MECH 471

MICROCONTROLLERS FOR MECHATRONICS

Levitating Ping-Pong Ball

Maria Papadopoulos - 9350845

James Bracken - 6251420

Date Submitted: April 20, 2015

ABSTRACT

Microcontrollers allow programming languages and hardware to interface with various devices. The goal of this project is to levitate a ping-pong ball inside a cylindrical acrylic tube and to control its transient and steady state characteristics accurately. The use of a microcontroller, DC electric fan and motor driver circuit is used to maintain a steady state point set by the user. It was possible to control both the transient and steady state positions of the ping-pong ball within the acrylic tube. It was determined that there is an inherit benefit to using the PID control over typical positional control.

1. INTRODUCTION AND PROBLEM FORMULATION

Microcontrollers are often found in many autonomous devices to control their behavior. The objective of this project is to levitate a ping-pong ball inside a cylindrical tube to a desired height. The circuit of the system will possess several buttons, each representing a specific desired height, which serves as a user interface. The ultrasonic sensor will sense the position of the current height of the ping-pong ball. This will serve as the sensor input to the Arduino Mega microcontroller which will control the speed of motor which serves as a fan. The speed of the fan will propel the ball upwards and maintain the desired height inputted by the user via the switch. The system set up is illustrated in Figure 1. The ultrasonic sensor is to be placed on the top of the cylinder and the fan on the bottom of the cylinder. This design decision was made in order to avoid uneven airflow distribution within the cylinder.

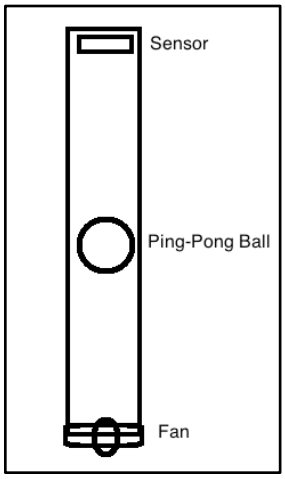

Figure 1 - Mechanical Set-up

Considering that the purpose of this project is to bring the ping-pong ball to a desired height and maintain its position using airflow from a fan, the dynamics of the this system must first be understood. The acrylic cylinder restricts the airflow to a confined space and volume thus eliminating possible external disturbances to the system. The fan provides air flow, which can either be laminar or turbulent in nature depending on the speed of the fan. Assuming that the fan produces laminar airflow, the ping-pong ball will fly up until it reaches a point of balance whereby its weight, influenced by gravity, will equal to the force to the air beneath it as illustrated in Figure 2.

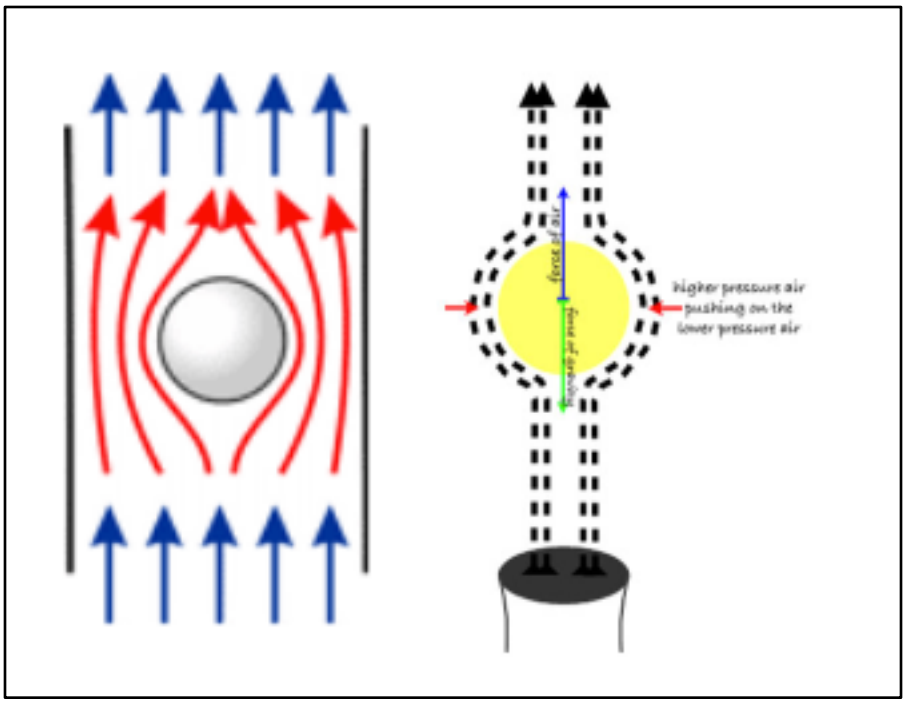

Figure 2 - Bernoulli floating ball [1]

If the system was not confined inside a cylinder, the ball would still remain balanced if there were no external disturbances. The air beneath the ping-pong ball is at a higher speed than the air surrounding it. In other words, there is a higher air pressure surrounding the ping-pong ball

compared to the air pressure beneath it. [2] The relationship between pressure, speed and height at two points in a non-viscous, laminar, incompressible fluid is known as Bernoulli's equation in fluid dynamics. [3] Thus it is the higher pressured air surrounding the ping-pong ball that keeps it within the laminar stream of air produced by the fan.

The effect of the ping-pong ball spinning can also be explained using Bernoulli's equation. The side of the ball which turns in the direction parallel to the air flow, denoted as A in Figure 3, creates a higher air speed and a lower pressure while side B will possess a lower speed and a higher pressure. This causes the ping-pong ball to move towards side A. [4]

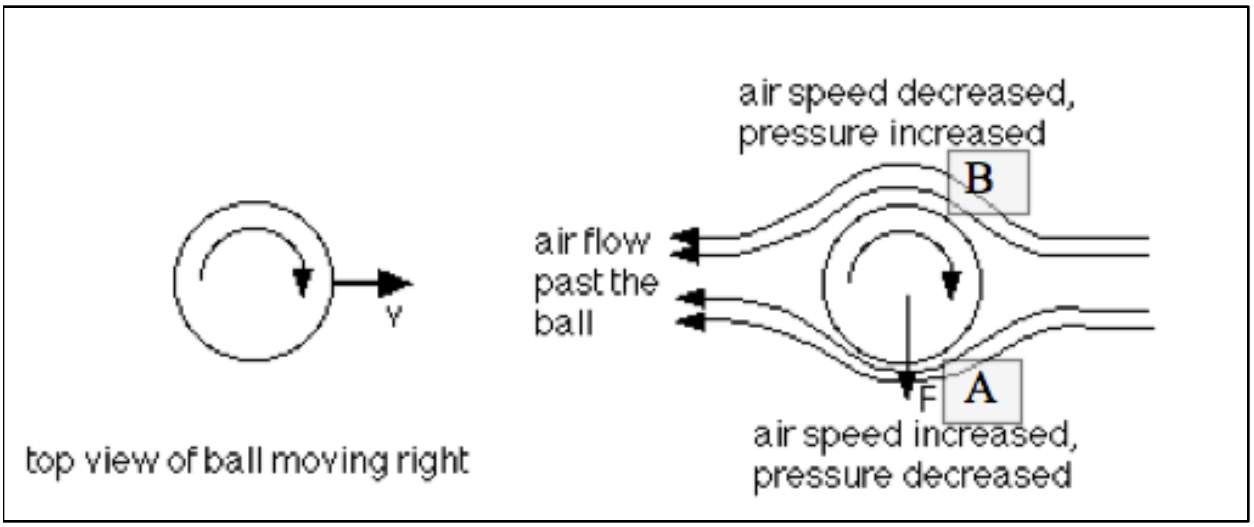

Figure 3 - Spinning ball [2]

2. SOLUTION

2.1 Hardware

The following parts were required for the hardware aspect of the project:

- Arduino Mega 2560
- Ultrasonic range measuring module (SEN136B5B)
- Clear acrylic cylinder (1 meter in height, 4.5cm inner diameter, 5.5cm outer diameter)
- Power MOSFET, IRF530 (TO-220 package)
- Resistors (R1= 10kΩ, R2=100Ω, R3-R6=100kΩ)
- Diode
- DC motor, with fan and shroud
- Ping-Pong ball (4cm diameter)
- External power supply, provided in MECH 471 laboratory room H-1057

The microcontroller used for this project is the Arduino Mega 2560. It offers 54 digital input/ output pins whereby 15 can serve as pulse-width modulation (PWM) and 16 can serve as analog inputs/outputs. The primary use of this particular model is due to availability for no cost through a team member. Additionally, Arduinos are programmed using C++ language.

The Ultrasonic range measurement module used is a Seeed model SEN136B5B. It is capable of detecting a 3m to 4m range with a 1cm resolution. Considering that the cylinder is one meter in length, this sensor is more than sufficient for the suggested project. It operates at a 40k Hz ultrasonic frequency and has a 5VDC supply voltage. Since it consumes 15mA during use, there is no need for an external power supply and can therefore be powered by the Arduino. The main

benefit of this sensor is that it includes an integrated circuit board that has a driver, controller and converter; thus allowing for one pin communication to the Arduino.

The simple plastic ducted fan shroud with electric motor and blade was provided by the lab technician Gilles Huard. This fan has been tested to ensure that its thrust is powerful enough to propel the ping-pong ball upwards from a rest state. The motor will be controlled using a simple MOSFET switching circuit. The motor will be switched on and off via a PWM output from the Arduino to allow for variable motor speed.

How these parts interact are demonstrated in flowchart of Figure 4. The Arduino Mega serves as the central processing unit. The ultrasonic sensor measures the distance of the ball and inputs this information to the microcontroller. The microprocessor then judges whether the ball is at the desired height by comparing the actual height to the desired height which is inputted via a button. After analyzing this comparison, the microcontroller outputs a certain PWM to the MOSFET driver which either drives the ball further upward, downward or holds steady.

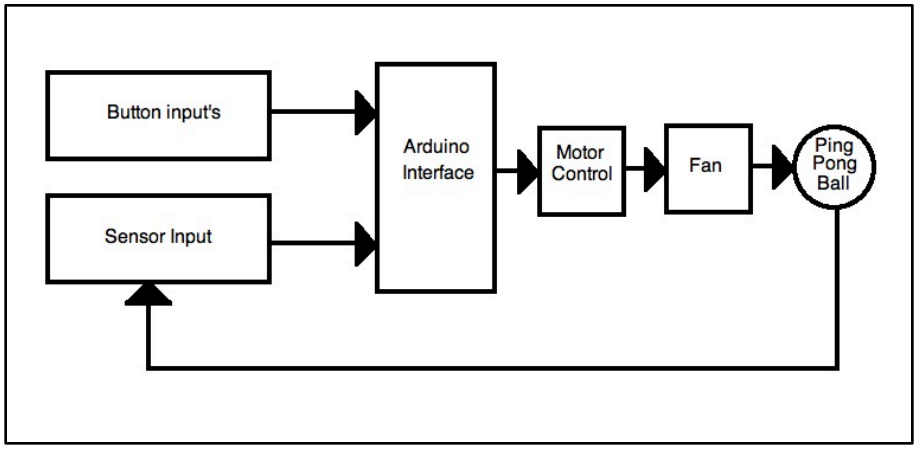

Figure 4 - Flowchart

This flowchart was achieved by creating the circuit illustrated in Figure 5. Pins 18, 19, 20, and 21 of the Arduino have been attached to switches S1, S2, S3 and S4 and to resistors R3, R4, R5 and R6. This serves are the user interface whereby the user will press one of the four buttons to indicate a desired height the ball is move towards and rest, the resistors are used as pull-down resistors that hold the pin at a logic LOW level when there is no power applied. The ultrasonic sensor is attached to pin 50 of the Arduino, to the Arduino's power source of 5V, and to ground. The PWM pin 10 is connected to the driver circuit which consists of a power MOSFET for motor control, resistors R1 and R2 coupled with the motor (resistive load) are sized in order to switch the MOSFET between its cut-off region and its saturation region. The diode that connects across the motor is to provide a safe path for the inductive current return of the motor when it is switched off. Because the motor has inductive characteristics (as well as being a resistive load) it will increase the voltage in order to maintain current flow when the motor is switched off, this results in an unsafe voltage spike that can damage the MOSFET. [5]

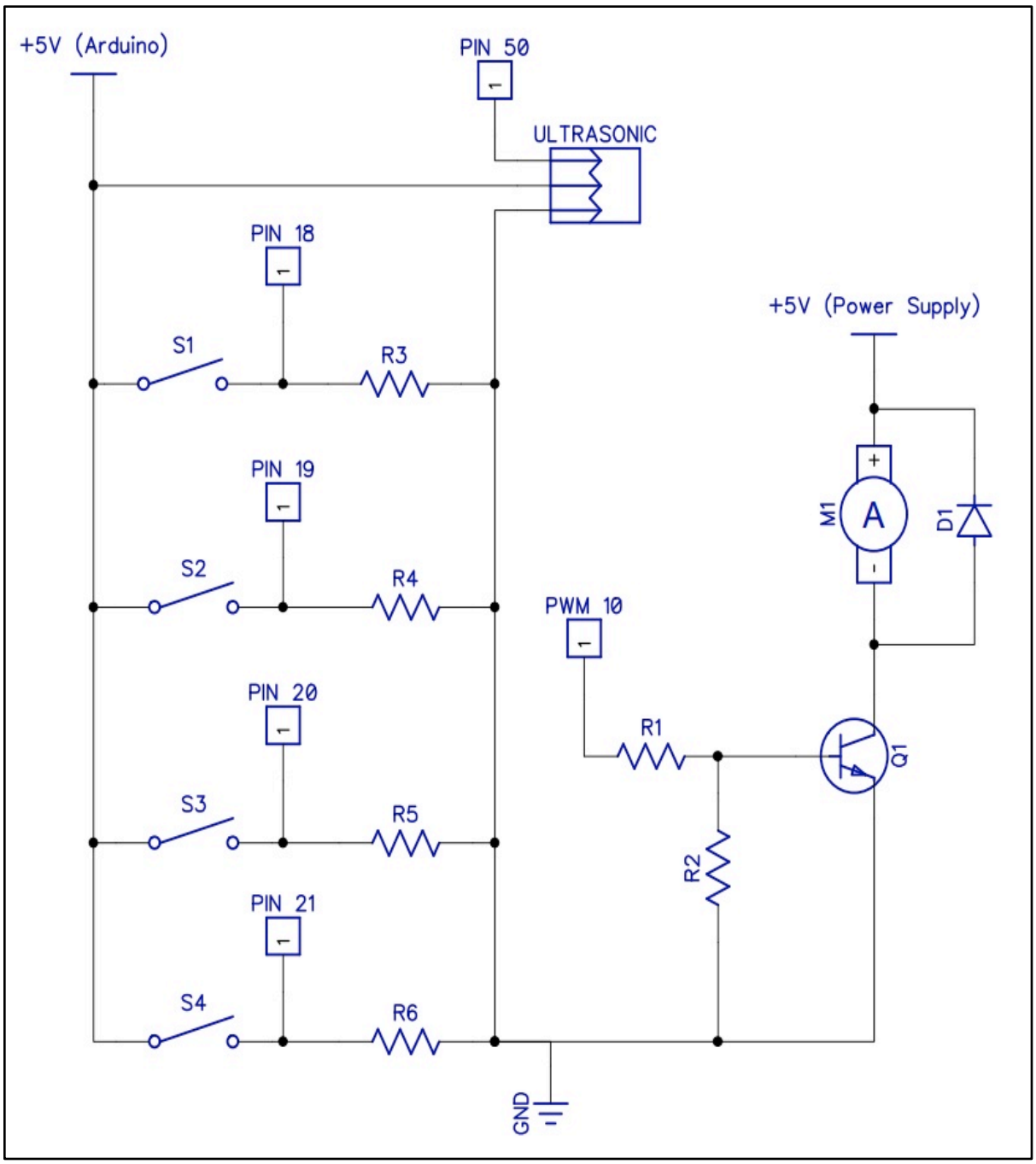

Figure 5 - Circuit

2.2 Software

The program for this project will be thoroughly described in this subsection. Note that the complete program is found in the Appendix.

Firstly, all variables must be declared appropriately in the beginning of the program so that the software allocates the appropriate amount of bytes in memory location. For example, an integer (int) requires 4 bytes, which is 16 bits, of memory. Each variable will be described along with its use throughout the description of the code.

```
int x, x0, index;
int calibrate;
double height, actual_height, max_height, previous_height;
double previous_error, previous_time, current_time, change_in_time;
double error, kp, integral, ki, derivative, kd;
double total, average_height;
double readings[num];
long duration;
bool start;
```

Interrupt functions are defined before the setup and loop functions instead of writing their prototypes before the setup and loop functions and defining at the very end of the program. This is due to the preference of the programmer. There are four interrupt functions which have no returns thus making them void functions. How interrupt functions work is that as the program is running, once the interrupt is called, the program stores its current location in the stack memory and goes to the interrupt function. Once the interrupt function is complete, the program returns to its previous location which was recalled from the stack. Considering that it is not favorable that the interrupt function be very long, a simple line of code is written in each interrupt function

which declares the desired height of the user. Function `two` declares the variable `height`, which represents the desired height of the user, to `80.0` cm. The logic behind the name of the function and its value lies in the mechanical setup of the system. Function `two` yields a physical height of 20% of the maximum height from the base where the fan lies. Since the ultrasonic sensor is located on top of the system (ie. opposite of the fan), the value the ultrasonic sensor would be the maximum height minus this desired physical heigh thus yielding the value 80. The same logic applies to the other three interrupt functions.

```
void two()
      {     height = 80.0;     }

void four()
      {     height = 60.0;     }

void six()
      {     height = 40.0;     }

void eight()
      {     height = 20.0;     }
```

For Arduino programming, the `setup` function is required and only occurs once. Here, the pins of the Arduino are declared and variables are initialized. The PWM pin 10 is attached to the MOSFET motor circuit and is set as an output through the use of the `pinMode` function. This is required so that the microcontroller can modulate the speed of the motor. Next, the `attachInterrupt` function sets up the microcontroller to deal with the interrupts which were previously mentioned. Pins 18 to 21 are used for this purpose [6]. Since buttons are used to set off the interrupt, when the button is pressed the input of the microcontroller sees a high voltage,

once the button is released the pin is grounded and held low. Due to this nature, the microcontroller is programmed to capture the `RISING` edge of the button. Note that the ultrasonic sensor will be initialized in the `loop` function as will be described later.

```
void setup()
{
      pinMode(10, OUTPUT);

      attachInterrupt(2, two, RISING);          // pin 21
      attachInterrupt(3, four, RISING);         // pin 20
      attachInterrupt(4, six, RISING);          // pin 19
      attachInterrupt(5, eight, RISING);        // pin 18
```

The variables `height`, `actual_height` and `max_height` represent the desired height of the ping-pong ball, the actual height of the ping-pong ball sensed by the ultrasonic sensor and the maximum height the ball can physically reach, respectively. The `height` variable is initialized to 40 so that the system brings the ping-pong ball to a significant height during startup.

```
      height = 40.0;
      actual_height = 0.0;
      max_height = 95.0;                  //range in cm
```

The `x` variable represents the PWM output of the microcontroller. The `x0` represents the PWM required to keep the ping-pong ball stable at any height. Since `x0` changes in accordance to temperature of the MOSFET, motor and ambient conditions `x0` is to be calibrated in the beginning of each run. It is initialized to 170 since that is near the last `x0` gained experimentally. The startup sequence, which will be discussed further on, will be maintained as long as the boolean `start` variable remains true. The exit of the startup sequence relies of the `calibrate` variable, as seen later on.

```
x = 0;
x0 = 170;
start = true;
calibrate = 0;
```

In anticipation of stabilizing the ping-pong ball once it arrives within range of the desired height, and considering the non-linear behavior of the system, a Proportional-Integral-Derivative (PID) controller is implemented to control the PWM output (`x`) of the microcontroller. Here, the PID constants, which have been calibrated through trial and error, have been initialized. A small proportional gain is desired since it will result in a small overshoot in the response of the system. The consequence of this, however, is that it results in a slower control speed. Next, a small integral gain is desired in order to reduce the settling time of the system's response. Similar to the proportional gain, this too reduces the speed of control. Lastly, a large derivative gain is desired to damped the system's response. [7] The variables `error`, `integral`, `derivative`, `previous_error` and `previous_time` are initialized here and will be used for PID calculations in the `loop` function further on. Note that Arduino contains the function `micros()` which returns the amount of microseconds that has passed since the start of the program. Although other more sophisticated control systems could have also been used, such as Fuzzy Logic PID controller or LQR, considering the short timeline and simplicity of the project, the PID control system sufficed in yielded the desired behavior.

```
kp = 0.25;
ki = 0.1;
kd = 200.0;

error = 0.0;
integral = 0.0;
derivative = 0.0;
```

```
	previous_error = 0.0;
	previous_time = micros();
```

One method in reducing noise and smoothing out the analog input data is to take the average of this data. The technique used to do so will be discussed later.

```
	index = 0;                    //index of current reading
	int num = 30
	total = 0.0;                  //current total
	average_height = 0.0;
	previous_height = 0.0;

	for(int reading = 0; reading < num; reading++)
		{readings[reading] = 0;}      //initializes all 10 samples to 0
```

Concluding the `setup` function is initializing the serial port communication. This is needed to keep track of desired values while the program is running.

```
	Serial.begin(9600);
}
```

Thus begins our entry into the `loop` function which serves an an infinite loop. The `loop` function is divided into three main sections: 1) the startup sequence; 2) averaging the data; and 3) controlling the position of the ping-pong ball.

```
void loop()
{
```

The entry of the startup sequence and the position control of the ping-pong sequence are conditional. If the system has not been calibrated, then the program is to enter the startup

sequence which begins with the line "`if (calibrate <= 9)`". Once the startup sequence is complete, the condition to enter this sequence because false and the entry into the position control becomes true with the line "`if (calibrate >= 10)`" seen further below. Within the startup sequence, the very first command is a while loop which occurs only once. This is ensured by initializing the `start` variable variable once in the `setup` function, as previously discussed. The while loop ends by making its `start` condition false, thus restricting reentry into this loop. The while loop contains an initial PWM (ie. `x`) of 255, which is the maximum value `x` can be, and outputs it the motor via the `analogWrite()` function for a duration of 2000 milliseconds. This is done to propel the ping-pong ball to a decent height before calibrating the `x0` variable which will be the PWM required to keep the ping-pong ball steady at any height. If the ping-pong ball falls below a height of 35cm, the PWM is set to output a quick high pulse so that the ball rises above this point. Similarly, if the ball goes beyond 80cm in height, the PWM is set to output a low pulse so that the ball falls below this point. Furthermore, the `x` is incremented or decremented, respectively, each time either of these situations occur to eventually yield an `x` that stabilizes the ping-pong ball. The reason for these limits is because the ping-pong ball begins to experience various rotational forces past these points making it difficult to control and making the calibrated `x0` variable incorrect for stable operation. Otherwise, if the ping-pong ball falls within the 35cm to 80cm range and that the current height, named `average_height`, is within a reasonable range of previous height recorded, named `previous_height`, the variable `calibrate` increments. Once ten consecutive recordings of height are the same, `x0` is set, the entry of the startup sequence is restricted and the target position of the ping-pong ball can now be manually selected via the buttons.

```
if (calibrate <= 9)                  // start-up sequence
{
      while (start == true)
      {
            x = 255;
            Serial.println("        Initializing        ");
            analogWrite(10, x);
            delay(2000);

            x = x0;
            start = false;
      }
      if ((previous_height < 35) )
      {
            analogWrite(10, 250);
            delay(500);
            x++;
            calibrate=0;
      }
      else if ((previous_height > 80) )
      {
            analogWrite(10, 80);
            delay(200);
            x--;
            calibrate=0;
      }
      else if (average_height <= previous_height+1
            && average_height >= previous_height-1)
      {
            calibrate++;
      }

      if (calibrate == 10)
      {
            x0 = x;
            Serial.print("        Calibrated        ");
            Serial.println(x0);
      }
}
```

Located outside of all conditional statements is the command to output the PWM (ie. `x`) to the motor via the `analogWrite` function.

```
analogWrite(10, x);
```

To keep tack of the position of the ping-pong ball, the previous height must be defined before the next current height is measured.

```
previous_height = average_height;
```

In order to reduce noise, thirty consecutive sample heights are recorded and stored in an array to then be able to take the average. A sample size, indicated by variable `num`, of thirty was selected through testing, decreasing the sample size allowed for the process to run faster however it did not reflect an accurate average and did not always account for the change in PWM value. Increasing the sample number slowed down the process because the system had to calculate a larger average of numbers before it calculates changes to the PWM value, this resulted in a very sluggish response. The sampling occurs within a while loop under the condition that the `index` variable, which is the current sample, is less than the sample size `num` [9]. Here, a specific code is required to acquire data from the ultrasonic sensor. The ultrasonic sensor behaves both as an output and an input since it works by emitting an ultrasonic sound and listens for the echo after the ultrasonic sound bounces off an object. The pin of the ultrasonic sensor must first be set as an output to produce a short `LOW` pulse to ensure a clean `HIGH` pulse which follows. Then, the same pin is set as an input to read the signal from the “PING” sound it produced when outputting the `HIGH` pulse [8]. The duration of the input can be translated into a distance through the calculation

of `actual_height` seen below. This value is stored in the array `readings` and the `index` variable is incremented by one until thirty samples have been recorded and stored thus breaking out of this while loop.

```
while (index < num)           // indexing the values to take average
{
      pinMode(50, OUTPUT);
      digitalWrite(50, LOW);
      delayMicroseconds(2);
      digitalWrite(50, HIGH);
      delayMicroseconds(5);
      digitalWrite(50, LOW);

      pinMode(50, INPUT);
      duration = pulseIn(50, HIGH);

      actual_height = (max_height - (duration / 29 / 2));

      delay(50);
      readings[index] = actual_height;
      total += readings[index];
      index++;
}
```

The thirty samples recorded have been added up together within the previous while loop yielding the variable `total`. To obtain the average height, the `total` is divided by the sample size. Then, the `total` and `index` variables must be reinitialized to zero to allow reentry into the previous while loop.

```
average_height = (total / num);
total = 0.0;
index = 0;
```

Once the average height is obtained, the amount of error between the average height, named `average_height`, and the desired height, named `height`, inputed from the button interrupts is calculated. The `x` will be adjusted via a PID controller, as previously mentioned, in accordance to amount of error which exists. The `integral` equation anticipates the future by adding itself to the amount of error which has occurred over a defined interval of time. The `derivative` equation looks at the behavior from the past by subtracting the current amount of error to the amount of error which occurred in the previous interval over a defined interval of time. This interval in time, named `change_in_time`, takes the current time from run time, named `current_time`, and subtracts it to the previous recorded time, named `previous_time`. After all necessary calculations have been made concerning the `previous_error` variable, it is redefined to store the current error for the next loop. [9]

```
current_time = micros();
change_in_time = (current_time - previous_time);
previous_time = current_time;

error = (height - average_height);
integral += (error / change_in_time);
derivative = ((error - previous_error) / change_in_time);
previous_error = error;
```

The position control of the ping-pong ball can only occur once the system is calibrated in the startup sequence, as previously mentioned. If the ping-pong ball's average height is within ±10cm of the desired height, the PWM is adjusted with use of the PID controller equation `kp*error + ki*integral + kd*derivative` to bring the ping-pong ball to steady state at the desired height. This range is large enough to account for the proportional gain's overshoot. When the ping-pong ball is outside this ±10cm range, the system is considered to be in transient state.

The approach taken for controlling the ping-pong ball in transient state is imposing pulses of high or low PWMs to move it towards the ±10cm range for the PID to take over. Therefore, if the ping-pong ball is above 10cm from the desired height, the PWM is set to 80 to slowly bring the ping-pong ball within the ±10cm range. The amount of time required for this drop depends on the current position of the ping-pong ball. When the ping-pong ball is at a very high position, it requires a longer delay to make a significant drop in height. When the ping-pong ball is at a low position, it requires lower amount of delay. Note that this characteristic of implementing a variable delay is due to the choice of having a constant value of 80 for the PWM. The variable delay has been created based on the current height multiplied by a constant which was gained experimentally. After the drop occurs, the PWM is reset to its calibrated PWM to maintain steady state in its new position. Along similar logic, when the ping-pong ball is located below 10cm from the desired height, a PWM output of 255 is implemented with a variable delay which is proportional to the current position and the PWM is reinitialized to the calibrated PWM to maintain steady state.

```
if (calibrate >= 10)                // position control sequence
{
      if ((error <= 10) && (error >= -10))      // for steady state
      {
            x += kp*error + ki*integral + kd*derivative;
      }
      else if (error < -10)         // if the ball needs to go down
      {
            analogWrite(10, 80);
            delay(average_height*1.5);
            x = x0;
      }
```

```
        else if (error > 10)    // if ball needs to go up
        {
              analogWrite(10, 255);
              delay(150+average_height*1.5);
              x = x0;
        }
}
```

As previously mentioned, when the ping-pong ball approaches the maximum height, it begins to rotate upon itself and rolls along with inner wall of the cylinder instead of floating still in the center of the cylinder. To avoid this, if the average height of the ping-pong ball is beyond 91cm, a sudden drop is to take place by outputting a low PWM to the motor for 2500 milliseconds. Similar behaviors occur when the ping-pong ball is near its minimum height. Therefore, when the current height of the ping-pong ball is less than 12cm, a PWM of 255 occurs for 1000 milliseconds.

```
if (average_height >91)
{
      analogWrite(10, 80);
      delay(2500);
}

if (average_height <12)
{
      analogWrite(10, 255);
      delay(1000);
}
```

Note that the range of PWM that is hardwired inside the Arduino is between 0 to 255. To ensure that the calculated PWM (ie. `x`) stays within this range, the following conditions have been set. If `x` goes beyond the maximum PWM of 255, `x` is then set back to 255. Moreover, if `x` goes below a PWM of 1, `x` is then set back to 1. The purpose of setting a minimum PWM of 1 is to have a

lower bound for the PWM. In most cases, however, the motor will never need to have a PWM less than 80.

```
if (x > 255)
{
      x = 255;
}
if (x < 1)
{
      x = 1;
}
```

The following code is serial port communication. It helps keep track of the values during experimentation.

```
if (calibrate < 10)
{
      Serial.print(actual_height);
      Serial.print("  cm  act      pwm: ");
      Serial.print(x);
      Serial.print("     Calibrate: ");
      Serial.println(calibrate);
}

if (calibrate >= 10)
{
      Serial.print(actual_height);
      Serial.print("  cm act        ");
      Serial.print(height);
      Serial.print("  cm  set      pwm: ");
      Serial.println(x);
}
```

Lastly, a delay of 10 milliseconds is required before looping back to the beginning of the `loop` function.

```
        delay(10);
}
```

2.3 Result

The system has the following behavior and requirements:

- The system requires a cold start warm up of 10 minutes
- The system requires a calibration sequence to establish the PWM required to keep the ball steady at one position (which is initiated on startup)
- The system takes 60 seconds to achieve the change in desired position
- The system is intended for constant operation no greater than 20 minutes once warmed up
- Accuracy of ± 4cm is expected, this represents 2cm range plus the 2cm radius of the ball.

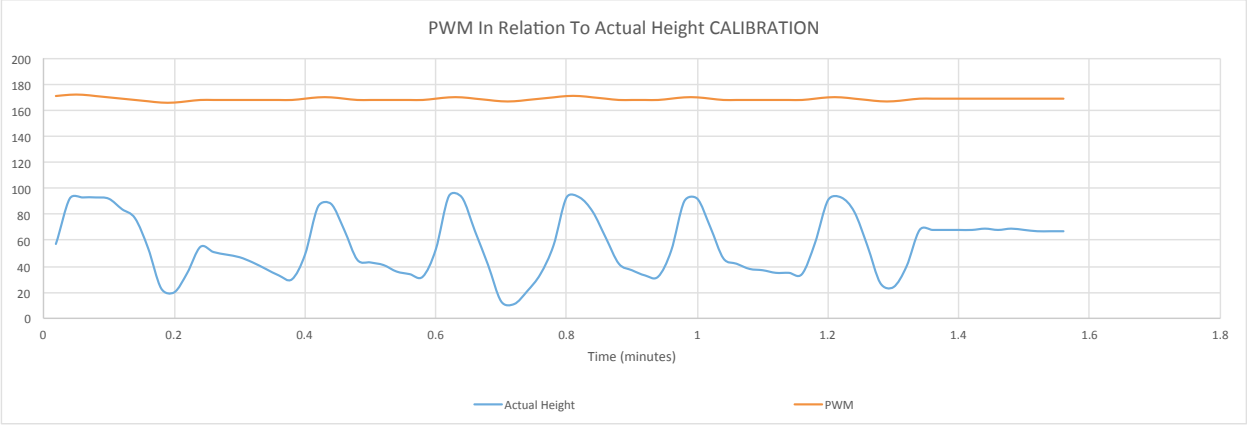

Figure 6 - PWM In Relation To Actual Height CALIBRATION

When the system is first started, it enters the program's startup sequence whereby initialization and calibration processes occur. As seen in Figure 6 the height of the ball varies significantly based on slight PWM changes, after a stable height is achieved the PWM value is recorded as a

baseline. After this the ping pong ball goes to the initialized set height of 40. The ball drops or rises in steps until it is within its 10cm range of the set height, this allows the ball to get to its desired height more efficiently because the transient response is different than that of the steady-state. This is primarily due to the inertia the ball has when falling from a set height and the flow in the tube when the ball is rising up. After the ball is within its range the PID equation takes control of the PWM and accurately adjusts itself based on error. At any point before or after the system stabilizes the user can press one of the other buttons to change the height that the set point is move to. The system generally gets close to its desired set point quickly and then it operates at a slower rate to ensure a steady output is achieved. The non linear behavior of the ball in a cylinder restricts the speed of the response. In other words, the system's response is oscillatory while attempting to attain steady-state. The amount of oscillation was reduced using a PID controller to control the PWM, thus creating a faster settling time. It is possible to achieve all four set points that are programmed.

A representation of the system's behavior is illustrated in Figure 7 once calibrated. The system is set to go to a desired height of 20cm. The ping-pong ball, at approximately 40cm, drops to be within the range of ± 10cm of the set 20cm. Here, it oscillates for a short period of time until it's within ± 4cm of the desired height. Next, the system is set to a desired height of 60cm. The ping-pong ball quickly ascends and oscillates for a short period of time until it's found within ± 4cm of the desired height. Similarly for when the desired height of 20cm is selected.

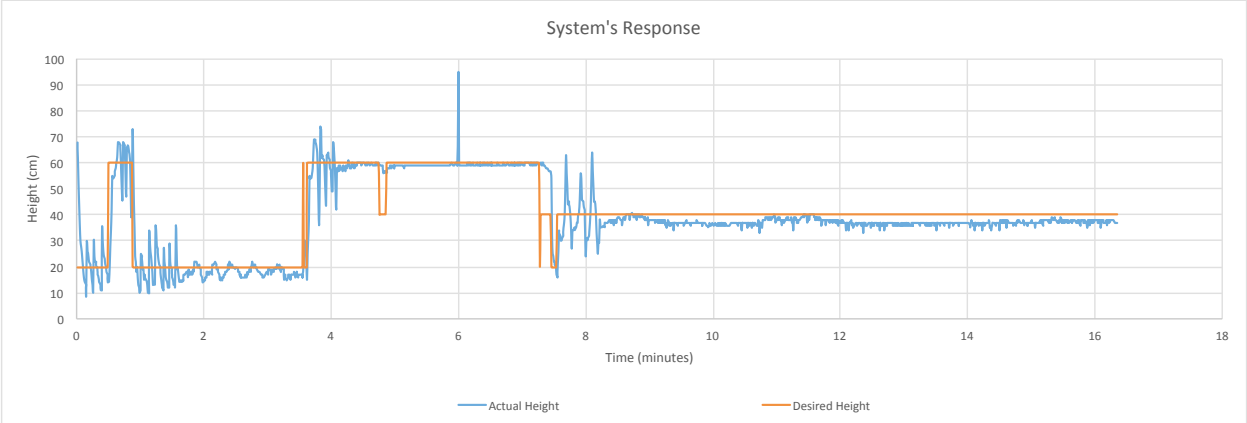

Figure 7 - Actual vs. Desired Behavior once calibrated

The change in PWM is illustrated in Figure 8. The PWM oscillates accordingly to the PID controller when the ping-pong ball is within ± 10cm of the desired height. The proportional gain controls the amount of overshoot, and in turn oscillation, that exists in the system. The integrator attempts to bring the system to steady state earlier. The derivative attempts to dampen the system's response to prevent as much oscillation. These PID constants changed in accordance to how long the system has been running. For this set of data, the system had been running for over an hour which might explain the relatively long amount of time taken before reaching steady state of ± 4cm of the desired height. The graph also demonstrates that once the desired height is reached, the PWM remains constant to keep the ball afloat at the set position.

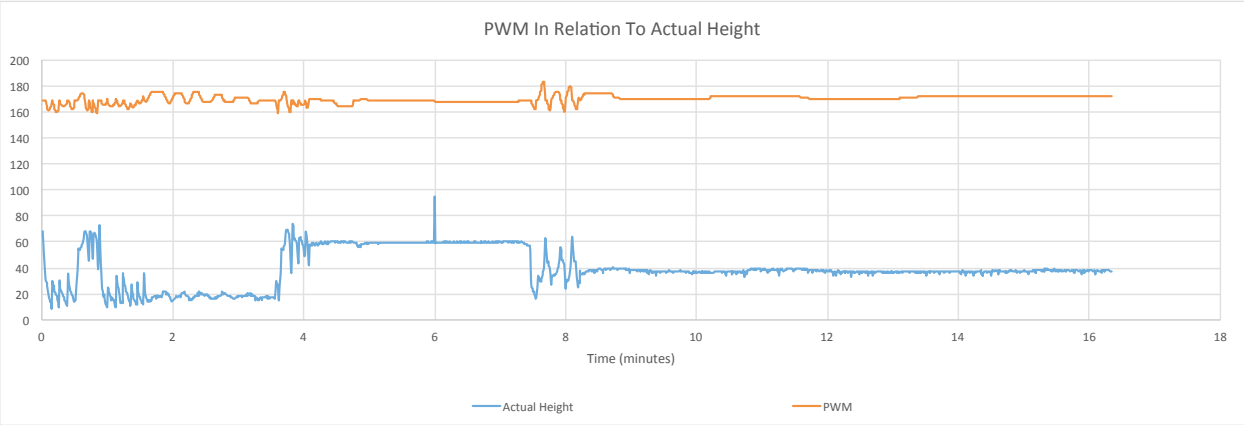

Figure 8 - PWM in relation to actual height

Some difficulties arise was trying to get the upper and lower limits of the system since the system's characteristics appear to change, such as the ping-pong ball spinning and twirling around the inner diameter of the tube. To avoid such problems, limits were set before these thresholds to keep the ball within a workable range as previously mentioned. However rare, it is possible for the ball to get stuck in the upper or lower limit and the system needs to be restarted.

As perviously mentioned, the accuracy of the ball is within ± 4cm is expected. Possible reasons for this is that there are round off errors when implementing PID to the PWM calculation. When error is small, the PWM's decimal numbers change. These decimal values, represented in C++ as a `double`, are omitted when outputting the PWM as an `int` thus causing a 2cm error range. Additionally, the movement of the ping-pong ball along the interior wall of the cylinder my cause additional error when recording the actual height of the ping-pong ball. Furthermore, the curvature of the ping-pong ball may yield small errors as well. Therefore an additional 2cm range, representing the radius of the ping-pong ball, is added to the error giving an overall error of 4cm.

3. CONCLUSION

The project is deemed successful in completely it's goal of moving the ping-pong ball to a desired location. This was achieved by using a PID controller to bring the ball to steady state at a specific position and short pulses of air controlled the ping-pong ball during transient state. These decisions were made through trial and error of applying different methods to achieve the same goal. Initially, a linear control for PWM was implemented. This yielded an unfavorable response of positioning the ping-pong ball. The ping-pong ball stayed at the minimum height while the PWM incremented. Then suddenly the ping-pong ball would launch itself to the maximum height and begin spiraling uncontrollably. This is where we learned that a limited range of PWM is to be outputted to keep the ping-pong ball afloat and move it slightly upwards or downwards. Thus a PID is used to move the ping-pong ball within this small PWM range until it reaches the desired position. Additionally, since PWMs outside this region move the ping-pong significantly upwards and downwards, small pulses of air was implemented a small yet abrupt change in position before becoming balanced again. It is necessary for the ping-pong ball to stay balanced occasionally to give time for the ultrasonic sensor to pick up its current position accurately. Averaging was necessary to reduce noise and make input data, being distance from the ultrasonic sensor more reliable. Without averaging, the ultrasonic sensor would not take into account that the ball was overcoming airflow forces to change heights, the response between the fan speed and having an accurately controlled rise or descent was very tricky within the tube, averaging ensured that an accurate position was being used and compared for error calculations.

Further work includes improving the transient state of the system. The short burst of air occasionally overshoots the ping-pong ball outside of the ±10cm range of the desired height required for the condition to execute the PID controller to be met. Moreover, the PID controller can also better fine tuned to reduce the amount of oscillations thus bringing the system to a steady state at a specific position faster. Other control methods include changing the PID constants Kp, Ki and Kd based on error since to this system behaves similarly to a 4^{th} order system thus requiring a higher level of control.

Because the system needs to operate smoothly, another possible solution to better control the stability of the object is to implement a simple resistor capacitor circuit (RC) to smooth out the operation of the motor. Other changes include improving object stability by using a more stable shape, injecting a fluid into the ping pong ball or removing the system from the confinements of the cylinder and creating an enclosure that would not create airflow issues. Finally the addition of a measuring device to ensure the accuracy of the ultrasonic sensor should be implemented.

REFERENCES

[1] "Bernoulli's Principle Demos," Mr.Kent, 2014. [Online] Available: http://teacherweb.com/MA/SERSD/Kent/apt51.aspx

[2] Aliya Merali, "Floating Ping-Pong Balls," Physics Central, 2015. [Online] Available: http://physicscentral.com/experiment/physicsathome/ping-pong-physics.cfm

[3] "Fluid dynamics and Bernoulli's equation," 1999. [Online] Available: http://physics.bu.edu/~duffy/py105/Bernoulli.html]

[4] Tom Benson, "Ping-Pong Ball curves," NASA, 2014. [Online] Available: http://www.grc.nasa.gov/WWW/k-12/TRC/Aeronautics/Ping_Pong_Curve.html

[5] "H-Bridge Circuitry and Flyback diodes," NorthWestern. [Online] Available: http://mechatronics.mech.northwestern.edu/design_ref/actuators/hbridge_circuitry.html

[6] "attachInterrupt()," Arduino. [Online] Available: http://www.arduino.cc/en/Reference/attachInterrupt

[7] Sigurd Skogestad, "Probably the best simple PID tuning rules in the world," Norwegian University of Science and Technology, 2001.

[8] "Ping Ultrasonic Range Finder," Arduino. [Online] Available: http://www.arduino.cc/en/Tutorial/Ping?from=Tutorial.UltrasoundSensor

[9] "Smoothing," Arduino. [Online] Available: http://www.arduino.cc/en/Tutorial/Smoothing

APPENDIX

```
int x, x0, index;
int calibrate;
const int num = 30;                                    // was 10
double height, actual_height, max_height, previous_height;
double previous_error, previous_time, current_time, change_in_time;
double error, kp, integral, ki, derivative, kd;
double total, average_height;
double readings[num];
long duration;                         //try int**********
bool start;


void two()
{                                                      // Because the Ultrasonic sensor is opposite to
the fan
        height = 80.0; //70.0;  ///(8 / 10)*(max_height);          // 20% of max height is 80% from the
top down
}


void four()
{
        height = 60.0; //52.0; // (6 / 10)*max_height;          // 40% of max height is 60% from the
top down
}
```

```
void six()
{
	height = 40.0; // 35.0; // (4 / 10)*max_height;		// 60% of max height is 40% from the
top down
}

void eight()
{
	height = 20.0; //17.0;  // (2 / 10)*max_height;		// 80% of max height is 20% from the
top down
}

void setup()
{
	pinMode(10, OUTPUT);			// pwn output

	attachInterrupt(2, two, RISING);	// pin 21
	attachInterrupt(3, four, RISING);	// pin 20
	attachInterrupt(4, six, RISING);	// pin 19
	attachInterrupt(5, eight, RISING);	// pin 18

	height = 40.0;
	actual_height = 0.0;
	max_height = 95.0;			//range in cm
```

```
x = 0;
x0 = 170;
start = true;
calibrate = 0;

kp = 0.25;                    // was 0.25 .. and before that was 0.1
ki = 0.1;
kd = 200.0;                   // was 200

error = 0.0;
integral = 0.0;
derivative = 0.0;

previous_error = 0.0;
previous_time = micros();

//arduino.cc/en/Tutorial/Smoothing

index = 0;                    //index of current reading
total = 0.0;                  //current total
average_height = 0.0;
previous_height = 0.0;

for (int reading = 0; reading < num; reading++) readings[reading] = 0;      //initialises all 10
samples to 0
```

```
        Serial.begin(9600);                 // helps you keep track of the values
}

void loop()
{

        if (calibrate <= 9)
        {
                while (start == true)
                {
                        x = 255;            //initial input
                        Serial.println("       Initializing       ");
                        analogWrite(10, x);
                        delay(2000);

                        x = x0;
                        start = false;
                }

                if ((previous_height < 35) )
                {
                        analogWrite(10, 250);
                        delay(500);
                        x++;
```

```
                calibrate=0;


        }
        else if ((previous_height > 80) )
        {
                analogWrite(10, 80);
                delay(200);
                x--;
                calibrate=0;
        }
        else if (average_height <= previous_height+1 && average_height >= previous_height-1)
calibrate++;


        if (calibrate == 10)
        {
                x0 = x;
                Serial.print("       Calibrated       ");
                Serial.println(x0);
        }
    }


    previous_height = average_height;


    analogWrite(10, x);
```

```
        while (index < num)      //0-9
        {
                pinMode(50, OUTPUT);
                digitalWrite(50, LOW);          // Give a short LOW pulse beforehand to ensure a clean
HIGH pulse
                delayMicroseconds(2);
                digitalWrite(50, HIGH);         // The PING is triggered by a HIGH pulse of 2 or more
microseconds
                delayMicroseconds(5);
                digitalWrite(50, LOW);

                pinMode(50, INPUT);             // The same pin is used to read the signal from the PING
                duration = pulseIn(50, HIGH);
                actual_height = (max_height - (duration / 29 / 2));

                delay(50);
                readings[index] = actual_height;
                total += readings[index];
                index++;
        }

        average_height = (total / num);         //calculates average of 10 heights
        total = 0.0;
        index = 0;
```

```
// time delay might hinder time based response, majority kp control
current_time = micros();
change_in_time = (current_time - previous_time);
previous_time = current_time;

error = (height - average_height);
integral += (error / change_in_time);
derivative = ((error - previous_error) / change_in_time);
previous_error = error;

if (calibrate >= 10)
{

        if ((error <= 10) && (error >= -10))                    // for steady state
        {
                x += kp*error + ki*integral + kd*derivative;    //pwm can only have 0-255 (int)
        }


        else if (error < -10)                   // if the ball needs to go down
        {
                //analogWrite(10, (x0-40));                     //want to make smoother
                analogWrite(10, 80);
                delay(average_height*1.5);                      // was 150
                x = x0;
```

```
        }


        else if (error > 10)          // if ball needs to go up
        {
                //analogWrite(10, (x0 + 50));                   //want to make smoother
                analogWrite(10, 255);
                delay(150+average_height*1.5);
                x = x0;
        }
        if (average_height >91)
        {
        analogWrite(10, 80);
        delay(2500);
        }
        if (average_height <12)
        {
        analogWrite(10, 255);
        delay(1000);
        }
}




if (x > 255)
{
```

```
            x = 255;
}
if (x < 1)
{
            x = 1;
}

if (calibrate < 10)
{
            Serial.print(actual_height);
            Serial.print("  cm  act      pwm: ");
            Serial.print(x);
            Serial.print("     Calibrate: ");
            Serial.println(calibrate);
}

if (calibrate >= 10)
{
            Serial.print(actual_height);
            Serial.print("  cm act        ");
            Serial.print(height);
            Serial.print("  cm  set      pwm: ");
            Serial.println(x);
}
```

```
        delay(10);


        //Serial.println (x);                   // control + shift + m
}
```