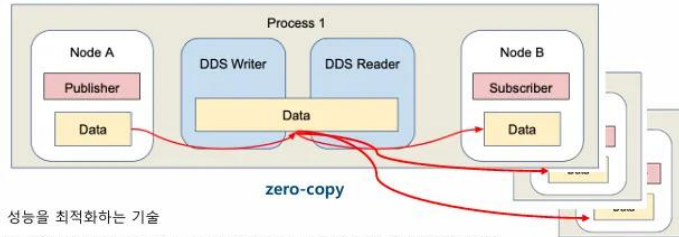


스터디 주간 활동 보고서

팀명	별꿀오소리	제출자 성명	정찬원
참여 명단	정찬원, 송주훈, 강인우, 정민섭		
모임 일시	2025년 05월 01일 22시 ~ 23시 00분(총 1시간)		
장소	Google meet 화상 회의	출석 인원	4/4
학습목표	ROS2 평가에 대비하여 강의별로 단원을 하나씩 선정 후 리뷰해본다.		
학습내용	<p>- 정찬원: IPC(Intra-process communication), QOS</p> <p>1. IPC: ROS2에서는 여러 노드들이 데이터를 주고 받는데, 기본적으로는 DDS라는 미들웨어를 사용해서 통신한다. 이때, 각 노드가 별개의 프로세스일 경우 데이터는 메모리 복사와 네트워크 전송을 거치기 때문에 속도 저하와 메모리 낭비가 생기기에 ROS2에서는 이를 해결하고자 IPC를 제공한다.</p> <p>아래 사진처럼 서로 다른 프로세스는 송수신되는 데이터가 여러 번 메모리에 복사되어 성능이 저하된다.</p> <div data-bbox="379 1653 1120 1982"> <p>■ 서로 다른 프로세스는 송수신되는 데이터가 여러 번 메모리에 복사되어 성능 저하가 발생</p> <p>두 노드 간의 일반적인 데이터 흐름</p> <p>[기본적인 데이터 복사]</p> <p>일반적으로 ROS 2에서 노드 간에 데이터를 전달할 때, 메시지는 다음과 같은 단계를 거칩니다</p> <ol style="list-style-type: none"> 1. 퍼블리셔가 메시지를 생성 <ul style="list-style-type: none"> • 사용자가 메시지를 생성하면, 해당 데이터가 메모리에서 특정 주소에 저장됨 2. DDS 미들웨어를 통한 데이터 전달 <ol style="list-style-type: none"> 1. ROS2는 DDS(Data Distribution Service)를 사용하여 메시지를 전달함 2. 일반적인 DDS 구현에서는 데이터를 네트워크 버퍼 또는 공유 메모리로 복사하여 송신함 3. 구독자가 데이터를 수신 <ol style="list-style-type: none"> 1. 수신된 데이터는 다시 사용자 프로그램의 메모리로 복사됨 2. 여러 개의 구독자가 존재할 경우, 각 구독자에게 별도로 복사됨 <p>이 과정에서 여러 번의 메모리 복사가 발생하며, 특히 대용량 데이터(이미지, LIDAR 포인트 클라우드 등) 전송 시 메모리 사용량 증가와 성능 저하 발생</p> </div>		

그러나 IPC를 이용하면 여러 개의 노드를 단일 프로세스에서 처리하기 때문에 메모리가 복사되지 않는다. 이를 zero-copy라 부른다. 메모리 복사 없이 주소 참조만으로 데이터를 전달하는 방식이다.

- IPC를 이용하면 복수개의 노드를 단일 프로세스에서 처리하여 해당 문제를 해결



[Zero-Copy란?]

데이터 복사를 최소화하여 성능을 최적화하는 기술

ROS 2에서는 Fast DDS SHM (Shared Memory) 및 Cyclone DDS Iceoryx 등의 DDS 미들웨어에서 지원

[Zero-Copy 방식의 데이터 흐름]

1. 퍼블리셔가 메시지를 생성하면, 데이터가 공유 메모리(SHM, Shared Memory)에 저장됨
2. DDS는 데이터를 네트워크로 전송하지 않고, 같은 프로세스 또는 동일한 머신 내의 구독자들에게 참조 방식으로 전달
3. 구독자는 데이터를 복사 없이 직접 참조하여 사용함
4. 즉, 데이터가 한 번만 생성되고 여러 구독자가 이를 직접 읽을 수 있어 메모리 복사가 필요 없음

2. QoS(Quality of Service)

ROS2는 DDS 기반이기에 QoS 설정을 통해 데이터 통신의 방식(신뢰성, 속도, 저장 방식 등)을 유연하게 제어할 수 있다. DDS에서 publisher 혹은 subscriber를 선언할 때 QoS를 매개변수로 지정하여 데이터 전송 시 실시간성, 대역폭, 데이터 지속성, 중복성 등을 유연하게 설정할 수 있다.

QoS의 대표적인 항목으로는 크게 6개가 있다.

1) Reliability: 신뢰도 우선 설정하거나, 통신 속도 최우선 설정

- BEST_EFFORT: 데이터 송신에 집중하기에 전송 속도를 중시한다.

- RELIABLE: 데이터 수신에 집중하기에 신뢰성을 중시한다.

2) History: 정해진 크기만큼 데이터 보관하는 기능 (=depth)

- KEEP_LAST: 정해진 메시지 큐 사이즈만큼 데이터를 보관하여 최근 N개만 보관하다.

- KEEP_ALL: 모든 데이터를 보관한다.

3) Durability: 데이터 수신하는 서브스크라이버 생성되기 전, 데이터 사용 유무 설정

- TRANSIENT_LOCAL: SubSubscription이 생성되기 전 데이터도

보관(Publisher에만 적용 가능). 새로운 구독자도 최근 데이터를 받을 수 있다.

- VOLATILE: Subscription 생성되기 전 데이터는 무효이다. 새로운 구독자가 생기면 그 시점부터 데이터를 수신한다.

4) Deadline: 정해진 주기 내 데이터 발신 및 수신에 없는 경우 이벤트 함수 실행

- deadline_duration을 통해 deadline의 주기를 확인할 수 있다.

5) Lifespan: 정해진 주기 내 수신되는 데이터에만 유효 판정, 이외 데이터 삭제

- lifespan_duration을 통해 lifespan의 주기를 확인할 수 있다.

6) Liveliness: 정해진 주기 내 노드 또는 토픽 생사 확인

- AUTOMATIC, MANUAL_BY_NODE, MANUAL_BY_TOPIC을 자동 또는 매뉴얼을 통해 확인한다.

- 강인우: RQt, Lifecycle, security

1. RQt: ROS2 시스템을 시각적으로 조작하기 위한 GUI 툴킷이며 Qt 기반으로 플러그인 방식으로 기능을 확장하는 것이 가능하다. ROS 생태계 내에서 특정한 목적을 가지고 개발된 프레임워크이다. 단순 바인딩처럼 Qt의 모든 기능을 그대로 사용할 수 있게 하는 것이 아닌 ROS 환경에 맞춰 쉽게 GUI 도구를 사용할 수 있도록 여러 가지 편의 기능 및 구조를 제공한다.

- RQt 플러그인 패키지

패키지 이름	설명
rqt_gui	여러 rqt 위젯을 단일 창에 도킹할 수 있는 위젯 패키지 rqt 명령어를 실행하면 이게 작동함
rqt_gui_cpp	C++ 클라이언트 라이브러리를 사용하여 제작할 수 있는 RQt GUI 플러그인 API 제공
rqt_gui_py	Python 클라이언트 라이브러리를 사용하여 제작할 수 있는 RQt GUI 플러그인 API 제공
rqt_py_common	Python으로 작성된 RQt 플러그인에서 공용으로 사용되는 기능을 모듈로 제공하는 패키지
rqt_common_plugins	rqt_action, rqt_bag 등 20여개의 RQt 플러그인을 포함하는 메타패키지
qt_gui_core	qt_gui, qt_gui_cpp, qt_gui_py.common, qt_gui_app, qt_dotgraph 등을 담은 메타패키지
python_qt_binding	QtCore, QtGui, QtWidgets 등을 사용할 때 Python 언어 기반의 Qt API를 제공하는 바인딩 패키지

- 실습 자료 요약(rqt_example)

```

kiwi@kiwi-sam0: ~/ros2_ws/src/rqt_example
kiwi@kiwi-sam0: ~/ros2_ws/src/rqt_example 80x21
kiwi@kiwi-sam0:~/ros2_ws/src/rqt_example$ tree
.
├── CHANGELOG.rst
├── CMakeLists.txt
├── launch
│   └── turtlesim.launch.py
├── package.xml
├── plugin.xml
├── resource
│   └── rqt_example.ui
├── scripts
│   └── rqt_example
├── src
│   └── rqt_example
│       ├── __init__.py
│       ├── examples.py
│       └── examples_widget.py
└── 5 directories, 10 files
kiwi@kiwi-sam0:~/ros2_ws/src/rqt_example$

```

- 1) CHANGELOG.rst: 패키지의 버전별 변경 이력
- 2) CMakeList.txt
- 3) __init__.py: 파이썬 패키지 인식하기 위한 용도. 비어있는 스크립트.
- 4) src/rqt_example/examples.py: Plugin 클래스를 상속하여 UI 초기화 및 위젯을 등록한다.
- 5) src/rqt_example/examples_widget.py: ui 파일을 불러와 GUI 구성을 적용하며 ROS2 topic publisher/subscriber 및 service server/client를

생성한다. 버튼, 키보드 입력으로 속도를 제어하며 수신한 속도값을 슬라이더와 LCD에 실시간으로 표시한다. 또한 라디오 버튼으로 LED 상태 제어 서비스를 호출한다.

6) package.xml: ROS 2 패키지의 메타데이터와 의존성, 플러그인 등록 정보를 정의하며 RQt에 이 패키지에서 제공하려는 플러그인을 추가한다.

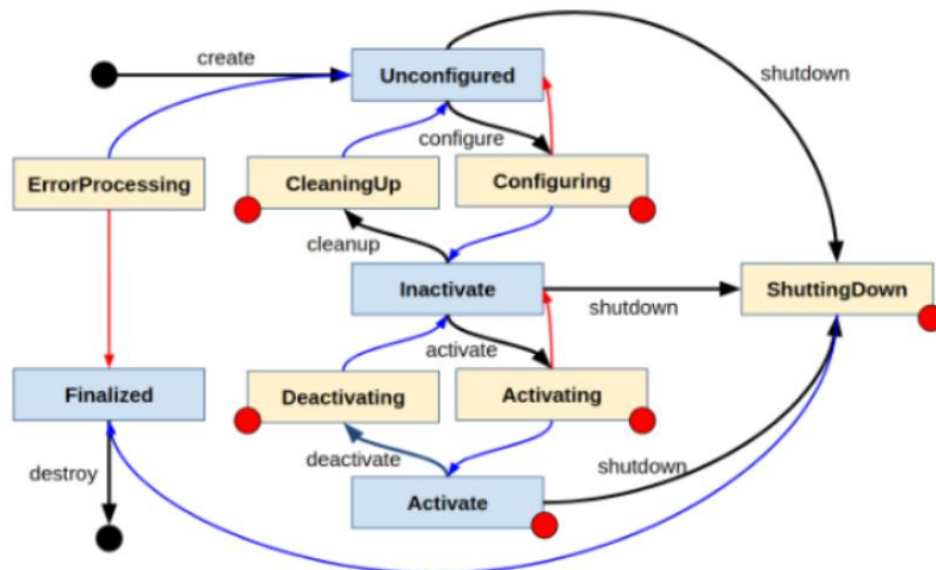
7) plugin.xml: Group 태그가 메뉴의 세부 항목이 되며 <label>, <icon>, <statustip>이 해당 RQt 플러그인의 속성이 된다.

8) resource/rqt_example.ui: RQt 플러그인의 GUI 레이아웃을 정의한 Qt Designer용 XML 파일이다.

9) launch/turtlesim.launch.py: /turtle1/rqt_example, /turtlesim 노드를 실행한다.

10) scripts/rqt_example: RQt 플러그인을 단독으로 실행할 수 있도록 설정된 진입점이다.

2. ROS2 Lifecycle : ROS2에서는 노드 상태를 관리하기 위해 Lifecycle 인터페이스를 제공하여 노드를 동작 이전 상태에서 설정하며 안전하게 중단 및 재설정을 가능하게 한다.



3. ROS2 security: SROS의 유틸리티로, DDS-Security를 ROS2에서 사용하기 위해 필요한 도구를 모아둔 것이다.

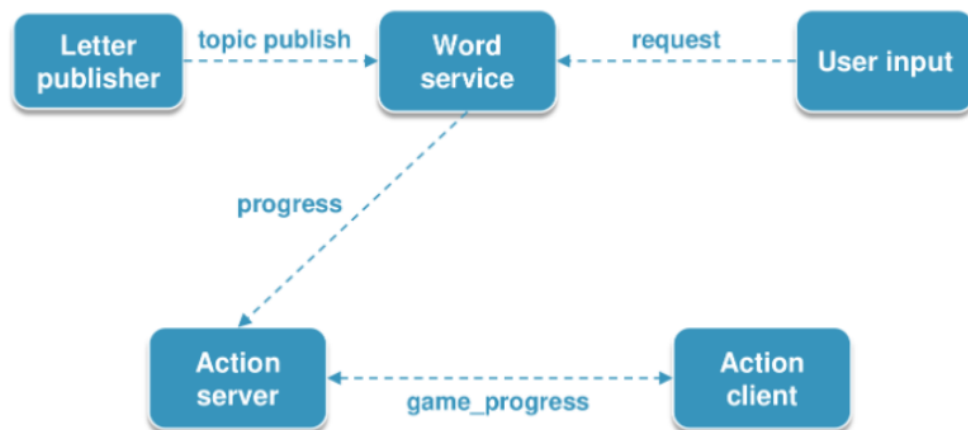
환경변수 3가지로는,

- 1) ROS_SECURITY_KEYSTORE: 보안 설정 파일 저장소 경로
- 2) ROS_SECURITY_ENABLE: 보안 활성화 여부 true, false
- 3) ROS_SECURITY_STRATEGY: 보안 설정 방법. Enforce는 보안이 없으면 통신을 차단하며 Permissive는 비보안 노드도 허용한다.

- 정민섭, 송주훈: 인터페이스 프로그래밍(Hangman 게임 구현)

1. Hangman게임 개요: 단어나 문장을 추측하는 고전 게임으로써 시스템은 단어를 숨기고, 사용자는 글자를 하나씩 맞추며 승패를 결정한다. 틀릴 때마다 기회가 줄어들며 정답을 전부 맞추면 승리, 모든 기회를 소진하면 패배한다.

2. 시스템 구조



3. 인터페이스 파일 정의

CheckLetter.srv

```
# Request
string letter

---

# Response
string updated_word_state # 예: p y _ _ o n
bool is_correct           # 입력한 글자가 단어에 포함되었는지
string message            # "Correct" 또는 "WRONG"
```

- 클라이언트가 추측한 **letter**를 보내고,
- 서버는 현재 단어 상태, 정답 여부, 메시지를 응답

Progress.msg

```
string current_state # 현재 상태 예: p y _ o n
int32 attempts_left # 남은 기회 수
bool game_over      # 게임 종료 여부
bool won            # 승리 여부
```

- 서비스 응답 이후, 전체 상태를 토픽으로 퍼블리시하기 위한 메시지
- 액션 서버에서도 이 토픽을 구독하여 상태 업데이트에 활용

GameProgress.action

```
# Goal (사용 안 함)
---

# Result
bool game_over # 게임이 끝났는지
bool won       # 사용자가 이겼는지
---

# Feedback
string current_state # 현재 맞춘 상태
int32 attempts_left  # 남은 기회 수
```

- 액션 서버가 클라이언트에게 주기적으로 피드백을 제공 (현재 상태, 남은 시도 횟수)
- 최종적으로 게임 종료 후 승패 여부를 전달

4. 주요 노드

1) letter_publisher.py: publisher_letter를 통해 하나의 알파벳 a~z를 순서대로 publish한다.

2) word_service.py: letter_callback은 letter_topic을 통해 수신된 메시지를 처리하고 저장한다. check_letter_callback은 수신한 문자가 현재 단어에 포함되어 있는지 확인하고 상태를 업데이트한다. 업데이트된 상태는 progress.msg를 통해 publish하며 임의의 단어를 선택하고 행맨 게임을 진행한다.

3) user_input.py: input_thread는 enter키 입력을 기다리며 입력이 들어오면 send_request 함수를 호출한다. send_request는 CheckLetter 서비스에 요청을 보내 현재 선택된 문자가 단어에 포함되어 있는지 확인한다.

4) progress_action_client.py: goal을 설정하고 action server와 상호 작용을 통해 게임의 진행 상황을 업데이트한다.

5) progress_action_server.py: progress_callback을 통해 progress

topic으로부터 수신한 메시지를 처리한다. 추가로 `execute_callback`을 통해 클라이언트의 목표 요청을 수신하고 게임 진행 상황을 피드백하며 게임 종료시 결과를 반환한다. 주기적으로 클라이언트에게 게임 상태를 전달한다.

또한 `MultiThreadExecutor()`를 사용하여 동작한다.

5. `setup.py`, `hangman_interfaces/CMakeLists.txt` 수정

`hangman_interfaces/CMakeLists.txt`

```
#find dependencies
find_package(ament_cmake REQUIRED)
find_package(builtin_interfaces REQUIRED)
find_package(rosidl_default_generators REQUIRED)

ament_export_dependencies(rosidl_default_runtime)

# uncomment the following section in order to fill in
# further dependencies manually.
# find_package(<dependency> REQUIRED)
```

```
set(msg_files
  "msg/Progress.msg"
)

set(srv_files
  "srv/CheckLetter.srv"
)









set(action_files
  "action/GameProgress.action"
)

rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
  ${srv_files}
  ${action_files}
  DEPENDENCIES builtin_interfaces
)
```


setup.py

```
entry_points={
    'console_scripts': [
        'letter_publisher = hangman_game.letter_publisher:main',
        'word_service = hangman_game.word_service:main',
        'user_input = hangman_game.user_input:main',
        'progress_action_server = hangman_game.progress_action_server:main',
        'progress_action_client = hangman_game.progress_action_client:main',
    ],
}
```

6. 요약

노드	ROS 2 통신 방식	설명
letter_publisher	 Publisher (<code>std_msgs/msg/String</code>)	- 알파벳을 <code>letter_topic</code> 으로 퍼블리시- 다른 노드가 이를 수신해 사용
word_service	 Subscriber (<code>letter_topic</code>)  Service Server	- 퍼블리시된 알파벳을 구독- 서비스 요청을 받아 정답 판별- 게임 진행 상태를 토픽으로 퍼블리시
	(<code>CheckLetter.srv</code>)  Publisher (<code>Progress.msg</code>)	
user_input	 Service Client (<code>CheckLetter.srv</code>)	- 사용자가 Enter 키 누르면 → <code>check_letter</code> 서비스 요청 전송
progress_action_server	 Subscriber (<code>progress</code>)  Action Server (<code>GameProgress.action</code>)	- <code>word_service</code> 에서 퍼블리시된 진행 상황을 구독- 액션 클라이언트에게 피드백 & 최종 결과 전송
progress_action_client	 Action Client (<code>GameProgress.action</code>)	- 액션 서버에 목표(goal)를 전송- 피드백과 결과를 비동기적으로 수신

활동평가

ROS2 정기평가에 대비하여 각자 중요하다고 생각되는 단원 하나씩 리뷰하며 수업시간 때 들었던 강의 내용을 복기할 수 있었다. IPC와 QoS, RQt와 ROS2 Lifecycle, security, Hangman game이 어떤 식으로 동작하는지 다시 한번 확인할 수 있었던 시간이었다.

과제	ROS2 평가 5/8 평가 잘 보기
향후 계획	5/9 대면 프로젝트 때 건강한 얼굴로 만나기
첨부 자료	<p>1. 활동 사진</p> 