산업인공지능개론

Mini Project #1

2023254006
이선경

CONTENS

01. 현업 업무 소개

비행체 시뮬레이터 소프트웨어 개발

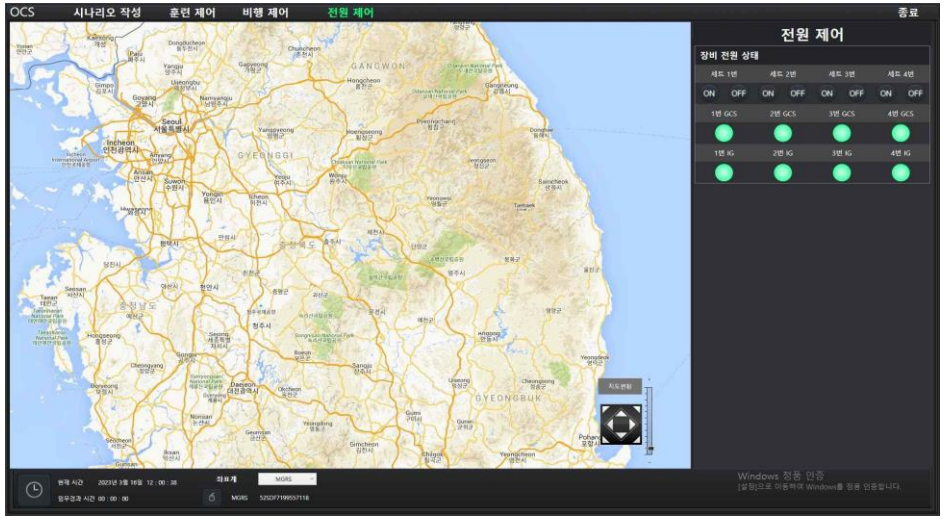
Prepar3D

- 비행 시뮬레이터

MFD

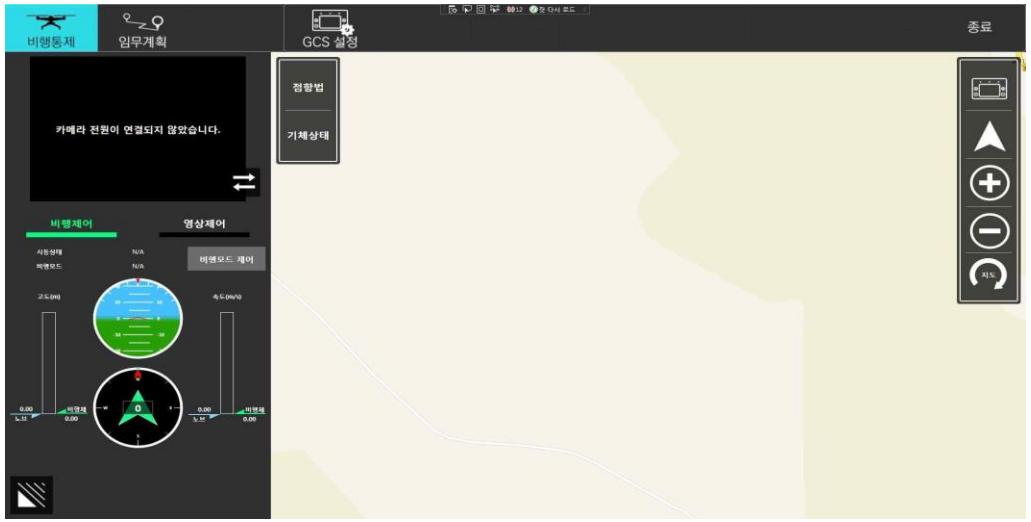
- 항공용 다기능 전시기

01. 현업 업무 소개



운용통제 컴퓨터

- 비행 제어, 초기 위치 시나리오 작성 등



지상통제 컴퓨터

- 비행모드 제어, 임무계획 작성 등

02. 코드 구현

1. durable_rules 설치

```
C:₩Users₩sk>pip install durable_rules
Requirement already satisfied: durable_rules in c:₩users₩sk₩appdata₩local₩programs₩python₩python310₩lib₩site-packages (2.0.28)
```

```
C:₩Users₩sk>pip list
Package       Version
------------- -------
durable-rules 2.0.28
pip           22.2.1
setuptools    63.2.0
```

02. 코드 구현

2. P3D 관련 규칙

```
from durable.lang import *

with ruleset('p3d'):
    @when_all(c.first << (m.predicate == '걸리지 않는다') & (m.object == '시동이'),
        (m.predicate == '걸린다') & (m.object == '시동이') & (m.subject == c.first.subject))
    def 시동데이터추가(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '추가한다','object': '시동 데이터를'})

    @when_all(c.first << (m.predicate == '조종기와 반대로 작동된다') & (m.object == '기체가'),
        (m.predicate == '조종기와 동일하게 작동된다') & (m.object == '기체가') & (m.subject == c.first.subject))
    def 부호수정(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '수정한다','object': '부호를'})

    @when_all(c.first << (m.predicate == '반응속도가 느리다') & (m.object == '기체의'),
        (m.predicate == '반응속도가 빨라진다') & (m.object == '기체의') & (m.subject == c.first.subject))
    def 고도수정(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '낮춘다','object': '고도를'})

    @when_all(c.first << (m.predicate == '인식하지 못한다') & (m.object == '기체를'),
        (m.predicate == '인식한다') & (m.object == '기체를') & (m.subject == c.first.subject))
    def 기체파일추가(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '추가해준다','object': '기체 파일을'})

    @when_all(c.first << (m.predicate == '기체의 데이터를 못받아온다') & (m.object == 'VR이'),
        (m.predicate == '기체의 데이터를 받아온다') & (m.object == 'VR이') & (m.subject == c.first.subject))
    def 기체데이터추가(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '추가해준다','object': '기체의 데이터를'})

    @when_all(c.first << (m.predicate == 'Rudder가 제대로 작동하지 않는다') & (m.object == '기체의'),
        (m.predicate == 'Rudder가 제대로 작동한다') & (m.object == '기체의') & (m.subject == c.first.subject))
    def Rudder데이터확인(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '데이터 값이 잘 들어오는 지 확인한다','object': '기체의'})

    @when_all(c.first << (m.predicate == 'VR이 보이지 않는다') & (m.object == 'p3d에'),
        (m.predicate == 'VR이 보인다') & (m.object == 'p3d에') & (m.subject == c.first.subject))
    def VR소프트웨어설치(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '설치한다','object': 'VR 소프트웨어를'})

    @when_all(+m.subject)
    def output(c):
        print('Fact: {0} {1} {2}'.format(c.m.subject, c.m.object, c.m.predicate))
```

02. 코드 구현

2. P3D 관련 규칙

```
assert_fact('p3d', {'subject':'시동불량: ', 'predicate': '걸리지 않는다', 'object': '시동이' })
assert_fact('p3d', {'subject':'시동불량: ', 'predicate': '걸린다', 'object': '시동이' })
assert_fact('p3d', {'subject':'기체반대반응: ', 'predicate': '조종기와 반대로 작동된다', 'object': '기체가' })
assert_fact('p3d', {'subject':'기체반대반응: ', 'predicate': '조종기와 동일하게 작동된다', 'object': '기체가' })
assert_fact('p3d', {'subject':'반응속도: ', 'predicate': '반응속도가 느리다', 'object': '기체의' })
assert_fact('p3d', {'subject':'반응속도: ', 'predicate': '반응속도가 빨라진다', 'object': '기체의' })
assert_fact('p3d', {'subject':'기체인식불가: ', 'predicate': '인식하지 못한다', 'object': '기체를' })
assert_fact('p3d', {'subject':'기체인식불가': , 'predicate': '인식한다', 'object': '기체를' })
assert_fact('p3d', {'subject':'VR기체오류: ', 'predicate': '기체의 데이터를 못받아온다', 'object': 'VR이' })
assert_fact('p3d', {'subject':'VR기체오류: ', 'predicate': '기체의 데이터를 받아온다', 'object': 'VR이' })
assert_fact('p3d', {'subject':'Rudder오류: ', 'predicate': 'Rudder가 제대로 작동하지 않는다', 'object': '기체의' })
assert_fact('p3d', {'subject':'Rudder오류: ', 'predicate': 'Rudder가 제대로 작동한다', 'object': '기체의' })
assert_fact('p3d', {'subject':'p3d_VR사용불가: ', 'predicate': 'VR이 보이지 않는다', 'object': 'p3d에' })
assert_fact('p3d', {'subject':'p3d_VR사용불가: ', 'predicate': 'VR이 보인다', 'object': 'p3d에' })
```

```
Fact: 시동불량:  시동이 걸리지 않는다
Fact: 시동불량:  시동 데이터를 추가한다
Fact: 시동불량:  시동이 걸린다
Fact: 기체반대반응:  기체가 조종기와 반대로 작동된다
Fact: 기체반대반응:  부호를 수정한다
Fact: 기체반대반응:  기체가 조종기와 동일하게 작동된다
Fact: 반응속도:  기체의 반응속도가 느리다
Fact: 반응속도:  고도를 낮춘다
Fact: 반응속도:  기체의 반응속도가 빨라진다
Fact: 기체인식불가:  기체를 인식하지 못한다
Fact: 기체인식불가:  기체 파일을 추가해준다
Fact: 기체인식불가:  기체를 인식한다
Fact: VR기체오류:  VR이 기체의 데이터를 못받아온다
Fact: VR기체오류:  기체의 데이터를 추가해준다
Fact: VR기체오류:  VR이 기체의 데이터를 받아온다
Fact: Rudder오류:  기체의 Rudder가 제대로 작동하지 않는다
Fact: Rudder오류:  기체의 데이터 값이 잘 들어오는 지 확인한다
Fact: Rudder오류:  기체의 Rudder가 제대로 작동한다
Fact: p3d_VR사용불가:  p3d에 VR이 보이지 않는다
Fact: p3d_VR사용불가:  VR 소프트웨어를 설치한다
Fact: p3d_VR사용불가:  p3d에 VR이 보인다
```

02. 코드 구현

3. 지상통제 관련 규칙

```
with ruleset('지상통제'):
    @when_all(c.first << (m.predicate == '받아오지 못한다') & (m.object == '해당 조종기의 데이터를'),
        (m.predicate == '잘 받아온다') & (m.object == '해당 조종기의 데이터를') & (m.subject == c.first.subject))
    def 장치명추가(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '코드에 추가한다','object': '조종기의 장치명을'})

    @when_all(c.first << (m.predicate == '저장이 안된다') & (m.object == '시나리오가'),
        (m.predicate == '정상적으로 저장된다') & (m.object == '시나리오가') & (m.subject == c.first.subject))
    def 시나리오경로(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '확인한다','object': '시나리오 저장되는 폴더와 경로를'})

    @when_all(c.first << (m.predicate == '기체의 데이터를 읽어오지 못한다') & (m.object == '지상통제 컴퓨터에'),
        (m.predicate == '기체의 데이터가 도시된다') & (m.object == '지상통제 컴퓨터에') & (m.subject == c.first.subject))
    def 기체연결확인(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '확인한다','object': '기체의 연결상태를'})

    @when_all(c.first << (m.predicate == '저장되지 않는다') & (m.object == '임무 작성 시 경로점이'),
        (m.predicate == '정상적으로 저장된다') & (m.object == '임무 작성 시 경로점이') & (m.subject == c.first.subject))
    def 경로점코드수정(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '코드를 수정한다','object': '경로점이 저장되는'})

    @when_all(c.first << (m.predicate == '전송이 안 된다') & (m.object == '기체에 임무가'),
        (m.predicate == '정상적으로 전송된다') & (m.object == '기체에 임무가') & (m.subject == c.first.subject))
    def 기체연결확인(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '연결 되었는지 확인한다','object': '기체가 정상적으로'})

    @when_all(c.first << (m.predicate == '정상적으로 작동되지 않는다') & (m.object == '노브 모드가'),
        (m.predicate == '정상적으로 작동된다') & (m.object == '노브 모드가') & (m.subject == c.first.subject))
    def 노브PORT확인(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '연결 및 PORT 번호를 확인한다','object': '노브의'})

    @when_all(c.first << (m.predicate == '정항점으로 이동하지 않는다') & (m.object == '정항법 모드 시 기체가'),
        (m.predicate == '잘 작동된다') & (m.object == '정항법 모드 시 기체가') & (m.subject == c.first.subject))
    def 정항법코드수정(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '코드를 수정한다','object': '정항법 모드의'})

    @when_all(c.first << (m.predicate == '회전익 기체의 모드로 도시된다') & (m.object == '고정익 기체 실행 시'),
        (m.predicate == '고정익 모드로 도시된다') & (m.object == '고정익 기체 실행 시') & (m.subject == c.first.subject))
    def 고정익코드수정(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '고정익으로 수정한다','object': '회전익 기체의 모드를'})

    @when_all(c.first << (m.predicate == '목록에 도시되지 않는다') & (m.object == '저장한 임무계획 파일이'),
        (m.predicate == '목록에 도시된다') & (m.object == '저장한 임무계획 파일이') & (m.subject == c.first.subject))
    def 임무계획목록수정(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '불러오는 코드를 수정한다','object': '임무계획 목록을'})

    @when_all(c.first << (m.predicate == '도시되지 않는다') & (m.object == '지도 데이터가'),
        (m.predicate == '도시된다') & (m.object == '지도 데이터가') & (m.subject == c.first.subject))
    def 지도경로확인(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '경로를 확인한다','object': '지도 데이터의'})

    @when_all(c.first << (m.predicate == '이미지가 표시되지 않는다') & (m.object == '지도에 기체의'),
        (m.predicate == '이미지가 잘 표시된다') & (m.object == '지도에 기체의') & (m.subject == c.first.subject))
    def 이미지경로확인(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '기체 이미지 경로를 확인한다','object': '지도 데이터의'})

    @when_all(c.first << (m.predicate == '통신이 안 된다') & (m.object == '지상통제 컴퓨터와 운용통제 컴퓨터 간'),
        (m.predicate == '통신이 정상적으로 이뤄진다') & (m.object == '지상통제 컴퓨터와 운용통제 컴퓨터 간') & (m.subject == c.first.subject))
    def IP_PORT확인(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': 'IP를 맞춰준다','object': 'PORT 번호와'})

    @when_all(c.first << (m.predicate == '반대로 나온다') & (m.object == 'MFD 화면이'),
        (m.predicate == '제대로 보인다') & (m.object == 'MFD 화면이') & (m.subject == c.first.subject))
    def MFD순서수정(c):
        c.assert_fact({'subject':c.first.subject, 'predicate': '순서를 수정한다','object': 'MFD 화면의'})

    @when_all(+m.subject)
    def output(c):
        print('Fact: {0} {1} {2}'.format(c.m.subject, c.m.object, c.m.predicate))
```

02. 코드 구현

3. 지상통제 관련 규칙

```
assert_fact('지상통제', {'subject':'조종기값확인: ', 'predicate': '받아오지 못한다', 'object': '해당 조종기의 데이터를' })
assert_fact('지상통제', {'subject':'조종기값확인: ', 'predicate': '잘 받아온다', 'object': '해당 조종기의 데이터를' })
assert_fact('지상통제', {'subject':'시나리오저장불가: ', 'predicate': '저장이 안된다', 'object': '시나리오가' })
assert_fact('지상통제', {'subject':'시나리오저장불가: ', 'predicate': '정상적으로 저장된다', 'object': '시나리오가' })
assert_fact('지상통제', {'subject':'기체데이터값오류: ', 'predicate': '기체의 데이터를 읽어오지 못한다', 'object': '지상통제 컴퓨터에' })
assert_fact('지상통제', {'subject':'기체데이터값오류: ', 'predicate': '기체의 데이터가 도시된다', 'object': '지상통제 컴퓨터에' })
assert_fact('지상통제', {'subject':'경로점저장오류: ', 'predicate': '저장되지 않는다', 'object': '임무 작성 시 경로점이' })
assert_fact('지상통제', {'subject':'경로점저장오류: ', 'predicate': '정상적으로 저장된다', 'object': '임무 작성 시 경로점이' })
assert_fact('지상통제', {'subject':'임무전송오류: ', 'predicate': '전송이 안 된다', 'object': '기체에 임무가' })
assert_fact('지상통제', {'subject':'임무전송오류: ', 'predicate': '정상적으로 전송된다', 'object': '기체에 임무가' })
assert_fact('지상통제', {'subject':'노브모드오류: ', 'predicate': '정상적으로 작동되지 않는다', 'object': '노브 모드가' })
assert_fact('지상통제', {'subject':'노브모드오류: ', 'predicate': '정상적으로 작동된다', 'object': '노브 모드가' })
assert_fact('지상통제', {'subject':'점항법오류: ', 'predicate': '점항점으로 이동하지 않는다', 'object': '점항법 모드 시 기체가' })
assert_fact('지상통제', {'subject':'점항법오류: ', 'predicate': '잘 작동된다', 'object': '점항법 모드 시 기체가' })
assert_fact('지상통제', {'subject':'고정익기체오류: ', 'predicate': '회전익 기체의 모드로 도시된다', 'object': '고정익 기체 실행 시' })
assert_fact('지상통제', {'subject':'고정익기체오류: ', 'predicate': '고정익 모드로 도시된다', 'object': '고정익 기체 실행 시' })
assert_fact('지상통제', {'subject':'임무계획목록오류: ', 'predicate': '목록에 도시되지 않는다', 'object': '저장한 임무계획 파일이' })
assert_fact('지상통제', {'subject':'임무계획목록오류: ', 'predicate': '목록에 도시된다', 'object': '저장한 임무계획 파일이' })
assert_fact('지상통제', {'subject':'지도데이터오류: ', 'predicate': '도시되지 않는다', 'object': '지도 데이터가' })
assert_fact('지상통제', {'subject':'지도데이터오류: ', 'predicate': '도시된다', 'object': '지도 데이터가' })
assert_fact('지상통제', {'subject':'기체이미지오류: ', 'predicate': '이미지가 표시되지 않는다', 'object': '지도에 기체의' })
assert_fact('지상통제', {'subject':'기체이미지오류: ', 'predicate': '이미지가 잘 표시된다', 'object': '지도에 기체의' })
assert_fact('지상통제', {'subject':'통신오류: ', 'predicate': '통신이 안 된다', 'object': '지상통제 컴퓨터와 운용통제 컴퓨터 간' })
assert_fact('지상통제', {'subject':'통신오류: ', 'predicate': '통신이 정상적으로 이뤄진다', 'object': '지상통제 컴퓨터와 운용통제 컴퓨터 간' })
assert_fact('지상통제', {'subject':'MFD화면오류: ', 'predicate': '반대로 나온다', 'object': 'MFD 화면이' })
assert_fact('지상통제', {'subject':'MFD화면오류: ', 'predicate': '제대로 보인다', 'object': 'MFD 화면이' })
```

02. 코드 구현

3. 지상통제 관련 규칙

```
Fact: 조종기값확인:  해당 조종기의 데이터를 받아오지 못한다
Fact: 조종기값확인:  조종기의 장치명을 코드에 추가한다
Fact: 조종기값확인:  해당 조종기의 데이터를 잘 받아온다
Fact: 시나리오저장불가:  시나리오가 저장이 안된다
Fact: 시나리오저장불가:  시나리오 저장되는 폴더와 경로를 확인한다
Fact: 시나리오저장불가:  시나리오가 정상적으로 저장된다
Fact: 기체데이터값오류:  지상통제 컴퓨터에 기체의 데이터를 읽어오지 못한다
Fact: 기체데이터값오류:  기체의 연결상태를 확인한다
Fact: 기체데이터값오류:  지상통제 컴퓨터에 기체의 데이터가 도시된다
Fact: 경로점저장오류:  임무 작성 시 경로점이 저장되지 않는다
Fact: 경로점저장오류:  경로점이 저장되는 코드를 수정한다
Fact: 경로점저장오류:  임무 작성 시 경로점이 정상적으로 저장된다
Fact: 임무전송오류:  기체에 임무가 전송이 안 된다
Fact: 임무전송오류:  기체가 정상적으로 연결 되었는지 확인한다
Fact: 임무전송오류:  기체에 임무가 정상적으로 전송된다
Fact: 노브모드오류:  노브 모드가 정상적으로 작동되지 않는다
Fact: 노브모드오류:  노브의 연결 및 PORT 번호를 확인한다
Fact: 노브모드오류:  노브 모드가 정상적으로 작동된다
Fact: 점항법오류:  점항법 모드 시 기체가 점항점으로 이동하지 않는다
Fact: 점항법오류:  점항법 모드의 코드를 수정한다
Fact: 점항법오류:  점항법 모드 시 기체가 잘 작동된다
Fact: 고정익기체오류:  고정익 기체 실행 시 회전익 기체의 모드로 도시된다
Fact: 고정익기체오류:  회전익 기체의 모드를 고정익으로 수정한다
Fact: 고정익기체오류:  고정익 기체 실행 시 고정익 모드로 도시된다
Fact: 임무계획목록오류:  저장한 임무계획 파일이 목록에 도시되지 않는다
Fact: 임무계획목록오류:  임무계획 목록을 불러오는 코드를 수정한다
Fact: 임무계획목록오류:  저장한 임무계획 파일이 목록에 도시된다
Fact: 지도데이터오류:  지도 데이터가 도시되지 않는다
Fact: 지도데이터오류:  지도 데이터의 경로를 확인한다
Fact: 지도데이터오류:  지도 데이터가 도시된다
Fact: 기체이미지오류:  지도에 기체의 이미지가 표시되지 않는다
Fact: 기체이미지오류:  지도 데이터의 기체 이미지 경로를 확인한다
Fact: 기체이미지오류:  지도에 기체의 이미지가 잘 표시된다
Fact: 통신오류:  지상통제 컴퓨터와 운용통제 컴퓨터 간 통신이 안 된다
Fact: 통신오류:  PORT 번호와 IP를 맞춰준다
Fact: 통신오류:  지상통제 컴퓨터와 운용통제 컴퓨터 간 통신이 정상적으로 이뤄진다
Fact: MFD화면오류:  MFD 화면이 반대로 나온다
Fact: MFD화면오류:  MFD 화면의 순서를 수정한다
Fact: MFD화면오류:  MFD 화면이 제대로 보인다
```