부록 E 외부 스크립트 제어 API

E.0 개념

협동로봇 RB Series 는 다양한 환경과 목적으로 운용될 수 있습니다. 다수의 RB Series 혹은 다른 시스템과 연동되어 사용될 수 있습니다. 비전시스템과 연동되어 이동 좌표를 실시간으로 변경하거나 사용자가 기존에 사용하던 시스템의 한 부분으로 사용할 수도 있습니다.

주어진 태블릿 UI 로 로봇을 제어할 수도 있지만, 사용자의 편의나 운용의 이점을 위해서 외부의 임의의 제어기로부터 로봇을 제어할 수 있는 방법을 제공합니다.

RB Series 는 기본적으로 스크립트 명령을 받고 그 명령을 수행합니다. 태블릿 UI 로 태스크 모션을 작성하고, 그 파일의 스크립트를 차례대로 수행하는 것이 일반적인 운용방안이라면, 본 문서에서 설명하고자 하는 방법은 외부의 다른 디바이스로부터 명령어 스크립트를 입력 받아 해석하여 움직이는 방법입니다.

태블릿 UI 에서 제공하던 IF, REPEAT 등의 제어 구문은 사용자가 외부 제어 디바이스에서 직접 구현하고, 로봇의 동작 명령어와 IO 제어 명령어를 상황에 맞게 보내주어 로봇을 구동합니다.

본 문서에서는 위 개념으로 로봇을 구동하는 예제를 설명합니다.

E.1 외부 제어 스크립트 API

본 문서에서 제공하는 스크립트의 설명은 외부 제어를 위한 전용 스크립트로 태블릿을 사용해 작성하는 ".wsl" 워크 문서의 스크립트와 유사해 보이지만 다릅니다. 워크 문서에는 "repeat", "if-else", "break" 등과 같은 흐름을 제어하는 구문이 들어가기 때문에 하나의 문장(statement)의 완성이 동작으로 직결되지 않고, 그 문장의 상위 문장도 완성되어야 합니다.

예를 들어 move 명령 안에 point 들이 존재한다고 하였을 때 다음과 같이 표현될 수 있습니다.

1)

```
move joint {
   point ( ) absolute 0.4, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
      point ( ) absolute 0.4, 0.1, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0
}
```

2)

```
move joint {
   point ( ) absolute 0.4, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
      point ( ) absolute 0.4, 0.1, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0
```

1)과 2)의 차이는 마지막에 "}"의 유무입니다. 두 경우에서 point 문장은 완성이 되어 있습니다. 하지만 1)과 달리 2)는 point 의 상위 문장인 move 문장이 완성되지 않았기 때문에 동작할 수 없는 구문이고, 파서(parser)는 문장이 완성되길 기다릴 것입니다.

3)

```
folder( ) {
      move joint {
      point ( ) absolute 0.4, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
            point ( ) absolute 0.4, 0.1, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0
      }
```

같은 논리로 3)의 경우에도 파서는 folder 문장이 완성되길 기다리며 동작되지 않습니다.

위와 같이 여러 줄의 구문들이 완성되어야 동작하는 방법은 외부 제어 방법에는 적합하지 않습니다. 사용자는 외부 제어를 통해 명령을 보내는 순간 그 명령을 파싱(parsing)해서 동작하길 기대합니다. 여러 번의 명령을 보내서 그 명령들이 3)과 같이 여러 줄의 문장들을 완성시켜 나가는 것을 원하지 않습니다.

그래서 외부 제어는 모든 명령이 한 문장으로 끝날 수 있도록 간결하게 구성되어 있습니다. 대신에 흐름을 제어하는 구문이나 기타 불필요한 기능들을 제공하지 않습니다. 워크 문서에 있던 "repeat", "if-else", "break"와 같은 구문이나 "wait"와 같은 구문들도 외부 제어를 수행하는 소프트웨어에서 자체적으로 해결하는 것이 구조상이나 논리상 더 적합하고 간결합니다.

우선 설명드릴 스크립트는 실제로 로봇을 움직이는 동작 명령어입니다.

동작 명령은 크게 5 가지입니다.

1) jointall
2) movetcp
3) movecircle
4) blend_jnt
5) blend_tcp

jointall

명령어	jointall
스크립트	jointall spd, acc, joint1, joint2, joint3, joint4, joint5, joint6
설명	이 명령은 로봇의 각 관절을 Joint Space 에서 움직입니다. joint1 부터 joint6 는 차례대로 base, shoulder, elbow, wrist1, wrist2, wrist3 관절의 목표 관절의 절대 각도 값입니다. degree 단위의 실수형 상수를 입력합니다. spd 와 acc 는 각각 속도와 가속도 값을 나타냅니다. 0~1 사이의 실수형 상수를 입력합니다. 0 에 가까울수록 느린 값을 나타냅니다. -1 을 입력할 경우에는 시스템에 설정된 기본 속도와 가속도 값으로 움직입니다. 기존에 다른 동작 명령을 수행 중이었다면 이 명령은 무시됩니다.
예시	“jointall 0.4, 0.1, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0”

movetcp

명령어	movetcp
스크립트	movetcp spd, acc, x, y, z, rx, ry, rz
설명	이 명령은 입력된 TCP 의 위치와 회전각을 목표로 Cartesian Space 에서 움직입니다. x, y, z 는 TCP 의 위치를 나타내는 값으로 mm 단위의 실수형 상수입니다. rx, ry, rz 는 TCP 의 방향을 결정하기 위한 Yaw-Pitch-Roll 순서의 오일러 각도를 나타내고, degree 단위의 실수형 상수를 입력합니다. spd 와 acc 는 각각 속도와 가속도 값을 나타냅니다. 0~1 사이의 실수형 상수를 입력합니다. 0 에 가까울수록 느린 값을 나타냅니다. -1 을 입력할 경우에는 시스템에 설정된 기본 속도와 가속도 값으로 움직입니다. 기존에 다른 동작 명령을 수행 중이었다면 이 명령은 무시됩니다.
예시	“movetcp 0.4, 0.1, 100.0, 100.0, 300.0, 0.0, 90.0, 0.0”

movecircle

명령어	movecircle(three points 모드)
스크립트	movecircle threepoints orientation_option spd, acc, x1, y1, z1, rx1, ry1, rz1, x2, y2, z2, rx2, ry2, rz2
설명	이 명령은 TCP 의 현재 위치를 시작으로 입력된 중간 위치와 최종 위치를 이용해 Cartesian Space 에서 원을 그리는 동작입니다. orientation_option 을 통해서 원을 그릴 때 TCP 의 방향을 결정하는 3 가지 옵션을 선택할 수 있습니다. “intended”는 중간 위치와 최종 위치에 입력한 방향 (rx1, ry1, rz1 과 rx2, ry2, rz2)을 모두 지키면서 원을 그립니다. “constant”는 시작점인 TCP 의 현재 방향을 유지한 채로 원을 그립니다. “radial”은 그리게 되는 원의 중심방향으로 TCP 의 방향이 상대적으로 변합니다. 방향의 기준은 현재 TCP 의 방향입니다. x1, y1, z1 는 원의 중간 위치에서의 TCP 의 위치를 나타내는 값으로 mm 단위의 실수형 상수입니다. rx1, ry1, rz1 는 원의 중간 위치에서의 TCP 의 방향을 결정하기 위한 Yaw-Pitch-Roll 순서의 오일러 각도를 나타내고, degree 단위의 실수형 상수를 입력합니다. x2, y2, z2 는 원의 최종 위치에서의 TCP 의 위치를 나타내는 값으로 mm 단위의 실수형 상수입니다. rx2, ry2, rz2 는 원의 최종 위치에서의 TCP 의 방향을 결정하기 위한 Yaw-Pitch-Roll 순서의 오일러 각도를 나타내고, degree 단위의 실수형 상수를 입력합니다. spd 와 acc 는 각각 속도와 가속도 값을 나타냅니다. 0~1 사이의 실수형 상수를 입력합니다. 0 에 가까울수록 느린 값을 나타냅니다. -1 을 입력할 경우에는 시스템에 설정된 기본 속도와 가속도 값으로 움직입니다. 기존에 다른 동작 명령을 수행 중이었다면 이 명령은 무시됩니다.
예시	“movecircle threepoints intended 0.4, 0.1, 100.0, 100.0, 300.0, 0.0, 90.0, 0.0, 200.0, 200.0, 200.0, 0.0, 90.0, 45.0” “movecircle threepoints constant 0.4, 0.1, 100.0, 100.0, 300.0, 0.0, 90.0, 0.0, 200.0, 200.0, 200.0, 0.0, 90.0, 45.0” “movecircle threepoints radial 0.4, 0.1, 100.0, 100.0, 300.0, 0.0, 90.0, 0.0, 200.0, 200.0, 200.0, 0.0, 90.0, 45.0”

명령어	movecircle(axis 모드)
스크립트	movecircle axis orientation_option spd, acc, rot_angle, cx, cy, cz, ax, ay, az
설명	이 명령은 TCP 의 현재 위치에서 시작해서 입력된 회전축을 중심으로 입력된 각도만큼 원을 그리는 동작입니다. orientation_option 을 통해서 원을 그릴 때 TCP 의 방향을 결정하는 3 가지 옵션을 선택할 수 있습니다. “intended”는 아래의 “constant”와 동일합니다. “constant”는 시작점인 TCP 의 현재 방향을 유지한 채로 원을 그립니다. “radial”은 그리게 되는 원의 중심방향으로 TCP 의 방향이 상대적으로 변합니다. 방향의 기준은 현재 TCP 의 방향입니다. cx, cy, cz 는 회전 중심점의 위치를 나타내는 값으로 mm 단위의 실수형 상수입니다. ax, ay, az 는 회전의 축을 나타내는 벡터로 실수형 상수를 입력합니다. rot_angle 은 회전할 양을 나타내는 값으로 degree 단위의 실수형 상수입니다. spd 와 acc 는 각각 속도와 가속도 값을 나타냅니다. 0~1 사이의 실수형 상수를 입력합니다. 0 에 가까울수록 느린 값을 나타냅니다. -1 을 입력할 경우에는 시스템에 설정된 기본 속도와 가속도 값으로 움직입니다. 기존에 다른 동작 명령을 수행 중이었다면 이 명령은 무시됩니다.
예시	“movecircle axis constant 0.4, 0.1, 180.0, 200.0, 200.0, 200.0, 1.0, 0.0, 0.0” “movecircle axis radial 0.4, 0.1, 180.0, 200.0, 200.0, 200.0, 1.0, 0.0, 0.0”

blend_jnt

명령어	blend_jnt
스크립트	blend_jnt clear_pt
설명	이 명령은 Joint Space 에서 연속된 목적 관절 각도들을 연결하는 동작 명령 중에서 저장된 목적 값들을 지우는 명령어입니다. 새로운 연속 관절 각도들을 입력하기 전의 사전 동작으로 사용합니다.
예시	“blend_jnt clear_pt”

명령어	blend_jnt
스크립트	blend_jnt add_pt spd, acc, joint1, joint2, joint3, joint4, joint5, joint6
설명	이 명령은 Joint Space 에서 연속된 목적 관절 각도들을 연결하는 동작 명령 중에서 목적 관절 각도들을 입력하는 명령어입니다. joint1 부터 joint6 는 차례대로 base, shoulder, elbow, wrist1, wrist2, wrist3 관절의 목표 관절의 절대 각도 값입니다. degree 단위의 실수형 상수를 입력합니다. spd 와 acc 는 각각 속도와 가속도 값을 나타냅니다. 0~1 사이의 실수형 상수를 입력합니다. 0 에 가까울수록 느린 값을 나타냅니다. -1 을 입력할 경우에는 시스템에 설정된 기본 속도와 가속도 값으로 움직입니다. 마지막으로 입력된 spd 와 acc 로 전체 “blend_jnt” 명령의 속도와 가속도가 결정됩니다.
예시	“blend_jnt add_pt 0.4, 0.1, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0”

명령어	blend_jnt
스크립트	blend_jnt move_pt
설명	이 명령은 Joint Space 에서 연속된 목적 관절 각도들을 연결하는 동작 명령 중에서 동작을 시작시키는 명령어입니다. 기존에 “blend_jnt add_pt”로 입력한 관절 값들로 차례대로 움직이는 동작이 수행됩니다.
예시	“blend_jnt move_pt”

blend_tcp

명령어	blend_tcp
스크립트	blend_tcp clear_pt
설명	이 명령은 Cartesian Space 에서 연속된 목적 위치들을 연결하는 동작 명령 중에서 저장된 목적 값들을 지우는 명령어입니다. 새로운 연속 목적 위치들을 입력하기 전의 사전 동작으로 사용합니다.
예시	“blend_tcp clear_pt”

명령어	blend_tcp
스크립트	blend_tcp add_pt spd, acc, radius, x, y, z, rx, ry, rz
설명	이 명령은 Cartesian Space 에서 연속된 목적 위치들을 연결하는 동작 명령 중에서 목적 위치들을 입력하는 명령어입니다. radius 는 목적 위치들을 연달아 이동할 때 곡선을 그리며 다음 목적 위치로 이동하기 위한 거리를 나타낸 값으로 mm 단위의 상수를 입력합니다. 하나의 목적 위치에서 다음 목적 위치로의 이동을 선분으로 생각할 때 그 절반 값이 radius 가 가질 수 있는 최대 값입니다. 이 값보다 크게 주면 내부적으로 알아서 값을 줄여서 사용합니다. x, y, z 는 TCP 의 위치를 나타내는 값으로 mm 단위의 실수형 상수입니다. rx, ry, rz 는 TCP 의 방향을 결정하기 위한 Yaw-Pitch-Roll 순서의 오일러 각도를 나타내고, degree 단위의 실수형 상수를 입력합니다. spd 와 acc 는 각각 속도와 가속도 값을 나타냅니다. 0~1 사이의 실수형 상수를 입력합니다. 0 에 가까울수록 느린 값을 나타냅니다. -1 을 입력할 경우에는 시스템에 설정된 기본 속도와 가속도 값으로 움직입니다. 마지막으로 입력된 spd 와 acc 로 전체 “blend_tcp” 명령의 속도와 가속도가 결정됩니다.
예시	“blend_tcp add_pt 0.4, 0.1, 30.0, 100.0, 100.0, 300.0, 0.0, 90.0, 0.0”

명령어	blend_tcp
스크립트	blend_tcp move_pt
설명	이 명령은 Cartesian Space 에서 연속된 목적 위치들을 연결하는 동작 명령 중에서 동작을 시작시키는 명령어입니다. 기존에 “blend_tcp add_pt”로 입력한 목적 위치들로 차례대로 움직이는 동작이 수행됩니다.
예시	“blend_tcp move_pt”

다음으로 설명드릴 스크립트는 배전반과 툴플랜지의 디지털과 아날로그 포트의 출력 값을 제어하는 명령입니다. 명령어는 3 가지입니다.

1) digital_out
2) analog_out
3) tool_out

digital_out

스크립트	digital_out d0, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13, d14, d15
설명	이 명령은 제어반의 디지털 출력 포트를 통해서 신호를 내보냅니다. d0~d15 는 각각의 디지털 포트의 on/off 를 나타냅니다. 값이 1 이면 on 이고, 0 이면 off 입니다. d0~d15 에 0 과 1 이외에 -1 을 입력할 수 있습니다. 이 경우에는 해당 포트의 기존 출력 값을 유지합니다.
예시	“digital_out 1, 1, 1, 1, 0, 0, 0, 0, -1, -1, -1, -1, -1, -1, -1, -1”

analog_out

스크립트	analog_out a0, a1, a2, a3
설명	이 명령은 제어반의 아날로그 출력 포트를 통해서 신호를 내보냅니다. a0~a3 은 각각의 아날로그 포트의 출력 값을 나타냅니다. 값이 0 에서 10 의 값을 가지며 voltage 단위입니다. a0~a3 에 -1 을 입력할 수 있습니다. 이 경우에는 해당 포트의 기존 출력 값을 유지합니다.
예시	“analog_out 5.0, 5.0, -1, -1”

tool_out

스크립트	tool_out volt, d0, d1
설명	이 명령은 툴플랜지의 전압 레벨을 설정하고, 디지털 출력 포트를 통해서 신호를 내보냅니다. volt 는 툴플랜지의 디지털 출력의 전압 레벨을 설정합니다. 값은 0, 12, 24 가 가능하고, 이는 voltage 단위입니다. 이 세개의 값 이외의 입력은 무시됩니다. volt 에 -1 을 입력할 수 있습니다. 이 경우에는 기존의 전압 설정을 유지합니다. d0, d1 은 각각의 디지털 포트의 on/off 를 나타냅니다. 값이 1 이면 on 이고, 0 이면 off 입니다. d0, d1 에 0 과 1 이외에 -1 을 입력할 수 있습니다. 이 경우에는 해당 포트의 기존 출력 값을 유지합니다.
예시	“tool_out 12, 1, 0”

다음으로 설명드릴 스크립트는 초기화, 종료, 동작모드 변경, 속도 변경입니다.

1) mc
2) shutdown
3) pgmode
4) sdw

mc

스크립트	mc jall init
설명	이 명령은 로봇 하드웨어를 초기화합니다.
예시	“mc jall init”

shutdown

스크립트	shutdown
설명	이 명령은 로봇 동작을 종료시키고, 전원을 내립니다.
예시	“shutdown”

pgmode

스크립트	pgmode mode_type
설명	이 명령은 로봇의 동작 모드를 변경합니다. mode_type 을 통해서 동작 모드를 선택합니다. 동작 모드에는 “real”과 “simulation”이 있습니다. “real”에서는 동작 명령을 내릴 때 실제로 로봇이 움직입니다. “simulation”에서는 동작 명령을 내릴 때 내부 레퍼런스만 변경되고, 그 명령이 로봇을 움직이지는 않습니다. 로봇을 처음 초기화 하였을 때 기본 모드는 “simulation” 모드입니다.
예시	“pgmode real” “pgmode simulation”

sdw (shared data write)

스크립트	sdw default_speed spd
설명	이 명령은 로봇의 모든 동작에 대한 기본 속도 배율을 지정합니다. spd 에는 0~1 사이의 실수형 상수를 입력합니다. 0 에 가까울수록 느린 값을 나타냅니다. 이 값이 0 이면 로봇은 움직이지 않습니다. 또한 레퍼런스도 변하지 않습니다. 이 속도 값은 팬던트가 연결되어 있을 경우에, 특히 “Make” 모드에 진입한 상태일 경우에 팬던트에 의해서도 값이 제어가 되기 때문에 사용에 유의하시기 바랍니다. 외부 제어와 팬던트 사이에 별도의 우선권이 존재하지 않고, 마지막에 입력한 값이 최종 값입니다.
예시	“sdw default_speed 0.5”

마지막으로 설명드릴 스크립트는 task 스크립트입니다.

task

스크립트	task load work_file_name
설명	이 명령은 기존에 작성된 워크 파일을 로드합니다. work_file_name 에는 “.wsl” 파일의 이름이 사용됩니다. 이때에 “.wsl”은 생략하고, 파일의 상대경로와 이름만 입력합니다. 이 파일은 팬던트에 존재하는 파일이 아니라, 팬던트를 통해서 한번이라도 제어반에 연결하여 로드 혹은 저장했었던 파일이어야 합니다. 그렇기 때문에 팬던트가 연결되어 있지 않더라도 해당 파일을 로드할 수 있습니다.
예시	“task load test_file”

스크립트	task play option
설명	이 명령은 로드된 워크 파일을 실행합니다. option 에는 아무런 값이 들어가지 않거나 “once”가 입력될 수 있습니다. option 에 아무런 값을 주지 않으면 설정된 횟수만큼 워크파일을 실행합니다. option 에 “once”를 입력할 경우에 워크 파일은 단 한 번만 실행됩니다.
예시	“task play” “task play once”

스크립트	task repeat num
설명	이 명령은 워크 파일을 실행할 경우의 반복 횟수를 지정합니다. num 에는 반복 횟수를 정수형으로 입력합니다. 만약 num 에 -1 을 입력하면 무한대를 의미합니다. 이 명령으로 설정한 반복 횟수는 배전반의 전원을 리붓 시키기 전까지 유지됩니다. 그 이후에는 팬던트로 설정한 값으로 다시 치환됩니다.
예시	“task repeat 5” “task repeat -1”

스크립트	task pause
설명	이 명령은 실행 중이던 로봇 동작을 일시정지 시킵니다. 외부 제어를 통한 동작 명령과 “task play” 명령으로 인해 실행 중이던 워크 파일에 모두 사용할 수 있습니다. 일시정지된 상태에서는 “task resume_a” 명령을 통해서 동작을 재개할 수 있습니다. 로봇이 일시정지된 상태일 때에는 동작이 끝난 것으로 생각하지 않기 때문에 외부 제어에서 다른 동작 명령을 주어도 무시됩니다.
예시	“task pause”

스크립트	task stop
설명	이 명령은 실행 중이던 로봇 동작을 정지시킵니다. 외부 제어를 통한 동작 명령과 “task play” 명령으로 인해 실행 중이던 워크 파일에 모두 사용할 수 있습니다. 동작이 정지된 상태에서는 “task resume_a”로 동작이 재개되지 않습니다. 동작이 완전 종료된 상태입니다. 동작을 정지 명령을 내리면 로봇이 빠른 동작을 하고 있던 중에 급격히 멈출 수 있습니다. “task pause” 이후에 “task stop” 명령을 수행하기를 권장합니다.
예시	“task stop”

스크립트	task resume_a
설명	이 명령은 “task pause”나 워크 파일에서 alarm 혹은 debug 명령으로 일시정지된로봇의 동작을 재개합니다.
예시	“task resume_a”

스크립트	task resume_b
설명	이 명령은 외부 충돌 감지로 인해 일시정지된 로봇의 동작을 재개합니다.
예시	“task resume_b”

외부 제어를 사용하기 위해서 제어반의 프로그램과 연결을 해야 합니다. 연결은 TCP/IP 통신을 사용하고, 해당하는 IP 주소는 팬던트에서 설정할 수 있습니다. 그리고 그 결과는 제어반의 LCD 화면에 표시됩니다. 외부 제어를 위해 5000 번과 5001 번 포트가 열립니다. 5000 번 포트는 명령어를 받기 위한 포트이고, 5001 번 포트는 로봇 상태를 나타내는 데이터를 요청받고 보내주는 역할을 위한 포트입니다. 편의상 5000 번 포트는 명령 포트라고 하고, 5001 번 포트를 데이터 포트라고 하겠습니다.

명령 포트로는 앞에서 설명한 스크립트 명령어를 전송하면 됩니다. 명령 포트에는 첫 명령어에 대한 필터가 있어서 시작이 “jointall”, “movetcp”, “mc”, “pgmode”, 등과 같은 앞서 설명한 스크립트 명령어가 아닐 경우에는 “The command is not allowed”라는 문구를 응답으로 보냅니다. 정상적인 명령어로 시작되어 파서로 입력 문장을 넘겼을 경우에는 “The command was executed”라는 문구의 응답이 옵니다.

데이터 포트로는 “reqdata” 라는 명령어를 보내면 그에 대한 응답으로 현대 로봇 상태에 대한 정보가 데이터 포트로 들어옵니다. 데이터의 형식은 아래와 같습니다.

Header (4 Byte)				Data (n Byte)
0x24	Size&0xFF	(Size>>8)&0xFF	0x03	Data

그리고 Data 의 형식은 아래와 같습니다. 시스템의 버전에 따라서 Data 의 사이즈가 다를 수도 있습니다. 하지만 순서는 일치하니 아래 표를 참조하시기 바랍니다.

Offset	타입	설명
0	Float	Task 가 수행된 시간(초단위 – 태스크가 시작할 때 리셋)
1	Float	현재 base 관절의 레퍼런스 각도(degree 단위)
2	Float	현재 shoulder 관절의 레퍼런스 각도(degree 단위)
3	Float	현재 elbow 관절의 레퍼런스 각도(degree 단위)
4	Float	현재 wrist1 관절의 레퍼런스 각도(degree 단위)
5	Float	현재 wrist2 관절의 레퍼런스 각도(degree 단위)
6	Float	현재 wrist3 관절의 레퍼런스 각도(degree 단위)
7	Float	현재 base 관절의 엔코더 각도(degree 단위)
8	Float	현재 shoulder 관절의 엔코더 각도(degree 단위)
9	Float	현재 elbow 관절의 엔코더 각도(degree 단위)
10	Float	현재 wrist1 관절의 엔코더 각도(degree 단위)
11	Float	현재 wrist2 관절의 엔코더 각도(degree 단위)
12	Float	현재 wrist3 관절의 엔코더 각도(degree 단위)
13	Float	현재 base 관절의 전류값(Ampere 단위)
14	Float	현재 shoulder 관절의 전류값(Ampere 단위)
15	Float	현재 elbow 관절의 전류값(Ampere 단위)
16	Float	현재 wrist1 관절의 전류값(Ampere 단위)
17	Float	현재 wrist2 관절의 전류값(Ampere 단위)
18	Float	현재 wrist3 관절의 전류값(Ampere 단위)
19	Float	현재 TCP 레퍼런스의 위치 X 값(mm 단위)
20	Float	현재 TCP 레퍼런스의 위치 Y 값(mm 단위)
21	Float	현재 TCP 레퍼런스의 위치 Z 값(mm 단위)
22	Float	현재 TCP 레퍼런스의 방향 각도 RX 값(degree 단위)
23	Float	현재 TCP 레퍼런스의 방향 각도 RY 값(degree 단위)
24	Float	현재 TCP 레퍼런스의 방향 각도 RZ 값(degree 단위)
25	Float	19 와 동일
26	Float	20 와 동일
27	Float	21 와 동일
28	Float	22 와 동일
29	Float	23 와 동일
30	Float	24 와 동일
31	Float	제어반 아날로그 신호 0 번 포트 입력값(voltage 단위)
32	Float	제어반 아날로그 신호 1 번 포트 입력값(voltage 단위)
33	Float	제어반 아날로그 신호 2 번 포트 입력값(voltage 단위)
34	Float	제어반 아날로그 신호 3 번 포트 입력값(voltage 단위)
35	Float	제어반 아날로그 신호 0 번 포트 출력값(voltage 단위)
36	Float	제어반 아날로그 신호 1 번 포트 출력값(voltage 단위)
37	Float	제어반 아날로그 신호 2 번 포트 출력값(voltage 단위)
38	Float	제어반 아날로그 신호 3 번 포트 출력값(voltage 단위)
39	Int	제어반 디지털 신호 0 번 포트 입력값(on:1 / off:0)
40	Int	제어반 디지털 신호 1 번 포트 입력값(on:1 / off:0)
41	Int	제어반 디지털 신호 2 번 포트 입력값(on:1 / off:0)
42	Int	제어반 디지털 신호 3 번 포트 입력값(on:1 / off:0)
43	Int	제어반 디지털 신호 4 번 포트 입력값(on:1 / off:0)
44	Int	제어반 디지털 신호 5 번 포트 입력값(on:1 / off:0)
45	Int	제어반 디지털 신호 6 번 포트 입력값(on:1 / off:0)

46	Int	제어반 디지털 신호 7 번 포트 입력값(on:1 / off:0)
47	Int	제어반 디지털 신호 8 번 포트 입력값(on:1 / off:0)
48	Int	제어반 디지털 신호 9 번 포트 입력값(on:1 / off:0)
49	Int	제어반 디지털 신호 10 번 포트 입력값(on:1 / off:0)
50	Int	제어반 디지털 신호 11 번 포트 입력값(on:1 / off:0)
51	Int	제어반 디지털 신호 12 번 포트 입력값(on:1 / off:0)
52	Int	제어반 디지털 신호 13 번 포트 입력값(on:1 / off:0)
53	Int	제어반 디지털 신호 14 번 포트 입력값(on:1 / off:0)
54	Int	제어반 디지털 신호 15 번 포트 입력값(on:1 / off:0)
55	Int	제어반 디지털 신호 0 번 포트 출력값(on:1 / off:0)
56	Int	제어반 디지털 신호 !번 포트 출력값(on:1 / off:0)
57	Int	제어반 디지털 신호 2 번 포트 출력값(on:1 / off:0)
58	Int	제어반 디지털 신호 3 번 포트 출력값(on:1 / off:0)
59	Int	제어반 디지털 신호 4 번 포트 출력값(on:1 / off:0)
60	Int	제어반 디지털 신호 5 번 포트 출력값(on:1 / off:0)
61	Int	제어반 디지털 신호 6 번 포트 출력값(on:1 / off:0)
62	Int	제어반 디지털 신호 7 번 포트 출력값(on:1 / off:0)
63	Int	제어반 디지털 신호 8 번 포트 출력값(on:1 / off:0)
64	Int	제어반 디지털 신호 9 번 포트 출력값(on:1 / off:0)
65	Int	제어반 디지털 신호 10 번 포트 출력값(on:1 / off:0)
66	Int	제어반 디지털 신호 11 번 포트 출력값(on:1 / off:0)
67	Int	제어반 디지털 신호 12 번 포트 출력값(on:1 / off:0)
68	Int	제어반 디지털 신호 13 번 포트 출력값(on:1 / off:0)
69	Int	제어반 디지털 신호 14 번 포트 출력값(on:1 / off:0)
70	Int	제어반 디지털 신호 15 번 포트 출력값(on:1 / off:0)
71	Float	현재 base 관절의 모터제어기 온도(celcius 단위)
72	Float	현재 shoulder 관절의 모터제어기 온도(celcius 단위)
73	Float	현재 elbow 관절의 모터제어기 온도(celcius 단위)
74	Float	현재 wrist1 관절의 모터제어기 온도(celcius 단위)
75	Float	현재 wrist2 관절의 모터제어기 온도(celcius 단위)
76	Float	현재 wrist3 관절의 모터제어기 온도(celcius 단위)
77	Int	태스크의 프로그램 카운터 위치(Step 명령이 수행될 위치)
78	Int	태스크 목표 반복 횟수
79	Int	현재 수행 중이 태스크의 액션 번호
80	Int	태스크의 현재 반복 횟수
81	Float	태스크 수행 시간(초단위 – 태스크가 시작할 때 리셋 안됨)
82	Int	태스크 상태(1: Idle, 2: Paused, 3: Run)
83	Float	로봇 동작의 기본 속도 배율(0~1)
84	Float	로봇 동작 상태(1: 정지, 3: 동작중)
85	Float	파워 상태 - LSB offset 기준으로 0: 48V 입력 1: 48V 출력 2: 24V 상태 3: 비상스위치 상태 4: PC 스위치 상태 5: 모터 제어기 상태
86	Float	Not used
87	Float	Not used
88	Float	Not used
89	Float	Not used
90	Float	Not used
91	Float	Not used

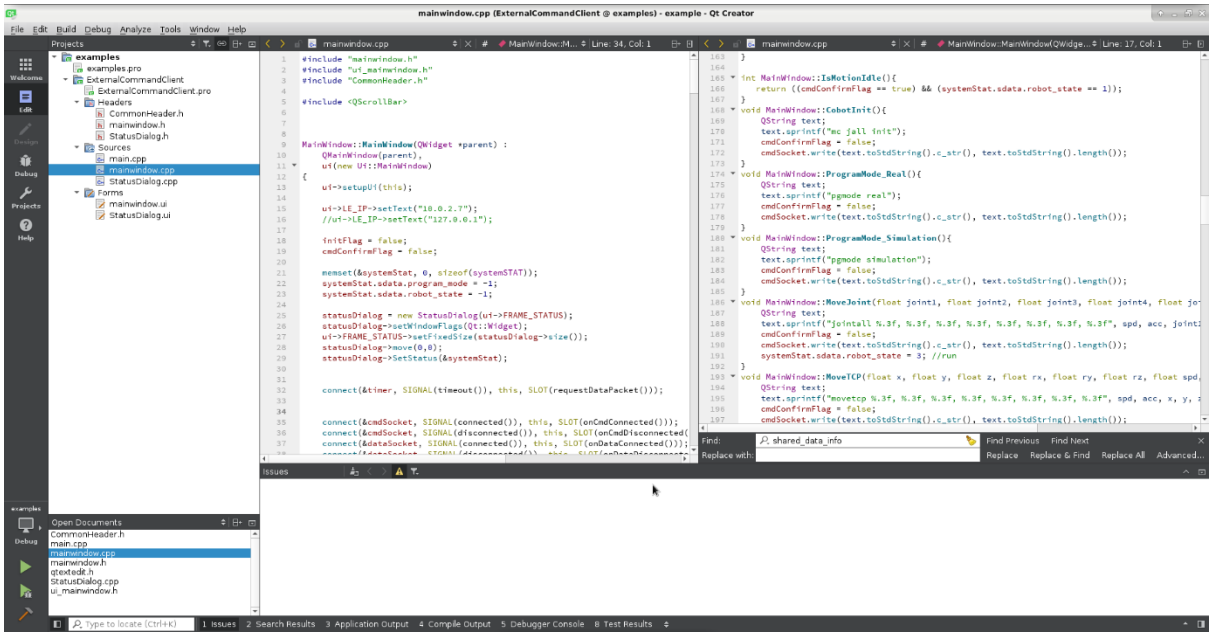
92	Int	현재 base 관절의 모터제어기 상태
93	Int	현재 shoulder 관절의 모터제어기 상태
94	Int	현재 elbow 관절의 모터제어기 상태
95	Int	현재 wrist1 관절의 모터제어기 상태
96	Int	현재 wrist2 관절의 모터제어기 상태
97	Int	현재 wrist3 관절의 모터제어기 상태
		모터제어기 상태 정보 – LSB offset 기준으로 0: FET 상태 1: 위치제어 상태 2: 초기화 결과 3: 제어 모드 4: Nonius 에러 5: Low Battery 6: 캘리브레이션 모드 7: Multiturn 에러 8: JAM 에러 9: Over Current 에러 10: Big 에러 11: Input 에러 12: FET 드라이버 에러 13: 온도 에러 14: 위치 에러(Low) 15: 위치 에러(High)
98	Int	충돌 감지 on-off(1: on, 0: off)
99	Int	직접 교시 상태(1: on, 0: off)
100	Int	로봇 동작 모드(1: simulation mode, 0: real mode)
101	Int	로봇 초기화 상태 정보 0: 기본값 1: 전압 체크 2: 디바이스 체크 3: 로봇 위치제어 시작 4: 변수 체크 5: 충돌 체크 6: 초기화 완료
102	Int	로봇 초기화 에러 코드 0: 에러 없음 1: SMPS 에러 2: 비상 스위치 에러 3: 컨트롤 박스 내부 파워 전환 에러 1 4: 컨트롤 박스 내부 파워 전환 에러 2 5: 로봇 접속 에러 6: 로봇 초기화 에러 7: 부하 세팅 검사 에러 8: 툴플랜지 접속 에러 9: 툴플랜지 각도 에러 10: 모터 제어기 엔코더 초기값 에러 1 11: 모터 제어기 엔코더 초기값 에러 2 12: 디지털 인풋 16/17 단락 에러 13: 48V 스위치 에러 14: 직접교시 버튼 눌림 에러
103	Float	툴플랜지 아날로그 신호 0 번 포트 입력값(voltage 단위)
104	Float	툴플랜지 아날로그 신호 1 번 포트 입력값(voltage 단위)
105	Int	툴플랜지 디지털 신호 0 번 포트 입력값(on:1 / off:0)

106	Int	툴플랜지 디지털 신호 1 번 포트 입력값(on:1 / off:0)
107	Int	툴플랜지 디지털 신호 0 번 포트 출력값(on:1 / off:0)
108	Int	툴플랜지 디지털 신호 1 번 포트 출력값(on:1 / off:0)
109	Float	툴플랜지의 출력 전압 레벨(voltage 단위)
110	Int	외부 충돌 감지 상태(on:1 / off:0)
111	Int	로봇 장치 오류 발견 상태 0: 오류 없음 1: PVL 에러 2: CPU 에러 3: Big 에러 4: Input 에러 5: JAM 에러 6: Over Current 에러 7: 관절 각도 에러 8: 제어 모드 에러 9: 레퍼런스 & 엔코더 오차 에러 10: 상위제어기 전류 에러 11: 온도 에러 12: 직접교시 속도 초과 에러
112	Int	자가 충돌 감지 상태(on:1 / off:0)
113	Int	로봇 일시정지 상태(on:1 / off:0)
114	Int	로봇 동작 오류 발견 상태 0: 오류 없음 1: 로봇이 쭉 뻗은 상태에서 TCP 명령 들어옴 2: TCP 명령의 한계 도달 3: 관절 각도 명령의 한계 도달 4: 해가 없는 TCP 명령
115	Int	제어반 디지털 신호 16 번 포트 입력값(on:1 / off:0)
116	Int	제어반 디지털 신호 17 번 포트 입력값(on:1 / off:0)
117	Int	Inbox 0 Trap 발생
118	Int	Inbox 1 Trap 발생
119	Int	Inbox 0 체크 모드
120	Int	Inbox 1 체크 모드

E.2 예제 프로그램 개발환경

본 예제는 Debian 9.8 버전과 Ubuntu 18.04 에서 테스트되었습니다. 유사한 계열의 Linux 시스템에서도 동작 가능할 수 있습니다. 별도의 커널 패치는 필요하지 않습니다.

프로그래밍을 작성하는 IDE(Integrated Development Environment)로는 Qt 5.8 버전 (https://www.qt.io)를 사용합니다.



주의

주의:

Qt 기반의 C++ 예제, Visual Studio 기반의 C# 예제 등이 있습니다. 예제 프로그램은 제조사 또는 대리점을 통해 받을 수 있습니다.

E.3 프로그래밍 방법

본 예제에는 태블릿 UI(User Interface)가 제공하는 모든 기능이 포함된 것이 아닙니다. 외부 제어를 통해 로봇을 움직이는 동안에 사용자가 모니터링하면 유용한 정보들에 대해서만 구현되어 있습니다.

예제를 실행시켰을 때 다음 그림과 같은 UI 가 실행됩니다.

각 항목별 기능은 아래와 같습니다.

1) 네트워크 연결

RB Series 로봇의 컨트롤 박스에 있는 랜(LAN) 포트를 통해서 로봇의 메인 제어기에 접속합니다. 외부 제어를 위한 기본 IP 주소는 '10.0.2.7'로 고정되어 있습니다. 외부 제어 명령어를 받기 위한 서버는 포트 5000 으로 접속하고, 로봇의 상태 정보를 요청하고 받기 위한 서버는 포트 5001 로 접속합니다. 각각을 접속하기 위한 버튼은 따로 존재하고, 접속에 성공하면 버튼에 적혀 있는 'Connect'라는 단어가 'Disconnect'로 변경됩니다. 접속이 끊겼을 때에는 반대로 됩니다.

2) 로봇 초기화

로봇의 메인 제어기에 접속이 된 상태에서 'Cobot Init'이 적혀 있는 버튼을 누르면 로봇이 초기화 과정을 수행합니다. 로봇은 'Power Set', 'Device Set', 'System Set', 그리고 'Robot Init'이라는 과정을 순차적으로 거칩니다. 각 과정을 수행 중일때에는 각 과정의 앞에 있는 흰색 에디트 박스가 노란색으로 바뀝니다. 수행이 완료된 과정은 녹색으로 바뀌고, 아직 수행하지 못한 과정은 빨간색으로 유지됩니다. 모든 4 개의 박스가 초록색으로 바뀌면 로봇의 초기화 과정이 완료된 상태로, 동작을 수행할 준비가 됩니다.

3) 로봇 상태

로봇의 상태는 컨트롤 박스에 있는 메인 프로그램으로부터 전달받는 데이터로부터 알 수 있습니다. 이 데이터는 포트 5001 로 'reqdata'를 요청하면 메인 프로그램에서 응답으로 보내줍니다. 데이터의 형식은 'CommonHeader.h'의 'systemSTAT' 구조체의 형태로 전달됩니다.

run mode: 로봇의 동작 모드를 표시합니다. 리얼 모드와 시뮬레이션 모드가 있습니다. 리얼 모드에서는 동작 명령이 실제로 로봇에 인가되어 로봇이 움직입니다. 시뮬레이션 모드에서는 동작을 수행하기는 하지만 그 명령이 로봇에는 전달되지 않습니다. 태블릿이 연결되어 있다면 반투명한 로봇이 움직이는 모습을 볼 수 있습니다. 로봇 동작 모드는 'systemSTAT'의 'program_mode' 변수 값으로 대변됩니다. 이 변수의 값이 0 이면 리얼 모드이고, 1 이면 시뮬레이션 모드입니다.

robot state: 로봇이 현재 움직이는 중인지 아니면 동작 명령을 받을 수 있는 상태인지 표시합니다. 로봇상태는 'systemSTAT'의 'robot_state' 변수값으로 부터 알 수 있습니다. 만약 이 값이 1 이면 동작 명령을 받을 수 있는 idle 상태이고, 3 이면 로봇이 움직이고 있는 상태입니다. 로봇이 움직이고 있는 동안에는 동작 명령어는 무시됩니다. 만약 'robot_state'값이 2 이면 로봇이 불특정한 이유로 동작이 중단된 상태이거나 일시정지 명령을 통해 멈춰 있는 상태입니다. 이때에는 'status'항목의 paused 로 표시됩니다.

status: 현재 로봇의 특수 동작 상태 혹은 이상 상태를 표시합니다. 직접 교시 중이면 'teaching', 외부 충돌 감지로 인해 정지된 상태면 'ext. collision', 동작 중에 자가 충돌 직전이면 'self collision', 일시정지 명령에 의해 멈춘 상태면 'paused', 로봇 제어 알고리즘 상의 솔루션이 없는 입력이 들어오면 'ems', 전원 문제나 로봇 제어에 문제가 발생하면 'sos' 에디트 창의 색이 변합니다. 'systemSTAT'의 'op_stat_collision_occur', 'op_stat_sos_flag', 'op_stat_self_collision', 'op_stat_soft_estop_occur', 'op_stat_ems_flag' 와 'robot_state' 값을 참조하면 표시됩니다.

joint reference: 각 관절의 레퍼런스 입력 값을 표시합니다(degree 단위).

joint encoder: 각 관절의 현재 엔코더 값을 표시합니다(degree 단위).

TCP reference: TCP 의 레퍼런스 위치 값을 표시합니다(mm 와 degree 단위).

digital in: 컨트롤 박스의 디지털 입력 값을 표시합니다.

digital out: 컨트롤 박스의 디지털 출력 값을 표시합니다.

analog in: 컨트롤 박스의 아날로그 입력 값을 표시합니다(voltage 단위).

analog out: 컨트롤 박스의 아날로그 출력 값을 표시합니다(voltage 단위).

tool out voltage: 현재 설정된 툴 플랜지 보드의 출력 전압을 표시합니다(0V, 12V, 24V 중 택일 1).

tool digital in: 툴 플랜지 보드의 디지털 입력 값을 표시합니다.

tool digital out: 툴 플랜지 보드의 디지털 출력 값을 표시합니다.

tool analog in: 툴 플랜지 보드의 아날로그 입력 값을 표시합니다.

4) 모드변환

로봇은 시뮬레이션 모드와 리얼 모드의 두가지 동작 모드를 가질 수 있습니다. 시뮬레이션 모드에서는 로봇은 움직이지 않지만 입력 레퍼런스의 값은 변경할 수 있습니다. 리얼 모드에서는 사용자의 입력에 따라 로봇이 실제로 움직입니다. 'Real'과 'Simulation'이 적힌 버튼을 누름에 따라서 로봇의 동작 모드를 변환할 수 있습니다. 로봇의 초기화 과정을 마친 직후에는 시뮬레이션 모드입니다.

5) 속도조절

로봇 동작의 전체적인 속도를 조절합니다. 슬라이더바를 옮겨서 0%에서 100%사이에서 조절할 수 있습니다. 이 속도는 로봇의 동작 명령에 주어지는 속도에 곱해져서 작용합니다.

6) 모션 정지 및 재개

로봇이 동작하는 중간에 일시정지를 하거나 멈출 수 있습니다. 일시정지를 하기 위해서는 'Motion Pause' 버튼을 누르고, 모션을 완전히 멈추기 위해서는 'Motion Halt'를 누릅니다. 'Motion Halt'의 경우에는 로봇이 급정지 하기 때문에 로봇을 안정적으로 사용하기 위해서는 일시정지를 먼저 사용하시는 것이 좋습니다. 로봇이 일시정지 상태일 경우에는 로봇에게 다른 동작 명령을 주더라도 수행되지 않습니다. 현재 동작을 멈추고 다른 동작을 하기 위해서는 'Motion Pause' 버튼 이후에 'Motion Halt' 버튼을 통해서 현재 동작을 완전히 끝내야 합니다.

반대로 일시정지 했거나 외부 충돌 감지로 인해서 멈춘 모션을 재개할 수 있습니다. 일시정지를 했던 모션을 재개하기 위해서는 'Motion Resume' 버튼을 누르고, 외부 충돌 감지로 인해 멈춘 모션을 재개하기 위해서는 'Collision Resume' 버튼을 누릅니다.

7) 디버깅 메시지 창

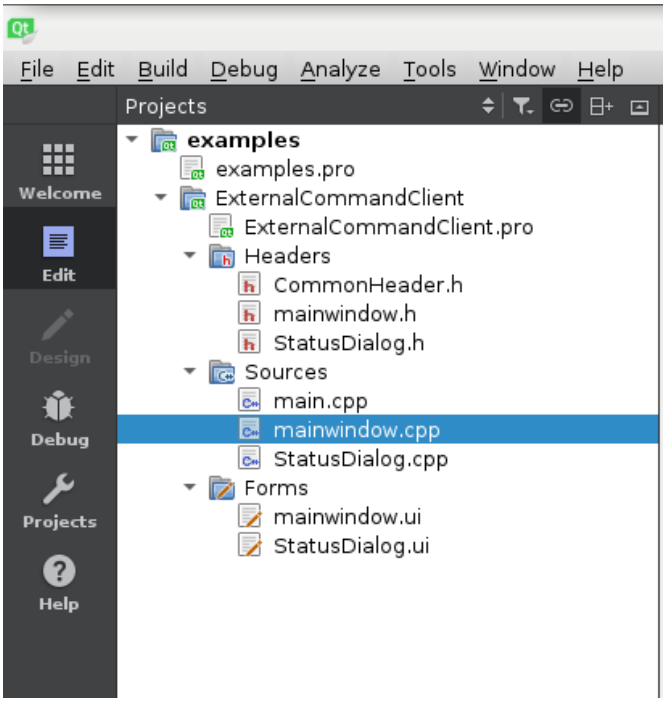
디버깅을 위한 메시지를 띄울 수 있는 창입니다.

8) 테스트 모션

'Motion Test'버튼을 누르면 기본으로 짜인 세가지 동작을 연속으로 수행합니다. 주변에 환경을 고려해서 실행하시기 바랍니다.

'Get Joint and TCP'버튼을 누르면 바로 옆의 에디트 창에 현재 로봇의 관절의 레퍼런스 각도와 TCP 값을 ',' 구분자로 표현합니다. 로봇 동작 시퀀스를 프로그램에 코딩할 때 이 값을 복사해서 사용하면 도움이 됩니다.

본 예제는 GUI 를 포함한 단일 프로세스 예제입니다. Qt 에서는 GUI 를 손쉽게 배치하고 버튼 클릭과 같은 이벤트를 생성하고 프로그램과 연결할 수 있습니다. 'mainwindow.ui'를 참조하시기 바랍니다.



예제의 핵심 내용은 'mainwindow.cpp'와 'mainwindow.h'에 포함되어 있습니다. 'CommonHeader.h'에는 로봇 상태 데이터의 형태를 확인할 수 있습니다.

```
    // Cobot Control API -------------------
    // <CobotInit>
    // : initialize Cobot
    void CobotInit();

    // <MoveJoint>
    // : move to target posture in joint coordinate
    // joint1~joint6 : target joint angle in deg unit
    // spd : speed parameter (0~1: user define   or   -1: default setting)
    // acc : acceleration parameter (0~1: user define   or   -1: default setting)
    void MoveJoint(float joint1, float joint2, float joint3, float joint4, float joint5, flo

    // <MoveTCP>
    // : move to target posture in cartesian coordinate
    // x, y, z : target TCP(tool center point) position in mm unit
    // rx, ry, rz : target TCP orientation (Yaw-Pitch-Roll Euler angle) in degree unit
    // spd : speed parameter (0~1: user define   or   -1: default setting)
    // acc : acceleration parameter (0~1: user define   or   -1: default setting)
    void MoveTCP(float x, float y, float z, float rx, float ry, float rz, float spd = -1, fl

    // <ControlBoxDigitalOut>
    // control digital out ports in control box
    // d0~d15 : digital out value (0 or 1)
    void ControlBoxDigitalOut(int d0, int d1, int d2, int d3, int d4, int d5, int d6, int d7

    // <ControlBoxAnalogOut>
    // control analog out ports in control box
    // a0~a3 : analog out value in voltage unit (0~10)
    void ControlBoxAnalogOut(float a0, float a1, float a2, float a3);

    // <ToolOut>
    // control digital out ports and voltage level in tool flange board
    // volt : reference voltage of tool flange board in voltage unit(0, 12, 24)
    // d0, d1 : digital out value (0 or 1)
    void ToolOut(int volt, int d0, int d1);

    // <ProgramMode_Real>
    // change to 'real robot' mode -- robot will move
    void ProgramMode_Real();

    // <ProgramMode_Simulation>
    // change to 'simulation' mode -- robot will not move except teaching
    void ProgramMode_Simulation();

    // <BaseSpeedChange>
    // change base speed -- base speed will be multiplied to motion velocity
    // spd : normalized base speed (0~1)
    void BaseSpeedChange(float spd);

    // <MotionPause>
    // pause the current motion
    void MotionPause();

    // <MotionHalt>
    // halt the current motion
    // !! CAUTION : user would better escape the motion sequence
    //              : if not, the next motion will be activated immediately
    void MotionHalt();

    // <MotionResume>
    // resume the paused motion
    void MotionResume();

    // <CollisionResume>
    // resume the motion which is paused due to external collision
    void CollisionResume();
    // -------------------------------------
```

사용자가 사용할 수 있는 로봇 제어 명령어는 위와 같이 ‘mainwindow.h’에 명시되어 있습니다. 자세한 설명은 아래와 같습니다.

함수	CobotInit(void)
스크립트	“mc jall init”
설명	로봇의 하드웨어를 초기화합니다. 초기화 과정은 ‘systemSTAT’의 ‘init_stat_info’ 와 ‘init_error’를 통해서 알 수 있습니다.

함수	MoveJoint(float joint1, float joint2, float joint3, float joint4, float joint5, float joint6, float spd = -1, float acc = -1);
스크립트	“jointall spd, acc, joint1, joint2, joint3, joint4, joint5, joint6”
설명	입력된 관절 값을 목표로 Joint Space 에서 움직입니다. joint1~joint6 은 목적 관절 값이고, degree 단위입니다. spd 와 acc 는 각각 속도와 가속도로 0~1 로 정규화 된 값을 가집니다. -1 을 입력할 경우에는 기본 세팅 된 속도와 가속도가 이용됩니다.

함수	MoveTCP(float x, float y, float z, float rx, float ry, float rz, float spd = -1, float acc = -1);
스크립트	“movetcp spd, acc, x, y, z, rx, ry, rz”
설명	입력된 TCP 의 위치와 회전각 값을 목표로 Cartesian Space 에서 움직입니다. x, y, z 는 TCP 의 위치 값으로 mm 단위입니다. rx, ry, rz 는 TCP 의 방향을 결정하기 위한 Yaw-Pitch-Roll 순서의 오일러 각도를 degree 단위로 표시한 값입니다. spd 와 acc 는 각각 속도와 가속도로 0~1 로 정규화 된 값을 가집니다. -1 을 입력할 경우에는 기본 세팅 된 속도와 가속도가 이용됩니다.

함수	MoveCircle_ThreePoint(int type, float x1, float y1, float z1, float rx1, float ry1, float rz1, float x2, float y2, float z2, float rx2, float ry2, float rz2, float spd = -1, float acc = -1);
스크립트	“movecircle threepoints intended spd, acc, x1, y1, z1, rx1, ry1, rz1, x2, y2, z2, rx2, ry2, rz2” “movecircle threepoints constant spd, acc, x1, y1, z1, rx1, ry1, rz1, x2, y2, z2, rx2, ry2, rz2” “movecircle threepoints radial spd, acc, x1, y1, z1, rx1, ry1, rz1, x2, y2, z2, rx2, ry2, rz2”
설명	TCP 의 현재위치, 중간 위치, 그리고 최종 위치의 3 가지 포인트를 이용해서 원을 그리는 동작 명령입니다. type 에 따라서 TCP 의 방향을 3 가지로 결정할 수 있습니다. type=0: 스크립트에는 ‘intended’로 사용됩니다. 사용자가 입력한 방향을 지키면서 움직입니다. type=1: 스크립트에는 ‘constant’로 사용됩니다. TCP 의 현재 방향을 유지한 채로 원을 그립니다. type=2: 스크립트에는 ‘radial’로 사용됩니다. 세개의 위치로 결정되는 원의 중심방향으로 TCP 의 방향이 상대적으로 변합니다. 방향의 기준은 현재 TCP 의 방향이 시작 기준입니다. x1, y1, z1 는 TCP 의 중간 위치 값으로 mm 단위입니다. rx1, ry1, rz1 는 TCP 의 중간 위치의 방향을 결정하기 위한 Yaw-Pitch-Roll 순서의 오일러 각도를 degree 단위로 표시한 값입니다. x2, y2, z2 는 TCP 의 최종 위치 값으로 mm 단위입니다. rx2, ry2, rz2 는 TCP 의 최종 위치의 방향을 결정하기 위한 Yaw-Pitch-Roll 순서의 오일러 각도를 degree 단위로 표시한 값입니다. spd 와 acc 는 각각 속도와 가속도로 0~1 로 정규화 된 값을 가집니다. -1 을 입력할 경우에는 기본 세팅 된 속도와 가속도가 이용됩니다.

함수	MoveCircle_Axis(int type, float cx, float cy, float cz, float ax, float ay, float az, float rot_angle, float spd = -1, float acc = -1);
스크립트	“movecircle axis intended spd, acc, rot_angle, cx, cy, cz, ax, ay, az” “movecircle axis constant spd, acc, rot_angle, cx, cy, cz, ax, ay, az” “movecircle axis radial spd, acc, rot_angle, cx, cy, cz, ax, ay, az”
설명	TCP 의 현재위치에서 시작해서 회전 축을 중심으로 입력한 각도만큼 원을 그리는 동작 명령입니다. type 에 따라서 TCP 의 방향을 3 가지로 결정할 수 있습니다. type=0: 스크립트에는 ‘intended’로 사용됩니다. 사용자가 입력한 방향을 지키면서 움직입니다. type=1: 스크립트에는 ‘constant’로 사용됩니다. TCP 의 현재 방향을 유지한 채로 원을 그립니다. type=2: 스크립트에는 ‘radial’로 사용됩니다. 세개의 위치로 결정되는 원의 중심방향으로 TCP 의 방향이 상대적으로 변합니다. 방향의 기준은 현재 TCP 의 방향이 시작 기준입니다. cx, cy, cz 는 회전 중심점의 위치 값으로 mm 단위입니다. ax, ay, az 는 회전의 축을 나타내는 벡터입니다. rot_angle 은 회전할 양을 나타내는 값으로 degree 단위입니다. spd 와 acc 는 각각 속도와 가속도로 0~1 로 정규화 된 값을 가집니다. -1 을 입력할 경우에는 기본 세팅 된 속도와 가속도가 이용됩니다.

함수	MoveJointBlend_Clear(void);
스크립트	“blend_jnt clear_pt”
설명	Joint Space 에서 연속된 목적 위치들을 연결하는 동작 명령 중에서 저장된 목적 위치들을 지우는 명령어입니다.

함수	MoveJointBlend_AddPoint(float joint1, float joint2, float joint3, float joint4, float joint5, float joint6, float spd = -1, float acc = -1);
스크립트	“blend_jnt add_pt spd, acc, joint1, joint2, joint3, joint4, joint5, joint6”
설명	Joint Space 에서 연속된 목적 위치들을 연결하는 동작 명령 중에서 목적 위치들을 입력하는 명령어입니다. joint1~joint6 은 목적 관절 값이고, degree 단위입니다. spd 와 acc 는 각각 속도와 가속도로 0~1 로 정규화 된 값을 가집니다. -1 을 입력할 경우에는 기본 세팅 된 속도와 가속도가 이용됩니다. 마지막으로 입력된 spd 와 acc 로 전체 명령이 구동됩니다.

함수	MoveJointBlend_MovePoint(void);
스크립트	“blend_jnt move_pt”
설명	Joint Space 에서 연속된 목적 위치들을 연결하는 동작 명령 중에서 동작을 시작하는 명령어입니다. MoveJointBlend_AddPoint 로 입력한 위치들로 차례대로 움직이는 동작이 수행됩니다.

함수	MoveTCPBlend_Clear(void);
스크립트	“blend_tcp clear_pt”
설명	Cartesian Space 에서 연속된 목적 위치들을 연결하는 동작 명령 중에서 저장된 목적 위치들을 지우는 명령어입니다.

함수	MoveTCPBlend_AddPoint(float radius, float x, float y, float z, float rx, float ry, float rz, float spd = -1, float acc = -1);
스크립트	“blend_tcp add_pt spd, acc, radius, x, y, z, rx, ry, rz”
설명	Cartesian Space 에서 연속된 목적 위치들을 연결하는 동작 명령 중에서 목적 위치들을 입력하는 명령어입니다. radius 는 목적 위치들을 연달아 이동할 때 곡선을 그리며 다음 목적 위치로 이동하기 위한 거리를 나타낸 값으로 mm 단위입니다. 하나의 목적 위치에서 다음 목적 위치로의 이동을 선분으로 생각할 때 그 절반 값이 radius 가 가질 수 있는 최대 값입니다. 이 값보다 크게 주면 내부적으로 알아서 값을 줄여서 사용합니다. x, y, z 는 TCP 의 위치 값으로 mm 단위입니다. rx, ry, rz 는 TCP 의 방향을 결정하기 위한 Yaw-Pitch-Roll 순서의 오일러 각도를 degree 단위로 표시한 값입니다. spd 와 acc 는 각각 속도와 가속도로 0~1 로 정규화 된 값을 가집니다. -1 을 입력할 경우에는 기본 세팅 된 속도와 가속도가 이용됩니다. 마지막으로 입력된 spd 와 acc 로 전체 명령이 구동됩니다.

함수	MoveTCPBlend_MovePoint(void);
스크립트	“blend_tcp move_pt”
설명	Cartesian Space 에서 연속된 목적 위치들을 연결하는 동작 명령 중에서 동작을 시작하는 명령어입니다. MoveTCPBlend_AddPoint 로 입력한 위치들로 차례대로 움직이는 동작이 수행됩니다.

함수	ControlBoxDigitalOut(int d0, int d1, int d2, int d3, int d4, int d5, int d6, int d7, int d8, int d9, int d10, int d11, int d12, int d13, int d14, int d15)
스크립트	“digital_out d0, d1, d2, d3, d4,d5, d6, d7, d8, d9, d10, d11, d12, d13, d14, d15”
설명	컨트롤 박스의 디지털 출력 포트를 통해서 신호를 내보냅니다. d0~d15 는 각각의 디지털 포트의 on/off 를 나타냅니다. 값이 1 이면 on 이고, 0 이면 off 입니다.

함수	ControlBoxAnalogOut(float a0, float a1, float a2, float a3)
스크립트	“analog_out a0, a1, a2, a3”
설명	컨트롤 박스의 아날로그 출력 포트를 통해서 신호를 내보냅니다. a0~a3 은 각각의 아날로그 포트의 출력 값을 나타냅니다. 값은 0 에서 10 의 값을 가지며 voltage 단위입니다.

함수	ToolOut(int volt, int d0, int d1)
스크립트	“tool_out volt, d0, d1”
설명	툴 플랜지의 전압 레벨을 설정하고, 디지털 출력 포트를 통해서 신호를 내보냅니다. volt 는 툴 플랜지의 디지털 출력의 전압 레벨을 설정합니다. 값은 0, 12, 24 가 가능하고 voltage 단위입니다. 이 세개의 값 이외의 입력은 무시됩니다. d0, d1 은 각각의 디지털 포트의 on/off 를 나타냅니다. 값이 1 이면 on 이고, 0 이면 off 입니다.

함수	ProgramMode_Real(void)
스크립트	“pgmode real”
설명	로봇의 동작 상태를 리얼 모드로 변경합니다. 리얼 모드에서는 동작 명령을 내릴 때 실제로 로봇이 움직입니다.

함수	ProgramMode_Simulation(void)
스크립트	“pgmode simulation”
설명	로봇의 동작 상태를 시뮬레이션 모드로 변경합니다. 시뮬레이션 모드에서는 동작 명령을 내릴 때 로봇이 움직이지는 않습니다.

함수	BaseSpeedChange(float spd)
스크립트	“sdw default_speed spd”
설명	입력된 spd 에 모든 동작 명령의 속도가 곱해져서 동작합니다. 즉, 전체 동작의 속도를 결정하는 값입니다. spd 는 0~1 로 정규화 된 값을 가집니다.

함수	MotionPause(void)
스크립트	“task pause”
설명	로봇 동작을 일시정지 시킵니다. MotionResume 으로 동작을 재개시켜 주거나 MotionHalt 로 동작을 완전히 끝내기 전에는 다른 동작 명령을 주어도 무시됩니다.

함수	MotionResume(void)
스크립트	“task resume_a”
설명	MotionPause 로 인해 일시 정지된 로봇 동작을 재개합니다. 외부 충돌로 인해 정지된 동작을 재개시키지는 못합니다. 이 경우에는 CollisionResume 을 수행해야 합니다.

함수	CollisionResume(void)
스크립트	“task resume_b”
설명	외부 충돌로 인해 일시 정지된 로봇 동작을 재개합니다. MotionPause 로 인해 정지된 동작을 재개시키지는 못합니다. 이 경우에는 MotionResume 을 수행해야 합니다.

함수	MotionHalt(void)
스크립트	“task stop”
설명	로봇의 동작을 완전히 멈춥니다. 또한 MotionPause 혹은 외부 충돌로 인해 일시 정지된 로봇의 동작을 완전히 멈춥니다.

```
int test_flag = false;
int test_state = 0;
void MainWindow::onLogic(){

    // Motion Sequence ================================================
    if(test_flag == true){
        switch(test_state){
        case 0:
            if(IsMotionIdle()){
                print("test_state 0\n");
                MoveJoint(0,0,0,0,0,0);
                test_state = 1;
            }
            break;

        case 1:
            if(IsMotionIdle()){
                print("test_state 1\n");
                MoveJoint(50,50,50,50,50,50);
                test_state = 2;
            }
            break;
        case 2:
            if(IsMotionIdle()){
                print("test_state 2\n");
                MoveTCP(400,400,400,150,30,-100);
                test_state = 3;
            }
            break;
        case 3:
            if(IsMotionIdle()){
                print("test_state 3\n");
                test_flag = false;
                test_state = 0;
            }
            break;
        }
    }
    // ================================================================

}
```

위의 코드는 두개의 관절 제어 모션과 하나의 TCP 제어 모션을 순차적으로 수행하는 동작 코드입니다. 동작 시퀀스를 수행할지 말지 결정하는 'test_flag'가 있고, 만약 이 값이 참일 경우에 시퀀스의 진행을 나타내는 'test_state'값에 따라서 이전 모션에서 다음 모션으로 순차적으로 넘어 갑니다.

이때 이전 모션이 끝났는지 끝나지 않았는지 확인을 해야 하고, 이를 쉽게 하기 위해 제공하는 'IsMotionIdle' 함수가 있습니다. 이 함수는 명령어를 로봇의 메인 제어기로 보내고 응답 메시지의 수신 여부와 현재 로봇 상태가 'Idle'인지 'Moving'인지 여부를 확인합니다.

```
int MainWindow::IsMotionIdle(){
   return ((cmdConfirmFlag == true) && (systemStat.sdata.robot_state == 1));
}
```

동작 시퀀스가 들어있는 'onLogic' 함수는 Qt 에서 제공하는 타이머에 연동되어 있는 함수로, 예제에서는 10ms 주기로 설정되어, 매 10ms 마다 이 함수가 실행됩니다.

```
409
410 void MainWindow::on_BTN_TEST_clicked()
411 {
412     test_state = 0;
413     test_flag = true;
414 }
415
```

로봇 동작을 실행시키는 부분은 간단합니다. 동작 시퀀스 상태를 나타내는 'test_state' 값을 동작의 시작 지점인 0 으로 세팅하고, 동작을 수행하는 'test_flag' 값을 true 로 주면 로봇이 움직입니다.

제공된 예제 코드에서 보여주는 동작은 매우 단순하지만 구조에 얽매임이 없이 자유롭습니다. 사용자는 이 예제 코드를 기반으로 원하는 어플리케이션을 구현할 수도 있고, 스크립트만 이해해서 별도의 어플리케이션을 구현할 수도 있습니다.