Automatic Pet Feeder

ENG5220: Real Time Embedded Programming

Team 21: Peihan Song(2703170S), Yifei Wang(2554706W),

Shujing Yan(2658203Y), Xudong Fu(2528996F)

I. Team and Collaboration

Name	GUID	Responsibility	Description
Peihan Song	2703170S	Software	Complete program logic, main method
Shujing Yan	2658203Y	Hardware	Hardware configuration, adjust equipment, report
Xudong Fu	2528996F	Software	Unit test, social media, assists in programming
Yifei Wang	2554706W	Mechanical Design	Build the device, presentation

Github link: https://github.com/Shujing106/AutoPetFeeder

Video link: https://youtu.be/EHu-TGHlu9M

II. Introduction

This project uses raspberry PI to make an automatic feeder, real-time monitoring of pet movement and food weight. Realize intelligent feeding, protect the health of pets.

Design ideas:

Ultrasonic sensors detect the distance from the pet, and the feeder starts working when the pet approaches. The weight sensor detects the weight of the food in the bowl. When the weight of food less than 10g, start the motor to make the baffle turn forward to replenish food. When the weight of food more than 50g, the motor rotates backward to stop adding. In addition, we added the cooldown time in this project. There will be a 4h cooldown time after each addition of food. Meanwhile, the machine can only add food three times within 24 hours.

III. Projects Components

3.1 Configuration requirement

- Raspberry Pi 3B+
- C++

3.2 Equipment

Raspberry Pi 3B+

Ultrasonic sensor (HC-SR04)

Weight sensor (HX711)

Step motor (28BYJ-48)

3.3 Circuit diagram

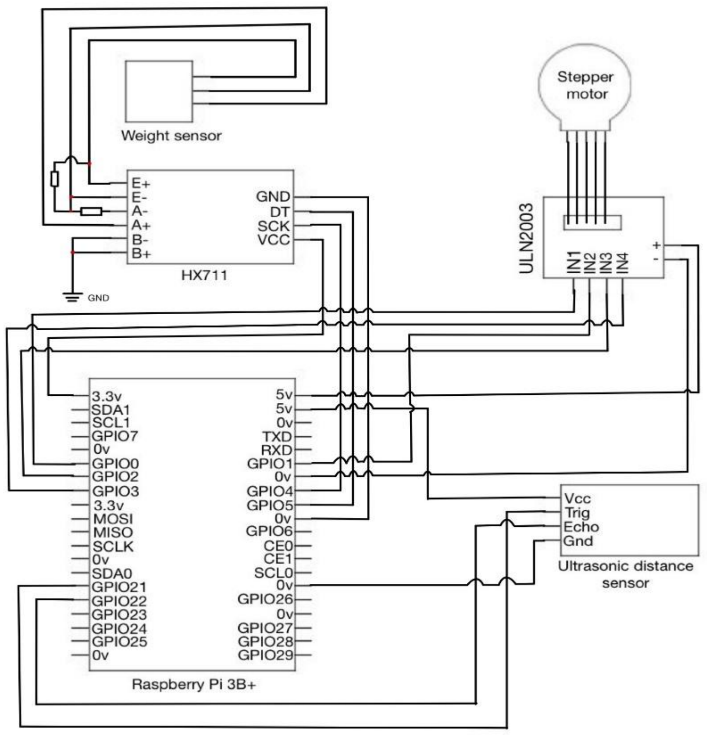


Figure 1. circuit diagram

3.4 Principle of Sensor

3.4.1 Step motor

The stepper motor is an open loop control element which converts the electric pulse signal into angular displacement or linear displacement. The speed and stop position of the motor depend on the frequency and number of pulse signals. Add a pulse signal to the motor and the motor will turn a step angle.

3.4.2 Ultrasonic sensor

Trig: Receives control signals from raspberry PI   Echo: sends ranging results to raspberry PI

The Raspberry PI sends a pulse to the Trig that lasts 10 microseconds. Hc-SR04 receives the 10 microseconds pulse signal sent by the Raspberry PI, sends eight 40khz square waves, then sets Echo to high and prepares to receive the returned ultrasonic waves. When HC-SR04 receives the returned ultrasonic wave, Echo will be set to low. The duration of the high level of Echo output is the time from transmission to return of the ultrasonic wave. The distance can be measured by recording the duration of high level.

3.4.3 Weight sensor

The principle of the weighing sensor is that the metal resistance wire is stretched and thinned under the action of tension, so that the resistance increases, that is, the strain borne by the metal resistance changes. The weighing sensor is used to convert the weight signal or pressure signal into electric signal.

IV. Software

4.1 Unit test

4.1.1 Ultrasonic

In *ult.cpp*, the Trig port receives a high level of 10 microseconds, producing a pulse. Then set the level low.

```
digitalWrite(Trig, HIGH);
delayMicroseconds(10);
digitalWrite(Trig, LOW);
```

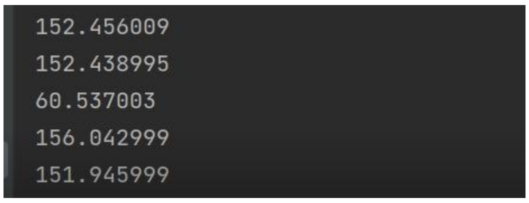
Record the duration of the high level on the Echo port.

```
while(!(digitalRead(Echo) == 1));
gettimeofday(&tv1, NULL);
while(!(digitalRead(Echo) == 0));
gettimeofday(&tv2, NULL);
```

Record the start time and stop time, calculate the distance.

```
start = tv1.tv_sec * 1000000 + tv1.tv_usec;
stop = tv2.tv_sec * 1000000 + tv2.tv_usec;
dis = (float)(stop - start) / 1000000 * 34000 / 2;
```

Figure 2 shows the output of ultrasonic sensor test, constantly update real-time distance data.



```
152.456009
152.438995
60.537003
156.042999
151.945999
```

Figure 2. Ultrasonic sensor test output

4.1.2 Weight sensor

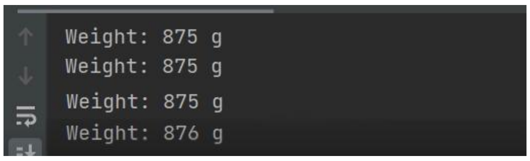
In *weight.cpp*, perform AD conversion. Delay an appropriate time to prevent inaccurate output due to unfinished AD conversion.

```
digitalWrite(value->SCK,LOW);
while(digitalRead(value->SCK));
while(digitalRead(value->SDA));
delay(100);
```

Loop 24 times to read the data. Value is multiplied by 2 to convert binary to decimal: *value->value = value->value*2.*

```
for(i=0;i<24;i++){
    digitalWrite(value->SCK,HIGH);
    while(0 == digitalRead(value->SCK))delay(1000);
    value->value = value->value*2;
    digitalWrite(value->SCK,LOW);
    while(digitalRead(value->SCK));
    if(digitalRead(value->SDA))
        value->value = value->value + 1;
}
```

Figure 3 shows the output of weight sensor test, constantly update real-time weight data.



```
Weight: 875 g
Weight: 875 g
Weight: 875 g
Weight: 876 g
```

Figure 3. Weight test

4.1.3 Step motor

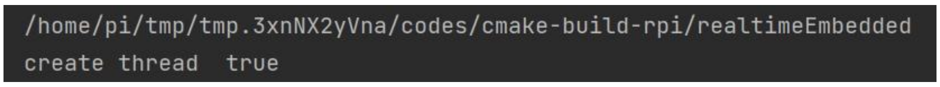
The stepper motor has two modes of forward turn and backward. Changing the order of input high and low levels can change the direction of rotation. *int t* Controls the speed of the stepper motor, and *int steps* controls the rotation angle.

```
void motor::forward(int t, int steps)
{
     for(i = 0; i < steps; i++){
         setStep(1, 0, 0, 0);
         setStep(0, 1, 0, 0);
         setStep(0, 0, 1, 0);
         setStep(0, 0, 0, 1);
    }
}
void motor::backward(int t, int steps)
{
      for(i = 0; i < steps; i++){
         setStep(0, 0, 0, 1);
         setStep(0, 0, 1, 0);
         setStep(0, 1, 0, 0);
         setStep(1, 0, 0, 0);
    }
}
```

4.2 Feeder

In *main.cpp*, create a time thread. For testing purposes, we set the time interval to minutes. Substituting *int time_ctl=60*60*1* can change the time unit to hours.

```
void time_thread_run(void *ptr)
{    //int time_ctl=60*60*1;     //one hour
       int time_ctl=60*1;   // one minute
    while(1){
        sleep(time_ctl);
        hour++;
        printf("Time%d\r\n",hour);
    }
}
```



```
/home/pi/tmp/tmp.3xnNX2yVna/codes/cmake-build-rpi/realtimeEmbedded
create thread  true
```

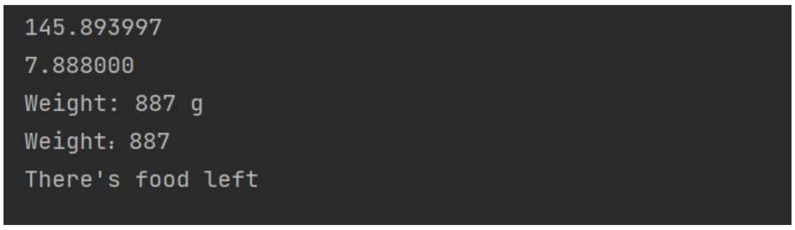
Figure 4. Time thread

For weight sensor, when the weight is less than the set minimum, the stepper motor is moving forward to add food. When the weight reaches the maximum, the stepper motor moves backward to stop adding. After finished the adding, *add_times++* add the total number of times, *hour=0* reset the cooling time

```
    void add_food(void){
        if(weii<WEIGHT_LOW){
            mot->forward(2,20);
            while (wei->loop()<WEIGHT_HIGH)
            {
                printf("Keep adding food\r\n");
            }
            mot->backward(2,20);
            add_times++;
              hour=0;
```

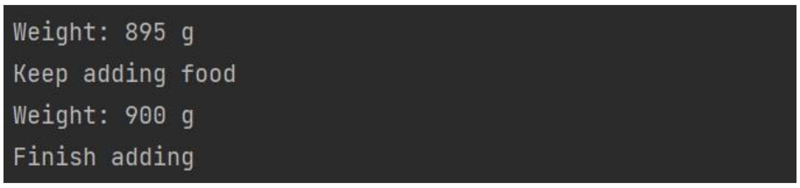
When the ultrasonic sensor detects an approach, check the weight condition. When meet the weight condition, distance condition, not in the cooling time and feeding times less than three at the same time, add food. (For the test, the minimum weight is set to 880, which indicates food surplus. The maximum weight is set to 900, at which point feeder will stop adding.)

```
        diss=dis->disMeasure();
        if(diss<DISTANCE){
            diss=dis->disMeasure();
            if(diss<DISTANCE){
                if(hour>=t&&add_times<=3){
                    add_food();
                }
```



```
145.893997
7.888000
Weight: 887 g
Weight: 887
There's food left
```

Figure 5. Food left



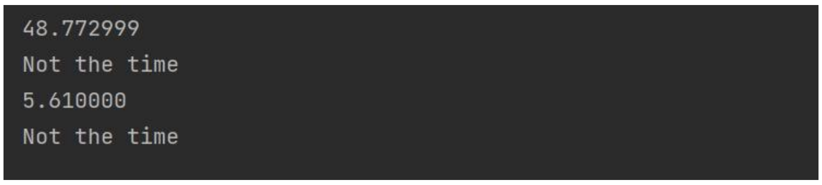
```
Weight: 895 g
Keep adding food
Weight: 900 g
Finish adding
```

Figure 6. Adding food

When feeding more than three times *add_times>3*, print "It has been fed three times". If *hour<t*, which means under the cold time, print "Not the time".

```
        if(add_times>3){
            printf("It has been fed three times\r\n");
        }
```

```
if(hour<t){
        printf("Not the time\r\n");
}
```



```
48.772999
Not the time
5.610000
Not the time
```

Figure 7. Under the cold time

V. Further Development

Due to time constraints, we have only completed the basic functions of the feeder. Here are some features that could be improved in the future.

5.1 Led

For different states of the feeder, different color LED lights can be added, so that people can intuitively understand the feeding situation. For example, when the number of feeds is less than three, the cooling time has expired, and the food weight is below the minimum, the green light is on, indicating that food can be added. If it is in the cooling period and feeding times are less than 3, the yellow light is on. If the number of feeding times is more than three, the red light is on, indicating that the daily feeding amount has reached the upper limit.

5.2 Storage weight

Mechanical Settings can be changed that turn the container for storage to a straight tube. The second weight sensor can be placed in the container to monitor food surplus in the feeder.

5.3 User interface (QT)

Use QT to create user interfaces and achieve interaction. Users can change the feeding quantity, feeding frequency and cooling time according to the pet's situation, increasing the flexibility of the product.

VI. Project Management

6.1 Timetable

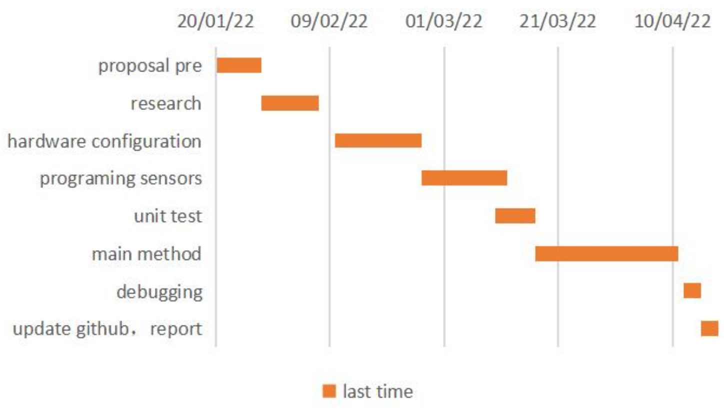


Figure 8. Timetable

6.2 Budget

Item	Quantity	Price
RaspberryPi 3B+	1	£40
Step sensor (HC-SR04)	1	£6
Weight sensor (HX711)	1	£10
Ultrasonic sensor (HC-SR04)	1	£7
Total		£63

Appendix

Unit test

Motor-test

motor.cpp

```
#include "motor.h"

void motor::setStep(int a, int b, int c, int d)
{
	digitalWrite(IN1, a);
	digitalWrite(IN2, b);
	digitalWrite(IN3, c);
	digitalWrite(IN4, d);
}

void motor::forward(int t, int steps)
{
	int i;
	for(i = 0; i < steps; i++){
		setStep(1, 0, 0, 0);
		delay(t);
		setStep(0, 1, 0, 0);
		delay(t);
		setStep(0, 0, 1, 0);
		delay(t);
		setStep(0, 0, 0, 1);
		delay(t);
	}
}

void motor::stop()
{
	setStep(0, 0, 0, 0);
}
```

```
void motor::backward(int t, int steps)
{
	int i;

	for(i = 0; i < steps; i++){
		setStep(0, 0, 0, 1);
		delay(t);
		setStep(0, 0, 1, 0);
		delay(t);
		setStep(0, 1, 0, 0);
		delay(t);
		setStep(1, 0, 0, 0);
		delay(t);
	}
}
```

motor.h

```
#ifndef REALTIMEEMBEDDED_MOTOR_H
#define REALTIMEEMBEDDED_MOTOR_H

#include <wiringPi.h>
#include <cstdio>

class motor{

public:
#define IN1	0	// pin11
#define IN2	1	// pin12
#define IN3	2	// pin13
#define IN4	3	// pin15

	static void setStep(int a, int b, int c, int d);
	static void forward(int t, int steps);
	static void stop();
	static void backward(int t, int steps);
};

#endif //REALTIMEEMBEDDED_MOTOR_H
```

motor_test.cpp

```
#include "motor.h"

int main()
{
	if (wiringPiSetup() < 0) {
		printf("Setup wiringPi failed!\n");
```

```
                return -1;
        }

        /* set pins mode as output */
        pinMode(IN1, OUTPUT);
        pinMode(IN2, OUTPUT);
        pinMode(IN3, OUTPUT);
        pinMode(IN4, OUTPUT);

        while (1) {
                printf("forward...\n");
                motor::forward(2, 100);

                printf("stop...\n");
                motor::stop();

                printf("backward...\n");
                motor::backward(2, 100);

                printf("stop...\n");
                motor::stop();
                delay(2000);
        }
}
```

weight_test

weight.cpp

```
#include "weight.h"

void weight::set_pin(weight::hx711_args *value) {
        value->EN = 1;
        value->coefficient = 415;
}

void weight::init_pin() {
        pinMode(SCK, OUTPUT);
        pinMode(SDA, INPUT);
        pullUpDnControl(SDA, PUD_UP);
}

void weight::start(weight::hx711_args *value) {
        int i;
        digitalWrite(SCK, LOW);
        while (digitalRead(SCK));
        value->value = 0;
```

```
	while (digitalRead(SDA));
	delay(100);

	for (i = 0; i < 24; i++) {
		digitalWrite(SCK, HIGH);
		while (0 == digitalRead(SCK))delay(1000);
		value->value = value->value * 2;
		digitalWrite(SCK, LOW);
		while (digitalRead(SCK));
		if (digitalRead(SDA))
			value->value = value->value + 1;
	}
	digitalWrite(SCK, HIGH);
	digitalWrite(SCK, LOW);
	if ((value->EN == 1) && (value->value < 25000)) {
		value->EN = 0;
		value->calibration = value->value;
	} else {
		i = (value->value - value->calibration + 50) / value->coefficient;
	}
	if (i < 5000)value->weight = i;
	printf("Weight: %d g\n", value->weight);
}


int weight::setup(weight::hx711_args *value) {
	if (wiringPiSetup() == -1)return 1;
	set_pin(value);
	init_pin();
	return 0;
}


void weight::loop(struct weight::hx711_args *value){
	while(1)
		weight::start(value);
}
```
weight.h

```
#ifndef REALTIMEEMBEDDED_WEIGHT_H
#define REALTIMEEMBEDDED_WEIGHT_H

# include <stdio.h>
# include <wiringPi.h>
#define SCK 4
```

```
#define SDA 5

class weight{
public:

    struct hx711_args{
        int EN;
        int calibration;
        int coefficient;
        int weight;
        unsigned long value;
    };
   static void set_pin(struct hx711_args *value);
   static void init_pin();
   static void start(struct hx711_args *value);
   static int setup(struct hx711_args *value);
   static void loop(struct weight::hx711_args *value);
};

#endif //REALTIMEEMBEDDED_WEIGHT_H
```

weight_test.cpp

```
#include "weight.h"

int main(void){
    struct weight::hx711_args value;
    if(0 == weight::setup(&value))
        weight::loop(&value);
    return 0;
}
```

ult_test

ult.cpp

```
#include "ult.h"

void ult::ultraInit(void)
{
    pinMode(Echo, INPUT);
    pinMode(Trig, OUTPUT);
}

float ult::disMeasure(void)
{
    struct timeval tv1;
    struct timeval tv2;
    long start, stop;
    float dis;

    digitalWrite(Trig, LOW);
    delayMicroseconds(2);

    digitalWrite(Trig, HIGH);    //produce a pluse
    delayMicroseconds(10);
    digitalWrite(Trig, LOW);

    while(!(digitalRead(Echo) == 1));
    gettimeofday(&tv1, NULL);                //current time

    while(!(digitalRead(Echo) == 0));
    gettimeofday(&tv2, NULL);                //current time

    start = tv1.tv_sec * 1000000 + tv1.tv_usec;
    stop   = tv2.tv_sec * 1000000 + tv2.tv_usec;

    dis = (float)(stop - start) / 1000000 * 34000 / 2;    //count the distance

    return dis;
}
```

ult.h

```
#ifndef REALTIMEEMBEDDED_ULT_H
#define REALTIMEEMBEDDED_ULT_H

#include <wiringPi.h>
#include <stdio.h>
#include <sys/time.h>
#include <iostream>
#define   Trig      21
#define   Echo       22

class ult{
public:



    static void ultraInit();
    static float disMeasure();
};

#endif //REALTIMEEMBEDDED_ULT_H
```

ult_test.cpp

```
#include "ult.h"
```

```
int main(void)
{
    float dis;

    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !\n");
        return -1;
    }

    ult::ultraInit();

    while(1){
        dis = ult::disMeasure();
        printf("Distance = %0.2f cm\n",dis);
        delay(1000);

    }

    return 0;
}
```

Feeder

distance.cpp

```
#include "distance.h"

distance::distance()
{
    pinMode(Echo, INPUT);
    pinMode(Trig, OUTPUT);
}
float distance::disMeasure(void)
{
    struct timeval tv1;
    struct timeval tv2;
    long start, stop;
    float dis;
    digitalWrite(Trig, LOW);
    delayMicroseconds(2);
    digitalWrite(Trig, HIGH);    //produce a pluse
    delayMicroseconds(10);
    digitalWrite(Trig, LOW);
    while(!(digitalRead(Echo) == 1));
    gettimeofday(&tv1, NULL);                    //current time
    while(!(digitalRead(Echo) == 0));
    gettimeofday(&tv2, NULL);                //current time
    start = tv1.tv_sec * 1000000 + tv1.tv_usec;
    stop  = tv2.tv_sec * 1000000 + tv2.tv_usec;
    dis = (float)(stop - start) / 1000000 * 34000 / 2;    //count the distance
    return dis;
}

distance::~distance()
{
}
```

distance.h

```
#ifndef DISTANCE_H
#define DISTANCE_H

#define  Trig    21
#define  Echo    22
#include <wiringPi.h>
#include <stdio.h>
#include <sys/time.h>

class distance
{
private:
    /* data */
public:
    distance();
    ~distance();
    float disMeasure(void);
};

#endif
```

motor.cpp

```
#include "motor.h"

void motor::setStep(int a, int b, int c, int d)
{
    digitalWrite(IN1, a);
    digitalWrite(IN2, b);
    digitalWrite(IN3, c);
    digitalWrite(IN4, d);
}

void motor::forward(int t, int steps)
{
	int i;
	for(i = 0; i < steps; i++){
		setStep(1, 0, 0, 0);
		delay(t);
		setStep(0, 1, 0, 0);
		delay(t);
		setStep(0, 0, 1, 0);
		delay(t);
		setStep(0, 0, 0, 1);
		delay(t);
	}
}

void motor::stop()
{
	setStep(0, 0, 0, 0);
}

void motor::backward(int t, int steps)
{
	int i;

	for(i = 0; i < steps; i++){
		setStep(0, 0, 0, 1);
		delay(t);
		setStep(0, 0, 1, 0);
		delay(t);
		setStep(0, 1, 0, 0);
		delay(t);
		setStep(1, 0, 0, 0);
		delay(t);
	}
}
motor::motor(){
	pinMode(IN1, OUTPUT);
	pinMode(IN2, OUTPUT);
	pinMode(IN3, OUTPUT);
	pinMode(IN4, OUTPUT);
}
```

motor.h

```
#ifndef REALTIMEEMBEDDED_MOTOR_H
#define REALTIMEEMBEDDED_MOTOR_H

#include <wiringPi.h>
```

```
#include <cstdio>

class motor{

public:
#define IN1    0        // pin11
#define IN2    1        // pin12
#define IN3    2        // pin13
#define IN4    3        // pin15

        static void setStep(int a, int b, int c, int d);
        static void forward(int t, int steps);
        static void stop();
        static void backward(int t, int steps);
        motor();
};

#endif //REALTIMEEMBEDDED_MOTOR_H
```

weight.cpp

```
#include "weight.h"

void weight::set_pin(weight::hx711_args *value) {
        value->EN = 1;
        value->coefficient = 415;
}

void weight::init_pin() {
        pinMode(SCK, OUTPUT);
        pinMode(SDA, INPUT);
        pullUpDnControl(SDA, PUD_UP);
}

void weight::start(weight::hx711_args *value) {
        int i;
        digitalWrite(SCK, LOW);
        while (digitalRead(SCK));
        value->value = 0;
        while (digitalRead(SDA));
        delay(100);

        for (i = 0; i < 24; i++) {
                digitalWrite(SCK, HIGH);
                while (0 == digitalRead(SCK))delay(1000);
                value->value = value->value * 2;
        digitalWrite(SCK, LOW);
        while (digitalRead(SCK));
        if (digitalRead(SDA))
            value->value = value->value + 1;
    }
    digitalWrite(SCK, HIGH);
    digitalWrite(SCK, LOW);
    if ((value->EN == 1) && (value->value < 25000)) {
        value->EN = 0;
        value->calibration = value->value;
    } else {
        i = (value->value - value->calibration + 50) / value->coefficient;
    }
    if (i < 5000)value->weight = i;
    printf("Weight: %d g\n", value->weight);
}


int weight::setup(weight::hx711_args *value) {
    if (wiringPiSetup() == -1)return 1;
    set_pin(value);
    init_pin();
    return 0;
}


int weight::loop(void){
    weight::start(&value);
    return value.weight;
}

weight::weight(){
    weight::setup(&value);
}
```
weight.h
```
#ifndef REALTIMEEMBEDDED_WEIGHT_H
#define REALTIMEEMBEDDED_WEIGHT_H

#include <stdio.h>
#include <wiringPi.h>
#define SCK 4
#define SDA 5

class weight{
```

```
public:

        struct hx711_args{
                int EN;
                int calibration;
                int coefficient;
                int weight;
                unsigned long value;
        };
        struct hx711_args value;
        static void set_pin(struct hx711_args *value);
        static void init_pin();
        static void start(struct hx711_args *value);
        static int setup(struct hx711_args *value);
        int loop(void);
        weight();
};

#endif //REALTIMEEMBEDDED_WEIGHT_H
```

main.cpp

```
#include <stdio.h>
#include <wiringPi.h>
#include <pthread.h>
#include "lib/motor.h"
#include "lib/weight.h"
#include "lib/distance.h"
#include <unistd.h>

#define DISTANCE 50
#define WEIGHT_LOW    880
#define WEIGHT_HIGH 900


distance * dis;
weight      *wei;
motor        *mot;


int t=1;
//current time
int hour=1;
int add_times=0;
//time thread
pthread_t time_thread;
```

```
//add food
void add_food(void){
	int weii=wei->loop();

	printf("Weight：%d\r\n",weii);
	if(weii<WEIGHT_LOW){
		printf("Adding some food\r\n");

		mot->forward(2,20);
		while (wei->loop()<WEIGHT_HIGH)
		{
			printf("Keep adding food\r\n");
			delay(100);
		}
		printf("Finish adding\r\n");
		mot->backward(2,20);
		add_times++;
		//refresh the time
		hour=0;

	}else{
		printf("There's food left \r\n");
	}

}

void time_thread_run(void *ptr)
{
	//one hour
	//int time_ctl=60*60*1;
	// one minute
	int time_ctl=60*1;
	while(1){
		sleep(time_ctl);
		hour++;
		printf("Time%d\r\n",hour);
	}
}


int main(){
	wei=new weight();
	dis=new distance();
```

```
    mot=new motor();

    int ret_thread1;
    ret_thread1 = pthread_create(&time_thread,NULL,(void*
(*)(void*))&time_thread_run,NULL);
    if(ret_thread1 == 0)
        printf("create thread    true\n");
    else
        printf("create thread    false\n");
    static int distance_times=0;
    float diss=0;
    while(1){
        diss=100;
        diss=dis->disMeasure();
        printf("%f \r\n",diss);
        if(diss<DISTANCE){
            //Detection of proximity
            delay(200);
            diss=dis->disMeasure();
            if(diss<DISTANCE){
                if(hour>=t&&add_times<=3){
                    //Meet the conditions
                    add_food();
                }
                if(add_times>3){
                    printf("It has been fed three times\r\n");
                }
                if(hour<t){
                    printf("Not the time\r\n");
                }
            }
        }

        sleep(1);
    }

}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.13)
project(realtimeEmbedded)

set(CMAKE_CXX_STANDARD 14)

add_executable(realtimeEmbedded    main.cpp    lib/motor.h    lib/motor.cpp    lib/distance.h
lib/distance.cpp lib/weight.h lib/weight.cpp)

target_link_libraries(realtimeEmbedded    wiringPi)
target_link_libraries(realtimeEmbedded    pthread)
```