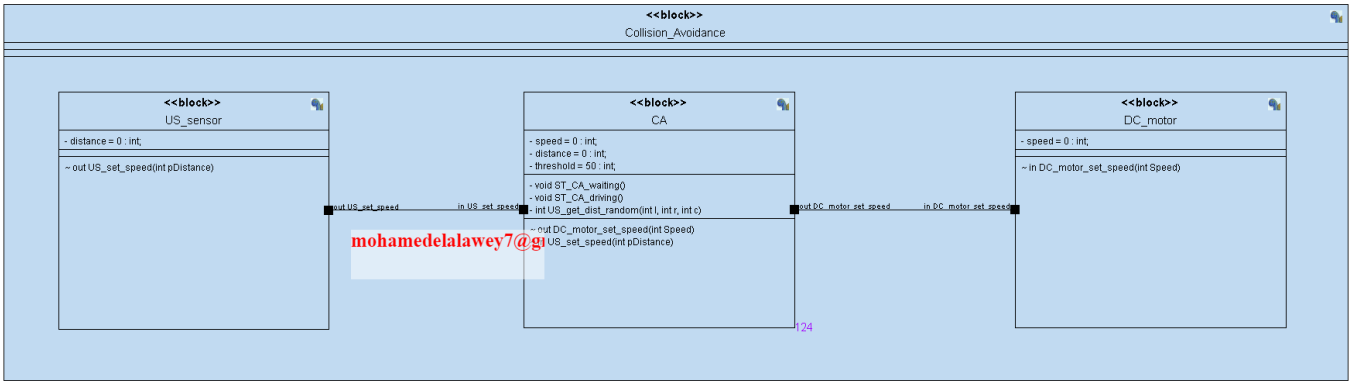
Collision Avoidance Project Report

1. Introduction

In this project, I have implemented a basic collision avoidance system using a US (Ultrasonic) sensor, a central controller block (CA), and a DC motor. The system mimics a simplified embedded software behavior that reads distance from the US sensor, processes the data through a central logic (CA), and adjusts the motor speed accordingly to avoid collisions. The model is based on SysML using block definition and sequence diagrams.

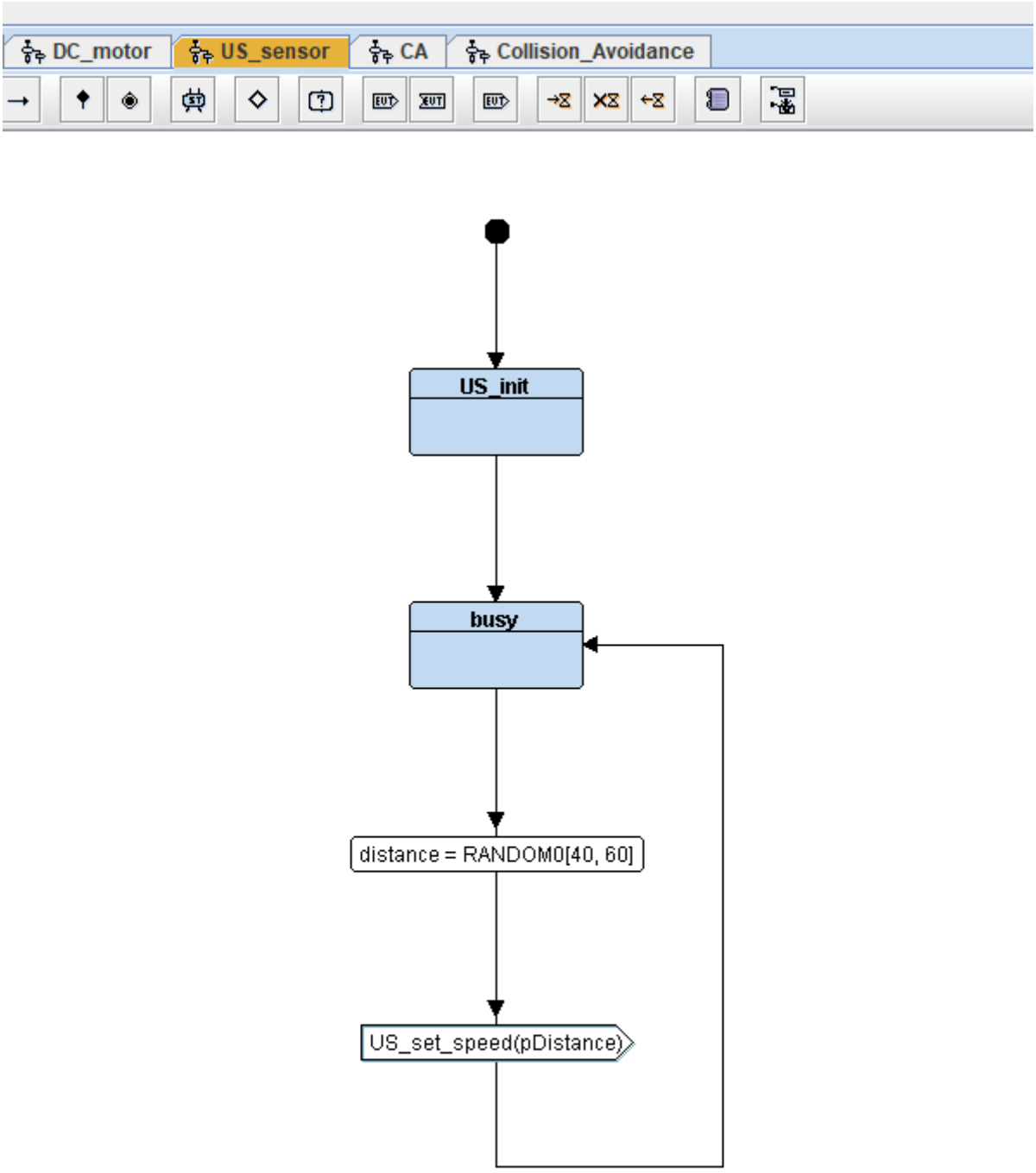
2. System Overview



2.1 Main Components

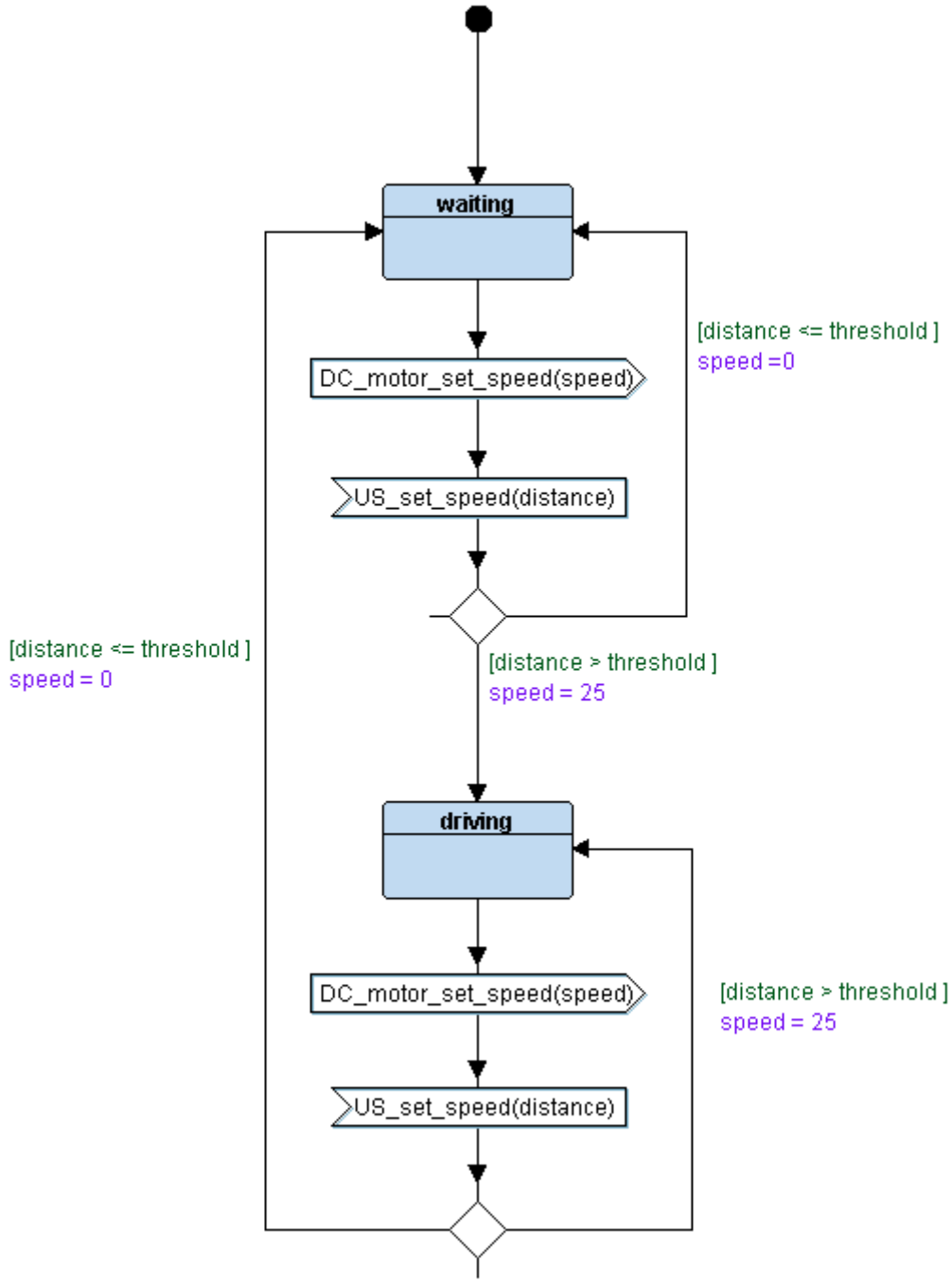
- **US_sensor Block**

  This block simulates an ultrasonic sensor responsible for measuring the distance to the nearest object. It outputs this distance to the CA block.

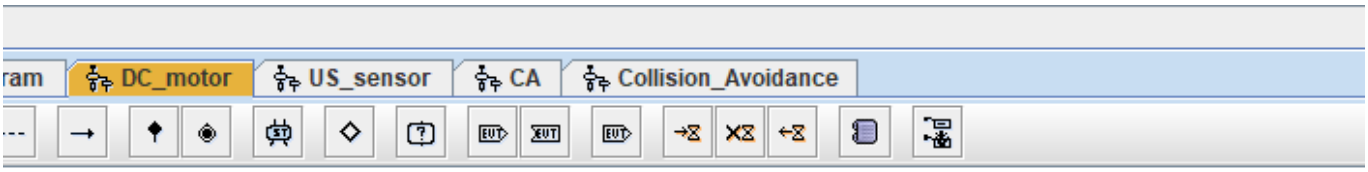


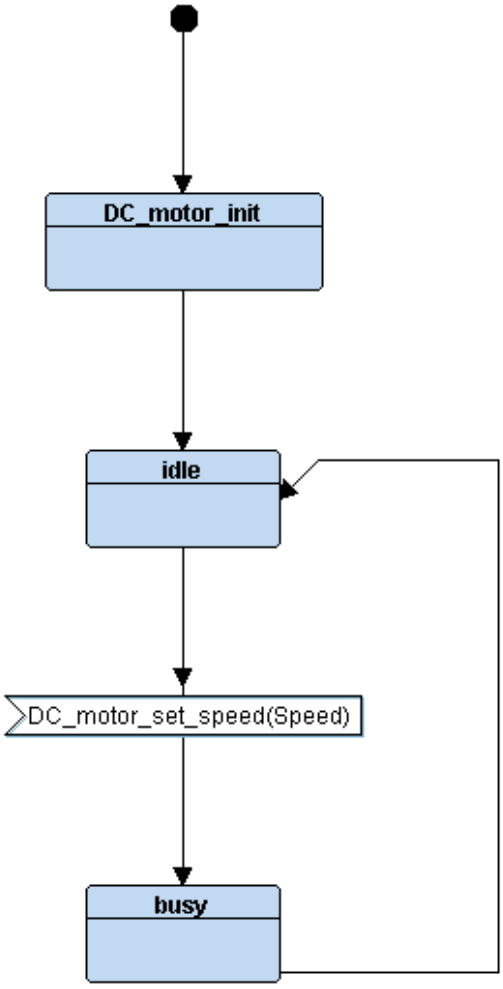
- **CA Block (Collision Avoidance Controller)**

  Acts as the decision-making unit of the system. It receives distance data from the sensor and calculates the appropriate motor speed. It applies a simple rule-based logic using a predefined threshold value.



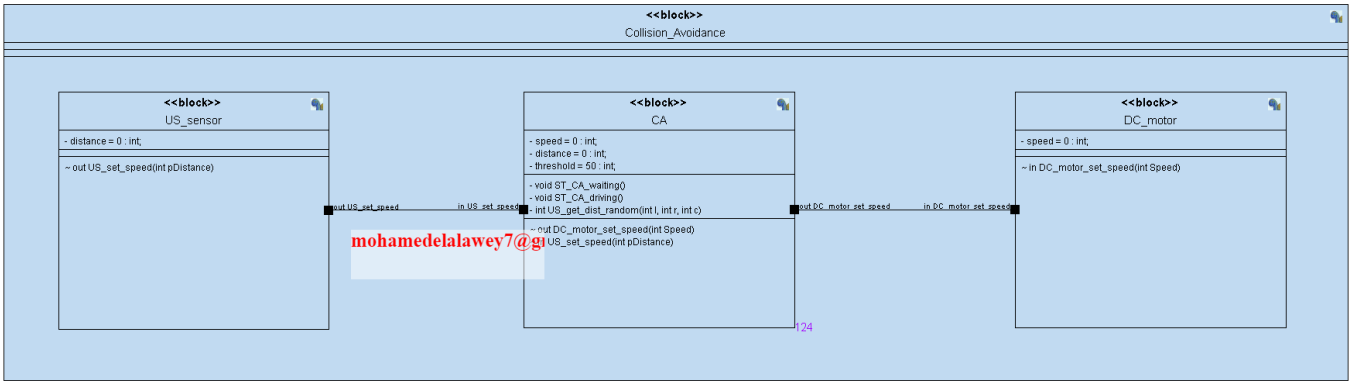
- **DC_motor Block** Receives speed commands from the CA block and simulates the response of the motor based on the speed value.





ıamedelalawey7@gı

3. UML Block Diagram Description



The **block definition diagram** presents the structural view of the system:

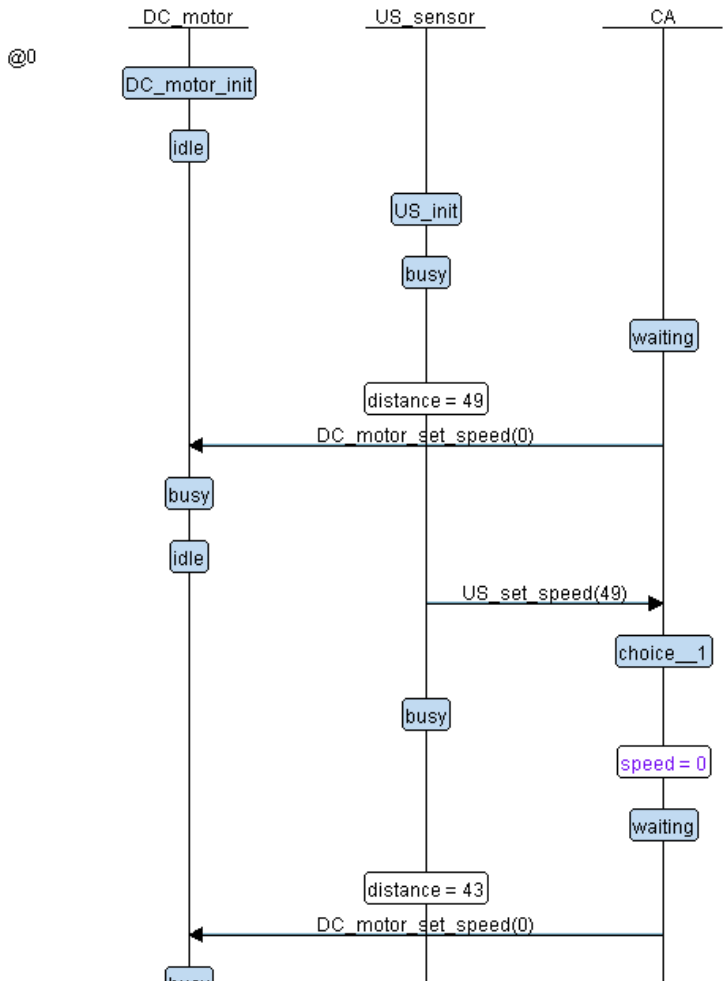
- The `US_sensor` has a distance attribute and sends its value to CA via `US_set_speed(int pDistance)`.
- The `CA` block contains the logic for determining the speed depending on the received distance. It has methods like `ST_CA_waiting`, `ST_CA_driving`, and `US_get_dist_random`.
- The `DC_motor` receives speed from the CA via `DC_motor_set_speed(int Speed)`.

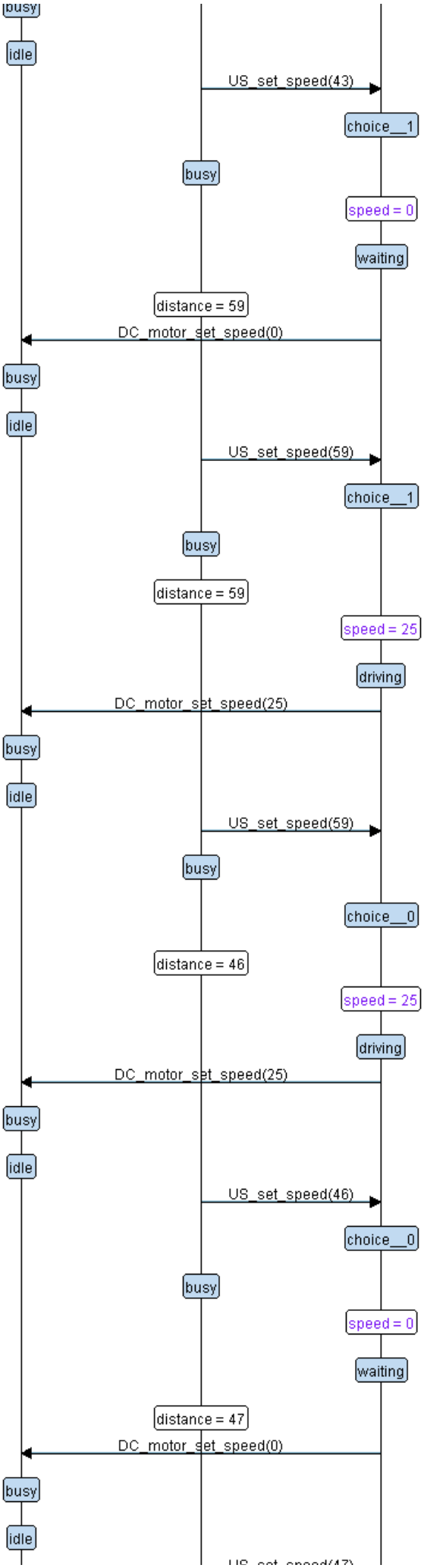
Communication between blocks is defined using directed connectors, indicating the flow of data between components.

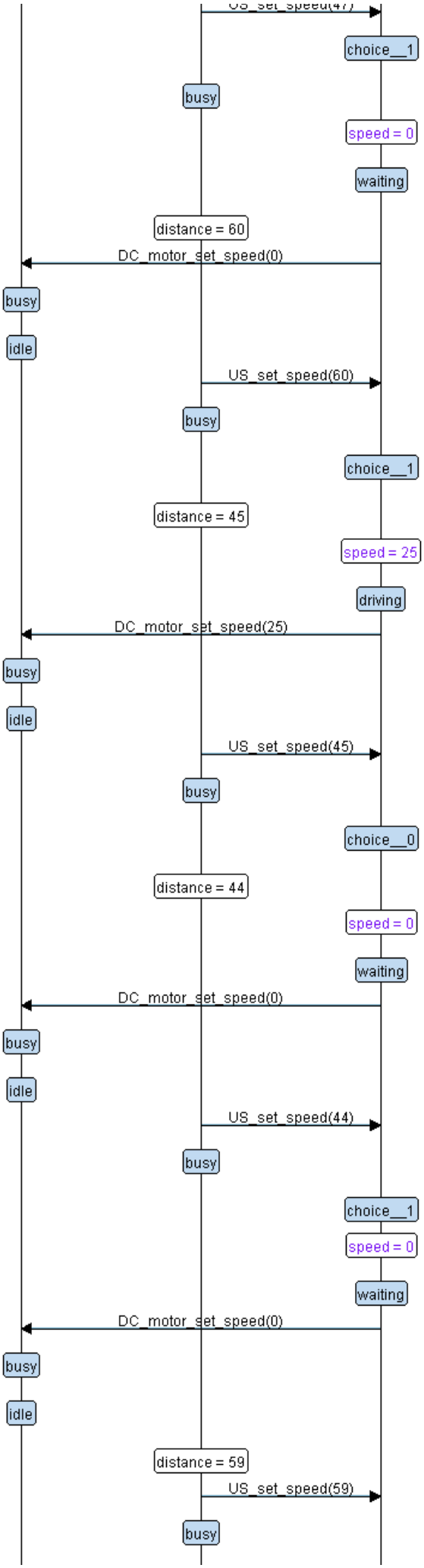
---

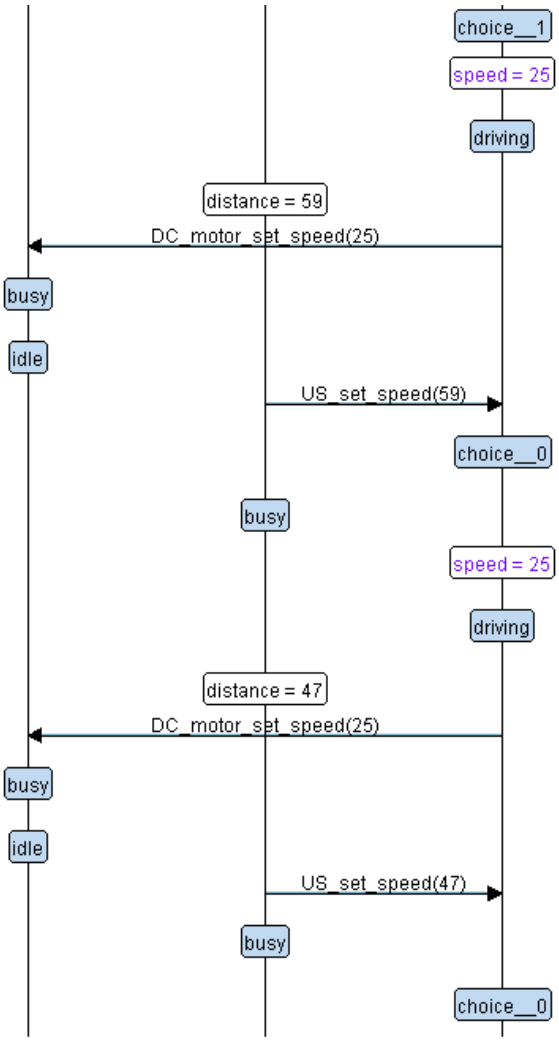
4. Behavior Modeling

4.1 Sequence Diagram Description




busy
idle
US_set_speed(43)
choice__1
busy
speed = 0
waiting
distance = 59
DC_motor_set_speed(0)
busy
idle
US_set_speed(59)
choice__1
busy
distance = 59
speed = 25
driving
DC_motor_set_speed(25)
busy
idle
US_set_speed(59)
busy
choice__0
distance = 46
speed = 25
driving
DC_motor_set_speed(25)
busy
idle
US_set_speed(46)
choice__0
busy
speed = 0
waiting
distance = 47
DC_motor_set_speed(0)
busy
idle


US_set_speed(47)
choice__1
busy
speed = 0
waiting
distance = 60
DC_motor_set_speed(0)
busy
idle
US_set_speed(60)
busy
choice__1
distance = 45
speed = 25
driving
DC_motor_set_speed(25)
busy
idle
US_set_speed(45)
busy
choice__0
distance = 44
speed = 0
waiting
DC_motor_set_speed(0)
busy
idle
US_set_speed(44)
busy
choice__1
speed = 0
waiting
DC_motor_set_speed(0)
busy
idle
distance = 59
US_set_speed(59)
busy



The **sequence diagram** depicts the dynamic behavior of the system over time:

1. **Initial State**: The `CA` is in the `waiting` state and begins to receive distance data from the `US_sensor`.
2. **Distance Update Loop**:
    - The sensor sends a new distance value (e.g., 60, 55, 40, etc.).
    - The controller (`CA`) processes this distance.
    - Based on the threshold (50), it sets an appropriate motor speed:
        - If `distance > threshold`: motor speed = 30
        - If `distance <= threshold`: motor speed = 0 (to stop)
    - This continues for multiple iterations simulating real-time decision-making and collision avoidance behavior.

---

5. Logic Implementation

5.1 Controller Logic

```c
/*
 * ca.c
 *
 *  Created on: May 16, 2025
 *      Author: Muhamad Elalawy
 */
#include "ca.h"

//variables
static int CA_speed = 0;
static int CA_distance =0;
static int CA_threshold=50;

//state pointer to function
void (*pCA_state)();



//APIs
void US_get_distance(int dist){


    CA_distance = dist;
    (CA_distance <= CA_threshold)? (pCA_state = STATE(CA_driving)) :(pCA_state =
STATE(CA_waiting));
    printf("US module sending distance = %d to CA module\n",CA_distance);
}



STATE_define(CA_waiting){
    CA_state_id = CA_waiting;
    printf("CA_waiting state: distance = %d , Speed = %d
\n",CA_distance,CA_speed);
    CA_speed = 0;
    DC_motor(CA_speed);
}
STATE_define(CA_driving){
    CA_state_id = CA_driving;
    printf("CA_driving state: distance = %d , Speed = %d \n",CA_distance ,
CA_speed);
    CA_speed = 25;
    DC_motor(CA_speed);
}
```

```c
/*
 * DC.c
 *
 *  Created on: May 16, 2025
```

```
 *      Author: Muhamad Elalawy
 */
#include "dc_motor.h"

//variables
static int DC_speed = 0;


//state pointer to function
void (*pDC_state)();



//APIs

void DC_init(){

    printf("DC_init.. \n");
}

void DC_motor(int s){
    DC_speed = s;
    pDC_state = STATE(DC_busy);
    printf("CA module sending speed = %d to DC module\n",DC_speed);

}

STATE_define(DC_idle){
    DC_state_id = DC_idle;
    printf("DC_idle state: speed = %d \n" , DC_speed);

}
STATE_define(DC_busy){
    DC_state_id = DC_busy;
    printf("DC_busy state: speed = %d \n" , DC_speed);
    pDC_state = STATE(DC_idle);
}
```

```
/*
 * US.c
 *
 *  Created on: May 16, 2025
 *      Author: Muhamad Elalawy
 */
#include "us_sensor.h"

//variables
static int US_distance =0;

//state pointer to function
void (*pUS_state)();
```

```c
//APIs
int US_get_distance_rand(int min , int max , int count){
    int i;
    for(i = 0; i<count ; i++){
        int rand_num=(rand()%(max-min+1))+min;
        return rand_num;
    }
}

void US_init(){
    printf("US_init.. \n");

}

STATE_define(US_busy){
    US_state_id = US_busy;
    US_distance = US_get_distance_rand(40,60,1);
    printf("US_waiting state: distance = %d \n",US_distance);
    US_get_distance(US_distance);
    pUS_state= STATE(US_busy);
}
```