

[두산로보틱스] 지능형 로봇틱스 엔지니어

## 협동로봇을 활용한 주유 시스템 자동화

협동2 - AI(Computer Vision) 기반 협동 로봇 작업 어시스턴트 구현 프로젝트

TEAM E-1조 아사히(Asahi)

[팀원] 신동현, 하종우

[팀장] 김서희

[멘토] 권영선 강사님

# 목 차



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

# 01

K-Digital Training

## 프로젝트 개요

1

### 프로젝트 주제 및 선정 배경, 기획의도

**주제:** 주유소 주유 시스템  
자동화

**특화 포인트:** Doosan Api  
사용, Yolo와 STT keyword  
mapping, 예외상황 발생 처리

**차별성:** 사람이 직접 내려서  
주유 해야되는 기존 무인  
주유소와 차별화 되는 완전  
자동화

2

### 프로젝트 내용

**컨셉:** 알고리즘 과 로봇Arm을  
통해 운전자가 차에서 내리지  
않고, 주유 및 결제 과정을  
완전 자동화 해주는 주유소  
시스템

**훈련 연관성:** ROS2와 Yolo,  
그리고 STT를 활용하여  
작성한 python code로  
협동로봇(m0609)작동

3

### 활용 장비 및 재료

**하드웨어:** Doosan m0609,  
주유소 모형

**소프트웨어:** Ubuntu 22.04,  
ROS2 Humble, Python,  
Doosan API, YOLO, RViz2,  
GPT 4-o LangChain, STT, LLM

4

### 프로젝트 구조

**프로젝트 목적:** 기존 주유소의  
불편함과 안전 문제를  
해결하고 사용자가 차에서  
내리지 않고도 음성  
명령만으로 주유부터 결제까지  
가능한 완전 자동화/비접촉  
주유 시스템 개발

**진행 일정:** 2025년 06월 23일 ~  
2025년 07월 4일

5

### 활용방안 및 기대 효과

**기대효과:** 로봇 팔 기술의  
발전을 통한 주유소 전면  
자동화로 안전하고 편리한  
주유소 경험 가능

**실무 활용성:** 무인주유소 및  
스마트 주유소 현장 적용 가능.  
차량 서비스 산업과 연계 가능.  
안전 강화 및 고객 만족도 향상.  
차세대 스마트 주유소 레퍼런스

## 02

K-Digital Training








## 프로젝트 팀(E-1) 구성 및 역할

훈련생	역할	담당 업무
김서희	팀장	 STT,LLM  Node 통신 구현  STT-YOLO keyword mapping
신동현	팀원	 협동 로봇 동작 제어 코드 구현  테스트 물리 환경 구성
하종우	팀원	 Yolo 객체탐지  GUI 제작

## 03

K-Digital Training

## 프로젝트 수행 절차 및 방법

구분	기간	활동	비고
프로젝트 이론 학습	06/23(월) ~ 06/25(수)	 Camera Calibration  STT, LLM 학습	
프로젝트 아이디어 회의 및 계획 수립	06/25(수) ~ 06/26(목)	 프로젝트 주제 선정  system design	시나리오 작성
Yolo, STT 및 세부기능과 동작구현	06/27(금) ~ 07/2(수)	 좌표 추출, 시나리오 알고리즘 확인  Python code 작성	중간발표, Yolo, STT
코드 통합 및 테스트	07/2(수) ~ 07/3(목)	 코드 구현 기능 test  개별 기능 통합	코드 통합 및 버그 수정
영상 및 PPT제작	07/3(목) ~ 07/4(금)	 코드 최종본 추출  PPT제작 및 발표 준비	
총 개발기간	06/23(월) ~ 07/4(금) (총 2주)		

# 04-1 K-Digital Training 프로젝트 배경, 기획의도

## 배경

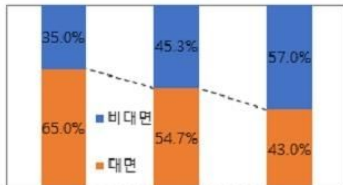
- 비대면/비접촉 서비스 수요 급증
- 무인 주유의 한계
- 주유 안전성 문제
- 스마트 주유소/스마트 시티 트렌드

## 기획의도

- 차량 내에서 편리한 주유 경험 제공
- 로봇 기술과 AI융합을 통한 주유 자동화
- 음성 인식 기반 서비스
- 비접촉/비대면 편의성 강화
- 주유소 운영 효율성 향상

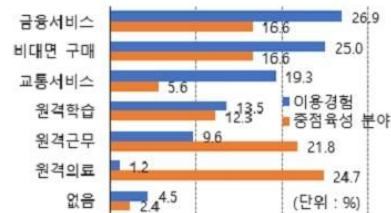
### ◎ 언택트 서비스 분야별 소비자 인식조사 결과

<언택트 소비 비중 변화(금액기준)>



출처: <https://www.yna.co.kr/view/AKR20200521127300061>

<언택트 서비스 이용경험 vs. 육성분야>



출처: <https://www.hankyung.com/article/2018111873671>

# 04-1 프로젝트 고도화, 사업 모델링 확장, 시행착오

- 수익창출, 고도화 가능성
  - 차량 모델마다 경유차/휘발유차/전기차 인지 객체인식하여 음성명령 없이도 주유 자동화
  - 무인 주유소 / 셀프 정비 / 세차 스테이션 등에서의 자동화 키트 **B2B**판매
- 다른 분야로 사업 모델링이 확장 가능한지
  - 쓰레기장
  - 약국
  - 전기차 충전소
  - 폭발성/독성 화학물질 취급
- 시행착오
  - **gas station** 고정이 안되는 문제->고정판
  - **move1**로 최단경로 이동시 테스크 환경과 충돌하여 **movej**로 경유점을 설정해 해결
  - 일부 **pc**에서 **STT-LangChain** 버전 호환 문제
  - **nfc**를 인식해서 **card\_moving**을 하려 했으나 재질 특성상 빛반사가 심해 인식률이 낮아 구현 실패

# 04-2 [환경] - Task 환경 구성





# 04-3 [시나리오]

K-Digital Training

## ▶ 전체 시나리오



1. 차량 상단의 버튼을 눌러 주유구 열기
2. **Wakeup - STT - LLM - Yolo** 검출로 주유기 검출  
이후 해당위치로 이동하여 주유기 집기
3. 차량에 주유후 **nudge**시 주유기 원위치
4. **Yolo**로 카드 검출후 결제진행후 카드 원위치
5. 차단기 올린후 차량 출발

# 04-3 [system design]

## ▶ 시스템 아키텍처 다이어그램

**get\_keyword.py**  
**keyword\_extraction.py**

- ① STT
- ② LLM 추론
- ③ 도구 토픽 퍼블리시
  - /oil
  - /target/CreditCard
  - /detected\_tool

**gas\_station\_yolo\_stt\_mapping.py**

- YOLO 인식
  - /detected\_tool 기반 필터링
  - 감지 결과 → /yolo\_result

**onrobot\_robot\_control.py**

- ① /oil → 연료 종류
- ② /target/CreditCard → 카드  
삽입
- ③ /yolo\_result → 좌표 인식
- ④ Doosan M0609 제어 및  
그리퍼 동작

# 04-3

K-Digital Training

## [STT]-get\_keyword.py

```
class AutoFuelNode(Node):
    def __init__(self):
        super().__init__("auto_fuel_node")

        dotenv_path = Path(__file__).parent.parent / "resource" / ".env"
        load_dotenv(dotenv_path=dotenv_path)
        api_key = os.getenv("OPENAI_API_KEY")

        self.stt = STT(openai_api_key=api_key)
        self.extractor = ExtractFuelInfo(api_key)

        mic_config = MicConfig(
            chunk=12000, rate=48000, channels=1,
            record_seconds=5, fmt=pyaudio.paInt16,
            device_index=10, buffer_size=24000,
        )
        self.mic_controller = MicController(mic_config)
        self.wakeup_word = WakeupWord(mic_config.buffer_size)

        self.fuel_pub = self.create_publisher(Int32, "/oil", 10)
        self.creditcard_pub = self.create_publisher(Bool, "/target/CreditCard", 10)
        self.detected_tool_pub = self.create_publisher(String, "/detected_tool", 10)
        # Add amount publisher for gui
        self.amount_pub = self.create_publisher(Int32, "/amount", 10)
        # Add finish

        self.get_logger().info("자동 주유 음성인식 노드 시작됨.")
        self.get_keyword_srv = self.create_service(
            Trigger, "get_keyword", self.get_keyword_callback
        )
    ]
```

- STT publish

- GPT-4o로 [연료/결제/금액] 추출

- 각 결과에 따라 다음을 publish함

- /oil: Int32 (0:경유, 1:휘발유)

- /target/CreditCard: Bool (결제수단)

- /detected\_tool: String (YOLO필터링)

# 04-3 K-Digital Training [STT]-get\_keyword.py

```
def get_keyword_callback(self, request, response):
    try:
        self.mic_controller.open_stream()
        self.wakeup_word.set_stream(self.mic_controller.stream)
    except OSError:
        self.get_logger().error("오디오 장치 오류")
        response.success = False
        response.message = "오디오 오류"
        return response

    self.get_logger().info("웨이크업 단어 대기 중...")
    while not self.wakeup_word.is_wakeup():
        pass

    self.get_logger().info("음성 수신 중...")
    spoken_text = self.stt.speech2text()

    fuel, payment, amount = self.extractor.extract(spoken_text)
    self.get_logger().info(f"[LLM 추출 결과] 연료: {fuel}, 결제: {payment}, 금액: {amount}")

    # 연료 퍼블리시
    if fuel == "경유":
        self.fuel_pub.publish(Int32(data=0))
    elif fuel == "휘발유":
        self.fuel_pub.publish(Int32(data=1))

    # 결제 수단 퍼블리시
    if payment.lower() in ["카드", "creditcard", "card"]:
        self.creditcard_pub.publish(Bool(data=True))

    # Add amount publish for gui
    if amount.isdigit():
        self.amount_pub.publish(Int32(data=int(amount)))
    # Add finish

    # YOLO 타겟 퍼블리시
    fuel_yolo = map_to_yolo_class(fuel)
    payment_yolo = map_to_yolo_class(payment)

    if fuel_yolo:
        self.detected_tool_pub.publish(String(data=fuel_yolo))
    if payment_yolo:
        self.detected_tool_pub.publish(String(data=payment_yolo))

    response.success = True
    response.message = f"[{fuel}] / [{payment}] / [{amount}]"
    return response
```

- 역할 : 음성 (STT) → LLM 분석 → 연료/결제/금액 추출 → ROS topic publish

- 핵심 흐름:

- Wakeup 단어 감지 후 speech2text() 호출
- GPT-4o로 [연료/결제/금액] 추출
- 각 결과에 따라 다음을 publish함

/oil: Int32 (0:경유, 1:휘발유)

/target/CreditCard: Bool (결제수단)

/detected\_tool: String (YOLO 필터링)

# 04-3 K-Digital Training [STT]-get\_keyword.py

다음 사용자 명령어를 분석해서 [연료 / 결제 / 금액] 형식으로만 결과를 출력하세요.

## <제한 사항>

- 출력은 반드시 [연료 / 결제 / 금액] 형식이어야 합니다.
- '[' 와 ']' 괄호와 '/' 구분자를 포함한 정확한 포맷을 지켜야 합니다.
- 연료: 휘발유 또는 경유
- 결제: 카드 또는 현금
- 금액: 숫자만 (예: "5만원" → 50000)
- 해당 값이 없으면 공백으로 두고 슬래시는 유지합니다.

## <예시>

입력: 경유로 5만원 카드 결제해줘 → [경유 / 카드 / 50000]

입력: 카드 결제할게 → [ / 카드 / ]

입력: 휘발유로 카드 결제 → [휘발유 / 카드 / ]

## <사용자 입력>

```
"{user_input}"
```

## <출력 형식>

```
"""
```

- 사용자 음성 명령->[연료 / 결제 / /금액] 형식으로 변환
- 누락된 항목은 비우되 / 구분자는 유지

```
def extract(self, text):
    response = self.lang_chain.invoke({"user_input": text})
    try:
        raw = response["text"].strip().strip("[ ]")
        parts = raw.split("/")
        if len(parts) != 3:
            raise ValueError("응답 형식 오류")
        fuel = parts[0].strip()
        payment = parts[1].strip()
        amount = parts[2].strip()
        return fuel, payment, amount
    except Exception as e:
        print(f"[ERROR] 응답 파싱 실패: {e}")
        return "", "", ""
```

- 인식한 **STT**가 3파트로 나뉘어지지 않으면 오류처리
- 인덱싱을 통해 **fuel, payment, amount** 분류

# 04-3 [STT]-get\_keyword.py

```
# === YOLO 클래스 이름과 사용자 표현 매핑 ===
YOLO_CLASS_MAPPING = {
    "creditcard": ["카드", "신용카드", "card"],
    "pump": ["펌프", "주유기", "fuelgun", "건", "주유기기"],
    "cash": ["현금", "돈"]
}
```

```
def map_to_yolo_class(keyword: str):
    keyword = keyword.strip().lower()
    for yolo_class, alias_list in YOLO_CLASS_MAPPING.items():
        if keyword in alias_list:
            return yolo_class
    return ""
```

- 중요 함수:

- `map_to_yolo_class()`

- 역할 : 사용자가 말한 키워드를 **YOLO classname**으로 매핑해 반환

ex) "카드" -> "creditcard"

사전에 정의된 **YOLO class**와 대응 표현들 순회하며 **alias**

목록에 포함되어있는지 확인

일치하는 **YOLO classname** 반환

일치하지 않으면 빈 문자열 반환

# 04-4 [LLM]-keyword\_extraction.py

K-Digital Training

```
def run_pipeline(self):
    self.mic_controller.open_stream()
    self.wakeup_word.set_stream(self.mic_controller.stream)
    self.get_logger().info("웨이크업 단어 대기 중...")

    while not self.wakeup_word.is_wakeup():
        pass

    self.get_logger().info("음성 입력 수신 중...")

    try:
        text = self.stt.speech2text()
        self.get_logger().info(f"[STT 결과] \"{text}\"")
    except Exception as e:
        self.get_logger().error(f"STT 실패: {e}")
        return

    fuel, payment, amount = self.extractor.extract(text)
    self.get_logger().info(f"[LLM 추출 결과] 연료: {fuel}, 결제: {payment}, 금액: {amount}")

    # 연료 퍼블리시
    if fuel in ["휘발유", "경유"]:
        self.pump_pub.publish(Bool(data=True))
        self.detected_tool_pub.publish(String(data="pump"))

    if fuel == "경유":
        self.fuel_pub.publish(Int32(data=0))
    elif fuel == "휘발유":
        self.fuel_pub.publish(Int32(data=1))

    # 결제 수단 퍼블리시
    if payment in ["카드", "creditcard", "card"]:
        self.card_pub.publish(Bool(data=True))
        self.detected_tool_pub.publish(String(data="creditcard"))

    # 콘솔 로그
    self.get_logger().info(f"주유 요청: {fuel}, 결제: {payment}, 금액: {amount}")
```

- 역할 : **get\_keyword.py**와 유사하지만 **Srv**가 아닌 실행형 **pipeline**
- 차이점:
  - 실행 즉시 음성 인식 및 **publish** 수행
  - **target/Pump, /target/CreditCard, /detected/tool, /oil**



## 04-4

K-Digital Training

## [YOLO-STT]-gas\_station\_yollo\_stt\_mapping.py

```

class YoloTargetFilter(Node):
    def __init__(self):
        super().__init__('yolo_target_filter')

        self.target_sub = self.create_subscription(String, '/detected_tool', self.target_callback, 10)
        self.target_keyword = None
        self.publisher_ = self.create_publisher(String, 'yolo_result', 10)

        # YOLO 모델 불러오기
        self.model = YOLO("/home/rokey/Downloads/gasStation_1.pt")
        self.names = self.model.names

        # 내부에서 ImgNode 사용
        self.img_node = ImgNode()

        self.get_logger().info("YOLO 추론 노드 시작 (RealSense 이미지 기반)")

    def _get_depth(self, x, y):
        frame = self._wait_for_valid_data(self.img_node.get_depth_frame, "depth frame")
        try:
            return float(frame[y, x])
        except IndexError:
            self.get_logger().warn(f"Coordinates ({x},{y}) out of range.")
            return None

    def _wait_for_valid_data(self, getter, description):
        data = getter()
        while data is None or (isinstance(data, np.ndarray) and not data.any()):
            rclpy.spin_once(self.img_node)
            self.get_logger().info(f"Retry getting {description}.")
            data = getter()
        return data

    def _pixel_to_camera_coords(self, x, y, z):
        fx = self.intrinsics['fx']
        fy = self.intrinsics['fy']
        ppx = self.intrinsics['ppx']
        ppy = self.intrinsics['ppy']
        return (
            (x - ppx) * z / fx,
            (y - ppy) * z / fy,
            z
        )

```

- 역할 : STT결과(/detected\_tool)를 기반으로 YOLO 결과 필터링해서 /yolo\_result publish
- 중요점 :
  - target\_callback(): target class 저장
  - publish\_detection(): 감지된 객체 정보 publish



# 04-4 [LLM]-keyword\_extraction.py

K-Digital Training

```
class ToolPublisherNode(Node):
    def __init__(self):
        super().__init__("tool_publisher_node")

        # env 불러오기
        dotenv_path = Path(__file__).parent.parent / "resource" / ".env"
        load_dotenv(dotenv_path=dotenv_path)
        api_key = os.getenv("OPENAI_API_KEY")

        # STT, LLM 초기화
        self.stt = STT(openai_api_key=api_key)
        self.extractor = ExtractKeyword(api_key)

        # 오디오 마이크 설정
        mic_config = MicConfig(
            chunk=12000, rate=48000, channels=1,
            record_seconds=5, fmt=pyaudio.paInt16,
            device_index=10, buffer_size=24000,
        )
        self.mic_controller = MicController(mic_config)
        self.wakeup_word = WakeupWord(mic_config.buffer_size)

        # 퍼블리셔
        self.fuel_pub = self.create_publisher(Int32, "/oil", 10)
        self.pump_pub = self.create_publisher(Bool, "/target/Pump", 10)
        self.card_pub = self.create_publisher(Bool, "/target/CreditCard", 10)
        self.detected_tool_pub = self.create_publisher(String, "/detected_tool", 10) # ← 추가됨

        # 시작 로그
        self.get_logger().info("도구 추출 및 퍼블리셔 노드 시작됨")

        # 바로 실행
        self.run_pipeline()
```

- 음성 명령으로부터 추출된 정보(연료, 결제, 대상 객체)를  
**ROS2 topic으로 publish**

# 04-4

K-Digital Training

## [YOLO-STT]-gas\_station\_yollo\_stt\_mapping.py

```
def run(self):
    while self.cap.isOpened() and rclpy.ok():
        ret, frame = self.cap.read()
        if not ret:
            self.get_logger().warn("카메라 프레임 읽기 실패")
            time.sleep(1)
            continue

        # YOLO 추론 수행
        results = self.model.predict(source=frame, conf=0.4, verbose=False)[0]

        for box in results.boxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            conf = float(box.conf[0])
            cls_id = int(box.cls[0])
            cls_name = self.names[cls_id].lower()

            # STT 대상 키워드가 있을 때만 필터링
            if self.target_keyword is None:
                continue

            if cls_name == self.target_keyword:
                self.publish_detection(cls_name, cls_id, x1, y1, x2, y2, conf)

                # 시각화
                label = f'{cls_name} ({cls_id}) {conf:.2f}'
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv2.putText(frame, label, (x1, y1 - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

        cv2.imshow("YOLO Detection", frame)
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break

    self.cap.release()
    cv2.destroyAllWindows()
```

- 역할 : STT결과(/detected\_tool)를 기반으로 YOLO 결과 필터링해서 /yolo\_result publish
- 중요점 :
  - run(): 실시간 프레임 추론 및 결과 필터링
  - YOLO감지 결과와STT 키워드가 일치하는 경우만 처리함
  - 감지된 결과를 시각화 함

# 04-4 [robot\_control] - onrobot\_robot\_control

- ▶ 협동로봇 move 관련 함수 정리

```
# 0. 주유구 열기 모션
def open_hole(self): ...

# 1. 주유소에서 주유기 뽑고 주유구 까지 가는 모션
def pump_moving(self): ...

# 2. 주유기 주유구에서 빼고 가져다 놓는 모션
# 넛지 함수
def wait_for_nudge_then_comeback(self, path_list=None, vel=100, acc=100, threshold=20.0): ...

# 3. 카드를 집고 계산까지 완료하는 모션
def card_moving(self): ...

# 4. 차단기를 올리는 모션
def move_stopsign(self): ...
```

# 04-4 K-Digital Training [robot\_control] - onrobot\_robot\_control

## ▶ 협동로봇 move 관련 함수 정리

함수명	주요 기능	사용 알고리즘 / 방식	비고
<code>open_hole()</code>	차량 주유구 개방 동작	movej 이동(경광등 누름)	
<code>pump_moving()</code>	주유기 픽업 및 주유구 삽입	YOLO bbox 중심 계산- 간단한 스케일 기반 좌표 변환 -> movej + movel 이동 조합	YOLO 객체탐지 실패시 예외처리 동작
<code>wait_for_nudge_then_comeback()</code>	주유기 원위치	z축 기반 nudge 감지- 경로 기반 복귀	넛지 트리거로 사용
<code>card_moving()</code>	카드 픽업 및 결제 -> 카드 원위치	YOLO bbox 중심 계산- 간단한 스케일 기반 좌표 변환 -> movej + movel 이동 조합	YOLO 객체탐지 실패시 예외처리 동작
<code>move_stopsign()</code>	차단기 픽업 및 상승	pcik and place → z축 상승	

# 04-4 K-Digital Training [robot\_control] - onrobot\_robot\_control

## ▶ pub/sub 정리

```
# RobotController Node
class AutoFuelControlNode(Node):
    def __init__(self):
        super().__init__("auto_fuel_control")
        self.get_logger().info("자동 주유 로봇 제어 노드 시작됨.")
        self.init_robot()
        self.get_logger().info("로봇 위치 초기화 완료")
        # --- 상태 변수 ---
        self.fuel_type = None
        self.detected_object = None
        self.stt_success = False # STT 성공 여부 플래그 추가
        self.detected_bbox = None

        # --- Subscribers ---
        self.yolo_sub = self.create_subscription(String, 'yolo_result', self.yolo_callback, 10)
        self.fuel_sub = self.create_subscription(Int32, '/oil', self.fuel_callback, 10)
        self.card_sub = self.create_subscription(Bool, '/target/CreditCard', self.card_callback, 10)
        # --- Publisher ---
        self.status_pub = self.create_publisher(String, '/robot/status', 10)

    # Publish용 함수
    def publish_status(self, message):
        msg = String()
        msg.data = message
        self.status_pub.publish(msg)
        self.get_logger().info(f"[STATUS] {message}")
```

- 구독 토픽

-/oil -> fuel\_callback() : 연료 종류 구분

-/target/CreditCard -> card\_callback()

-/yolo\_result -> yolo\_callback()

- 발행 토픽

-/robot/status -> 각 동작 함수 완료시 상태msg

발행

# 04-4 K-Digital Training [robot\_control] - onrobot\_robot\_control

## ▶ callback 함수 정리

```
# ===== 연료 종류 공백 =====  
def fuel_callback(self, msg):  
    self.fuel_type = msg.data  
    self.get_logger().info(f"[음성] 연료 인식됨: {'경유' if msg.data == 0 else '휘발유'}")  
    self.execute_fuel_task()  
  
def execute_fuel_task(self):  
    if self.fuel_type == 0:  
        self.get_logger().info("[TASK] 경유 주유 동작 실행")  
    elif self.fuel_type == 1:  
        self.get_logger().info("[TASK] 휘발유 주유 동작 실행")  
    self.open_hole()  
    mwait()  
    self.pump_moving()  
    mwait()  
    self.wait_for_nudge_then_comeback()  
    mwait()  
  
def card_callback(self, msg):  
    if msg.data:  
        self.get_logger().info("[음성] 카드 결제 감지됨 → 카드 삽입 동작 실행")  
        self.card_moving()  
        mwait()  
        self.move_stopsign()
```

- /oil 토픽 -> fuel\_callback()

조건: 연료 종류(경유(0), 휘발유(1))인식

동작: 주유구 열기 -> 주유기 픽업 및 주유구 삽입

-> 넋지 트리거로 주유기 원위치

- /target/CreditCard 토픽 -> card\_callback()

동작: 카드 픽업 및 결제 -> 차단기 픽업 및 상승

# 04-4 K-Digital Training [robot\_control] - onrobot\_robot\_control

## ▶ callback 함수 정리

```
def yolo_callback(self, msg):
    self.get_logger().info("[YOL0] STT 명령이 success.")
    try:
        parts = msg.data.split()
        if len(parts) != 5:
            self.get_logger().warn(f"YOL0 메시지 형식 오류: {msg.data}")
            return

        class_name, conf = parts[0].lower(), float(parts[1])
        cx, cy, cz = map(int, parts[2:])

        # if class_name == "pump": class_name = "creditcard"
        self.get_logger().info(f"[YOL0] {class_name} 감지됨 (conf={conf:.2f})")

        if conf < 0.5:
            return
        self.detected_bbox = (cx, cy, cz)

        if class_name == "pump":
            self.pump_moving()
            mwait()
            self.wait_for_nudge_then_comeback()

        elif class_name == "creditcard":
            self.card_moving()
            mwait()
            self.move_stopsign()

    except Exception as e:
        self.get_logger().error(f"YOL0 콜백 예외: {e}")
```

- 핵심 흐름 : **yolo**에서 감지된 객체별로 행동 함수 분기

-YOLO에서 **pump**를 인지한 경우

동작: 주유기 픽업 및 주유구 삽입

-> 낮지 트리거로 주유기 원위치

-YOLO에서 **creditcard**를 인지한 경우

동작: 카드 픽업 및 결제 -> 차단기 픽업 및 상승

# 04-4 K-Digital Training [robot\_control] - onrobot\_robot\_control

## ▶ pump\_moving

```
# 1. 주유소에서 주유기 뽑고 주유구 까지 가는 모션
def pump_moving(self):
    # 사용자 정의(주유기가 보이는 위치)로 이동
    JOne = [24.07, 40.60, 28.48, -1.66, 111.86, -161.23]
    movej(JOne, vel=VELOCITY, acc=ACC)
    mwait()

    # 주유기 detect했을경우 가는 좌표
    # yolo bounding box
    if self.detected_bbox:
        x1, y1, x2, y2 = self.detected_bbox
        center_x = (x1 + x2) / 2
        center_y = (y1 + y2) / 2

        self.get_logger().info(f"[YOLO 기반 이동] 중심 좌표: ({center_x}, {center_y})")

        # (카메라 좌표계 → 로봇 좌표계 변환)
        robot_x = center_x * 0.1 # 스케일링 계수
        robot_y = center_y * 0.1
        robot_z = 150.0 # 고정값 또는 depth

        target_pose = posx(robot_x, robot_y, robot_z, 0, 0, 0)
        movej(target_pose, vel=VELOCITY, acc=ACC, ref=DR_BASE)
    else:
        self.get_logger().warn("[경고] YOLO 좌표가 없어 기본 위치로 이동합니다.")
        # 주유기 detect했을경우 가는 좌표 (하드코딩)
        JTwo = [592.74, 136.66, 183.29, 82.59, -178.16, -102.06]
        movej(JTwo, vel=VELOCITY, acc=ACC)
        mwait()
```

동작: **pump pick** -> 주유구에 삽입

YOLO로 객체(**pump**) 인지 성공

조건 1: **yolo\_callback(msg)**에 YOLO결과 존재

조건 2: 객체(**pump**)인지 성공 -> **detected\_bbox**에 좌표 저장되어 있음

부분 동작: 해당 좌표 **movej**형식에 맞게 변환 해서 이동

YOLO로 객체(**pump**) 인지 실패

부분 동작: 하드코딩된 좌표로 **movej**이동



# 04-4 K-Digital Training [robot\_control] - onrobot\_robot\_control

## ▶ card\_moving

```
# 3. 카드를 집고 계산까지 완료하는 모션
def card_moving(self):
    move1(posx(0.0, 0.0, 130.0, 0, 0, 0), vel=VELOCITY, acc=ACC, radius=0.0, ref=D
    gripper.close_gripper()
    mwait()

    # 사용자 지정(카드가 보이는 좌표)
    J0ne = [-47.17, 45.54, 39.14, 40.73, 109.0, -26.49]
    movej(J0ne, vel=15, acc=15)
    mwait()

    # 카드를 yolo detect 했을 경우 가는 좌표
    if self.detected_bbox:
        x1, y1, x2, y2 = self.detected_bbox
        cx = (x1 + x2) / 2
        cy = (y1 + y2) / 2
        self.get_logger().info(f"[YOLO 카드 이동] 중심 좌표: ({cx}, {cy})")

        # 비례식 기반 간단한 변환 (실제 변환 필요시 조정)
        robot_x = cx * 0.1
        robot_y = cy * 0.1
        robot_z = 103.0 # 높이는 고정 또는 조정 가능

        move1(posx(robot_x, robot_y, robot_z, 0, 0, 0),
              vel=VELOCITY, acc=ACC, ref=DR_BASE)
    else:
        self.get_logger().warn("[YOLO] 감지된 카드 좌표 없음 → 기본 위치로 이동")
        # 카드를 yolo detect 했을 경우 가는 좌표(move1로 이동시 급발진)
        JTwo = [560.20, -70.72, 103.25, 65.47, 138.07, 77.18]
        move1(JTwo, vel=VELOCITY, acc=ACC)
        mwait()
```

동작: 카드 삽입(결제) 및 카드 원위치

YOLO로 객체(creditcard) 인지 성공

조건 1: yolo\_callback(msg)에 YOLO결과 존재

조건 2: 객체(creditcard)인지 성공 -> detected\_bbox에 좌표 저장됨

부분 동작: 해당 좌표 move1형식에 맞게 변환 해서 이동

YOLO로 객체(creditcard) 인지 실패

부분 동작: 하드코딩된 좌표로 move1이동

# 04-4 K-Digital Training [robot\_control] - onrobot\_robot\_control

## ▶ wait\_for\_nudge\_then\_callback

```
# 2. 주유기 주유구에서 빼고 가져다 놓기
# 낮지 함수
def wait_for_nudge_then_comeback(self, path_list=None, vel=100, acc=100, threshold=20.0):
    self.get_logger().info("Z축 방향으로 최소 20move.0N 이상 누르면 다음 동작을 수행합니다.")
    try:
        while rclpy.ok():
            if check_force_condition(axis=DR_AXIS_Z, min=threshold, ref=DR_TOOL):
                self.get_logger().info("Nudge 감지됨! 후속 동작 수행 중")
                break
            time.sleep(0.1)

        movel([posx(0.0, 40.0, 0.0, 0, 0, 0), vel=VELOCITY, acc=ACC, radius=0.0, ref=DR_BASE, mod=DR_MV_MOD_REL])
        # Back to place
        movel(posx(0.0, 0.0, 130.0, 0, 0, 0), vel=VELOCITY, acc=ACC, radius=0.0, ref=DR_BASE, mod=DR_MV_MOD_REL)
        movel(posx(135.0, 0.0, 0.0, 0, 0, 0), vel=VELOCITY, acc=ACC, radius=0.0, ref=DR_BASE, mod=DR_MV_MOD_REL)
        JTwo = [592.74, 196.66, 183.29, 82.59, -178.16, -102.06]
        movel(JTwo, vel=VELOCITY, acc=ACC)
        mwait()
        movel(posx(0.0, 0.0, -80.0, 0, 0, 0), vel=VELOCITY, acc=ACC, radius=0.0, ref=DR_BASE, mod=DR_MV_MOD_REL)
        gripper.open_gripper()

        self.card_moving()

    except Exception as e:
        print(f"예외 발생: {e}")
```

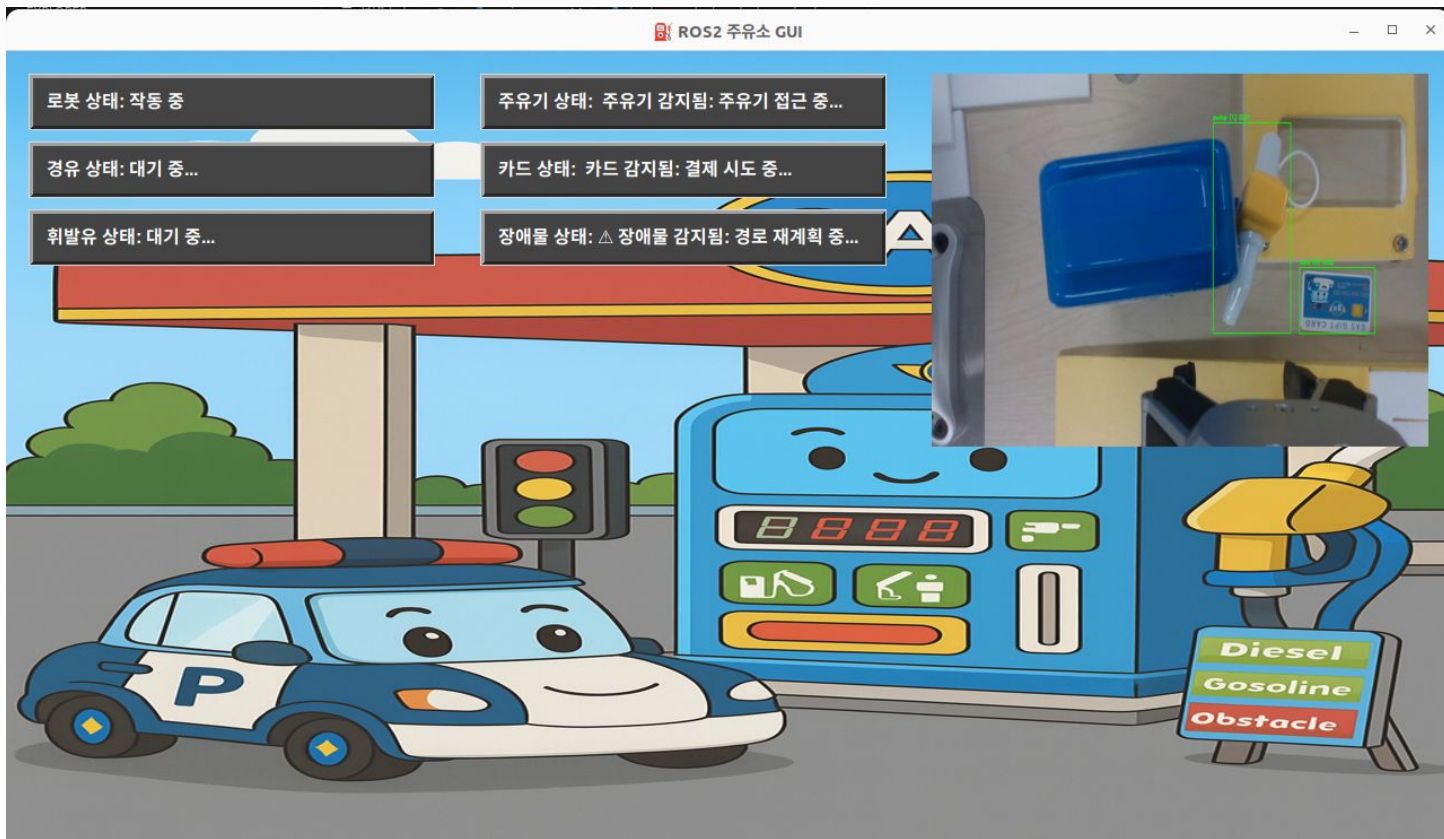
동작: 주유기 원위치

조건: z축 방향으로 20N이상의 외력 작용

함수 설명: ref\_DR\_TOOL에서 제공하는 힘 데이터를 읽어 (AXIS\_Z)방향의 값을 min\_threshold와 비교 조건 만족시 동작(주유기 원위치) 순차적으로 실행

# 04-5 K-Digital Training GUI

## ▶ gui 제작



=> 로봇/기름/주유기/카드/장애물 상태 메시지 출력 및 객체탐지 카메라 출력

# 04-5 K-Digital Training GUI - Yolo 객체 탐지

```
# YOLO 모델 & 카메라 초기화
model = YOLO("/home/rokey/Downloads/best.pt")
cap = cv2.VideoCapture(0) # USB 카메라 인덱스

video_label = tk.Label(main_frame)
video_label.place(x=820, y=20, width=440, height=330) # 오른쪽 하단 영역에 영상 표시

def update_gui():
    ret, frame = cap.read()
    if ret:
        results = model.predict(source=frame, conf=0.4, verbose=False)[0]
        for box in results.boxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            conf = float(box.conf[0])
            cls_id = int(box.cls[0])
            cls_name = model.names[cls_id]
            label = f'{cls_name} ({cls_id}) {conf:.2f}'
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.putText(frame, label, (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(rgb_frame).resize((440, 330))
        imgtk = ImageTk.PhotoImage(image=img)
        video_label.imgtk = imgtk
        video_label.config(image=imgtk)

    root.after(100, update_gui)

update_gui()
```

- **Yolo** 모델로 객체 탐지 추출이후 객체이름, 클래스 이름, 신뢰도를 **gui** 창에 출력

# 04-5 GUI -gui.py

```
# == sub =====
self.create_subscription(String, '/robot/status', self.status_callback, 10)
self.create_subscription(Int32, '/oil', self.diesel_callback, 10)
self.create_subscription(Int32, '/oil', self.gasoline_callback, 10)
self.create_subscription(Bool, '/target/Pump', self.pump_callback, 10)
self.create_subscription(Bool, '/target/Card', self.card_callback, 10)
self.create_subscription(Bool, '/target/Obstacle', self.obstacle_callback, 10)
self.create_subscription(Int32, '/payment/total', self.payment_callback, 10)
# == pub =====
self.payment_pub = self.create_publisher(Int32, '/payment/total', 10)
```

- 각종 토픽을 **sub** 하여 상단 메시지 출력창에 출력

```
def status_callback(self, msg):
    self.get_logger().info(f"[로봇 상태] {msg.data}")
    self.status_message = msg.data

def diesel_callback(self, msg):
    if msg.data == 0:
        self.diesel_message = "⛽ 경유 감지됨: 경유 주유 시작 중..."

def gasoline_callback(self, msg):
    if msg.data == 1:
        self.gasoline_message = "⛽ 휘발유 감지됨: 휘발유 주유 시작 중..."

def pump_callback(self, msg):
    if msg.data:
        self.pump_message = "🚰 주유기 감지됨: 주유기 접근 중..."

def card_callback(self, msg):
    if msg.data:
        self.card_message = "💳 카드 감지됨: 결제 시도 중..."

def obstacle_callback(self, msg):
    if msg.data:
        self.obstacle_message = "⚠️ 장애물 감지됨: 경로 재계획 중..."

def payment_callback(self, msg):
    amount = msg.data # 수신된 한 번의 결제 금액
    self.add_payment(amount)

def add_payment(self, amount: int):
    self.total_payment += amount
    self.payment_pub.publish(Int32(data=self.total_payment))
    self.get_logger().info(f"[결제 누적액] 현재까지 총 결제액: {self.total_payment} 원")
```



# 04-6 [고도화 실패]

K-Digital Training

```
# 병렬 추론 함수
def run_inference(model, frame):
    return model.predict(source=frame, conf=0.4, verbose=False)[0], model

# ROS2 퍼블리셔 노드 클래스
class YoloPublisher(Node):
    def __init__(self):
        super().__init__('yolo_publisher')
        self.publisher_ = self.create_publisher(String, 'yolo_result', 10)

    def publish_detection(self, class_name, x1, y1, x2, y2, conf):
        msg = String()
        msg.data = f"{class_name} {conf:.2f} {x1} {y1} {x2} {y2}"
        self.publisher_.publish(msg)
        self.get_logger().info(f"Published: {msg.data}")

# 메인 실행 함수
def main():
    rclpy.init()
    node = YoloPublisher()

    # 모델 로드
    custom_model = YOLO("/home/rokey/Downloads/gasStation_1.pt")
    coco_model = YOLO("yolov8n.pt")

    # 카메라 열기
    cap = cv2.VideoCapture(6)
    executor = ThreadPoolExecutor(max_workers=2)

    MAX_RETRY = 5
    retry_count = 0

    while cap.isOpened() and rclpy.ok():
        ret, frame = cap.read()

        if not ret:
            retry_count += 1
            node.get_logger().warn(f"[WARN] 카메라 입력 실패... 재시도 {retry_count}/{MAX_RETRY}")
            time.sleep(1)
            if retry_count >= MAX_RETRY:
                node.get_logger().error("[ERROR] 최대 재시도 횟수를 초과했습니다. 종료.")
                break
            continue
        retry_count = 0
        input_frame = frame.copy()

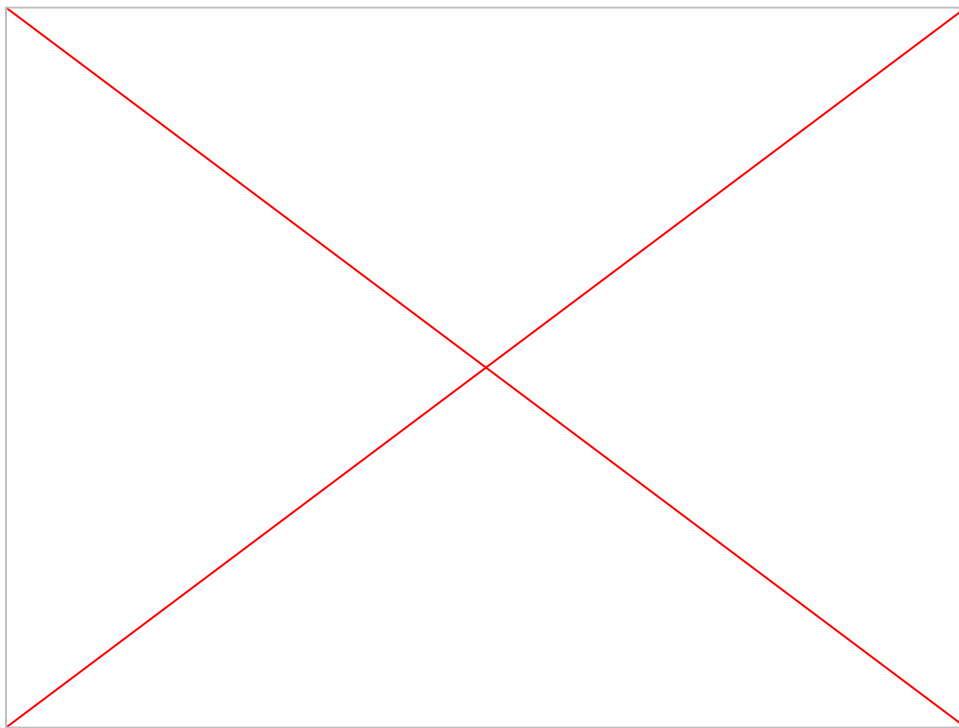
        # 병렬 추론 실행
        future_custom = executor.submit(run_inference, custom_model, input_frame)
        future_coco = executor.submit(run_inference, coco_model, input_frame)
```

- **labeling**을 통해 **creditcard**와 **pump**를 학습시킨 **custom yolo data**와 **stop sign**이 내장된 **coco dataset**을 통합해 병렬추론을 실행하려 했음
- **detect**는 되지만 인식되는 **config**값을 수정해도 **coco dataset**에 있는 **class**들과 충돌하여 사용하지 못함

- ↓
- **coco dataset**을 제외시키고 **custom data**만 사용하여 **credit card, pump**인식하고 **stop sign**은 하드코딩해서 제거하는 것으로 **변경**

# 04-6 Auto fuel system

▶ 동영상



## 프로젝트 결과물에 대한 완성도 평가(10점 7점)

동작을 시나리오대로 구현을 했고 STT도 인식을 해서 움직이지만 yolo bounding box에서 추출된 좌표로 움직이는 과정을 완성하지 못해 예외처리로 진행한 하드코딩 좌표로 움직여서 완성도가 낮아졌다. 하지만 여러 topic들을 유기적으로 연결시켜서 코드 간 pub-sub이 원활하게 이루어지는 동작을 로그 출력을 통해 확인했다.

시간도 짧았고 인원도 적은 상태에서 다양한 고도화를 실행했고, 구현하지 못 했다고 하더라도 결국 학습이 있었기 때문에 각자 얻어가는 부분이 있다고 생각한다.

## 추후 개선점이나 보완할 점

1. 프롬프트 고도화(STT음성 인식 후 LLM이 판단해서 행동하도록 개선)를 할 수 있음
2. 구현하지 못한 yolo bounding box 좌표를 통해 객체에 다가가는 코드를 개선 할 수 있음
3. custom dataset과 coco dataset을 통합하여 pump, card, stop sign을 한 번에 인식하도록 해 하드코딩 하는 경우의 수를 줄일 수 있음
4. service 통신을 더 활용하여 실시간성을 높일 수 있음



## 자체 평가 의견

## 느낀 점 / 경험한 성과

협동 로봇이 정해진 테스크 환경에서 움직임을 티칭 시키는 과정에서 물리적인 충돌이 발생하지 않도록 하기 위한 경유점 설정이 중요하다는 것을 느꼈습니다. 더 나아가 협동 로봇 티칭 전에 안전구역설정 과정이 필수적이라고 느꼈습니다.

최근 LLM이 각광받으며 관심이 생겼었는데 좋은 기회로 프로젝트에서 구현 할 수 있어서 좋았다. 프롬프트 작성을 하며 키워드 추출 구조와 로직을 파악 할 수 있었다. STT에 포함 서비스 기능이 포함된 코드를 분석하며 topic으로 분해하는 방법도 학습하고 여러 노드를 작성하며 ROS2의 통신에 대해 더 깊고 자세하게 배울 수 있는 기회였다. 또한 여러가지 케이스에 대해서 고도화를 하고싶었는데 시간 분배를 적절하게 하지 못해서 구현하지 못 한게 아쉽다는 생각이 많이 들었다. 구현해놓은 코드를 더 면밀하게 파악해서 활용 가능한 부분을 파악하고 고도화시키는 것을 목표로 삼게 되었다.

로봇암, 비전, 음성인식을 하나의 시스템으로 통합하는 과정에서 기술간 연동의 중요성을 경험하였고, 주유라는 위험요소가 존재하는 작업에 로봇을 적용하면서 제어와 안전 설계의 필요성을 깨달았습니다. 또한 사용작가 쉽고 안전하게 사용할수있도록 GUI와 음성 인터페이스를 개선하며 사용자 경험의 중요성을 체감하였으며, 결과적으로 음성명령부터

STT나 LangChain과 같이 설치하여 사용하는 외부 모듈 간에는 버전 차이로 인해 충돌이 발생할 수 있다는 점을 충분히 인지하지 못한 채 코드를 구성하였습니다. 각 모듈의 버전에 따라 내장 함수의 사용 방식이나 알고리즘 구조가 달라질 수 있기 때문에, 이러한 차이를 사전에 인식하고 충돌이 발생하지 않도록 버전 호환성을 고려한 코드 작성이 필요했음을 느낍니다.

## 잘 한 부분 / 아쉬운 부분

- 잘한 점:
  1. m0609로봇암, Yolo 비전, STT 음성인식, GUI 등 다양한 기술을 유기적으로 연동하여 목표한 자동화 기능을 구현한점.
  2. STT에서 추출 가능한 여러 단어를 YOLO에 mapping해서 다양성 확보, 각 노드 간 원활한 통신을 구현함
- 아쉬운 점:
  1. 다양한 차종에 대한 급유구 형태와 위치 차이를 고려해 보다 보편적으로 적용할 수 있는 설계가 미흡했고, 음성 인식에서 배경 소음 등 환경적 요소에 대한 예외처리 기능이 부족했던 점
  2. YOLO를 조금 더 고도화하여 차량 모델을 인식하고 자동으로 경유/취발유를 판단하는 프롬프트를 작성하지 못함