Politecnico di Milano

AIR FLOW BALL LEVITATION AND LIGHT CONTROLLER

Project for the course
Computing Systems for Engineering Physics

Giovanni Masciocchi
Ariana Mirshokraee Seyed
Stefan Šušnjar

A.Y. 2018/2019

Table of Contents

1 INTRODUCTION

The purpose of this report is to describe a group project we have realized for the course Computing Systems for Engineering Physics during the academic year 2018-2019 at Politecnico di Milano.

We were asked to design and realize a simple embedded system using a prototyping board such as Arduino Uno.

Our idea is to use cheap electronic components and the open source controller to control the position of an object (plastic ball) inside a transparent tube. The position will control the intensity of an LED providing a "visual" feedback.

The interesting part is to implement a generic PID model for automation control in a relatively simple physical model. This control approach is indeed widely used in many kinds of systems and with different controllers.

Despite the simplicity of our system, the usage of cheap sensors and actuators, has request some coding strategies to solve issues related to precision, stability and speed in order to reach a good compromise.

In the following pages we will start presenting the hardware of our project and then, after some theoretical recalls about the PID control law, we will highlight some features of our Arduino code.

The code will be given as an appendix after the results of the operating testing.

2 BUILDING THE PROTOTYPE

Before start programming using the Arduino board, was needed to design and realize the structure (hardware) capable to make the controller interact with the physical word. We have realized the prototype form zero, starting from the electrical components and plastic and wood parts.

2.1 AN OVERVIEW

The entire system is schematized in Figure 1. The Arduino board (6) has two sensors (4-1) to detect the position of the ball in the plastic tube and of the hand next to it. The actuation is done using an electric fan (7) that blows air in the tube. The LED (5) changes the intensity according to the position of the ball in the tube. A greed LED (2) indicates the lamp is on, and a button (3) is used to turn on the system.

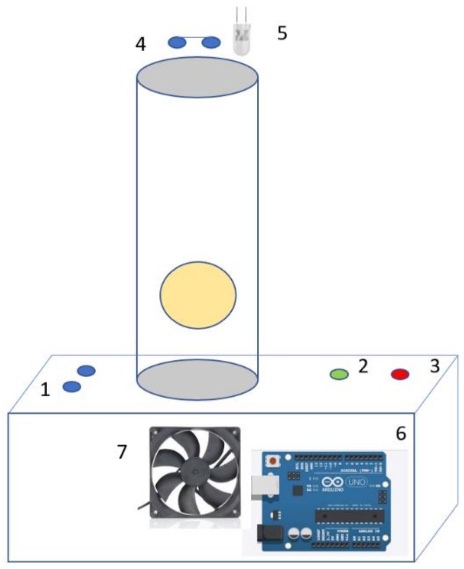


Figure 1 – System with its main components

The goal of the controller is to set the ball in the same (vertical) position of the user hand with a good precision and reasonable speed. The result is a "touchless lamp".

We now describe into details the used components explaining the reasons of the choice and the specifications.

Let us start with the sensors. The physical quantity we have to measure is the position (distance) of a moving object. We require a range from $2\,\mathrm{cm}$ to $100\,\mathrm{cm}$, speed of $40\,\mathrm{Hz}$ and a resolution of at least $0.5\,\mathrm{cm}$. Of course, the cost should be as low as possible.

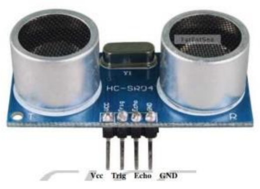


Figure 2 – Ultrasonic sensor HC-SR04

We decided to use the ultrasonic sensor HC-SR04 (Fig. 2). It is made of two circular parts which contain, respectively, an emitter and a receiver sensitive to ultrasound ($40\,\mathrm{kHz}$) frequencies. It requires a supply of $5\,\mathrm{V}$ and so can be directly connected to the Arduino.

The sensor emits an ultrasound pulse and waits for the same signal to come back when an object in the sensor range reflects the wave. What is measured is the flying time in microseconds. To convert this into a distance we use the formula

$$d[\mathrm{cm}] = 0.03434 \cdot \frac{t[\mu\mathrm{s}]}{2}.$$

The speed of sound is assumed constant and equals to $343{,}4\frac{\mathrm{m}}{\mathrm{s}}$ at $20\,°\mathrm{C}$. The factor two at denominator is because the sound wave travels twice the distance between sensor and object.

This sensor is quite cheap, with price less than 3 \$, and has a range of $(2-400)\,\mathrm{cm}$. For what concerns the speed, in order not to overlap two measurements, it is sufficient to wait the signal returns before emitting the next one. Interval of $10\,\mathrm{ms}$ is far above the time required to perform our short distance measurements.

Of course, due to the reduced costs, the possibility of wrong values can be an important problem for our application, since the code relies just on the data provided by sensors. As will be further discussed, we have improved precision averaging values and imposing some conditions (e.g. too large time values…) in the code.

This particular sensor has been used both to sense the position of the user's hand and the position of the ball in the tube.

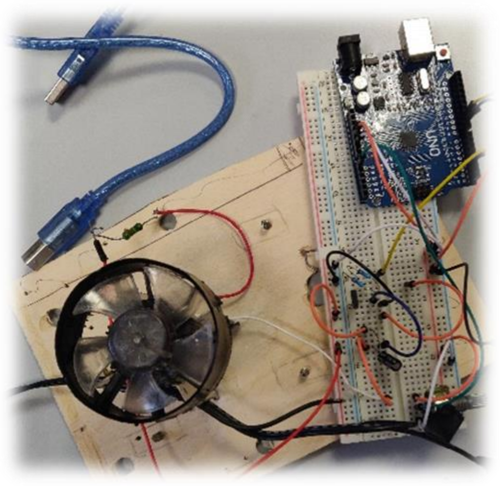


Figure 3 – Hairdryer fan connected to a supporting circuitry

To reach our goal, the control of the position of a ball, we also need actuators. To "beat" gravity the pressure of a forced air flow on the bottom of the ball is used. To do so we used a quite powerful fan taken from a hairdryer (Fig. 3).

The speed of rotation is proportional to the DC voltage $(0-32)\,\mathrm{V}$ applied; by controlling the voltage we control the velocity and then the upward force impinging on the ball, which mass is $40\,\mathrm{g}$.

Since Arduino can supply $5\ \mathrm{V}$ and few tens of mA of current, we used a DC power supply capable of providing $32\ \mathrm{V}$ and up to $700\ \mathrm{mA}$ of current. The easiest way to set a variable voltage with Arduino Uno is using a PWM port. With a PWM port it is possible to select a voltage from 0 to $5\ \mathrm{V}$ on 255 levels and provide it to a specific pin.

This voltage will drive the base of a BJT transistor and will control the voltage on the fan itself. The schematic is shown in Figure 4. The resistor R4 limits the current flowing in the motor, the capacitor and the diode are placed to avoid peaks and reverse current flows. The resistor R3 is connected to the PWM port of Arduino.

Due to the high current flowing into the circuit, the resistor R4 was made by the parallel of two $25\ \mathrm{W}$ resistors. This choice was made to avoid overheating.

We used an NPN bipolar junction transistor (type MJE 800) capable of resisting up to $60\ \mathrm{V}$ (collector base voltage) and $1.5\ \mathrm{A}$ of current in the collector.

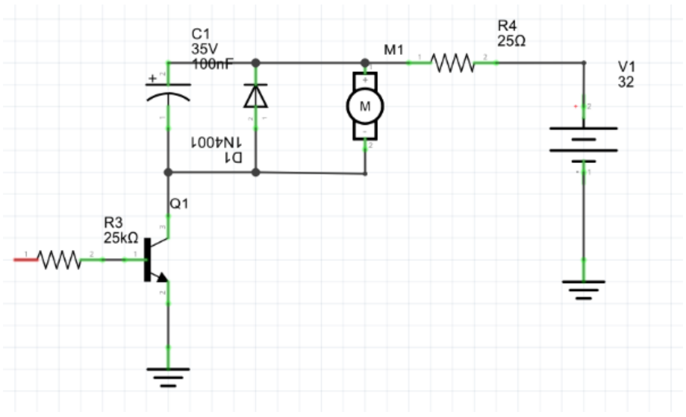


Figure 4 – Electrical circuit that controls the voltage on the fan motor

We said the position of the hand controls a lamp. The light source we decided to use is a series of RGB LEDs. The light produced is white (all the tree LEDs are on) and its intensity is controlled by the position of the ball inside the tube. In this case, as in the previous, we used an NPN transistor connected to the PWM port of our controller: the scheme is similar to the one used for the motor.

To supply our Arduino board and the LED lights we decided to use a simple $9\ \mathrm{V}$ battery. In this way we can supply both the Arduino Uno and the RGB LED array placed on the top of the tube.

The prototype is shown in the Figure 5. It is possible to see the position of the two sensors and the Arduino board.

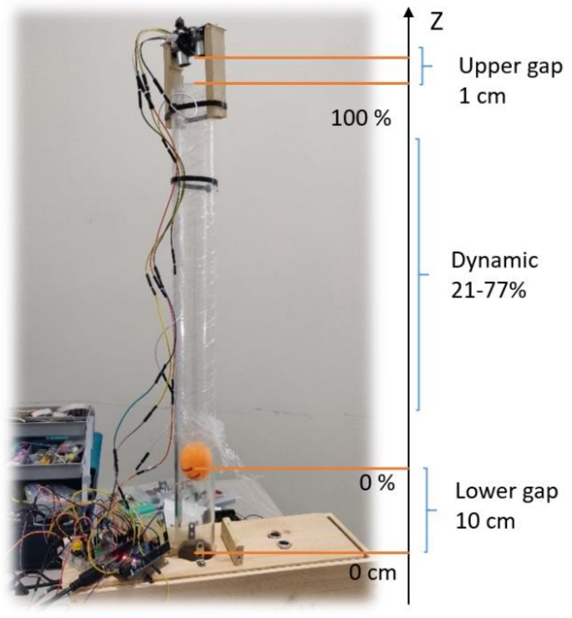


Figure 5 – Prototype: tube, sensors, Arduino board (fan is inside the wooden box); on the right side of the image, reference levels and gaps are pointed out

The object to control is a ping pong ball (diameter of $40\ \mathrm{mm}$ and mass of $40\ \mathrm{g}$) inside a tube whose diameter is $44\ \mathrm{mm}$ and length $50\ \mathrm{cm}$. The air comes from below and, since the air flux is turbulent in the lower part of the tube, we decided to define an operating zone for our system (dynamic, 38.5 cm). The flow is laminar in the central part of the tube so that is the part we want to control our object.

In the scheme (Fig. 5), it is possible to visualize the axis we used as a reference for the position of the ball and the hand. Indeed, the data from the sensor will be scaled in the code to be compared and processed. In particular, the position (in cm) has been converted in percentage values so that to make easier driving the motor speed.

After the built, we were ready to implement and test the code, which will be described in the next paragraph.

2.2 Sensors test and precision

The sensors are the way our controller gets in touch with the real word and having reliable information is crucial for our controller. If a false data about the position (both of the hand or of the ball) is read, our control system will perturb the system (using the motor fan) too much or not enough.

As an example: if the ball gets close to the sensor, below the minimum range, the sound wave cannot reach the echo pin and very large time values (the limit one, set by the producer) will be given as output. This means the ball is supposed to be far from the reference (since it is far from the sensor) and a bigger voltage is applied: the ball sticks up.

We collected some sample data at known distances to understand how good our sensors are. What we would find not satisfactory, will be corrected and processed by the software between the data collection and the actuation voltage control e.g. by averaging or by imposing an existence range.

Some collected data are shown in the graphs below (Fig. 6 and Fig. 7).

In the first one (Fig. 6) the position of the ball has been measured at rest i.e. placed on the bottom of the tube. As we can notice, the results

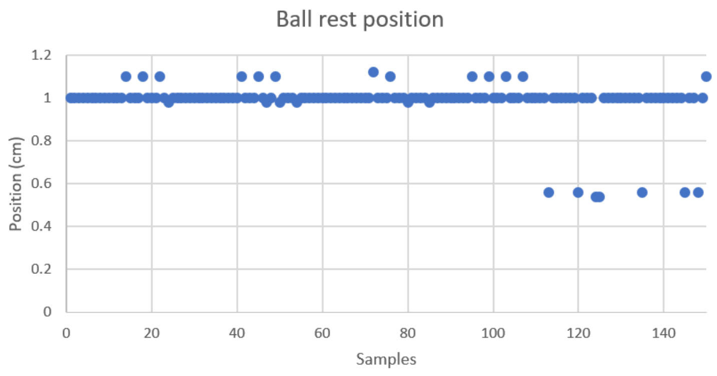


Figure 6 – Ball position at rest – measured data

are scattered around the value 1 [cm] and some missed readings are present corresponding to the points at 0.54 [cm]. This means that our sensor is quite accurate, but precision should be improved as an example, averaging the collected data.

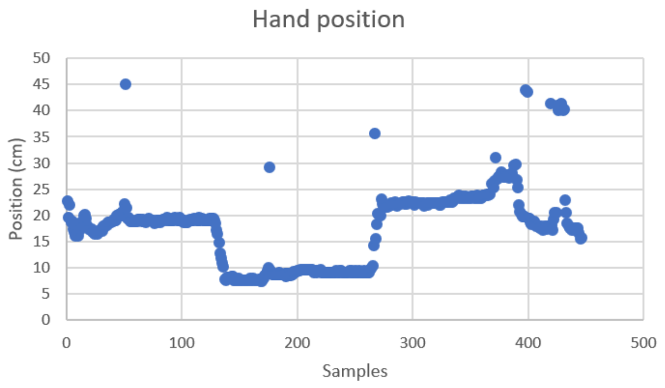


Figure 7 – Hand position measurement

A similar comment can be made about the second graph (Fig. 7). In this case the hand was moved up and downward and its position has been measured. Again, some false values are evident (e.g. a negative distance).

3 BALL POSITION REGULATION

By construction, our control system has one input – voltage sent to DC motor (actually, to the base of the BJT) and one output – position of the ball (on vertical axis). So, we have a single input single output system, which is by its nature non-linear, and it would require detailed physical analysis to be described accurately with mathematical expressions. This is very difficult in practice and eventually the outcome, which might not be satisfactory with the given hardware, even if we knew the explicit model of the system. That is the reason why in this case it is much more convenient to design a linear controller which would be tuned experimentally, without any knowledge of mathematical model of the system. The aim of the controller is to provide the appropriate voltage to the motor of the fan which would, finally, lead the ball to the desired (specified) reference position and keep it steady as much as possible. In theory, we would like to have zero error (difference between the desired position, i.e. reference and current position of the ball) in steady-state and also the quickest possible and stable transient (without big oscillations).

3.1 PID CONTROL LAW

To fulfil all the requested points, we implemented the feedback control law with three parallel actions: proportional, integral and derivative. It is known from system control theory that integral action is needed if one wants to have zero error in the steady state. Also, derivative action is sometimes useful to stabilize the system if it happens to have oscillatory behaviour that cannot be controlled in other ways. Proportional action is obviously the most intuitive one and it is important to drive the ball in the early stage of transient, when error is significant. We will give the explicit expressions in the following lines. Firstly, let us define r – reference (desired ball position), y – ball position, both as numbers between 0 and 100% (scaled in correspondence with the allowed region of ball position in tube). The error is defined as follows $e = r - y$.

Total control u, i.e. voltage provided to the DC motor, is defined as:

$$u = P + I + D,$$

and it expresses the percentage of maximal control that can be applied (considering all physical limitations).

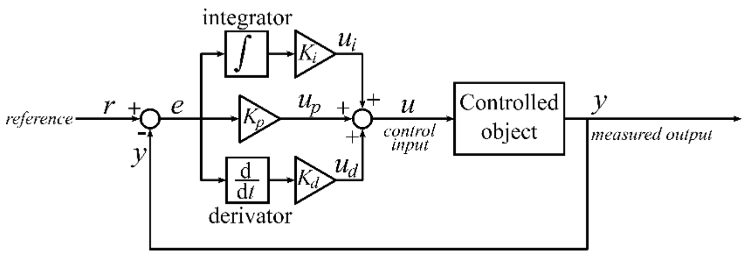


Figure 8 – Block scheme of parallel PID controller for continuous time systems

In the former expression, P is proportional part, I is integral part and D is derivative part and they are calculated in the following way, at certain time instance t:

$$P = K_p \cdot e(t),$$

$$I = I_0 + K_i \int_0^t e(\tau)\mathrm{d}\tau,$$

$$D = K_d \frac{\mathrm{d}e(t)}{\mathrm{d}t},$$

where constants K_p, K_i and K_d are to be tuned to obtain the best possible control and I_0 is the initial condition of the integrator.

3.2 Implementation of PID on our system

As we are using Arduino to produce the desired control voltage, all operations are executed digitally, and the final output is sent through the Arduino PWM port. This means that theoretical PID control law is implemented as a simple summation of three components, which are computed in discrete time instants. Interval between the computations is called sampling time T_s. During each of these intervals, the computed output of the PWM port remains constant. Update of a control value, and therefore the change of the output of the PWM port occurs after each sampling period. Let us denote $u[k] = u(k \cdot T_s)$, for any discrete time moment $k \cdot T_s$, where k is an integer, and analogously we define $P[k]$, $I[k]$ and $D[k]$. It is obvious that in each iteration of PID control value update, Arduino must compute $u[k] = P[k] + I[k] + D[k]$, and before that the values of all the addends. If the current values of the reference and the ball position are stored in memory as $r[k]$ and $y[k]$, respectively, then the error is calculated directly: $e[k] = r[k] - y[k]$. The following operations are to be executed: $P[k] = K_p \cdot e[k]$, $I[k] = I[k-1] + K_i \cdot e[k] \cdot T_s$, and $D[k] = K_d \frac{y[k-1]-y[k]}{T_s}$. The last expression suggests that we also need to store some previous values of ball position or other measurements of the system. As a whole, last three expressions represent a conversion from analog to digital domain. The discrepancy between the expressions for $D[k]$ and $D(t)$ is intentional and it will be explained in the end of the following paragraph.

For our system, one particular implementation is described with the program code, but there are three main features which are additionally considered and realized, and we are going to mention them at this point.

First, the region where the ball can move is divided in three sub-regions, each of them having a specific PID controller with different parameters (K_p, K_i and K_d). This is important because our system is non-linear and PID is a linear controller, so it would work in the best way only around the operating point for which it was designed, or in other words, for which the parameters were fine-tuned. Further from the chosen operating point in the design phase, inappropriate behaviour may be observed and that is the reason why we have decided to have three operating points at different heights for parameter tuning, thus having specific controllers for different regions of the tube.

Practically, the change is only in parameters and it is completely done in software code, the principle of operation is the same from the point of view of hardware.

Secondly, whenever there is an integrator, there is a possibility to have a so-called integrator wind-up, which is undesirable and is to be prevented. The solution we implemented is the best in cases where it is possible to compute the control digitally which is the case with Arduino and our code. It completely solves the problem of integrator wind-up. One should notice that even if the required control is greater than 100% (or less than 0%), it cannot be realized. However, we dealt with the integrator initial conditions in these cases to prevent storage of bad values (those not in range $(0-100)\%$) in integrators that could affect the future behaviour of the whole control system. The solution is simple: if the computed total control becomes greater than 100% or less than 0% than fix it and do not integrate during the last period, or in digital version, do not sum up the latest sample's integral component. With this anti-wind-up implementation, our system is guaranteed to work in real time without false computations.

Third implemented feature is so-called bump-less transition between different regions, where the PID parameters are different. If we hadn't taken care about that, we would have had step-like changes in computed control and it is also to be avoided, because it might introduce unwilling oscillations. Our code solves that problem too, by setting the appropriate value of initial condition to integrator immediately after the ball leaves one sub-region and enters another.

Finally, additional thing which is useful for calculation of derivative part is that when the step-like change of reference appears, then large value of derivative of error is to be expected (in theory infinite, in practice large, but finite, because of the finite difference method used to calculate the derivative of error). The simple way to solve this problem is just to neglect the reference part of the error derivative, writing it in approximate form $\frac{\mathrm{d}e(t)}{\mathrm{d}t} = \frac{\mathrm{d}(r(t)-y(t))}{\mathrm{d}t} \approx -\frac{\mathrm{d}y(t)}{\mathrm{d}t}$.

3.3 Practical realization – sensors and actuator

Information about the ball position is provided by the ultrasonic sensor (described in 2.1) and is expressed as a percentage of height from zero level (ball is at the lowest possible position) to maximum level (upper end of the tube is $38.5\,\mathrm{cm}$ above the zero level). Information about the reference level is obtained by calculating the average of the last three samples received from another ultrasonic sensor and expressed as a percentage of height from the same zero and maximum levels as for the ball position. One sampling period lasts for $T_S \cong 0{,}061\ \mathrm{s}$, and it is considered for computation of required control, although it might differ slightly (on the third decimal), depending on unpredictable hardware delays due to execution of operations. All measurements from sensors are stored in memory, used for computations, and, if needed, kept for next few sampling intervals. Value that is finally computed and defined as a control voltage is sent to Arduino PWM output which controls the base voltage of the BJT (as described in 2.1). During the time interval for which the PWM output is at high level, the transistor will amplify the base current, thus providing the voltage drop between the pins of the fan motor. For the rest of the cycle, base voltage is equal to zero and there is no collector current, thus forcing the fan motor voltage to drop to zero. Voltage transients are not instantaneous, and the system properties (such as electrical circuit time constant, fan inertia, etc) are all considered when referring to the non-linearity of a system. Bearing in mind that the mechanical subsystem time constant is not negligible, and that the sensor acquisition frequency is limited, certain limitation to the sampling time is necessarily imposed. Our sampling time must not be smaller than $50\ \mathrm{ms}$, only by considering sensors, but also, we have to take into account other operations that are executed by Arduino. With very big sampling time intervals it could be impossible to have a stable regulation. However, there is also no much sense in making it smaller than it is, because of the response time of the mechanical subsystem.

4 Writing the Code

Function of the written program is to get the distances between the hand and the hand's sensor (reference distance) and between the ball and the ball's sensor, and then by using the PID control system for controlling the power of the fan, "bring" the ball at the desired distance from the bottom of the tube.

We will also explain how the code is used to print some information on LCD, synchronize the brightness of a LED with the position of the ball and use a button to turn on/off the system.

4.1 Overview

In the first lines of the code, we have defined the constants of the program. Most of these constants resulted from the physical shape of our device. In particular, the constants used in the PID part have been tuned experimentally according to the Ziegler-Nichols method as a starting point.

Then, we have defined the input and the output ports of system. It means that, we assign certain Arduino's ports to each component of the hardware. The pin configuration is shown in the picture below.

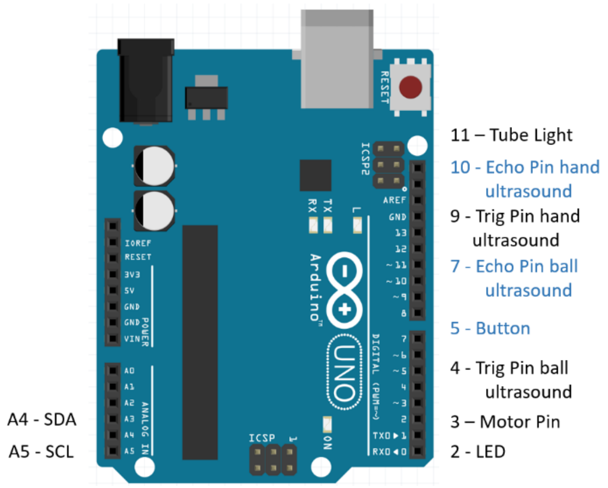
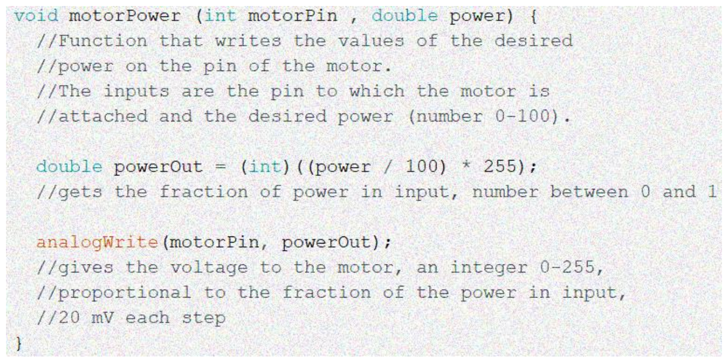

11 – Tube Light
10 - Echo Pin hand ultrasound
9 - Trig Pin hand ultrasound
7 - Echo Pin ball ultrasound
5 - Button
4 - Trig Pin ball ultrasound
3 – Motor Pin
2 - LED
A4 - SDA
A5 - SCL
Arduino
UNO

Figure 9 – Pin configuration on the Arduino controller; blue text refers to input pins, while the black text designates the outputs

4.2 Functions

There are some functions which are used in the code for different goals. We will describe them in this section.

With the function "**motorPower**" (Fig. 10), we set the fan motor power obtained from the PID control part as input. Our input is a percentage and we want to scale it to a number between 0 and 255. Then, we will send the resulting number to the motor pin. In this way, we drive the fan motor.



```
void motorPower (int motorPin , double power) {
  //Function that writes the values of the desired
  //power on the pin of the motor.
  //The inputs are the pin to which the motor is
  //attached and the desired power (number 0-100).

  double powerOut = (int)((power / 100) * 255);
  //gets the fraction of power in input, number between 0 and 1

  analogWrite(motorPin, powerOut);
  //gives the voltage to the motor, an integer 0-255,
  //proportional to the fraction of the power in input,
  //20 mV each step
}
```

Figure 10 – A detail from code written in Arduino IDE; this particular code snippet shows the implementation of "motorPower" function

The function "**getDistance**" helps us to calculate the distance in centimetres measured by the ultrasonic sensors. These sensors just give us the time interval between sending the sound wave and receiving it, so an intermediate step calculating the distance is needed.

First, we set the sensor's trigger off. Then after 2 microseconds, turn the sensor's trigger on for 10 microseconds. After this, when the reflected pulse comes back, the receiver reads the Echo pin, and with this instruction

```
duration = pulseIn (echoPin, HIGH);
```

we get the sound travel time in microseconds. Now we are able to calculate the distance by multiplying the duration time with $0.03434\ \left[\frac{\mathrm{cm}}{\mu\mathrm{s}}\right]$ (speed of sound) and dividing it by 2 (because sound travels the same distance twice, in both directions).

It would be interesting to show some details related to system on an LCD. We have used the proper libraries for LCD: **<Wire.h>** and **"LiquidCrystal_I2C.h"**. The function "**printInfoLCD"** prints the whole LCD screen and is used for initialization, the latter, "**updateInfoLCD**", is used only for updating the screen with the current values. The information printed: reference (%), ball height (%) and the power delivered to motor (%).

4.3 MAIN PART

In this section, we will describe the main part of the code which is the loop part. Arduino immediately after turning on executes the setup; and then starts running the main part and after finishing, repeats it continuously.

We designed a four-state Moore machine (Fig. 11) to turn on/off the system using a button. In this case, if the button is pressed, HIGH signal appears at an input, otherwise LOW signal is registered.

Suppose starting from state 0 i.e. the state where every part and variable of the system is reset. When we press the button, the system goes to state 1 and there it remains while the button is pressed. At the moment we release the button, the system will go to the state 2. State 2 is the state in which our device begins to work. When we press the button again, it goes to state 3 and at the moment we release the button the system gets back to state 0.

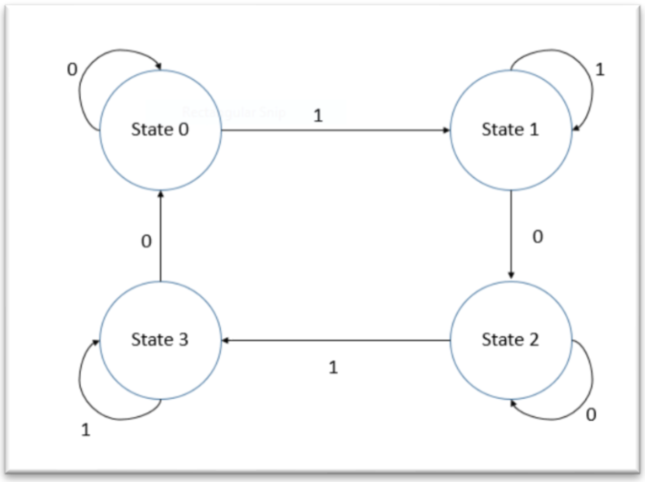

$$\begin{tikzcd}
\text{State 0} \arrow[loop left, "0"] \arrow[r, "1"] & \text{State 1} \arrow[loop right, "1"] \arrow[d, "0"] \\
\text{State 3} \arrow[loop left, "1"] \arrow[u, "0"] & \text{State 2} \arrow[loop right, "0"] \arrow[l, "1"]
\end{tikzcd}$$

Figure 11 – Moore machine with four states that enables turning on/off the system by pressing the button

While our system stays in state 2 (on state), the position of the ball and hand (reference's position) is calculated in each loop by using the "**getDistance**" function.

For the reference value, we get the hand distance from the sensor. Then we subtract the lower gap to scale the value to the reference system. After that, the value is stored in three-element array.

For scaling the ball position we use the equation below

```
ballPos = (columnL + upperGap)
    - (distanceB + ballDiam / 2);
```

where **"distanceB"** is obtained by the **"getDistance"** function and other operands are physical constants.

Since in the implemented PID control a constant sampling time is assumed, we have to assign a specific duration to each operation during one loop cycle. The ultrasound measurement time is intrinsically variable, so a delay is needed to be introduced according to the following expression

$$delay\,[\mathrm{ms}] = 30 - duration[\mu\mathrm{s}] \cdot 0.001;$$

30 millisecond is our choice for the duration of one part of the loop. This decision is based on the maximum sampling frequency of sensor ($40\ \mathrm{Hz}$) and the worst case measurement return value.

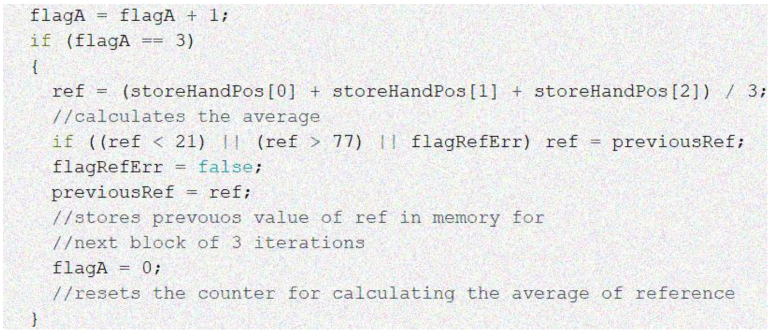
After that, we decided to scale the distances to percentages.

ballPos = ballPos * (100 / columnL);

This because numbers between 0 and 100 can be easily handled.

In section 2.2 we have commented sensors precision. To avoid measurement errors affecting our control on the position of the ball and the hand, some strategies have been adopted.

We expect the user hand not to move as fast as the ball inside the tube, therefore, an averaging over three consecutive samples can be performed in order to reduce random errors without any additional speed issue.



```
flagA = flagA + 1;
if (flagA == 3)
{
  ref = (storeHandPos[0] + storeHandPos[1] + storeHandPos[2]) / 3;
  //calculates the average
  if ((ref < 21) || (ref > 77) || flagRefErr) ref = previousRef;
  flagRefErr = false;
  previousRef = ref;
  //stores prevouos value of ref in memory for
  //next block of 3 iterations
  flagA = 0;
  //resets the counter for calculating the average of reference
}
```

Figure 12 – A code snippet showing the algorithm for reference averaging; it keeps system from too frequent reference changes and also uses some logic to discard false sensor measurements

On the ball position instead, it was not possible to make an average, since a control as fast as possible is desirable. A simple comparison is done at each obtained value: the ball cannot stay higher than 100% or lower than 0%, if this is detected the previous stored value is used in this new loop iteration.

The limitations in the condition are calculated by considering the physical terms of our device.

As we mentioned, the ultrasound sensor has a minimum range for detection ($2\ \mathrm{cm}$) so that data collected when the ball is stick to the sensor is not reliable. We noticed during testing that this situation might happen and, once it is reached, the sensor gives distance 0%: the fan increases the power and the ball does not move.

We solved this issue in a simple, but effective way: if the ball overcome an upper limit (85%) the system is reset; the fan turns off and the reference is set to 70%. The ball does not stick, but falls down to reach the indicated value.

4.4 Separating the tube in three parts for better control

As we mentioned in section 3, the PID control law is suitable for linear systems. Nevertheless, the air flow in the tube, blown up by the fan, is not linear; thus, the system itself is nonlinear.

A standard procedure (if not implementing a nonlinear control law based on the exact model of the system) in this case is taking advantage of linearization of the mathematical description of the system around one point; in our case – a certain position z_0 on the vertical axis. For small displacements the approximation works well, while the approximation error (defined as the difference between the actual behaviour and the linearized one) increases when we get further from the point z_0.

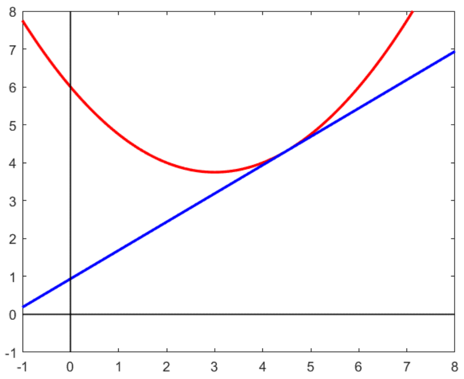


Figure 14 – Idea of linearization. The blue line represents a linear function that approximates the function defined by the red curve – the approximation is best in point x=4.5 in this example, and the approximation error increases as we go away from that point.

A solution that came to our mind is to separate the whole length into three regions (**A**, **B**, **C**), and to linearize separately these three sub-systems around three points z_{a0}, z_{b0}, z_{c0}. With this approach, the distance between the ball and the point z_{i0} is small enough to reduce the linearity (approximation) error.

The regions are divided as shown in the picture below.

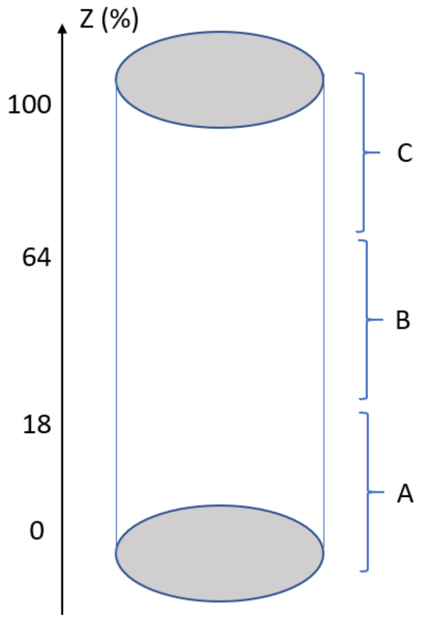


Figure 13 – Division of the whole tube length in 3 parts

The three regions are different, so also the constant of the PID controller are different and have been calibrated separately. During calibration we have considered the physics of our system. As an example, in the lower region (**A**) we need stronger proportional part since the air flow is more turbulent and the force impinging on the ball is not as efficient as in the upper part, where the air flow is laminar and the movement of the ball is more sensitive to changes in the fan power.

Simple if conditions allow us to select the region in which we are: according to the z coordinate of the ball, the proper PID constants are selected. Later the proper voltage is applied to the fan motor.

Six possible situations may occur (at least in theory). Indeed, we have distinguished the case when the ball is in a certain region after making a transition upward or downward (e.g. from **C** to **B** or from **A** to **B**). This has been done to avoid sudden changes in the PID constant, thus making the transition between two regions smooth.

As an example, for a transition from region **B** to **C**:

```
prevControll = control - Kpc * (ref - ballPos)
-Kic * (ref - ballPos) * samplingTime -
-Kdc * (-ballPos + prevBallPos) / samplingTime;
```

As last, it is worth to mention that the intensity of the LED inside the tube is proportional to the position of the ping pong ball using a PWM port and the function "**ledPower**".

5 RESULTS AND CONCLUSIONS

The system has been calibrated to optimize speed in reaching the reference value (indicted by the user hand) and to reduce oscillations. During the testing operation we have collected the position as a function of the time, and it is possible to see the plot in the Figure 15. Here the ball, starting from rest position reaches a reference of 70%.

As it is possible to notice, the rise time (time required to come from 10% of reference to 90%) is approximately 1 s, the overshot is not significant, but oscillations due to the difference in PID parameters in different regions appear. The oscillations have a maximum amplitude of 10% (of the tube dynamics length) and a period of ≈ 1.8 s. The equilibrium is reached after 5.5 seconds and the ball is "stable", i.e. oscillations are decreasing in amplitude and vanishing. Approximately 10 s after the step excitation, we can say that the system is in a steady state, noticing that the fluctuations around the desired reference position are within $\pm 5\%$ of reference 70%.

Quite a satisfactory result considering the extremely low-cost hardware used. To have an effective control on the ball position the user hand should not be suddenly moved with large step-like changes, but instead smoothly lifted or lowered especially in downward displacements.

An interesting fact is that, as we expected, the part of the tube where the control is more effective is between 50% and 80%. This is because the air flow became more linear after few tens of centimetres from the fan i.e. it is more predictable.

Therefore, an improvement to this project might be using a longer tube in order to have a larger dynamics since the linear region would be larger. Also, better sensors (in terms of reliability and speed) are highly recommended. From the physical point of view, having the tube only 2 mm wider in diameter than the ball (instead of current 4 mm difference) would make the control easier, since the air flow would be less turbulent and more predictable, because the ball will have less freedom to move in undesirable directions.

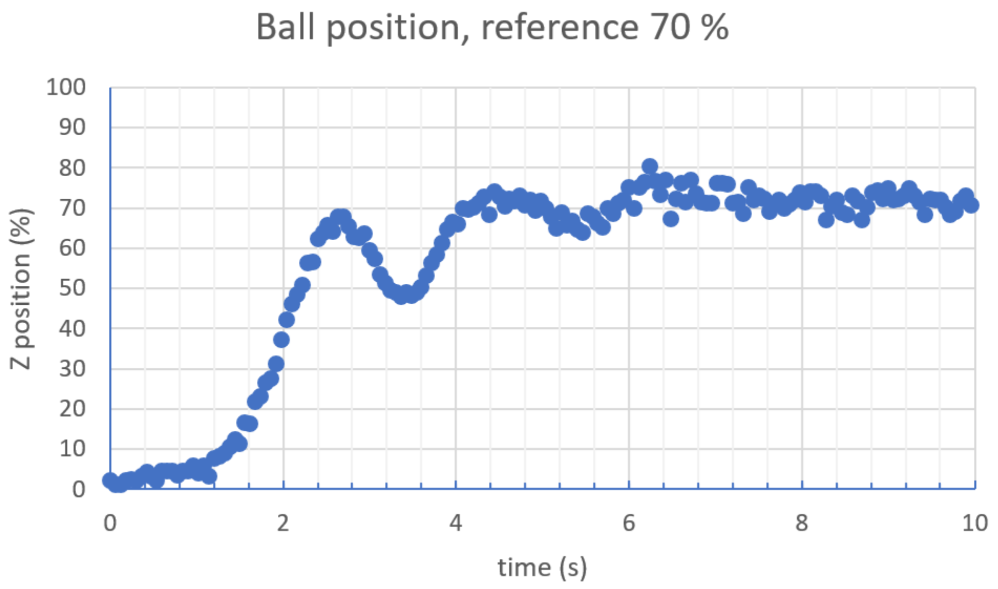


Figure 15 – Step response from 0 to 70%