System design

Student(s)

Eng : Taha Omar BOULMANE

Unit

System design & data structures

Date of Submission

Sunday, July 11, 2021

Lecturer/Tutor

Keroles Shenoda

Abstract

By definition Systems design is the process of defining elements of a system like modules, architecture, components and their interfaces and data for a system based on the specified requirements. It is the process of defining, developing and designing systems which satisfies the specific needs and requirements of a business or organization. In this report we will try to design a simple collision avoidance system, using diagrams and state machines.

INTRODUCTION

Every system is a result of thinking about a solution to a specific problem. Before starting the process of building a hose we have to think about alternatives and solutions we will choose. As in construction field as in software area the architecting phase is inevitable. In this report we summary a simple entry point to system design by modeling the system in diagrams such as block diagram, state machine or flowchart. Our system consists on a simple collision avoidance which stop a motor (virtual) after receiving a sensed data from an ultrasonic sensor.

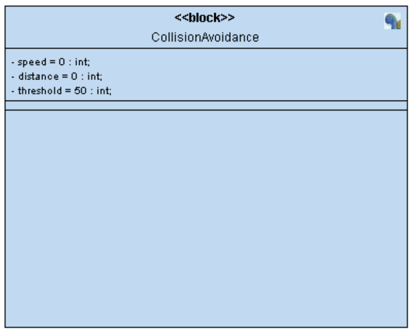
THEORY

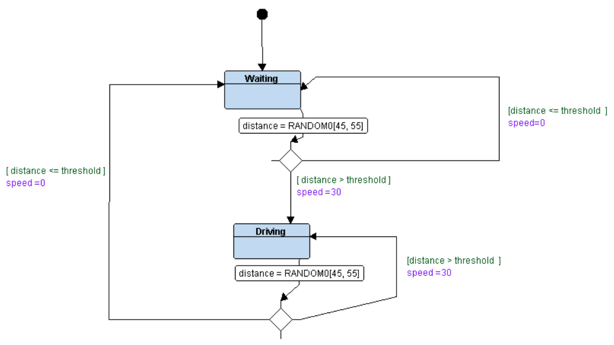
Product is today a system of subsystems, components and parts, and often embeds some kind of smartness and some capability of communicating. Consequently, the number of functions exploited is increasing, together with the number of interfaces. This leads to a certain complexity, poorly manageable through the tools widely used in the past. [1]

PROCEDURE

COLLISION AVOIDANCE SYSTEM ONE MODULE

in this section we will try to model a simple collision avoidance system which consists of an ultrasonic sensor an algorithm to decide if the actuators (motors) should run or stop, to do so we used the TTool. First, we created a block diagram for the whole system as shown in the figure below, we defined the variables speed distance and threshold then we tried to make a flowchart based on the states in which the system can be.





We used random values generated automatically from the tool to test the logic before adding the ultrasonic as an external module. In the next section we wil describe the code implementing the system in C.

ONE MODULE COLLISION AVOIDANCE C CODE:

To implement the block, we needed to create a header file for the CA, it contains the definition of the states of the system and the APIs, to clarify the code we define the states in a separate header called state.h

CA.H

```
#ifndef CA_H_
#define CA_H_
        #include "state.h"
        //define states
        enum
                {
                waiting,
                driving
                }state_id;

        //functions prototypes
                state_define(CA_waiting);
                state_define(CA_driving);

        //global pointer to function
                extern void (*CA_state)();
#endif /* CA_H_ */
```

STATE.H

```
#ifndef STATE_H_
#define STATE_H_
        //automatic state function generate
        #define state_define(_state_func_)            void st_##_state_func_()
        #define state(_state_func_)                           st_##_state_func_
#endif /* STATE_H_ */
```

Then the core of states is done on a C file

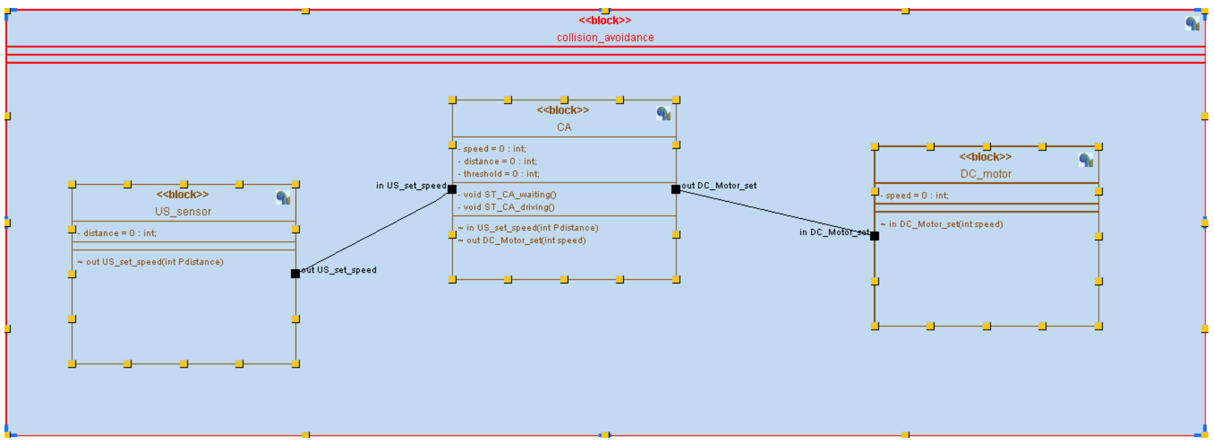
CA.C

```
#include <stdio.h>
#include <stdlib.h>
#include "CA.h"
#include "state.h"
int get_distance_random (int l, int r, int count);
//global variables
unsigned int speed = 0;
unsigned int distance = 0;
unsigned int threshold = 50;
//global pointer to function
void (*CA_state)();
state_define(CA_waiting)
{
        //state name
        state_id = waiting;
        //state action
        speed = 0;
        //generate random distance
        distance = get_distance_random (45, 55, 1);

        //check event
        (distance <= threshold) ? (CA_state = state(CA_waiting)) : (CA_state =
state(CA_driving));
        printf("Waiting state : distance = %d speed = %d\n", distance, speed);
}
state_define(CA_driving)
{
	//state name
	state_id = driving;
	//state action
	speed = 30;
	//generate random distance
	distance = get_distance_random (45, 55, 1);

	//check event
	(distance <= threshold) ? (CA_state = state(CA_waiting)) : (CA_state =
state(CA_driving));
	printf("Driving state : distance = %d speed = %d\n", distance, speed);
}
int get_distance_random (int l, int r, int count)
{
	int i, rand_num;
	for(i=0; i<count; i++)
	{
		rand_num =( rand() % (r - l + 1) ) + l;
	}
	return rand_num;
}
```
Then to test the system we need the main

MAIN.C

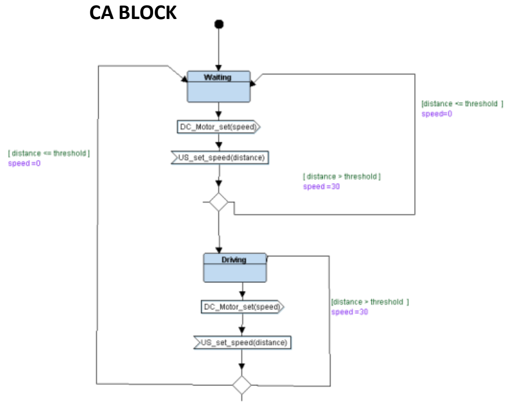
```
#include <stdio.h>
#include <stdlib.h>
#include "CA.h"

void setup ();
int main()
{
	volatile int d;
	setup();
	while(1)
	{
		//call state for each block
		CA_state();
		for(d=0; d<=1000; d++);
	}
	return 0;
}
void setup ()
{
	//initialize all drivers
	//initialize IRQ
	//initialize HAL US_DRIVER DC_DRIVER
	//set states pointers for each block
	CA_state = state(CA_waiting);
}
```

COLLISION AVOIDANCE SYSTEM MULTI MODULES MODEL

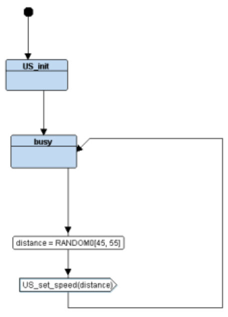


as shown in the block we can now add more blocks and think about the separately, also we can divide the development on teams each engineer or team of engineers can develop a block in the condition of respecting the interfaces that the integrators can gather all pieces easily and rapidly. The state machine of each block is shown in the figures below

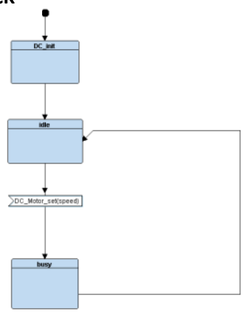
CA BLOCK



US BLOCK

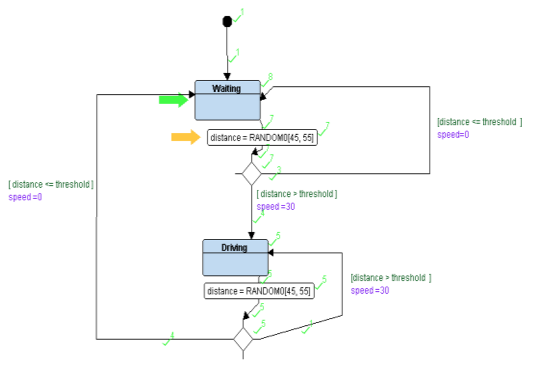


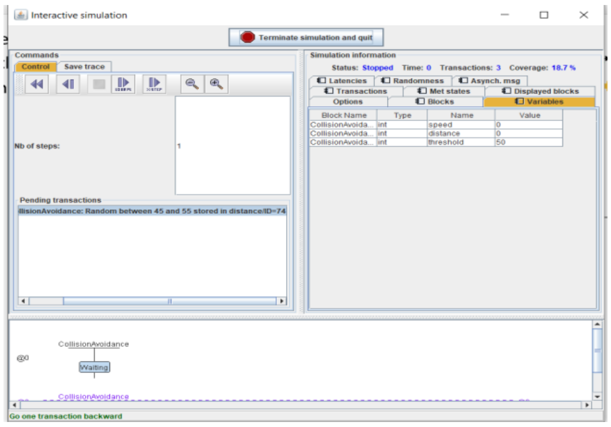
DC MOTOR BLOCK



RESULTS

after analyzing the syntax of the system, we can simulate interactively the system, the tool shows you the variables, the states and a other parameters to help you on understanding and rectifying the system logic.




Interactive simulation

Terminate simulation and quit

Commands — Control | Save trace

Nb of steps: 1

Pending transactions
llisionAvoidance: Random between 45 and 55 stored in distance/ID=74

Simulation information
Status: Stopped Time: 0 Transactions: 3 Coverage: 18.7 %

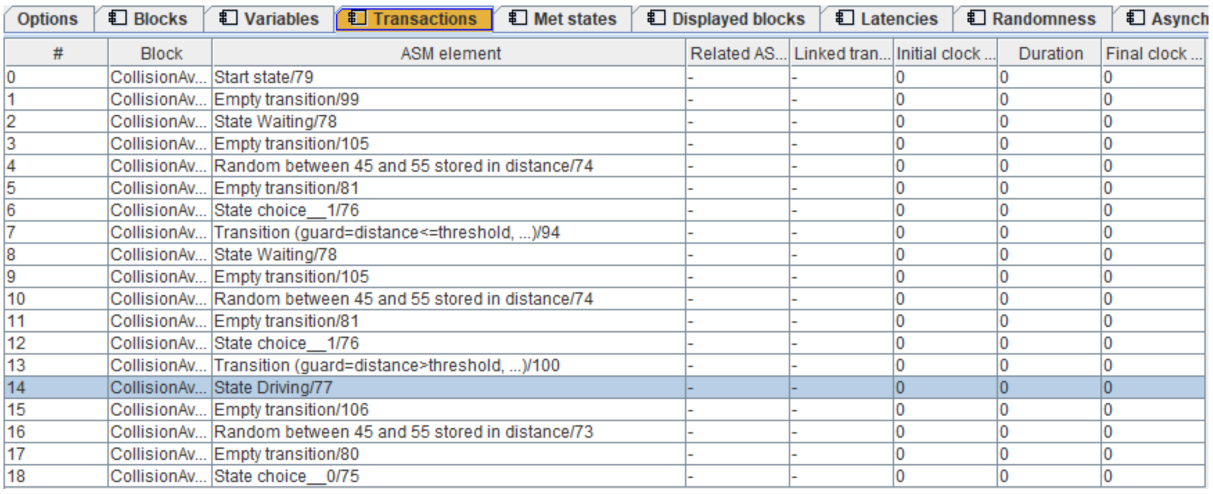
Latencies | Randomness | Asynch. msg | Transactions | Met states | Displayed blocks | Options | Blocks | Variables

Block Name	Type	Name	Value
CollisionAvoida...	int	speed	0
CollisionAvoida...	int	distance	0
CollisionAvoida...	int	threshold	50

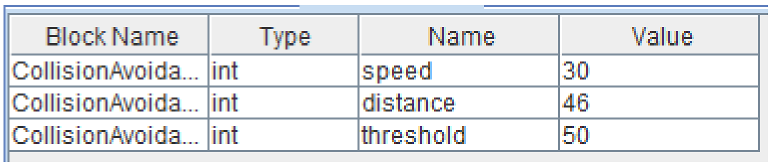
@0 CollisionAvoidance Waiting CollisionAvoidance

Go one transaction backward

Also we can visualize the number of transitions the variables values on each step


Options | Blocks | Variables | Transactions | Met states | Displayed blocks | Latencies | Randomness | Asynch

#	Block	ASM element	Related AS...	Linked tran...	Initial clock ...	Duration	Final clock ...
0	CollisionAv...	Start state/79	-	-	0	0	0
1	CollisionAv...	Empty transition/99	-	-	0	0	0
2	CollisionAv...	State Waiting/78	-	-	0	0	0
3	CollisionAv...	Empty transition/105	-	-	0	0	0
4	CollisionAv...	Random between 45 and 55 stored in distance/74	-	-	0	0	0
5	CollisionAv...	Empty transition/81	-	-	0	0	0
6	CollisionAv...	State choice__1/76	-	-	0	0	0
7	CollisionAv...	Transition (guard=distance<=threshold, ...)/94	-	-	0	0	0
8	CollisionAv...	State Waiting/78	-	-	0	0	0
9	CollisionAv...	Empty transition/105	-	-	0	0	0
10	CollisionAv...	Random between 45 and 55 stored in distance/74	-	-	0	0	0
11	CollisionAv...	Empty transition/81	-	-	0	0	0
12	CollisionAv...	State choice__1/76	-	-	0	0	0
13	CollisionAv...	Transition (guard=distance>threshold, ...)/100	-	-	0	0	0
14	CollisionAv...	State Driving/77	-	-	0	0	0
15	CollisionAv...	Empty transition/106	-	-	0	0	0
16	CollisionAv...	Random between 45 and 55 stored in distance/73	-	-	0	0	0
17	CollisionAv...	Empty transition/80	-	-	0	0	0
18	CollisionAv...	State choice__0/75	-	-	0	0	0


Block Name	Type	Name	Value
CollisionAvoida...	int	speed	30
CollisionAvoida...	int	distance	46
CollisionAvoida...	int	threshold	50

RESULT OF CODE COMPILATION

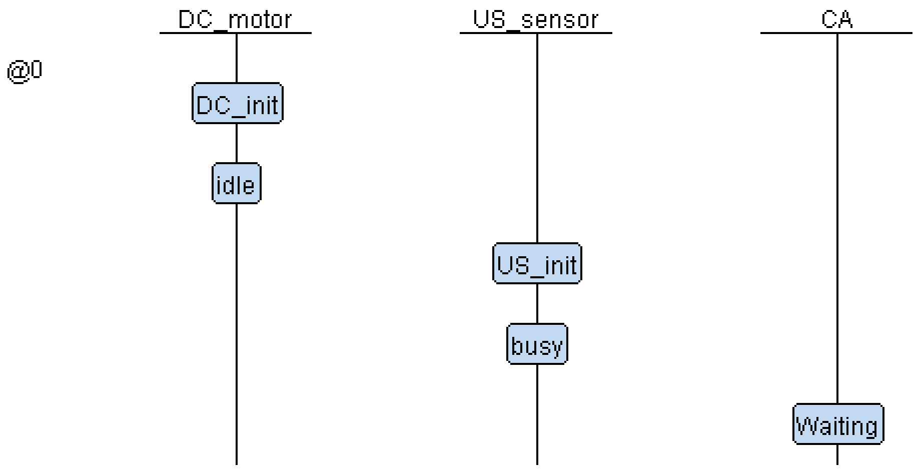
as seen before the code we wrote shows the results of the system logic

```
Driving state : distance = 47 speed = 30
Waiting state : distance = 46 speed = 0
Waiting state : distance = 55 speed = 0
Driving state : distance = 46 speed = 30
Waiting state : distance = 47 speed = 0
Waiting state : distance = 55 speed = 0
```

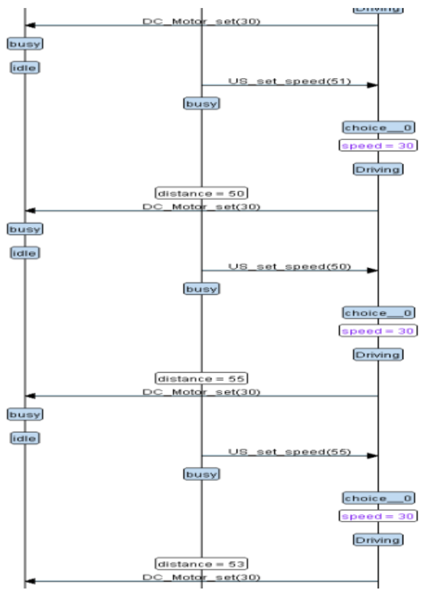
```
Driving state : distance = 53 speed = 30
Driving state : distance = 52 speed = 30
Driving state : distance = 54 speed = 30
Driving state : distance = 47 speed = 30
Waiting state : distance = 51 speed = 0
Driving state : distance = 54 speed = 30
Driving state : distance = 49 speed = 30
Waiting state : distance = 45 speed = 0
Waiting state : distance = 45 speed = 0
Waiting state : distance = 49 speed = 0
Waiting state : distance = 49 speed = 0
Waiting state : distance = 54 speed = 0
Driving state : distance = 47 speed = 30
```

MULTI MODULE COLLISION AVOIDANCE

after modeling the blocks, we can check the syntax the we get as result the sequence diagram



After stepping in the simulation, the sequence diagram grows



Conclusions

As seen through this lab we learned how to think before starting the work, we plan and we think on a high level of abstracting phenomena before coding the application. In this work we tried our first step in modeling by thinking about a small collision avoidance system, in the first version we generate random values to test our algorithms after that we enhanced our system by modularizing it on three modules an ultrasonic sensor, a collision avoidance algorithm and a DC motor

References

[1] E. Brusa, *System engineering and Its Application to industrial product Development*. 2018.