# UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

**Colegio de Ciencias e Ingeniería**

# Vultur Gryphus Artificial Incubator Automatization

**Artículo Académico**

**José Antonio Carrera Matute**

**Ingeniería Electrónica**

Trabajo de titulación presentado como requisito
para la obtención del título de
Ingeniero Electrónico

Quito, 16 de mayo de 2017

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

COLEGIO DE CIENCIAS E INGENIERIAS

**HOJA DE CALIFICACIÓN**
**DE TRABAJO DE TITULACIÓN**

**Vultur Grypghus Artificial Incubator Automatization**

**José Antonio Carrera Matute**

| | |
|---|---|
| Calificación: | |
| Nombre del profesor, Título académico | Diego Benítez , Ph.D. |
| Firma del profesor | ______________________________ |

Quito, 16 de mayo de 2017

## Derechos de Autor

| | |
|---|---|
| Firma del estudiante: | _______________________________ |
| Nombres y apellidos: | José Antonio Carrera Matute |
| Código: | 00113947 |
| Cédula de Identidad: | 1720598968 |
| Lugar y fecha: | Quito, mayo de 2017 |

## AGRADECIMIENTOS

Agradezco a mis padres José Till Carrera Pérez y Silvia Guillermina Matute quienes han sido un gran apoyo para mí en todo el transcurso de mi carrera tanto económica como emocionalmente.

También agradezco a mis hermanos que me han acompañado en todo este trayecto y me han brindado su afecto para seguir adelante.

A mis compañeros de universidad que me ayudaron a seguir adelante en mis clases y proyecto final y hacer mi permanencia en la universidad agradable y cómoda.

A mis profesores, los cuales me prestaron su ayuda para la culminación de mi proyecto con aportes de su propia experiencia. Especialmente a Diego Benítez, que fue un gran apoyo para la culminación de mi trabajo de titulación.

## RESUMEN

El siguiente articulo muestra el trabajo realizado para mejorar una incubadora artificial realizada en un proyecto anterior. Los parámetros para mejorar la incubadora fueron: el diseño de un controlador PID para temperatura con un relé de estado sólido como actuador, la automatización del giro del huevo con un servo para controlar la posición y velocidad de giro del eje controlado por un reloj de tiempo real (RTC), se cambió la manguera del humidificador para obtener una mayor estabilidad en el punto establecido, y se coloca un nuevo sensor de temperatura y humedad para monitorear esos parámetros con mayor precisión.

*Palabras clave:* Temperatura, Humedad, PID, Incubadora, Servo, RTC.

## ABSTRACT

This paper shows the work realized to improve an artificial incubator which was made in a previous project. The areas that were improved in this project were: the temperature controller with a PID design in Arduino and a SSR (Solid State Relay). The automatization of the turning of the egg, using a servo motor to control the position and the velocity of the turn, and a RTC (Real Time Clock). The hose of the humidifier was replaced to have more stability at the set-point of the humidity controller. And finally, new humidity and temperature sensor were installed in the equipment for monitoring those parameters with more accuracy.

*Key words:* Temperature, Humidity, PID, Incubator, Servo, RTC.

# TABLA DE CONTENIDO

# VULTUR GRYPHUS ARTIFICIAL INCUBATOR AUTOMATIZATION

José Carrera and Diego Benítez
Universidad San Francisco de Quito
Colegio de Ciencias e Ingeniería
Campus Cumbayá
Quito, Ecuador
jose.carrera@estud.usfq.edu.ec, dbenitez@usfq.edu.ec

***Abstract---*** **This paper shows the work realized to improve an artificial incubator which was made in a previous project. The areas that were improved in this project were: the temperature controller with a PID design in Arduino and a SSR (Solid State Relay). The automatization of the turning of the egg, using a servo motor to control the position and the velocity of the turn, and a RTC (Real Time Clock). The hose of the humidifier was replaced to have more stability at the set-point of the humidity controller. And finally, new humidity and temperature sensors were installed in the equipment for monitoring those parameters with more accuracy.**

***Index Terms---*****Temperature, Humidity, PID, Incubator, Servo, RTC.**

## I. INTRODUCTION

The Andes are home of the Andean Condor (Vultur Gryphus). Nowadays in Ecuador, there are less than fifty specimens living in the wild and less than ten breeding pairs in captivity. The Andean Condor can lay up to two eggs per year, if the first egg has been taken away or damage. The goal of having an artificial incubator, is to be able to have two broods with the same breeding pair instead of just one [2].

The initial version of the design and construction of the artificial incubator was first made by Danny Xavier Ron Castro, a former student of Universidad San Francisco de Quito, The objective of this project is to repair and improve some of the parameters that did not work properly in the previous version of the incubator.

The first thing that was improved is the temperature controller by using a PID and a thermal resistance with more power. The previous temperature controller kept the temperature under the set point, and had a slow disturbance response. Then a servo motor was added and controlled by a clock for turning the egg in a specific time. The servo control helps the egg to have a smooth turn, since the speed and torque can be controlled. Another thing that was improved was the humidity controller. However, this was a mechanical improvement since the electrical part was working properly. Finally, the last thing was to replace the humidity and temperature sensor to monitor the incubator with more accuracy.

## II. IMPROVEMENTS AND REPARATIONS FOR THE ARTIFICIAL INCUBATOR

This project is a continuation of a previous project, that consisted in the design and construction of an artificial incubator with a mechanical emphasis. The improvements made in this project are more related to the electrical part, such as, a PID to control temperature. Also, the turning control of a servo with an RTC incorporated. A temperature and humidity controller display in a LCD. Finally, a mechanical reparation to fix the humidity problem was also implemented.

### *A. PID temperature controller*

To obtain the input signal for the PID controller, a LM35 temperature sensor was used. The LM35 is a precision integrated-circuit temperature sensor with an linear output voltage proportional to the Centigrade temperature. The sensor sensitivity is $10\ \frac{mV}{°C}$. It has $0.5\ °C$ ensured accuracy, its full range goes from $-55°C\ to\ 150°C$, and it can operate from 4V to 30V [5]. For further explanation of the LM35 it is recommended to use its datasheet. Since it is necessary to have a very accurate reading of the temperature, the LM35 was placed right behind the egg.

The actuator implemented for temperature control is a thermal resistance which has approximately 700 W at full power. Since the actuator uses AC, a Solid-State Relay (SSR) was required. The SSR is the equivalent of an Electro-Mechanical Relay (EMR) without moving parts, and with a faster electrical response. It allows to control AC circuits with DC components since provide complete electrical isolation. The SSR used has an input that operates from 4V to 32V DC and an output from 110V to 280V AC with a 40 A maximum current capacity.

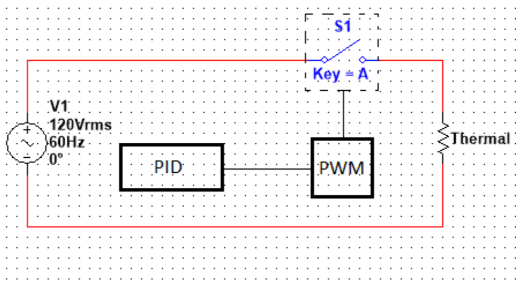


Fig. 1. Schematic of the temperature controller.

The PID controls a Pulse Width Modulation (PWM) output of Arduino that is connected to de SSR input, and the output of the SSR goes to the thermal resistance as shown in Fig. 1. The Duty Cycle of the PWM has a scale of 0 to 255, where 0 is the 0%, and 255 is the 100% of the Duty Cycle. The S1 in Fig. 1 represents the SSR used to control the thermal resistance.

The temperature controller was made using the “PID.h” library of Arduino which uses the basic equation:

$$Ops = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \qquad (1)$$

Where $Ops$ is the Output signal, $e(t)$ is the error signal defined as follow:

$$e(t) = Setpoint - Input \qquad (2)$$

$K_P$ is the proportional constant, $K_I$ is the integral constant, and $K_D$ is the derivative constant.

This equation can be represented in a block diagram for a better explanation.

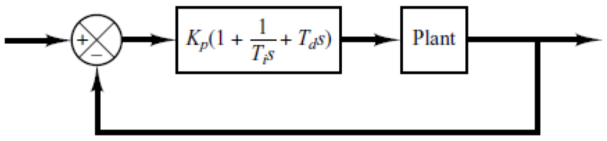


Fig. 2. Block diagram of a PID controller [3].

The PID block in Fig. 2 uses the LaPlace transformation to make calculations simpler. Also, $K_I$ and $K_D$ can be define as [4]:

$$K_I = \frac{K_P}{T_I}, and\ K_D = K_P T_D$$

The integral and derivative constants are dependent of the proportional constant. It means that for the two parameters, what is changing are the integral time $T_I$, and the derivative time $T_D$.

To get the data for the close-loop transfer function without a PID controller, the values of the parameters were:

$K_P = 1, K_I = 0, and\ K_D = 0$

This means $T_I \to \infty$ (or a large number, since the infinite can not be represented), and $T_D = 0$. The Arduino library has its algorithm to make this possible, so in the program it was changed the $K_I$ and $K_D$ values only. The reason these values were taken, was to have equation (1) simplified as follow:

$$Ops = e(t) = Setpoint - Input \qquad (3)$$

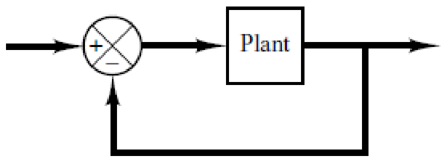


Fig. 3. Block diagram of a Plant.

Fig. 10 in section III shows the behavior of the plant with the constants written above.

The temperature got stable under the set-point for 1℃ aproximetaly. To improve the response, first was needed to increase the $K_P$ to a value where it is close to the set point, but does not surpasses the set-point at any moment.

The main condition of the controller was to do not have an overshoot bigger than 0.5%. However, the response time was not necessary fast. Using a try and error experiment, the proportional constant was changed slightly until the response got stable between 0.3℃ under the set point. The new value of $K_P$ was therefore:

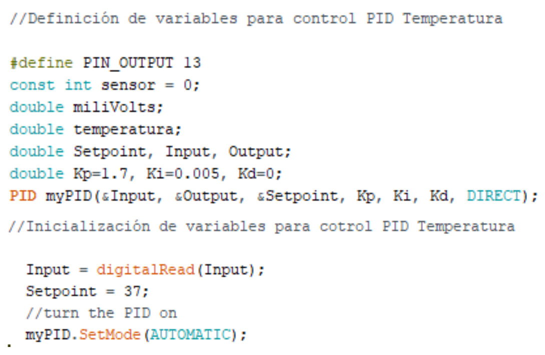
$$K_P = 1.7$$

This simulates a P-Controller that has 0.3℃ of offset. Once the new proportional constant was added, the next parameter to change was the integral constant. The integral action allows a PI controller to eliminate the offset, which cannot be done with a P controller only [4]. $K_I$ was changed until the response gets stable at the set point. Therefore, the new value of the integral constant was:

$$K_I = 0.005$$

This new constant made the response to oscillate between $\pm 0.2$℃ around the set-point. Since the control parameter is temperature, the response is already very slow, in other words a derivative constant was not required for this controller [4]. The final response is shown in section III in Fig. 11.

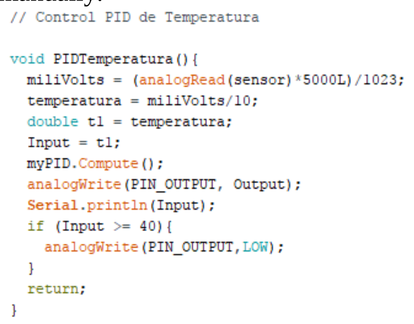
The code in Arduino for the PID controller is shown below:



```
//Definición de variables para control PID Temperatura

#define PIN_OUTPUT 13
const int sensor = 0;
double miliVolts;
double temperatura;
double Setpoint, Input, Output;
double Kp=1.7, Ki=0.005, Kd=0;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
//Inicialización de variables para cotrol PID Temperatura

  Input = digitalRead(Input);
  Setpoint = 37;
  //turn the PID on
  myPID.SetMode(AUTOMATIC);
```

Fig. 4. Variable definitions for PID controller.

Fig. 4 shows the definitions of the PID controller. Before the definitions, the “PID.h” library must be added. The pin output 13 of the Arduino sends the signal to the SSR to control the thermal resistance. The A0 input of Arduino receives the signal from the LM35. Some variables must be defined for further use in the void loop, such as, “miliVolts”, and “temperatura” which are used for the LM35 to read the temperature. The remaining variables are required for the PID library. Kp, Ki, and Kd are the control of the PID. For the PID to work continuously, the set mode must be put in automatic, if not, the output must be controlled manually.



```
// Control PID de Temperatura

void PIDTemperatura(){
  miliVolts = (analogRead(sensor)*5000L)/1023;
  temperatura = miliVolts/10;
  double t1 = temperatura;
  Input = t1;
  myPID.Compute();
  analogWrite(PIN_OUTPUT, Output);
  Serial.println(Input);
  if (Input >= 40){
    analogWrite(PIN_OUTPUT,LOW);
  }
  return;
}
```

Fig. 5. Algorithm for the PID controller.

The LM35 sends the signal in millivolts, so in order to read in Celcius, a transformation has to be done as shown in Fig. 5. The last two lines are used for an emergency stop to do not overpass the 40℃ because it could kill the embryo, and if the error signal shoots a bigger temperature, it could also damage the sensors.

## B. *Temperature and humidity monitor*

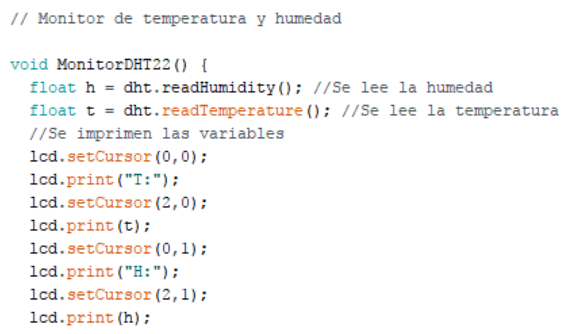
To monitor the temperature and humidity of the incubator, the DHT22 sensor was required. The DHT22 is an Arduino sensor module that measures humidity and temperature. Some of the characteristics of the DHT22 are shown below:

TABLE I
DHT22 CHARACTERISTICS [1]

| | |
|---|---|
| Model | AM2302 or DHT22 |
| Power supply | 3.3V-5.5V DC |
| Humidity operating range | 0%-100%RH (Relative Humidity) |
| Temperature operating range | From -40 to 80 °C |
| Humidity accuracy | $\pm 2\% RH$ |
| Temperature accuracy | $\pm 0.5 ^\circ C$ |

For the same reason of the LM35, the DHT22 was placed very close to the egg.

Since the DHT22 is an Arduino module, the code to make it run was simple. The “DHT.h” library of Arduino. was used, then the only thing left to do was to initialize the temperature and humidity variable.



```
// Monitor de temperatura y humedad

void MonitorDHT22() {
  float h = dht.readHumidity(); //Se lee la humedad
  float t = dht.readTemperature(); //Se lee la temperatura
  //Se imprimen las variables
  lcd.setCursor(0,0);
  lcd.print("T:");
  lcd.setCursor(2,0);
  lcd.print(t);
  lcd.setCursor(0,1);
  lcd.print("H:");
  lcd.setCursor(2,1);
  lcd.print(h);
```

Fig. 6. Algorithm of the DHT22 monitor and LCD program.

To show the parameters explained above, a LCD (Liquid Crystal Display) of 16x2 was used, it was programed to show temperature, humidity, and time.

## C. *Turning control of the egg*

In a natural incubation, the egg must be turned several times in one day to avoid the embryo to get stuck in the shell of the egg.

The actuator to make the turn is a MG996R servo motor that has the characteristics shown in table II:

TABLE II
MG996R CHARACTERISTICS [6]

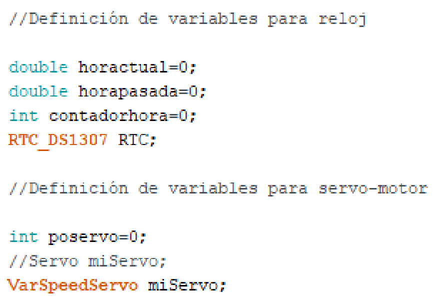
| | Min | Max |
|---|---|---|
| Stall torque | $9.4\ [kgf * cm](4.8V)$ | $11\ [kgf * cm](6V)$ |
| Operating voltage | 4.8 [V] | 6 [V] |
| Running Current | 500 [mA] | 900 [mA] |

To make the time control, the RTC I2C DS1307 was used. It is a Real-Time Clock (RTC) device, which is also an Arduino module. The library used for the RTC was the “RTClib.h”. With this library, it is possible to decide which values are to be show. In this case, the hours, minutes, and seconds were required. This clock can provide the exact time of the computer in which the program was load, and stores the data with a clock battery to keep running [3]. The time is shown in the LCD as explained previously.

Since the servo consumes from 500mA to 900mA [6]. The safest way to make a connection is with an external power source, if the Arduino is used as source, it could cause misread values and even damage some of the components of the Arduino.

The egg of a Vultur Gryphus must be turn every eight hours. However, this parameter can be switched by the user depending on the type of bird egg.

The Arduino’s code for the turning time control is as follow:

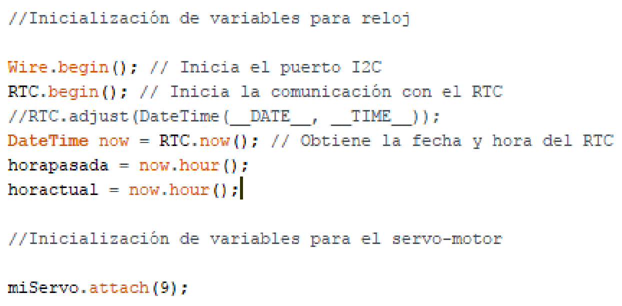


```
//Definición de variables para reloj

double horactual=0;
double horapasada=0;
int contadorhora=0;
RTC_DS1307 RTC;

//Definición de variables para servo-motor

int poservo=0;
//Servo miServo;
VarSpeedServo miServo;
```

Fig. 7.  Variable definition for the RTC program and servo program.

For the RTC program, the first thing to do is to define parameters that helped to control the position of the servo. Also, it was needed to define the function for the library.

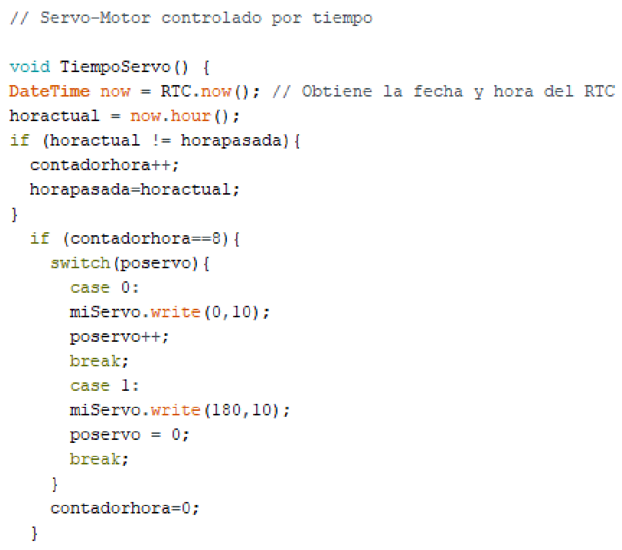
For the servo, the position had to be in 0° when the program starts running. Moreover, as in the RTC, the function for the library had to be defined.



```
//Inicialización de variables para reloj

Wire.begin(); // Inicia el puerto I2C
RTC.begin(); // Inicia la comunicación con el RTC
//RTC.adjust(DateTime(__DATE__, __TIME__));
DateTime now = RTC.now(); // Obtiene la fecha y hora del RTC
horapasada = now.hour();
horactual = now.hour();

//Inicialización de variables para el servo-motor

miServo.attach(9);
```

Fig. 8.  Initialization of variables and pins for the RTC and servo.

To get the date from the computer, the "RTC.adjust" command was used. After the first time running the program, this line must be commented so the module of the RTC can run the time automatically, and show the exact time even after the system has been turned off. For the servo, the program must know where is sending the data, in this case the output is pin 9.



```
// Servo-Motor controlado por tiempo

void TiempoServo() {
DateTime now = RTC.now(); // Obtiene la fecha y hora del RTC
horactual = now.hour();
if (horactual != horapasada){
  contadorhora++;
  horapasada=horactual;
}
  if (contadorhora==8){
    switch(poservo){
      case 0:
      miServo.write(0,10);
      poservo++;
      break;
      case 1:
      miServo.write(180,10);
      poservo = 0;
      break;
    }
    contadorhora=0;
  }
```

Fig. 9.  Algorithm to run a servo controlled by a Real-Time Clock.

The "contadorhora" variable is a counter that is adding 1 unit every time an hour had passed, when the counter reaches 8 (eight hours), the program passes to the if-loop, and depending of the case in the switch-loop, the servo will be in either 0° or 180°. The velocity can be controled thanks to the VarServoSpeed library, in which the velocity has a scale of 0 to 100, where 0 is minimum speed and 100 is the maximum speed of the servo.

### D. *Humidity problem fixed*

The humidity controller uses the WH8040. It uses the HM40 humidity sensor. It also operates at 110V-220V, and uses an external humidifier [2].

As explained in the introduction, this is a continuity of a previous project. One of the problems that had to be fixed, was to maintain the humidity stable. However, this was a mechanical reparation since the humidity controller was working properly. The work in this part, was to switch the hose for one with a bigger diameter. The diameter of previous hose was about 1cm wide, and for the new one is about 2cm wide. With the new implementation, the controller can now reach to the set point and stabilize it in quicker manner.

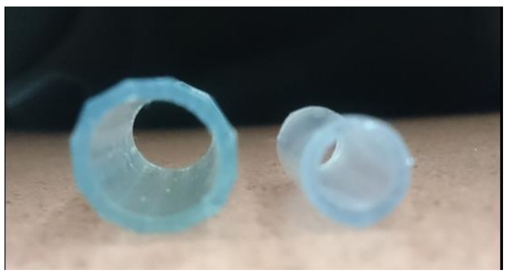


Fig. 10.  On the left is placed the new hose of 2cm wide and on the right, is placed the old hose of 1cm wide.

The WH8040 can be set to any value of humidity, so it is useful for other types of bird or animal eggs. However, for a Vultur Gryphus egg, the relative humidity must be set at 47.1% [2].

## III. Results

### A. *PID temperature controller.*

The next graph is showing the response of the plant when the proportional constant is 1, and the integral and derivative constant is 0.

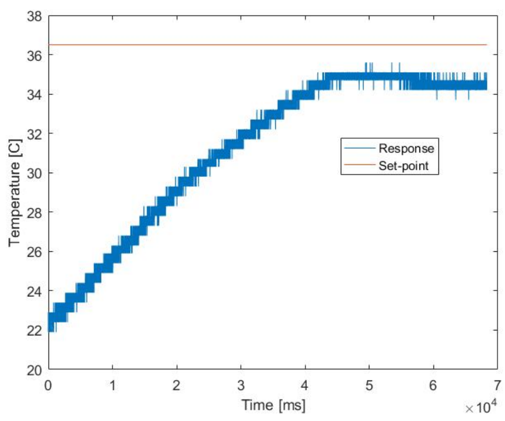


Fig. 11.  Temperature vs time graph of a P-controller.

The response reaches a value of temperature under the set-point with small oscillations between $34\ to\ 35℃$. The value of the set-point for this case is $36.5℃$. It means that still has an offset. This offset can be eliminated with the integral action.

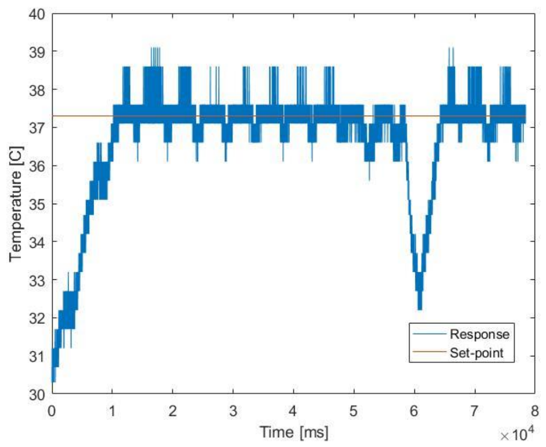


Fig. 12. Temperature vs time graph of a PI-controller.

The set-point for the response in fig. 12 is $37.5℃$. With the new $K_I$ of 0.005, the response oscillates in around that value. Also, the door of the incubator was open to see the behavior of the plant. The temperature decreases fast with the door open. However, the main problem was to not exceed the set point. Finally, the oscillations do not affect the incubation because the maximum value remains for a small period.

### B. *Humidity problem fixed*

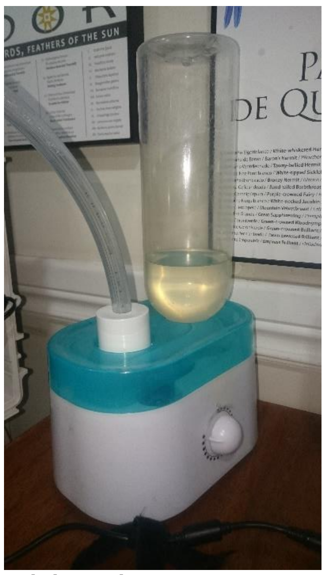


Fig. 13. Humidifier with the new hose.

Fig. 13 shows the final state of the humidifier which stabilizes the humidity in any range set, even at high temperatures.

### C. *Turning control of the egg.*

The system to turn the egg is a pivot with a metal basket in the center to place the egg. Fig. 14 shows an exact replica of a Vultur Gryphus egg. Since the basket is bigger than the egg, it must be attach. For an Andean Condor egg, the pivot must turn 180° every 8 hours [2].

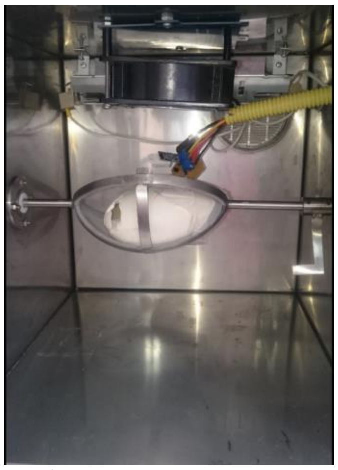


Fig. 14. Egg replica of an Andean Condor placed on the pivot with a 180 degree turn.

## IV. Conclusions

For the PID temperature controller, all kind of disturbances had to be analyzed to put the right values on the proportional and integral constants. The humidity is produced by high frequency vibrations in a tank filled with water, when the steam enters into the incubator, the temperature decreases and the PID must increase the heat. Therefore, before putting the constants, the humidity controller needs to stabilize. Also, the egg must be removed from the incubator to be tested, once the door is open and close moments later, it cannot have an overshoot that could kill the embryo.

The try and error method for a PID controller can be used when there is not a transfer function that describes the plant. Although, it can give good results, is recommended to get the transfer function to find the constants with more precise design methods such as the Zeigler-Nichols method. Also, with the transfer function, simulations can be done, to prevent the error to expand to the infinite putting in risk the egg and the devices inside the incubator. However, an emergency stop program is always needed for situations when the control is not working properly.

The LM35 is a good sensor to make a PID controller because it has an analogical output therefore, the response is given in real time and the only limit is the micro-controller that is programed. However, this sensor has $\pm 0.5℃$ of error, and to have a more accurate response, it is recommended to use another type of sensor. This is most likely to rise the prices of the project, but it could be safer for the embryo.

A servo can consume more current than the Arduino can give. This could cause the Arduino to send wrong signals to other outputs used at the same time. Also, this excessive current could be dangerous for a computer if is powering the Arduino, since a computer usually takes no more than 0.5 A. To avoid this problem, it is recommended to use an external source to power the servo, and to check that the grounds of the Arduino and the source are connected. The power source in this case, had to provide at least 1 A. However, some servos consume more current.

The DHT22 has the same accuracy of the LM35. However, since it sends a digital signal, it takes more time for processing; therefore, it was not use for the PID controller.

The humidity factor is not critical in the incubation, for that reason, it can oscillate if it remains around the set-point.

## REFERENCES

[1] Adafruit Industries. Digital relative humidity & temperature sensor AM2302/DHT22. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf

[2] D. X. Ron. (2015, May). Diseño y construcción de una incubadora y criadora para huevos de cóndor andino. Ecuador. [Online]. Available: http://repositorio.usfq.edu.ec/bitstream/23000/4230/1/114112.pdf

[3] Elecrow. (2013, September). Tiny RTC. [Online]. Available: https://www.openhacks.com/uploadsproductos/tiny_rtc_-_elecrow.pdf

[4] K. Ogata, "Chapter 8 PID Controllers and Modified PID controllers," in *Modern Control Engineering*, fifth ed. New Jersey: Prentice Hall, 2010, pp. 567-577.

[5] Texas Instruments. (2016, August). LM35 Precision Centigrade Temperature Sensors. [Online]. Available: http://www.ti.com/lit/ds/symlink/lm35.pdf

[6] TowerPro. MG996R High Torque Metal Gear Dual Ball Bearing Servo. [Online]. Available: http://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf