

[두산로보틱스] 지능형 로봇틱스 엔지니어

협동로봇을 활용한 굿즈 이미지 제작 자동화

협동1 - ROS2를 활용한 협동로봇 자동화 공정 시스템 구현

TEAM E-1조 아사E(Asahi)

[팀원] 김진환, 신동현, 하종우

[팀장] 김서희

[멘토] 김민수

목 차



- 01 프로젝트 개요
- 02 프로젝트 팀 구성 및 역할
- 03 프로젝트 수행 절차 및 방법
- 04 프로젝트 수행 결과
- 05 자체 평가 의견

01

K-Digital Training

프로젝트 개요

1

프로젝트 주제 및 선정 배경, 기획의도

주제: 굿즈용 이미지 제작

특화 포인트: 다양한 Doosan
Api사용, OpenCV활용
알고리즘으로 자동화된 그리기
과정

차별성: 캐릭터의 이미지
선택시 알고리즘을 사용해
원하는 도안 생성

2

프로젝트 내용

컨셉: 알고리즘을 통해
사용자가 원하는대로 로봇의
팔을 제어하여 캐릭터 도안
구현

훈련 연관성: ROS2를
활용하여 작성한 python
code로 협동로봇(m0609)

작동

3

활용 장비 및 재료

하드웨어: Doosan m0609

소프트웨어: Ubuntu 22.04,
Window, ROS2 Humble,
Python, Doosan API, OpenCV,
RViz2

4

프로젝트 구조

프로젝트 목적: 수정이 많은
캐릭터 도안을 로봇을
활용하여 노동력으로 초안
진행

진행 일정: 2025년 06월 09일 ~
2025년 06월 20일

5

활용방안 및 기대 효과









기대효과: 로봇 팔 기술의
발전으로 3D캐릭터 디자인
가능, 자동화된 도안작성 가능

실무 활용성: 글로벌
애니메이션 및 게임 산업에서의
대중화

02

K-Digital Training












프로젝트 팀(E-1) 구성 및 역할

훈련생	역할	담당 업무
김서희	팀장	 nudge 함수 구현  프로젝트의 방향 및 진행 상황 관리
김진환	팀원	 OpenCV image 좌표 추출 알고리즘 제작  rviz2 시뮬레이션
신동현	팀원	 input image data 제작  gui 제작
하종우	팀원	 좌표 추출 알고리즘 자료 조사 및 테스트  데스크 환경 구현

03

K-Digital Training

프로젝트 수행 절차 및 방법

구분	기간	활동	비고
프로젝트 이론 학습	06/09(월) ~ 06/12(목)	 DRL, Doosan API 이론 학습  Dart Studio code 작성	
프로젝트 아이디어 회의 및 계획 수립	06/12(목) ~ 06/13(금)	 프로젝트 주제 및 캐릭터 선정  사용 알고리즘 조사 및 구현	중간 발표
캐릭터 라인 검출 및 세부기능 구현	06/16(월) ~ 06/17(화)	 opencv-skeleton 좌표 검출  Doosan API 세부 함수 구현	한 붓 그리기 이미지 제작, 좌표 그리기 순서 지정
코드 통합 및 테스트	06/17(화) ~ 06/19(목)	 개별 기능 통합  통합 테스트	코드 통합 및 버그 수정
영상 및 PPT제작	06/19(목) ~ 06/20(금)	 시연 영상 촬영 및 제작  PPT제작 및 발표 준비	
총 개발기간	06/09(월) ~ 06/20(금) (총 2주)		

04-1 프로젝트 배경, 기획의도

- 빈번한 수정과 반복작업이 많은 디자인 직무(캐릭터, 타투 도안 제작 등)에서 인간의 육체적 노동을 로봇이 대체하여 피로도를 줄이고 제작자의 순수 창작 집중도 증가
- 일반적인 프린터기와 달리 3D(컵, 신체부위 등) 물체들위에도 그림 프린팅 가능하며 수정 완료 후 출력과정을 거쳐 확인하지 않고 수정한 부분을 실시간으로 확인이 가능
- 로봇 팔 기술의 발전으로 3D펜 사용시 3D object나 캐릭터 디자인이 가능



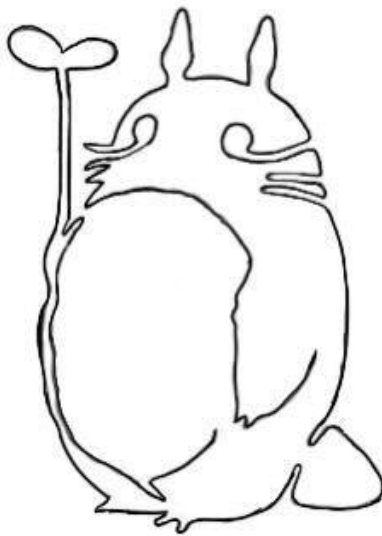
04-2

[환경] - input_data(토토로 이미지) 분할

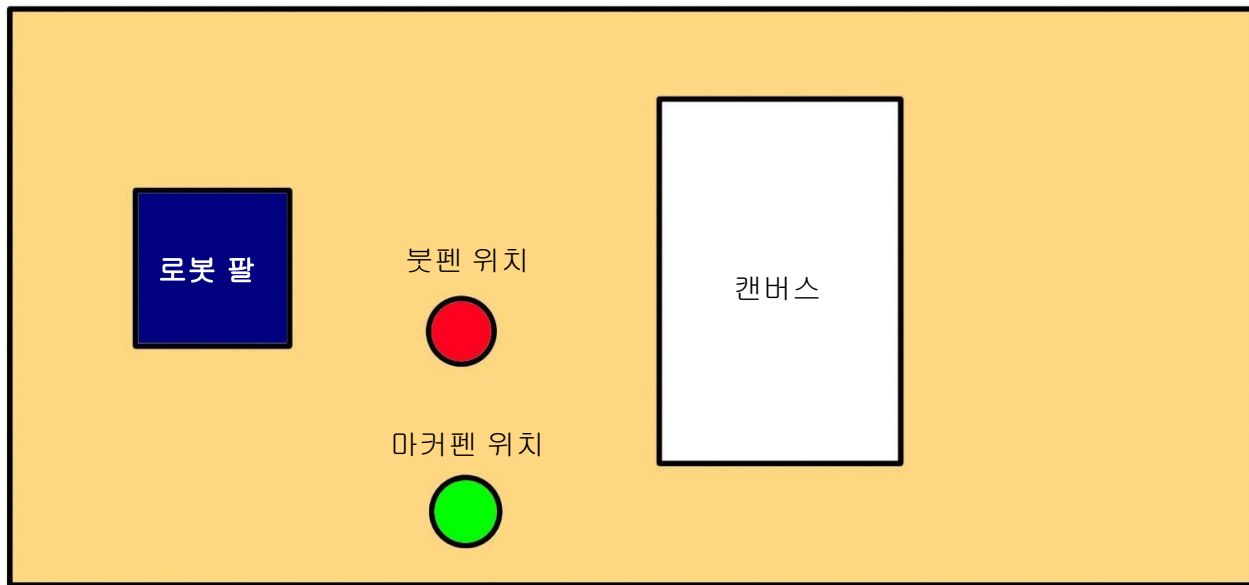
- ▶ 그래픽 디자인 소프트웨어 활용 이미지 분할 처리



토토로 image 분할

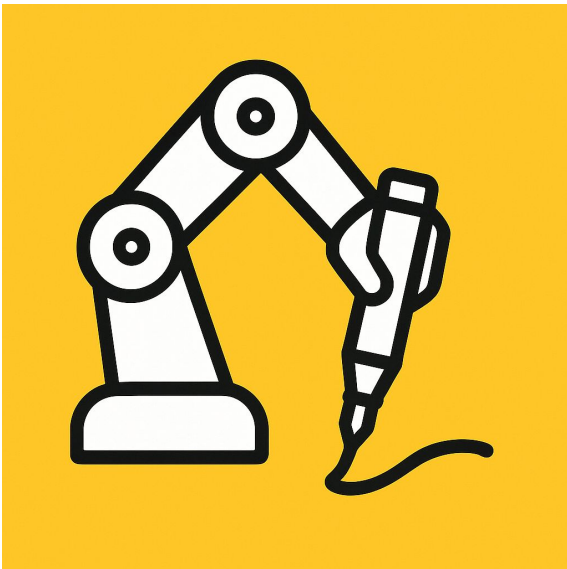


04-2 [환경] - Task 환경 구성



04-3 [시나리오] - 마커펜 잡고 이동

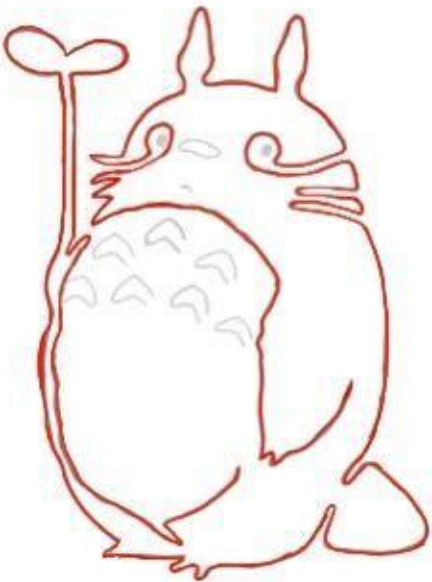
▶ 마커펜 **grip** -> 캔버스 위로 이동



- **move**로 로봇 팔이 마커펜 위치로 이동
- **pick and place** 방식으로 마커펜 **grip**
- **move**로 캔버스 위로 이동

04-3 [시나리오] - 토도로 그리기

- ▶ 전체적인 윤곽선 그리기 (한붓 그리기 방식)



- **OpenCV**를 활용 **point**를 순서대로 추출
- **point**를 **80**개씩 **list**에 저장
- **movesx**방식으로 경로 이동

04-3 [시나리오] - 토도로 그리기

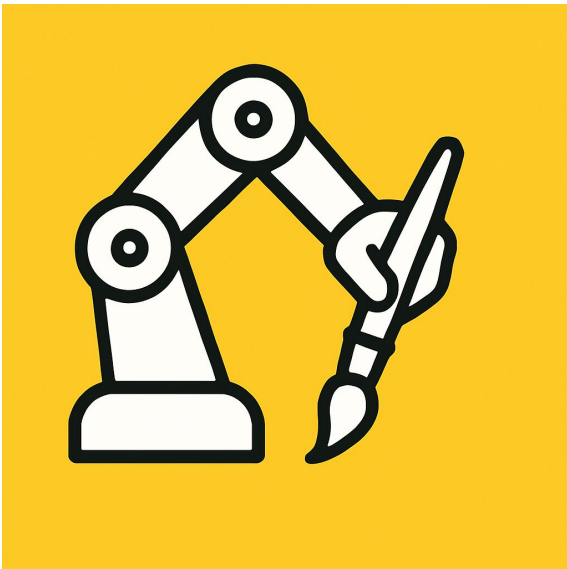
▶ Detail(코, 입, 배 무늬) 그리기



- **OpenCV**를 활용 **point**를 순서대로 추출
- **point**를 **n**개씩 **list**에 저장 (총 9개)
- **moveI**로 **z**축 방향(위)로 이동
- **movesx**방식으로 경로 이동
- 총 9번 반복

04-3 [시나리오] - 붓펜 잡고 이동

▶ 넋지 -> 붓펜 **grip** -> 캔버스로 이동



- 넋지(z축 방향 힘) - 트리거 신호
- **move1**로 이동하여 마커펜을 원위치
- **move1**로 붓펜 위치로 이동
- **pick and place**방식으로 붓펜 **grip**
- **move1**로 캔버스 위로 이동

04-3 [Nudge]

K-Digital Training

▶ Nudge 코드 리뷰

```
def wait_for_nudge_then_draw(threshold=20.0):
    print("Z축 방향으로 최소 20.0N 이상 누르면 다음 동작을 수행합니다.")
    try:
        while rclpy.ok():
            while not check_force_condition(axis=DR_AXIS_Z, min=threshold, ref=DR_TOOL):
                print("Nudge 감지됨! 후속 동작 수행 중")
                brush_pen()
                draw_eyes()
                # draw_nose()
                break
            time.sleep(0.1)

        print("경로 실행 중...")
        time.sleep(3.0)
        print("경로 실행 완료")
    except Exception as e:
        print(f"예외 발생: {e}")
```

threshold = z축 힘의 최소 임계값(20N)

check_force_condition는 ref_DR_TOOL에서 제공하는 힘 데이터를 읽어 (AXIS_Z) 방향의 값을 min_threshold와 비교

Nudge 감지 시 다음 시퀀스 실행

04-3 [시나리오] - 토도로 그리기

▶ Detail(눈) 그리기



- **move1**로 -z축 방향으로 이동
- 힘 제어 **on**(tool좌표계 기준 -z축zx 방향 **8N**)
- 3초 유지 후 힘제어 **off**
- **move1**로 +z축 방향으로 이동
- 왼쪽눈, 오른쪽눈 총**2**번 반복

04-4 [opencv] - 좌표추출

▶ 이미지 전처리 (image pre-processing) & 스켈레톤 중심선 추출

```
# 1. 이미지 불러오기 및 리사이즈 (210 x 297)
img_path = "/home/jin/Downloads/IMG_1452.jpg"
img = cv2.imread(img_path)
# row = 2408 // 10
# column = 3508 // 10
# img = cv2.resize(img, (row, column)) # (width, height)

# 2. 흑백 변환 및 이진화
gray_img = rgb2gray(img)
thresh_val = threshold_otsu(gray_img)
binary_img = gray_img < thresh_val # 검정 선 추출을 위해 반전

# 3. 스켈레톤 중심선 추출
skeleton = skeletonize(binary_img)
```

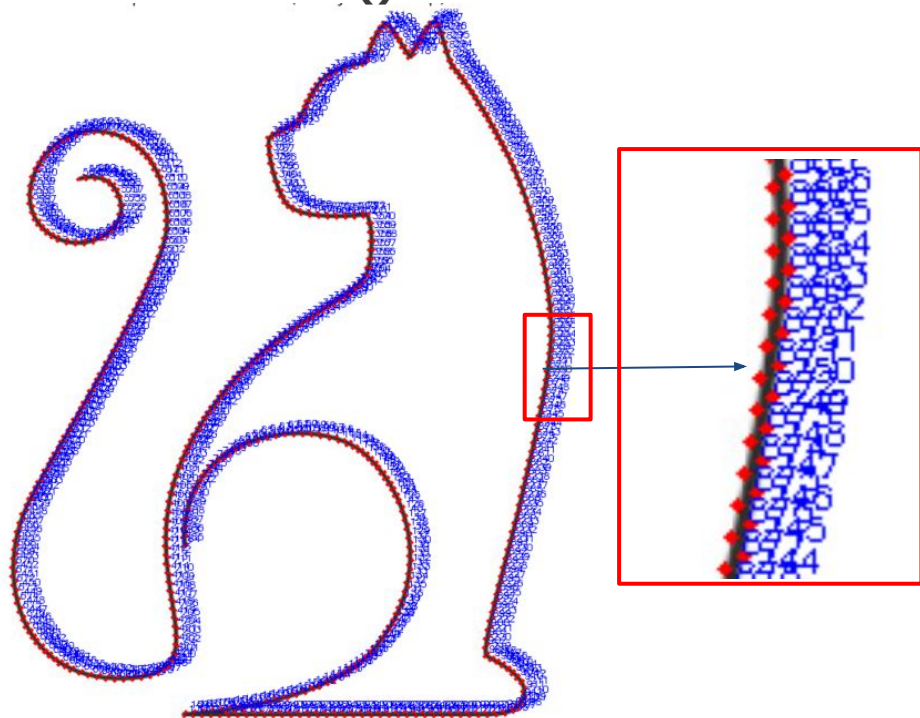
=> 이미지 불러오기

=> 이미지 흑백 변환 및 otsu method를 활용한 thresholding 및 이진화

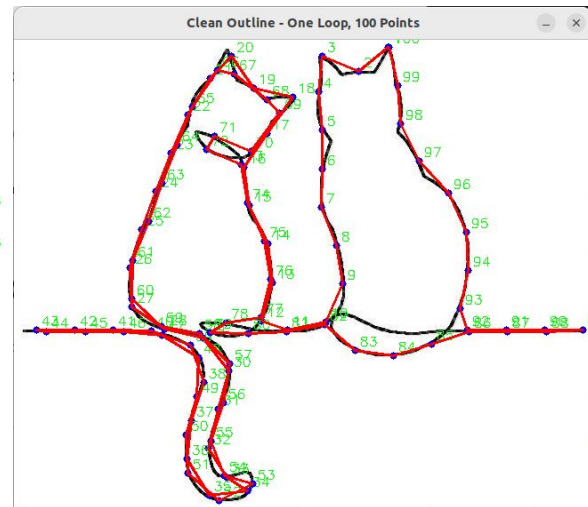
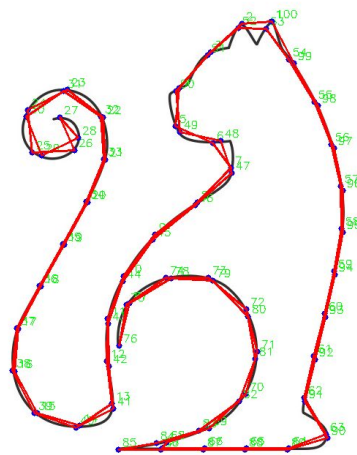
=> skimage.morphology.skeletonize -
skeletonize(binary_img)를 활용하여
중심선 추출 - 1 픽셀 두께 선의 경로 추적
회전함

04-4 [opencv] - 좌표추출

▶ skeletonize() 사용이유 및 시행착오



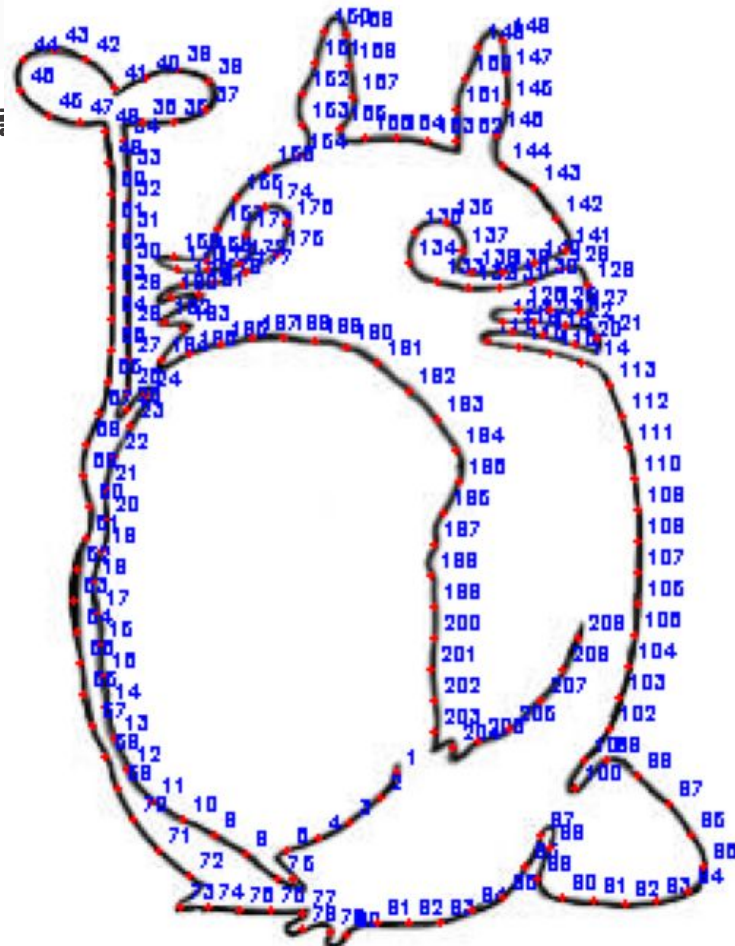
Hough Transform



openCV's contours() fn

04-4 [opencv] - 좌표추출

- ▶ **skeletonize** 이후 **bfs**를 활용한 좌표 추출



04-4 [opencv] - 좌표추출

▶ 한붓 그리기 시작점 & 시작점 찾기 함수

```
# 4. 끝점 찾기 함수
def find_endpoints(skel_img):
    endpoints = []
    for y in range(1, skel_img.shape[0] - 1):
        for x in range(1, skel_img.shape[1] - 1):
            if skel_img[y, x] == 1:
                neighbors = np.sum(skel_img[y-1:y+2, x-1:x+2]) - 1
                if neighbors == 1:
                    endpoints.append((y, x))
    return endpoints
```

=> 시작점 찾기 함수

스켈레톤화된 이미지를 스캔하여
스켈레톤의 일부이며 (i.e. value = 1),
단 한점의 스켈레톤 점에 둘러싸인 픽셀
추출

장점: BFS / DFS 같은 경로추적
알고리즘의

시작점으로 매우 유용함

추후에 BFS 을 이용하여 추출된
점들을

순차적으로 이어줌으로써 한붓그리기
형태의 외곽선을 그릴 수 있음

04-4 [opencv] - 좌표추출

▶ BFS 기반 경로 추적 및 좌표 추출

5. BFS 기반 경로 추적

```
def fast_trace_path(skel_img, start):
    visited = np.zeros_like(skel_img, dtype=bool)
    path = []
    q = deque()
    q.append(start)
    visited[start] = True
    while q:
        y, x = q.popleft()
        path.append((y, x))
        for dy in [-1, 0, 1]:
            for dx in [-1, 0, 1]:
                if dy == 0 and dx == 0:
                    continue
                ny, nx = y + dy, x + dx
                if (0 <= ny < skel_img.shape[0]) and (0 <= nx < skel_img.shape[1]):
                    if skel_img[ny, nx] == 1 and not visited[ny, nx]:
                        visited[ny, nx] = True
                        q.append((ny, nx))
    return path
```

=> BFS 기반 좌표추출

이진화된 2D 스켈레톤 이미지에
너비 우선 경로 탐색 알고리즘을 적용하여
찾아진 시작점 에서부터 특정 간격으로
떨어진

모든 점 픽셀들의 좌표를 추적하고 기록함

전체 경로를 구성하는 (y,x) 좌표들의
리스트를

출력하며 이 점들을 이음으로써
원래의 이미지의 외곽선을 면밀히 재구성

04-4 [opencv] - 좌표추출

▶ BFS 기반 경로 추적 및 좌표 추출

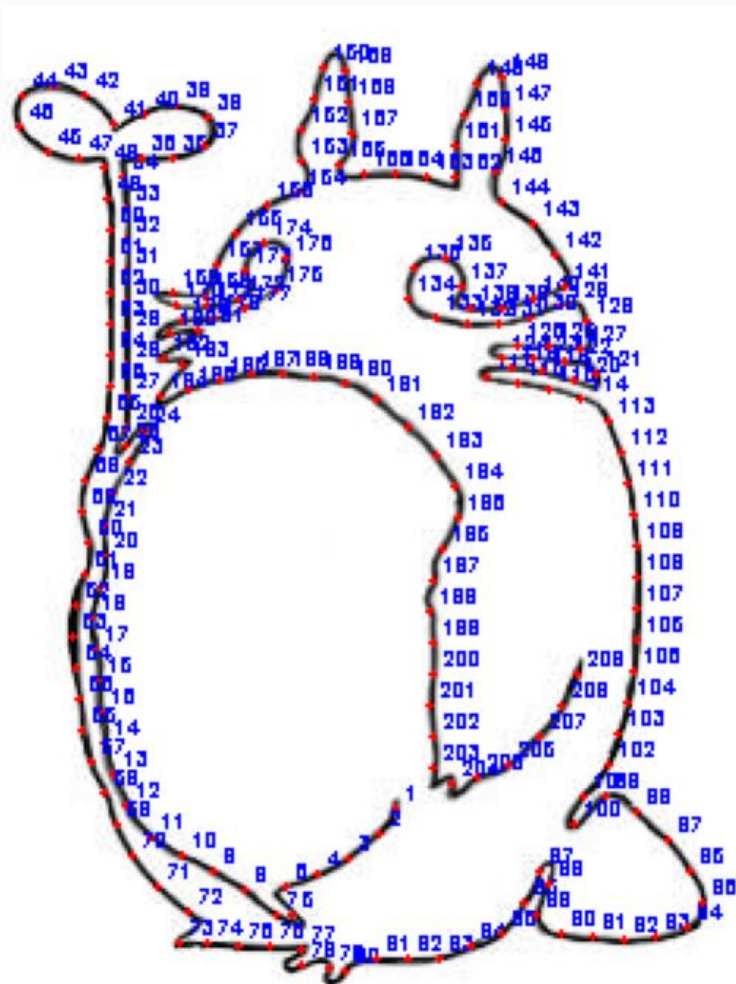
```
rokey@rokey-550XBE-350XBE:~/ros2_ws/src/openCV$ python3 contours2.py
```

```
↑ Sampled Coordinates (100 Points):
```

```
1: (420, 8)
2: (387, 35)
3: (346, 18)
4: (342, 58)
5: (346, 101)
6: (346, 145)
7: (345, 188)
8: (362, 231)
9: (369, 274)
10: (350, 317)
11: (307, 326)
12: (278, 316)
```

```
....
```

```
90: (596, 326)
91: (553, 326)
92: (509, 326)
93: (501, 302)
94: (510, 259)
95: (508, 216)
96: (488, 172)
97: (455, 136)
98: (434, 94)
99: (431, 51)
100: (421, 8)
```



04-4

K-Digital Training

[opencv] - 좌표추출

▶ 로봇 좌표로 변환

```
# 6. 시작점 설정 및 경로 추출
endpoints = find_endpoints(skeleton)
start_point = (100, 107) # 수동 설정 가능: (y, x)
skeleton_path = fast_trace_path(skeleton, start_point)

# 7. 시각화 (10개마다 점찍기)
img_vis = img.copy()
for idx, (y, x) in enumerate(skeleton_path[::3]):
    cv2.circle(img_vis, (x, y), 2, (0, 0, 255), -1)
    cv2.putText(img_vis, str(idx + 1), (x + 3, y - 3),
                cv2.FONT_HERSHEY_SIMPLEX, 0.15, (255, 0, 0), 1)

# plt.figure(figsize=(8, 11))
# plt.imshow(cv2.cvtColor(img_vis, cv2.COLOR_BGR2RGB))
# plt.title("Skeleton Path (Resized to 210x297)")
# plt.axis("off")
# plt.show()

# 8. 로봇 좌표로 변환
nose_draw = [(x, y) for y, x in skeleton_path]
print(len(nose_draw)//5)
return nose_draw
```

=> 이전에 정의된 함수들을 활용하여
시작점 설정 및 경로 추출

=> 점 시각화

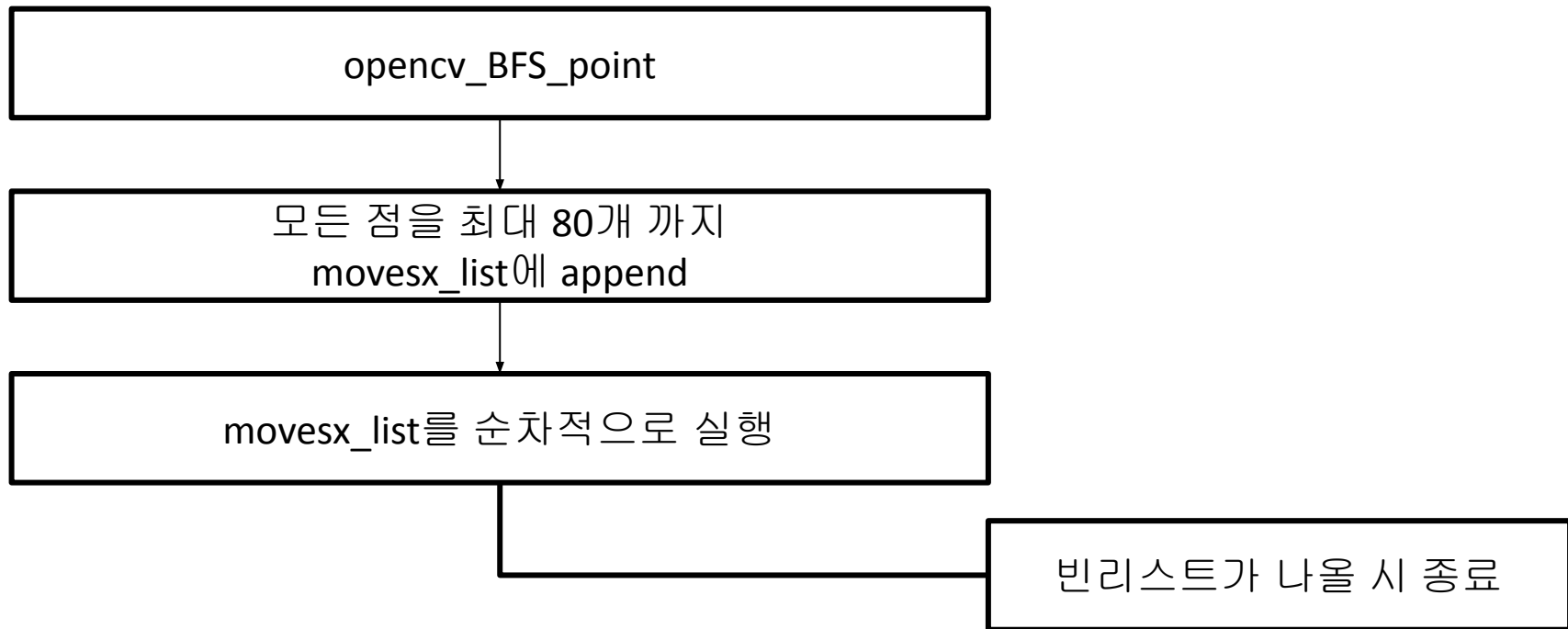
=> 로봇 좌표로 변환하여
선 그리기 함수들로 전달

04-4

K-Digital Training

[그리기] - 인식된 점들을 하나의 경로로 만들기

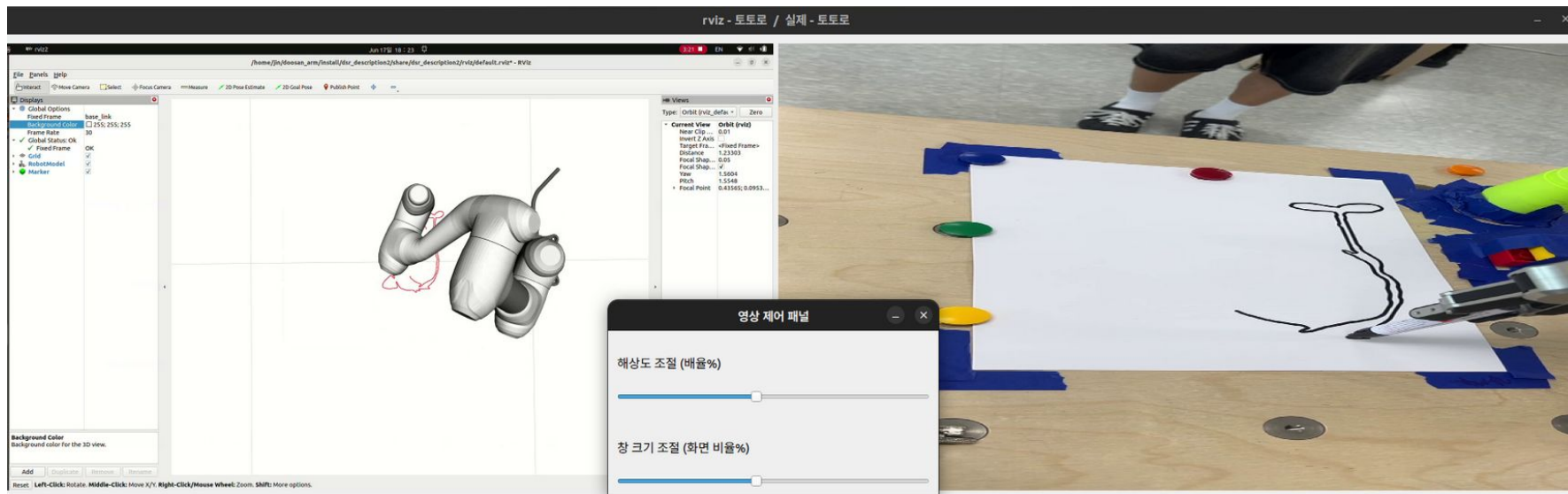
- ▶ opencv로 나온 path를 movesx()로 실행



04-5 GUI

K-Digital Training

▶ gui 제작

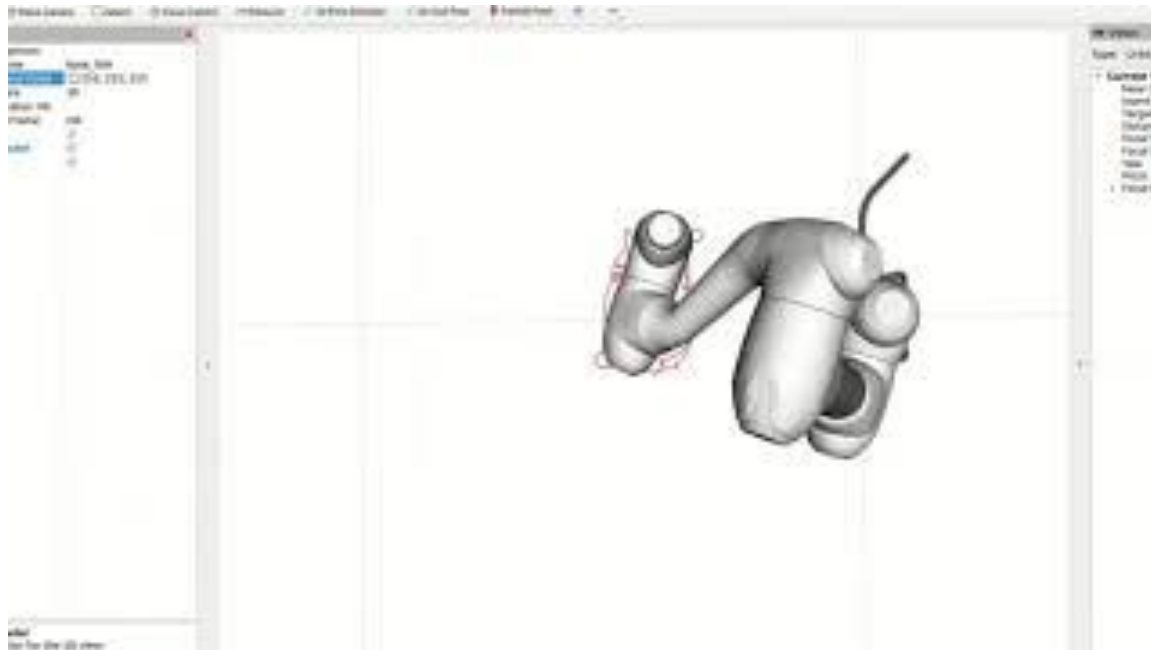


=> 비교적 간단한 gui 제작

04-5 GUI

▶ marker를 활용한 가상 시뮬레이션

10배속

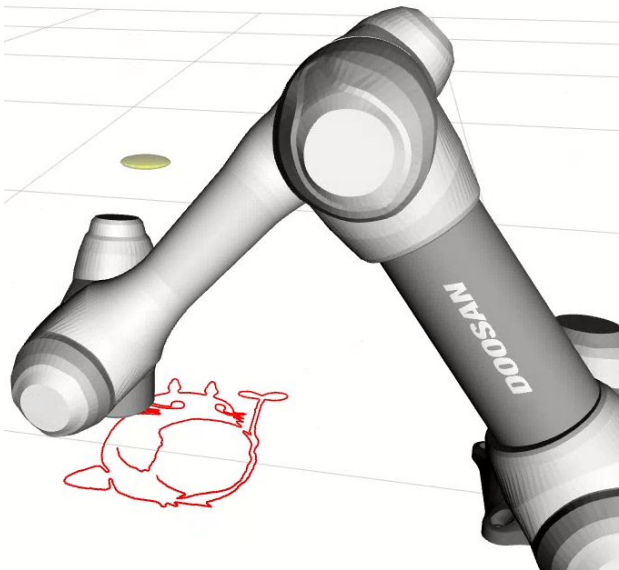


=> 마커를 통해 가상 그림 제작

=> 시뮬레이션 환경에서 검증 후 실제 로봇 작동

04-5 K-Digital Training GUI

▶ marker를 활용한 가상 시뮬레이션

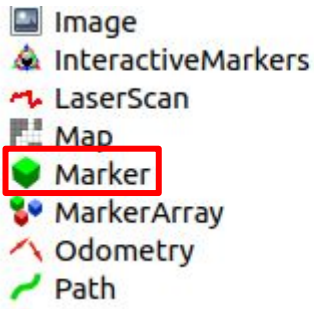


```
from visualization_msgs.msg import Marker
```

=> 노드에서 마커 정보를 퍼블리셔

```
marker_pub = node.create_publisher  
(Marker, "/skeleton_path_marker", 10)
```

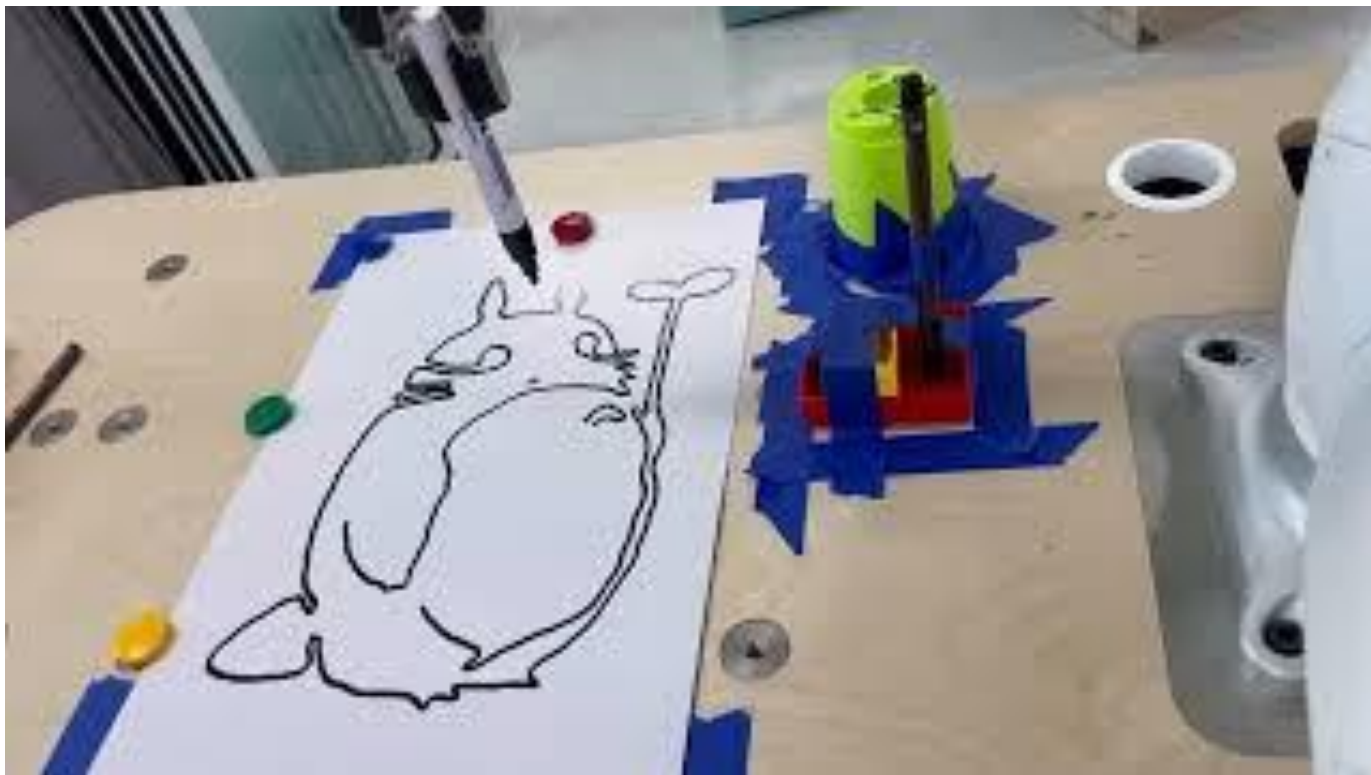
=> 현재 x,y,z좌표를 가져와 rviz2 상에서 정보를 제공받음



=> 0.1초마다 로봇 tcp의 좌표를 퍼블리셔, rviz2에서 시각화

04-5 real_robot_draw totoro

▶ 동영상



프로젝트 결과물에 대한 완성도 평가(10점 8점)

spiral, periodic을 사용하고싶었지만 API 버그
이슈로 인해 구현하지 못했다.

또한 movesx를 통해 힘 제어를 추가해 펜의
높이를 조정하고싶었지만 힘 제어를 잘 사용하지
못 해 구현에 실패했다.

◇ 하지만 이 외에 opencv 좌표 검출, 검출된 좌표
순서 정해서 line그리기, nudge 함수 제작으로
다음 동작 진행하기, 힘 제어 단독 사용으로 눈
그리기, 그림을 레이어드하여 배의 무늬 그리기
등 다양한 기능들을 사용하려고 했고 학습을 통해
성공해냈음에 의의를 둔다.

추후 개선점이나 보완할 점

- 1.제공된 Doosan API함수들을 충분히 숙지하고
시작하지 않아 오류가 많이 발생했다. (spiral,
기준좌표계 변환 함수 등)
- 2.캐릭터에 세세한 무늬나 선이 추가되면 좌표
검출이 힘들어 원하는 그림이 한 번에 나오지
않았다.
- 3.move_periodic함수를 사용하여 주기와
진폭으로 -sin함수와 cos함수의 그래프를
활용하여 구현하려 하였으나 기능적으로
불가능하다고 판단이 되어 사용에 실패했다.
4. 단일노드만 사용하여 로봇을 작동시켰는데 더
많은 노드를 사용해서 고도화 하지 못 한 점이
아쉽다.

느낀 점 / 경험한 성과

turtlebot 로봇 모델을 진행하다가 처음으로 6축 로봇팔 모델을 사용했다. Manipulator와 비슷할거라는 예상과는 다르게 Dart Platform이라는 로봇 동작 구현 도구가 있어 1차로 Task Build를 하고 추후에 코드 작성시에 TB를 기준으로 조금 더 큰 틀, 함수 단위로 코드를 작성할 수 있었다. 여태까지는 세부 단위부터 코드를 작성했는데, 새로운 시각을 제공받고 더 성장했다.

하드웨어의 연결성이 비교적 튼튼하지 않고 무선으로 data를 주고 받아 여러가지 오류가 발생하는 turtlebot을 활용한 프로젝트와 달리 상용화된 협동로봇을 사용하니 하드웨어적 오류로 인해 낭비되는 시간이 없어 효율적으로 프로젝트 진행이 되어 좋았다.

한붓그리기 (eulerian path) 를 활용해 중복을 최소화하고 자연스러운 선 연결을 구현하면서, 경로 계획이 단순한 문제 같지만 실제 결과물의 품질에 큰 영향을 미친다는 점을 깨달았습니다. 또한 단순한 기술 구현을 넘어서 예술적인 표현(그림 그리기)을 실험하면서, 공학과 창의성의 융합이 매우 흥미롭고 보람있다는 느낌을 받을 수 있었습니다

이미지를 여러가지 방식으로 처리해보면서 opencv에 대한 이해도를 높일 수 있었습니다. 또한 추출된 점들을 어떻게 부드럽게 연결할 수 있을지에 대한 고민을 하면서, path planning에 대한 부분을 공부할 수 있어 좋았습니다.

잘 한 부분 / 아쉬운 부분

- API 기능들을 최대한 여러가지 사용하려고 했고, 결국 구현에 실패한 기능들이 있긴 하지만 많이 학습하며 프로젝트를 진행하려는 취지에 걸맞게 실행했다고 생각한다.
- 선을 구성하는 좌표들을 추출하는 과정에서 openCV의 hough transform 이나 contours 등을 사용하며 시행착오를 겪었으나 결국 skeletonize 함수와 bfs를 활용하여 순차적인 좌표들을 완벽하게 추출하여 그림을 movesx함수로 그리기 용이하게 한 작업을 잘했다고 생각한다.
- 소프트웨어적 한계로 직면한 문제점을 하드웨어적으로 해결한 점이 잘한점이라고 생각한다.
- 단순히 점들을 찍어서 그림을 그리는게 아니라 opencv로 좌표를 추출해서 path를 만들어서 효율적인 프로젝트를 수행할 수 있었다.