TRAFFIC COLLISION AVOIDANCE SYSTEM

Lucas López Llorca

Álvaro Pérez Pecharromán **Group 821**

Raed Mahmoud Al-Chabon **Aerospace Engineering**

Table of contents

1. TCAS system description.

Traffic collision avoidance system, also known as TCAS, was developed to reduce the risk of mid-air collisions between aircraft. This system implements different devices that when working together provide collision protection.

1.1. Electronics of detection.

The TCAS involves communication between 2 or more aircraft equipped with an appropriate transponder. This transponder is an essential device, as it allows the TCAS system to build a three-dimensional map of the aircraft in the space. To build this 3D map, we need first to incorporate the range between the aircraft which is calculated by the interrogation and response time between the transponders. Then, the altitude should be estimated, and this is also done by the transponder that interrogates the other aircraft to automatically receive its altitude. In order to finish this 3D map, it is also needed to stablish some antennas that allow the system to estimate the intruder direction. Finally, when all this information is known the system extrapolates it to anticipate future values and determine if a potential collision risk exists. If this potential collision exists, the TCAS automatically negotiates a mutual avoidance manoeuvre which nowadays is restricted to changes in altitude.

1.2. Alerts and Instructions.

It is important to highlight that there are two types of alerts, the traffic advisory, and the resolution advisory. The criteria for giving a traffic advisory (TA) is between 35 to 48 seconds before the closest point of approach while a resolution advisory (RA) is given between 20 to 30 seconds before the closest point of approach. These advisories are generated by the TCAS computer unit, which also determines and indicates the optimal instructions to the crew.

When a traffic advisory is emitted, the pilots does not have to do any manoeuvre, but they are instructed to initiate a visual search for the traffic and to maintain the visual separation from the intruder aircraft. However, when a resolution advisory is emitted, pilots are expected to respond immediately to the instruction that for example can be ascend, descend or maintain the vertical speed.

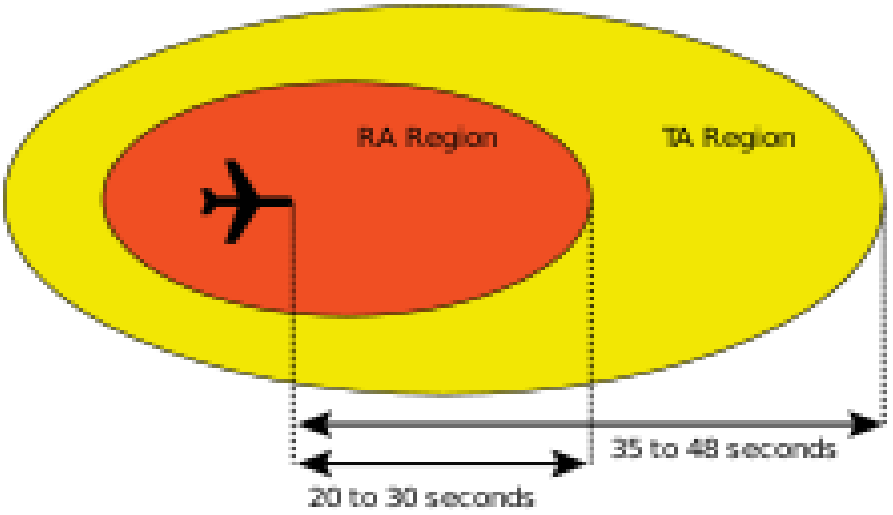


Figure 1.1: Example of TCAS protection criteria.

1.3. Information Display.

Finally, the cockpit presentation of all this information is a critical aspect as it allows the crew to better understand the situation. The display of this information can be done in different ways, but it is always built from the 3D map we explained before.

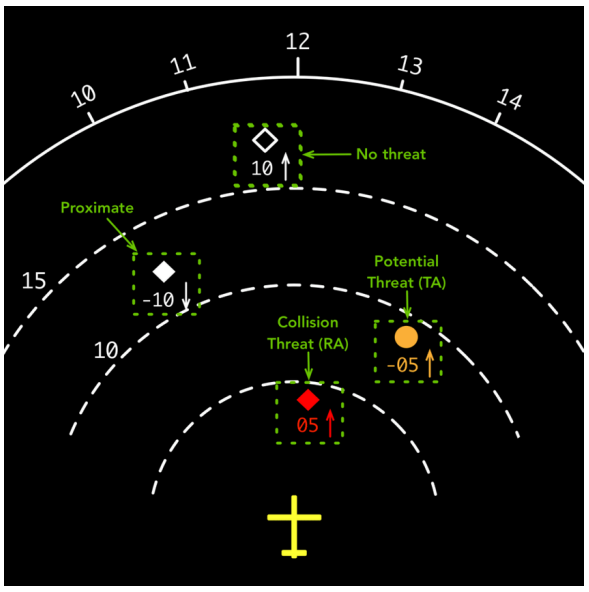


Figure 1.2: Example of TCAS representation.

2. Our simplifications of the system.

We made some simplifications in order to be able to simulate the system in proteus and make it work properly.

Starting with the electronics of detection, as we cannot incorporate a transponder in proteus, we implemented 3 LDR that make the function of proximity sensor, determining also in which direction the intruder aircraft is approaching.

Regarding the instructions and the alerts, what we are going to do is depending on the proximity measured by our three LDR and the height difference, which is determined by a potentiometer, an orange or red led will be switched on. This will also affect the manoeuvre by changing the position of the servomotor so that there is a proportion between the proximity and the ascending or descending velocity.

Once we have simplified the system in order to be able to detect the presence of the intruder aircraft and execute the proper instruction depending on the case, we also simplified the way in which the information is displayed, by implementing a matrix in proteus. In this matrix the position of our aircraft will always be at centre, and depending on the direction of the intruder, it will be displayed at left, centre or right of the matrix.

With all this we can start describing the hardware and software of our project.

3. Description of the simulated system

3.1. Block diagram

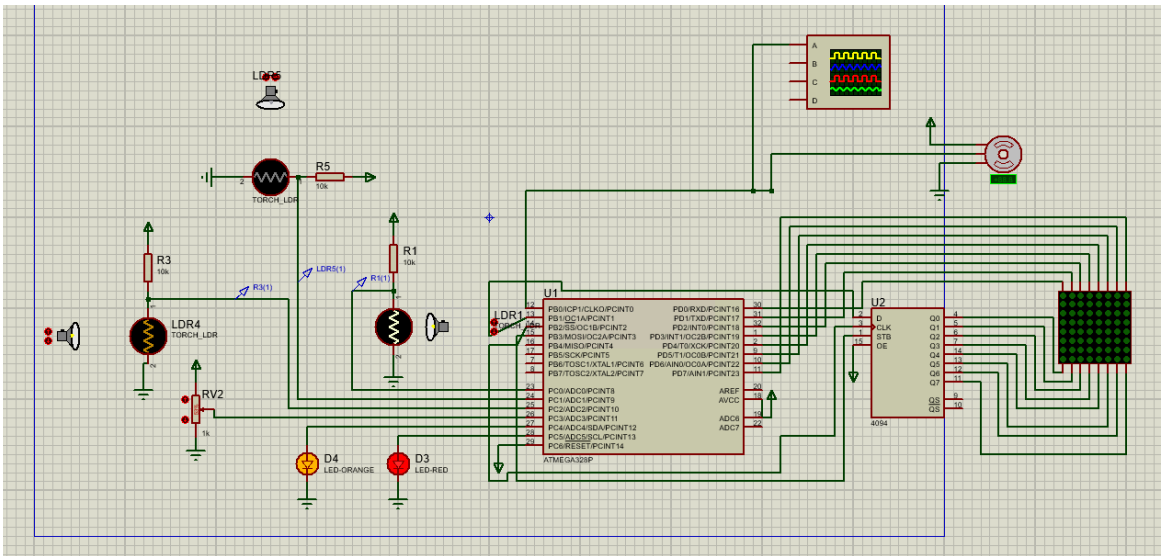


Figure 1.3: Complete system

3.2. Description of the Hardware

3.2.1. Inputs

For representing the incoming airplanes we have used LDRs. These are intended to simulate the trasponder. Moreover, to introduce to the system the difference in heights between both planes, a potentiometer is used.

A LDR is an electronic component whose resistance changes when it is exposed to light. Assembling it with a voltage supply and a resistance we can produce differences in voltage that can be received by the microcontroller. There are a total of three LDRs. One on the left, one on top and the last one on the right. This has been designed in this way to cover all the frontal area of the aircraft.

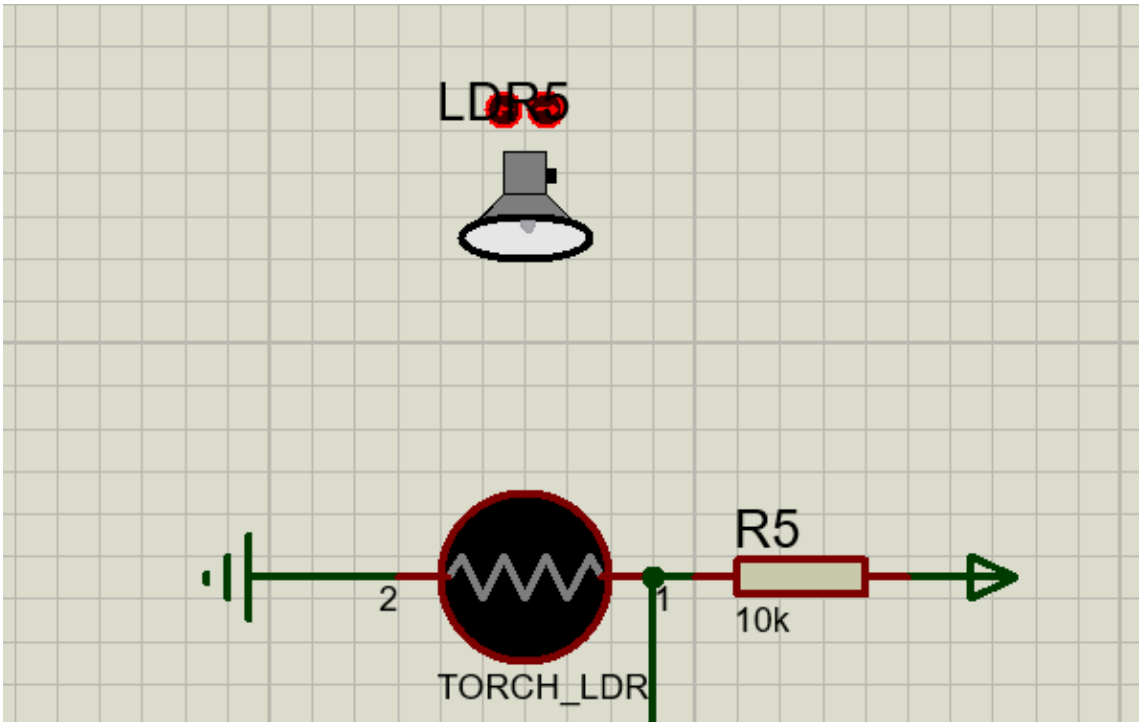


Figure 1.4: Circuit of the LDR

A potentiometer is a component whose resistance can be manually changed. This project is used to create a signal that can be received by the microcontroller that represents the difference between the height of both planes. A 50% will mean both planes have the same altitude while a 0 or 100 % will mean maximum distance.

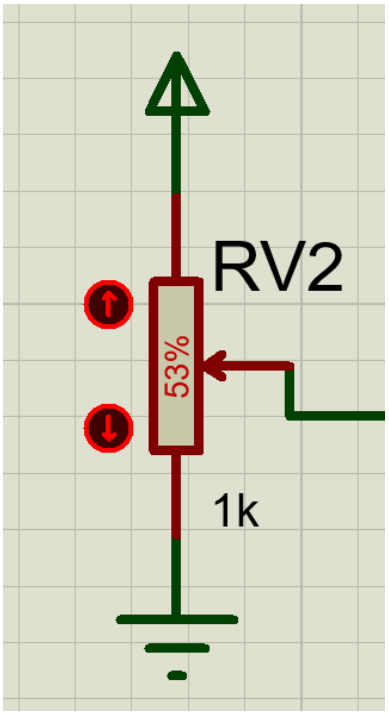


Figure 1.5: Circuit of the potentiometer

3.2.2. Leds

Two leds have been included in the circuit to simulate the warnings that the pilot will receive. It should be noted that the orientation is important for these components. One of the leds has been selected to be orange and represents a traffic advisory, something to be known but not critical. A red led, on the other hand, is used to represent an alert that needs immediate action (resolution advisory) because it represents a critical thread.

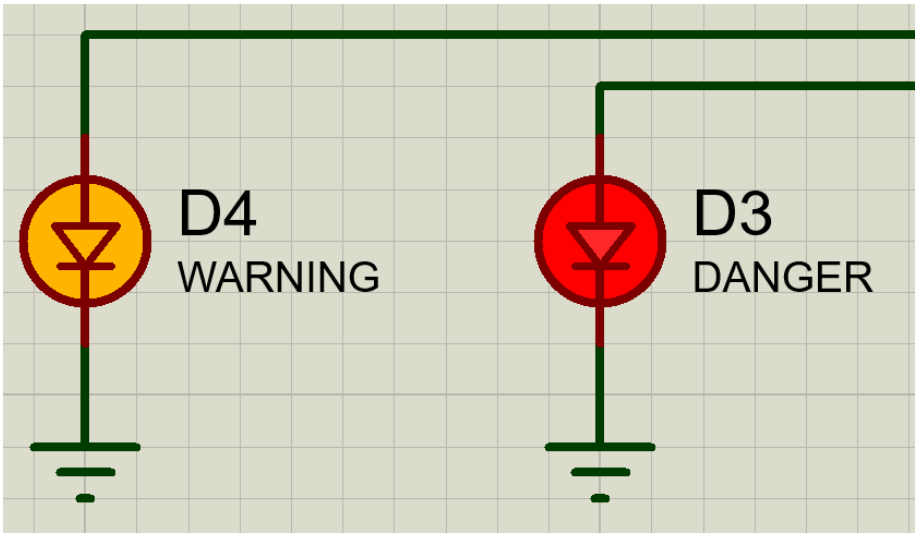


Figure 1.6: Circuit of the LEDs

3.2.3. Matrix display

In our simplified Traffic Collision Avoidance System (TCAS), we use one 8x8 LED matrix dot display to represent 4 different alerts: frontAlert, rightAlert, leftAlert and idleAlert. The 3 first alerts indicate a potential collision from the front, right, or left, respectively, while the last one indicates no potential alert.

The display of one image or another will obviously depend on the value of the input signals, this is the 3 LDR´s and the potentiometer.

So, an 8x8 LED matrix consists of 64 LEDs arranged in an 8-row by 8-column grid. Each LED is positioned at the intersection of a specific row and column. The matrix has two sets of connections:

- **Upper Connections (Rows):** Used to select which row is currently active.
- **Lower Connections (Columns of the displayed row):** Used to control which LEDs in the active row are lit. The 8-bit data for each row of the corresponding array is sent to these connections, determining the light pattern for that row.

The matrix display operates through a process called *multiplexing*. This means that only one row is switched on at any given time. The microcontroller scans through the rows rapidly, activating each row one by one while sending the corresponding column data to the lower part

connections to light the appropriate LEDs. This scanning occurs at a high frequency, creating the illusion of a stable image to the human eye.

To perform the scanning process, where each row is activated one at a time, 8 pins of the microcontroller are needed. Additionally, to send the corresponding data (the information byte for the activated row), another 8 pins of the ATmega328P microcontroller would be required. However, since the microcontroller has a limited number of pins, this data will be sent serially from the microcontroller and then converted to parallel using a shift register.

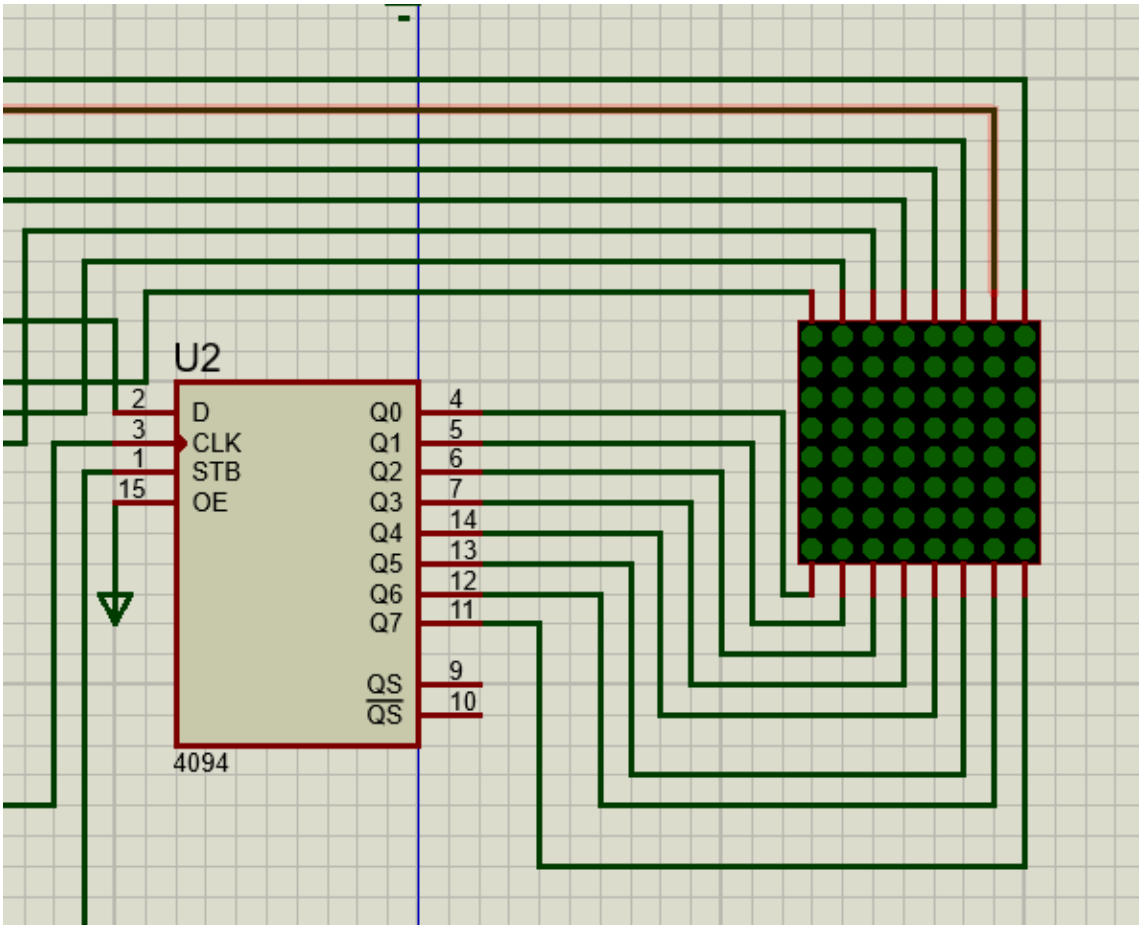


Figure 1.7: Circuit of the 4094 register and the matrix display

3.2.4. 4094 Register

To facilitate the scanning process using limited pins on the microcontroller, we use a serial-to-parallel shift register, specifically the 4094 register.

The microcontroller sends the data serially to the 4094 shift register. This data represents which LEDs in the current row of the scanning process should be switched on. The 4094 shift register then converts the serial data into parallel data. The parallel outputs from the 4094 shift register

are connected to the lower part connections of the display matrix. These outputs determine which LEDs in the active row will be lit.

Using a serial-to-parallel shift register like the 4094, as previously said, significantly reduces the number of required microcontroller pins. Instead of needing 16 pins (8 for rows and 8 for columns of the displayed row), a total of 11 pins are only needed for serial communication and row control. This method efficiently manages pin limitations by serializing the data transmission.

3.2.5 PWM Servo motor

For controlling the position of the aircraft and avoiding the possible collision a servomotor is employed. This servo simulates the control on the tail of the plane to produce a pitch angle and avoid the crash. It receives PWMs with the signal on between 1 and 2 ms and the signal low for 20 ms. It produces rotations between 90 and -90 º.

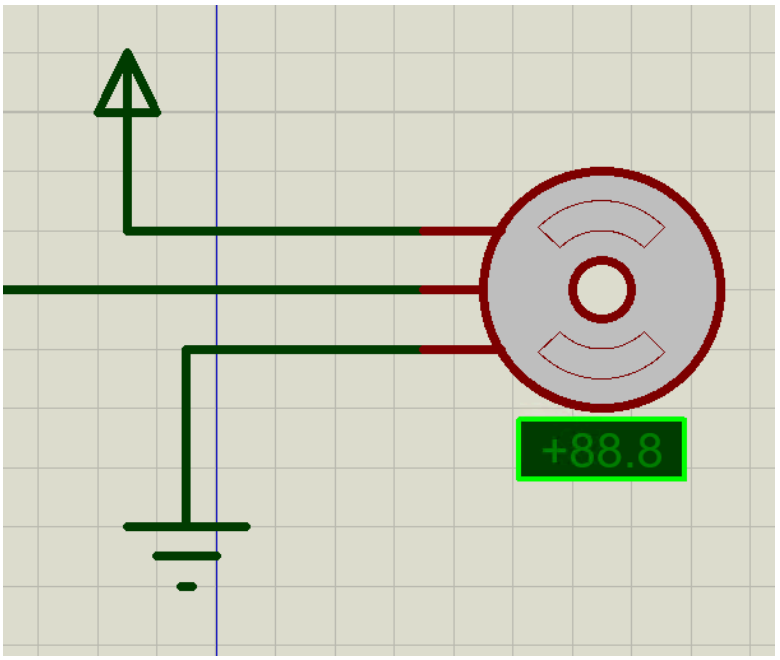


Figure 1.8: Circuit of the servomotor

3.2.6. ATMega328P

The microcontroller is in charge of interpreting all the information from the sensors and apply the corresponding control action through the leds, the servomotor and the matrix display. For converting the analog signals, four ADCs are used. More information about these and the PWMs created for the servomotor can be found in Section 3.3 Description of the Software.

In Figure 1.9 a image with all the conections can be seen.

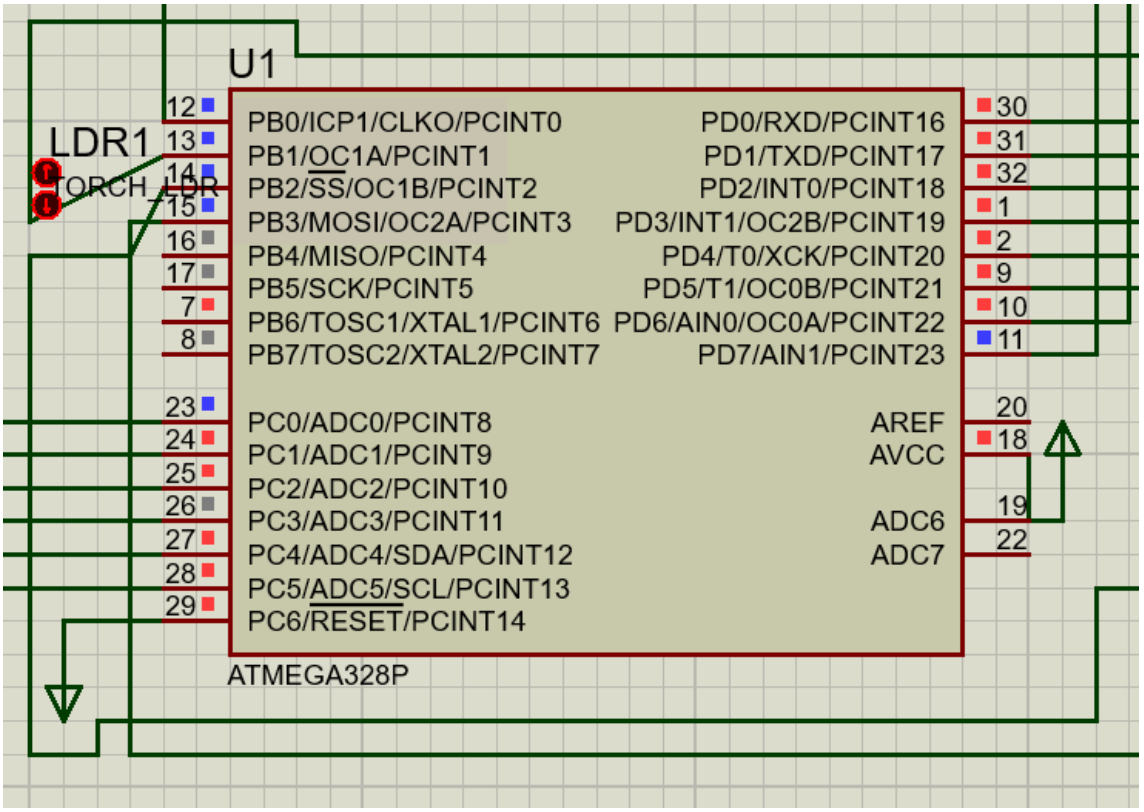


Figure 1.9: Circuit of the microcontroller

3.3. Description of the Software

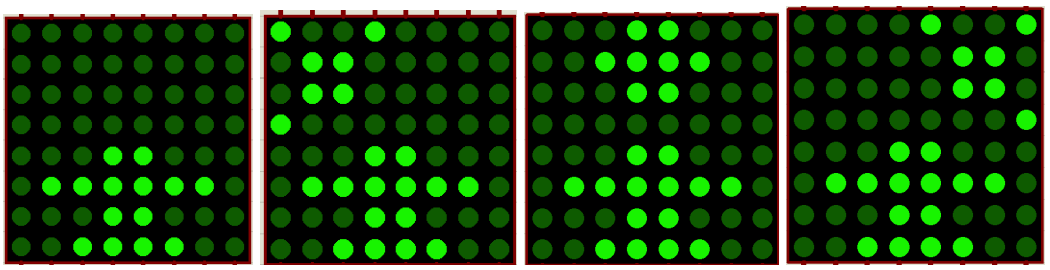
3.3.1. **Main variables definition**

```
float detectionLeft = 0;

float detectionRight = 0;

float detectionFront  = 0;

float detectionHeigth  = 0;

float VLeft = 0;

float VRight= 0;

float VFront= 0;

float VHeigth = 0;


float d_left = 0;

float d_rigth = 0;
```

```
float d_front= 0;

float d_heigth = 0;

int alerta = 0;


int duty = 70;

unsigned char count = 0;


char adc_flag = 0;

int ac = 0;

int ac_timer = 0;

unsigned int servo = 0;

int row=0xFE;

int i=0;

unsigned char leftAlert[8]={0b00001001, 0b00000110, 0b00000110, 0b00000001, 0b00011000, 0b01111110, 0b00011000, 0b00111100};

unsigned char frontAlert[8]={0b00011000, 0b00111100, 0b00011000, 0b00000000, 0b00011000, 0b01111110, 0b00011000, 0b00111100};

unsigned char rigthAlert[8]={0b10010000, 0b01100000, 0b01100000, 0b10000000, 0b00011000, 0b01111110, 0b00011000, 0b00111100};

unsigned char idle[8]={0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00011000, 0b01111110, 0b00011000, 0b00111100};
```

The leftAlert, rightAlert, frontAlert and idle are 8x8 matrices that store the information of the images to be displayed on the LED dot matrix. Each row of these arrays contain the data to be transmitted to the lower part connections of the dot matrix. A '1' indicates that the corresponding column of the displayed row will light up, while a '0' indicates an LED that will remain off.

The rest of the variables are the ones related to the functionality of the ADC and the logic related with the alert system.



3.3.2 **Setup Function**

```
void setup ()
{
```

First, the inputs and outputs are defined:

```
DDRC=0b0110000;// PC6 AND PC5 set as outputs
PORTC=0x00; // Pull-up resistor disable for PC0
DDRB=0b00001111 //
PORTB &= ~(1 << PB0); // Inic. PB0 a '0'
DDRD=0xFF;//Set PORTD as output
PORTD=0xFF;// Initialize PORTD pins as high state
```

Then, the registers related for the Timer 1 Overflow are described:

```
TCCR1A=0x00;//Normal mode operation
TCCR1B=0x02; // WGM13..12=00 / CS12..10=010 => prescaler=8
TCNT1H = 25536/256; // Recarga para temporizador de 20 ms: 65536-(20E-3*16E6)/8
TCNT1L = 25536%256;

TIMSK1|=(1<<TOIE1); // Enable Timer 1 overflow interrupt
```

Enable the ADC bit ADEN and define the Prescaler for the clk signal of the ADC.Fadc must be between 50 and 200 KHz.

```
ADCSRA =0b10000111; // ADEN=1 (ADC on) / ADATE=0 (single conversion mode)
// ADPS2..ADPS0=111 (division factor=128 => Fadc=125 kHz


}
```

3.3.3 **MAIN LOOP**

The first thing that the main loop does is to check if the ADC flag is on. If so, it starts the ADC conversion of the three LDRs and the potentiometer. The code for doing so is shown hereunder.

```
 if (adc_flag == 1)
   {
 ADMUX=0b01000000; // REFS1=0;REFS0=1 (VREF=AVCC)// ADLAR=0 (Result is right justified => 10-bit resolution )///MUX3..0=0000 (Channel 0 is selected)
ADCSRA|=(1<<ADSC); // ADSC set to 1 to start conversion
while ((ADCSRA&0b01000000)!=0); // Wait loop until ADSC is '0' (end of conversion)
detectionLeft=ADCL + 256*ADCH; ; // The result is read, and the 10-bit value is calculated/ /IMPORTANT: ADCL should be read first

ADMUX=0b01000001; // REFS1=0;REFS0=1 (VREF=AVCC)// ADLAR=0 (Result is right justified => 10-bit resolution )///MUX3..0=0001 (Channel 1 is selected)
ADCSRA|=(1<<ADSC); // ADSC set to 1 to start conversion
while ((ADCSRA&0b01000000)!=0); // Wait loop until ADSC is '0' (end of conversion)
detectionFront=ADCL + 256*ADCH; // The result is read, and the 10-bit value is calculated/ /IMPORTANT: ADCL should be read first
```

```
 ADMUX=0b01000010; // REFS1=0;REFS0=1 (VREF=AVCC)// ADLAR=0 (Result is right justified => 10-bit resolution )///MUX3..0=0010(Channel 2 is selected)
ADCSRA|=(1<<ADSC); // ADSC set to 1 to start conversion
while ((ADCSRA&0b01000000)!=0); // Wait loop until ADSC is '0' (end of conversion)
detectionRight=ADCL + 256*ADCH; // The result is read, and the 10-bit value is calculated/ /IMPORTANT: ADCL should be read first

ADMUX=0b01000011; // REFS1=0;REFS0=1 (VREF=AVCC)Result is right justified => 10-bit resolution )///MUX3..0=0011(Channel 3 is selected)
ADCSRA|=(1<<ADSC); // ADSC set to 1 to start conversion
while ((ADCSRA&0b01000000)!=0); // Wait loop until ADSC is '0' (end of conversion)
detectionHeigth=ADCL+256*ADCH; // The result is read, and the 10-bit value is calculated/ /IMPORTANT: ADCL should be read first

        adc_flag = 0;
}
```

Once the value from the ADC is obtained a conversion is required to obtaining the value of the distance to the intruder. To begin with, the value is converted to a voltage between 0 and 5 V, equivalent to the analog input. Afterwards, the value is converted to distance. For the LDR it goes from 0 to 10 km and for the potentiometer from -0.5 to 0.5 km.

```
  VLeft = detectionLeft / 1023 * 5;
  d_left = VLeft/5 * 10;

  VFront = detectionFront / 1023 * 5;
  d_front = VFront/5 * 10;

  VRight = detectionRight / 1023 * 5;
  d_rigth = VRight/5 * 10;

  VHeigth = detectionHeigth / 1023 * 5;
  d_heigth = VHeigth/5 * 1 -0.5;
```

The next step is to complete the alert state accounting for the distance and the heigth to the intruder. For doing so, first the distance is checked. If the distance to the nearest target is under 5.3 km, the alert is set to one. If on top of that, the difference in height to the target is 0.259 km above or below, the alert is set to 2 as the maximum danger is detected. These values are based on real procedures employed by planes.

Finally, a proportional control of the tail of the plane accounting for the difference in height is performed. If the plane is above the target the servo will take positive values to try to increase that distance. On the other hand, if the plane is below the target, the servo will go to negative angles.

```
if (d_left < 5.3 || d_rigth < 5.3 || d_front < 5.3 )
{
        alerta = 1;
        PORTC |= (1<<4);
```

```
if (d_heigth < 0.259 && d_heigth > -0.259)
{
        alerta = 2;
        PORTC |= (1<<5);
}
else
{
        PORTC &=~ (1<<5);
}

}

else
{
        alerta = 0;
        PORTC &=~ (1<<4);
        PORTC &=~ (1<<5);
}
```

The other function of the main loop is to perform the scanning and activation process of the rows through the matrix display.

Firstly, PORTD is connected to the upper part of the 8X8 matrix display. Something important to take into account is that these matrices operate using inverse logic. That means that in order to activate a specific row in the matrices, you must set the corresponding bit in Port D to '0' while ensuring all other bits are set to '1',so that only one row is refreshed at a time.Each time the activation process is executed, we move to the next row (scanning process) by means of the following code:

```
if (count > 7) {
   count = 0;
   row = 0xFE;
}
```

The Purpose *of this block is to ensure that after cycling through all 8 rows, the count and row variables are reset to their initial states.*

- ➔ *if (count > 7): Checks if count has gone beyond the number of rows (since there are 8 rows in the array , indexed from 0 to 7).*
- ➔ *count=0: Resets the count variable to 0 to start from the first row of the 8x8 matrix again.*
- ➔ *row=0xFE: Resets the row variable to 0xFE (binary 11111110).By doing this, the scanning process is restarted and the 1st row of the dot matrix is displayed.*

```
PORTD=row;
```

```
row=2*row+1;
```

The Purpose *of this block is firstly to set the PORTD to the value of row to select the current row to be activated on the LED matrix and update the row variable to move the zero bit to the left, effectively activating the next row in the dot matrix.*

- ➔ *2 * row: Shifts all bits in row to the left by one position.*
- ➔ *+ 1: Sets the least significant bit (LSB) to 1 after the shift.*
- ➔ *row=2*row+1: This could also be expressed equivalently as row = ((row << 1) | 1) The OR operation sets the LSB to 1.*

This operation ensures that the next row is switched on by shifting the zero bit to the left and maintaining the 8-bit width.

For example, for the 1st iteration we would have:

First Iteration:

- count = 0, row = 0xFE.
- if (count > 7) is false.
- PORTD = row sets PORTD to 0xFE.
- row = 2 * 0xFE +1 results in 0xFD-> 0b11111101-> which means that the 2nd row will be switched on on the 2nd iteration

```
count = count + 1;
```

The **Purpose** of this instruction is to increment the count variable with every iteration to move to the next row in the 8x8 array of the displayed image and also to establish a limit in order to restart the scanning process of the rows.

Once explained the scanning and refresh process of the rows of the dot matrix, it is relevant to note that the data of the image to be displayed is sent serially to the shift register by using the function **display,** which will be described in detail later. It is also important to consider that the display function is called before count=count+1 instruction, as count value must start at 0.

```
if (alerta > 0)

{

  if (d_left < d_rigth && d_left < d_front)

    display(leftAlert[count]);

  else if (d_front < d_rigth )

    display(frontAlert[count]);

  else
```

```
    display(rigthAlert[count]);
}
else
  display(idle[count]);
```

This code checks for an alert condition and then determines which direction has the closest obstacle (left, front, or right). Based on this determination, it displays the corresponding alert pattern on the LED matrix. If there is no alert, it displays an idle pattern. This helps the system visually indicate where the potential obstacle or alert is coming from.

3.3.4 **Display function**

```
void display(unsigned char data) {
  for (int i = 0; i < 8; i++) {
    if (data & 0x01)
      PORTB |= (1 << 1);
    else
      PORTB &= ~(1 << 1);
    PORTB |= (1 << 2);
    PORTB &= ~(1 << 2);
    data =data/2 ;
  }
  PORTB |= (1 << 3);
  PORTB &= ~(1 << 3);
}
```

The display function sends an 8-bit data pattern,which corresponds to each row of the 8x8 arrays of the variables to the 8x8 LED dot matrix display, controlling the state of each LED in a specific column for the currently activated row. The function processes each bit of the data byte sequentially. For each bit, if the bit is 1, it sets PB1 to 1 to turn on the corresponding LED in the column. If the bit is 0, it clears PB1 to turn off the corresponding LED. After setting or clearing the pin based on the bit's value, the function generates a clock pulse in order to send the data to

the register.It then shifts the data byte to the right to process the next bit. After all 8 bits are processed, it generates a latch pulse by setting and then clearing another pin on PORTB, which updates the display with the new column data.

```
if (data & 0x01)

    PORTB |= (1 << 1);

else

    PORTB &= ~(1 << 1);
```

- data & 0x01: This checks the least significant bit (LSB) of data.
    - If the LSB is 1, it means the corresponding LED in the column should be turned on.
    - If the LSB is 0, it means the corresponding LED in the column should be turned off.
- PORTB |= (1 << 1): If the LSB is 1, this sets the bit 1 of PORTB to 1 (turns on the corresponding LED).
- PORTB &= ~(1 << 1): If the LSB is 0, this clears the bit 1 of PORTB to 0 (turns off the corresponding LED).

```
PORTB |= (1 << 2);

    PORTB &= ~(1 << 2);
```

These lines generate a clock pulse on PB2.

- PORTB |= (1 << 2): Sets PB2 to 1.
- PORTB &= ~(1 << 2): Immediately clears PB2 to 0.

This clock pulse is used to latch the data bit into the register.

```
  data =data/2 ;
```

This shifts the data byte one bit to the right.It is equivalently to data >>= 1;

The next bit in data is now the LSB, ready to be processed in the next iteration of the loop.

```
PORTB |= (1 << 3);

  PORTB &= ~(1 << 3);
```

After the for loop finishes, which means that the 8-bit data corresponding to a row of the array has been processed,these lines generate a pulse in order to send the byte to the output of the register,effectively displaying the new column data on the LED matrix of the currently displayed row.

Finally, the timer interrupt is presented. This interrupt is used to create the PWM that goes into the servomotor. It alternates between a timer of 20 ms where the signal is set to low and a variable one between 1 and 2 ms where the signal is up. The period is defined by the control implemented in the loop of the code.

```
ISR(TIMER1_OVF_vect) // Interrupción del TIMER1
{
        if (PORTB & (1<<PB0))
        {
                PORTB &= ~(1 << PB0); // PB0 a '0'
                TCNT1H = 25536/256; // Recarga para temporizador de 20 ms: 65536-(20E-3*16E6)/8
                TCNT1L = 25536%256;
                adc_flag = 1;
        }
        else
        {
                PORTB |= (1 << PB0); // PB0 a '1'
                TCNT1H = servo/256;
                TCNT1L = servo%256;
        }
}
```

4. Simulation results

After extensive testing, the code has been found to work as expected. A couple of examples can be found hereunder:

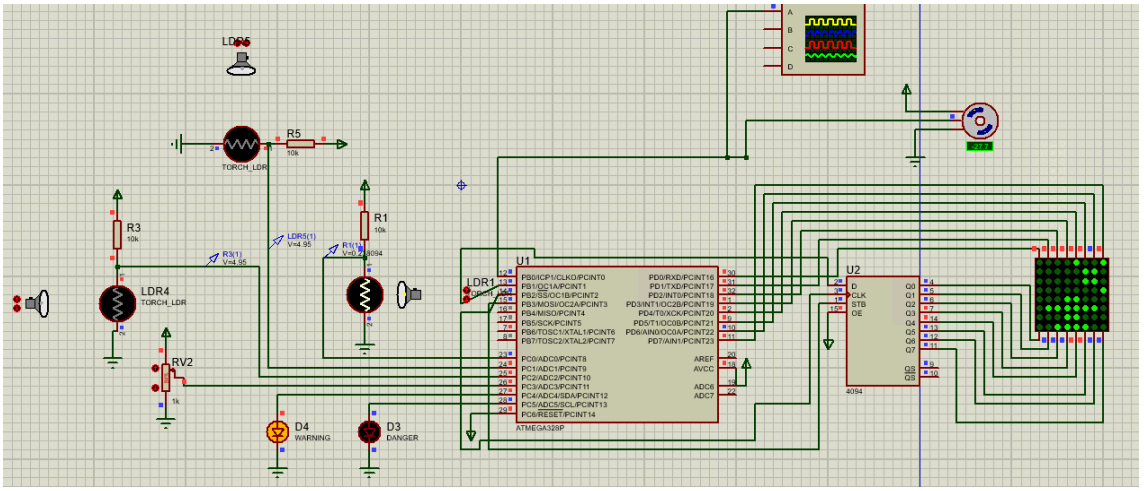


Figure 2.0: Complete system simulation 1

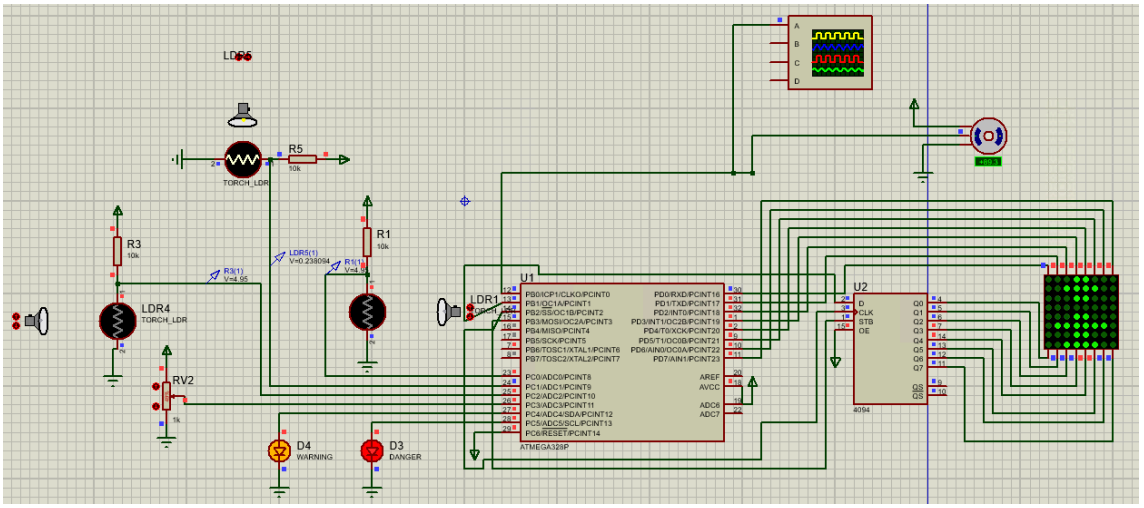


Figure 2.1: Complete system simulation 2

5. Conclusion

This project has greatly benefited us in two significant ways. Firstly, it required extensive research and information gathering related to the aeronautics sector. Through this exploration, we gained a deeper understanding of the Traffic Collision Avoidance System (TCAS), which is crucial for ensuring the safety of aircraft by preventing mid-air collisions. We learned about the different types of TCAS systems and how they detect and resolve potential collision threats. Furthermore, it has sparked a heightened interest among the participants in aerospace safety systems.

Secondly, it is known the importance of applying into practice the theoretical knowledge that a student learns in class. That is why we reckon that the project has significantly reinforced and expanded our understanding from the electronic engineering theory classes. By delving into the inner workings of the microcontroller , and through modifying the code to ensure proper system functionality, we were able to practically apply theoretical concepts acquired. Furthermore, it has equipped us with essential skills for our future professional careers as mastery in C programming and the implementation of digital systems are critical competencies in the aerospace industry.