Collision Avoidance in Self-Driving Cars

Report

Name: Mohamed Waleed Elsayed Abdelbaset

CODE

- **MAIN PROGRAM**

  All driver header files are included for full implementation of the program in the main loop.

  Drivers are firstly initialized then starting states for each one is declared.

  In the while loop we continuously check for the states of each driver, that is then sent and processed with linking functions,

```c
#include "CA.h"
#include "US.h"
#include "DC.h"

void setup()
{
    //init drivers
    US_init();
    DC_init();

    CA_state =  STATE(CA_waiting);
    US_state =  STATE(US_busy);
    DC_state =  STATE(DC_idle);
}


void main()
{
    setup();

    while(1)
    {
        US_state();
        CA_state();
        DC_state();
        for(int i =0 ; i<10000; i++);

    }
}
```

- # States.h

  this header file gives convenient function builders for the rest of the driver files, also has the prototypes for the linking functions.

```c
#define STATE_define(_statFUN_) void ST_##_statFUN_ ()
#define STATE(_statFUN_) ST_##_statFUN_

void US_set_distance(int d);
void DC_motor(int s);

```

- # CA .h/.c

  Enum for waiting and driving states is made, also a pointer to a state functions is declared as well as the functions themselves

  The CA files determines its state from the condition of the calculated distance from the US sensor, if its driving, meaning the condition is false, its state id is updated as well as the current speed which is then sent to the DC motor via the linking function provided in the states.h file

```c
#include "states.h"
#include "stdio.h"
#include "stdlib.h"


enum{
    CA_waiting,
    CA_driving
};
extern int CA_state_id;

STATE_define(CA_driving);
STATE_define(CA_waiting);

extern void (*CA_state)();
```

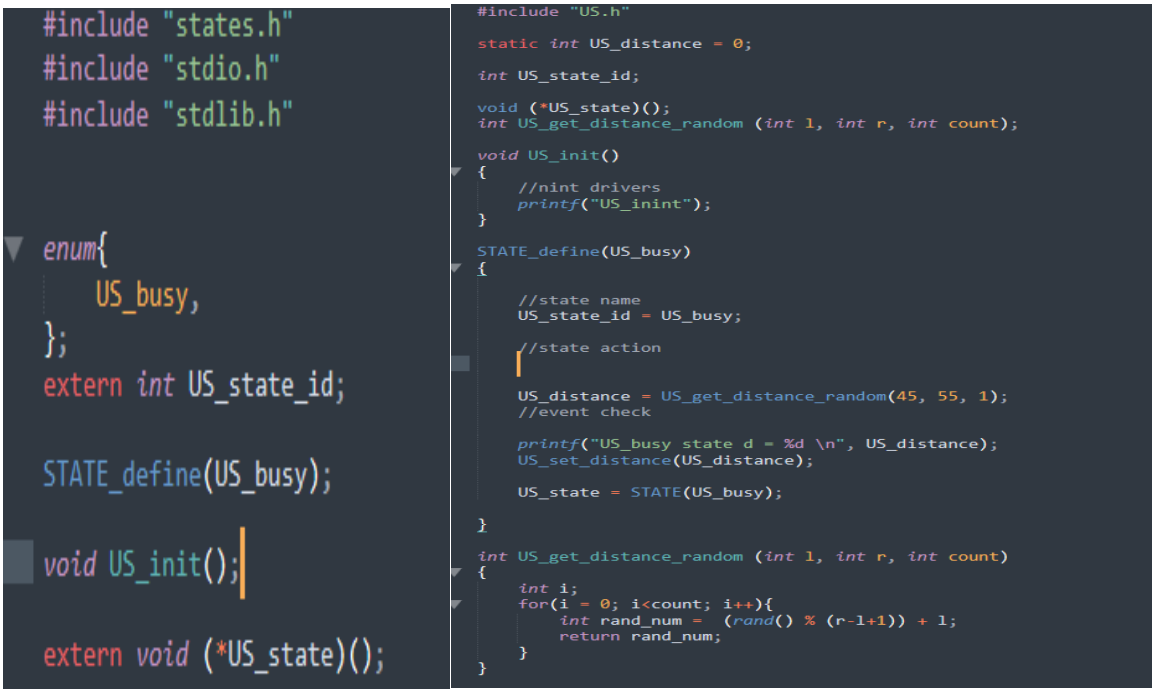
```c
#include "CA.h"

static int CA_speed  = 0;
static int CA_distance = 0;
static int CA_threshold  = 0;

int CA_state_id;

void US_set_distance(int d)
{
    CA_distance = d;
    CA_state = (CA_distance > CA_threshold) ? STATE(CA_driving) : STATE(CA_waiting);
    printf("US------------DISTANCE=%d-----------> CA\n", CA_distance);

}

void (*CA_state)();
int US_get_distance_random (int l, int r, int count);

STATE_define(CA_driving)
{
    //state name
    CA_state_id = CA_driving;
    printf("CA_driving state d = %d speed = %d \n", CA_distance, CA_speed);
    //state action
    CA_speed = 30;
    DC_motor(CA_speed);



}

STATE_define(CA_waiting)
{

    //state name
    CA_state_id = CA_waiting;
    printf("CA_waiting state d = %d speed = %d \n", CA_distance, CA_speed);
    //state action
    CA_speed = 0;
    DC_motor(CA_speed);
}
```

- # US .h/.c

  Only one state is provided, the bust state

  Firstly the US has an init function, and then defining the busy state and its functionality

  The state id is updated, the distance fitched (pseudo random values) and then outputted through the linking function “us_set_distance”



```c
#include "states.h"
#include "stdio.h"
#include "stdlib.h"


enum{
    US_busy,
};
extern int US_state_id;

STATE_define(US_busy);

void US_init();

extern void (*US_state)();
```

```c
#include "US.h"

static int US_distance = 0;

int US_state_id;

void (*US_state)();
int US_get_distance_random (int l, int r, int count);

void US_init()
{
    //nint drivers
    printf("US_inint");
}

STATE_define(US_busy)
{

    //state name
    US_state_id = US_busy;

    //state action


    US_distance = US_get_distance_random(45, 55, 1);
    //event check

    printf("US_busy state d = %d \n", US_distance);
    US_set_distance(US_distance);

    US_state = STATE(US_busy);

}

int US_get_distance_random (int l, int r, int count)
{
    int i;
    for(i = 0; i<count; i++){
        int rand_num =  (rand() % (r-l+1)) + l;
        return rand_num;
    }
}
```

- # DC .h/.c

  Two states are provided, idle and busy.

  Firstly its initialized then both state functions are implemented.

  The DC gets its speed from the CA and then goes into busy state.

```h
#include "states.h"
#include "stdio.h"
#include "stdlib.h"


enum{
    DC_idle,
    DC_busy
};
extern int DC_state_id;

STATE_define(DC_idle);
STATE_define(DC_busy);

void DC_init();

extern void (*DC_state)();
```

```c
#include "DC.h"

static int DC_speed  = 0;

int DC_state_id;

void (*DC_state)();


void DC_init()
{
    printf("DC_init \n");

}

void DC_motor(int s)
{
    DC_speed = s;
    DC_state = STATE(DC_busy);
    printf("CA-----------SPEED=%d----------> DC\n", DC_speed);
}

STATE_define(DC_idle)
{
    //state name
    DC_state_id = DC_idle;

    printf("DC_idle state speed = %d \n", DC_speed);
}

STATE_define(DC_busy)
{

    //state name
    DC_state_id = DC_busy;

    DC_state = STATE(DC_idle);
    printf("DC_busy state speed = %d \n", DC_speed);

}
```

- # Ouputs and diagrams:

C:\Users\mw296\OneDrive\Desktop\TEST C\unit4\lesson2\CA.c - Sublime Text (UNREGISTERED)

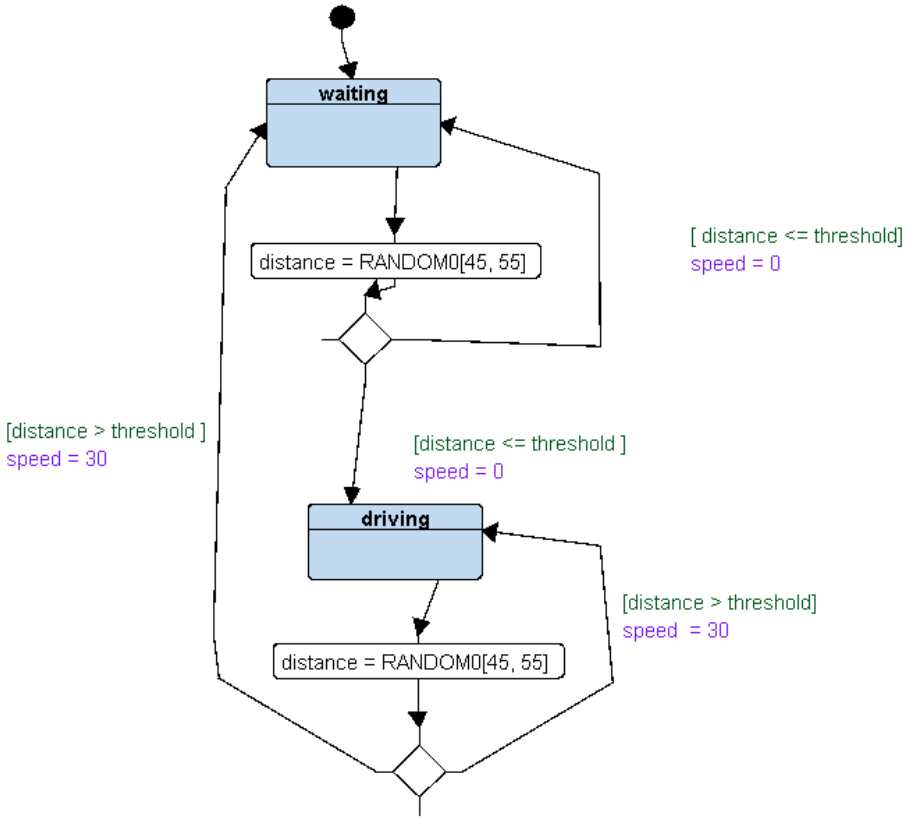
File Edit Selection Find View Goto Tools Project Preferences Help

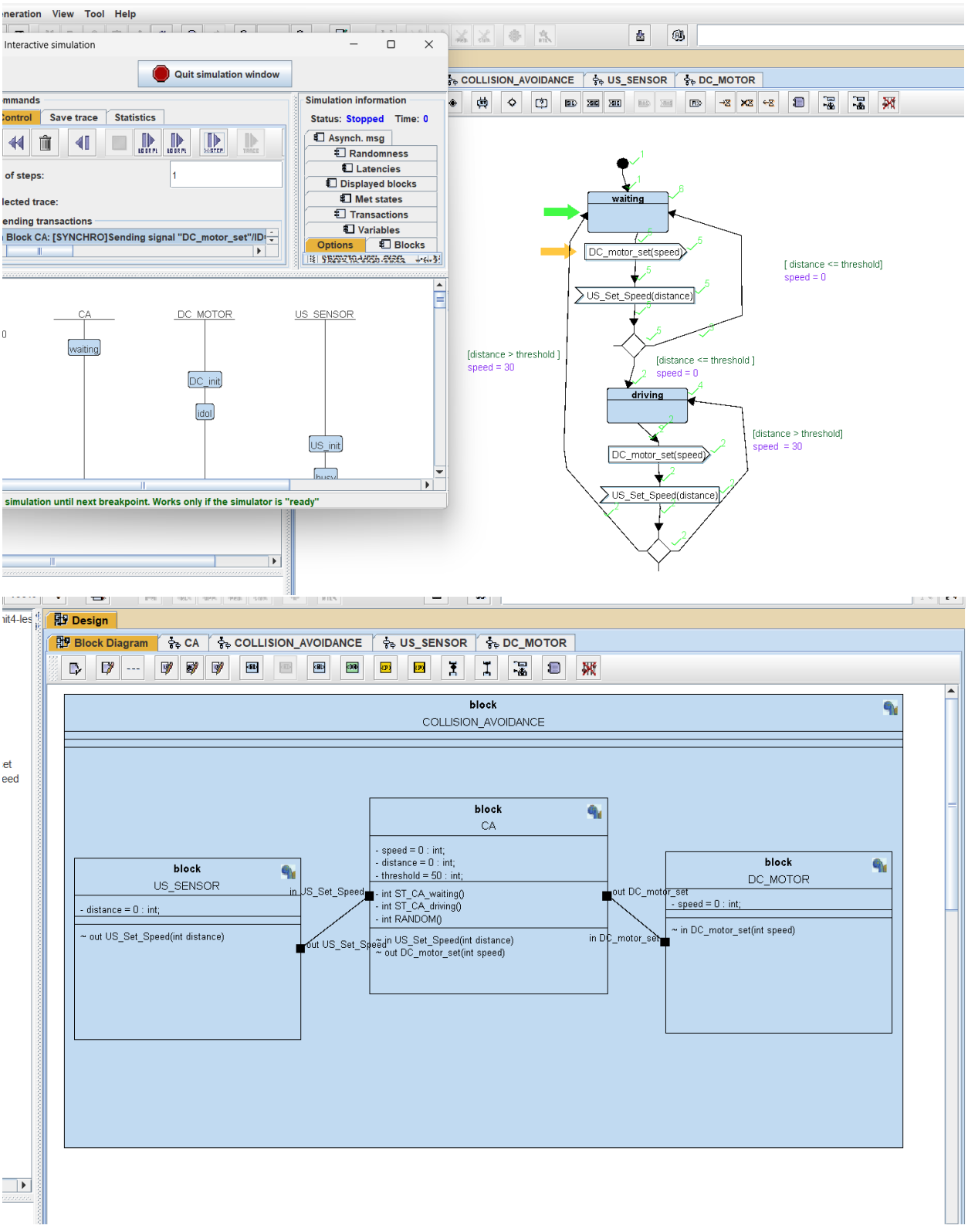
CA.h | states.h | main.c | CA.c

```c
{
    //state name
    CA_state_id = CA_driving;

    //state action
    CA_speed = 30;

    CA_distance = US_get_distance_random(45, 55, 1);
    //event check
    CA_state = (CA_distance > CA_threshold) ? STATE(CA_driving) : STATE(CA_waiting);

    printf("CA_driving state d = %d speed = %d \n", CA_distance, CA_speed);
}

STATE_define(CA_waiting)
{

    //state name
    CA_state_id = CA_waiting;

    //state action
    CA_speed = 0;

    CA_distance = US_get_distance_random(45, 55, 1);
    //event check
    CA_state_id = (CA_distance > CA_threshold) ? STATE(CA_driving) : STATE(CA_waiting);
    printf("CA_waiting state d = %d speed = %d \n", CA_distance, CA_speed);

}

int US_get_distance_random (int l, int r, int count)
{
    int i;
    for(i = 0; i<count; i++){
        int rand_num =  (rand() % (r-l+1)) + l;
        return rand_num;
    }
}
```

C:\Users\mw296\OneDrive\D

```
CA_waiting state d = 48 speed = 0
CA_waiting state d = 48 speed = 0
CA_waiting state d = 52 speed = 0
CA_waiting state d = 48 speed = 0
CA_waiting state d = 49 speed = 0
CA_waiting state d = 49 speed = 0
CA_waiting state d = 53 speed = 0
CA_waiting state d = 53 speed = 0
CA_waiting state d = 45 speed = 0
CA_waiting state d = 48 speed = 0
CA_waiting state d = 48 speed = 0
CA_waiting state d = 47 speed = 0
CA_waiting state d = 50 speed = 0
CA_waiting state d = 47 speed = 0
CA_waiting state d = 46 speed = 0
CA_waiting state d = 47 speed = 0
CA_waiting state d = 54 speed = 0
CA_waiting state d = 45 speed = 0
CA_waiting state d = 55 speed = 0
CA_waiting state d = 53 speed = 0
CA_waiting state d = 50 speed = 0
CA_waiting state d = 50 speed = 0
CA_waiting state d = 48 speed = 0
CA_waiting state d = 48 speed = 0
CA_waiting state d = 47 speed = 0
CA_waiting state d = 53 speed = 0
CA_waiting state d = 47 speed = 0
CA_waiting state d = 54 speed = 0
CA_waiting state d = 50 speed = 0
CA_waiting state d = 52 speed = 0
```




Interactive simulation
Quit simulation window
Simulation information
Status: Stopped Time: 0
Asynch. msg
Randomness
Latencies
Displayed blocks
Met states
Transactions
Variables
Options
Blocks
Save trace
Statistics
of steps:
Block CA: [SYNCHRO]Sending signal "DC_motor_set"/ID
CA
DC_MOTOR
US_SENSOR
waiting
DC_init
idol
US_init
simulation until next breakpoint. Works only if the simulator is "ready"
COLLISION_AVOIDANCE
US_SENSOR
DC_MOTOR
waiting
DC_motor_set(speed)
US_Set_Speed(distance)
[distance <= threshold]
speed = 0
[distance > threshold]
speed = 30
[distance <= threshold]
speed = 0
driving
DC_motor_set(speed)
US_Set_Speed(distance)
[distance > threshold]
speed = 30
Design
Block Diagram
CA
COLLISION_AVOIDANCE
US_SENSOR
DC_MOTOR
block
COLLISION_AVOIDANCE
block
US_SENSOR
- distance = 0 : int;
~ out US_Set_Speed(int distance)
in US_Set_Speed
out US_Set_Speed
block
CA
- speed = 0 : int;
- distance = 0 : int;
- threshold = 50 : int;
- int ST_CA_waiting()
- int ST_CA_driving()
- int RANDOM()
~ in US_Set_Speed(int distance)
~ out DC_motor_set(int speed)
out DC_motor_set
in DC_motor_set
block
DC_MOTOR
- speed = 0 : int;
~ in DC_motor_set(int speed)