Project in DIGITAL INSTRUMENTATION

GROUP 3

23.12.2021

Authors:

Kim R. H. Christensen
s181554

Jørgen D. Greve
s181519

DTU - ELEKTRO
30021 - Digital Instrumentation

Contents

1 Problem definition 3

2 Design 3
2.1 Step motor 3
2.2 LSM9DS1 4
2.3 HC-SR04 4
2.4 Fan 4

3 Implementation 4
3.1 Flow chart 5
3.2 System overview 6
3.3 Window handle 6
3.4 Thermometer 8
3.5 Distance sensor 8

4 Test 8

5 System demonstration 9

6 Conclusion 9

A Appendix 10
A.1 main.c code 10

1 Problem definition

The problem we would like to solve in this project is to control the indoor climate in a greenhouse by making an automated ventilation system. This should be achieved by using the components used in the course, here the HC-SR04, LSM9DS1 and step motor will be the key components.

2 Design

To utilize some of the components we have worked with in the course we decided to use the stepper motor for window manipulation, the LSM9DS1 to get temperature readings, the HC-SR04 to detect if the window is open and how much and the fan for accelerating the cooling of the green house. All the components would then be tied together with the brain of the system, the ST NUCLEO prototyping board featuring an ARM STM32 microcontroller.
The system should measure the green house temperature and based on that open or close the window. If opening the window is not sufficient, then the fan should kick in and help with getting fresh outside air into, and warm air out of, the green house.

2.1 Step motor

The step motor is driven of electrical coils that have to be controlled by a micro controller that will send a certain amount of impulses to the electrical coils in order to get the motor turning a series of steps. The step motor issued for this course were an unipolar step motor which means that it was not necessary to consider alternating currents as the bipolar step motor needs.

In order to drive the step motor two control methods were considered, the full drive and the half drive. As seen in the tables below the order of pulses for each of the drive types are shown:

Full drive				
Step	Coil 1	Coil 2	Coil 3	Coil 4
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

For full drive mode two coils are activated at a time and by that the torque is the greatest, however the power consumption will be higher when using this mode.

Half drive				
Step	Coil 1	Coil 2	Coil 3	Coil 4
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

For half drive, the coils are alternating between activating one coil and two coils. This is used to increase resolution, however the torque is less than full drive when only one coil is activated.
Of the two types it was chosen to use half drive in order to conserve energy as well as there is no need for a great amount of torque for this project.
For controlling the speed of the motor, a timer will be used to create a configurable delay between each step of the motor.

In order to drive the motor an ULN2003A will be utilized to deliver the necessary amount of voltage and current, hence the micro controller will not be able to drive it directly from the I/O ports.

2.2 LSM9DS1

The LSM9DS1 has a temperature sensor built in so this would be used for the monitoring the temperature of the greenhouse.
At page 14 in the datasheet, it is given that the temperature sensor would have a reading of 0 at 25°C, however this is a typical reading and the given Vdd is 2.2V which means that some calibration of the reading might be necessary. Furthermore, at page 38 a description of the temperature register addresses to read are found. Here register address $0x15$ is the OUT_TEMP_L, the lowest 8 bits and at address $0x16$ is the OUT_TEMP_H, which is the highest 8 bits of the combined 16 bit temperature reading.
In order to operate this sensor SPI communication is necessary to be implemented.

2.3 HC-SR04

For determining whether the window is open or closed the HC-SR04 will be used. This sensor functions by getting a pulse, of minimum $10\mu S$, on the trigger pin, where after the Echo pin will return a high pulse representing the distance to the target. Here a timer on the micro controller should be utilized to control the length of the trigger pulse as well as measure the length of the returning pulse of the Echo pin.
Further it is described that the formula for the range is, where "Length" is the length of the Echo pulse and "VOS" is velocity of sound:

$$Range = \frac{Length \cdot VOF}{2} \Rightarrow \frac{Length \cdot 340\frac{m}{s}}{2}$$

The length of the Echo pulse represents the distance from the sensor to the target and back, why it has to be divided by 2. Furthermore, in order to represent the distance in centimeters the VOS can be converted to:

$$Range = \frac{Length \cdot 340\frac{m}{s}}{2} \Rightarrow \frac{Length \cdot 0.034\frac{cm}{\mu s}}{2}$$

With these calculations it should be possible to determine the state of the window.

2.4 Fan

For the acceleration of cooling the greenhouse a 12V fan will be used. This will as well be connected to the ULN2003A and be controlled by an I/O port of the micro controller.

3 Implementation

This project was assembled by taking each major component and build a program for them. When they worked as expected they would be combined one component at a time in order to ease the troubleshooting process. By doing this it was possible to make fast progress and the assembly of the final program went as expected. However, building the setup into the prototype green house revealed a number of challenges, for example connecting more jumperwires with each other in order to install the LSM9DS1 inside the green house resulted in unstable readings of temperature, so this was kept with as short wire distance to the micro controller as possible.

3.1 Flow chart

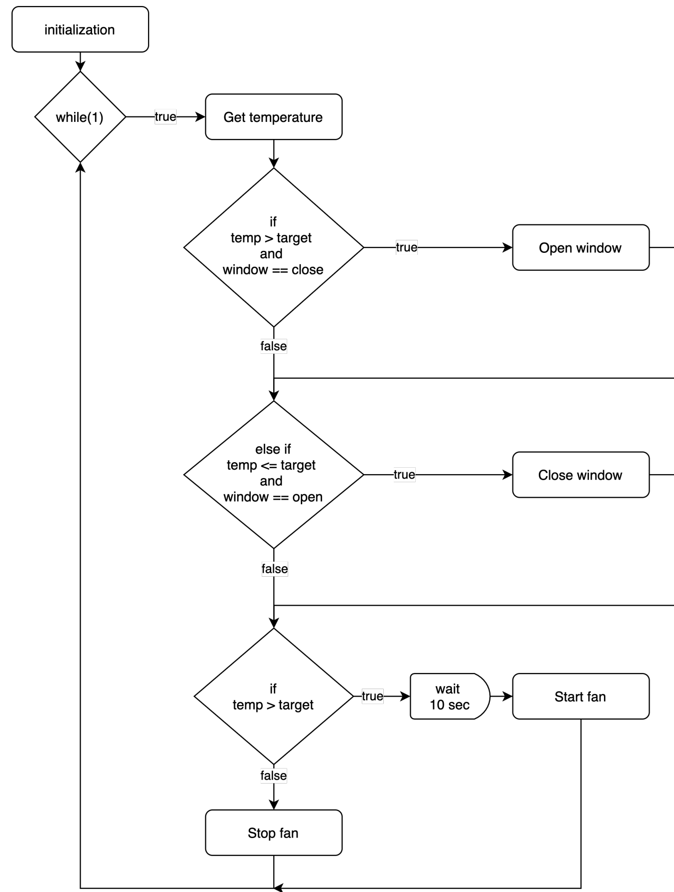

Figure 1: System flowchart

In figure1 a flow chart of the system is shown. First the initialization where UART (For the terminal), SPI, GPIOs, Timer and the ADC are initialized. Moreover, most of the variables used in the program are created here. Though three global variables are needed, these are for the timer and to make sure that the fan will activate when the temperature is above the target temperature.
In the while loop the position of the window is determined by shooting a distance to it with the HC-SR04. After this, the temperature is read. The next two if statements compare the current state of the window and the last temperature reading and the open or close the window compared is needed.
Lastly, an if statement determines when the fan should start. In the project it is programmed to start if the temperature has been above the target temperature for more than 10 seconds and keep cooling until the temperature has dropped below the target temperature.

3.2 System overview

Pin selection			
PIN	I/O	Component	Type
PA05	Out	Step motor	Coil 1
PA06	Out	Step motor	Coil 2
PA07	Out	Step motor	Coil 3
PB01	Out	Step motor	Coil 4
PB08	Out	LSM9DS1	CS
PC10	AF	LSM9DS1	CLK
PC11	AF	LSM9DS1	MOSI
PC12	AF	LSM9DS1	MISO
3.3V	-	LSM9DS1	Vdd
PC06	In	HC-SR04	Echo
PC08	Out	HC-SR04	Trigger
5V	-	HC-SR04	Vcc
PB11	Out	Fan	Fan control

In the table above an overview of the pins used for this project is shown. Be advised that the LSM9DS1s supply voltage is ranged between 1.9V-3.6V whereas the HC-SR04s supply voltage is rated at 5V. Usually sensors will work with a supply voltage of 3.3V but in the latter case 5V is required for it to function.

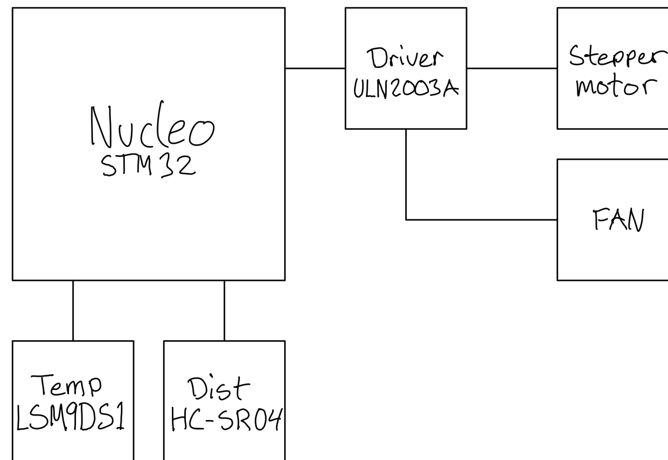

Figure 2: System overview

In figure 2 a simple schematic of the system is seen. The LSM9DS1 will communicate with the micro controller using SPI, and the HC-SR04 measurements is converted by timing the length of the high pulse. For motor and fan control an ULN2003A driver is used, which is supplied by an external power source delivering 12V as needed for the motor and fan to function as intended.

3.3 Window handle

Creating the function for controlling the motor was one of the trickiest to create. First, a function was made that needed inputs regarding which direction the motor should drive, the distance it should match, and the velocity the motor should run at. However, this function could not take into account what happened if the motor opened the window too much, and could not adjust the speed in order to slow down when getting close to the determined distance.

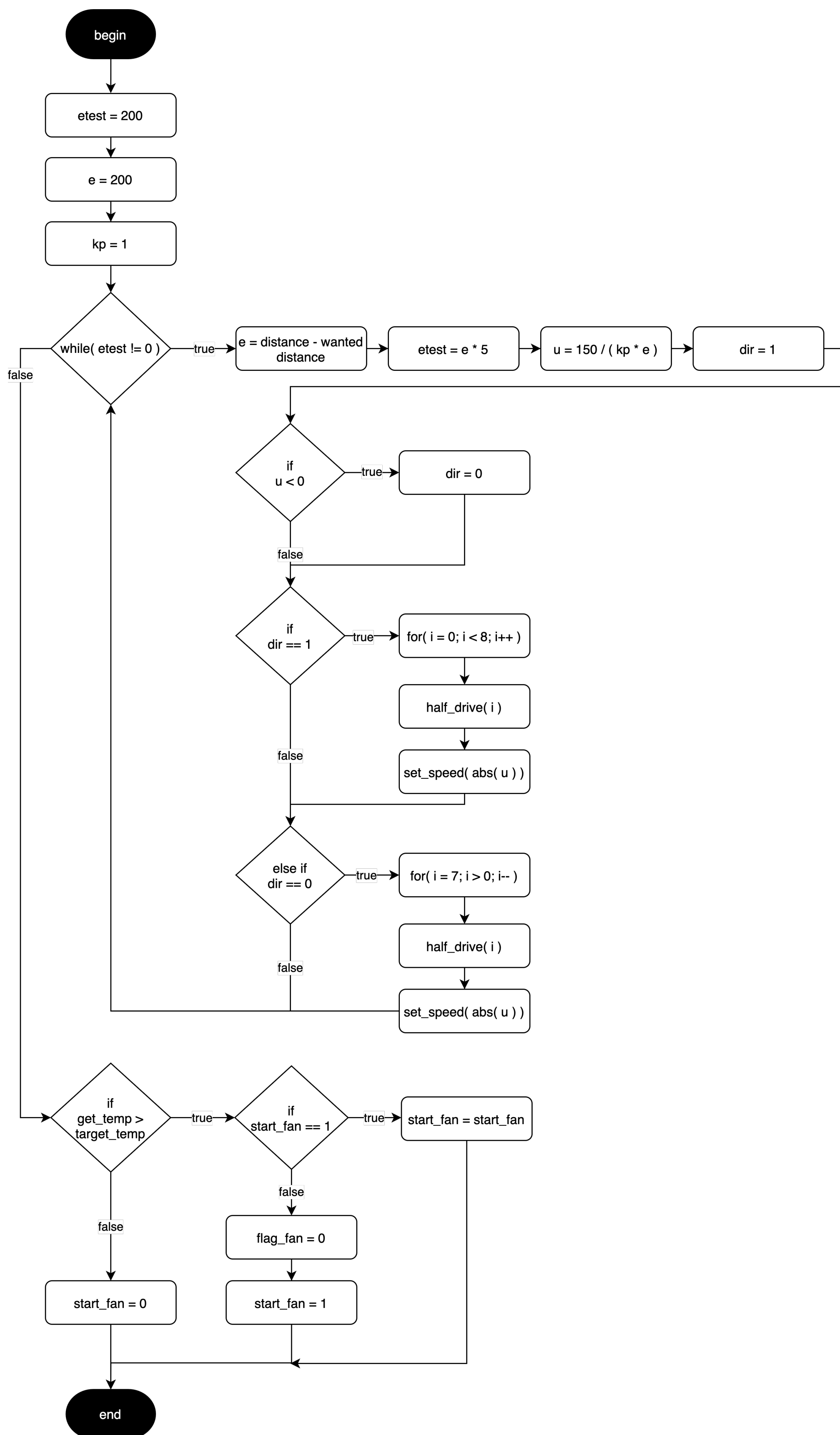

Figure 3: Window handle flowchart

In figure 3 a flow chart of the function is seen. The while loop will keep running until the variable etest is zero. Etest was implemented since the original error variable (e) is a float variable and there was issues with it never reaching zero and keeping adjusting the position of the window back and forward. However, using the etest variable that is the e variable multiplied with 5, a reasonable result was seen. So by adding:

$$velocity = \frac{150}{kp \cdot error}$$

The velocity would decrease as the error distance decreases as well. Next the dir = 1 represents the direction for the motor to drive. The first if statement test whether u is positive or negative - if negative the motor will change dir = 0 and go reverse. The variable kp was implemented in order to adjust the gain of the controller, though this was not needed since the function worked as intended, so kp = 1. The if and else if statements tests what value dir has and decide the direction of the motor. In these two statements, the velocity is regulated by the set_speed function in order to decrease the revolution of the motor as the error distance decrease.

By doing this, the system is also able to correct the window position if the motor opens the window to much, or if a wind gust opens the window more than the desired position.

When the window is positioned as commanded, the while loop will break. Before returning from the function, there is an if statement with another nested if statement. This determines whether the temperature is above the target temperature - if yes, another statement checks if the start_fan = 1. If this is true start_fan will keep the same value, and if not true the timer for flag_fan = 0 in order to reset the 10 second timer and start_fan = 1.
If the first if statement is not true, start_fan = 0 since the temperature is not at a level where fan cooling is necessary. These if statements were needed since the flag_fan would otherwise reset each time the function was called. So if the window was open and had to re-adjust the position while the fan was already running the timer would reset and the fan would stop and continue after a new 10 second delay.

3.4 Thermometer

As described in section 2.2 the temperature sensor on the LSM9DS1 would read out 0 at 25°C with a Vdd of 2.2V. When calibrating the thermometer for being supplied with 3.3V from the micro controller it was discovered that the offset should be 18 and not 25 as expected. However, after implementing this and figuring out how to interpret the readings, stable reading was achieved.

3.5 Distance sensor

For the HC-SR04, a function was created that would set the trigger pin high for $10\mu s$ and afterwards a while loop would hold the function until the Echo pin is not low. When this state changes the time of the high pulse is recorded and the distance is converted with the formula from section 2.3.

4 Test

When testing the prototype some adjustments had to be made, for example the position of the HC-SR04 had to be positioned some distance away from the window when it was closed due to the sensors minimum range of 2 cm, however after this was corrected some minor adjustments of the distance for the open and closed states were an easy fix.

Further finding the span of velocity of where the step motor would run smooth took some testing, but by trial and error it was discovered that the set_speed functions maximum velocity was 10 and minimum was approximately 150. Staying between these two values is controlled in the handle_window by setting a minimum and maximum value for the motor with two if statements.

The controlling of when the fan had to turn on and off also gave rise to minor issues. At start the fan timer would simply reset every time the temperature was above the target temperature, however sometimes the window will re-adjust after opening the window, resulting in the function to be called again hence the timer would reset again.

This was overcome by having a global control variable that had to be true before the counter flag for the fan would reset.

5 System demonstration

We have made a short video demonstration of the system, which can be viewed on Youtube through this link: https://youtu.be/YMrg52bXoww

6 Conclusion

When testing the system it worked as expected. Though it was necessary to add a factor to the handle_window function in order to avoid the function not opening the window to the correct distance and afterwards have to run the function multiple times where the steps from the motor finally hit the correct distance. This is possible due to the HC-SR04s inaccuracy that sometimes varies when measuring the distance. By adding 0.5 to the defined window_open variable this issue was overcome and the function worked as it should.

For the timer there was an issue with getting the correct sequence, so when for example $10\mu s$ was needed the timer would only count to $5\mu s$. The wrong setting in the timer initialization was not found, but the problem was overcome by multiplying the timed period, this is seen the get_distance function.

For the LSM9DS1 an attempt to install the sensor inside the prototype was made. However, by adding multiple jumpercables together, in order to having it to reach the green house, the temperature readings were rather jumpy and the issue stopped when shortening the wires as much as possible. It would likely be possible to fix the problem by using longer cables to minimize the number of connecting cables, however this is not tested, since the overall idea of the system was still possible to show with the sensor outside the prototype.

Furthermore, the project has been designed to fully open the window when above a certain temperature and fully close when below this temperature, so if the temperature in the green house is fluctuating around this target temperature the system will open and close the window until the temperature changes to a higher or lower value. However, this was not considered prior to the demonstration to the advisor and there has not been enough time to re-think the functions of the program.

Finally, the whole program consists of a lot of controlling if statements. This is because by using them for controlling the different parameters gave a rather simple but effective way to check and control the different variables.

All in all, this project has been a great challenge that has resulted in a great understanding in how the STM32 micro controller works and how to implement and work with multiple sensors continuously during a program. As described in the problem definition the project requirements have been met, except the goal number 2, which was humidity monitoring. This was not implemented since it was decided to use the issued LSM9DS1 for temperature measurements instead of another sensor, like the DHT-11 or DHT-22. But the main concept of monitoring and reacting to the changes of the climate inside a green house has been successful.

A Appendix

A.1 main.c code

```
#include "stm32f30x_conf.h"
#include "30021_io.h" // Input/output library for this course
#include "lcd.h"                  // LCD driver
#include <string.h>               // memset()
#include <stdlib.h>
#define close 0
#define open 1
#define unknown_L 2
#define unknown_H 3
#define window_open 5.0
#define window_close 3.0
#define target_temp 24

void GPIO_init();
void timer_Init();
void TIM2_IRQHandler(void);
void delay_us(int delayUs);
void delay_ms(int delayMs);
void delay_S(int delayS);
void half_drive(int step);
void full_drive(int step);
void set_speed(int speed);
float get_distance();
void writeRegister(int address, int data);
void init_SPI();
int readRegister(int address);
void handle_window(float distance);
int get_temp();



static int flag = 0;
static int flag_fan = 0;
static int start_fan = 0;


int main(void)
{
    init_usb_uart( 9600 ); // Initialize USB serial at 9600 baud
    GPIO_init();
    timer_Init();
    init_SPI();



    float distance = 0;
    int distInt = 0;
    float distBuf = 0;
    int distFrac = 0;
    int temp = 0;
    int window_state = 0;
    char test[512];
```

```
    memset(test,0x00,512);
    uint8_t fbuffer[513];       // Creates a buffer array
    memset(fbuffer,0x00,512);   // Sets each element of the buffer to 0x00

    writeRegister(0x1F,0b00111000); //setup XL CTRL REG 5
    writeRegister(0x20,0b11000000); //setup XL CTRL REG 6
    writeRegister(0x21,0b10100110); //setup XL CTRL REG 7
    writeRegister(0x10,0b11000000); //setup G CTRL REG1

    temp = get_temp();
    delay_S(5); //hold 1 second to get stable temperature readings



  while(1)
  {


        distance = get_distance(); // Measure distance to the  window

        // Used to convert the float distance to two integers representing
        // the integer and the decimal of the distance for printing to terminal
        distInt = distance;
        distBuf = distance - distInt;
        distFrac = (distBuf * 100);
        temp = get_temp(); //Get the temperature

        printf("Distance to target: %d.%02d cm\n", distInt, distFrac);
        printf("Temp: %d\n",temp);









        // Used to determine the position of the window
        if (distance <= 3.4 && temp <= target_temp)
            {
                window_state = close;
            }

        else if ((distance >= 6) && (temp > target_temp))
            {
                window_state = unknown_H;
            }

        else if ((distance >= 5) && (temp > target_temp))
            {
                window_state = open;
            }

        else
            {
                window_state = unknown_L;
```

```
    }


//Prints window state to terminal
switch (window_state)
    {
        case close:
            printf("Window closed\n");
            break;

        case open:
            printf("Window open\n");
            break;

        default:

            break;

    }





// Determines whether the window should be opened of closed
if(temp > target_temp && (window_state == close || window_state == unknown_L ||
   window_state == unknown_H))
    {
        handle_window(window_open + 0.5);
    }



else if(temp <= target_temp && (window_state == open || window_state == unknown_L ||
        window_state == unknown_H))
    {
        handle_window(window_close);
    }

// Start fan after 10 seconds if the window is still open
// and temperature is above target temp

if (flag_fan > 5000000 && start_fan == 1)
    {
        GPIO_WriteBit(GPIOB, GPIO_Pin_11, Bit_SET);
    }

else
    {
        GPIO_WriteBit(GPIOB, GPIO_Pin_11, Bit_RESET);
    }
```

```
        delay_S(10);









  }
}

int get_temp()
{
    int16_t offset = 14;
    int temp_L = 0;
    int16_t temp_H = 0;
    uint16_t temp_raw = 0;

    temp_H = readRegister(0x16);
    temp_L = readRegister(0x15);
    temp_raw = ((((int16_t)temp_H << 8) | temp_L) >> 8);
    return(offset + temp_raw/16);

}

void handle_window(float distance) // 1 = open window, 0 = close window
{


    float kp = 1;
    float e = 200;
    int etest = 200;
    int u = 0;
    int dir = 0;


while ( etest != 0)
{
    e = get_distance() - distance; //Compare the current distance to the wanted distance
    etest = e * 5; //Used for the while loop to break when target distance is reached










    u = 150/(kp * e); //Used to regulate the speed according to the distance to target


    dir = 1; // Direction if the variable u < 0 the motor will close the window and u > 0 the
             // motor will open the window
```

```
    if (u < 0) // if the u if negative the motor will reverse
        {
          dir = 0;
        }

    //Sets the speed between 10 and 150. 10 is the fastest speed
    if (abs(u) > 150)
        {
            u = 150;
        }

    else if (abs(u) < 10)
        {
            u = 10;
        }




    if (dir == 1)
        {

            for(int i = 0; i < 8; i++)
                {

                    half_drive(i);
                    set_speed(abs(u));
                }




        }

    else if (dir == 0)
        {
            for(int i = 7; i >= 0; i--)
                {

                    half_drive(i);
                    set_speed(abs(u));
                }

        }



}


    if(get_temp() > target_temp)
        {
```

```
            if(start_fan == 1)
                {
                    start_fan = start_fan;
                }

            else
                {
                    flag_fan = 0;
                    start_fan = 1;
                }

        }

    else
        {
            start_fan = 0;
        }
}


void init_SPI()
{

RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI3, ENABLE);

RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC,ENABLE); // Enable clock for GPIO Port C
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB,ENABLE); // Enable clock for GPIO Port B


    GPIO_InitTypeDef GPIO_InitStructAll;

    //SPI CLK
    GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
    GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_AF; // Set as Alternating Function
    GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_NOPULL; // Set as No Pull
    GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_10; // Set so the configuration is on PinC10
    GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_10MHz;

    GPIO_Init(GPIOC, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource10,GPIO_AF_6);
    //SPI MOSI
    GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
    GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_AF; // Set as Alternating Function
    GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_NOPULL; // Set as No Pull
    GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_11; // Set so the configuration is on PinC11
    GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_10MHz;

    GPIO_Init(GPIOC, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource11,GPIO_AF_6);


    //SPI CS
    GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
    GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as Output
    GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as Pull Down
```

```
    GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_8; // Set so the configuration is on PinB8
    GPIO_Init(GPIOB, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen

    //SPI MISO

    GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
    GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_AF; // Set as Alternating Function
    GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_NOPULL; // Set as No Pull
    GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_12; // Set so the configuration is on PinC12
    GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_10MHz;

    GPIO_Init(GPIOC, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource12,GPIO_AF_6);

    GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
    GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_IN; // Set as Input
    GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as Pull Down
    GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_4; // Set so the configuration is on PinC4
    GPIO_Init(GPIOC, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen

    SPI_InitTypeDef SPI_initstruct;
    SPI_StructInit(&SPI_initstruct);
    SPI_initstruct.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    SPI_initstruct.SPI_Mode = SPI_Mode_Master;
    SPI_initstruct.SPI_CPOL = SPI_CPOL_High;
    SPI_initstruct.SPI_CPHA = SPI_CPHA_2Edge;
    SPI_initstruct.SPI_BaudRatePrescaler = 4;
    SPI_initstruct.SPI_NSS = SPI_NSS_Soft;
    SPI_initstruct.SPI_FirstBit = SPI_FirstBit_MSB;
    SPI_initstruct.SPI_CRCPolynomial = 7;
    SPI_initstruct.SPI_DataSize = SPI_DataSize_8b;

    SPI_Init(SPI3, &SPI_initstruct);

    SPI_RxFIFOThresholdConfig(SPI3, SPI_RxFIFOThreshold_QF);

    SPI_Cmd(SPI3, ENABLE);


}


void writeRegister(int address, int data)
{

    GPIO_WriteBit(GPIOB, GPIO_Pin_8, Bit_RESET); // Set CS Low
    SPI_SendData8(SPI3, address);
    SPI_SendData8(SPI3, data);
    GPIO_WriteBit(GPIOB, GPIO_Pin_8, Bit_SET); // Set CS High
}

int readRegister(int address)
{

    uint16_t toRead = 0;
    address |= 0x80;
```

```
    GPIO_WriteBit(GPIOB, GPIO_Pin_8, Bit_RESET); //Set CS Low


    while(SPI_I2S_GetFlagStatus(SPI3, SPI_I2S_FLAG_TXE) != SET);

    SPI_SendData8(SPI3, address);
    SPI_SendData8(SPI3, 0x00);

    while(SPI_I2S_GetFlagStatus(SPI3, SPI_I2S_FLAG_TXE) != SET);
    while(SPI_I2S_GetFlagStatus(SPI3, SPI_I2S_FLAG_RXNE) != SET);
    toRead = SPI_ReceiveData8(SPI3);
    toRead = SPI_ReceiveData8(SPI3);

    GPIO_WriteBit(GPIOB, GPIO_Pin_8, Bit_SET); // Set CS High






  return toRead;
}

void set_speed(int speed) // Regulate the speed by adjusting the delay between steps
{
    delay_ms(speed);
}
void full_drive(int step)
{

        switch(step)
      {
        case 0:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_SET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_SET);
            break;
        case 1:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_SET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_SET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_RESET);
            break;

        case 2:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_SET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_SET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_RESET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_RESET);
            break;

        case 3:
```

```
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_SET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_RESET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_SET);
            break;


         default:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_RESET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_RESET);
            break;




    }




}

void half_drive(int step)
{
    switch(step)
      {
        case 0:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_RESET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_SET);
            break;
        case 1:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_SET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_SET);
            break;

        case 2:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_SET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_RESET);
            break;

        case 3:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_SET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_SET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_RESET);
            break;
        case 4:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_SET);
```

```
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_RESET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_RESET);
            break;

        case 5:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_SET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_SET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_RESET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_RESET);
            break;

        case 6:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_SET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_RESET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_RESET);
            break;

         case 7:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_SET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_RESET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_SET);
            break;

         default:
            GPIO_WriteBit(GPIOA, GPIO_Pin_5, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_6, Bit_RESET);
            GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_RESET);
            GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_RESET);
            break;



      }

}

float get_distance()
{
            float duration = 0;
            float distance = 0;


            duration = 0;

            GPIOC->BSRR = GPIO_Pin_8;
            flag = 0;
            while(flag < 5);
            GPIOC->BRR = GPIO_Pin_8;
            while(GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_6) == 0);
            flag = 0;
            while(GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_6) == 1);

            if (flag > 18500) // if no object detected discard reading
            {
```

```
                flag = 0;
            }
            else
            {

            }




        duration = flag*2;

        distance = duration*0.034/2;




        return distance;

}

void delay_us(int delayUs)
{

    flag = 0;
    while(flag < delayUs);
}

void delay_ms(int delayMs)
{
    delayMs = delayMs * 100;
    flag = 0;
    while(flag < delayMs);
}

void delay_S(int delayS)
{
    delayS = delayS * 100000;
    flag = 0;
    while(flag < delayS);
}

void TIM2_IRQHandler(void)
    {
        flag++;
        flag_fan++;
        TIM_ClearITPendingBit(TIM2,TIM_IT_Update); // Clear interrupt bit
    }

void timer_Init()
{
    // Timer
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC,ENABLE); // Enable clock for GPIO Port C
```

```
    TIM_TimeBaseInitTypeDef TIM_InitStructure;
    TIM_InitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_InitStructure.TIM_Period = 6138;
    TIM_InitStructure.TIM_Prescaler = 100;
    TIM_TimeBaseStructInit(&TIM_InitStructure);



    TIM_TimeBaseInit(TIM2,&TIM_InitStructure);
    // NVIC for timer
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_Init(&NVIC_InitStructure);



    TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);



    TIM_Cmd(TIM2,ENABLE);



    RCC->APB1ENR |= RCC_APB1Periph_TIM2; // Enable clock line to timer 2
    TIM2->CR1 = TIM_CR1_CKD_1; // Configure timer 2
    TIM2->ARR = 63; // Set reload value
    TIM2->PSC = 1; // Set prescale value
    TIM2->DIER |= 0x0001; // Enable timer 2 interrupts
    TIM2->CR1 = 1;


    NVIC_SetPriority(TIM2_IRQn, 0); // Set interrupt priority interrupts
    NVIC_EnableIRQ(TIM2_IRQn); // Enable interrupt




}

void GPIO_init()
{
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE); // Enable clock for GPIO Port A

    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB,ENABLE); // Enable clock for GPIO Port B

    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC,ENABLE); // Enable clock for GPIO Port C

    GPIO_InitTypeDef GPIO_InitStructAll;

    //Motor coil 1
```

```
    GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
    GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
    GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
    GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_5; // Set so the configuration is on pin 4
    GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen
    //Motor coil 2
    GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
    GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
    GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
    GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_6; // Set so the configuration is on pin 4
    GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen
    //Motor coil 3
    GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
    GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
    GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
    GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_7; // Set so the configuration is on pin 4
    GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen
    //Motor coil 4
    GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
    GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
    GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
    GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_1; // Set so the configuration is on pin 4
    GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen
    //Fan control pin
    GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
    GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
    GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
    GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_11; // Set so the configuration is on pin 4
    GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen

    //Trigger pin HC-SR04
    GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
    GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
    GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
    GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_8; // Set so the configuration is on pin 4
    GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen

    //Echo pin HC-SR04
    GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
    GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_IN; // Set as output
    GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_UP; // Set as pull down
    GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_6; // Set so the configuration is on pin 6
    GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen
}
```