# 사용자 매뉴얼

# KR-RTS

(Koras Robotics – Robot Tooling System)


**㈜코라스로보틱스**

**Koras Robotics**

www.korasrobotics.com

E) koras@korasrobotics.com, P) 02-929-8501, F) 02-923-3591

# 목 차

# 1. 서 론

이 문서는 (주)코라스로보틱스의 로봇 툴링 시스템인 KR-RTS (Koras Robotics – Robot Tooling System)에 대한 매뉴얼이다. 이 매뉴얼은 코라스로보틱스의 국제 특허인 모터 내장형 자동 툴 체인저MATC (Motorized Automatic Tool Changer) 및 이에 호환되는 다양한 툴의 사용 방법 및 사용 시의 주의 사항을 설명한다. 본 제품을 사용하기에 앞서 주의 사항과 사용법을 숙지하는 것은 사고 방지 및 효율적인 사용에 매우 중요하다.

## 1.1 주의 사항

MATC 또는 구동부가 내장된 일체형 툴의 사용 시에 다음의 주의 사항을 준수하여야 한다.

* 로봇을 작동하기 전에 MATC 또는 툴을 로봇에 견고하게 고정하여야 한다.
* 손상된 MATC 또는 툴을 설치하거나 작동시켜서는 안 된다.
* 케이블 연결은 반드시 전원이 꺼진 상태에서 진행하여야 한다.
* 제품이 물이나 이물질에 노출되지 않도록 주의하여야 한다.
* MATC 또는 툴에 정격 전압 및 전류 이외의 전원을 공급하여서는 안 된다.
* 전원 및 통신 케이블은 항상 장치에 제대로 연결되어 있는지를 확인한다.
* MATC 또는 툴의 동작 시에는 손이나 옷이 가동부와 접촉하지 않도록 주의한다.
* 강한 외력 및 충격 발생 시 MATC 및 툴에 손상이 발생할 수 있다.
* MATC 또는 툴을 부적절하게 사용하면 장치의 손상이 발생할 수 있다.

## 1.2 저작권



## 1.3 제품 보증

· 보증 기간: 제품 구매일로부터 12개월
· 보증 범위: 정상적인 사용 중에 발생한 제조상의 결함이나 고장
· 다음의 경우 보증 대상에서 제외됩니다:
  - 사용자 부주의 또는 임의 개조로 인한 고장
  - 외부 충격이나 천재지변으로 인한 손상
  - 보증 기간이 경과한 경우
  - 정품이 아닌 부품 또는 소모품의 사용으로 인한 문제

# 2. 로봇 툴링 시스템 KR-RTS

㈜코라스로보틱스의 로봇 툴링 시스템인 KR-RTS(Koras Robotics – Robot Tooling System)는 새로운 개념의 시스템이다. 기존의 로봇 툴링 시스템에서는 주로 하나의 툴만을 장착하여 작업을 하거나 툴을 교체하더라도 작업 전에 수작업을 통해서 툴과 툴의 전원/통신 케이블을 교체하는 방식을 취하였다. 반면에, KR-RTS에서는 모터가 내장되어 동력 제공 기능을 갖는 모터 내장형 자동 툴 체인저인 MATC (Motorized Automatic Tool Changer)를 이용하여 자체 구동부를 갖지 않지 않고 순수한 기구부만으로 구성되는 다양한 MATC용 툴을 자동으로 교체하면서 작업을 수행한다. 즉, 모든 툴이 MATC의 로봇측 유닛에 내장된 단일의 구동부(모터 및 제어기)를 공유하며, 따라서 단일의 전원/통신 케이블이 로봇 또는 상위 제어기에 연결되므로, 작업 중에 자동으로 툴의 교체가 가능하다. 그림 2.1에서 보듯이, MATC는 로봇 플랜지에 장착되는 로봇측 유닛(robot-side unit)과 툴에 장착되는 툴측 유닛(tool-side unit)으로 구성되어 있다. 그림 2.2는 MATC와 10여 종의 툴 라인업을 보여준다. 현재 3종의 MATC와 15여 종 이상의 MATC용 툴이 개발되어 판매 중이다.


그림 2.1 MATC의 외관: MATC 로봇측 유닛 (좌) 및 MATC 툴측 유닛 (우)

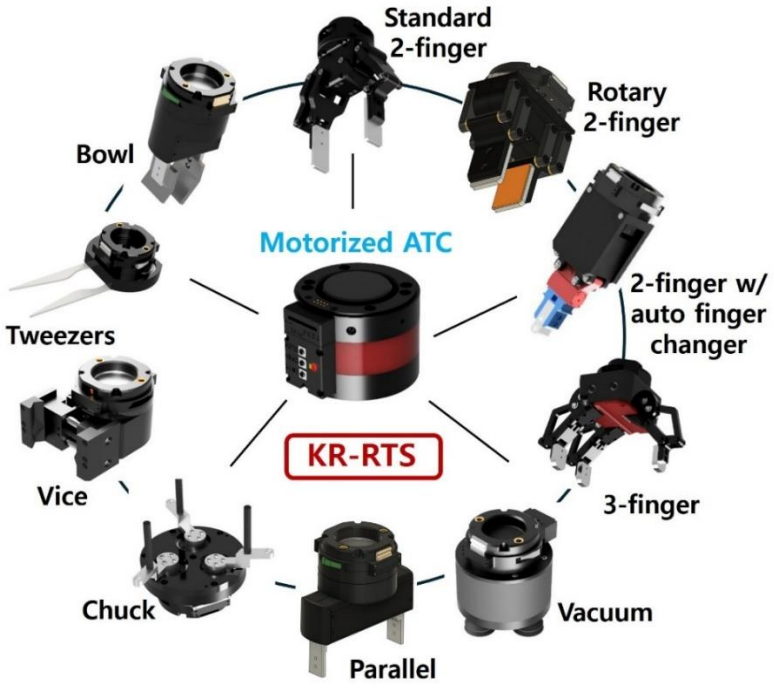



그림 2.2 KR-RTS을 구성하는 MATC와 대표적인 10종의 툴

로봇에 장착되는 툴의 대부분은 물체를 파지하는 용도의 그리퍼이므로, 앞으로는 툴 대신에 그리퍼라는 용

어를 사용하기로 하지만, 그리퍼 대신에 툴을 사용하여도 본 매뉴얼에서의 모든 설명이 유효하다. 또한, KR-RTS에도 MATC에 호환되는 툴 외에도 기존의 상용 그리퍼와 마찬가지로 구동부가 내장된 일체형 툴(integrated tool)도 제공된다.

KR-RTS의 적용에 대한 이해를 돕기 위해서, 2지 그리퍼의 두 가지 적용 예를 들어서 설명한다.

**방식 1) MATC + MATC용 툴**

그림 2.3에서와 같이 MATC의 로봇측 유닛은 로봇 플랜지에 장착되며, MATC의 툴측 유닛은 툴의 상단에 장착된다. 각 툴은 툴 교환 스테이션(tool change station)의 툴 홀더(tool holder)에 거치되어 있으며, 로봇의 동작을 통해서 MATC의 로봇측 유닛과 툴측 유닛이 체결되면 자동적으로 툴이 로봇에 결합되며, MATC 구동부에서 각 툴의 특성과 작업에 맞는 모터 토크/속도를 툴로 전달하여 툴이 동작하게 된다. 여기서 MATC용 툴은 자체 구동부를 내장하지 않으며, 로봇 유닛과의 결합을 위한 툴측 유닛이 툴의 상부에 장착되어 있다.

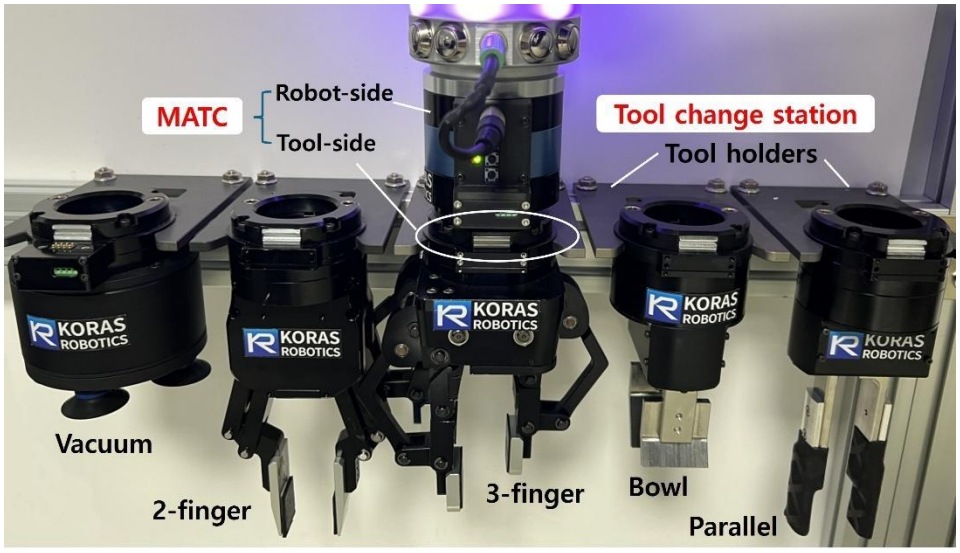



그림 2.3 다수의 툴 홀더로 구성된 툴 교환 스테이션.

**방식 2) 일체형 툴**

기존의 상용 툴과 동일하게 일체형 툴은 자체 구동부를 포함하고 있다. 로봇에 브라켓을 통하여 직접 체결되거나 일반적인 ATC를 통해서 체결된다. 그림 2.4는 MATC용 2지 그리퍼와 일체형 2지 그리퍼를 보여준다. 두 그리퍼는 대부분의 부품을 공유하므로 외관은 거의 비슷하지만, 일체형 그리퍼의 경우 로봇측 유닛과 툴측 유닛 간의 자석 커플링으로 구성된 동력 전달부가 없으므로 전체 높이가 작아진다.

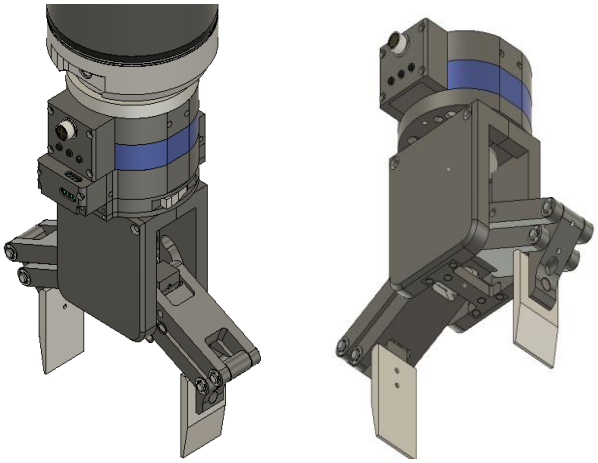


그림 2.4 MATC용 2지 그리퍼 (좌)와 일체형 2지 그리퍼(우)

위의 두 방식 중 어느 방식을 사용하든지 동일한 그리퍼 제어 시스템(Gripper Control System, GCS)인 KR-GCS를 통해서 툴의 제어가 수행된다. 즉, 방식 1에서 MATC에 내장된 모터 제어기와 방식 2에서 일체형 툴에 내장된 모터 제어기는 하드웨어, 제어 방식 및 펌웨어가 동일하다. 그러므로 본 매뉴얼에서는 이들 방식을 구별하지 않고 KR-GCS에 대해서 4.3절에서 설명하기로 하며, 만약 방식의 구별이 필요한 경우에는 별도로 설명하기로 한다.

KR-RTS에서는 주로 방식 1에 해당하는 MATC에 기반한 툴링 시스템을 사용하므로, 이에 대해서 부연 설명을 하기로 한다. 기존의 툴은 로봇의 작업 중에 자동으로 교체하는 것이 불가능한데, 이는 툴 외에도 전원/통신 케이블도 함께 교체되어야 하기 때문이다. 그러므로 툴의 자동 교체를 위해서는, 단일의 전원/통신 케이블을 공통으로 사용하여 어떤 툴을 사용하더라도 케이블이 교체되지 않아야 한다. 이를 위해서 툴을 위한 구동부(모터와 제어기)가 로봇 말단에 위치하여야 하는데, 이를 위해서 기존의 ATC 대신에 모터와 제어기를 내장하는 MATC 방식을 사용한다.

MATC는 로봇에 장착되는 로봇측 유닛과 툴에 장착되는 툴측 유닛으로 구성된다. 앞서 언급하였듯이, 툴을 위한 구동부가 MATC의 로봇측 유닛에 내장되어 있어서 툴이 교체되더라도 전원/통신 케이블은 교체될 필요가 없다. 그리고 툴의 상단에는 MATC의 툴측 유닛이 부착되어 있다. 따라서 MATC는 로봇측 유닛에 내장된 구동부에서 툴의 특성과 작업에 맞게 생성된 모터 토크와 속도를 생성하여 툴측 유닛으로 전달하게 된다.

MATC에서는 래치 잠금(latch locking) 방식의 순수한 기계적 체결 메커니즘을 통해 별도의 동력원 없이 로봇측 유닛과 툴측 유닛 간의 결합 및 분리를 수행하며, 결합 시에 두 파트 간의 강력한 결합 상태를 유지할 수 있다. 또한, MATC 로봇측 유닛과 툴측 유닛의 결합 시에 두 유닛의 동력 전달부가 기어 등과 같이 기계식 장치라면, 결합 및 분리 시에 접촉으로 인한 여러 문제가 발생할 수 있으므로, 기계식 기어 대신에 자석 커플링을 통해서 회전 동력을 전달하게 된다. 즉, 로봇측 유닛에서 모터에 연결된 원형 자석이 회전하면, 이와 약간 이격되어 있는 툴측 유닛의 또다른 원형 자석이 이와 동기화되어 회전하면서 동력을 전달하게 된다. 따라서 빈번한 결합 및 분리 시에도 동력 전달부 간의 기계적 접촉이 없으므로 원활한 교체가 가능하다.

앞서 언급하였듯이, MATC에 내장된 모터/제어기와 일체형 툴에 내장된 모터/제어기는 하드웨어, 제어 방식 및 펌웨어가 동일하다. 표 2.1은 MATC 또는 일체형 툴에 사용되는 모터와 제어기의 사양을 나타낸다.

표 2.1 MATC-10/20/30 또는 일체형 툴의 구동부 사양

| **Specifications** | **MATC-10** | **MATC-20** | **MATC-30** |
|---|---|---|---|
| Payload | 10 kg | 20 kg | 30 kg |
| Supply Voltage | 24 VDC | 24 VDC | 24 VDC |
| Rated current | 1.20 A | 1.20 A | 1.35 A |
| Peak current | 1.80 A | 1.80 A | 2.03 A |
| Communication | Modbus RTU | Modbus RTU | Modbus RTU |
| Rated motor torque | 0.15 Nm | 0.45 Nm | 0.60 Nm |
| Rated motor speed | 900 rpm | 300 rpm | 310 rpm |

# 3. 하드웨어 설치

## 3.1 MATC 방식

### 3.1.1 MATC 로봇측 유닛을 로봇 플랜지에 장착

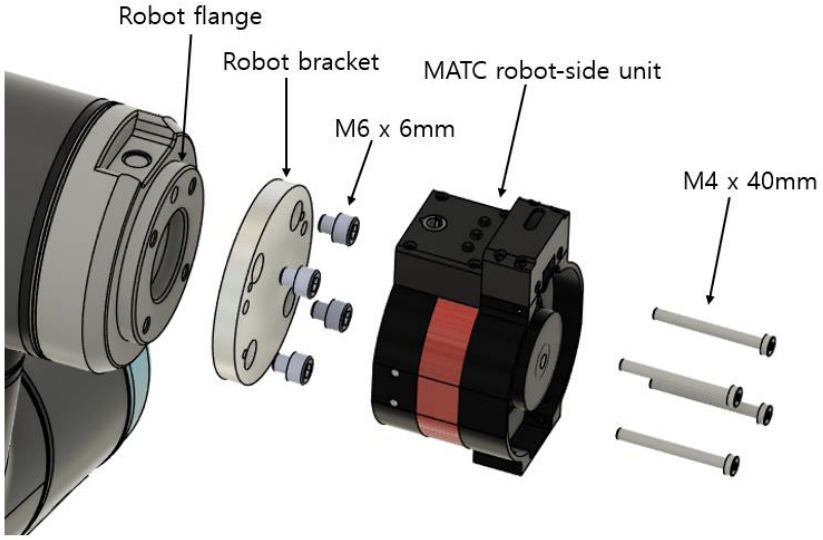



그림 3.1 로봇 브라켓을 이용하여 MATC 로봇측 유닛을 로봇 플랜지에 장착

제공되는 로봇 브라켓을 사용하여 MATC 로봇측 유닛을 로봇 플랜지에 다음과 같이 장착한다.

1. M6 x 6mm 볼트를 이용하여 로봇 브라켓을 로봇 플랜지에 장착한다.
2. M4 x 40mm 볼트를 이용하여 MATC 로봇측 유닛을 로봇 브라켓에 장착한다.

로봇 브라켓과 MATC 로봇측 유닛의 장착을 위한 치수는 다음 도면을 참고한다.

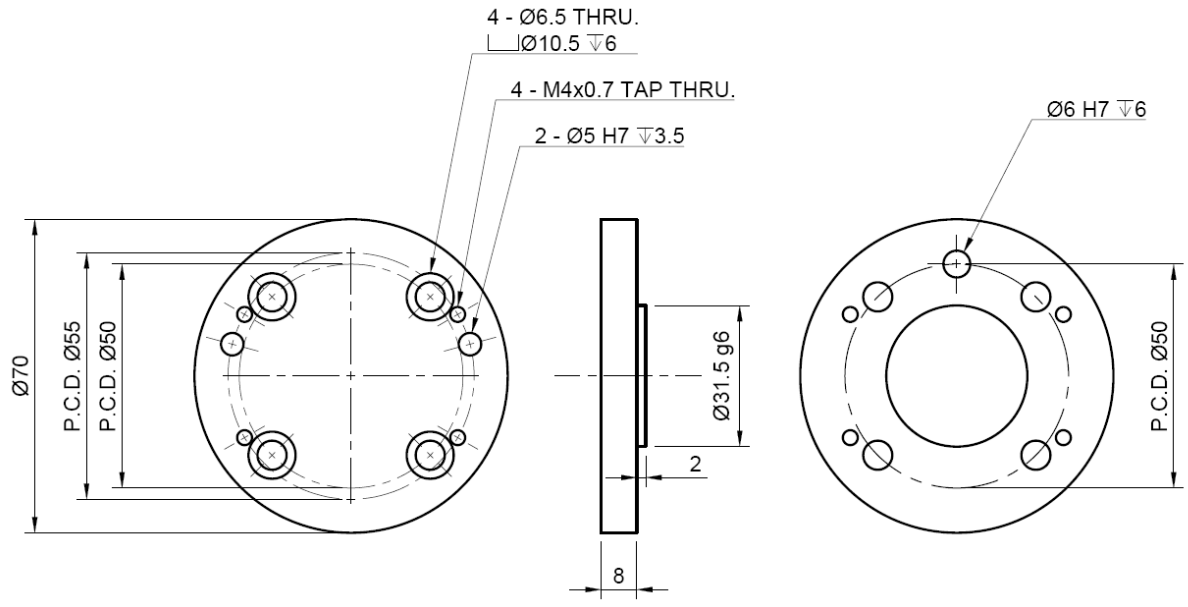



그림 3.2 로봇 브라켓 도면

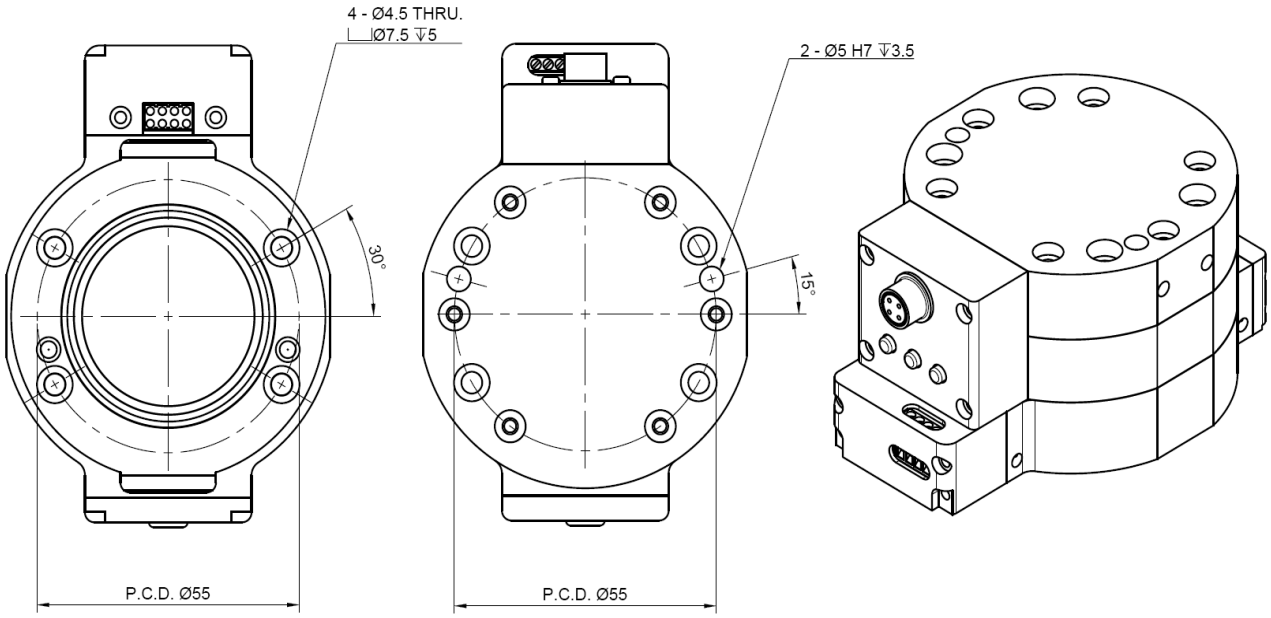



그림 3.3 MATC 로봇측 유닛 도면 (MATC-10-R)

### 3.1.2 툴 교환 스테이션을 이용한 MATC 로봇측 유닛과 툴측 유닛 간의 체결

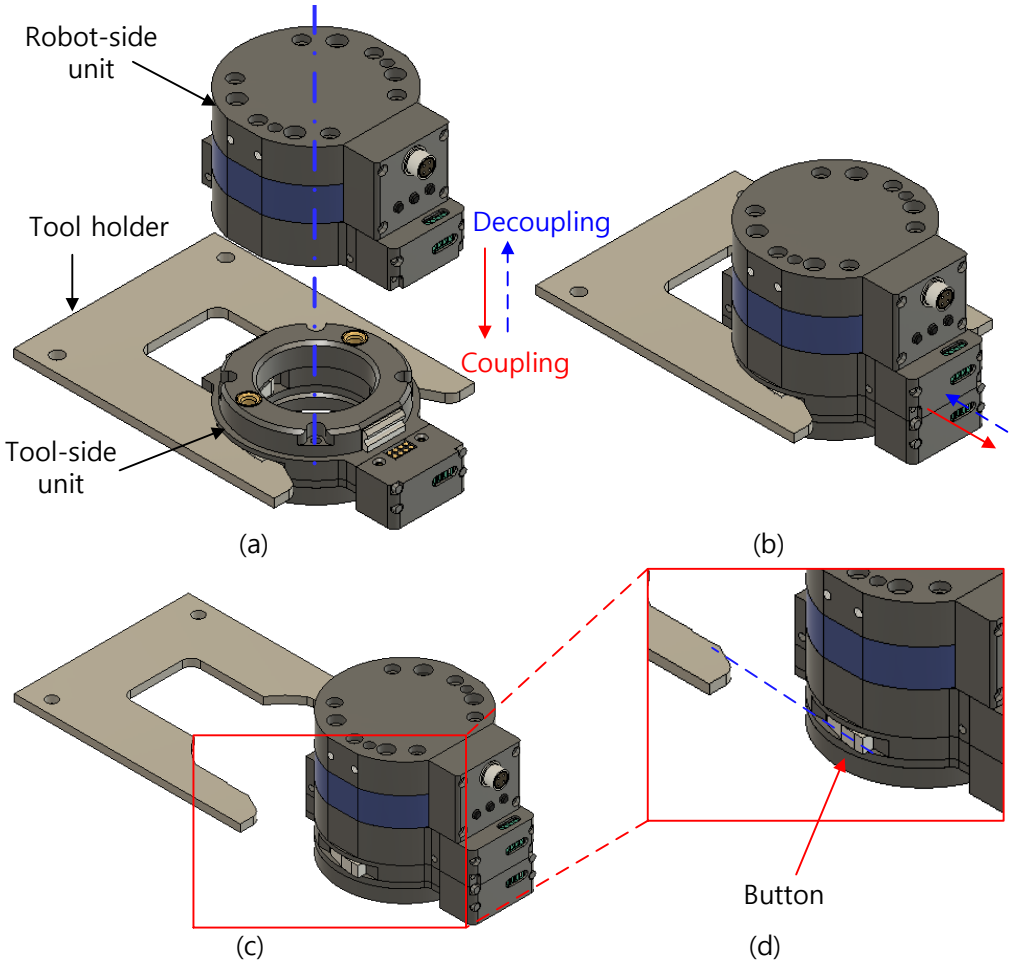



그림 3.4 툴 교환 스테이션을 이용한 MATC 로봇측 유닛과 툴측 유닛 간의 체결 및 해제

MATC 툴측 유닛이 그림 3.54(a)와 같이 툴 교환 스테이션의 툴 홀더에 위치하면 툴 홀더에 의해서 툴측 유

닛의 버튼이 눌려져서 해제 모드가 되어 MATC 로봇측 유닛과 툴측 유닛이 쉽게 분리될 수 있다. MATC 로봇측 유닛과 툴측 유닛을 툴 홀더를 사용하여 결합 및 분리하는 방법은 다음과 같다.

1. 로봇측 유닛의 중심이 툴 홀더에 거치된 툴측 유닛의 중심과 일치하도록 로봇측 유닛을 위치시킨다. (그림 3.4(a))
2. 로봇측 유닛을 수직 아래로 이동하여 툴측 유닛과 체결되도록 한다. (그림 3.4(b))
3. 로봇 제어를 통해서 결합된 MATC 유닛을 툴 홀더의 바깥 방향으로 수평 이동한다. (그림 3.4(c))
4. 툴 홀더와 분리되면서 눌려져 있던 툴측 유닛의 버튼이 원 위치가 되면 로봇측 유닛과 툴측 유닛이 견고한 결합 상태를 유지한다.
5. 로봇측 유닛과 툴측 유닛의 분리를 위해서는 툴측 유닛을 툴 홀더의 안쪽 방향으로 수평 이동시켜서 툴 홀더에 의해서 버튼이 눌려지게 하여 해제 모드로 전환한 후에, 로봇측 유닛을 수직 위로 이동시키면 로봇측 유닛과 툴측 유닛이 분리된다.

만약 툴 홀더를 사용하지 않는다면 MATC 툴측 유닛의 버튼을 손가락으로 세게 눌러서 분리 모드로 전환하면 로봇측 유닛과 툴측 유닛을 체결하거나 해제할 수 있다.

### 3.1.3 툴측 유닛 구조 및 기능

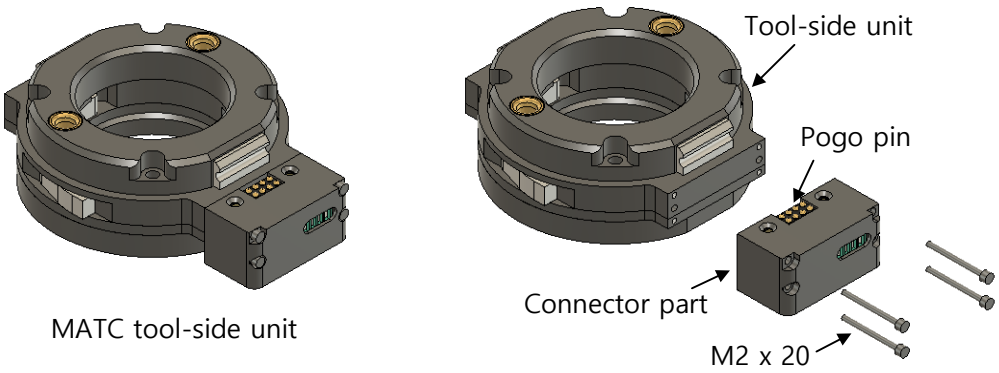



그림 3.5 MATC 툴측 유닛의 구조

그림 3.5에서 보듯이, MATC 툴측 유닛은 로봇측 유닛과의 체결에 필요한 하드웨어 파트로, 별도의 전기적 부품 없이 기계적 부품으로만 구성되어 있다. 앞 절에서 설명한 방법으로 MATC 로봇측 유닛 및 툴측 유닛 간에 체결 및 해제가 수행된다. 이 툴측 유닛에는 필요에 따라서 M2 볼트를 통해 커넥터가 부착될 수 있다.

그림 3.6(a)에서 보듯이, 로봇측 유닛과 툴측 유닛이 체결될 때 로봇측 유닛 커넥터와 툴측 유닛 커넥터에 위치한 포고 핀(pogo pin)이 서로 접촉하면서 전기적으로 연결된다. 이를 이용하면 로봇측 유닛 커넥터 터미널 블록의 단자 a, b, c, d를 전원 공급 및 통신을 위하여 할당할 수 있다. (예를 들어, a와 b는 전원선, c와 d는 통신선) 툴측 유닛 커넥터 터미널 블록의 단자 1, 2, 3, 4는 로봇측 유닛 커넥터 터미널 블록의 단자인 a, b, c, d과 각각 포고 핀을 통하여 연결되어 있으므로, 툴측 유닛에 장착되는 툴에 적절히 연결하여 사용할 수 있다.

이들 커넥터 시스템은 다음과 같이 활용될 수 있다.

첫째, MATC 및 MATC용 툴이 사용되는 경우: 툴의 제어는 MATC의 모터 제어를 통해서 수행되며, 이 경우에 그림 3.9와 같은 제공되는 인터페이스 케이블로 연결되므로, 커넥터 시스템은 툴 제어에는 사용되지 않는다. 그러므로 대부분의 경우에는 커넥터 시스템은 사용되지 않는다. 그러나 툴에 별도의 센서(예, 촉각 센서 등)가 사용된다면, 센서와의 전원 공급이나 통신을 위해서 커넥터 시스템을 사용할 수 있다.

둘째, MATC에 타사의 상용 그리퍼가 사용되는 경우: 타사 그리퍼가 장착된 툴측 유닛이 로봇측 유닛에 체

결된다면, 이 그리퍼에 전원 공급과 통신을 제공하는 데 커넥터 시스템을 사용할 수 있다. 즉, 타사 그리퍼의 전원선과 통신선을 툴측 유닛 커넥터에 연결하고, 로봇측 유닛 커넥터는 상위 제어기에 연결하면 그리퍼의 툴측 유닛이 로봇측 유닛과 체결되면 전원 공급 및 통신이 수행될 수 있다.

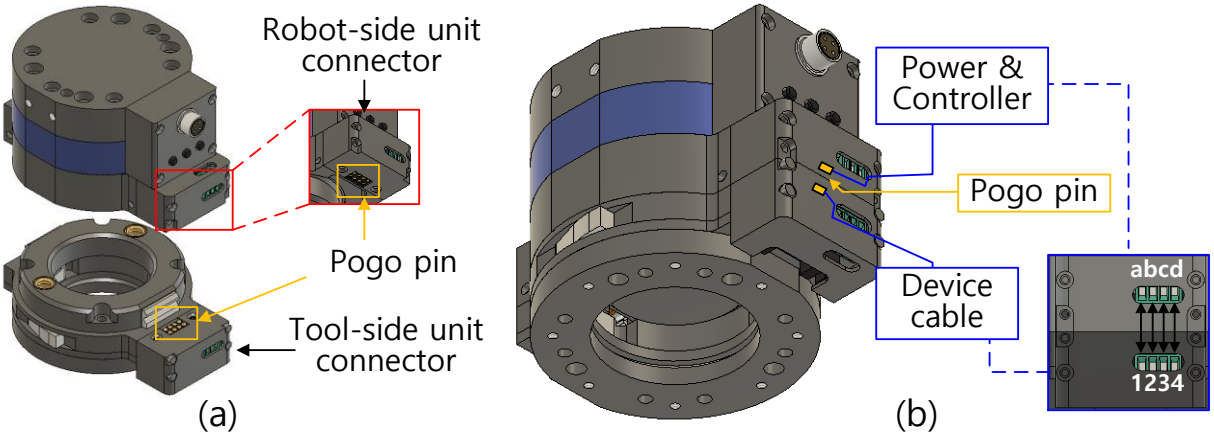



그림 3.6 MATC 툴측 유닛의 기능

위의 활용 사례에서 모터가 내장되어 있는 상용 그리퍼를 굳이 MATC에 체결하는 경우는 흔하지 않지만, 만약 타사의 2지 그리퍼와 그림 2.1의 RTS에서 제공하는 여러 그리퍼를 혼용하여 사용하는 경우에는 MATC를 기반으로 툴링 시스템이 구성되어야 한다.

## 3.2 일체형 툴 방식

구동부가 내장된 일체형 툴(integrated tool)의 경우에는 로봇 브라켓을 사용하여 툴을 로봇 플랜지에 다음과 같이 직접 장착한다.

1. M6 × 6mm 볼트를 이용하여 로봇 브라켓을 로봇 플랜지에 장착한다.
2. M4 × 8mm 볼트를 이용하여 툴을 로봇 브라켓에 장착한다.

로봇 브라켓과 그리퍼의 장착을 위한 치수는 다음 도면을 참고한다.

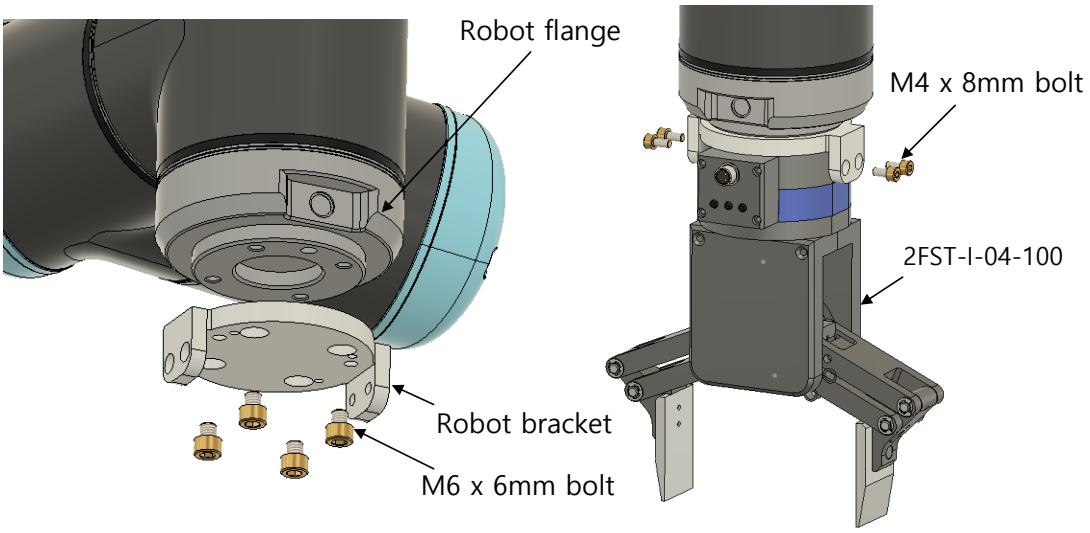



그림 3.7 로봇 브라켓을 이용하여 일체형 툴을 로봇 플랜지에 장착

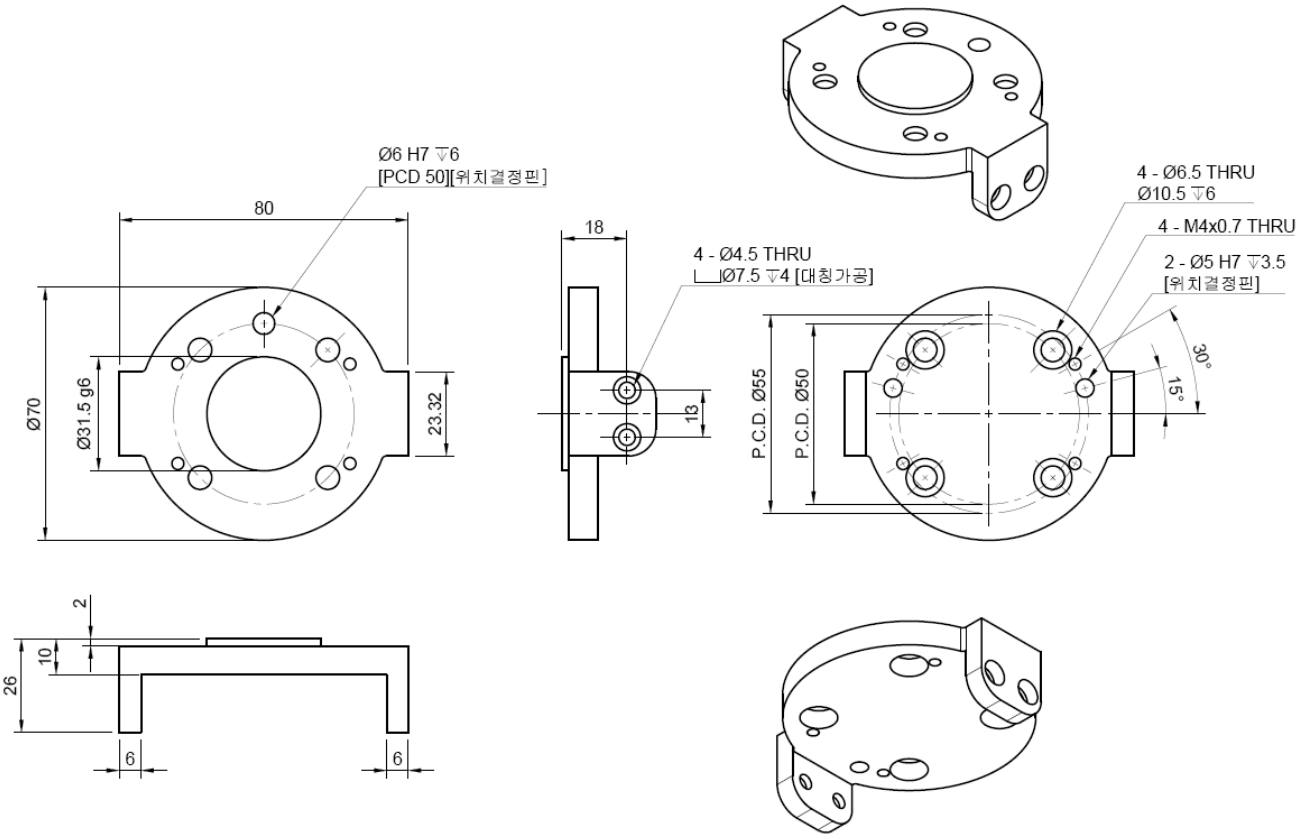



그림 3.8 로봇 브라켓 도면

## 3.3 인터페이스 케이블 연결

로봇-MATC 또는 로봇-일체형 툴 간의 전원 공급 및 통신은 그림 3.9과 같이 인터페이스 케이블을 통하여 수행된다. 케이블의 플러그에 표시된 화살표가 위로 향하도록 리셉터클(receptacle)에 끼워 놓으면 결합된다. 분리 시에는 플러그의 스틸 부분을 잡아당기면 커넥터를 분리할 수 있다. 리셉터클이 짧은 케이블을 통해서 MATC 또는 일체형 툴로부터 분리된 경우에도 유사한 방식으로 커넥터를 연결 또는 분리할 수 있다.

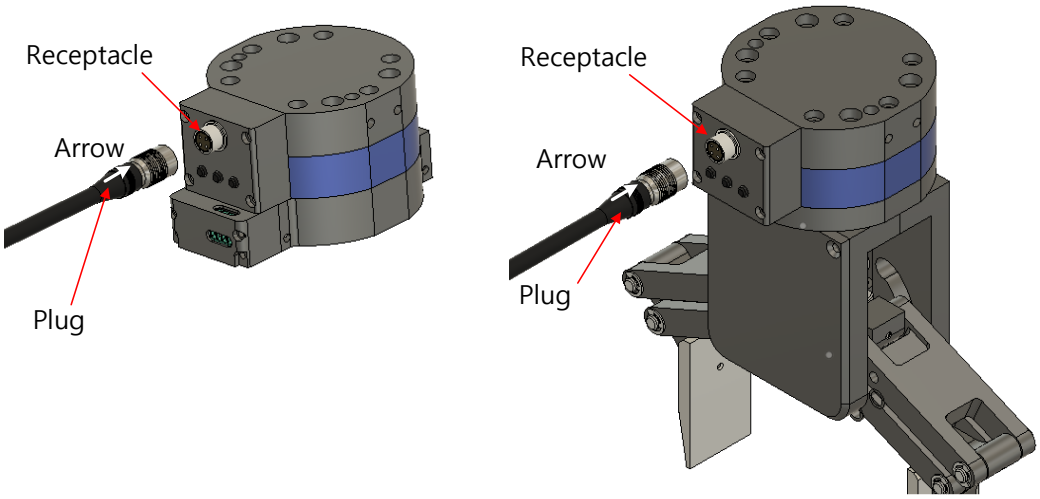



그림 3.9 커넥터 연결: MATC (좌측) 및 일체형 툴 (우측)

플러그 전선 배열은 그림 3.10과 같다.


그림 3.10 플러그 전선 배열

## 3.4 ATC 방식

### 3.4.1 ATC 로봇측 유닛을 로봇 플랜지에 장착

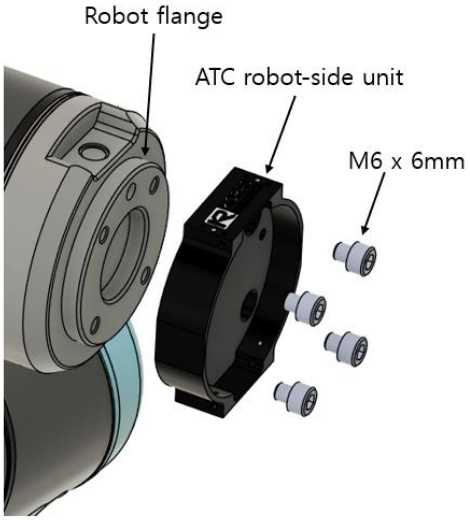



그림 3.11 로봇 브라켓을 이용하여 ATC 로봇측 유닛을 로봇 플랜지에 장착

제공된 로봇 브라켓을 이용하여 ATC 로봇측 유닛을 로봇 플랜지에 다음과 같이 장착한다.

1. M6 × 6mm 볼트를 이용하여 ATC 로봇측 유닛을 로봇 플랜지에 장착한다.

로봇 브라켓과 ATC 로봇측 유닛의 장착을 위한 치수는 다음 도면을 참고한다.

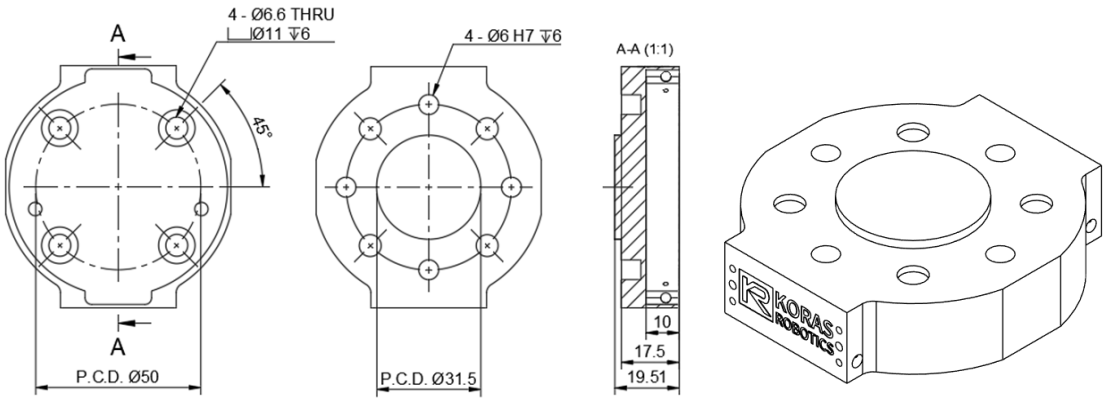



그림 3.12 ATC-10-R 도면

### 3.4.2 ATC 툴측 유닛을 툴에 장착

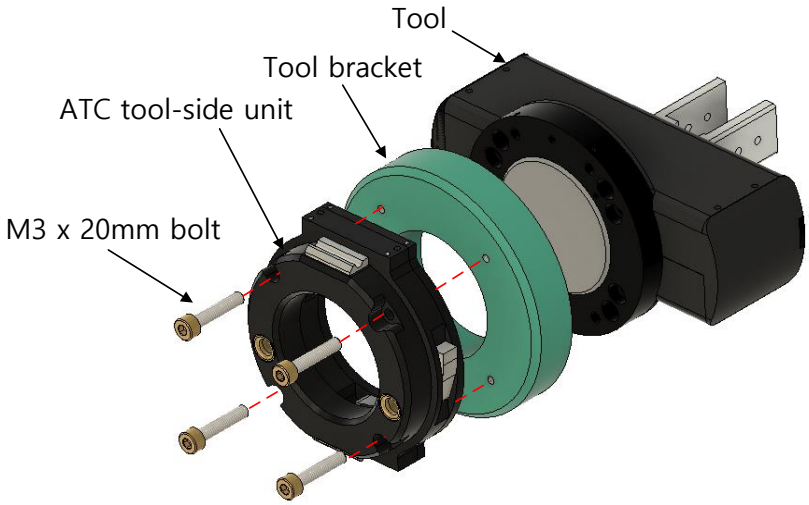



그림 3.13 툴 브라켓을 이용하여 ATC 툴측 유닛을 툴에 장착

툴 브라켓을 이용하여 ATC 툴측 유닛을 툴에 장착하는 방법은 다음과 같다.

1. 해당 툴용으로 제작된 툴 브라켓을 툴에 장착한다.
2. M3 × 20mm 볼트를 이용하여 ATC 툴측 유닛을 툴 브라켓에 장착한다.

ATC 툴측 유닛과 툴 브라켓의 장착을 위한 치수는 다음 도면을 참고한다.

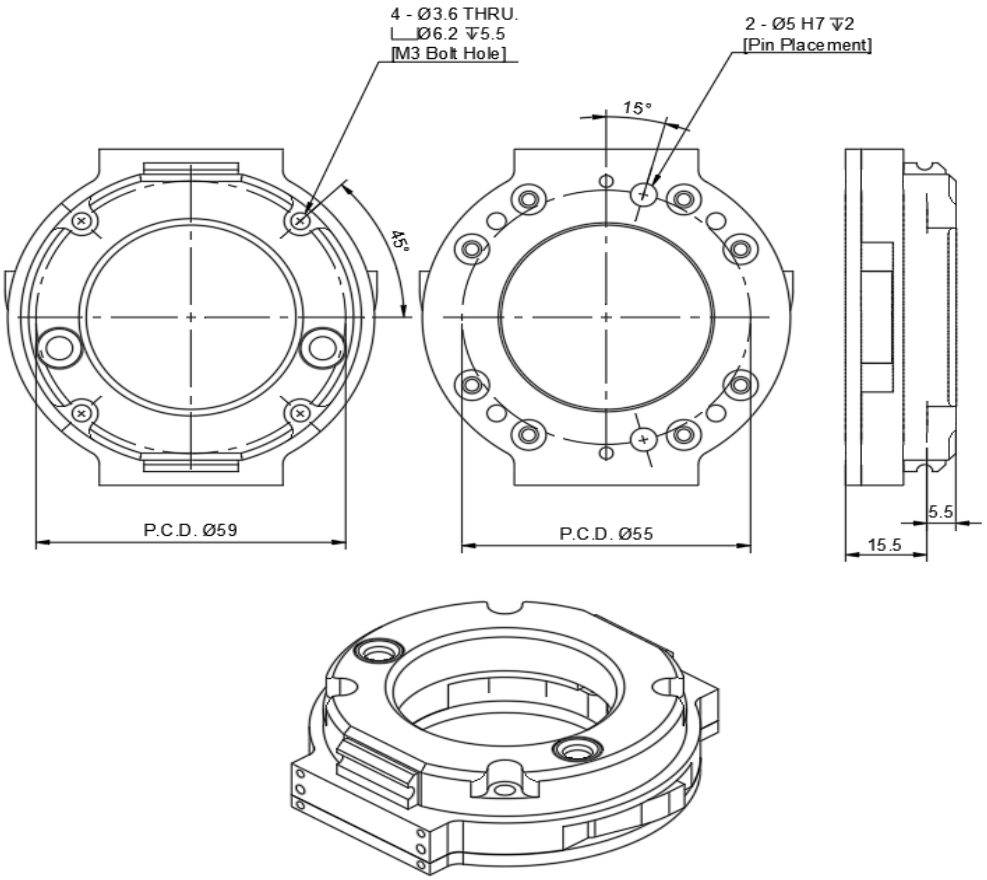



그림 3.14 ATC-10-T 도면

# 4. 그리퍼 제어 시스템 KR-GCS

코라스로보틱스의 그리퍼 제어 시스템(Gripper Control System)인 KR-GCS는 MATC 방식에서는 MATC 로봇측 유닛에 내장된 모터 제어기에 연결되고, 일체형 그리퍼 방식에서는 그리퍼에 내장된 모터 제어기에 연결된다. 로봇의 엔드 이펙터는 그리퍼 외에도 일반적인 툴이 사용될 수도 있지만, 주로 제어가 필요한 툴은 그리퍼이므로, 그리퍼 제어 시스템이란 용어를 사용한다. 그러나 동일한 시스템이 그리퍼 외에도 일반적인 툴에도 적용될 수 있다. 본 장에서는 KR-GCS의 통신 프로토콜과 그리퍼 제어 및 모터 제어를 위해서 필요한 명령어 등에 대해서 자세히 설명하기로 한다.

## 4.1 Modbus RTU 프로토콜

KR-GCS는 Modbus RTU 통신을 지원하며, 다음과 같은 통신 파라미터 및 프로토콜로 통신을 수행한다. 그림 4.1은 Modbus RTU의 데이터 프레임을 나타내며, 표 4.1은 통신 파라미터를 나타낸다. 표 4.2는 Modbus RTU의 프레임 포맷을 보여주는데, 각 프레임 간에는 적어도 3.5 characters 이상의 간격이 필요하다.

Modbus message

| Start | Address | Function | Data | CRC Check | End |
|---|---|---|---|---|---|
| 3.5 char (= 28 bits) | 8 bits | 8 bits | N × 8 bits | 16 bits | 3.5 char (= 28 bits) |

그림 4.1 Modbus RTU 데이터 프레임

표 4.1 Modbus RTU 통신 파라미터

| **Parameters** | **Default values** |
|---|---|
| Baud rate | 115,200 bps |
| Stop bit | 1 |
| Data bit | 8 |
| Parity | None |
| Slave ID | 1 (default) |
| Termination resistor (120Ω) | Yes |

표 4.2 Modbus RTU 프레임 포맷

| **Name** | **Length** (bits) | **Function** |
|---|---|---|
| Start | 28 | At least 3.5-character times of silence |
| Address | 8 | Slave address |
| Function | 8 | Function code (e.g., read coils / inputs) |
| Data | N × 8 | Data length is written depending on the message type |
| CRC | 16 | Error check |
| End | 28 | At least 3.5-character times of silence between frames |

Modbus buffer는 Discrete Inputs, Coils, Holding Registers 및 Input Registers로 구성된다. Modbus RTU

는 Holding registers를 이용한 통신을 수행한다.

1) Discrete Inputs (Read Only)

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - |

2) Coils (Read & Write)

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - |

3) Holding Registers (Read & Write)

| Address | Connection |
|---|---|
| 0 | o |
| 1 | o |
| 2 | o |
| … | … |

4) Input Registers (Read only)

| Address | Connection |
|---|---|
| 0 | - |
| 1 | - |
| 2 | - |
| … | … |

MATC의 Holding registers에 해당하는 주소(address)에 전달하고 싶은 명령(command)과 목표치값(target values)을 입력하여 MATC를 구동할 수 있다. 표 4.3과 같이 Holding Register의 Address 0번에 Command에 해당하는 값을 입력하고, Command에 따라 Target value를 필요로 하는 경우에 Address 1번에 목표치를 동시에 입력한다.

표 4.3 Modbus Holding Registers 0 ~ 9

| Address | Function (16 bits) |
|---|---|
| 0 | Command |
| 1 | Target value 1 |
| 2 | Target value 2 |
| 3 | … |
| 4 | … |
| 5 | … |
| 6 | … |
| 7 | … |
| 8 | … |
| 9 | … |

## 4.2 모터 제어

그리퍼 제어 시스템인 KR-GCS를 통하여 표 4.4에서 열거한 명령을 사용하여 모터 제어와 그리퍼 제어를 각각 수행할 수 있다. 모터 제어의 경우 위치 제어, 속도 제어 및 전류 제어가 가능하다. 대부분의 경우 사용자가 직접 모터를 제어할 필요는 없지만, 그리퍼 제어의 전체적인 동작의 이해를 돕기 위해서 간략히 설명하기로 한다. 모터 제어와 관련된 명령어는 표 4.4와 같다.

표 4.4 KR-GCS의 명령어 목록

| Command (Name) | Command (Decimal) | Value 1 (Decimal) | Value 2 (Decimal) |
|---|---|---|---|
| Motor Enable | 1 | x | x |
| Motor Stop | 2 | x | x |
| Motor Disable | 4 | x | x |
| Motor Position Control | 5 | Target degree (degree) | x |
| Motor Velocity Control | 6 | Target velocity (rpm) | x |
| Motor Current Control | 7 | Target current (mA) | x |
| Change Modbus Address | 50 | Desired address | x |
| Gripper Initialize | 101 | x | x |
| Gripper Open | 102 | x | x |
| Gripper Close | 103 | x | x |
| Set Finger Position | 104 | 0 ~ 1000<br>(0: closed & 1000: open) | x |
| Vacuum Gripper On | 106 | x | x |
| Vacuum Gripper Off | 107 | x | x |
| Impedance Control On | 108 | x | x |
| Impedance Control Off | 109 | x | x |
| Set Impedance Parameters | 110 | Gripper type (1 ~ 20) | Stiffness level (1 ~ 10)<br>(1: lowest stiffness, ...<br>10: highest stiffness) |
| Set Motor Torque | 212 | Ratio of target motor torque to default torque (%) | x |
| Set Motor Speed | 213 | Ratio of target motor speed to default speed (%) | x |

**Motor Enable**: 모터를 활성화하는 동시에 이 명령이 실행되는 시점의 모터 회전각이 유지되도록 위치 제어를 자동으로 실행한다.

**Motor Stop**: 모터의 위치 제어를 수행하여 모터가 이 명령이 실행되는 시점의 위치에서 정지하도록 제어한다.

**Motor Disable**: 모터에 더 이상 전원을 공급하지 않아서 모터를 비활성화한다.

**Motor Position Control**: 모터의 위치 제어를 통해서 Target degree에 해당하는 위치에 도달하도록 한다.

**Motor Velocity Control**: 모터의 속도 제어를 통해서 Target velocity에 도달하도록 한다.

**Motor Current Control**: 모터의 전류 제어를 통해서 Target current에 도달하도록 한다.

**Change Modbus Address**: Modbus RTU 통신의 slave address를 변경하고자 할 때 원하는 값으로 변경한 후에 제어 시스템의 전원을 껐다 켜면 변경된 slave address가 적용되도록 한다.

## 4.3 그리퍼 제어

그리퍼 제어와 관련된 명령어는 표 4.4와 같다. 이 명령들은 사용자가 그리퍼를 제어하기 위해서 필요하므로, 정확한 이해가 필요하다.

**Gripper Initialize**: 모터를 구동하여 MATC용 그리퍼 또는 일체형 그리퍼의 종류 및 최대 행정(stroke)의 크기를 확인하는 데 사용된다. 여기서 최대 행정이란 2지 그리퍼의 경우에 그리퍼가 최대로 열렸을 때 두 핑거 간의 최대 거리를 의미한다.

MATC 방식에서는 여러 종류의 그리퍼가 장착될 수 있는데, 각 그리퍼는 순수한 기계 부품으로만 구성되어 있으므로, MATC는 장착된 그리퍼의 종류를 알 수 없다. 그러므로 그리퍼를 장착한 후 최초 구동 시에는 반드시 Initialize를 수행하여 MATC가 그리퍼를 구동하여 특성을 조사함으로써, 장착된 그리퍼의 종류를 파악하고, 최대로 개폐되었을 때의 그리퍼의 핑거 위치 등을 파악하도록 한다.

일체형 그리퍼 방식에서는 그리퍼의 종류와 최대 행정의 크기를 미리 알 수 있지만, MATC 방식과 유사하게 처음 그리퍼 시스템을 켤 때 Initialize를 수행하여 최대 행정의 크기를 파악한다. 이 경우에는 그리퍼의 교체가 없으므로 그리퍼 시스템이 꺼질 때까지 이 파라미터가 계속 사용된다.

**Gripper Open & Gripper Close**: Gripper Initialize를 수행한 후에 동작할 수 있는데, Gripper Open은 그리퍼를 열고, Gripper Close는 그리퍼를 닫는 기능을 수행한다. Gripper Open 또는 Gripper Close를 수행하는 동안 핑거가 물체와 접촉하여 더 이상 움직이지 못하는 상태로 일정 시간이 지나면 그 위치에서 정지하도록 제어한다. 그리퍼의 종류에 따라서 그리퍼가 닫히는 중에 물체를 파지(예, 2지 그리퍼)할 수도 있고, 그리퍼가 열리는 중에 물체를 파지(예, 평행 그리퍼)할 수도 있다.

**Set Finger Position**: 완전히 닫힌 위치(0)와 완전히 열린 위치(1000) 사이에서 그리퍼의 핑거의 위치 설정할 때 사용한다. 통신에서는 정수만 허용되므로, 0 ~ 1000을 사용하지만, 실제 설정에서는 이는 0% ~ 100.0%를 의미한다. 즉, 전체 개폐 범위를 0.1% 단위로 핑거의 위치 제어를 수행할 수 있다. 예를 들어, value 1에 입력한 값이 505에서 506으로 증가하면 실제로는 핑거 위치가 50.5%에서 50.6%로 증가하므로 그리퍼 핑거는 0.1%만큼 더 열리게 된다.

**Set Motor Torque**: 모터의 발생 토크를 조절하는데, 이를 통해서 그리퍼의 파지력의 조정이 가능하다. 기본적으로는 모터의 정격 토크가 디폴트(default) 토크로 설정되어 있지만, 사용자는 이 디폴트 토크를 100%로 하였을 때 50 – 100%의 범위에서 목표 토크(target torque)를 설정할 수 있다. 예를 들어, 목표 토크를 80%로 입력하면 디폴트 토크 대비 80%에 해당하는 설정 토크에 해당하는 파지력으로 물체를 파지하게 된다. 이때 모터에 공급되는 전류가 정격 전류일 때는 디폴트 토크가 정격 토크에 해당하지만, 공급 전류가 정격 전류에 미치지 못하는 경우에는 디폴트 토크가 정격 토크보다 작아진다는 점에 유의하여야 한다.

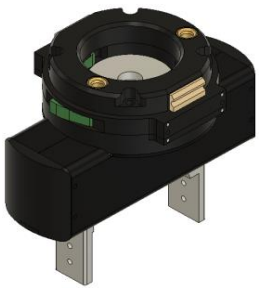
**Set Motor Speed**: 모터의 위치 제어 시에 생성하는 모터의 속도 프로파일(사다리꼴 프로파일)에서 정속 구간에 해당하는 모터 속도를 설정하는 데 사용된다. MATC-10에서는 모터의 정격 속도가 900 rpm이지만, MATC 또는 일체형 그리퍼의 위치 제어 시에 기본적으로 정격 속도의 75% (675 rpm)가 디폴트 속도로 설정되어 있다. 사용자의 필요에 따라 이 속도를 1 – 100% (MATC-10에서는 900 rpm)의 범위 내에서 조절할 수 있다. MATC-10의 예를 들면, 목표 속도를 100%로 설정하면 모터의 정속 구간 속도는 900 rpm이 되어 디폴트 속도(675 rpm)보다 빠르게 회전하고, 목표 속도를 20%로 설정하면 180 rpm이 되어 디폴트 속도보다 느리게 움직이게 된다.

**Vacuum Gripper On & Vacuum Gripper Off**: 이 명령을 통해서 진공 그리퍼를 동작시킬 수 있다. Vacuum Gripper On은 모터의 회전을 통해서 진공 그리퍼의 펌프를 구동하고, Vacuum Gripper Off는 모터의 회전을 정지시켜서 펌프의 구동을 멈출 수 있게 한다.

**Impedance Control On & Impedance Control Off**: 이들 명령을 보낸 뒤에 Gripper Initialize 명령을 수행하면, Impedance Control On은 그리퍼의 임피던스 제어를 활성화시키고, Impedance Control Off는 비활성화한다. 임피던스 제어 모드를 활성화하면 그리퍼의 핑거와 파지하는 물체 사이에 가상의 스프링을 장착하는 효과를 내므로, 스프링의 강성을 조절하여 물체를 파지하는 강도를 조절할 수 있다. 예를 들어, 스프링의 강성을 낮게 설정하면, 동일한 파지력으로 물체를 파지하더라도 가상 스프링의 역할로 파지 물체가 변형되는 것을 최소화할 수 있다. 그러므로 소프트 그리퍼로 물체를 파지하는 효과를 낼 수 있으므로, 변형이 쉽게 되는 물체를 파지할 때 매우 유용하다.

**Set Impedance Parameters**: 임피던스 제어 모드를 적용할 그리퍼 타입을 결정하고, 임피던스 파라미터 중에서 강성을 조정한다. 임피던스 제어를 활성화시킨 후에는 반드시 그리퍼 타입과 강성 레벨을 입력해 주어야 한다. 임피던스 제어는 표 4.5와 같이 정해진 몇 가지 그리퍼 타입에 대해서만 안정적으로 동작한다. 사용하고자 하는 그리퍼 타입에 맞는 번호를 첫 번째 입력(표 4.4의 Value 1)으로 지정하고, 임피던스 제어의 강성 단계(stiffness level)를 두 번째 입력(표 4.4의 Value 2)으로 결정한다. 여기서 강성 단계는 1 ~ 10 단계로 구성되며, 1단계(10단계)는 가장 낮은(높은) 강성을 설정하여 물체를 가장 약하게(강하게) 파지할 수 있다. 단계가 증가할수록 강성이 높아지며, 강성이 높아질수록 물체를 더 강하게 파지하게 된다. 임피던스 제어를 수행하는 그리퍼 타입과 다른 번호를 입력할 경우 임피던스 제어가 제대로 동작하지 않고, 파지 물체에 손상을 줄 수 있으므로, 각별히 주의하여야 한다.

표 4.5 임피던스 제어를 지원하는 그리퍼 타입

| Gripper type | Value 1<br>(Gripper type) | Value 2<br>(Stiffness level) |
|---|---|---|
|  <br><2FPA-M-03-035/080>[1)] | 1 | 1 ~ 10 |
|  <br><2FPA-M-03-035/080-CO (Compliance version)>[2)] | 2 | 1 ~ 10 |

| | | |
|---|---|---|
|  <br><2FST-M-04-100-HS (High-speed version)>[3)] | 3 | 1 ~ 10 |

1) 평행 그리퍼에서 감속비가 낮으면 역구동성이 높아서 임피던스 효과가 더 잘 구현된다.
2) 평행 그리퍼에서 감속기가 없는 compliance version (2FPA-M-03-035/080-CO )이 역구동성이 가장 높아서 임피던스 효과가 가장 잘 구현된다.
3) 고속형이 아닌 일반 2FST-M-04-100 그리퍼는 높은 감속비로 인하여 역구동성이 낮아서 임피던스 효과가 거의 나타나지 않으므로 임피던스 제어 적용의 대상이 아니다.

## 4.4 제어 상태 표시

표 4.6의 Address 10은 그리퍼 제어 상태를 모니터링하기 위해서 사용된다. Status의 각 비트에 상태가 표시되는데, 각 상태에 대해서는 표 4.7을 참고한다. 비트 0는 Motor Enable이 수행되고 있는지, 비트 1은 Gripper Initialize가 수행되고 있는지를 나타낸다. 예를 들어, Gripper Initialize가 진행되는 동안에는 비트 1이 1로 표시되지만, 이 작업이 종료되면 0으로 표시된다. 비트 2 - 4는 모터의 위치 제어, 속도제어, 전류 제어가 각각 수행 중인지를 나타낸다. 예를 들어, Position Control이 수행하는 동안에는 비트 2가 1로 표시되지만, 위치 제어가 종료되면 비트 2가 0으로 표시된다. 비트 5, 6은 그리퍼 명령 중 Gripper Open과 Gripper Close 명령이 진행 중인지 나타낸다. 마지막으로, 비트 9는 과전류 또는 과전압 등의 모터 오류가 발생했는지를 나타낸다.

표 4.6 Modbus Holding Registers 10 ~ 17

| Address | Read Value |
|---|---|
| 10 | Status |
| 11 | Motor Position<br>(absolute degree, 16 bit) |
| 12 | Motor Current (mA) |
| 13 | Motor Velocity (rpm) |
| 14 | Gripper Finger Position<br>(0: closed ~ 1000: open) |
| 15 | - |
| 16 | - |
| 17 | Bus Voltage (V) |

표 4.7 Status of Address 10

| Bit | Status | Value |
|---|---|---|
| 0 | Motor Enable | 0: False, 1: True |
| 1 | Gripper Initialize | 0: False, 1: True |

| | | |
|---|---|---|
| 2 | Motor Position Control | 0: False, 1: True |
| 3 | Motor Velocity Control | 0: False, 1: True |
| 4 | Motor Current Control | 0: False, 1: True |
| 5 | Gripper Open | 0: False, 1: True |
| 6 | Gripper Close | 0: False, 1: True |
| 7 | Not used | - |
| 8 | Not used | - |
| 9 | Motor Fault | 0: False, 1: True |
| 10 - 15 | Not used | |

## 4.5 외부 버튼을 통한 그리퍼 제어

MATC의 측면 또는 일체형 그리퍼의 몸체에는 내부 모터를 외부에서 간단하게 조작하기 위한 버튼이 위치해 있다. 버튼은 Initialize (I), Open (O), Close (C) 총 3개의 버튼으로 구성되어 있다. MATC 방식에서는 그리퍼와 처음 체결하게 되면 그리퍼의 종류 및 상태(즉, 그리퍼 핑거의 위치 등)를 알 수 없으므로, 일단 핑거를 완전히 열었다가 닫아서 작업을 위한 초기 위치를 설정한다. 이 과정에서 그리퍼의 최대 스트로크도 알게 되어, 이후 그리퍼 제어에 필요한 정보로 활용된다. 일체형 그리퍼에서는 그리퍼의 종류는 미리 알고 있지만, 처음 구동 시에 그리퍼 핑거의 위치는 알지 못하므로, 역시 핑거를 완전히 열었다가 닫아서 작업을 위한 초기 설정을 한다.

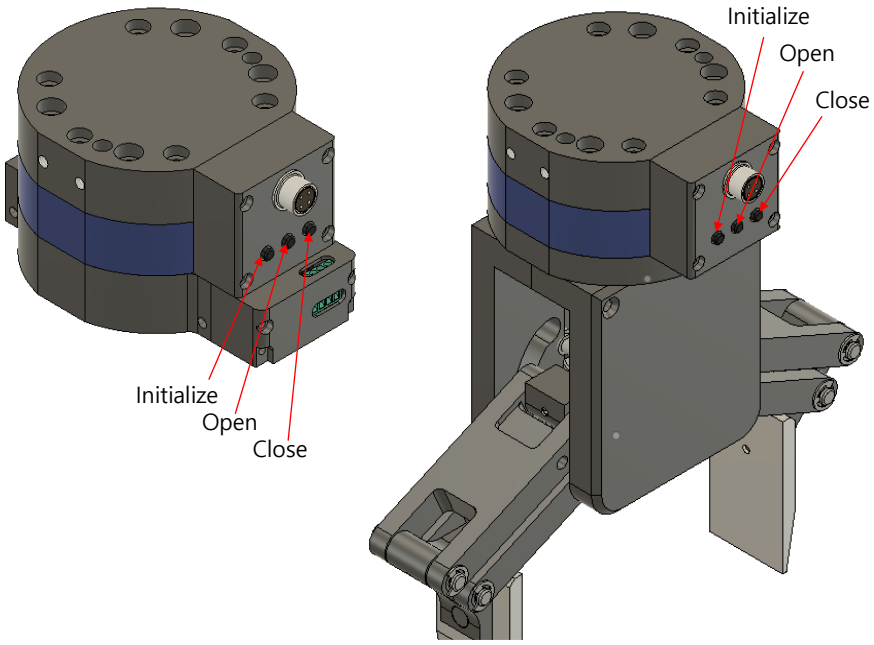



그림 4.2 외부 버튼 구성: MATC의 외부 버튼 (좌측) 및 일체형 툴의 외부 버튼 (우측)

각 버튼은 다음과 같은 기능을 수행한다.

**Initialize**: 해당 버튼 동작 시에 내부의 모터가 구동되어 MATC에 장착된 그리퍼의 종류 및 최대 스트로크를 확인한다. 그리퍼의 초기 구동 시에는 반드시 해당 버튼의 구동이 필요하며, 샘플 프로그램을 통해 이미 Initialize를 진행하였을 경우에는 해당 버튼 구동은 생략이 가능하다. 그리퍼의 종류에 따라 열림 및 닫힘의 순서가 반대일 수 있으며, 주로 최대 스트로크까지 열린 이후 반대로 구동하여 닫힌 상태를 유지한다.

**Open**: 해당 버튼을 누르고 있는 동안, 내부의 모터가 회전하여 장착된 그리퍼의 핑거를 여는 방향으로 구동한다. 앞서 설명한 Initialize에 의해 확인된 스트로크 내에서 구동 가능하며, 그리퍼의 종류에 따라 열림 및 닫힘이 반대일 수 있다.

**Close**: 해당 버튼을 누르고 있는 동안, 내부의 모터가 회전하여 장착된 툴의 핑거를 닫는 방향으로 구동한다. 앞서 설명한 Initialize에 의해 확인된 스트로크 내에서 구동 가능하며, 그리퍼의 종류에 따라 열림 및 닫힘이 반대일 수 있다.

# 5. 소프트웨어 설치 및 사용법

사용자가 MATC 또는 일체형 그리퍼를 쉽게 사용할 수 있도록 다음과 같은 소프트웨어 및 예제 코드를 제공한다.

1. TCP Socket 통신 기반의 KR-GCS
2. ROS 2 기반의 KR-GCS
3. C++ 예제 코드

## 5.1 TCP Socket 통신 기반의 KR-GCS

### 5.1.1 설치 방법

TCP Socket 통신 기반의 KR-GCS를 사용하기 위해 필요한 시스템 요구 사항은 다음과 같다.

- MATC 또는 일체형 그리퍼
- 운영 체제(OS): Windows 10/11 또는 Linux Ubuntu 22.04
- 설치 용량: 최소 100MB의 디스크 공간 필요

MATC 또는 일체형 그리퍼는 외부전원 공급원으로부터 전원을 공급받아야 하며, 3.3절과 같이 인터페이스 케이블을 통해서 PC나 로봇에 연결되어야 한다.

각 OS별 KR-GCS 설치 및 실행 방법은 다음과 같다.

**Window 10/11 사용자**

1. https://github.com/KorasRobotics/KR_GCS_user_interface에 접속하여 Release 탭에서 압축 파일 "KR_GCS_user_interface_window.7z"을 다운로드 한다.
2. "KR_GCS_user_interface_window.7z" 을 압축 해제한 후, 폴더 내의 .exe 파일 "kr_gcs_user_interface.exe"를 실행한다.
3. 프로그램 정상 동작 시, 그림 5.1과 같은 화면이 나타난다.

**Ubuntu 22.04 사용자**

1. https://github.com/KorasRobotics/KR_GCS_user_interface에 접속하여 Release 탭에서 압축 파일 "KR_GCS_user_interface_ubuntu.7z"을 다운로드 한다.
2. 연결된 MATC 또는 일체형 그리퍼를 사용하기 위해, 터미널을 실행한 후 다음과 같은 명령어를 입력하여 실행 권한을 설정한다. 이때, 실행 권한 및 포트 명은 환경에 따라 다를 수 있다.
   'sudo chmod 666 /dev/ttyUSB<port number>'
3. "KR_GCS_user_interface_ubuntu.7z"을 압축 해제 후,
   폴더 내의 bash 파일 "kr_gcs_user_interface.sh"를 실행한다.
4. 프로그램 정상 동작 시, 그림 5.1과 같은 화면이 나타난다.

### 5.1.2 사용 방법

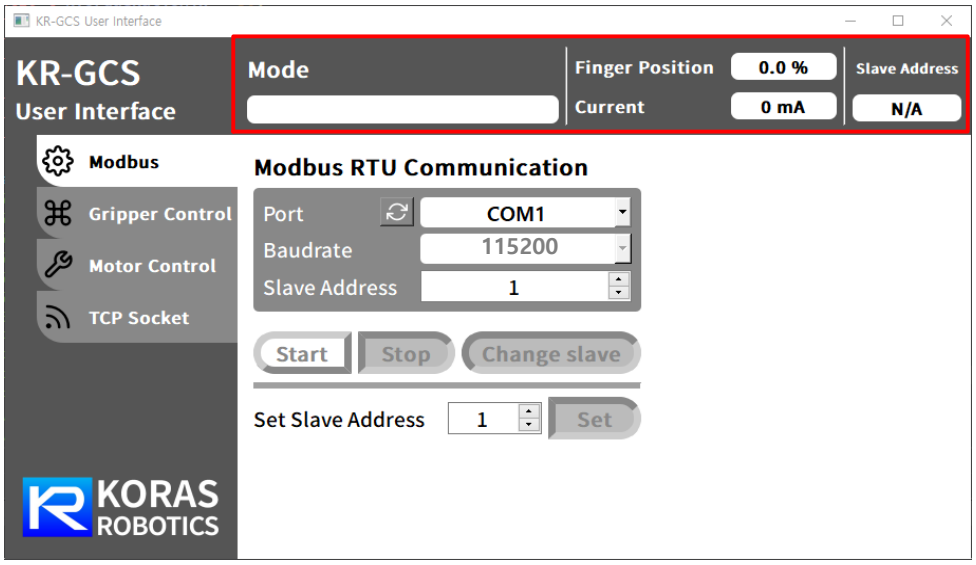



그림 5.1 KR-GCS UI 실행 화면

그림 5.1의 UI의 상단은 KR-GCS의 상태를 나타내며, 각 구성은 다음과 같다.

**Mode**: 현재의 모터 제어 모드(예: 위치제어, 속도제어, 토크제어 등)를 표기한다.

**Finger Position**: 그리퍼의 현재 위치를 % 단위로 나타낸다. 그리퍼가 완전히 닫힌 위치가 0%, 완전히 열린 위치가 100%이다.

**Current**: 모터에 흐르는 전류를 mA 단위로 나타낸다.

**Slave Address**: 현재 연결된 Modbus 통신의 슬레이브 주소를 나타낸다.

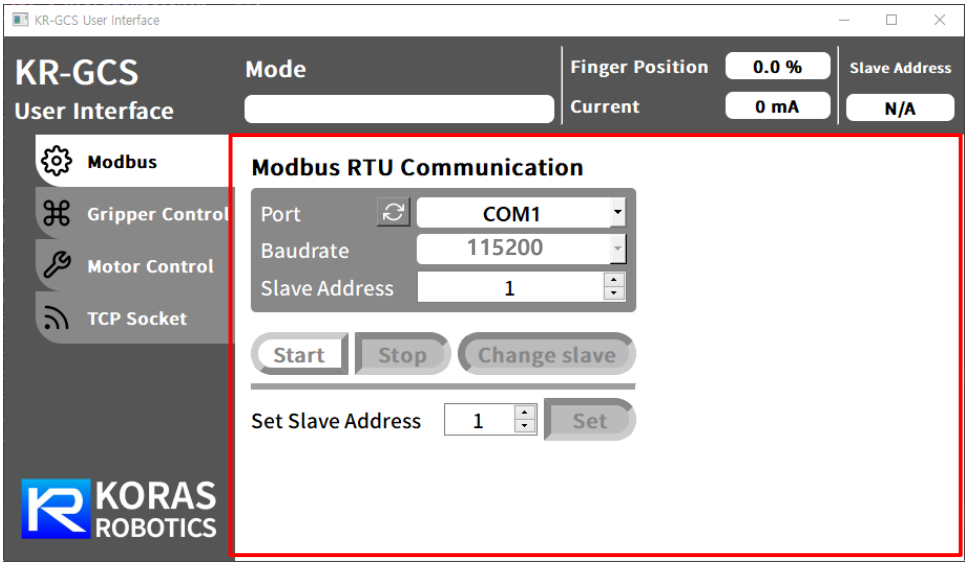



그림 5.2 KR-GCS UI의 Modbus 탭

그림 5.2의 UI의 Modbus 탭은 Modbus 통신 설정을 위해 다음과 같은 기능을 제공한다.

**Port**: MATC 또는 일체형 그리퍼와 PC 간의 통신을 위한 포트를 지정한다. KR-GCS가 설치된 PC에 MATC 또는 일체형 그리퍼를 연결한 후, Port 옆의 "새로고침" 버튼을 클릭하면 연결 가능한 포트의 후보를 자동으로 감지하여 표시한다. 이때, Window 10에서는 COM으로 시작되는 포트가 자동 검색되며, Ubuntu

에서는 "/dev/ttyUSB"로 시작되는 포트만 검색된다.

**Baudrate**: Modbus 통신 속도를 입력하며, 기본 값은 115,200 bps이다.

**Slave Address**: 연결을 희망하는 MATC 또는 일체형 그리퍼의 Modbus 통신의 슬레이브 주소를 입력한다. 이는 단일의 MATC를 사용하는 경우에는 필요 없다. 한 MATC에 여러 개의 그리퍼들이 교체되면서 사용되더라도, 여전히 단일 MATC를 사용한다는 점에 유의한다. 만약 아래의 "Set Slave Address" 명령에 의해서 복수 개의 MATC가 설정되었을 때, 이 중에서 희망하는 MATC에 연결하기 위해서는 해당 "Slave Address"를 입력하고, 다음의 "Change slave"를 수행하면 된다.

**Start** & **Stop**: MATC 또는 일체형 그리퍼와의 Modbus 통신을 시작하거나 중지한다.

**Change slave**: Modbus 통신의 슬레이브 주소를 "Slave Address"에 적힌 주소로 변경한다. "Slave address" 필드에서 원하는 슬레이브 어드레스를 지정한 후에, "Change slave" 버튼을 클릭하면 된다. 이때 Start 버튼을 클릭하여야 슬레이브 어드레스가 실제로 변경된다. 이 기능을 통해 동일한 포트에 연결되는 여러 MATC 또는 일체형 그리퍼 중에서 원하는 기기와 연결할 수 있다.

**Set Slave Address**: MATC 또는 일체형 그리퍼의 슬레이브 주소를 변경한다. 예를 들어, "Set Slave Address"에 2를 입력한 뒤 Set 버튼을 누르게 되면, 연결된 MATC 또는 일체형 그리퍼는 전원이 꺼진 뒤 재시작 시부터 슬레이브 주소가 2로 변경된다. 이 명령은 단일 MATC만을 사용할 때는 필요 없다. 예를 들어, MATC 1과 MATC 2의 2개의 MATC를 사용하고자 한다면, MATC 1과 2 모두 출고 시에 Slave Address는 디폴트로 1로 설정되어 있으므로, 이 중에서 한 MATC의 주소를 다른 값(예, 2)으로 변경해 주어야 하는데, 이때 Set Slave Address를 사용하여 주소를 변경하여 주면 된다. 따라서 이 작업은 위의 "Change slave" 명령에 앞서 초기 과정에서 먼저 수행되어야 한다. 이렇게 특정 MATC의 슬레이브 주소를 변경해 주면, 이 MATC는 계속 변경된 주소를 유지하므로, 처음 제품 구입 시의 디폴트인 주소 1이 아니라는 점에 유의하여야 한다.

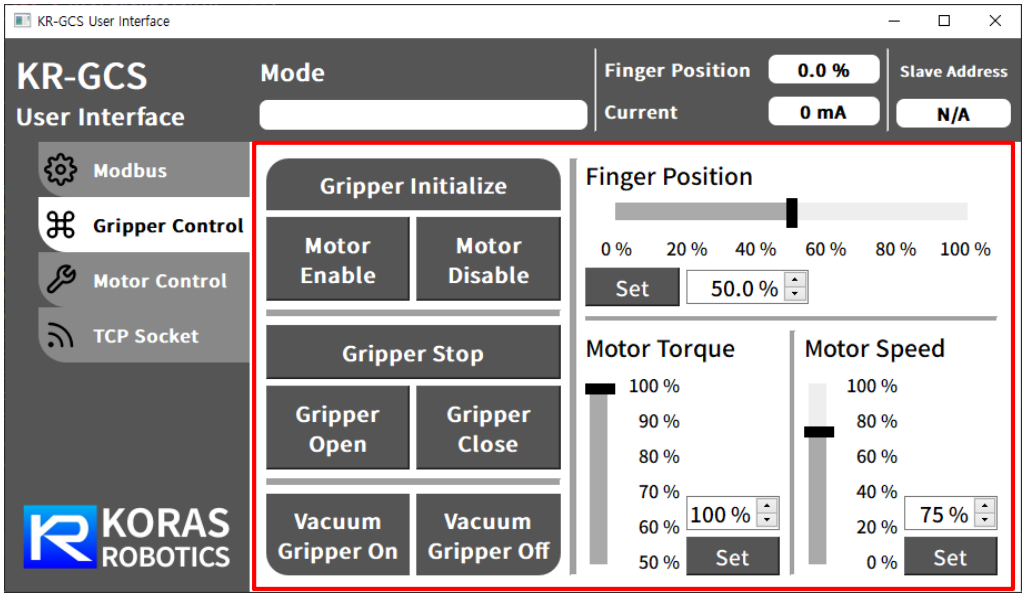



그림 5.3 KR-GCS UI의 Gripper Control 탭

그림 5.3의 UI의 "**Gripper Control**" 탭은 4.2절에서 설명한 모터 제어 및 4.3절에서 설명한 그리퍼 제어 기능을 제공한다.

**Gripper Initialize**: 4.3절의 "Gripper Initialize" 명령을 수행한다.

**Motor Enable & Motor Disable**: 4.2절의 "Motor Enable" & "Motor Disable" 명령을 수행한다.

**Gripper Stop**: 4.2절의 Motor Stop 명령과 동일한다. 모터가 정지하면 당연히 그리퍼도 정지하므로, Gripper Stop 명령은 실제로는 Motor Stop 명령을 수행하는 것이다.

**Gripper Open & Gripper Close**: 4.3절의 "Gripper Open" & "Gripper Close" 명령을 수행한다.

**Vacuum Gripper On & Vacuum Gripper Off**: 4.3절의 "Vacuum Gripper On" & "Vacuum Gripper Off" 명령을 수행한다.

**Finger Position**: 4.3절의 "Set Finger Position" 명령을 수행한다. 슬라이더를 움직이거나 "Set" 버튼 옆의 숫자를 수정하는 것으로 목표 위치를 설정할 수 있다. 이때, 슬라이더와 "Set" 버튼 옆의 숫자는 실시간으로 동기화되므로, 둘 중 하나만 조절하면 된다. "Set" 버튼을 누르는 것으로 제어를 수행한다.

**Motor Torque & Motor Speed**: 4.3절의 "Set Motor Torque" & "Set Motor Speed" 명령을 수행한다. 슬라이더를 움직이거나 "Set" 버튼 옆의 숫자를 수정한 뒤"Set" 버튼을 누르는 것으로 토크와 속도를 설정할 수 있다. 이때, 슬라이더와 "Set" 버튼 옆의 숫자는 실시간으로 동기화되므로, 둘 중 하나만 조절하면 된다.

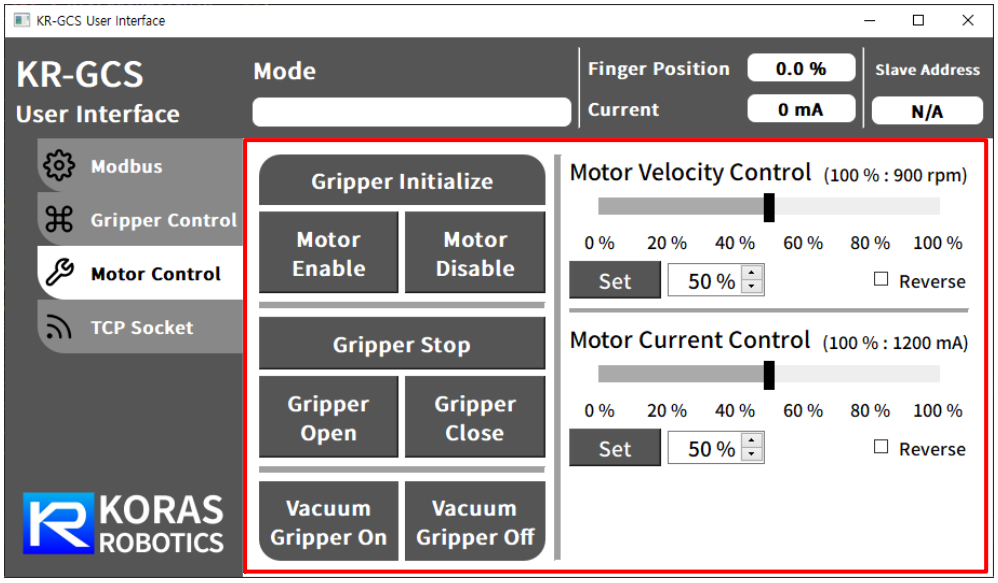



그림 5.4 KR-GCS UI의 Motor Control 탭

그림 5.4의 UI의 "**Motor Control**" 탭은 다음과 같은 모터 제어 기능을 제공하며, 탭 좌측의 버튼의 역할은 "Gripper Control" 탭의 버튼과 동일하다.

**Motor Velocity Control**: 4.2절의 "Motor Velocity Control" 명령을 수행한다. 슬라이더를 움직이거나 "Set" 버튼 옆의 숫자를 수정하는 것으로 목표 속도의 크기를 설정할 수 있으며, 목표 속도의 부호, 즉 회전 방향은 우측 하단의 "Reverse" 체크 박스를 통해 변경할 수 있다. 이때, 슬라이더와 "Set" 버튼 옆의 숫자는 실시간으로 동기화되므로, 둘 중 하나만 조절하면 된다. "Set" 버튼을 누르면 제어를 수행한다.

**Motor Current Control**: 4.2절의 "Motor Current Control" 명령을 수행한다. 슬라이더를 움직이거나 "Set" 버튼 옆의 숫자를 수정하는 것으로 목표 전류의 크기를 설정할 수 있으며, 목표 전류의 부호, 즉 전류가 흐르는 방향은 우측 하단의 "Reverse" 체크 박스를 통해 변경할 수 있다. 이때, 슬라이더와 "Set" 버튼 옆의 숫자는 실시간으로 동기화되므로, 둘 중 하나만 조절하면 된다. "Set" 버튼을 누르면 제어를 수행한다.

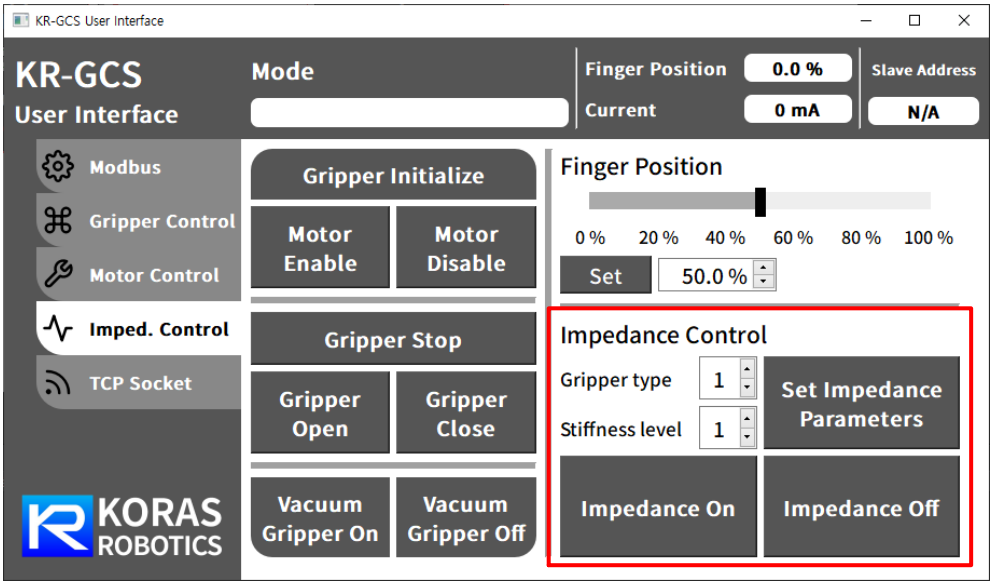



그림 5.5 KR-GCS UI의 Impedance Control 탭

그림 5.5의 UI의 "**Impedance Control**" 탭은 다음과 같은 임피던스 제어 기능을 제공하며, 적색으로 표시된 영역을 제외한 다른 기능 및 UI는 그림 5.3 KR-GCS UI의 "Gripper Control" 탭과 동일하다.

**Impedance On & Impedance Off**: 4.3절의 "Impedance Control On" & "Impedance Control Off" 명령을 수행한 뒤, "Gripper Initialize" 명령을 수행한다. 이를 통해 그리퍼의 임피던스 제어 모드를 활성화/비활성화할 수 있다.

**Set Impedance Parameters**: 4.3절의 "Set Impedance Parameters" 명령을 수행한다. 버튼 좌측의 "Gripper type"을 표 4.5를 참고하여 설정하고, 원하는 강성 단계에 따라 1 ~ 10 사이의 숫자를 "Stiffness level"에 설정한다. "Set Impedance Parameters" 버튼을 누르면 파라미터를 변경한다.

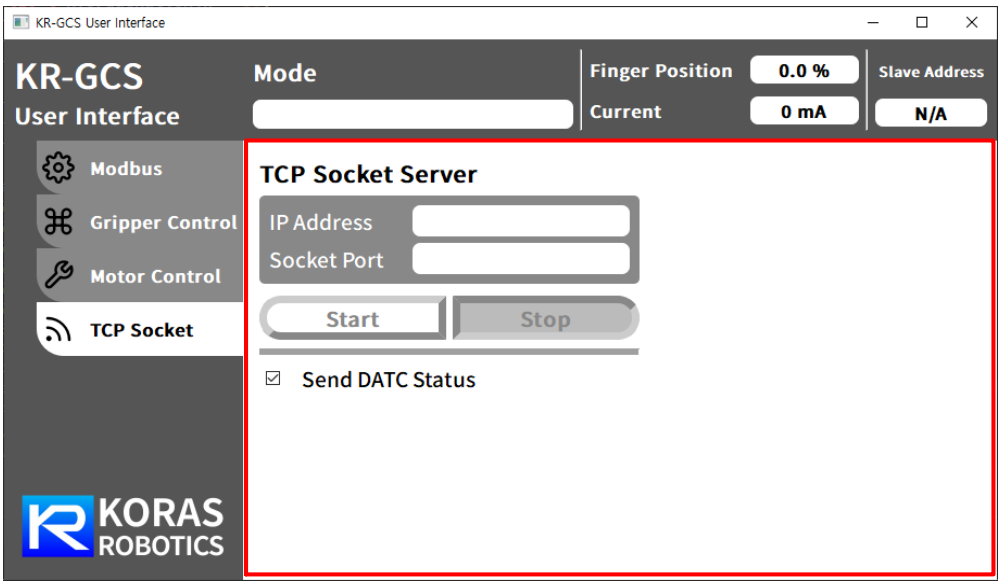



그림 5.6 KR-GCS UI의 TCP Socket 탭

그림 5.6의 UI의 TCP Socket 탭에서는 TCP Socket 통신 설정을 위한 다음과 같은 기능을 제공한다.

**IP Addreess**: MATC 또는 일체형 그리퍼가 연결될 네트워크의 IP 주소를 입력한다. 이 주소를 통해 MATC

또는 일체형 그리퍼 장치는 TCP socket 통신으로 데이터를 송수신할 수 있다.

**Socket Port**: TCP Socket 통신에 사용될 소켓 포트 번호를 입력한다.

### 5.1.3 Modbus 통신 예제

본 절에서는 KR-GCS UI를 통해 PC와 MATC 또는 일체형 그리퍼를 연결하는 Modbus 통신 예제를 제공한다. 운영 체제에 관계없이 사용법은 동일하며, Modbus 통신에 대해서는 그림 5.2을 참고한다.

1. UI를 실행시킨 후, 그림 5.2의 "Modbus" 탭을 선택한다.
2. "Port" 목록에서 MATC 또는 일체형 그리퍼가 연결된 USB 포트를 선택한다. 새로고침 버튼 클릭 시에 연결 가능한 포트 목록을 갱신한다. 환경에 따라 새로고침 버튼 클릭 시에 자동으로 나타나는 포트가 MATC 또는 일체형 그리퍼와 연결된 포트가 아닐 수 있음에 유의한다.
3. "Baudrate"를 설정한다. 소프트웨어 버전에 따라서 고정된 Baudrate를 사용해야 할 수도 있다. 권장되는 Baudrate는 115,200 bps이다.
4. "Slave Address"에 MATC 또는 일체형 그리퍼의 Modbus 주소를 입력하는데, 기본 주소는 1이다.
5. 통신 설정이 완료되면 "Start" 버튼을 클릭하여 Modbus 통신을 시작할 수 있다. 정상적인 동작 시에 그림 5.7과 같이 KR-GCS UI의 상단에 그리퍼 상태가 변하는 것을 확인할 수 있다.
6. "Stop" 버튼을 클릭하여 Modbus 통신을 종료한다.

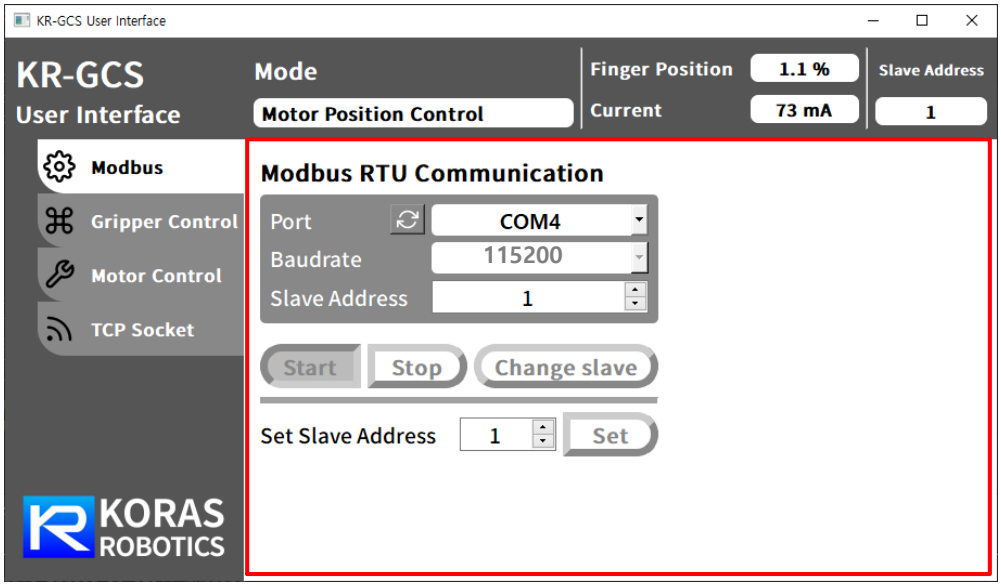



그림 5.7 Modbus 통신 예제 사진.

### 5.1.4 TCP Socket 통신 예제

본 절에서는 telnet 소프트웨어를 이용하여, TCP Socket 통신을 테스트하는 예제를 제공한다. TCP Socket 통신에 대해서는 그림 5.5를 참고한다.

UI를 실행하고 Modbus 통신을 연결한 후, 그림 5.6의 'TCP Socket' 탭으로 이동한다. IP Address와 Socket Port에 포트 번호를 설정한 후에, Start 버튼을 누르면 Scoket 통신이 시작된다. 본 예제에서는 그림 5.8과 같이 Socket Port로 1234를 사용한다. IP Address는 입력이 없을 시에는 "localhost"가 디폴트로 설정된다.

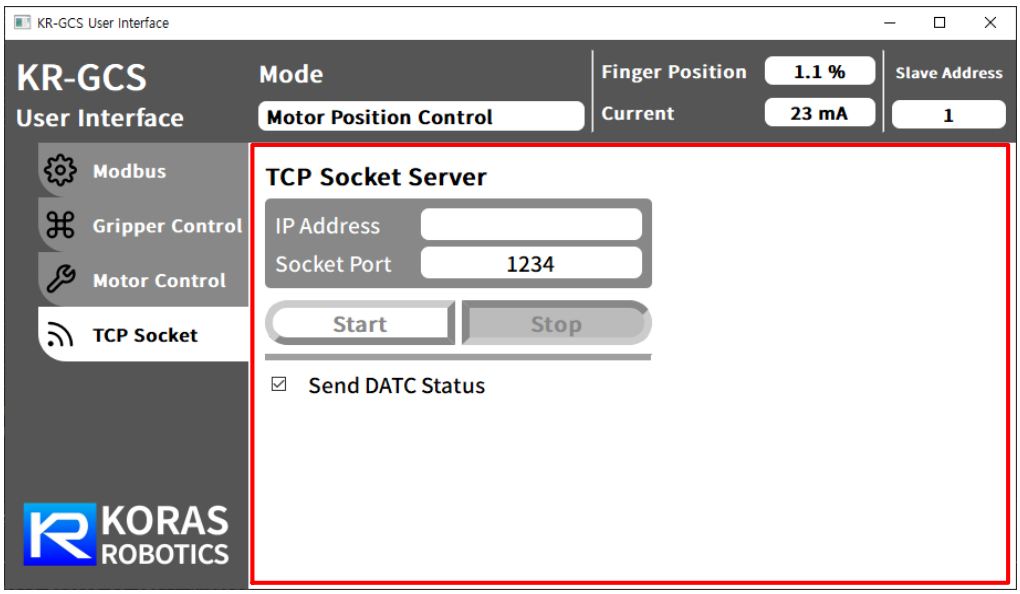



그림 5.8 TCP Socket 통신 예제

Windows의 명령 프롬프트 창 또는 Ubuntu의 터미널 창에서 그림 5.9와 같이 IP Address는 "localhost"로, Socket Port는 1234로 telnet을 실행한다.

**telnet localhost 1234**

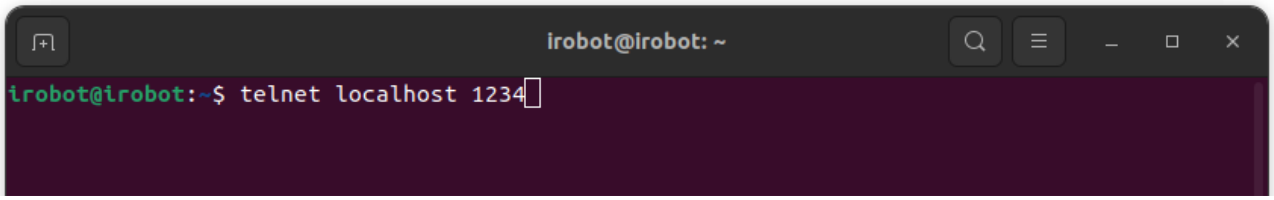



그림 5.9 telnet을 통한 TCP Socket 서버 접속 방법

그림 5.8의 UI에서 Send MATC Status를 체크하면, Socket 서버에서 그리퍼의 상태를 전송한다. telnet이 실행되는 창을 확인하면, 그림 5.10과 같이 그리퍼의 핑거 위치, 모터 전류, 모터 위치, 모터 속도, 상태, 전압 등이 실시간으로 표시된다. 이때, 각 변수와 값은 소프트웨어 버전에 따라 다를 수 있으므로, 소프트웨어를 배포하는 Github Repository에서 확인하여야 한다.

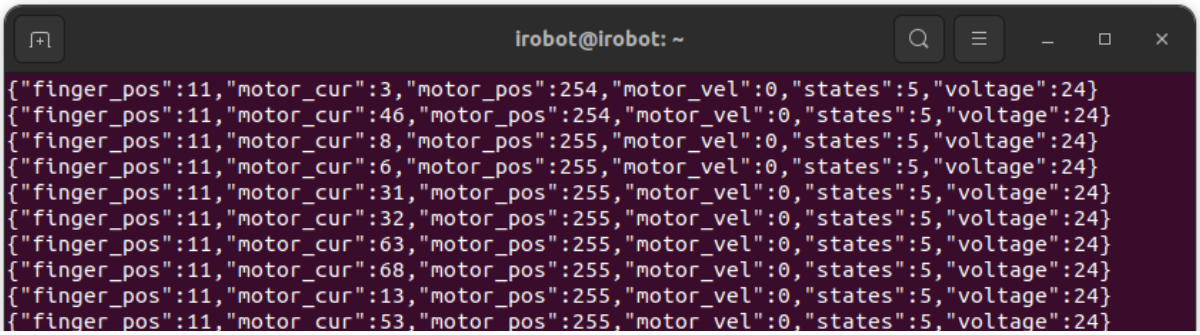



그림 5.10 TCP Socket 서버에서 보내는 그리퍼 상태

telnet으로 JSON 형식의 명령을 입력하여 그리퍼 제어 명령을 보낼 수 있다. 그림 5.8에서 Send MATC

Status를 해제한 후, 그림 5.11과 같이 명령 프롬프트에 명령어를 입력한다. 이때, Send MATC Status는 해제하지 않아도 무방하나, 명령어를 입력할 때 방해되므로 해제하는 것을 권장한다.

- {"command":101} : Intializing the gripper
- {"command":102} : Openging the gripper
- {"command":103} : Closing the gripper

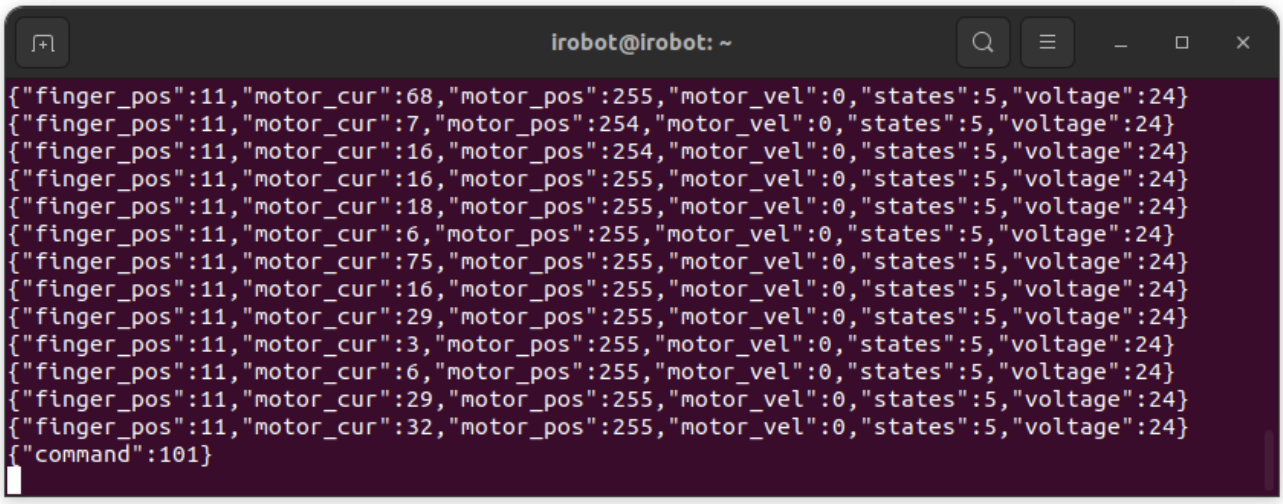



그림 5.11 TCP Socket 서버에 보내는 그리퍼 명령

## 5.2 ROS 2 기반의 KR-GCS

ROS 2 기반의 KR-GCS는 Ubuntu 22.04 및 ROS 2 Humble 버전에서 동작하는 소프트웨어이다. 기본적으로 5.1절의 TCP Socket 기반의 KR-GCS와 동일한 설치 방법 및 UI를 가지고 있으며, UI상의 유일한 차이는 그림 5.5와 같은 TCP Socket 탭이 없다는 점이다. ROS 2의 topic과 service는 소프트웨어의 실행과 동시에 자동으로 시작된다. 의존성, 빌드 방법, 그리고 topic과 service 이름 및 규칙은 소프트웨어 버전에 따라 다를 수 있으므로, 소프트웨어를 배포하는 Github Repository에서 확인하여야 하는데, 주소는 다음과 같다.

https://github.com/KorasRobotics/KR_GCS_user_interface_ROS2

## 5.3 C++ 예제 코드

본 절에서 제공하는 예제 코드는 Ubuntu 22.04에서 개발 및 시험하였으며, 필수 dependency의 설치 방법은 다음과 같다.

```
$ sudo apt install cmake
$ sudo apt install build-essential
$ sudo apt install libmodbus-dev
```

예제 코드는 소스 파일인 main.cpp와 빌드를 위한 CMakeLists.txt이며, 다음의 표와 같다. 이 코드는 코라스로보틱스 홈페이지(www.korasrobotics.com)의 게시판에서 다운로드 받을 수 있다.

표 5.1 CMakeLists.txt 코드

```
cmake_minimum_required(VERSION 3.5)
project(kr_gcs_test)

# Default to C++17
if(NOT CMAKE_CXX_STANDARD)
    set(CMAKE_CXX_STANDARD 17)
endif()

file (GLOB ${PROJECT_NAME}_SRCS
    *.cpp
)

add_executable(${PROJECT_NAME} ${${PROJECT_NAME}_SRCS})

target_link_libraries(${PROJECT_NAME}
    modbus
)
```

표 5.2 main.cpp 코드

```
#include <stdio.h>
#include <errno.h>
#include <csignal>
#include <thread>
#include <modbus/modbus-rtu.h>

#define BAUDRATE      115200
#define DATA_BIT      8
#define STOP_BIT      1
#define PARITY_MODE   'N'
#define DEBUG_MODE    false
#define SLAVE_ADDRESS 1

enum GrpCmd {
    INIT                = 101,
    OPEN                = 102,
    CLOSE               = 103,
    FINGER_POS          = 104,
    VACUUM_GRP_ON       = 106,
    VACUUM_GRP_OFF      = 107,
    MOTOR_TORQUE        = 212,
    MOTOR_SPEED         = 213,
    IMPEDANCE_ON        = 108,
    IMPEDANCE_OFF       = 109,
    SET_IMPEDANCE_PARAMS = 110,
};

struct GrpRecvData {
    uint16_t status;
    uint16_t position;
    uint16_t current;
    uint16_t velocity;
    uint16_t grp_pos_percent;
};

bool modbusInit(modbus_t **modbus_object, uint slave_address, char *port_name) {
    *modbus_object = modbus_new_rtu(port_name, BAUDRATE, PARITY_MODE, DATA_BIT, STOP_BIT);
    modbus_rtu_set_serial_mode(*modbus_object, MODBUS_RTU_RS485);
    modbus_rtu_set_rts_delay  (*modbus_object, 3000);

    modbus_set_debug(*modbus_object, DEBUG_MODE);

    if (*modbus_object == NULL) {
        fprintf(stderr, "Unable to create the libmodbus context\n");
```

```
            return false;
        }

        if (modbus_set_slave(*modbus_object, slave_address) == -1) {
            fprintf(stderr, "server_id = %d Invalid slave ID: %s\n", 1, modbus_strerror(errno));
            modbus_free(*modbus_object);
            return false;
        }

        if (modbus_connect(*modbus_object) == -1) {
            fprintf(stderr, "Unable to connect %s\n", modbus_strerror(errno));
            modbus_free(*modbus_object);
            return false;
        }

        printf("Modbus initiated\n");
        return true;
    }

    void sendOrder(modbus_t **modbus_object, GrpCmd grp_cmd, uint16_t value) {
        uint16_t holding_resister[2];
        uint16_t start_address = 0;
        uint16_t data_num = 2;

        holding_resister[0] = (uint16_t) grp_cmd;
        holding_resister[1] = value;

        modbus_write_registers(*modbus_object, start_address, data_num, holding_resister);
    }

    void sendOrder(modbus_t **modbus_object, GrpCmd grp_cmd, uint16_t value_1, uint16_t value_2)
{
        uint16_t holding_resister[3];
        uint16_t start_address = 0;
        uint16_t data_num = 3;

        holding_resister[0] = (uint16_t) grp_cmd;
        holding_resister[1] = value_1;
        holding_resister[2] = value_2;

        modbus_write_registers(*modbus_object, start_address, data_num, holding_resister);
    }

    GrpRecvData recvData(modbus_t **modbus_object) {
        int start_address = 10;
        int data_num = 5;
        uint16_t holding_resister[8] = {0,};

        if (modbus_read_registers(*modbus_object, start_address, data_num, holding_resister) == -
1) {
            fprintf(stderr, "Failed to read input registers! : %s\n", modbus_strerror(errno));
        }

        return (GrpRecvData) {holding_resister[0], holding_resister[1],
                              holding_resister[2], holding_resister[3], holding_resister[4]};
    }

    int main(int argc, char **argv) {
        modbus_t *modbus_object;

        if (!modbusInit(&modbus_object, SLAVE_ADDRESS, argv[1])) {
            printf("A modbus connection error occurred. Exit the software.\n");
            return -1;
        }

        static bool flag_sigint(false), flag_waiting_command(false);

        signal(SIGINT, [] (int signum) {
            printf("SIGINT received. Exiting...\n");
```

```
        printf("If the program does not terminate, enter the number 0\n");
        flag_sigint = true;
    });

    std::thread recv_thread([&] () {
        while (!flag_sigint) {
            if (!flag_waiting_command) {
                GrpRecvData grp_data = recvData(&modbus_object);

                printf("Status (binary): %b\n", grp_data.status);
                printf("Current (mA)   : %d\n", grp_data.current);
                printf("Gripper position (\%): %.2f\%\n\n", (double)grp_data.grp_pos_percent
/ 100);
            }

            sleep(2);
        }
    });

    while (!flag_sigint) {
        int input, input_2, input_3;
        scanf("%d", &input);

        if (flag_sigint) {
            input = 0;
        }

        switch (input) {
            case 0:
                printf("\'Program termination\' received.\n");
                flag_sigint = true;
                break;
            case 1:
                printf("\'Gripper Initialize\' received.\n");
                sendOrder(&modbus_object, INIT, 0);
                break;
            case 2:
                printf("\'Gripper Open\' received.\n");
                sendOrder(&modbus_object, OPEN, 0);
                break;
            case 3:
                printf("\'Gripper Close\' received.\n");
                sendOrder(&modbus_object, CLOSE, 0);
                break;
            case 4:
                printf("\'Vacuum Gripper On\' received.\n");
                sendOrder(&modbus_object, VACUUM_GRP_ON, 0);
                break;
            case 5:
                printf("\'Vacuum Gripper Off\' received.\n");
                sendOrder(&modbus_object, VACUUM_GRP_OFF, 0);
                break;
            case 6:
                flag_waiting_command = true;
                printf("\'Set Finger Position\' received.\n");
                printf("Enter the tartet finger position. (range: 0(closed) ~
10000(open)).\n");
                scanf("%d", &input_2);

                if (input_2 < 0) {
                    sendOrder(&modbus_object, FINGER_POS, 0);
                } else if (input_2 > 1000) {
                    sendOrder(&modbus_object, FINGER_POS, 1000);
                } else {
                    sendOrder(&modbus_object, FINGER_POS, input_2);
                }

                flag_waiting_command = false;
                break;
```

```
            case 7:
                flag_waiting_command = true;
                printf("\'Set Motor Torque\' received.\n");
                printf("Enter the target torque. (range: 50 ~ 100, unit: %).\n");
                scanf("%d", &input_2);

                if (input_2 < 50) {
                    sendOrder(&modbus_object, MOTOR_TORQUE, 50);
                } else if (input_2 > 100) {
                    sendOrder(&modbus_object, MOTOR_TORQUE, 100);
                } else {
                    sendOrder(&modbus_object, MOTOR_TORQUE, input_2);
                }

                flag_waiting_command = false;
                break;
            case 8:
                flag_waiting_command = true;
                printf("\'Set Motor Speed\' received.\n");
                printf("Enter the target speed. (range: 1 ~ 100, unit: %).\n");
                scanf("%d", &input_2);

                if (input_2 < 1) {
                    sendOrder(&modbus_object, MOTOR_SPEED, 1);
                } else if (input_2 > 100) {
                    sendOrder(&modbus_object, MOTOR_SPEED, 100);
                } else {
                    sendOrder(&modbus_object, MOTOR_SPEED, input_2);
                }

                flag_waiting_command = false;
                break;
            case 9:
                printf("\'Impedance Control On\' received.\n");
                sendOrder(&modbus_object, IMPEDANCE_ON, 0);
                break;
            case 10:
                printf("\'Impedance Control Off\' received.\n");
                sendOrder(&modbus_object, IMPEDANCE_OFF, 0);
                break;
            case 11:
                flag_waiting_command = true;
                printf("\'Set Impedance Parameters\' received.\n");
                printf("Enter the gripper type. (range: 1 ~ 20).\n");
                scanf("%d", &input_2);

                if (input_2 <= 20 && input_2 >= 1) {
                    printf("Enter the stiffness level. (range: 1 ~ 10).\n");
                    scanf("%d", &input_3);

                    if (input_3 <= 10 && input_3 >= 1) {
                        sendOrder(&modbus_object, SET_IMPEDANCE_PARAMS, input_2, input_3);
                    } else {
                        printf("[Error] input range error.\n");
                    }
                } else {
                   printf("[Error] input range error.\n");
                }

                flag_waiting_command = false;
                break;
        }

        sleep(1);
    }

    recv_thread.join();

    modbus_close(modbus_object);
```

```
        printf("Modbus port closed\n");
        modbus_free (modbus_object);
        printf("Modbus port released\n");

        return 0;
    }
```

예제 코드는 cmake를 통해 빌드할 수 있다. 실행 파일명이 "kr_gcs_test" 이고, 연결된 시리얼 포트가 "/dev/ttyUSB0"일 때 실행 방법은 다음과 같다.

```
$ ./kr_gcs_test /dev/ttyUSB0
```

실행된 소프트웨어는 기본적으로 2초마다 그리퍼의 상태, 위치 및 전류 정보를 터미널에 출력한다. 이때 그리퍼와 정상적으로 연결되었다면, 4.3에서 언급한 명령들을 사용하여 그리퍼를 구동하거나 프로그램을 종료할 수 있다.

## 매뉴얼 수정 이력

| 수정일 | 버전 | 수정 내용 |
|---|---|---|
| 2023.07.20 | 1.0 | 신규 작성 |
| 2023.09.04 | 1.1 | 이미지 변경 및 슬레이브 내용 추가 |
| 2023.12.30 | 1.2 | 샘플 코드, 주의사항 추가 |
| 2024.11.01 | 1.3 | 임피던스 파라미터 |