작성함으로써 쉽게 응용 프로그램을 개발할 수 있다.

Splash는 ROS 2에 이미 존재하는 기능 및 통신 메커니즘의 이점을 이용하기 위해서 ROS 2 기반으로 개발되었다[2]. 이러한 이유 때문에

Splash를 사용하여 작성된 응용 프로그램은 Splash 런타임 라이브러리와 ROS 2 런타임 라이브러리가 설치된 단일 혹은 분산 컴퓨팅 환경에서 구동된다. 그런데 분산 컴퓨팅 환경에서 Splash 응용 프로그램을 사용하기 위해선 각 머신의 환경에 맞게 응용 프로그램과 런타임을 빌드 및 설치해야 한다. 결과적으로 이는 소프트웨어 업데이트 및 유지보수를 하는데 많은 시간과 비용이 들도록 한다. 따라서 이런 불편함은 분산 컴퓨팅 머신에 확장성(scalability), 가용성(availability), 관리성(manageability)을 고려한 소프트웨어 빌드 및 배포의 필요성을 제기한다.

본 학위 논문에서는 효율적인 소프트웨어 배포를 위해서 도커 기반 Splash 빌드 유닛을 소개한다. 도커는 소프트웨어를 컨테이너라는 표준화 유닛으로 패키징 하여 빌드, 배포 및 관리할 수 있는 오픈소스 소프트웨어 플랫폼이다[5]. 빌드 유닛은 Splash 컴포넌트 집합으로 운영체제 프로세스에 매핑되는 개념이다. 해당 학위 논문에서는 빌드 유닛들을 소스 코드, Splash 런타임 라이브러리, ROS 2 런타임 라이브러리, 의존성 파일 등을 포함한 하나의 도커 이미지 파일로 생성되도록 한다. 그리고 개발자는 빌드 유닛 배포 툴을 이용해서 소프트웨어 배포 및 빌드 프로세스를 자동화하고 프로그램의 동시성 및 병렬성을 최적화할 수 있다. 본 논문에서 제안된 도커 기반 빌드 유닛을 Splash 프레임워크에 구현하였다. 또한, 분산 시스템에서 Splash 빌드 유닛을 이용하여 오픈소스 자동차 시뮬레이션 프로그램인 TORCS(The Open Racing Car Simulator), 딥러닝 기반 차선 인식 응용 프로그램, 딥러닝 모델 학습 응용 프로그램의 예제를 진행하여 제안된 빌드 유닛의 유용성을 확인하였다.

본 학위 논문의 나머지 구성 부분은 다음과 같다. 2장에서는 본 논문의 이해를 돕기 위해 ROS 2와 Splash의 배경지식에 대해서 설명한다. 3장에서는 도커 기반 빌드 유닛과 자동화 프로그램 배포에 대해서 소개한다. 4장에서는 딥러닝 기반 차선 인식 자율 주행 응용 프로그램 예제를 통해서 제안하는 빌드 유닛의 유용성을 보인다.

마지막으로 5장에서는 본 학위 논문을 요약하고 결론을 맺는다.

제 2 장 배경 지식

본 장에서는 본 연구의 이해를 돕기 위해 필요한 배경지식에 대해서 소개한다. 먼저 논문의 이해를 위하여 자율기기 소프트웨어 개발에 널리 사용되는 오픈소스 소프트웨어 프레임워크인 ROS 2를 설명한다. 이어서 ROS 2 기반으로 개발된 프로그래밍 프레임워크인 Splash를 설명한다.

제 1 절 ROS 2

ROS는 자율기기 소프트웨어 개발을 위한 소프트웨어 프레임워크이다[6]. 이름의 Robot Operating System처럼 운영체제가 아닌 운영체제를 포함한 미들웨어이다. 이 프레임워크는 개발자들이 하드웨어에 대한 걱정 없이 로봇을 포함한 자율기기 응용 프로그램을 개발할 수 있도록 라이브러리, 디바이스 드라이버, message passing layer, 프로그래밍을 위한 도구 등을 포함하는 추상화 계층을 제공한다. 이러한 이유 때문에 ROS는 개발의 자유도가 높고 다양한 런타임 라이브러리를 사용할 수 있다는 장점이 있다. ROS는 다양한 장점이 있음에도 불구하고 몇 가지 한계점이 존재한다. ROS는 자체 message passing layer를 사용하는데 이는 timing constraints를 명시할 수 있는 방법을 제공하지 않아서 실시간 시스템을 지원하지 못한다. 또한, 프로그램이 하나의 마스터 노드에 의해 관리되기 때문에 마스터 노드가 정상적으로 작동하지 않으면 전체 시스템에 문제가 생기는 안정성 문제도 있다. ROS에 존재하는 이러한 한계점을 해결하기 위해서 ROS 2가 등장하였다.

기존 ROS의 한계점을 극복하기 위해서 DDS(data distribution

service) 기반 ROS 2가 새롭게 제안되었다. DDS는 publish/subscribe 커뮤니케이션을 이용해서 실시간성(real-time), 안정성(reliability)과 확장성(scalability)를 보장해주는 통신 미들웨어의 표준이다[7]. 그림 1은 ROS와 ROS 2의 시스템 아키텍처이다[6]. 그림 1의 ROS와 달리 ROS 2는 자체 message passing layer를 사용하지 않고 DDS를 사용하며 DDS와 통신하기 위한 인터페이스가 존재한다. 본 절에서는 ROS 2의 computation 그래프 모델, 태스크의 콜백 함수 수행을 스케줄링하는 executor, 그리고 ROS 2의 한계점에 대해서 설명한다.

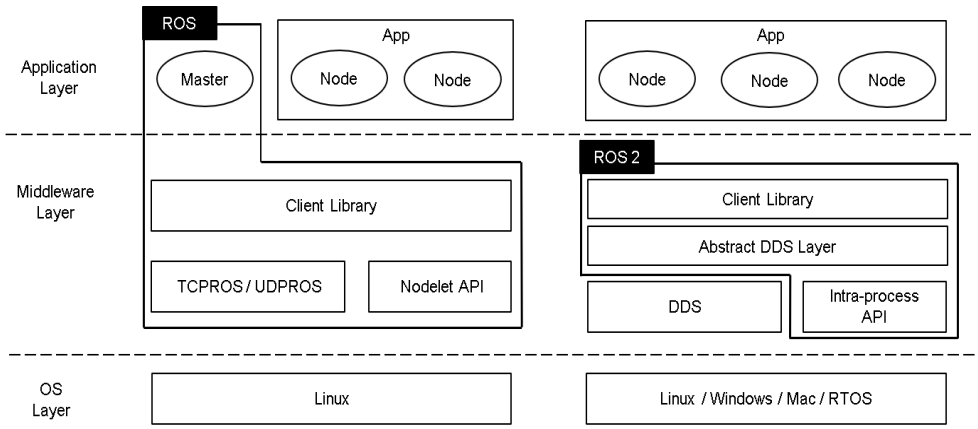

그림 1. ROS & ROS 2 시스템 아키텍처

1.1 ROS 2 Computation 그래프 모델

ROS 프로그램은 노드(node)와 노드 사이의 간선(edge)으로 이루어진 방향성 그래프의 모양을 지니고 있다[6][7]. ROS는 computation graph 컨셉을 기반으로 하고 노드, 메시지(message), 토픽(topic) 이라는 프로그래밍 컨셉이 존재한다. 그림 2는 노드와 간선으로 이루어진 ROS 프로그램의 예시이다[7].

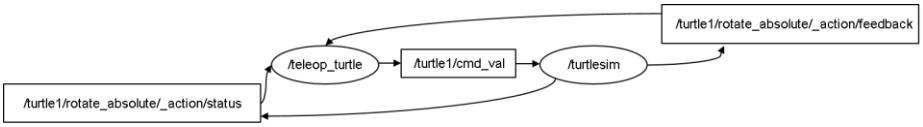

그림 2. ROS 프로그램 예시

노드는 ROS 프로그램을 구성하는 기본 수행의 단위이다. 그래프 상에서 타원형의 모습을 하고 있고 모든 노드는 각자 이름이 있으며 각 노드는 특정 기능을 수행한다. 메시지는 노드끼리 통신을 위해 전달하는 데이터의 방식이다. 노드는 (1) subscription, (2) client, (3) service 세 종류의 메시지를 노드끼리 송수신 할 수 있다. 노드는 메시지를 수신하면 메시지에 해당하는 콜백 함수를 수행한다.

토픽은 노드들이 메시지를 교환하는 버스이다. 노드가 토픽에 메시지를 publish하면 다른 노드가 해당 토픽을 subscribe하여 메시지를 수신할 수 있다. Publisher는 메시지를 publish하고 다른 일을 수행할 수 있고 subscriber는 다른 일을 수행하다 메시지가 토픽에 도착하면 subscribe 하여 메시지를 수신하면 되는 동기적인 통신 방법이다. 서비스는 request와 reply와 이루어진 client/server 통신과 같은 비동기적 통신 방법이다.

1.2 ROS 2 Executor

ROS 2의 태스크의 콜백 함수 스케줄링을 위해서 executor라는 개념이 도입되었다[6][8]. ROS 2에서는 multiple nodes가 운영체제 프로세스에 매핑되기 때문에 multiple nodes에 속한 콜백 함수들 수행의 스케줄링이 필요하다. Executor는 ROS의 client library인 rclcpp(C++) 또는 rclpy(Python)에 구현되어 있고 single threaded executor와 multi threaded executor를 제공한다. 또한 ROS에서 사용자가 executor의 scheduling policy를 custom 할 수 있으며 executor의 스케줄링 방식이 전체 시스템 타이밍에 중요한 영향을 끼친다.

ROS 2 executor 스케줄링 방식은 다음과 같다. Executor가 idle 하면 DDS layer에 있는 큐로부터 대기중인 콜백 함수의 정보를 가지고 와서 executor의 readySet이라는 자료구조를 업데이트 한다. 이때 DDS layer의 큐에 있는 콜백 함수의 종류로는 subscriber, service, client가 있다. 콜백 함수의 카테고리는 총 네 가지인데 (1) system

level timer에 의해서 trigger 되는 timer callback, (2) subscribed topic 메시지에 의해서 trigger 되는 subscriber 콜백, (3) 서비스 request에 의해서 trigger 되는 서비스 콜백, (4) client request에 의해 트리거 되는 클라이언트 콜백이 있다. 이들의 우선순위는 timer, subscriber, service, client 순으로 높다[6]. 따라서 Executor의 readySet에 있는 콜백 함수 중에서 가장 우선순위가 높은 콜백부터 실행이 된다. 콜백 함수의 실행이 끝나면 readySet에서 제외시켜주고 만약 readySet에 대기중인 콜백 함수가 없게 되면 readySet을 다시 업데이트 시켜준다. 그림 3은 executor의 스케줄링 방식을 보여준다[6].

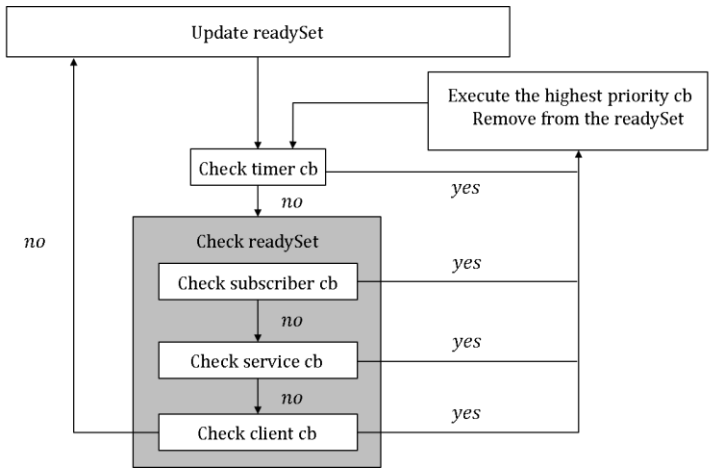

그림 3. Executor 스케줄링 방식

1.3 ROS 2의 한계점

자율주행차 및 로봇에 ROS 2 기반으로 개발된 소프트웨어가 탑재되어 활발히 연구가 진행되고 제품들이 상용화 되고 있다. ROS에 비해 많은 발전이 있음에도 불구하고 ROS 2에도 다음과 같은 한계점이 존재한다[2].

(1) ROS 2는 완전한 실시간 시스템 환경을 지원하지 못 한다. ROS 2는 DDS라는 통신 표준을 이용하고 DDS는 QoS를 통해서

데드라인을 명시할 수 있다. 하지만 이러한 데드라인은 통신 레벨에서만 존재하기 때문에 end-to-end time constraint를 명시할 수 없어서 완전한 실시간 시스템 환경을 보장해줄 수 없다.

(2) ROS 2는 개발자에게 프로그래밍 관련 편의성 제공이 부족하다. ROS 2의 경우 그래픽 프로그래밍을 지원하지 않아서 프로그래밍에 익숙하지 않은 엔지니어의 경우에는 ROS 2가 사용하기 쉽지 않다. ROS 2의 computation graph의 경우에는 개발 과정에서 확인할 수 있는 것이 아닌 runtime 과정에서 확인할 수 있기 때문에 개발 과정에서는 그래프 사용이 불가능하다. 또한, 자율기기 소프트웨어 개발에 필수적인 센서 퓨전이나 모드 변경과 같은 semantic을 지원하지 않는다.

(3) 자율기기에 많이 탑재되는 딥러닝 응용 프로그램의 경우 분산 시스템에서의 구동이 필수적인데, ROS 2는 사용자가 원격으로 응용 프로그램의 설치, 시작, 중지 및 모니터링 할 수 있는 자동화된 배포를 지원하지 않는다. 따라서 사용자는 분산된 컴퓨터에 수동으로 응용 프로그램을 배포해야 하고 수행 중인 프로그램의 상태를 모니터링 하기 힘들다.

제 2 절 Splash

Splash는 자율 기기의 소프트웨어 개발을 위해 개발된 그래픽 기반 프로그래밍 프레임워크이다[2][3][4]. 본 연구진이 기개발한 Splash는 통신 미들웨어 DDS와 Linux 커널 기반이었지만 ROS 2에 이미 존재하는 많은 기능과 이점을 사용하기 위해 Splash를 ROS 2 기반으로 재설계하였다. 그림 4는 ROS 2 기반 Splash의 layered architecture이다[2]. 본 절에서 ROS 2 기반 Splash에 대해서

소개하고 Splash와 ROS 2간의 인터페이스 구축에 대해서 설명한다.

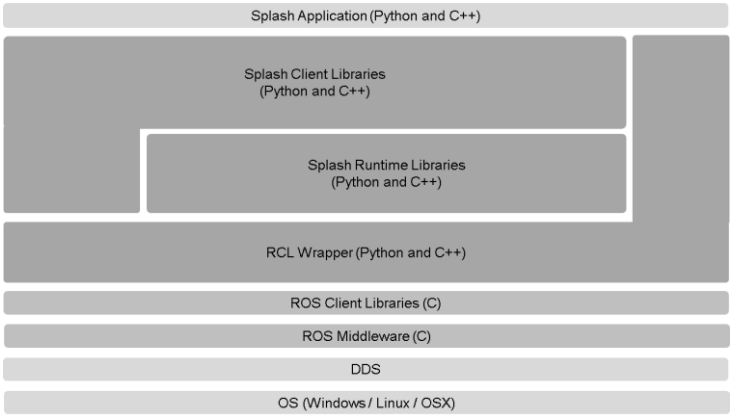

그림 4. Splash layered architecture

개발자는 그림 5의 Splash의 웹 기반 schematic editor를 이용해서 프로그램을 개발한다. Schematic editor에서 전체적인 프로그램의 알고리즘을 그래픽 기반으로 작성하고 code generator에 의해 자동 생성되는 skeleton code에 유저 소스 코드를 구현하여 프로그램 구현을 완성한다. Splash 프레임워크가 데이터 송수신, 데이터 센서 퓨전, 모드 변경에 대한 코드를 자동 생성 해주기 때문에 개발자는 세부 알고리즘 개발에만 집중할 수 있다. 프로그램의 원활한 개발을 위해 Splash는 Splash client library, Splash runtime library를 제공한다[3]. 이 절의 나머지 항목은 Splash의 구성요소 각각에 대해서 설명한다.

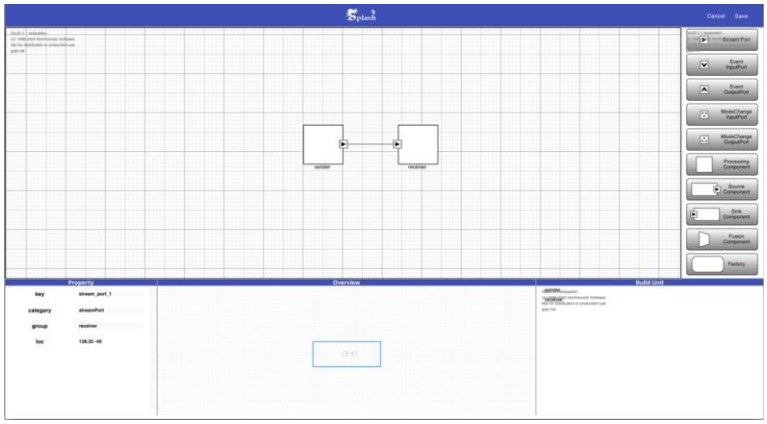

그림 5. Splash schematic editor

2.1 Language construct

Splash 프로그램 또한 ROS 2의 프로그램처럼 노드(node)와 간선(edge)으로 이루어진 방향성 그래프의 형태를 가지고 있다. 그림 6은 Splash로 작성한 LFA(lane following assist) 응용 프로그램의 예시이다. 개발자는 Splash schematic editor에서 language construct를 이용해서 응용을 작성한다. Splash의 language construct는 컴포넌트(component), 포트(port), 채널(channel), 씨링크(clink)로 구분된다[2].

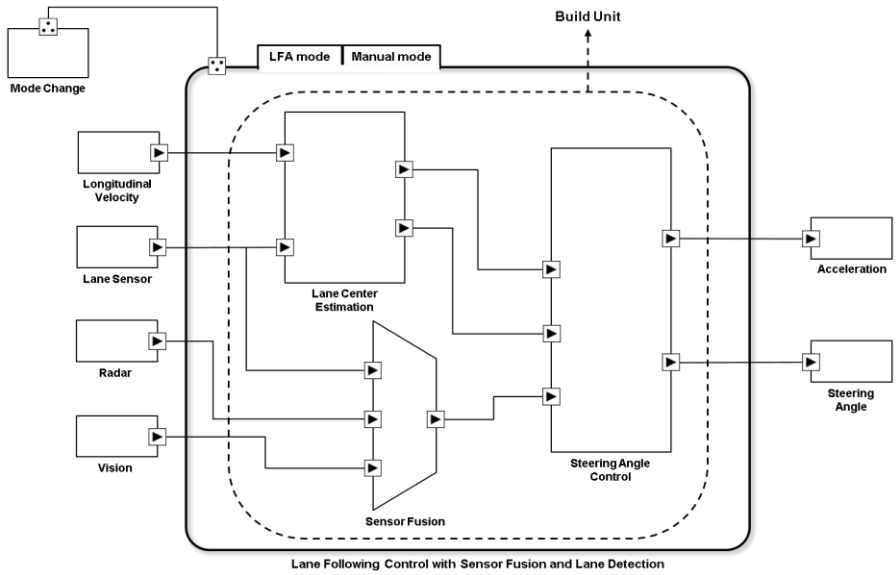

그림 6. Splash로 작성한 LFA 응용의 예시

Splash에서 기본 수행의 단위를 컴포넌트로 정의한다. 방향성 그래프에서 노드에 해당하는 것이 Splash에서는 컴포넌트이다. Splash의 컴포넌트는 ROS 2의 노드로 매핑된다. 컴포넌트는 atomic 컴포넌트와 composite 컴포넌트로 분류된다. Atomic 컴포넌트는 기능에 따라서 1. 소스 컴포넌트(source component), 2. 싱크 컴포넌트(sink component), 3. 프로세싱 컴포넌트(processing component), 4. 퓨전 오퍼레이터(fusion operator)로 분류된다. 소스

컴포넌트는 센서로부터 데이터를 받아온다. 싱크 컴포넌트는 연산이 끝난 데이터를 받아서 자율기기의 액추에이터(actuator)에게 전달한다. 프로세싱 컴포넌트는 데이터를 입력으로 받아서 사용자가 정의한 연산을 수행하는 가장 기본적인 단위이다[2][3]. 퓨전 오퍼레이터는 두개 이상의 데이터 입력을 받아 사용자가 정의한 퓨전 규칙에 따라 데이터의 튜플을 출력하는 컴포넌트이다. Splash에서는 팩토리(factory)가 composite 컴포넌트에 해당된다. 팩토리는 하나 이상의 컴포넌트를 포함한다. 그리고 사용자가 팩토리에 모드를 명시할 수 있어서 하나 이상의 모드를 가질 수 있다.

각 컴포넌트는 데이터, 이벤트, 모드 변경 신호의 입출력을 위해서 포트를 가질 수 있고 포트의 종류는 1. 스트림 입출력 포트(stream input/output port), 2. 이벤트 입출력 포트(event input/output port), 3. 모드 변경 입출력 포트(mode change input/output port)가 있다[2]. Splash의 출력 포트는 ROS 2의 publisher, Splash의 입력 포트는 ROS 2의 subscriber와 매핑된다. 소스 컴포넌트는 출력 포트만 가질 수 있고 싱크 컴포넌트는 입력 포트만 가질 수 있는 특수성을 가지고 있다. 컴포넌트는 스트림 입출력 포트를 이용해서 데이터의 송수신을, 이벤트 입출력 포트를 이용해서 이벤트의 송수신을, 모드 변경 입출력 포트를 이용해서 모드 변경 신호를 송수신 하여 응용의 필요한 작업을 수행한다. 포트는 큐를 가지고 있어서 데이터, 이벤트 및 모드 변경 신호를 송수신 하기 전에 큐에 메시지를 보관한다. 스트림 입력 포트에서는 데이터에 명시되어 있는 birthmark를 확인하여 순차적으로 큐에 데이터를 보관한다. 또한, 스트림 출력 포트에서는 rate control을 명시하여 특정 주기로 데이터를 송신하는 환경을 설정할 수 있다[2].

채널은 방향성 그래프의 간선에 해당하는 Splash의 개념이다. ROS 2의 토픽이 채널과 매핑된다. 채널은 스트림 입력 포트와 스트림 출력 포트를 연결하는 실선에 해당하고, 스트림 데이터를 전달하는 경로이다. 채널은 단일 fan-in 구조로 제한하지만 다수의 fan-out을 허용한다.

스트림 데이터를 전달하는 경로가 채널이라면, 이벤트와 모드 변경

신호를 전달하는 경로는 씨링크이다[2]. 씨링크 또한 ROS 2의 토픽과 매핑된다. 씨링크의 경우 점선으로 표현되며 모드 변경 입력 포트와 출력 포트 사이를 이어주거나 이벤트 입력 포트와 이벤트 출력 포트 사이를 이어준다. 채널과 달리 씨링크는 fan-in 및 fan-out을 지원한다.

2.2 Splash client library

Splash client library는 파이썬으로 작성된 객체 지향 프로그래밍 API를 제공함으로써 개발자가 Splash 개발에 필요한 프로그래밍 추상화를 지원한다. Splash가 ROS 2 기반으로 개발되었기 때문에 Splash client library가 제공하는 API는 ROS client library를 사용할 수 있게 하는 인터페이스를 제공한다. Splash 개발자는 schematic editor에서 language constructs를 이용하여 프로그램을 설계한다. 설계된 프로그램은 코드 자동 생성기에 의해서 skeleton code가 자동으로 생성되고 개발자는 필요한 유저 소스 코드를 작성하여 프로그램 개발을 완성한다. Splash client library에는 language construct의 클래스와 메소드 함수가 구현되어 있다. 따라서 이러한 코드 생성 과정에서의 Splash language construct 오브젝트 생성과정에 Splash client library가 필수적으로 사용된다.

사용자가 schematic editor에서 설계한 Splash graph의 language constructs 코드 생성 과정에서 Splash client library에 구현된 language construct들의 클래스와 메소드를 사용한다. 따라서 Splash client library에는 Splash의 language construct에 포함되는 빌드 유닛, 컴포넌트, 포트의 클래스와 메소드가 구현되어 있다. 추가적으로 콜백 함수의 수행을 스케줄링하는 executor와 예외상황 처리를 위한 exception이 구현되어 있다. 따라서 사용자는 처리하고 싶은 예외 상황을 정의하거나 처리 방법에 대해 구현할 수 있다.

Splash의 컴포넌트 클래스는 ROS 2의 노드 클래스의 서브 클래스이다. ROS 2 node class에는 publisher, subscriber와 콜백 함수들을 관리하는 메소드를 가지고 있다. 따라서, Splash의 컴포넌트

클래스는 ROS 2의 노드 클래스를 상속받음으로써 이의 메소드들을 사용할 수 있다. 또한, Splash 컴포넌트 클래스는 이름, 자신이 속해 있는 팩토리, 자신에 부착된 포트 정보, 콜백 함수 정보 등을 가지고 있다. Splash의 컴포넌트 종류는 processing component, source component, sink component, fusion operator로 총 네 가지 이므로, 네 가지에 해당하는 클래스 또한 구현되어 있고 이들은 컴포넌트 클래스를 상속받는다. 각각의 클래스는 자신의 오브젝트에 부착된 포트의 정보를 관리하기 위해서 dictionary 자료 구조를 이용한다. 예를 들어, 소스 컴포넌트의 경우 스트림 output port만 가질 수 있기 때문에 스트림 output port의 정보를 보관하는 dictionary에 key로 채널 이름을, value로 stream output port의 오브젝트를 저장한다. ROS 2의 노드 클래스를 상속받아 생성된 컴포넌트 오브젝트는 자신만의 메소드를 가질 수가 있다. 예를 들어, 퓨전 오퍼레이터 오브젝트는 fusion rule을 설정하는 메소드 함수인 `set_fusion_rule`을 포함한다.

컴포넌트에 부착되어 스트림 데이터, 이벤트의 입출력을 담당하는 포트는 input과 output으로 구분되어 구현된다. Stream input port는 자신과 연결된 채널의 정보, 메시지를 담을 리스트, 그리고 부착된 컴포넌트의 정보를 가진다. 또한, ROS 2의 subscriber 클래스의 메소드인 `create_subscription`을 통해 채널로부터 메시지를 받아 메시지를 담는 리스트에 보관한다. Stream output port의 경우 stream input port 클래스와 마찬가지로 컴포넌트의 정보, 채널의 정보를 가지지만 output port는 데이터를 수신하는 역할을 수행하기 때문에 ROS 2의 `create_publisher` 메소드를 이용한다. 또한, output port는 데이터의 출력 rate를 control하는 rate constraints를 설정하는 메소드를 가진다. Event input port 클래스의 경우 stream input port 클래스의 구현과 동일하다. Event output port 클래스의 경우 또한 stream output port 클래스의 구현과 유사하지만 이벤트의 경우 스트림 데이터와 달리 rate control을 지원하지 않기 때문에 해당 메소드가 존재하지 않는다. 모드 변경 클래스의 경우 자신이 부착된 컴포넌트의

정보, ROS 2의 `create_publisher` 메소드를 이용하여 publisher를 가진다.

Splash executor의 경우 ROS 2의 executor 클래스를 상속받는다. ROS 2의 executor와 다른 점이 있다면 Splash executor에는 `get_active_nodes`라는 메소드가 추가되었다. 기존 executor는 모든 node의 콜백 함수에 대해서 스케줄링을 하고 실행한다. 하지만, Splash에서는 모드 변경을 지원하기 때문에 현재 실행중인 모드에 대한 노드의 콜백 함수에 대해서만 고려가 필요하다. 따라서, 현재 실행중인 노드를 파악하기 위하여 해당 함수가 추가되었다.

2.3 Splash runtime library

Splash runtime library는 응용 프로그램에 필수적인 런타임 기능을 제공한다. 이 라이브러리에는 콜백 함수 실행(callback execution), rate 제어(rate control), 센서 퓨전(sensor fusion), 모드 변경(mode change) 와 같은 기능들이 파이썬으로 구현되어 있다.

Rate 제어는 단위 시간당 출력 되는 데이터의 수를 제어하는 메커니즘이다[3][4]. Outputport에 데이터가 있다고 해서 바로 ros 2의 publish 메소드를 호출하는 것이 아니라, 해당 rate 제어 메커니즘을 따른다. 이를 통해서 데이터 출력의 일관성을 보장 해 줄 수 있고 데이터 전송 과정에서 발생할 수 있는 bursty data traffic 등을 해결한다. 해당 메커니즘의 원리는 스트림 output port에 존재하는 큐에 rate를 설정할 수 있다. 설정한 rate에 따라 일정 주기마다 큐를 확인하여 birthmark가 가장 이른 데이터를 찾는다. 그러므로 가장 먼저 큐에 진입한 데이터를 출력할 수 있게 된다. 내부적으로는 큐에 설정된 rate에 맞춰서 ROS 2의 publish method를 호출한다.

센서 퓨전은 여러 개의 센서를 사용하는 경우 두개 이상의 센서 데이터를 입력 받아 사용자가 정의한 퓨전 연산을 수행하고 하나의 스트림 데이터로 출력하는 메커니즘이다[3][4]. 센서 퓨전은 자율주행차를 포함한 자율기기에 탑재되는 다양한 센서들의 데이터를

효과적으로 처리하기 위해 제안되었다. 그림 7처럼 센서 퓨전을 위해서 퓨전 오퍼레이터가 존재하고 퓨전 오프레이터에는 두개 이상의 stream input port가 부착된다. 또한, 하나의 stream output port가 부착되는데 이는 사용자가 정의한 연산을 하고 데이터의 튜플을 출력시킨다. 개발자는 입력된 스트림 데이터들 간의 correlation constraint을 명시하여 정해진 시간 내의 입력 스트림 데이터만의 조합으로 튜플을 만든다. Input data가 stream input port에 도착하여 port가 퓨전 오퍼레이터에 데이터를 publish하면 센서 퓨전 메소드가 각각의 데이터를 상응하는 큐로 옮기고 사용자가 정의한 퓨전 규칙에 따라서 튜플을 생성하여 output port에 전송한다. 만약 만족하는 퓨전 규칙이 없으면 사용자가 정의한 예외 처리 메소드가 불려진다.

모드 변경은 두개 이상의 모드를 가진 팩토리의 모드를 변경해주는 메커니즘이다[3][4]. 팩토리에 속해 있는 컴포넌트는 하나의 빌드 유닛에 속해 있기 때문에 구현상으로는 빌드 유닛의 모드를 변경해주는 메커니즘이다. 그림 8은 모드 변경 입출력 포트의 예시이다[9]. Splash runtime library는 모드 변경을 위해 mode manager를 제공한다. 해당 mode manage는 빌드 유닛당 하나씩 존재하여 빌드 유닛 단위로 모드를 관리하여 준다. Mode manager는 모드 변경 신호를 ROS 2 topic에 publish 한다. 모드 변경 input port에 팩토리의 이름과 이벤트 이름이 publish 되면 해당 빌드 유닛에 속하여 있는 컴포넌트들은 해당 메시지를 수신하여 모드에 맞게 컴포넌트를 활성화하거나 비활성화한다.

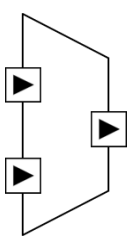

그림 7. 퓨전 오퍼레이터

그림 8. 모드 변경 입출력 포트

제 3 장 도커 기반 빌드 유닛

본 장에서는 분산 시스템에 확장성, 가용성 및 관리성을 고려한 도커 기반 Splash 빌드 유닛과 배포 방법을 소개한다. 3.1절에서는 빌드 유닛의 정의와 역할에 대해서 설명하고 3.2절에서 도커 기반 빌드 유닛 및 구현에 대해서 소개한다. 마지막으로 3.3절에서 빌드 유닛 배포 툴을 이용한 빌드 유닛 배포 프로세스에 대해서 설명한다.

제 1 절 빌드 유닛

빌드 유닛은 하나 이상의 컴포넌트의 집합이고 운영체제 프로세스에 매핑 되는 단위이다. 빌드 유닛의 등장 배경은 Splash 응용 프로그램의 런타임 오버헤드를 줄이기 위해서이다[9]. Splash 응용 프로그램의 런타임 오버헤드를 줄이기 위해서 2가지 오버헤드를 고려하여야 한다: 1. 응용 프로그램의 컴포넌트를 수행하는 프로세스 간의 문맥 교환 오버헤드, 2. 하나의 프로세스에 속한 컴포넌트들끼리 데이터 교환을 위해 통신을 할 때의 통신 오버헤드. 하나의 시스템에서 수행되는 컴포넌트의 개수가 많아지게 되면 이에 따른 프로세스의 개수도 증가하기 때문에 문맥 교환 오버헤드가 증가한다. 또한, 프로세스끼리 통신을 할 때 통신 미들웨어를 거치기 때문에 통신 오버헤드가 증가한다. 따라서 하나 이상의 컴포넌트를 빌드 유닛이라는 단위로 매핑하여 프로세스에 할당하면 문맥 교환 오버헤드를 감소시킬 수 있고 하나의 프로세스 내에서 데이터 교환을 할 때에 통신 미들웨어를 거치지 않기 때문에 통신 오버헤드도 감소시킬 수 있다[9].

Splash schematic editor에서 하나 이상의 컴포넌트를 선택하여 빌드 유닛으로 설정할 수 있다. 그림 9처럼 하나의 팩토리에 속한 컴포넌트들은 하나의 빌드 유닛에 포함된다. 또한, 사용자가 별도로

빌드 유닛을 설정하지 않은 컴포넌트는 기본으로 제공되는 빌드 유닛에 자동으로 추가된다.

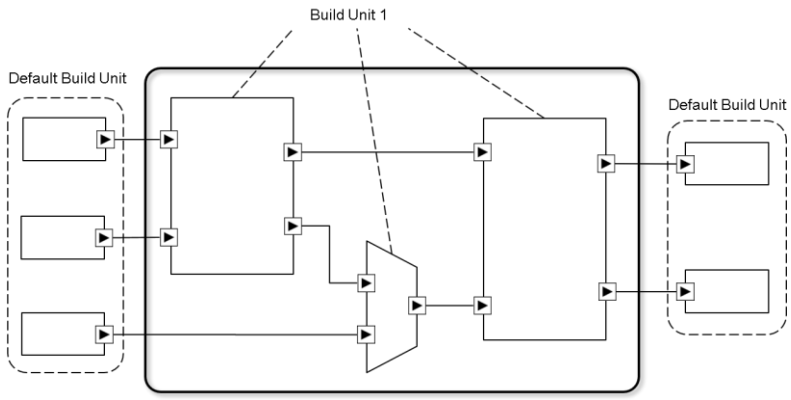

그림 9. 빌드 유닛 매핑 예시

빌드 유닛 하나당 모드 변경 매니저와 executor가 존재한다. 따라서, 빌드 유닛 단위로 모드 변경과 콜백 함수 수행의 스케줄링을 한다. 그림 10의 모드 변경 handler pseudo code에서 보이는 거처럼 빌드 유닛의 모드 변경 매니저는 모드 변경 신호를 받으면 해당 빌드 유닛에 속한 컴포넌트들의 모드를 변경시켜주는 작업을 수행한다. 또한, 빌드 유닛의 executor는 해당 빌드 유닛에 속한 컴포넌트의 콜백 함수들 수행의 스케줄링을 담당한다. 이러한 작업을 위해서 Splash client libraries에 빌드 유닛 API가 포함되어 있고 Splash runtime libraries에 모드 변경 및 callback execution을 수행하는 API가 포함되어 있다. 따라서, 그림 11에서 보이는 거처럼 빌드 유닛은 빌드 유닛 API를 사용하고, 빌드 유닛 API는 모드 변경 및 callback execution API를 사용한다.

```
def modechange_handler(self, factory, next_mode):
        …
    for mode in mode_list of factory:
        if mode name == next_mode:
            for component in mode:
                activate components
        else:
            for component in mode:
                deactivate components
        …
```

그림 10. 모드 변경 handler pseudo code

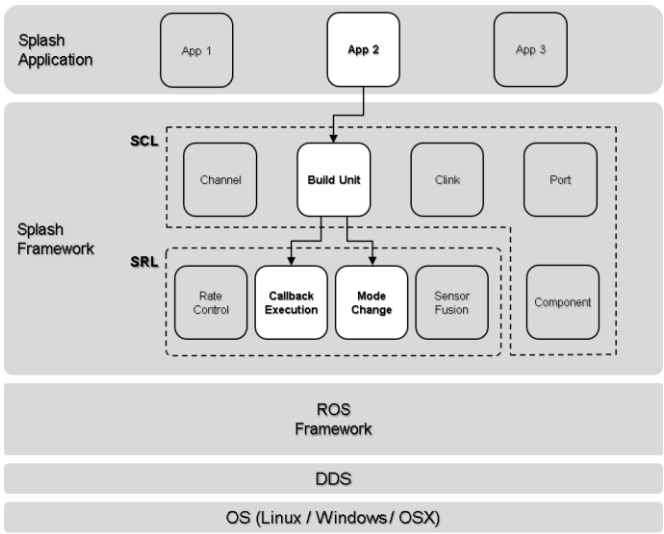

그림 11. Splash 응용 프로그램 빌드 유닛 API 사용 예시

제 2 절 도커 기반 빌드 유닛 및 구현

도커는 응용 프로그램의 개발, 배포, 및 실행을 자동화하는 오픈소스 플랫폼이다[5]. 도커 이미지는 여러 개의 계층으로 구성되어 있고 바이너리 파일의 형식을 가지고 있다. 도커 이미지는 응용 프로그램 실행에 필요한 소스 코드, 라이브러리, 런타임 등을 컨테이너화 하여 추가적으로 의존성 파일 등을 컴파일 할 필요가 없다. 도커 이미지를 실행하여 해당 이미지에 포함된 파일로 구성된 파일 시스템, 네트워크, 및 다양한 시스템 자원을 사용할 수 있는 공간을 도커 컨테이너라고 한다. 도커 이미지 파일은 도커 이미지 파일 저장소인 도커 registry에 저장되어 공유된다. 따라서 분산된 머신에서 도커 registry에 있는 도커 이미지 파일을 배포 받아 실행하여 사용할 수 있다. 그러므로 도커를 활용하여 프로그램을 신속하게 빌드 및 배포하면 프로그램 작성 및 분산 시스템 구축에서 발생할 수 있는 지연을 크게 줄일 수 있다.

빌드 유닛 클래스는 Splash client library에 구현이 되어 있다. 빌드 유닛 클래스는 해당 빌드 유닛에 매핑되는 컴포넌트를 담기 위해 컴포넌트 이름을 key, 컴포넌트 오브젝트를 value로 가지는 dictionary 자료구조를 이용한다. 그림 12는 빌드 유닛 클래스 소스 코드의 일부분으로 빌드 유닛 오브젝트 초기화와 add 메소드이다. 해당 클래스는 오브젝트 초기화 과정에서 빌드 유닛에 속하는 컴포넌트의 정보를 보관하기 위해서 dictionary 자료구조를 생성한다. 또한, add 메소드를 이용해서 해당 빌드 유닛에 속하는 컴포넌트를 추가할 수 있다.

```
import rclpy
from .executor import SplashExecutor
from srl.mode_change import ModeManager

class BuildUnit:
    def __init__(self, name):
         …
        self.context = rclpy.init()
        self._components = {}
        self._executor = SplashExecutor()
        self._mode_manager = ModeManager(self.context)
         …

    def add(self, component):
        self._components[component.name] = component

     …
```

그림 12. 빌드 유닛 클래스 코드 예시

사용자가 schematic editor에서 Splash 응용 프로그램을 작성하고 컴포넌트들의 빌드 유닛 매핑을 마친 후 코드 생성을 하면, schematic editor에서 생성된 JSON 타입 데이터를 입력으로 하는 코드 생성기를 통해 skeleton code가 생성된다. 코드 생성기를 통해 생성된 skeleton code에 소스 코드를 작성함으로써 사용자가 필요한 알고리즘의 구현을 한다. 이어서 코드를 빌드하면 빌드 유닛 이미지가 생성된다. Splash 빌드 유닛 이미지는 도커의 컨테이너화 기술을 이용하여 도커 이미지로 매핑된다. 빌드 유닛 이미지는 그림 13와 같이 Ubuntu 이미지를 기반으로 Splash layer, ROS layer, 응용 프로그램에 필요한 external library layer, 유저 소스 코드 layer로 구성되어 있다. OSRF(Open

Source Robotics Foundation)에서 공식으로 배포하는 ROS layer는 공식 Ubuntu 이미지와 Debian 패키지 기반으로 개발되어 ROS 2의 런타임 라이브러리 제공을 통해 각종 기능을 지원한다[10][11]. Splash layer는 Splash client library와 Splash runtime library로 구성되어 있다. 따라서 빌드 유닛 이미지는 Splash 런타임 라이브러리와 ROS 2 런타임 라이브러리를 포함하고 있기 때문에 어떤 머신에 배포되던지 간에 해당 런타임을 사용할 수 있다. ROS layer와 Splash layer는 모든 빌드 유닛 이미지가 동일하게 보유하고 있지만 응용 프로그램 layer의 경우 응용 프로그램에 따라 구성이 달라진다.

빌드 유닛 이미지가 만들어지는 과정은 다음과 같다. 도커 registry에서 공식 ROS 2 이미지를 pull한다. 해당 이미지로 컨테이너를 생성하고 해당 컨테이너에 Splash client library와 Splash runtime library를 구축한 뒤 이미지를 생성한다. ROS 2 layer와 Splash layer는 모든 빌드 유닛 이미지가 공통으로 가지고 있기 때문에 해당 layer로 구성된 이미지 파일을 미리 생성하여 Splash 서버에 보관한다. 사용자가 Splash 응용을 작성하고 빌드 할 때 Splash layer와 ROS 2 layer로 구성된 이미지 파일을 컨테이너화 하여 external library와 유저 소스 코드를 추가한 뒤 최종 빌드 유닛 이미지로 생성한다.

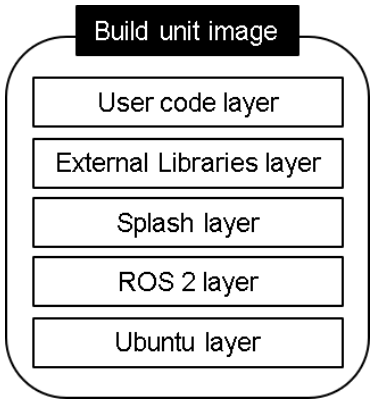

그림 13. 빌드 유닛 이미지 구성

제 3 절 빌드 유닛 배포

Splash schematic editor에서 사용자가 설정한 빌드 유닛에 따라 빌드 유닛 저장소에 빌드 유닛 이미지가 생성된다. 그림 14는 빌드 유닛의 배포 프로세스를 나타낸다.

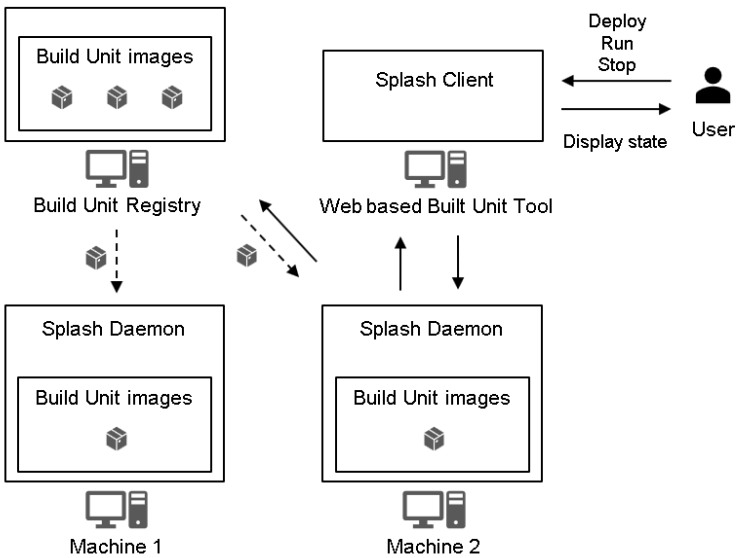

그림 14. 빌드 유닛 배포 프로세스

Splash editor에서 사용자가 하나 이상의 컴포넌트에 빌드 유닛을 설정하여 빌드하면 빌드 유닛 저장소에 ROS layer, Splash layer, 응용 layer를 가진 빌드 유닛 이미지가 생성된다. 그리고 웹 기반 빌드 유닛 빌드 및 배포 툴에서 배포할 빌드 유닛 이미지와 머신을 매칭 시켜줄 수 있다. 그림 15에서 보이는 것처럼 웹 기반 빌드 유닛 빌드 및 배포 툴에서 사용자가 드롭 다운(drop-down) 방식으로 빌드 유닛 이미지를 배포할 머신과 빌드 유닛 이미지를 매핑하여 배포한다.

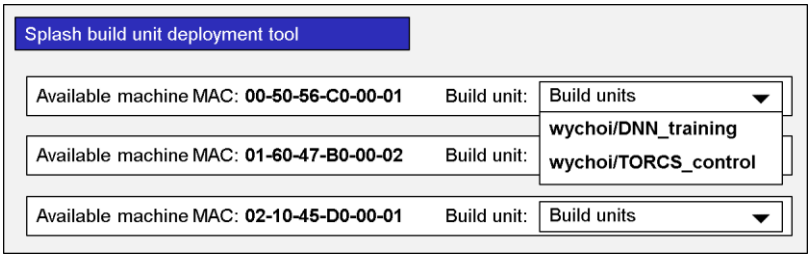

그림 15. 빌드 유닛 이미지와 머신 매핑 예시

웹 기반 빌드 유닛 빌드 및 배포 툴에는 Splash client라는 프로세스가 존재한다. Splash client는 배포 툴 단에서 빌드 유닛의 배포를 관리한다. 분산 시스템 머신에는 Splash daemon이라는 프로세스가 존재하는데 해당 프로세스는 분산된 머신 단에서 빌드 유닛의 실행을 관리한다. Splash client은 Splash daemon에게 request를 전송하고 Splash daemon은 response을 전송함으로써 빌드 유닛 저장소로부터 빌드 유닛 이미지를 배포 받고, 실행하고, 모니터링하는 작업을 수행한다. 그러므로 사용자는 분산된 머신에 Splash daemon만 실행시켜 주면 빌드 유닛 툴에서 빌드 유닛 이미지의 배포, 빌드, 실행, 중지 등 원격으로 빌드 유닛의 수행을 관리할 수 있다. 또한, 실행중인 빌드 유닛 컨테이너 내부의 Splash 프로세스에서 사용자가 예외처리를 구현하지 않은 예외 상황이 발생하면, Splash daemon을 통해 Splash client에 해당 예외 상황이 전달되어 사용자가 확인할 수 있다.

제 4 장 Case Study

본 장에서는 본 학위 논문에서 제안하는 Splash 빌드 유닛의 유용성을 검증하기 위해서 TORCS(the open racing car simulator)[12]와 딥러닝 기반 차선 인식 알고리즘[13]을 이용하여 진행한 예제에 대해서 소개한다. 4.1절에서는 예제의 하드웨어 및

소프트웨어 환경에 대해서 설명하고 4.2절에서는 예제 구성에 대해서 설명한다. 마지막으로 4.3절에서는 예제 결과 및 평가를 통해 빌드 유닛의 유용성에 대해서 설명한다.

제 1 절 예제 환경

자율주행 데모의 구축을 위해서 TORCS(the open racing car simulator)라는 오픈소스 3D 자동차 주행 시뮬레이터를 사용하였다[12]. 해당 소프트웨어의 소스 코드를 수정하여 사용자는 가상 환경에서 다양한 자율 주행 관련 실험을 진행할 수 있다. 예제에서 딥러닝 기반 차선 인식 알고리즘 또한 사용한다[13]. 해당 딥러닝 모델은 자동차 주행 전방 이미지 프레임을 데이터로 차선 학습 및 예측을 한다. 예제에 사용한 하드웨어 및 소프트웨어 사양은 표 1과 같다.

표 1. Case Study 환경

Hardware		
Computer 1 (DNN Training)	CPU	AMD Ryzen 9 5900X 12 core 3.7GHz
	GPU	Geforce rtx 3080
	Memory	64 GB
Computer 2 (TORCS)	CPU	Intel Core i7-1165G7 2.80GHz
	Memory	16 GB
Software		
DDS	eProsima Fast RTPS v2.3.0	
OS	Ubuntu 18.04	

제 2 절 예제 구성

그림 16은 예제의 전체 구성이다. 하나의 머신에 TORCS 프로그램과 TORCS의 자동차 제어 값 연산을 위한 빌드 유닛이 수행된다. 해당 빌드 유닛은 TORCS의 정면 주행 이미지와 각종 센서 값을 입력으로 받아 차량이 차선의 중앙으로 주행할 수 있도록 제어 값을 연산한다. 다른 하나의 머신에서는 자동차 주행 전방 이미지를 수신 받아 차선 인식 딥러닝 모델을 학습하는 빌드 유닛이 수행된다. Splash schematic editor에서 응용을 작성하고 빌드 유닛을 설정하였다. 해당 case study는 ROS layer, splash layer에 자동차 제어 값 연산 응용으로 이루어진 빌드 유닛 이미지와 딥러닝 모델 학습 응용 layer를 가진 빌드 유닛 이미지 두 개가 생성되고 각 머신에 배포되어 수행된다.

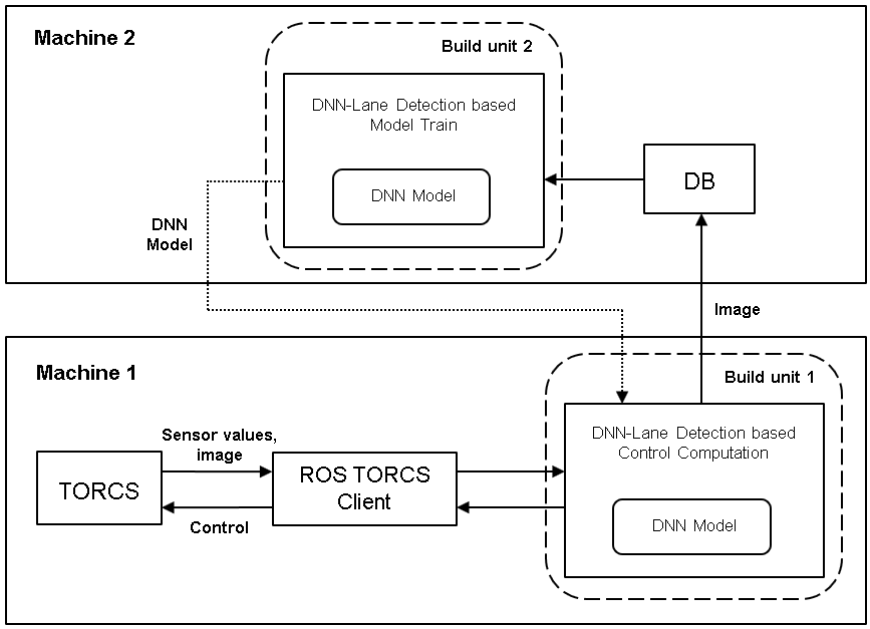

그림 16. 예제 전체 구성

TORCS 프로그램은 차량의 센서 값 및 주행 이미지를 UDP 통신을 통해서 ROS TORCS client에 전송한다. ROS TORCS client는 ROS

middleware를 통해 Splash 응용과 통신하여 센서 값 및 주행 이미지의 데이터를 Splash 응용에 전송한다. 그림 17은 schematic editor로 작성한 예제의 Splash 응용이다. 하나의 Splash 응용은 DNN 기반 차선 인식 모델을 사용하여 수신 받은 이미지 데이터와 차량의 조향각, 속도의 센서 값을 이용하여 차량이 차선의 중앙으로 주행할 수 있도록 차량의 제어 값을 연산한다. 차량의 제어 값을 연산하는 Splash 응용은 제어 연산 결과를 ROS TORCS client와의 통신을 통해 TORCS로 전송하여 차량을 제어한다. 또한, 해당 빌드 유닛은 DNN 기반 차선 인식 모델의 학습을 위해 TORCS로부터 받은 이미지 파일을 다른 분산 머신에서 수행중인 빌드 유닛으로 전송한다. 이미지 파일을 수신 받은 빌드 유닛은 해당 차량 전방 주행 이미지로 딥러닝 차선 인식 모델을 학습하여 모델을 업데이트 한다.

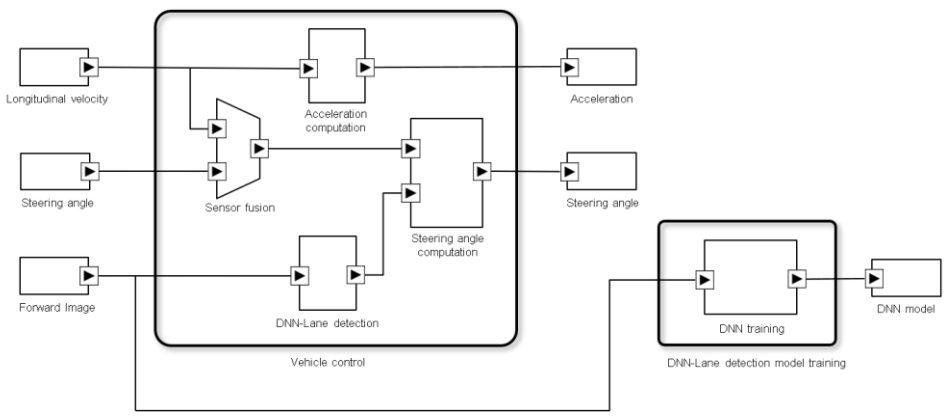

그림 17. 예제의 Splash 응용 구성

제 3 절 예제 결과 및 평가

그림 18은 딥러닝 기반 차선 인식 모델을 이용하여 TORCS 프로그램의 차선을 인식한 결과이다. 예제에서는 이와 같은 차선 인식 결과와 차량의 longitudinal velocity와 steering angle을 포함한 센서

값을 이용해서 차량이 차선의 중앙으로 주행할 수 있도록 제어 값을 연산 한다.

Splash 런타임, ROS 2 런타임과 응용 프로그램을 하나의 도커 기반 빌드 유닛 이미지 파일로 생성하여 이의 배포를 통해 복잡한 예제의 구성을 간편하게 분산 시스템에 구축할 수 있었다. 복잡한 응용 프로그램 일수록 사용하는 라이브러리 및 의존성 파일이 많아서 분산 시스템을 구축할 때 해당 소프트웨어를 일일이 배포 및 빌드해야 하는 번거로운 작업이 많다. Splash의 빌드 유닛을 이용하면 각종 소프트웨어를 하나의 이미지 파일로 만들고 배포 tool을 이용하여 간편하게 배포할 수 있다.

.

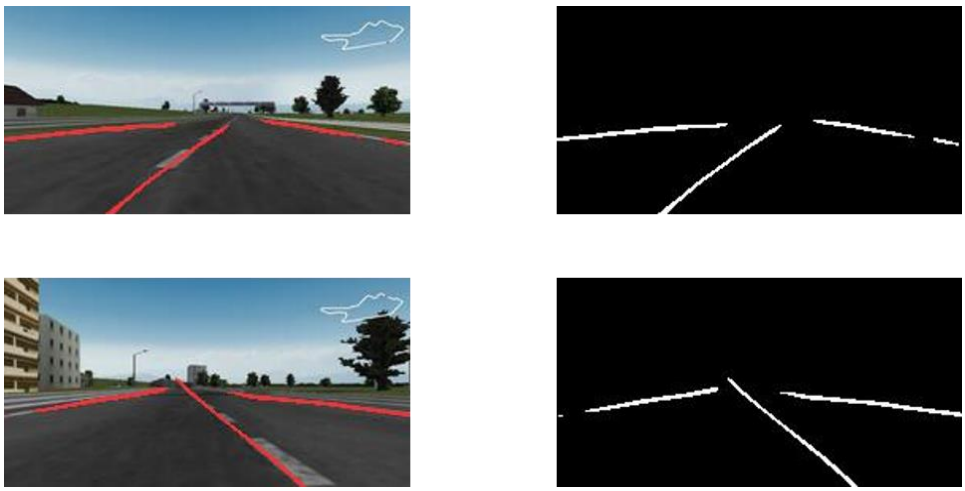

그림 18. TORCS 프로그램 딥러닝 기반 차선 인식 결과

제 5 장 결 론

본 학위 논문에서는 분산된 컴퓨팅 환경에서 Splash 응용 프로그램을 사용하기 위해서 Splash과 ROS 2 런타임을 포함한 다양한 소프트웨어 빌드 및 설치가 필요하여 소프트웨어 업데이트 및 유지보수를 하는데 시간과 비용이 많이 든다는 불편함을 해결하기 위해서 도커 기반 Splash 빌드 유닛을 제안하였다. 도커 기반 Splash 빌드 유닛을 통해 확장성, 가용성, 관리성을 고려한 소프트웨어 자동화 배포 및 관리가 가능하다. 또한, 이를 통해서 개발자는 소프트웨어 동시성과 병렬성을 최적화할 수 있다. 제안한 빌드 유닛을 Splash 프레임워크에 구현하였다. 또한 개발한 빌드 유닛의 유용성을 확인하기 위해서 분산 시스템에서 오픈소스 자동차 주행 시뮬레이션 프로그램과 딥러닝 기반 차선 인식 모델을 이용하여 Splash 응용 프로그램 예제를 진행하였다. 향후에는 Splash가 효과적으로 사용될 수 있는 분야인 자율 주행 시스템 및 딥러닝 응용 프로그램을 탑재한 자율 기기 임베디드 시스템에서 심층적인 사례 연구를 할 계획이다.

참고문헌

[1] T.J. Kim, “Automated Autonomous Vehicles: Prospects and Impacts on Society,” Journal of Transportation Technologies, 8, 137-150, 2018.

[2] H. Kang, C. Lee, W. Choi and S. Hong, “Splash on ROS 2: A Runtime Software Framework for Autonomous Machines,” accepted to proc. The 18th International Conference on Ubiquitous Robots (UR), 2021.

[3] S. Noh and S. Hong, “Programming language support for multisensor data fusion: the Splash approach,” in proc. The 17th International Conference on Ubiquitous Robot (UR), 2020, pp. 429-436.

[4] S. Noh and S. Hong, “Splash: A Graphical Programming Framework for an Autonomous Machine,” in proc. The 16th International Conference on Ubiquitous Robot (UR), 2019, pp. 660-666.

[5] C. Boettiger, “An Introduction to Docker for Reproducible Research,” in ACM SIGOPS Operating Systems Review, Vol. 49, No.1, 2015.

[6] D. Casini, T. Blaß, I. Lütkebohle and B. B. Brandenburg, “Response-time analysis of ROS 2 processing chains under reservation-based scheduling,” in proc. 31st Euromicro Conference on Real-Time Systems (ECRTS), 2019, pp. 6:1-6:23.

[7] Y. Maruyama, S. Kato, and T. Azumi, “Exploring the performance of ROS2,” in proc. The 13th International Conference on Embedded Software (ICESS), 2016, pp. 1-10

[8] Y. Yang, and T. Azumi, “Exploring Real-Time Executor on ROS

2," in proc. 2020 IEEE International Conference on Embedded Software and Systems (ICESS), 2020, pp. 1-8.

[9] B. Yang, "Splash Application Code Generation Method for Reducing Runtime Overhead," Masters Dissertation, Seoul National University Graduate School, 2020.

[10] R. White and H. Christensen, "ROS and Docker". In Robot Operating System (ROS)", Springer, Cham., pp. 285-307. 2017.

[11] Open Source Robotics Foundation (OSRF), "ROS 2," [online] Available: https://github.com/ros2.

[12] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "Torcs, the open racing car simulator," [online] Available: http://torcs.sourceforge.net

[13] Q. Zou et al., "Robust Lane Detection from Continuous Driving Scenes Using Deep Neural Networks," in IEEE Transactions on Vehicular Technology, Vol. 69, Issue.1, 2020.

Abstract

Automated Deployment of a Splash Application to the Distributed System via Splash Build Unit

Wooyoung Choi
Electrical and Computer Engineering
The Graduate School
Seoul National University

Splash is a programming framework that can effectively support the application development of increasingly complex autonomous machines. Splash not only offers user-friendly programming abstraction like a graphic user interface, but also provides essential programming semantics for a real-time control system such as real-time stream processing, sensor fusion, mode change, exception handling, etc. Splash was developed based on ROS 2 to take advantages of using already existing features and packages. Splash applications are eligible to run on a single or distributed computing systems where Splash runtime library, ROS 2 runtime library and any other software that are necessary for the Splash applications are installed. Therefore, in order to run Splash application on a distributed computing system, it is required to build and install the applications, additional external libraries and corresponding runtime according to the environment of each machine. This particular process comes with cost and time issue of

software update and maintenance. This inconvenience and burden raise the need for automated software deployment considering scalability, availability, and manageability in distributed computing systems. This thesis will introduce a Docker-based Splash build unit for automated deployment and management of Splash applications. Splash build unit allows developers to automate the software build and deployment process to distributed computing machines by using Docker containerization. It also allows developers to focus on optimizing parallelism and concurrency of programs. In order to demonstrate the effectiveness and utility of software deployment using Splash build unit, the case study of a LFA(Lane Following Assist) application with DNN-based lane detection and DNN-model training program was conducted using Splash build unit in a distributed system.

Keywords : Splash, programming framework, multicore distributed system, automated deployment, autonomous machine, ROS 2
Student Number : 2019-26977