

Table of Contents

1	Introduction	3
2	System architecting sequence	3
2.1	Case study.	3
2.1.1	Assumptions:	3
2.1.2	Versioning:.....	3
2.2	Method:	3
2.3	Requirement:.....	3
2.4	System analysis:	4
2.4.1	Use case diagram:.....	4
2.4.2	Activity diagram:	4
2.4.3	Sequence diagram	5
2.5	System design:.....	5
2.5.1	Overall system block diagram:.....	5
2.5.2	US logic:	6
2.5.3	CA logic:	6
2.5.4	DC logic:	7
3	Implementing code:.....	7
3.1	Collision avoidance main algorithm.....	7
3.2	Virtual Ultrasonic sensor Driver:.....	8
3.3	Virtual DC motor driver:.....	9
3.4	main.c Code:	10
4	Version one running on windows OS console:.....	11

Table of Figures

Figure 2.1 Requirement Diagram.	3
Figure 2.2 Use Case Diagram.	4
Figure 2.3 Activity Diagram.	4
Figure 2.4 Sequence diagram.	5
Figure 2.5 System Block Diagram.....	5
Figure 2.6 Virtual Ultrasonic Sensor Driver.	6
Figure 2.7 Collision Avoidance Main Algorithm.	6
Figure 2.8 Virtual DC Motor Driver.	7
Figure 3.1 Collision Avoidance Main Algorithm.....	7
Figure 3.2 Virtual Ultrasonic Driver Code.	8
Figure 3.3 Virtual DC Motor Driver Code.....	9
Figure 3.4 main.c Code.	10
Figure 4.1 Version One Test Result On Windows OS Console.....	11

1 Introduction

This simple prototype collision avoidance system is a sub-system of a self-driving car system, this report scope is to show a simple system architecting sequence using UML diagrams for only version one which is only simulate the collision avoidance system with printing data on console.

2 System architecting sequence

2.1 Case study.

2.1.1 Assumptions:

- Physical installation of components will not be designed.
- Sensors, controller, and actuators maintenance will not be considered.
- The controller will not face power cut ever.
- Sensors and actuators will never fail.
- The system will read distance values from an external sensor (sensor driver will not be modeled).
- System will control an external DC motor (DC motor driver will not be modeled).

2.1.2 Versioning:

- Version one: will simulate logic by printing data on windows OS console.
- Version two: full version which will be burned into controller.

2.2 Method:

This system will be implemented using V-model SDLC.

2.3 Requirement:

This section shows a very high view about how this system works and if it really meets the customer needs or not.

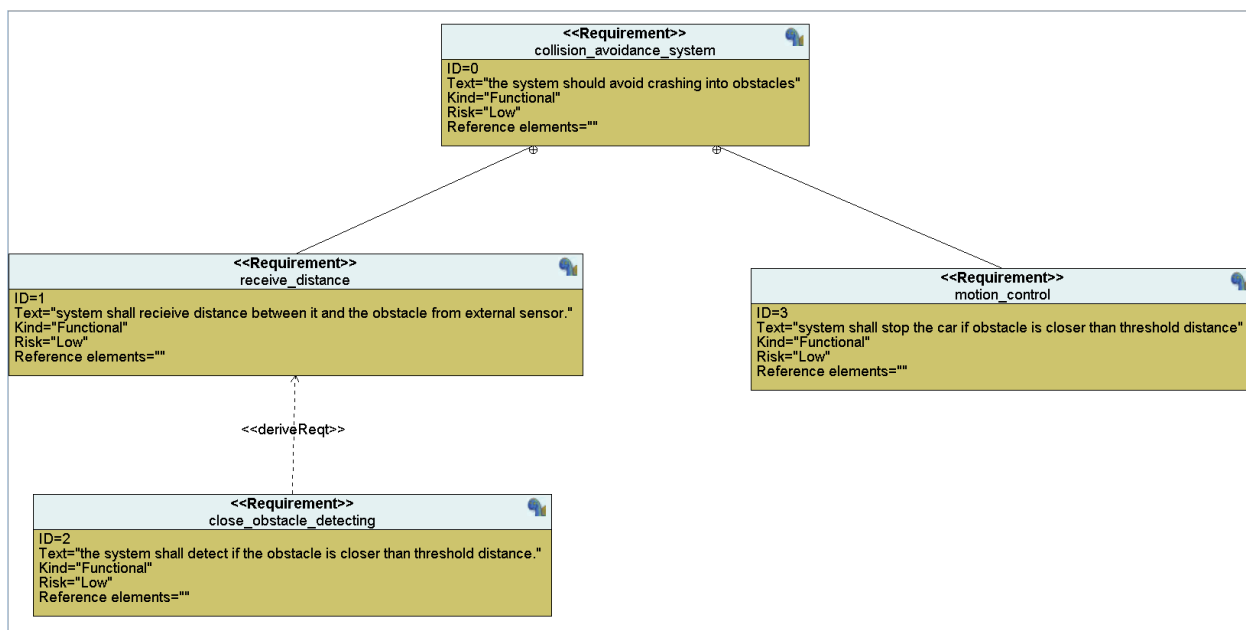


Figure 2.1 Requirement Diagram.

2.4 System analysis:

This section will give a deeper understanding of and how it works.

2.4.1 Use case diagram:

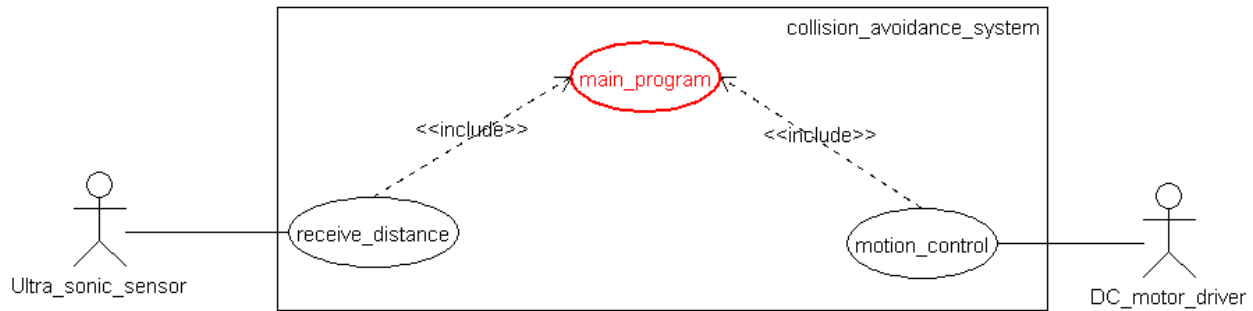


Figure 2.2 Use Case Diagram.

2.4.2 Activity diagram:

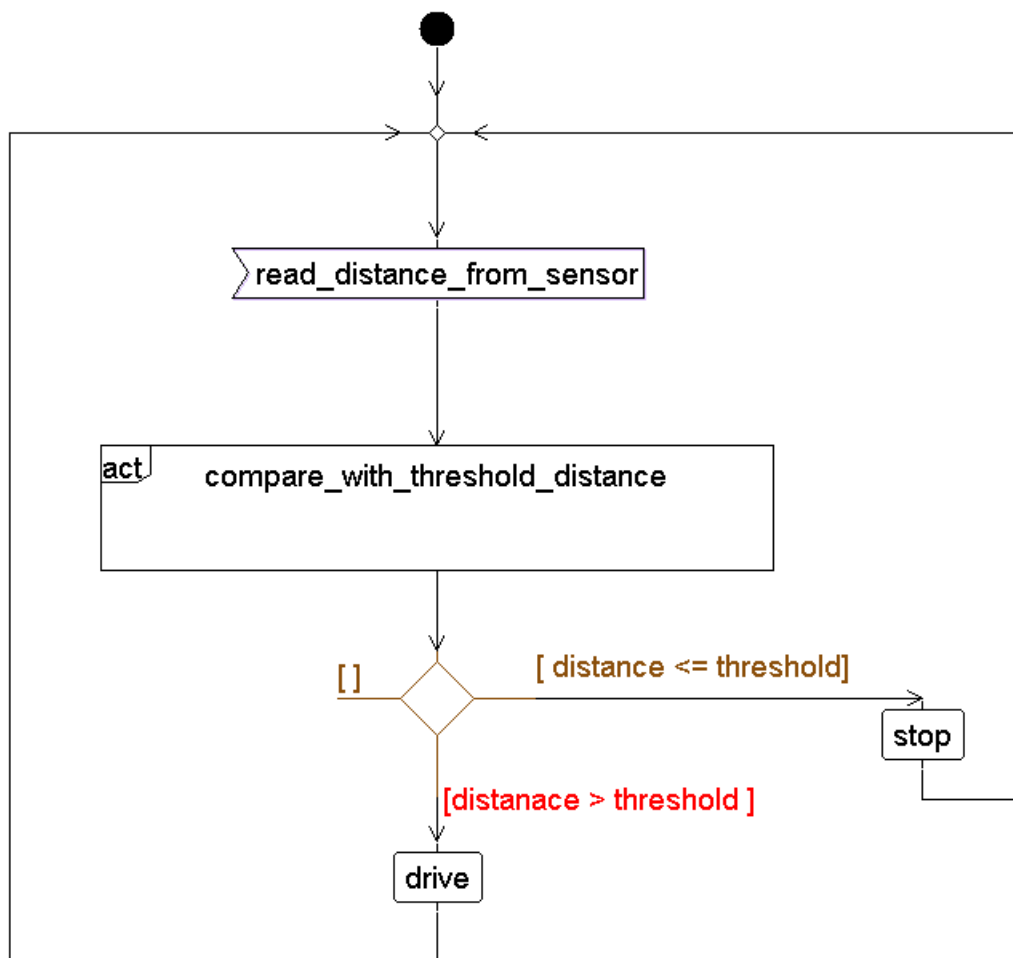


Figure 2.3 Activity Diagram.

2.4.3 Sequence diagram

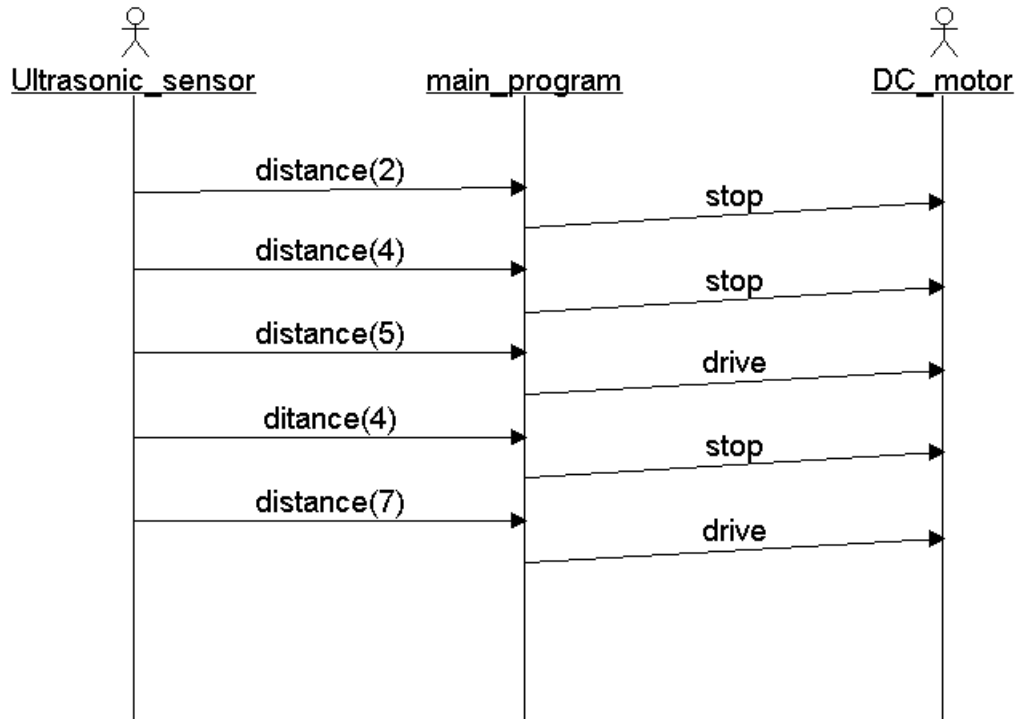


Figure 2.4 Sequence diagram.

2.5 System design:

This section will show how the program will be implemented by a low-level detail.

2.5.1 Overall system block diagram:

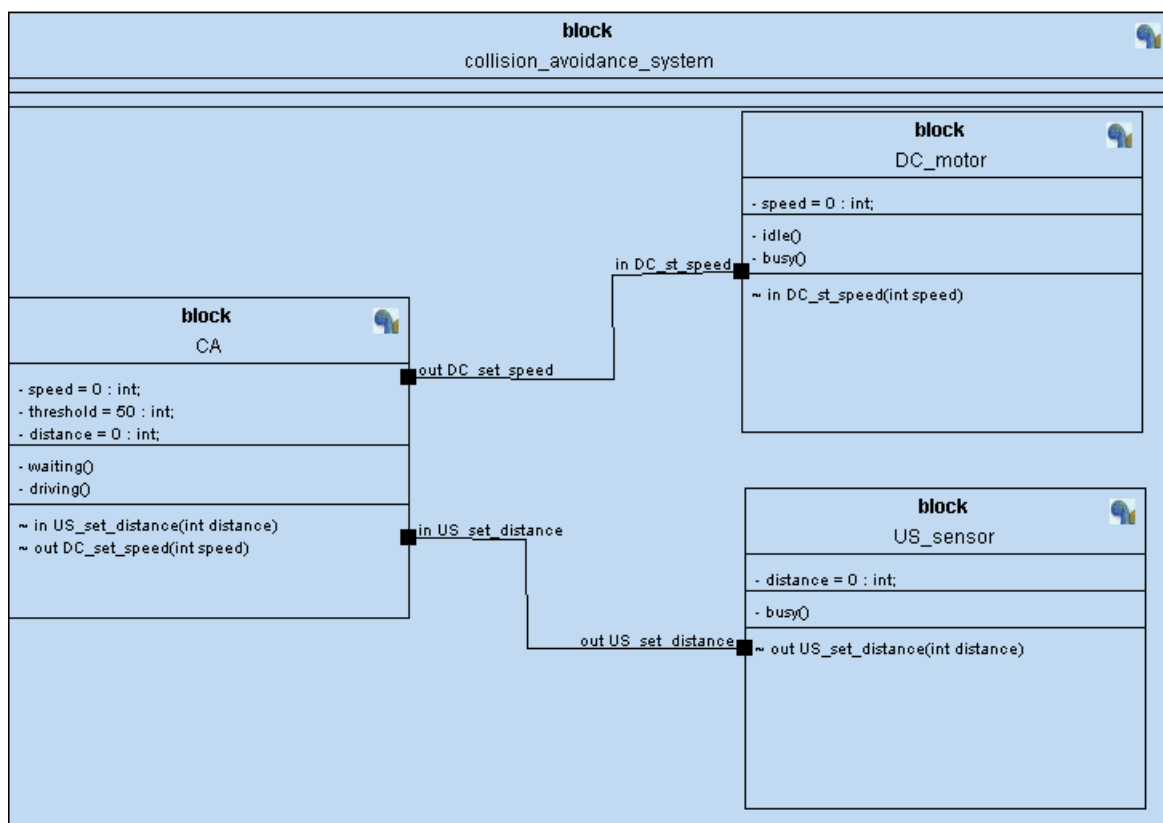


Figure 2.5 System Block Diagram

2.5.2 US logic:

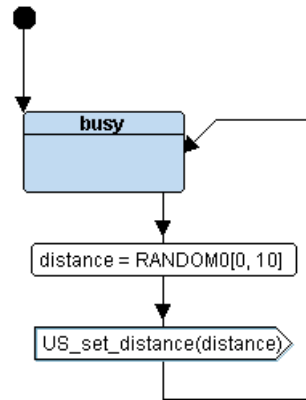


Figure 2.6 Virtual Ultrasonic Sensor Driver.

2.5.3 CA logic:

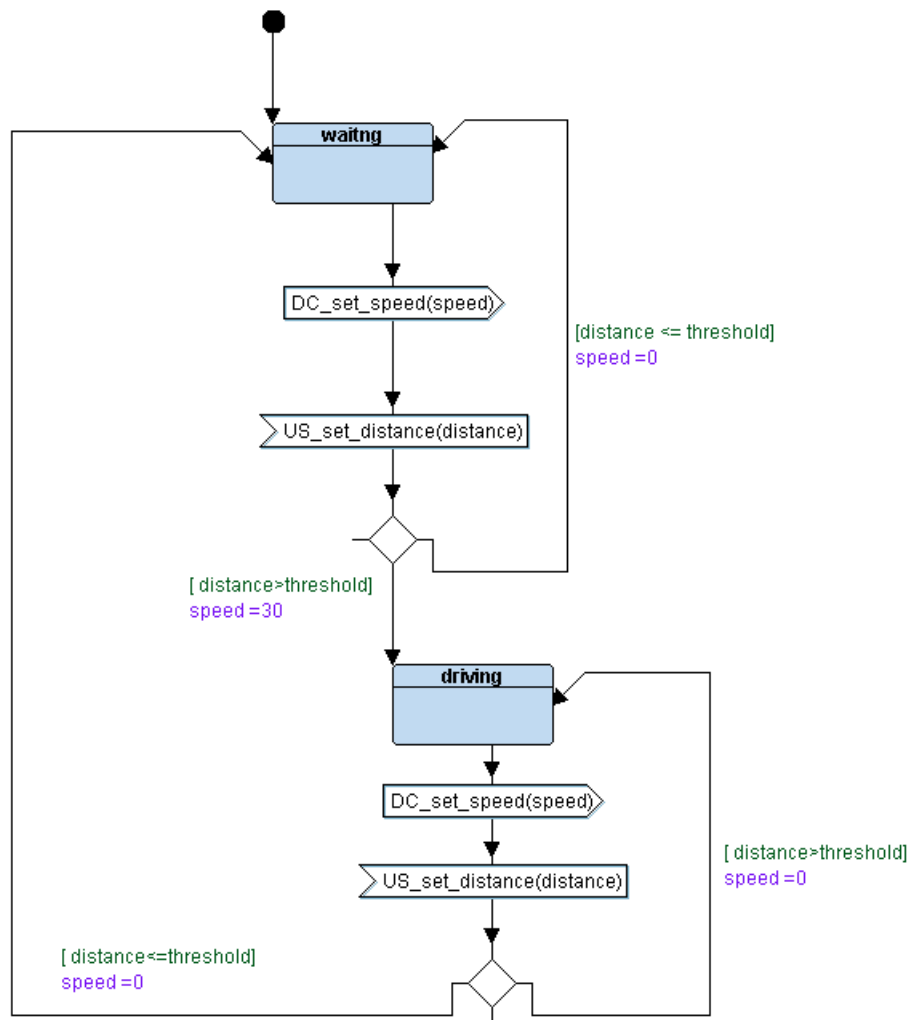


Figure 2.7 Collision Avoidance Main Algorithm.

2.5.4 DC logic:

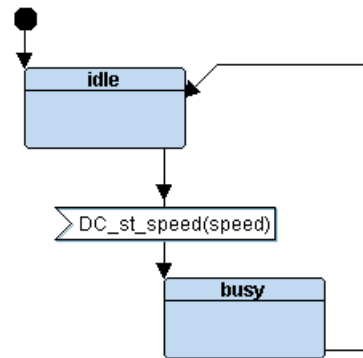


Figure 2.8 Virtual DC Motor Driver.

3 Implementing code:

3.1 Collision avoidance main algorithm

```
1  #include "stdio.h"
2  #include "DC.h"
3
4  #define threshold_distance 5
5
6  typedef enum {
7      _CA_waiting,
8      _CA_driving
9  }CA_status_ID_t;
10
11 void(* P_CA_state)();
12 int CA_distance = 0, CA_speed = 0;
13 CA_status_ID_t CA_ID;
14
15 void CA_waiting()
16 {
17     CA_ID = _CA_waiting;
18     DC_set_speed(0);
19     printf("waiting\n");
20 }
21
22 void CA_driving()
23 {
24     CA_ID = _CA_driving;
25     DC_set_speed(30);
26     printf("driving\n");
27 }
28
29 void US_set_dist(int distance)
30 {
31     CA_distance = distance;
32     (CA_distance <= threshold_distance)?(P_CA_state = CA_waiting):(P_CA_state = CA_driving);
33 }
```

Figure 3.1 Collision Avoidance Main Algorithm.

3.2 Virtual Ultrasonic sensor Driver:

This virtual driver is only used to generate a random value for distance to be used as a test cases.

```
1  #include "stdio.h"
2  #include "CA.h"
3
4  typedef enum {
5      _US_busy
6  }US_status_ID_t;
7
8  int distance;
9  US_status_ID_t US_ID;
10 void(* P_US_state)();
11
12 int random_value(int min, int max)
13 {
14     return (rand() % (max-min+1)) + 1;
15 }
16
17 void US_busy()
18 {
19     US_ID = _US_busy;
20
21     distance = random_value(0,10);
22     printf("US busy state, distance =%d.\n",distance);
23     US_set_dist(distance);
24     P_US_state = US_busy;
25 }
26
27 void US_init()
28 {
29     printf("-----US sensor is initialized-----\n");
30     P_US_state = US_busy;
31 }
```

Figure 3.2 Virtual Ultrasonic Driver Code.

3.3 Virtual DC motor driver:

This virtual driver will be only used to take order of driving or not from main algorithm to be tested by different distance values.

```
1  #include "stdio.h"
2
3  typedef enum {
4      _DC_idle,
5      _DC_busy
6  }DC_status_ID;
7
8  DC_status_ID DC_ID;
9  int speed;
10 void(* P_DC_state)();
11
12 void DC_idle()
13 {
14     DC_ID = _DC_idle;
15
16     printf("DC idle state, speed= %d.\n\n",speed);
17
18     P_DC_state = DC_idle;
19 }
20
21 void DC_busy()
22 {
23     DC_ID = _DC_busy;
24
25     printf("DC busy state, speed= %d.\n\n",speed);
26
27     P_DC_state = DC_idle;
28 }
29 void DC_set_speed(int s)
30 {
31     speed =s;
32     P_DC_state = DC_busy;
33 }
34
35 void DC_init()
36 {
37     printf("-----DC motor is initialized-----\n");
38     P_DC_state = DC_idle;
39 }
```

Figure 3.3 Virtual DC Motor Driver Code.

3.4 main.c Code:

```
1  #include "stdio.h"
2  #include "CA.h"
3  #include "DC.h"
4  #include "US.h"
5  void setup()
6  {
7      DC_init();
8      US_init();
9      printf("\n");
10 }
11
12 int main()
13 {
14     setup();
15     while (1)
16     {
17         P_US_state();
18         P_CA_state();
19         P_DC_state();
20     }
21     return 0;
22 }
23
24 }
```

Figure 3.4 main.c Code.

4 Version one running on windows OS console:

```
1  -----DC motor is initialized-----
2  -----US sensor is initialized-----
3
4  US busy state, distance =9.
5  driving
6  DC busy state, speed= 30.
7
8  US busy state, distance =10.
9  driving
10 DC busy state, speed= 30.
11
12 US busy state, distance =10.
13 driving
14 DC busy state, speed= 30.
15
16 US busy state, distance =2.
17 waiting
18 DC busy state, speed= 0.
19
20 US busy state, distance =8.
21 driving
22 DC busy state, speed= 30.
23
24 US busy state, distance =6.
25 driving
26 DC busy state, speed= 30.
27
28 US busy state, distance =6.
29 driving
30 DC busy state, speed= 30.
31
32 US busy state, distance =11.
33 driving
34 DC busy state, speed= 30.
35
36 US busy state, distance =2.
37 waiting
38 DC busy state, speed= 0.
```

Figure 4.1 Version One Test Result On Windows OS Console.