

# 음식점 서빙 시스템 구축

F-4조

기태훈  
김현아  
장석환  
홍유진

# INDEX

---

- 시나리오
- 시스템 설계
- 필수 구현요소

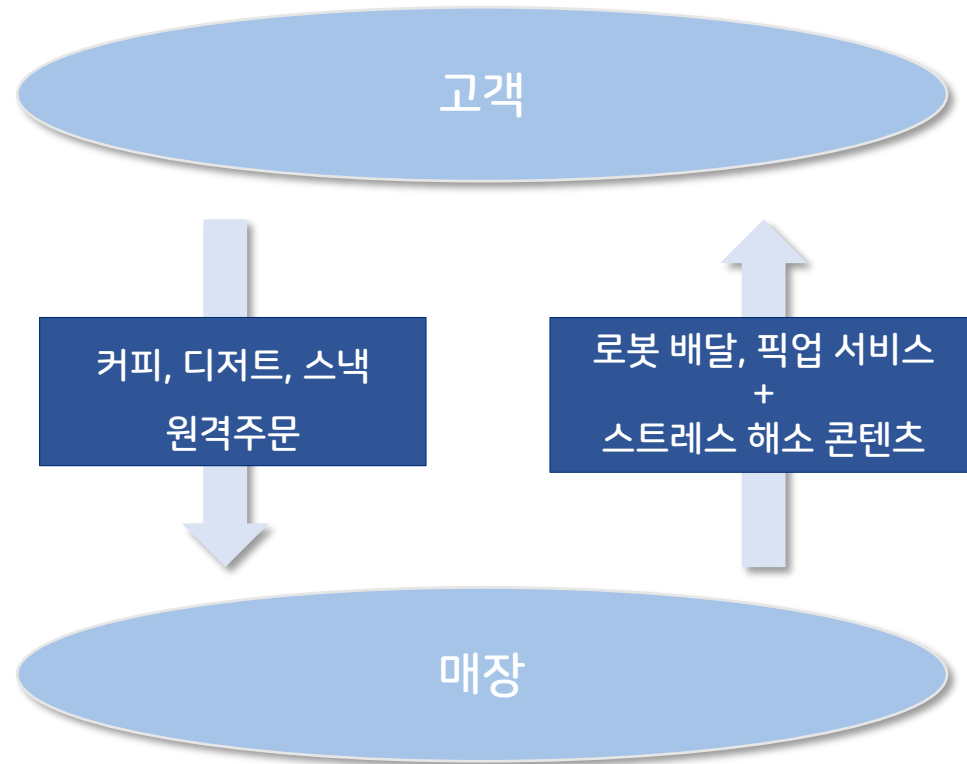
# 시나리오

1. Business

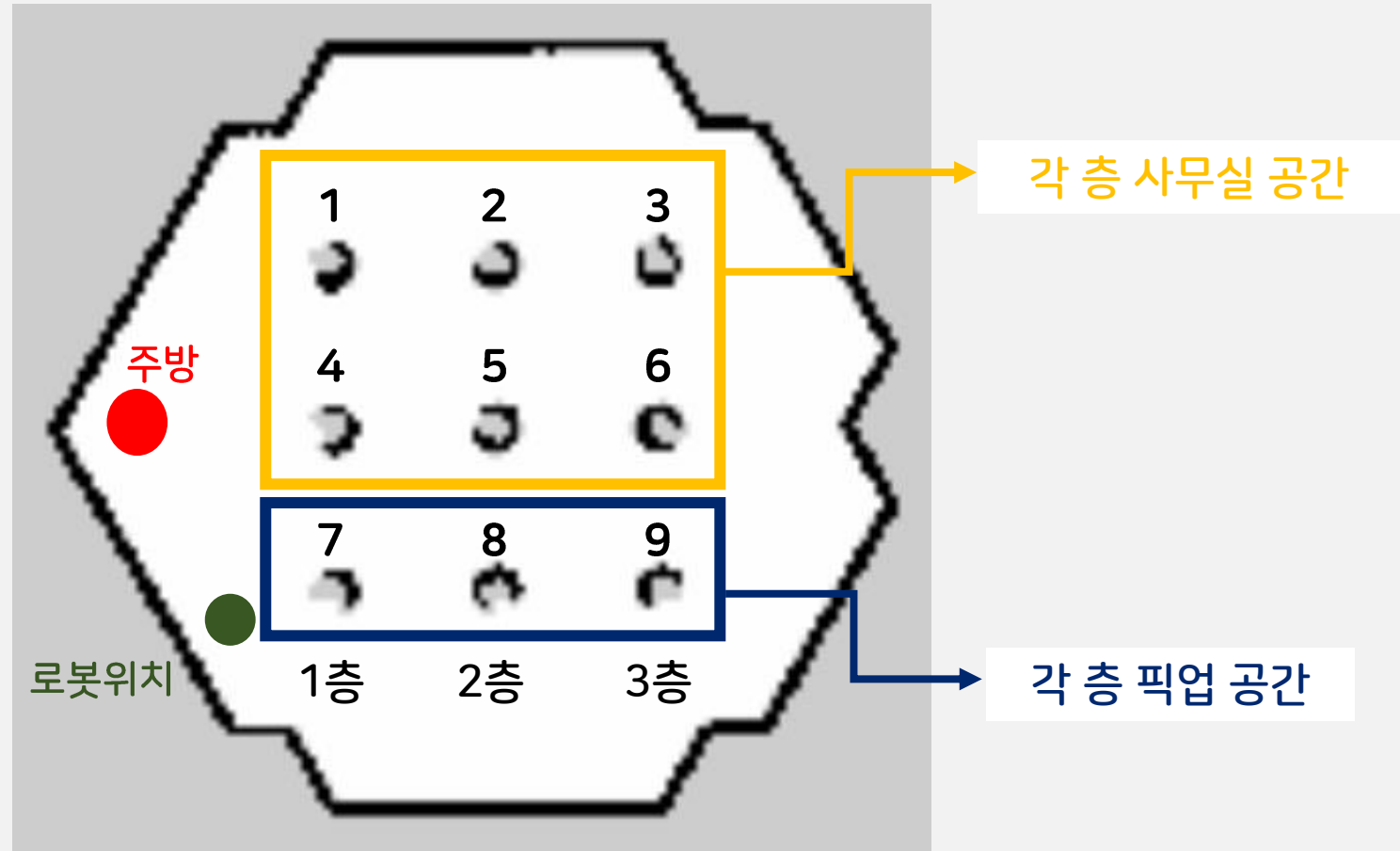
2. Situation

# 1. Business

사내 카페와 서빙 로봇을 결합한 커피 배달 서비스 구축



## 2. Situation



# 시스템 설계

1. 요구 조건
2. 노드 구성

# 1. 요구 조건

## 테이블 오더

- 실제 카페와 유사한 환경 구현
- 다양한 메뉴 및 커스텀 옵션 제공
- 겨울 시즌 메뉴 추가 (군밤고구마 라떼 , 군고구마, 붕어빵)
- 배달과 픽업으로 주문 세분화
- 주문 취소 사유를 제공 - 고객 납득 가능성 & 만족도 향상

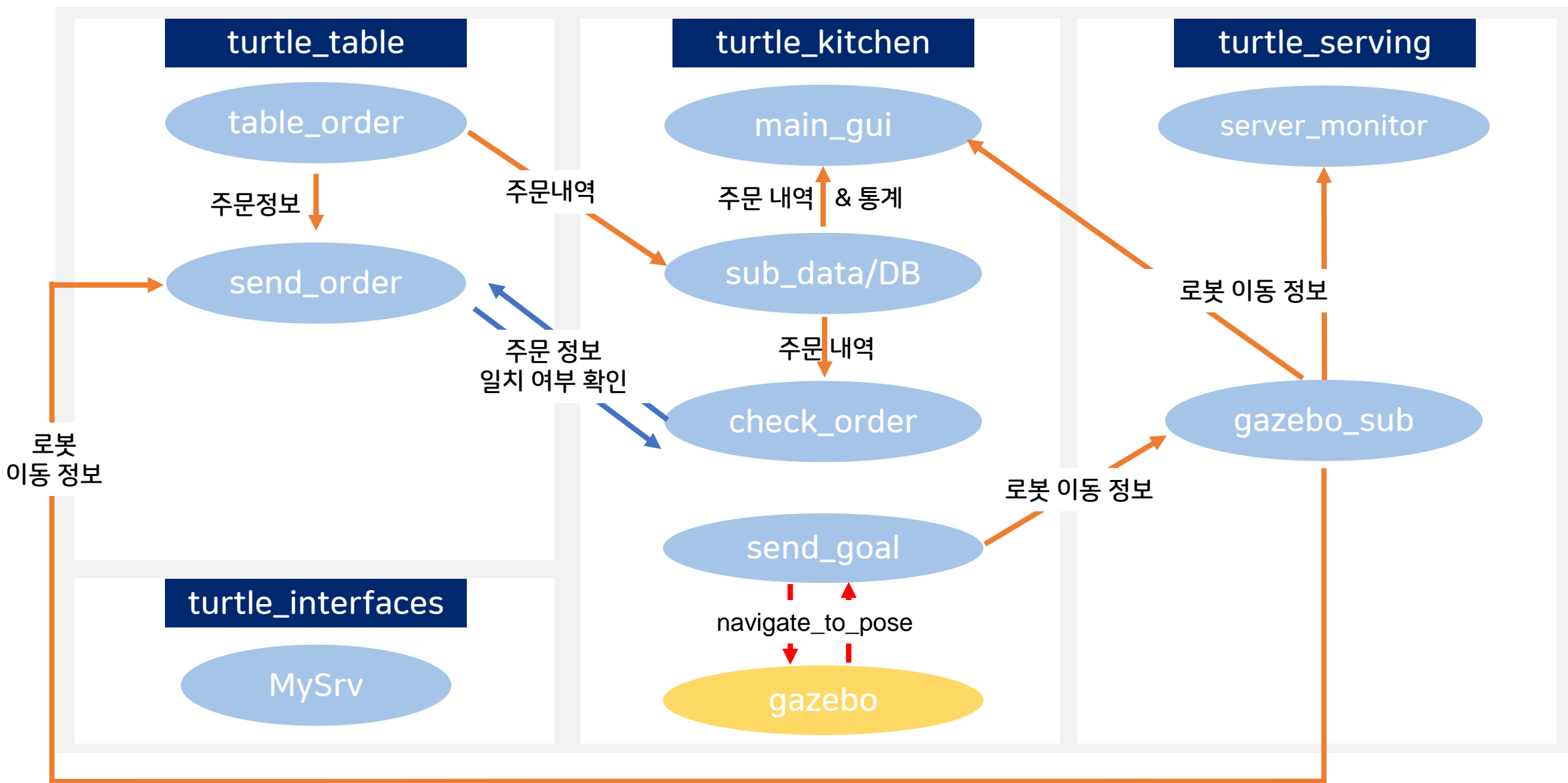
## 주방 디스플레이

- 테이블 오더와 주방 디스플레이 주문 정보를 비교하여 오조리 방지
- 주문 취소 사유를 제공해 고객의 만족도 증가
- 조리 완료 시 서빙 로봇에게 정확한 테이블 번호로 배달 명령 전달
- 총 매출액과 Top3 판매 메뉴를 분석하여 서비스 개선

## 서빙 로봇

- 눈 모양을 통해 친근감 제공
- 친절한 메시지와 퀴즈를 통해 고객에게 특별한 경험 제공
- 다른 테이블로 배달 기능 제공 - 팀 단위 주문에 유용

## 2. 노드 구성

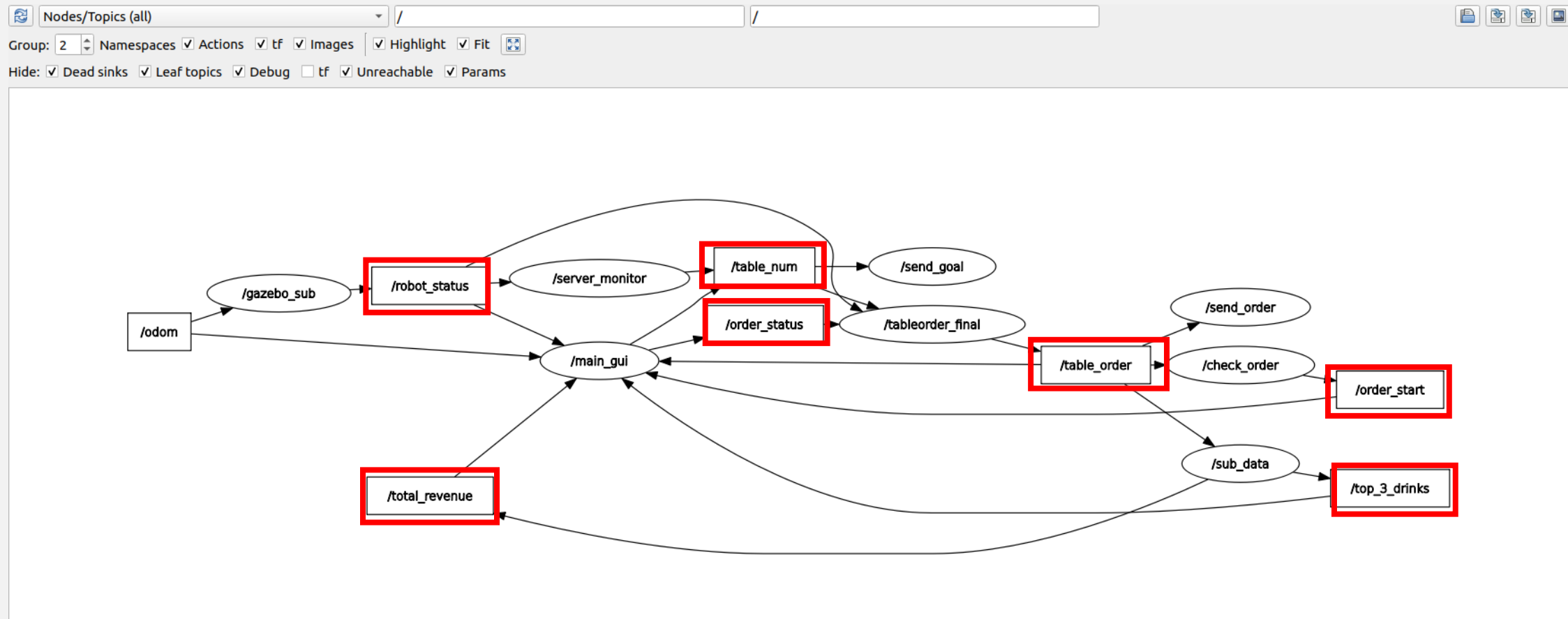




# 필수 구현 요소

1. 토픽/서비스/액션
2. 메시지 인터페이스
3. 테이블 오더
4. 주방 디스플레이
5. 데이터 베이스 저장 & 통계
6. 서빙 로봇
7. Logging
8. QoS

# 1. 토픽/서비스/액션 - 토픽



# 1. 토픽/서비스/액션 - 토픽

---

/robot\_status (String)

로봇 상태 (대기중, 이동중, 이동완료)

/table\_num (Int32)

로봇이 이동해야 할 테이블 번호

/table\_order (String)

고객이 주문한 메뉴와 테이블 번호

/order\_start (String)

주문 시작 또는 확인되었다는 메시지 전송

/order\_status (String)

주문 진행 상태 정보

/top\_3\_drinks (String)

가장 많이 팔린 음료 Top3 정보 (DB 통계에 사용)

/total\_revenue (Int32)

총 매출액 정보 (DB 통계에 사용)

# 1. 토픽/서비스/액션 - 액션

## send\_goal

```
31
32 # NAV2로 목표를 보내기 위한 액션 클라이언트 설정      액션 클라이언트 설정
33 self.action_client = ActionClient(self, NavigateToPose, 'navigate_to_pose')
34
35
36 # 노드가 시작될 때 initial pose를 한 번만 설정      로봇의 초기 위치 설정
37 self.set_initial_pose_service_client = self.create_client(SetInitialPose, '/set_initial_pose')
38
39
```

```
84 def create_goal_pose(self, table_num):      테이블 번호에 따른 위치 설정
85     # 목표 위치를 정의하는 함수. 테이블 번호에 따라 위치 설정
86     goal_pose = PoseStamped()
87     goal_pose.header.frame_id = "map"
88     goal_pose.header.stamp = self.get_clock().now().to_msg()
89
```

# 1. 토픽/서비스/액션 - 액션

## send\_goal

```
91     if table_num == 0: #초기 위치
92         goal_pose.pose.position.x = 0.0
93         goal_pose.pose.position.y = 0.0
94         goal_pose.pose.orientation.w = 1.0 # 방향 설정 (왼쪽 위 45도 각도로)
95         goal_pose.pose.orientation.z = 0.0
96         # 필요에 따라 다른 테이블 번호의 목표 위치 추가
97
98     elif table_num == 1:
99         goal_pose.pose.position.x = 0.55
100        goal_pose.pose.position.y = 1.20
101        goal_pose.pose.orientation.w = 0.9239 # 방향 설정 (왼쪽 위 45도 각도로)
102        goal_pose.pose.orientation.z = 0.3827
103
104
105    elif table_num == 2:
106        goal_pose.pose.position.x = 1.63
107        goal_pose.pose.position.y = 1.20
108        goal_pose.pose.orientation.w = 0.9239 # 방향 설정 (왼쪽 위 45도 각도로)
109        goal_pose.pose.orientation.z = 0.3827
110
```

# 1. 토픽/서비스/액션 - 액션

## gazebo\_sub

```
44
45 def update_status(self):
46     # 초기 대기 위치 확인 (예: x=-2, y=-0.5 주변에 있을 경우 "대기중"으로 설정, 가제보 중심(가운데)에서 설정한 초기위치)
47     if self.current_position and \
48         (self.current_position.x <= -1.9 and self.current_position.x >= -2.1) and \
49         (self.current_position.y <= -0.4 and self.current_position.y >= -0.6):
50         self.status = "대기중"
51
52     # 목표 위치와 현재 위치 간의 거리 계산
53     elif self.current_position and self.goal_position:
54         distance_to_goal = math.sqrt(
55             (self.current_position.x + 2.0 - self.goal_position.x) ** 2 +
56             (self.current_position.y + 0.5 - self.goal_position.y) ** 2
57         )
58
59         # 목표 위치 근처에 있을 경우 "이동 완료"
60         if distance_to_goal < self.tolerance:
61             self.status = "이동 완료"
62         else:
63             self.status = "이동중"
64
65     # 상태 출력 및 퍼블리싱
66     if self.status != self.previous_status:
67         self.get_logger().info(f"현재 상태: {self.status}")
68         status_msg = String()
69         status_msg.data = self.status
70         self.status_publisher.publish(status_msg)
71         self.previous_status = self.status # 상태 업데이트
72
73
74
```

# 1. 토픽/서비스/액션 - 서비스

## check\_order

```
2
3 import rclpy
4 from rclpy.node import Node
5 from std_msgs.msg import String
6 from turtle_interfaces.srv import MySrv # MySrv 서비스 유형 임포트
7
8 class CheckOrderClient(Node):
9     def __init__(self):
10         super().__init__('service_client')
11
12         # 서비스 클라이언트 설정
13         self.client = self.create_client(MySrv, 'check_order')
14
15         # table_order 데이터를 수신하기 위한 구독 설정
16         self.subscription = self.create_subscription( # table_order에서 주문 데이터 구독
17             String,
18             'table_order',
19             self.listener_callback,
20             10
21         )
22         self.order_data = None # 수신한 주문 데이터를 저장할 변수 # 데이터가 수신되면 서비스 요청
23
24         # Response publisher 설정
25         self.response_publisher = self.create_publisher(String, 'order_start', 10)
26
```

# 1. 토픽/서비스/액션 - 서비스

## check\_order

```
26
27 def listener_callback(self, msg):
28     # table_order 데이터를 수신했을 때 실행되는 콜백 함수
29     self.order_data = msg.data # 수신한 주문 데이터를 저장
30     self.get_logger().info(f'Received order data for confirmation: {self.order_data}')
31     self.check_order_match() # 수신된 데이터와 일치하는지 확인
32
33 def check_order_match(self):
34     # 서비스 요청 생성 및 실행
35     if self.client.wait_for_service(timeout_sec=1.0):
36         request = MySrv.Request()
37         request.x = 0.0 # x 좌표 값 예시 (필요한 값으로 수정 가능)
38         request.y = 0.0 # y 좌표 값 예시
39         request.table_number = int(self.extract_table_number(self.order_data)) # 테이블 번호 추출
40
41         future = self.client.call_async(request) # 비동기 요청
42         future.add_done_callback(self.handle_response) # 응답 처리 함수 연결
43     else:
44         self.get_logger().error('Service not available') # 서비스 사용 불가 시 에러 로그 출력
```

데이터가 수신되면 저장하여 로그 출력

서비스 요청 및 생성



# 1. 토픽/서비스/액션 - 서비스

## check\_order

```
45
46 def handle_response(self, future):
47     # 서비스 응답을 처리하는 함수
48     response_msg = String()
49     try:
50         response = future.result()
51         if response.success and response.message == self.order_data:
52             confirmation_message = '테이블과 주문이 일치합니다. 조리를 시작하세요!!!'
53             self.get_logger().info(confirmation_message)
54             response_msg.data = confirmation_message
55         else:
56             warning_message = '테이블과 주문이 일치하지 않습니다. 주문을 취소해주세요!!!'
57             self.get_logger().warning(warning_message)
58             response_msg.data = warning_message
59
60         # Publish the response message on 'order_start'
61         self.response_publisher.publish(response_msg)
62
63     except Exception as e:
64         error_message = f'Service call failed: {str(e)}'
65         self.get_logger().error(error_message)
66         response_msg.data = error_message
67         # Publish the error message on 'order_start'
68         self.response_publisher.publish(response_msg)
69
```

서비스 응답이 도착했을 때 호출

# 1. 토픽/서비스/액션 - 서비스

## send\_order

turtle\_table > turtle\_table > send\_order.py > main

```
1  #!/usr/bin/env python3
2
3  import rclpy
4  from rclpy.node import Node
5  from std_msgs.msg import String
6  from turtle_interfaces.srv import MySrv # 정의한 MySrv 서비스 유형 импорт
7
8  class SendOrderServer(Node):
9      def __init__(self):
10         super().__init__('service_server')
11
12         # 서비스 서버 설정
13         self.srv = self.create_service(MySrv, 'check_order', self.handle_check_order)
14
15         # 주문 데이터를 수신하기 위한 구독 설정
16         self.subscription = self.create_subscription(
17             String,
18             'table_order',
19             self.listener_callback,
20             10
21         )
22         self.order_data = None # 수신한 주문 데이터를 저장하기 위한 변수
23
```

table\_order에서 주문 데이터 구독

# 1. 토픽/서비스/액션 - 서비스

## send\_order

```
23
24 def listener_callback(self, msg):
25     # table_order 데이터를 수신했을 때 실행되는 콜백 함수
26     self.order_data = msg.data
27     self.get_logger().info(f'Received order data: {self.order_data}')
28
29 def handle_check_order(self, request, response): 데이터가 수신되면 로그 출력
30     # 클라이언트의 요청을 처리하여 응답 생성
31     if self.order_data:
32         response.success = True # 주문 데이터가 수신된 경우 성공
33         response.message = self.order_data # 현재 저장된 주문 데이터를 반환
34     else:
35         response.success = False # 주문 데이터가 없으면 실패
36         response.message = "아직 주문 데이터가 수신되지 않았습니다."
37     return response
38
```

서비스 요청이 들어왔을 때 응답 처리

## 2. 메시지 인터페이스 - srv

### MySrv

```
turtle_interfaces > srv > ≡ MySrv.srv
1  # 요청 필드 정의
2  float32 x
3  float32 y
4  int32 table_number
5  ---
6
7  # 응답 필드 정의
8  bool success
9  string message
10 bool is_valid
11
12
```

### 요청 필드

- 로봇이 특정 좌표에 있을 때 기준 테이블 번호가 정의
- 저장 & 요청된 테이블 번호 비교

### 응답 필드

- 요청 처리 성공 여부 확인
- 요청 처리 결과 반환
- 테이블 번호가 유효한지 판단

### 3. 테이블 오더 - 초기 화면

ROKEY F4 Cafe

커피

논커피

프라페

티

쉐이크/스무디

디저트

스낵

Your Order

선택한 메뉴 삭제

주문 상태: 주문 대기중

합계: 0원

주문 초기화

주문하기

### 3. 테이블 오더 - 메뉴 선택화면

ROKEY F4 Cafe

커피

논커피

프라페

티

쉐이크/스무디

디저트

스낵

아메리카노  
1,500원

꿀아메리카노  
2,000원

헤이즐넛 아메리카노  
2,000원

카페라떼  
2,000원

바닐라라떼  
2,000원

헤이즐넛라떼  
2,500원

연유라떼  
2,500원

카페모카  
2,500원

카라멜 마끼아또  
3,000원

아이스크림라떼  
3,500원

티라미수라떼  
4,000원

Your Order

선택한 메뉴 삭제

주문 상태: 주문 대기중

합계: 0원

주문 초기화

주문하기

### 3. 테이블 오더 - 세부옵션 선택화면

2,000원

Customize "바닐라라떼" X

핫 or 아이스 중 선택

핫

아이스

사이즈 선택

Tall (추가 금액 없음)

Grande (+500원)

Venti (+1000원)

에스프레소 샷 추가

샷 추가 없음

샷 추가 (+500원)

우유 선택

일반 우유

두유 (+500원)

아몬드 우유 (+500원)

취소

완료

선택한 메뉴 삭제

### 3. 테이블 오더 - 메뉴 선택 완료

ROKEY F4 Cafe

커피

논커피

프라페

티

셰이크/스무디

디저트

스낵

아메리카노 1,500원	콜아메리카노 2,000원	헤이즐넛 아메리카노 2,000원
카페라떼 2,000원	바닐라라떼 2,000원	헤이즐넛라떼 2,500원
연유라떼 2,500원	카페모카 2,500원	카라멜 마끼아또 3,000원
아이스크림라떼 3,500원	티라미수라떼 4,000원	

Your Order

바닐라라떼 (아이스 / 샷 추가 / 일반) - 2,500원

선택한 메뉴 삭제

주문 상태: 메뉴 선택중

합계: 2,500원

주문 초기화

주문하기



### 3. 테이블 오더 - 주문 확인 & 배달 방식 선택

1-6 배달

테이블 선택

테이블 번호를 선택해주세요

1	2	3
4	5	6

2,000원

카페모카  
주문 방식 선택

주문 방식을 선택해주세요

배달

픽업

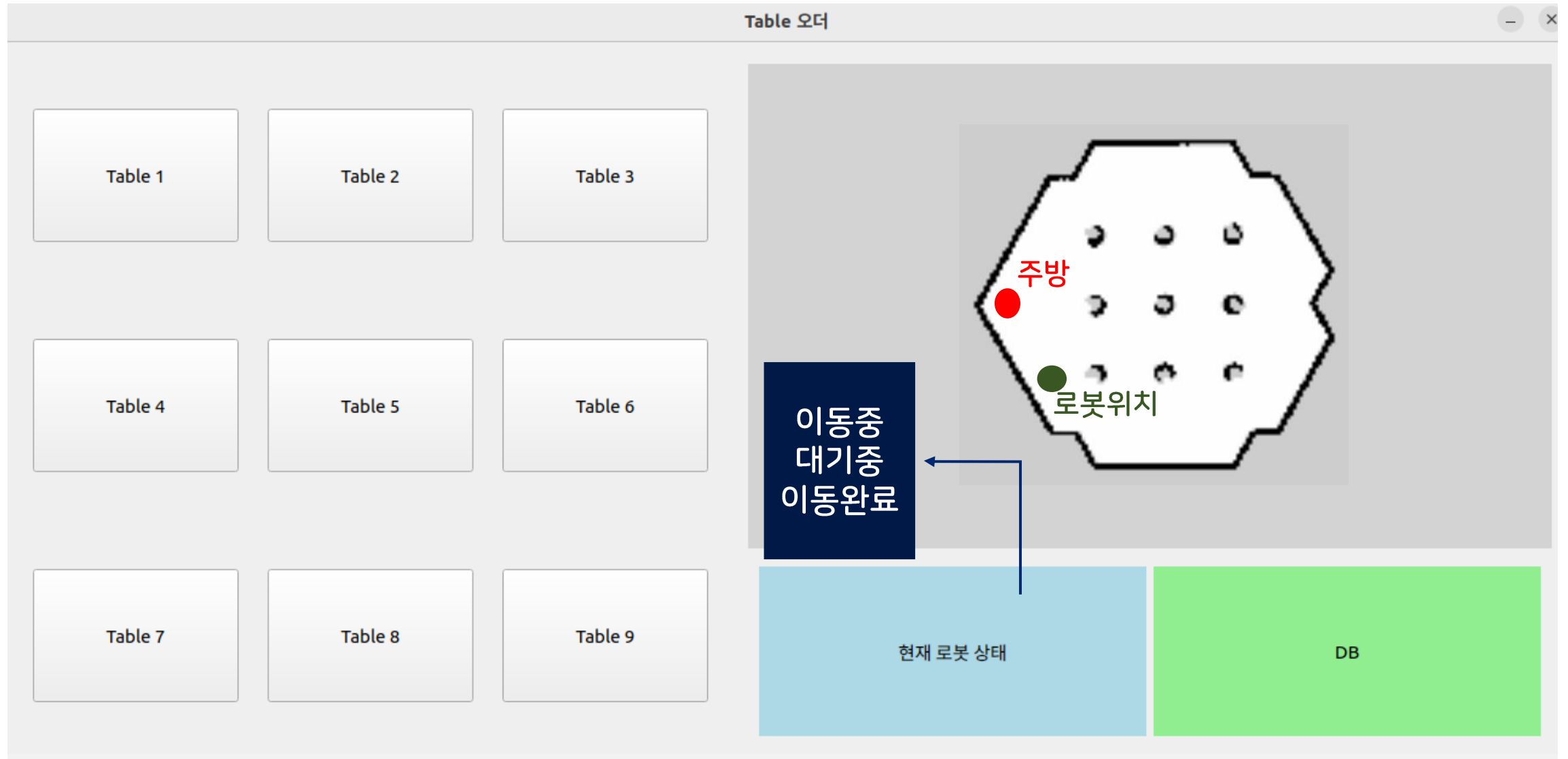
7-9 픽업

테이블 선택

테이블 번호를 선택해주세요

7 (1층 픽업대)	8 (2층 픽업대)	9 (3층 픽업대)
---------------	---------------	---------------

## 4. 주방 디스플레이 - 초기화면



## 4. 주방 디스플레이 - 주문 요청

주방 디스플레이

Table 1

Table 2

Table 3  
주문확인 - 합계: 2,500원  
바라떼 (아이스 / 샷 추가 / 일반) - 2,

Table 4

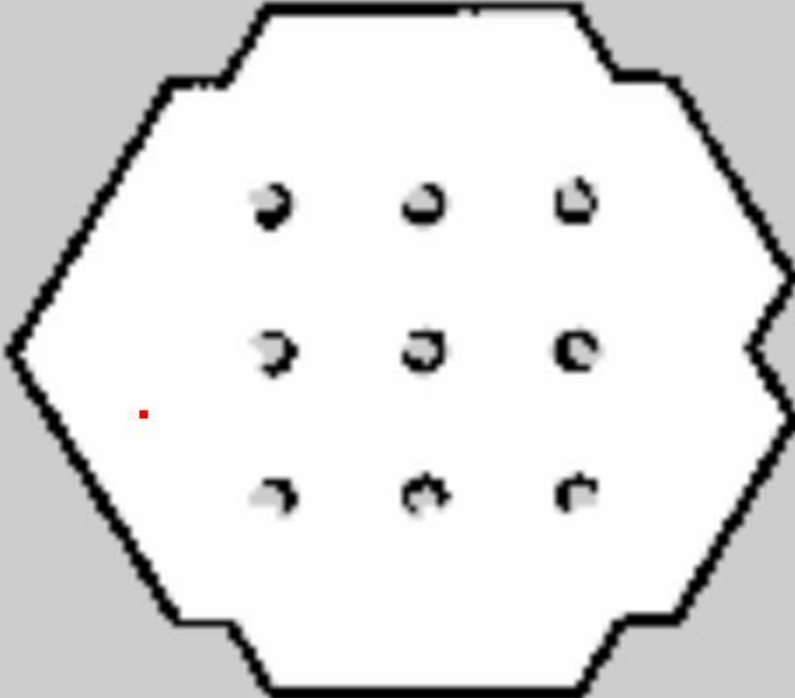
Table 5

Table 6

Table 7  
(1층 픽업)

Table 8  
(2층 픽업)

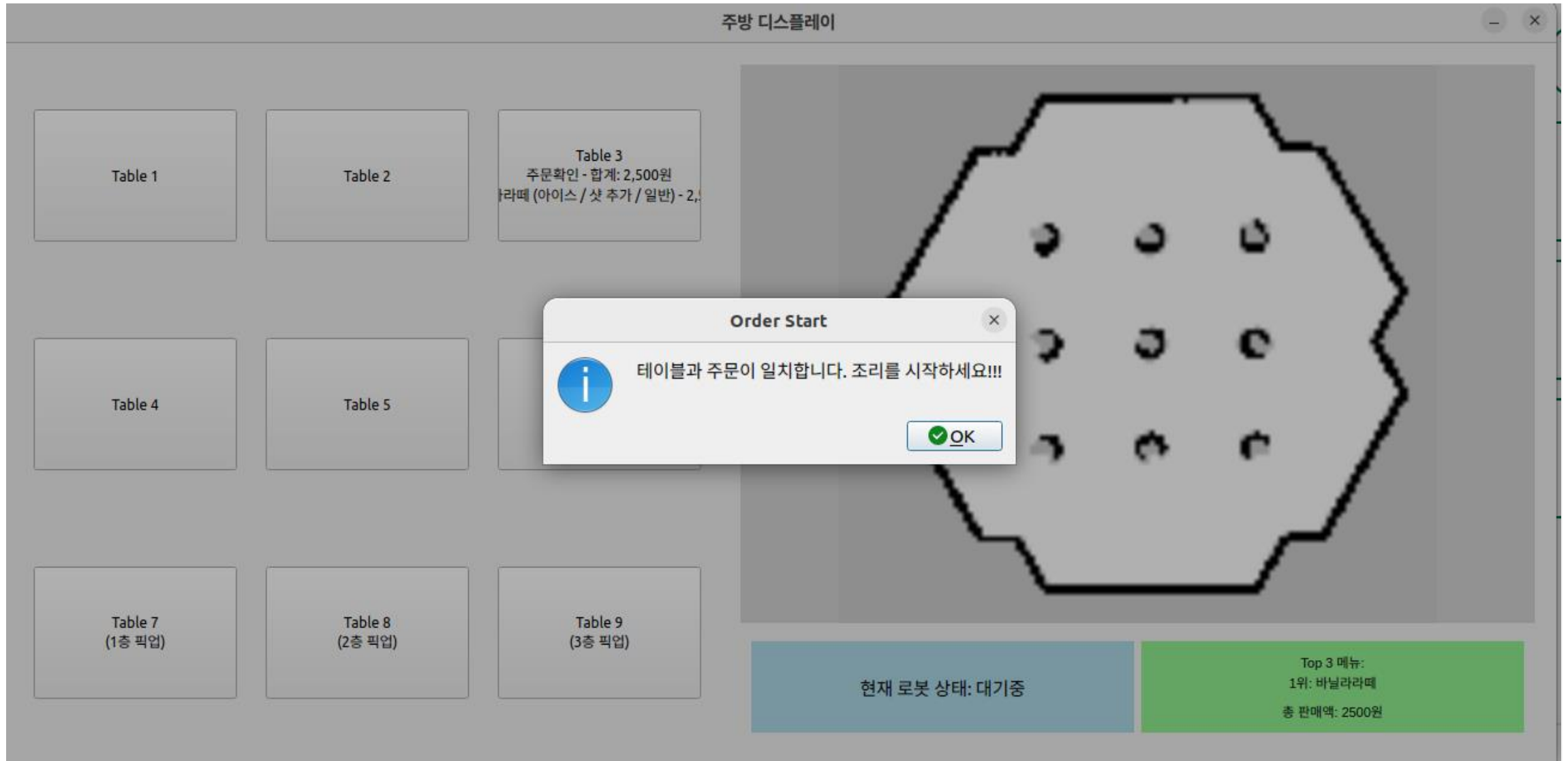
Table 9  
(3층 픽업)



현재 로봇 상태: 대기중

Top 3 메뉴:  
1위: 바닐라라떼  
총 판매액: 2500원

## 4. 주방 디스플레이 - 주문 요청



## 4. 주방 디스플레이 - 주문 취소



## 4. 주방 디스플레이 - 주문 취소

Table 3

주문확인 - 합계: 2,500원

라떼 (아이스 / 샷 추가 / 일반) - 2,500원

조리완료    주문취소

**품질**

시스템점검

가게사정

ROKEY F4 Cafe

커피    논커피    프라페    티    웨이크/스무디    디저트    스낵

초코 프라푸치노 4,000원    그린티 프라푸치노 4,000원    딸기 프라푸치노 4,000원

민트초코 프라푸치노 4,000원    자바칩 프라푸치노 4,000원

Your Order

**주문 취소**

9번 테이블 주문취소, 사유: 품질

OK

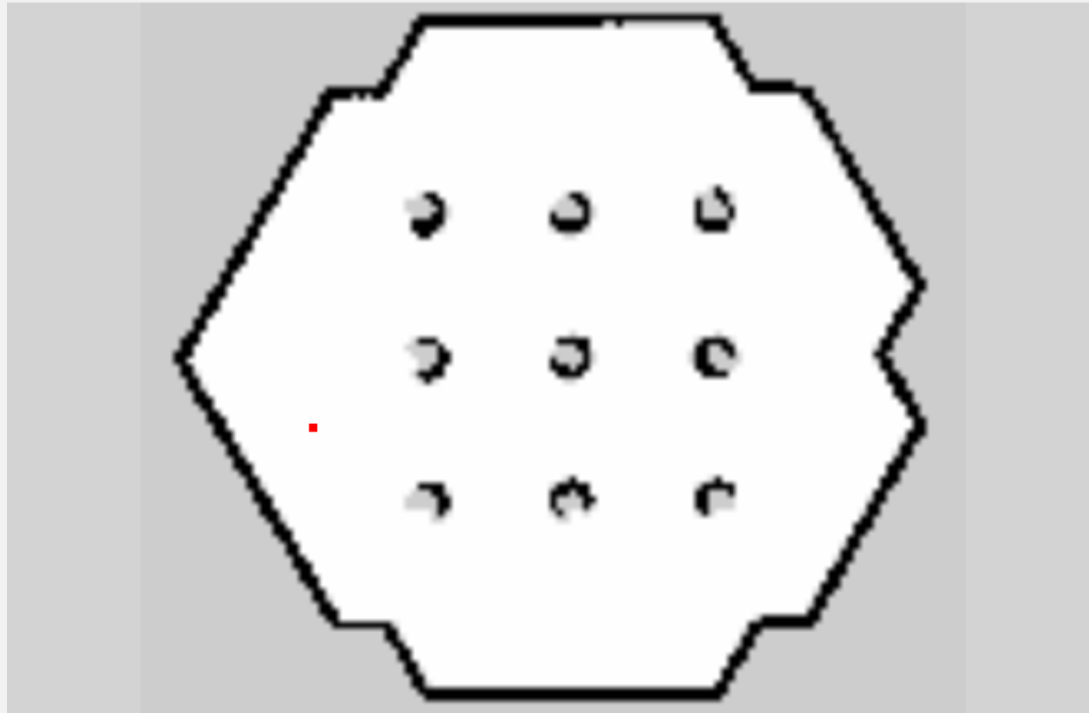
선택한 메뉴 삭제

주문 상태: 주문 완료

합계: 0원

주문 초기화    주문하기

## 5. 데이터 베이스



현재 로봇 상태: 대기중

Top 3 메뉴:  
1위: 카라멜 마끼아또  
2위: 바닐라라떼  
3위: 연유라떼  
총 판매액: 25500원

### Top3 메뉴 & 총 판매액 선정 이유

#### 1. 매출 증가

- 인기 메뉴 프로모션
- 수익성 높은 메뉴 집중 가능

#### 2. 운영 효율성 제고

- 인기 메뉴 재고 소진 방지
- 주문 대비 원재료를 준비하여 비용 절감

#### 3. 고객 만족도 향상

- 인기 메뉴 추천
- 수요 예측을 통한 대기 시간 감소

## 5. 데이터 베이스

### sub\_data

```
13 class TableOrderSubscriber(Node):
14     def __init__(self):
15         self.total_revenue_publisher = self.create_publisher(Int32, 'total_revenue', 10)
16
17         # 데이터베이스 파일 경로 설정
18         db_path = 'orders.db'
19
20         # 기존 데이터베이스 파일이 있다면 삭제
21         if os.path.exists(db_path):
22             os.remove(db_path)
23             self.get_logger().info('Deleted existing database file.')
24
25         # 데이터베이스 초기화
26         self.conn = sqlite3.connect(db_path)
27         self.cursor = self.conn.cursor()
28
29         # 테이블 생성 (options 컬럼 추가)
30         self.cursor.execute('''CREATE TABLE orders
31                                (id INTEGER PRIMARY KEY AUTOINCREMENT,
32                                 menu_name TEXT,
33                                 price INTEGER,
34                                 options TEXT, -- 옵션을 저장할 컬럼 추가
35                                 table_num INTEGER)''')
36
37         self.conn.commit()
38         self.get_logger().info('Created new orders table.')
```



## 5. 데이터 베이스

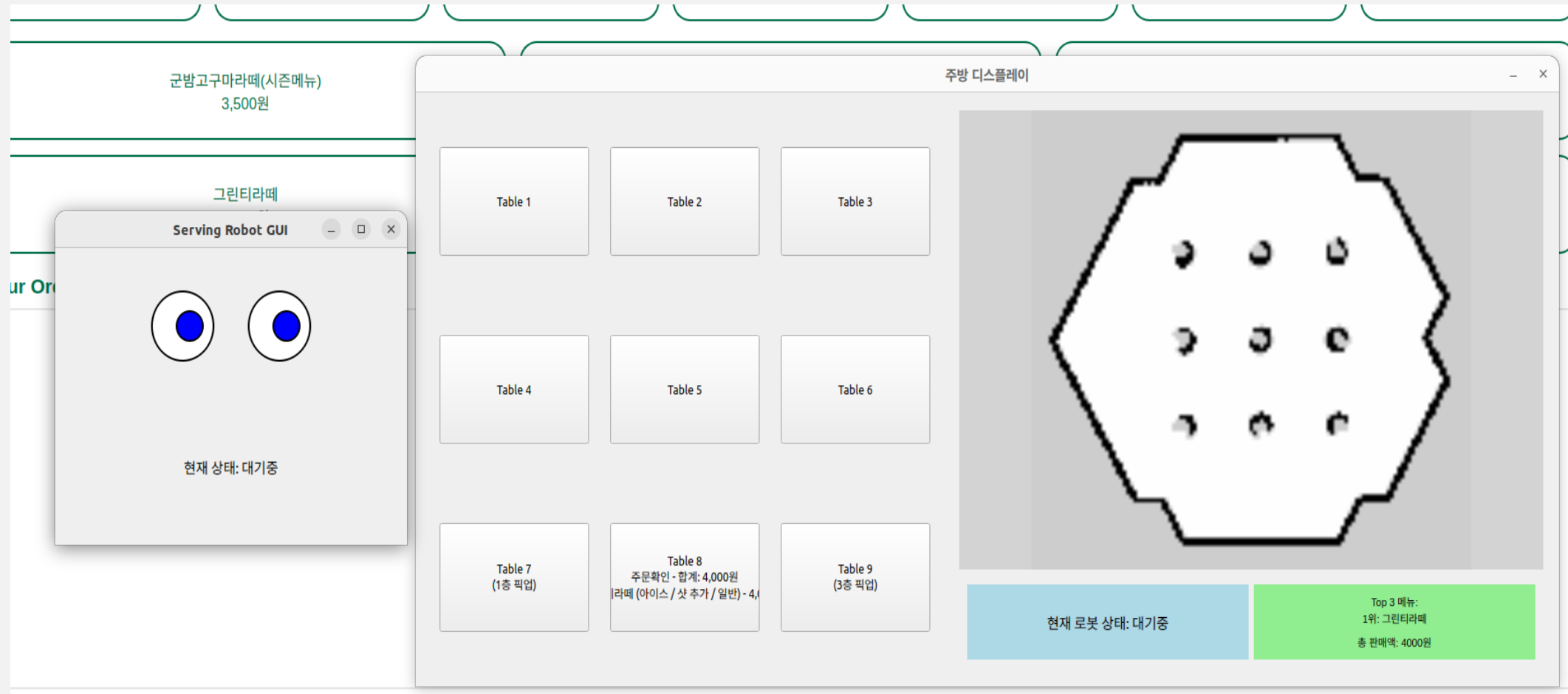
sub\_data

```
78
79     if menu_items and table_num:
80         try:
81             # 각 메뉴 항목을 개별적으로 저장하고 통계 업데이트
82             for menu_name, price in menu_items:
83                 # 데이터베이스에 저장
84                 self.cursor.execute("INSERT INTO orders (menu_name, price, table_num) VALUES (?, ?, ?)
85                                     (menu_name, price, table_num)")
86
87             # 메뉴별 판매 수 및 금액 추적
88             self.menu_sales[menu_name] += 1
89             self.menu_revenue[menu_name] += price
90
91             self.conn.commit()
92             self.get_logger().info(f"Saved to database: {len(menu_items)} items, total {total_amount}")
93         except sqlite3.Error as e:
94             self.get_logger().error(f"Database error: {e}")
95
```

## 5. 데이터 베이스

id	menu_name	price	options	table_num
1	민트초코 프라푸치노	5000	아이스 / Venti	2
2	자바칩 프라푸치노	4500	아이스 / Grande	2
3	자바칩 프라푸치노	5000	아이스 / Venti	9
4	딸기 프라푸치노	4500	아이스 / Grande	9
5	초코케이크	6000	포크0	9
6	군고구마	3500	시즌메뉴	6
7	민트초코 프라푸치노	4500	아이스 / Grande	6
8	연유라떼	3000	핫 / 샷 추가 / 일반	6
9	아메리카노	2500	핫 / Grande / 샷 추가	6

## 6. 서빙 로봇 - 대기 중



## 6. 서빙 로봇 - 이동 중

The screenshot displays the 'Serving Robot GUI' interface. At the top, there are tabs for '커피' (Coffee), '논커피' (Non-coffee), '프라페' (Frappé), '티' (Tea), '쉐이크/스무디' (Shake/Smoothie), '디저트' (Dessert), and '스낵' (Snack). The main area is titled '주방 디스플레이' (Kitchen Display) and shows a grid of tables. A red box highlights 'Table 1' and 'Table 2'. 'Table 1' shows '주문확인 - 합계: 4,500원' and '수라떼 (아이스 / 샷 추가 / 일반) - 4'. 'Table 2' shows '주문확인 - 합계: 3,500원' and '마끼아또 (아이스 / 샷 추가 / 일반) - 1'. Below these are buttons for '조리완료' (Cooking Complete) and '주문취소' (Cancel Order). A dialog box titled '조리 완료' (Cooking Complete) is open, displaying an information icon and the text '서빙로봇이 출발합니다: Table 2' (The serving robot is departing: Table 2), with an 'OK' button. The bottom right corner shows the '현재 로봇 상태: 이동중' (Current robot status: Moving) and a 'Top 3 메뉴' (Top 3 Menu) section listing: 1위: 카라멜 마끼아또, 2위: 바닐라라떼, 3위: 연유라떼, and '총 판매액: 25500원' (Total sales: 25,500 won).

아메리카노  
1,500원

Serving Robot GUI

현재 상태: 이동중

Order

주방 디스플레이

Table 1  
주문확인 - 합계: 4,500원  
수라떼 (아이스 / 샷 추가 / 일반) - 4

Table 2  
주문확인 - 합계: 3,500원  
마끼아또 (아이스 / 샷 추가 / 일반) - 1

조리완료 주문취소

Table 3

Table 4

Table 5

Table 6

Table 7 (1층 픽업)

Table 8 (2층 픽업)

Table 9 (3층 픽업)

조리 완료

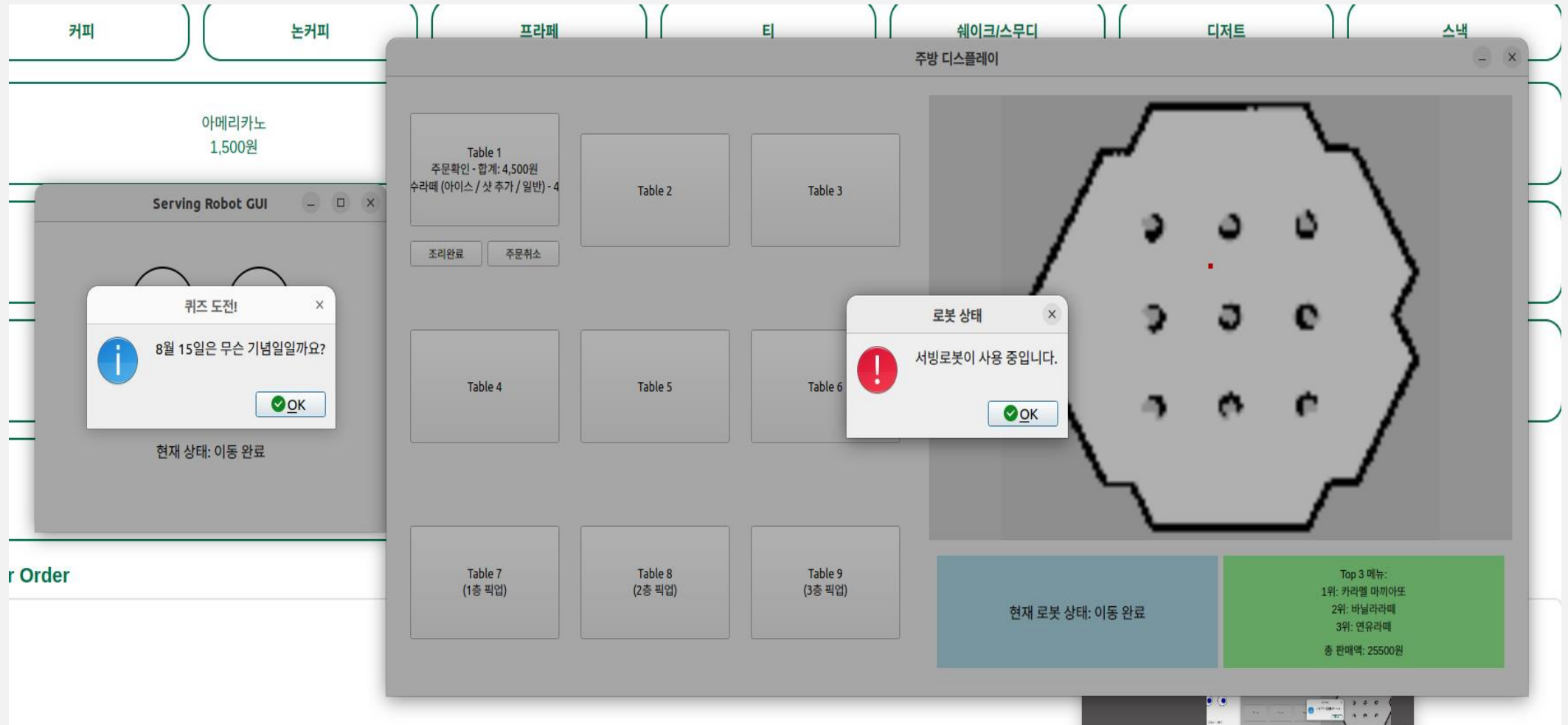
서빙로봇이 출발합니다: Table 2

OK

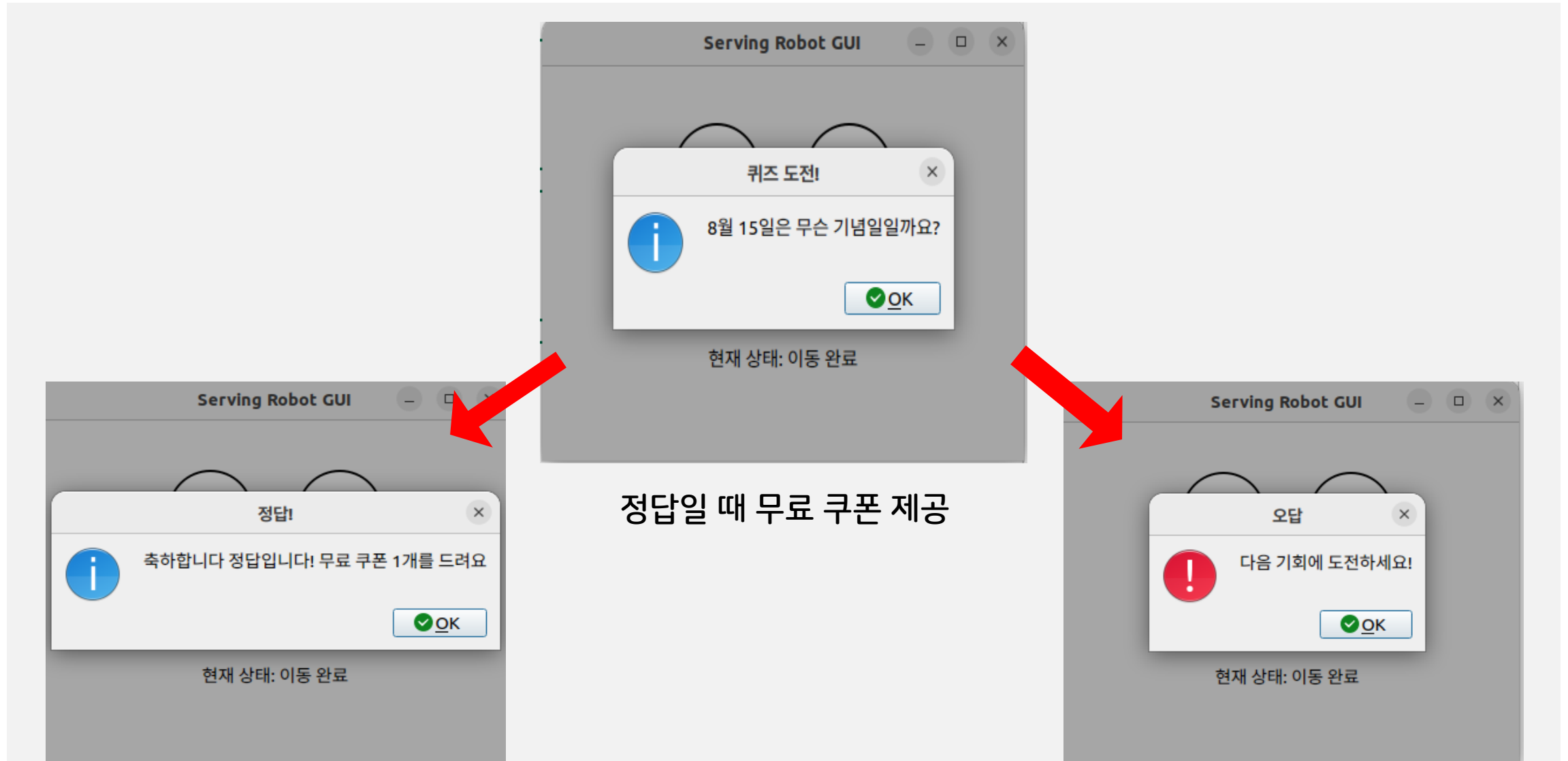
현재 로봇 상태: 이동중

Top 3 메뉴:  
1위: 카라멜 마끼아또  
2위: 바닐라라떼  
3위: 연유라떼  
총 판매액: 25500원

## 6. 서빙 로봇 - 이동 중



## 6. 서빙 로봇 - 이동 완료



## 6. 서버 로봇 - 이동 완료

```
223 223
224 224
225 225
226 226
227 227
228 228
229 229
230 230
231 231
232 232
233 233
234 234
235 235
236 236
237 237
238 238
239 239
240 240
241 241
242 242
243 243
244 244
245 245
246 246
247 247
248 248
249 249
250 250
251 251
252 252
253 253
254 254
255 255
256 256
257 257
258 258

def show_general_message(self):
    """일반 메시지 표시"""
    general_messages = [
        "감사합니다 다음에도 주문해주세요^_^!",
        "조금만 더 힘내요! 제가 늘 곁에 있어요!",
        "날씨가 쌀쌀하네요, 감기 조심하세요!",
        "힘든 하루였나요? 저의 배달로 위로가 되었으면 좋겠어요!",
        "천천히 여유 있게 드세요, 그게 제일 맛있답니다!",
        "하루가 조금 바빴다면 잠깐 쉬어가도 좋아요!",
        "오늘 하루도 대단했어요! 좀 더 힘내봐요!",
        "제가 그리웠나요? 맛있게 즐기고 에너지 충전하세요!",
        "주문 감사해요, 또 불러주시면 언제든지 달려올게요!",
        "귀한 분께 배달 드릴 수 있어 영광입니다 ^_^!",
    ]

    message = random.choice(general_messages)
    QMessageBox.information(self, "배달 완료", message)

def show_quiz_message(self):
    """퀴즈 메시지 표시"""
    quizzes = [
        {"question": "퀴즈: 로봇의 눈 색깔은(oo색)?", "answer": "파란색"},
        {"question": "퀴즈: 몽골의 수도는?", "answer": "울란바토르"},
        {"question": "8월 15일은 무슨 기념일일까요?", "answer": "광복절"},
    ]

    quiz = random.choice(quizzes)
    self.current_quiz_answer = quiz["answer"]
    QMessageBox.information(self, "퀴즈 도전!", quiz["question"])

    answer, ok = QInputDialog.getText(self, "퀴즈 답변", "정답을 입력하세요:")
    if ok:
        if answer == self.current_quiz_answer:
            QMessageBox.information(self, "정답!", "축하합니다 정답입니다! 무료 쿠폰 1개를 드려요")
        else:
            QMessageBox.warning(self, "오답", "다음 기회에 도전하세요!")
```

# 7. Logging

---

## 1. Info(기본적인 로봇 정보)

```
self.get_logger().info('테이블과 주문이 일치합니다. 조리를 시작하세요!!!')  
self.get_logger().info('주문완료')  
self.get_logger().info(f"로봇 현재 상태: {self.status}")
```

## 2. warning(조작 미숙 등의 경미한 오류)

```
self.get_logger().warning('테이블과 주문이 일치하지 않습니다. 주문을 취소해주세요!!!')  
self.get_logger().warning('서빙로봇이 사용 중입니다.')
```

## 3. error(시스템의 치명적인 오류)

```
self.get_logger().error(f"잘못된 테이블 번호 형식: {line}")  
self.get_logger().error(f"건너뛰는 라인: {line} (유효한 테이블 번호나 항목 없음)")  
self.get_logger().error(f"테이블 번호가 지정되지 않았습니다: {line}")  
self.get_logger().error("NAV2 액션 서버를 찾을 수 없습니다.")
```



## 8. QoS

---

```
qos_profile = QoSProfile(  
    reliability=QoSReliabilityPolicy.RELIABLE,  
    history=QoSHistoryPolicy.KEEP_LAST,  
    durability=QoSDurabilityPolicy.TRANSIENT_LOCAL,  
    depth=10  
)
```

- RELIABLE : 정보 손실 위험을 낮춰서 모든 메시지 반드시 전달
- KEEP\_LAST : 받아온 최신 정보 이용
- TRANSIENT\_LOCAL : 작동 중 오류로 꺼질 가능성을 고려해 최신 정보를 받아 임무 완료
- depth = 10 : 메시지를 처리하지 못할 상황을 대비

감사합니다