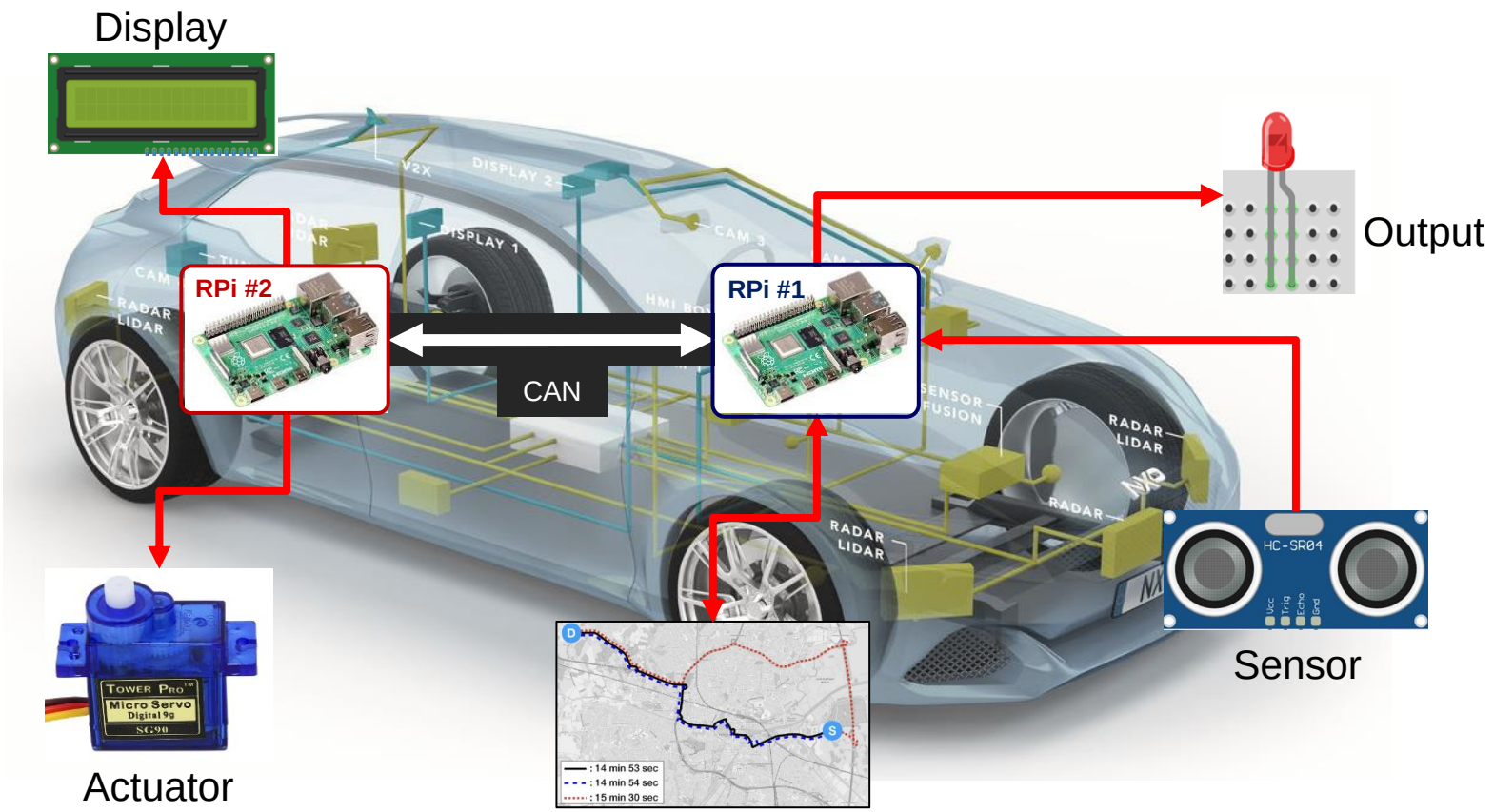
# [프로젝트]
# Distributed Control System over CAN

**Minsoo Ryu**

**Operating Systems and Distributed Computing Lab.**
**Hanyang University**

**msryu@hanyang.ac.kr**

# Distributed Control with Raspberry ECUs



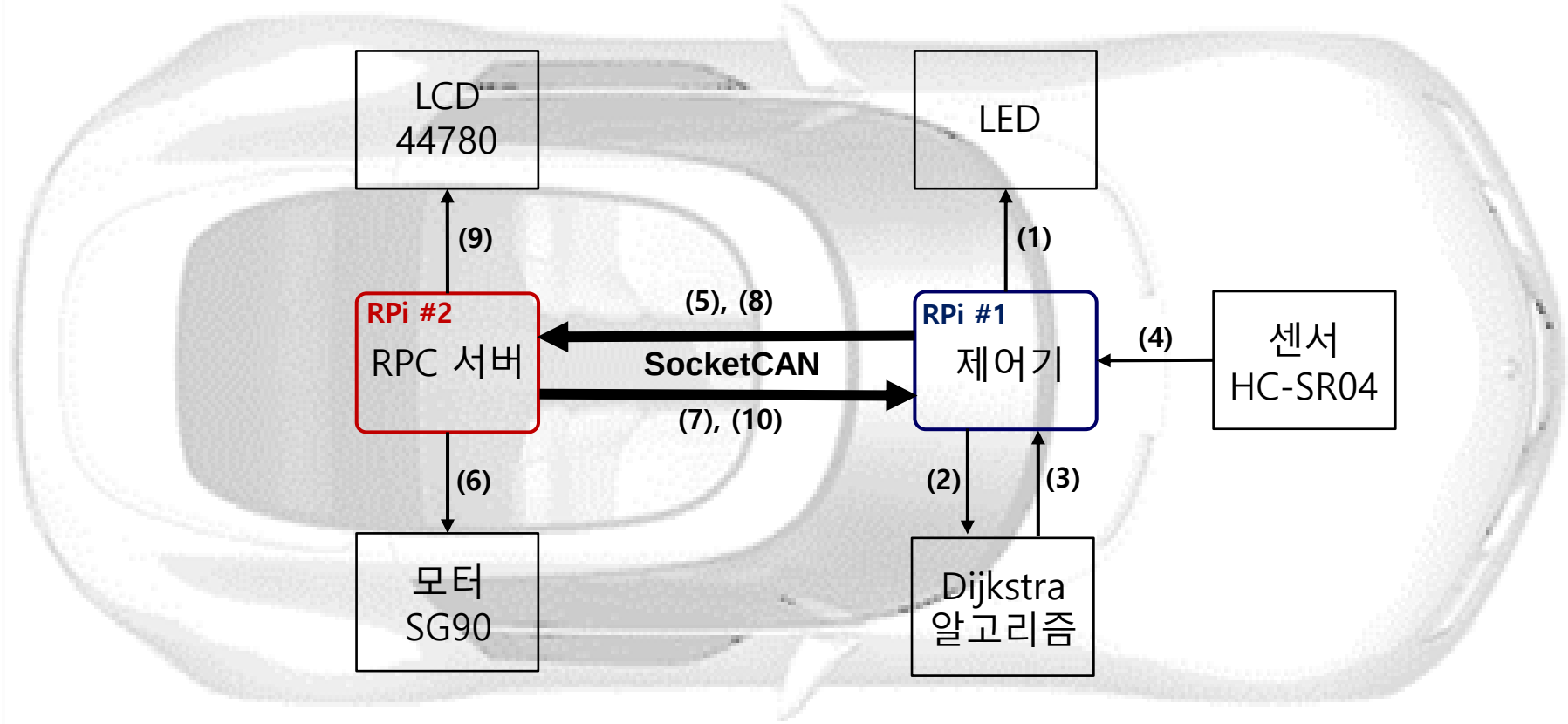
# 프로젝트의 목적

- 다양한 하드웨어를 통합적으로 제어한다
  - **LED, LCD**와 같은 입출력 장치의 **I/O** 프로그래밍
  - 초음파 센서 및 서보 모터의 제어 프로그래밍

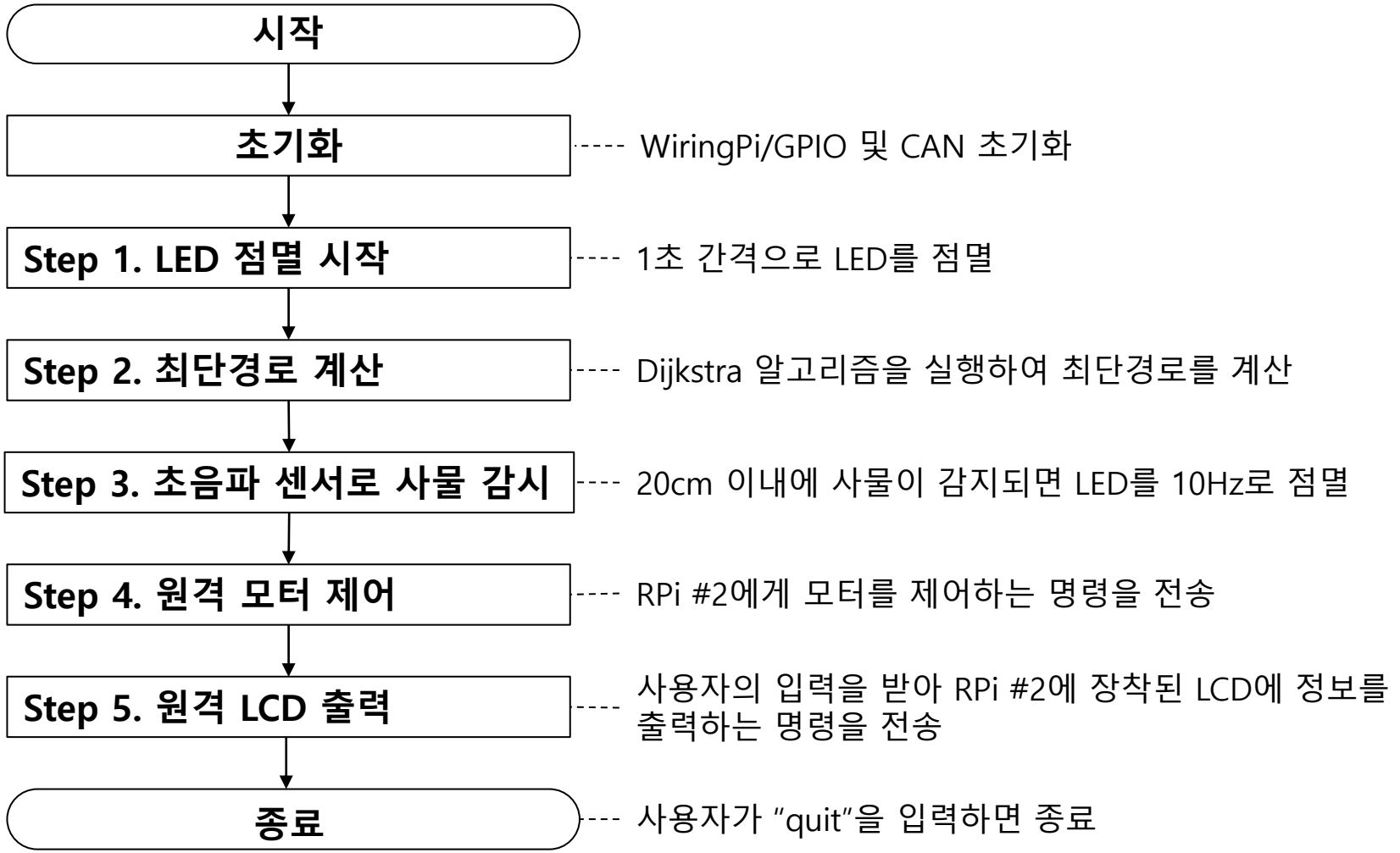
- **CAN** 통신을 이용하여 하드웨어를 제어한다
  - **SocketCAN**을 이용한 통신 프로그래밍
  - 원격함수호출**(RPC)** 개념의 이해와 구현

- 복잡한 형태의 프로그램을 설계하고 구현한다
  - 복수의 소스 파일 및 헤더 파일 사용
  - **Client/Server** 구조의 프로그램 구현
  - **Concurrent execution**을 위한 **multithreaded** 프로그래밍

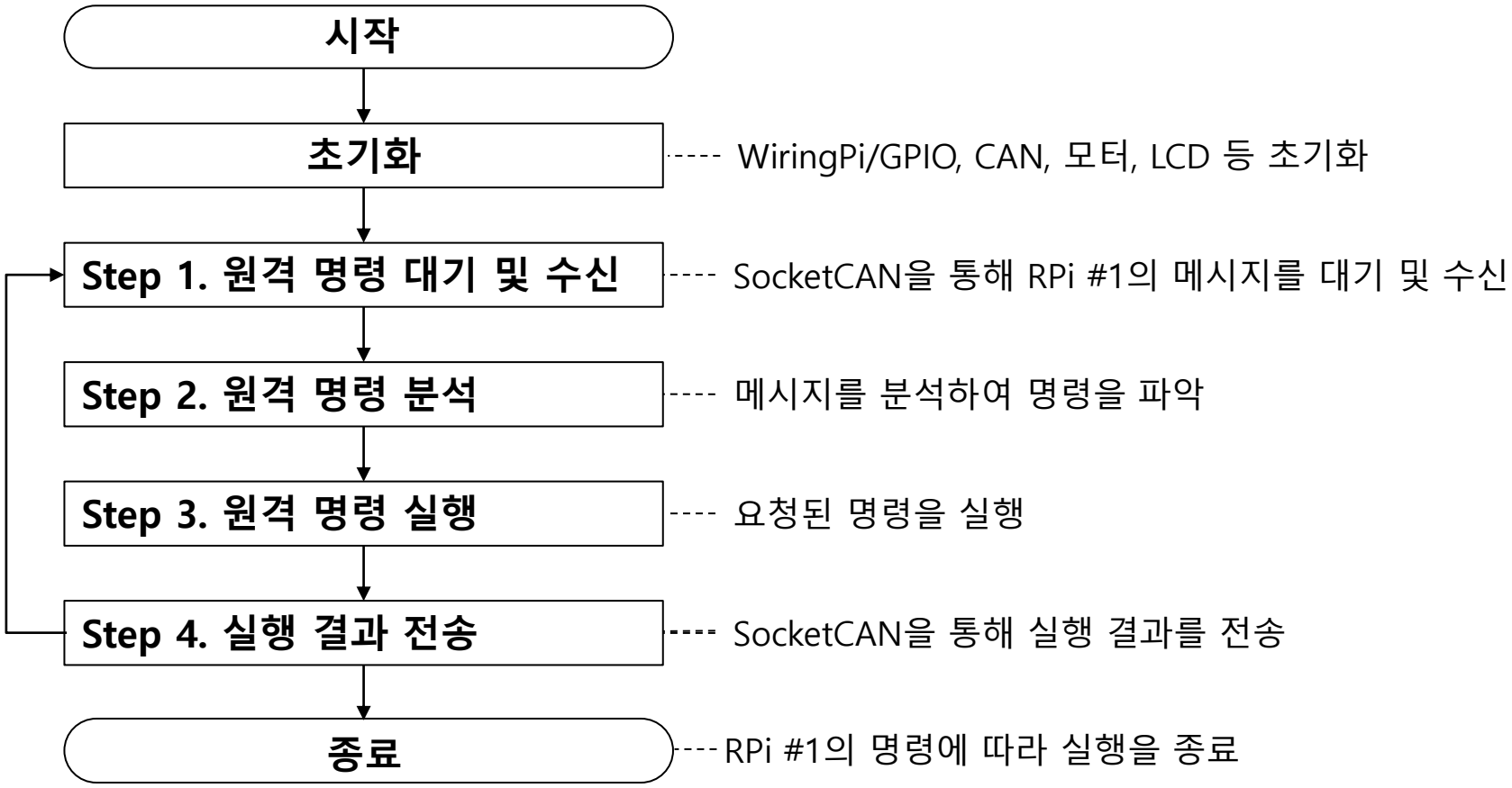
# 블록 다이어그램


LCD
44780
LED
(9)
(1)
RPi #2
RPC 서버
(5), (8)
SocketCAN
(7), (10)
RPi #1
제어기
(4)
센서
HC-SR04
(6)
(2)
(3)
모터
SG90
Dijkstra
알고리즘

# RPi #1 동작 흐름 “rpi_1_main.c”



# RPi #2 동작 흐름 “rpi_2_main.c”



# 요구사항 상세 (1/3)

- **RPC 기반 원격 제어 구조**
  - **R1. SocketCAN을 사용한다**
  - **R2. 원격 제어를 위해 Rpi #2에서는 아래의 함수를 구현한다**
    - int moveMotor(int inputValue)
      - ✓ 인자값 inputValue만큼 모터의 각도를 회전시킨다
    - int displayText(int lineNum, char *text)
      - ✓ 인자값 lineNum이 가리키는 LCD의 line에 인자값 text로 받은 문자열을 LCD에 출력한다
    - int terminateRPC(char *text)
      - ✓ 인자값 text로 “quit”을 받으면 Rpi #2의 실행을 종료한다 (main 함수 종료)
  - **R3. RPi #1의 main() 함수에서는 RPi #2에서 구현된 함수들과 동일한 프로토타입을 가지는 함수들을 호출한다**
    - int moveMotor(int inputValue)
    - int displayText(int lineNum, char *text)
    - int terminateRPC(char *text)

# 요구사항 상세 (2/3)

## ❒ RPC 기반 원격 제어 실행

- **R4.** 원격 제어를 통해 모터를 **180°** 회전시키고 다시 **0°** 위치로 회전시킨 후, 성공하면 **0**을 **RPi #1**에 리턴하고 실패하면 **-1**을 리턴한다
- **R5. RPi #1**에서 사용자의 입력을 받아 **RPi #2**의 **LCD**에 입력받은 문자열을 출력하고, 성공하면 출력된 문자열의 크기를 **RPi #1**에 리턴하고 실패하면 **-1**을 리턴한다
- **R6. RPi #1**에서 사용자가 **"quit"**을 입력하면 **RPi #2**의 **terminateRPC** 함수 호출을 요청하고, **RPi #2**는 이를 수신하면 **RPi #1**에 **0**을 리턴하고 종료하며, **RPi #1**은 리턴값 **0**을 확인하고 종료한다
- **R7.** 원격 제어를 위해 송수신하는 데이터를 변환하지 않는다
  - 예를 들면, int 데이터를 문자열로 변환하여 송신하지 않는다

## ❒ LED 및 초음파 센서 제어

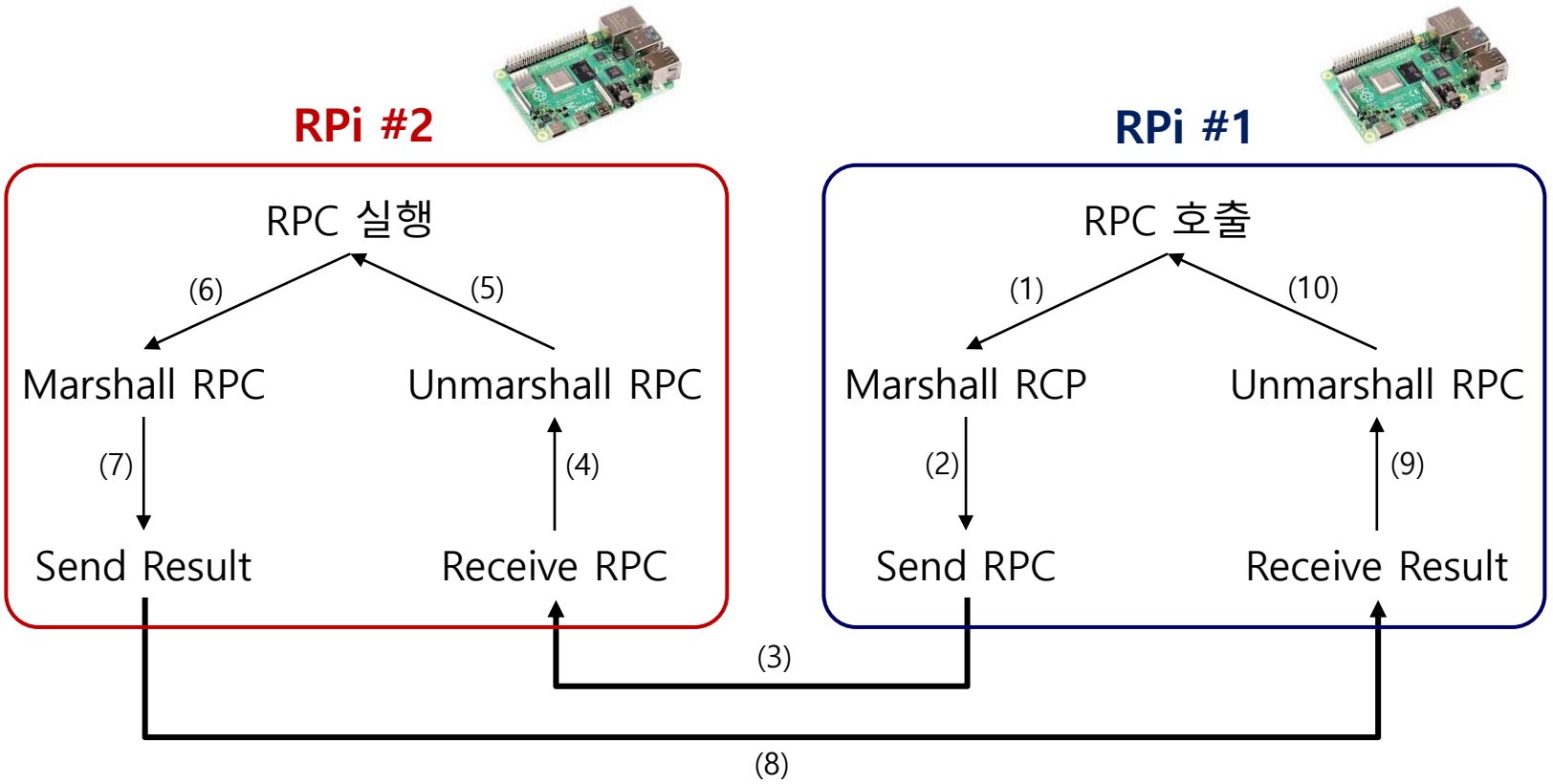
- **R8. RPi #1**에 장착된 **LED**를 **1**초 간격으로 점멸을 지속한다
- **R9. RPi #1**에 장착된 초음파 센서로 사물을 감시하며, **20cm** 이내의 사물이 감지되면 **LED**를 **1**초 동안 **100ms** 단위로 점멸한다

# 요구사항 상세 (3/3)

## ❒ 최단 경로 계산

- **R10. RPi #1에 아래의 함수를 구현하여 실행한다**
  - void findShortestPath(int source, int destination, char buffer[], int *len)
  - 인자값 source로부터 destination까지의 최단경로를 계산하여 문자열 형태로 text에 반환하며, len에는 문자열의 길이를 반환한다
  - 최단 경로는 제공될 “map_data.txt” 데이터로부터 계산한다
  - 제공된 데이터에서 노드 3에서 노드 6까지의 최단 경로를 찾도록 한다
  - 출발 노드 S와 도착 노드 D가 주어지면 최단 경로를 “S -> A -> B -> D” 형식으로 반환한다

# RPC 기반 원격 제어 흐름



# 프로그램 배치와 구성

**RPi #2**

"rpi_2_main.c"

"rpi_2_motor.c" "rpi_2_lcd.c"

"rpi_2_can.c" "rpi_2_stub.c"

**RPi #1**

"rpi_1_main.c"

"rpi_1_led.c" "rpi_1_dijkstra.c"

"rpi_1_ultrasonic.c"

"rpi_1_stub.c" "rpi_1_can.c"

**RPC over CAN**

# 터미널 상의 실행 예시

**RPi #1**

```
hwkim@hwkim:~/Lab-RPi/80_Project_RPC_CAN/1_RPi $ sudo ./executable_rpi_1

RPi #1 is ready.

Started LED.

The shortest path is 3 -> 4 -> 6.

Started Ultrasonic sensor.

Requested RPC moveMotor() and received return value 120
Requested RPC displayText() and received return value 12
Enter your text to display on RPi #2's LCD: Hello world
Enter your text to display on RPi #2's LCD: Good bye
Enter your text to display on RPi #2's LCD: quit

Terminating RPi #1.
```

**RPi #2**

```
msryu@rpi:~/Lab-RPi/80_Project_RPC_CAN/2_RPi $ sudo ./executable_rpi_2

RPi #2 is ready to accept RPC requests.

RPC request 'moveMotor(120)' received and processed.

RPC request 'displayText(1, 3 -> 4 -> 6) received and processed.

RPC request 'displayText(2, You made it!) received and processed.

RPC request 'displayText(1, Hello world) received and processed.

RPC request 'displayText(1, Good bye) received and processed.

RPC request 'QUIT' command received.

Terminating RPi #2.
```

# 완성도 평가

- 평가 방법: 요구 기능의 작동 여부에 따라 평가
  - **A+:** 요구사항 **R1 ~ R10** 모두 충족
  - **A:** 요구사항 **9**개 충족
  - **B+:** 요구사항 **8**개 충족
  - **B:** 요구사항 **7**개 충족
  - **C+:** 요구사항 **6**개 충족
  - **C:** 요구사항 **5**개 충족
  - **D:** 요구사항 **4**개 이하 충족

- **RPC** 구현의 완성도가 높을 경우 가점 부여

# 프로젝트 진행 일정

| 일자 | 진행 계획 |
|---|---|
| 1일차 | • ~ 2:30 (유민수): 프로젝트 소개 및 Dijkstra 알고리즘 강의<br>• 2:30 ~ 4:00 (팀별 작업): 설계 및 구현 계획 수립, 발표자료 작성 (발표자료는 공유폴더로 제출, 마감 3시 40분)<br>- Call and Return Architecture(함수 그래프), Flow Chart (순서도), Task Interaction Diagram 등을 포함<br>- 팀원별 역할 배정 및 구현 계획 (0.5일 단위로 개발 계획을 수립)<br>• 4:00 ~ 4:40 (팀별 발표): 설계 및 구현 계획 발표, 피드백 및 토의 |
| 2일차 | • 8:00 ~ 8:30 (유민수): 소프트웨어 구현 관련 강의<br>• 8:30 ~ 11:40 (팀별 작업): 구현 진행<br>• 1:00 ~ 2:00 (유민수): 팀별 진행상황 점검<br>• 2:00 ~ 4:40 (팀별 작업) 구현 진행<br>- 개별/팀별 질의응답 |
| 10일차 | • 8:00 ~ 11:40 (팀별 작업): 구현 진행<br>- 개별/팀별 질의응답<br>• 1:00 ~ 3:00 (팀별 작업): 구현 마무리 및 발표자료 작성 (발표자료는 공유폴더로 제출, 마감 3시)<br>• 3:00 ~ 4:00 (팀별 발표): 프로젝트 결과 발표<br>• 4:00 ~ 4:40 (팀별 시연): 데모 및 최종 평가 |


thank you!

# 원격함수호출(RPC) 개념도