Team : B5

팀원 I 김주원 류재준 양준혁 이상우

목차 페이지

01

팀원 소개

김주원 : Robot movement algorithm, Mapping

류재준 : Database, Integration, Main node

양준혁 : Mapping, Kiosk gui, Monoitoring gui

이상우 : World, Monoitoring gui

CHAPTER 02

목표 (Goal)

자율 주행 로봇을 활용한 자동 주차 시스템

본 프로젝트는 자율 주행 로봇을 활용한 자동 주차 시스템을 구현하는 것을 목표로 함

이를 위해, CCTV를 활용한 실시간 모니터링과 특정 CCTV에 YOLO 기반 객체 탐지 기술을 적용하여 차량의 입출고 상황을 인식함

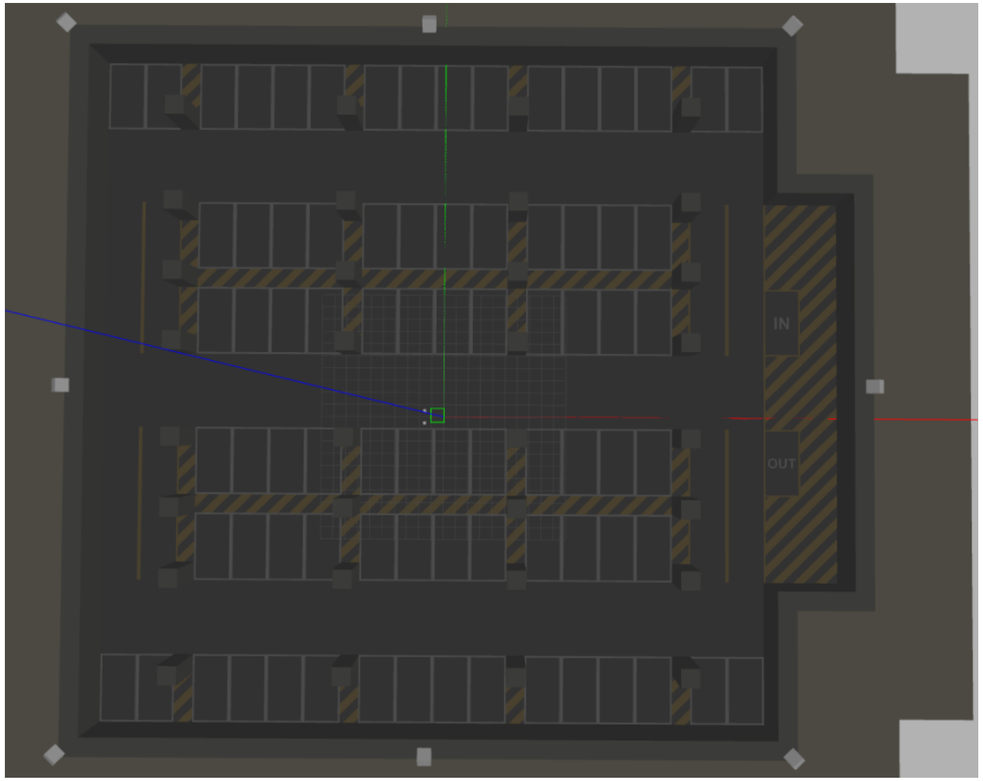
또한 주차장 상태, 차량 정보, 로봇 상태 등을 데이터베이스로 관리하여 로봇의 자율 주행 경로를 최적화하는 시스템을 구축함

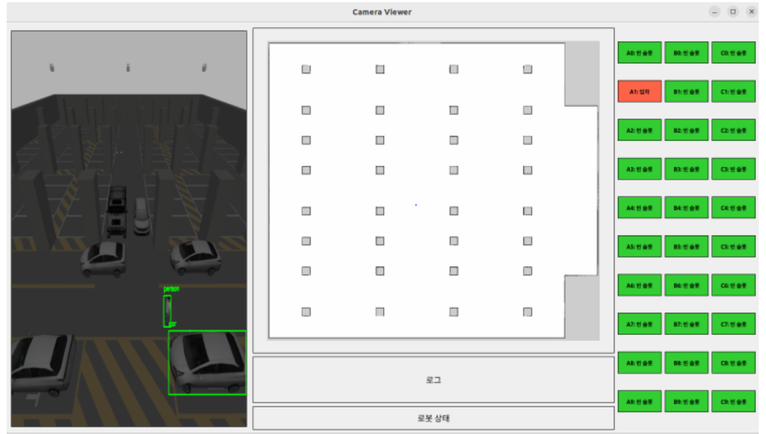
또한, 실시간 로봇의 위치 모니터링과 주차장 상태 시각화 시스템을 통해 주차 현황을 한눈에 파악할 수 있으며, 주차 상황에 맞춘 최적의 경로로 로봇이 자동 주차할 수 있도록 설계함

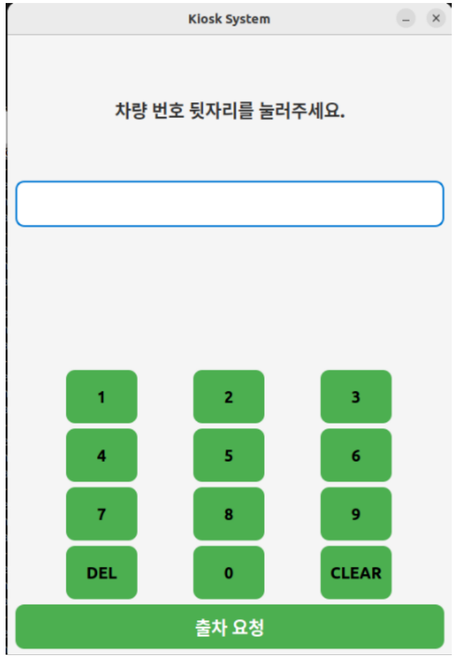
이를 통해, 주차 효율성을 높이고 사용자 편의를 증대하는 스마트 주차 시스템을 구현하기 위해 노력함

CHAPTER 02

목표 (Goal)







CHAPTER 03

Demo

Demo

CHAPTER 04

코드 리뷰 (Code Review)

주요 Node

1) Main (central_control_node)
2) Monitoring (gui_minimap)
3) Kiosk (kiosk_gui)
4) Launch file

CHAPTER 04

코드 리뷰 (Code Review) | Map & Gui

Map (World)

```
<sensor name='camera_wall_1' type='camera'>
  <camera name='head'>
    <horizontal_fov>1.39626</horizontal_fov>
    <image>
      <width>800</width>
      <height>800</height>
      <format>R8G8B8</format>
    </image>
    <clip>
      <near>0.02</near>
      <far>300</far>
    </clip>
  </camera>
  <plugin name='camera_controller_w1' filename='libgazebo_ros_camera.so'>
    <alwaysOn>1</alwaysOn>
    <updateRate>30.0</updateRate>
    <cameraName>camera_wall_1</cameraName>
    <imageTopicName>image_raw</imageTopicName>
    <cameraInfoTopicName>camera_info</cameraInfoTopicName>
    <frameName>camera_wall_1_frame</frameName>
    <node_name>camera_node_w1</node_name>
  </plugin>
</sensor>
<self_collide>0</self_collide>
<enable_wind>0</enable_wind>
<kinematic>0</kinematic>
```

Gui_minimap (Node)

```
self.camera_topics = {
    'camera_wall_1/image_raw': 0,
    'camera_wall_2/image_raw': 1,
    'camera_wall_3/image_raw': 2,
    'camera_wall_4/image_raw': 3,
    'camera_wall_5/image_raw': 4,
    'camera_wall_6/image_raw': 5,
    'camera_wall_7/image_raw': 6,
    'camera_wall_8/image_raw': 7
}

self.subscribers = []
for topic in self.camera_topics.keys():
    self.subscribers.append(
        self.create_subscription(
            Image,
            topic,
            self.image_callback,
            self.qos
        )
    )
```

```
def image_callback(self, msg):
    try:
        topic = msg.header.frame_id.split('_frame')[0] + '/image_raw'
        if topic in self.camera_topics:
            index = self.camera_topics[topic]
            cv_image = self.bridge.imgmsg_to_cv2(msg, "rgb8")

            # YOLO 적용 (3번 카메라에만)
            if index == 2:
                cv_image = self.detect_vehicle(cv_image)

            h, w, ch = cv_image.shape
            bytes_per_line = ch * w
            qt_image = QImage(cv_image.data, w, h, bytes_per_line, QImage.Format_RGB888)
            pixmap = QPixmap.fromImage(qt_image)
            self.image_labels[index].setPixmap(pixmap)
    except Exception as e:
        self.get_logger().error(f'Error processing image: {str(e)}')
```

CHAPTER 04

코드 리뷰 (Code Review) | Map

라이다의 범위가 너무 제한이 많기에 model.sdf (waffle_pi) 에서 라이다 range를 조정해준다

update_rate : 디폴트 5 → 설정값 20
samples : 디폴트 360 → 설정값 720
range : max 디폴트 3.5 → 설정값 30

```
<sensor name="hls_lfcd_lds" type="ray">
  <always_on>true</always_on>
  <visualize>true</visualize>
  <pose>-0.064 0 0.121 0 0 0</pose>
  <update_rate>20</update_rate> <!-- 5 -->
  <ray>
    <scan>
      <horizontal>
        <samples>720</samples> <!-- 360 -->
        <resolution>1.000000</resolution>
        <min_angle>0.000000</min_angle>
        <max_angle>6.280000</max_angle>
      </horizontal>
    </scan>
    <range>
      <min>0.120000</min>
      <max>30</max> <!-- 3.5 -->
      <resolution>0.015000</resolution>
    </range>
    <noise>
      <type>gaussian</type>
      <mean>0.0</mean>
      <stddev>0.01</stddev>
    </noise>
  </ray>
```

카토그래퍼 수행 시 실제로 탐지되는 영역이 다르기에
/turtlebot3_cartographer/config
에 있는 “turtlebot3_lds_2d.lua” 파일에서도 수정이 필요함

max_range : 디폴트 3.5 → 설정값 20
constraint_builder.min_score :디폴트 0.65 → 설정값 0.8
constraint_builder.global_localization_min_score : 디폴트 0.7 → 설정값 0.85

```
47 MAP_BUILDER.use_trajectory_builder_2d = true
48
49 TRAJECTORY_BUILDER_2D.min_range = 0.12
50 TRAJECTORY_BUILDER_2D.max_range = 20
51 TRAJECTORY_BUILDER_2D.missing_data_ray_length = 3.
52 TRAJECTORY_BUILDER_2D.use_imu_data = false
53 TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching = true
54 TRAJECTORY_BUILDER_2D.motion_filter.max_angle_radians = math.rad(0.1)
55
56 POSE_GRAPH.constraint_builder.min_score = 0.8
57 POSE_GRAPH.constraint_builder.global_localization_min_score = 0.85
58
59 -- POSE_GRAPH.optimize_every_n_nodes = 0
```

CHAPTER 04

코드 리뷰 (Code Review) | GUI

Gui_minimap Node

```python
def image_callback(self, msg):
    try:
        topic = msg.header.frame_id.split('_frame')[0] + '/image_raw'
        if topic in self.camera_topics:
            index = self.camera_topics[topic]
            cv_image = self.bridge.imgmsg_to_cv2(msg, "rgb8")

            # YOLO 적용 (3번 카메라에만)
            if index == 2:
                cv_image = self.detect_vehicle(cv_image)

            h, w, ch = cv_image.shape
            bytes_per_line = ch * w
            qt_image = QImage(cv_image.data, w, h, bytes_per_line, QImage.Format_RGB888)
            pixmap = QPixmap.fromImage(qt_image)
            self.image_labels[index].setPixmap(pixmap)
    except Exception as e:
        self.get_logger().error(f'Error processing image: {str(e)}')
```

```python
######################## yolov8 모델 로드 ###################
self.yolo_model = YOLO('yolov8m.pt')
self.vehicle_pub = self.create_publisher(Bool, '/vehicle_detected', 10)
self.vehicle_classes = ['car', 'truck', 'bus', 'motorcycle', 'person']
self.detection_times = deque()
self.roi = (450, 530, 640, 640)
```

```python
def detect_vehicle(self, frame):
    results = self.yolo_model(frame)
    detected = False

    for result in results:
        for box in result.boxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0].tolist())
            cls = int(box.cls)
            label = self.yolo_model.names[cls]

            if label in self.vehicle_classes:
                if (self.roi[0] <= x1 <= self.roi[2]) and (self.roi[1] <= y1 <= self.roi[3]):
                    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)
                    detected = True
```

CHAPTER 04

코드 리뷰 (Code Review) | Kiosk

Kiosk Node

```python
# ROS2 노드 클래스 (서비스 클라이언트)
class KioskNode(Node):
    def __init__(self):
        super().__init__('kiosk_node')
        self.cli = self.create_client(ExitRequest, 'exit_request')
        # 중앙 관제 시스템이 켜지는 것을 기다리는 로직
        while not self.cli.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Service /exit_request not available, waiting...')
        self.req = ExitRequest.Request()
        self.future = None

    # 서비스 요청을 서버로 보내는 메서드 (비동기)
    def send_exit_request(self, car_number):
        self.req.car_number = car_number
        self.future = self.cli.call_async(self.req)
        return self.future
```

Main Node

```python
# ExitRequest 서비스 서버 생성
self.exit_request_service = self.create_service(
    ExitRequest,
    'exit_request',
    self.handle_exit_request
)
self.get_logger().info("ExitRequest 서비스가 준비되었습니다.")
```

```python
def handle_exit_request(self, request, response):
    car_number = request.car_number
    self.get_logger().info(f"Exit request received for car number: {car_number}")
    # DB에서 요금 계산
    fee = self.db_manager.calculate_fee(car_number)
    # DB에서 입차 시간 가져오기
    entry_time = self.get_entry_time(car_number)
    self.get_logger().info(f"fee: {fee}, entry_time: {entry_time}")

    if fee > 0 and entry_time:
        # Task_Log에 출차 요청 기록 (end_time은 결제 후 설정)
        slot_id = self.get_slot_id(car_number)
        if slot_id is None:
            self.get_logger().error(f"No Parking_Slot found for vehicle_id: {car_number}")
            response.status = False
            response.entry_time = ""
            response.fee = 0
            response.log = "출차 요청 실패: 슬롯을 찾을 수 없습니다."
            self.log_publisher.publish(String(data=f"{car_number} 차량의 출차 요청이 실패했습니다: 슬롯을 찾을 수 없습니다."))
            return response

        task_data = {
            "robot_id": 1,  # 예시 로봇 ID, 실제 환경에 맞게 설정 필요
            "vehicle_id": car_number,
            "vehicle_img": "default_img.jpg",  # 실제 차량 이미지로 대체 필요
            "slot_id": slot_id,
            "task_type": "출차",
            "start_time": datetime.now().isoformat(timespec='seconds'),  # 마이크로초 제외
            "end_time": None,
            "status": "결제 중"
```

CHAPTER 04

코드 리뷰 (Code Review) | Main

```
def vehicle_detected_callback(self, msg):
    self.get_logger().info(f"Vehicle detected: {msg.data}")
    self.log_publisher.publish(String(data=f"Vehicle detected: {msg.data}"))
    vehicle_id = msg.data  # 차량의 ID
```

CCTV 카메라로 차량을 감지하고, 차량이 인식되면 입차 프로세스를 시작함
이 과정에서 vehicle_detected_callback() 함수가 호출되며, 차량 정보를 데이터베이스에 저장함

1) 차량 감지: CCTV 카메라로 차량을 감지하면 /entry_camera/vehicle_detected 토픽을 통해 차량 정보를 수신함

2) 로그 기록: 차량 탐지 정보를 /central_control/logs 토픽에 퍼블리시하여, 실시간으로 차량 감지를 기록함

CHAPTER 04

코드 리뷰 (Code Review) | Main

```python
def handle_exit_request(self, request, response):
    car_number = request.car_number
    self.get_logger().info(f"Exit request received for car number: {car_number}")
    # DB에서 요금 계산
    fee = self.db_manager.calculate_fee(car_number)
    # DB에서 입차 시간 가져오기
    entry_time = self.get_entry_time(car_number)
    self.get_logger().info(f"fee: {fee}, entry_time: {entry_time}")

    if fee > 0 and entry_time:
        # Task_Log에 출차 요청 기록 (end_time은 결제 후 설정)
        slot_id = self.get_slot_id(car_number)
        if slot_id is None:
            self.get_logger().error(f"No Parking_Slot found for vehicle_id: {car_number}")
            response.status = False
            response.entry_time = ""
            response.fee = 0
            response.log = "출차 요청 실패: 슬롯을 찾을 수 없습니다."
            self.log_publisher.publish(String(data=f"{car_number} 차량의 출차 요청이 실패했습니다: 슬롯을 찾을 수 없습니다."))
            return response

        task_data = {
            "robot_id": 1,  # 예시 로봇 ID, 실제 환경에 맞게 설정 필요
            "vehicle_id": car_number,
            "vehicle_img": "default_img.jpg",  # 실제 차량 이미지로 대체 필요
            "slot_id": slot_id,
            "task_type": "출차",
            "start_time": datetime.now().isoformat(timespec='seconds'),  # 마이크로초 제외
            "end_time": None,
            "status": "결제 중"
        }
        self.db_manager.insert_data("Task_Log", task_data)
        self.log_publisher.publish(String(data=f"Task_Log created for vehicle {car_number} with status 'Payment Pending'"))

        # 응답 설정
        response.status = True
        response.entry_time = entry_time
        response.fee = fee
        response.log = "출차 요청이 성공적으로 접수되었습니다. 요금을 결제해주세요."

        # 로그 퍼블리시
        self.log_publisher.publish(String(data=f"{car_number} 차량의 출차 요청이 접수되었습니다. 요금: {fee}원"))
```

사용자 출차 요청이 들어오면 요금을 계산하고
DB에 기록 및 작업 상태를 업데이트함

이 과정에서 handle_exit_request() 함수가 호출됨

- 출차 요청 수신: /exit_request 서비스로 출차 요청이 들어오면, 차량의 출차 프로세스가 시작됨
- 요금 계산: 차량의 입차 시간을 기반으로 주차 요금을 계산함
- DB 업데이트: 출차 작업을 Task_Log 테이블에 기록하고, 상태를 '결제 중'으로 설정함
- 로그 기록: 출차 요청 결과를 /central_control/logs 토픽에 퍼블리시하여 기록을 남김

CHAPTER 04

코드 리뷰 (Code Review) | Main

```python
def payment_confirmation_callback(self, msg):
    self.get_logger().info(f"Payment confirmed: {msg.data}")
    self.log_publisher.publish(String(data=f"Payment confirmed: {msg.data}"))
    # 결제 정보 파싱 및 DB 기록
    payment_info = self.parse_payment_info(msg.data)
    if payment_info:
        # Prepare data for Payment_Log table
        payment_data = {
            "vehicle_id": payment_info.get("vehicle_id"),
            "entry_time": payment_info.get("entry_time"),
            "exit_time": payment_info.get("exit_time"),
            "total_fee": payment_info.get("total_fee"),
            "payment_method": payment_info.get("payment_method"),
            "timestamp": datetime.now().isoformat()
        }
        self.db_manager.insert_data("Payment_Log", payment_data)
        self.log_publisher.publish(String(data=f"Payment recorded for vehicle {payment_data.get('vehicle_id')}"))

        # Task_Log을 업데이트하여 작업을 완료("Exited")로 표시
        update_data = {
            "end_time": payment_data["exit_time"],
            "status": "출차 중"
        }
        # "Payment Pending" 상태인 'Exit' Task_Log 찾기
        task_logs = self.db_manager.fetch_data(
            table="Task_Log",
            columns="task_id, slot_id",
            conditions=["vehicle_id = ?", "task_type = ?", "status = ?"],
            parameters=[payment_data['vehicle_id'], "출차", "결제 중"]
        )
        if task_logs:
            task_id, slot_id = task_logs[-1]  # 가장 최근 Task_Log 사용
            self.db_manager.update_data(
                table="Task_Log",
                data=update_data,
                conditions=["task_id = ?"],
                parameters=[task_id]
            )
            self.log_publisher.publish(String(data=f"Task_Log updated for vehicle {payment_data.get('vehicle_id')}"))

            # Parking_Slot을 업데이트하여 슬롯을 비어있는 상태로 표시
            update_slot_data = {
                "vehicle_id": None
            }
```

결제가 완료되면 payment_confirmation_callback() 함수가 호출되어 결제 정보를 DB에 기록하고, 작업 로그를 업데이트

- 결제 정보 파싱: /payment/confirmation 토픽으로부터 결제 정보를 수신함
- DB에 결제 내역 기록: Payment_Log 테이블에 결제 내역을 기록함
- 슬롯 해제 및 Task_Log 업데이트: 결제가 완료되면 해당 차량의 주차 슬롯을 빈 상태로 설정하고, Task_Log 상태를 '출차 중'으로 변경함

CHAPTER 04

코드 리뷰 (Code Review) | Database

Database는 자동 주차 시스템의 데이터 관리 허브로, 주차장의 모든 작업을 기록함

이 시스템의 주요 작업(입차, 출차, 결제 등)을 데이터베이스에 기록 및 조회하며, 실시간 주차장 상태를 관리함

테이블 이름	설명	중요 컬럼
Parking_Slot	주차장 슬롯 정보	slot_id, slot_name, vehicle_id
Task_Log	작업(입차/출차) 로그	task_id, task_type, status
Robot_Info	로봇의 현재 상태	robot_id, status, location_x, location_y
Parking_Fee_Policy	요금 정책	base_time, base_fee, 추가 요금 정보
Payment_Log	결제 내역 로그	payment_id, total_fee, payment_method
System_Logs	시스템 로그 (오류, 이벤트)	log_id, event_type, event_details

CHAPTER 05

한계점

1) 실린더 및 Plate의 기능
원래 구현하려던 로직은 주차로봇이 차량 밑에서 차량을 들어올리고 차량을 연결(구속)시키는 로직이었음
⇒ 실린더를 prismatic type으로 설정하여 수직이동을 해보려 했지만 plugin 이슈로 인해 구현하지 못함

2) 메카넘 바퀴 적용 미구현
⇒ 좌우측 평행 이동이 가능한 바퀴지만 실질적으로 구현하지 못함

3) 여러 층의 지하주차장, 다양한 구조나 좁은 주차장에 적용 어려움

CHAPTER 06

추후 계획

1) 모든 과정이 정상적으로 통합되지 않아서, 완벽한 통합을 해볼 예정

2) 시간이 된다면 구현하지 못했던 부분을 정상 구동이 가능하도록 구현해볼 예정