ME2400 Project (Jan-May 2022)

Self Balancing Robot

C117

Authors:
ME20B202 VENNU MITESH REDDY (Batch C)
ME20B142 RAGAVENDIRAN N (Batch C)
ME20B140 PUTUMBAKA KARTHIKEYA (Batch C)
ME20B143 RAGHAV JANGID (Batch C)

Aim & Objective

The aim is to build a “mini-Segway” that can house the components, provided in the project kit, on a chassis that can balance itself (any part of the chassis should not touch the ground), without any external power supply, and without toppling when subjected to small imposed external disturbances on (only) two axially aligned, independently actuated motorized wheels. It will be able to balance things put on the top of it having reasonable weight without external help.The spacing between the wheels, W, along its aligned axis cannot exceed 6½ inches – the width of the entire structure should be within this two-wheel distance (i.e. no “spilling over” beyond the wheel span allowed in the ‘W’ direction). In the other two directions H and L) the chassis structure can be as tall and wide as deemed necessary.

The objective is to build a self balancing bot under the given constraints and using PID control algorithm on a digital controller to achieve the aim stated above.

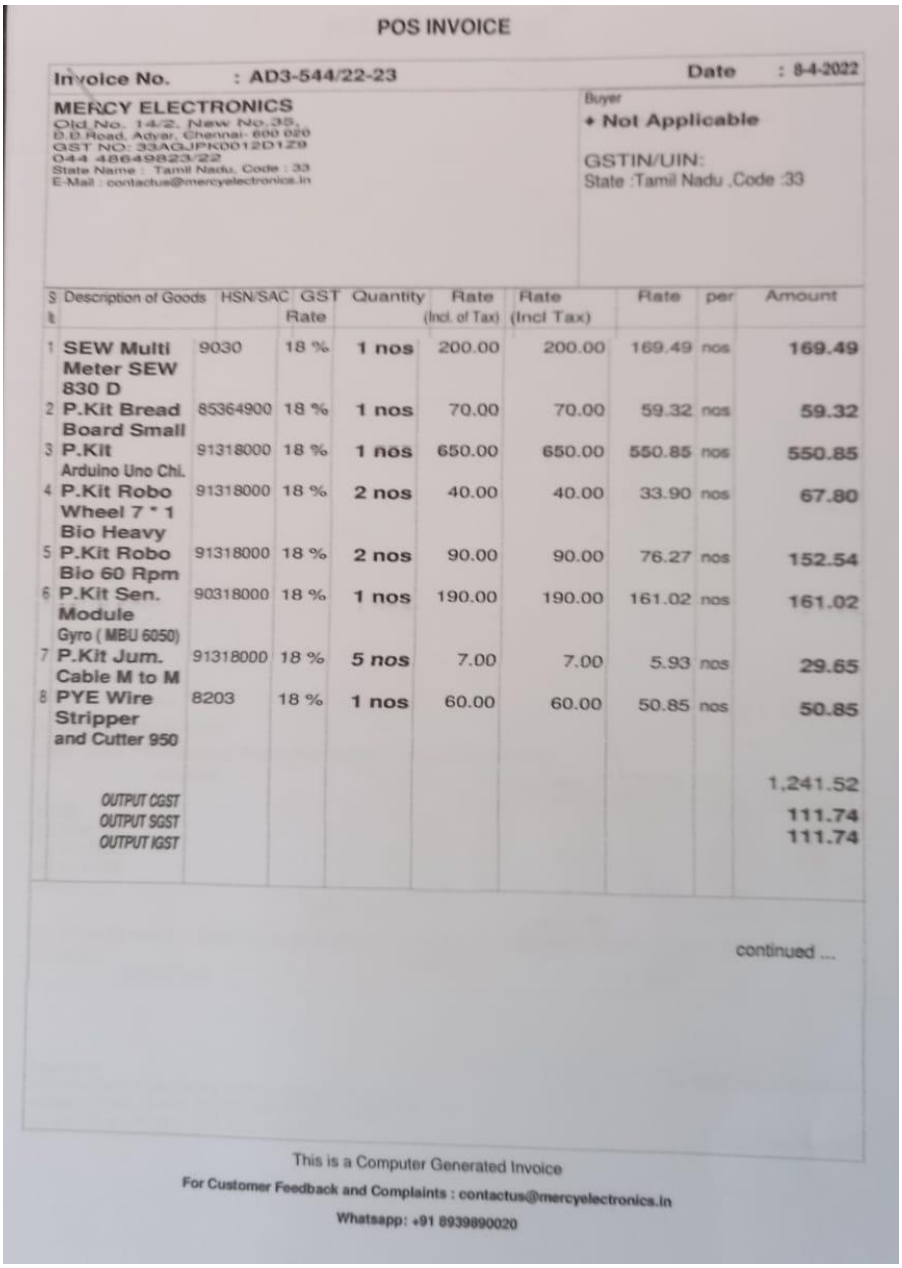
Component List & Sizing Justifications

1. PVC Sheet piece - 11x8 cm^2
2. 2- BO Geared Motors
3. Arduino Uno
4. 2- Wheels

5. 6- AA Batteries
6. MPU 6050
7. Motor driver - L293D
8. Breadboard
9. Jumper wires
10. Single Strand wires
11. Double sided tape

The chassis is made using the PVC sheet for it to be sturdy and not to wiggle when the bot is trying to balance. The components were placed in the chassis in such a way as to keep the center of gravity right above the wheel base as close as possible with symmetry in motor placement (the heaviest component) to have the center of gravity along the mid point and also placing layers of components to keep the c.g as close to the ground as possible.

Bill of Materials (BOM)



POS INVOICE

Invoice No. : AD3-544/22-23 Date : 8-4-2022

MERCY ELECTRONICS
Old No. 14/2, New No.35,
E.D.Road, Adyar, Chennai- 600 020
GST NO: 33AGJPK0012D1Z9
044 48649823/22
State Name : Tamil Nadu, Code : 33
E-Mail : contactus@mercyelectronics.in

Buyer
+ Not Applicable

GSTIN/UIN:
State :Tamil Nadu ,Code :33

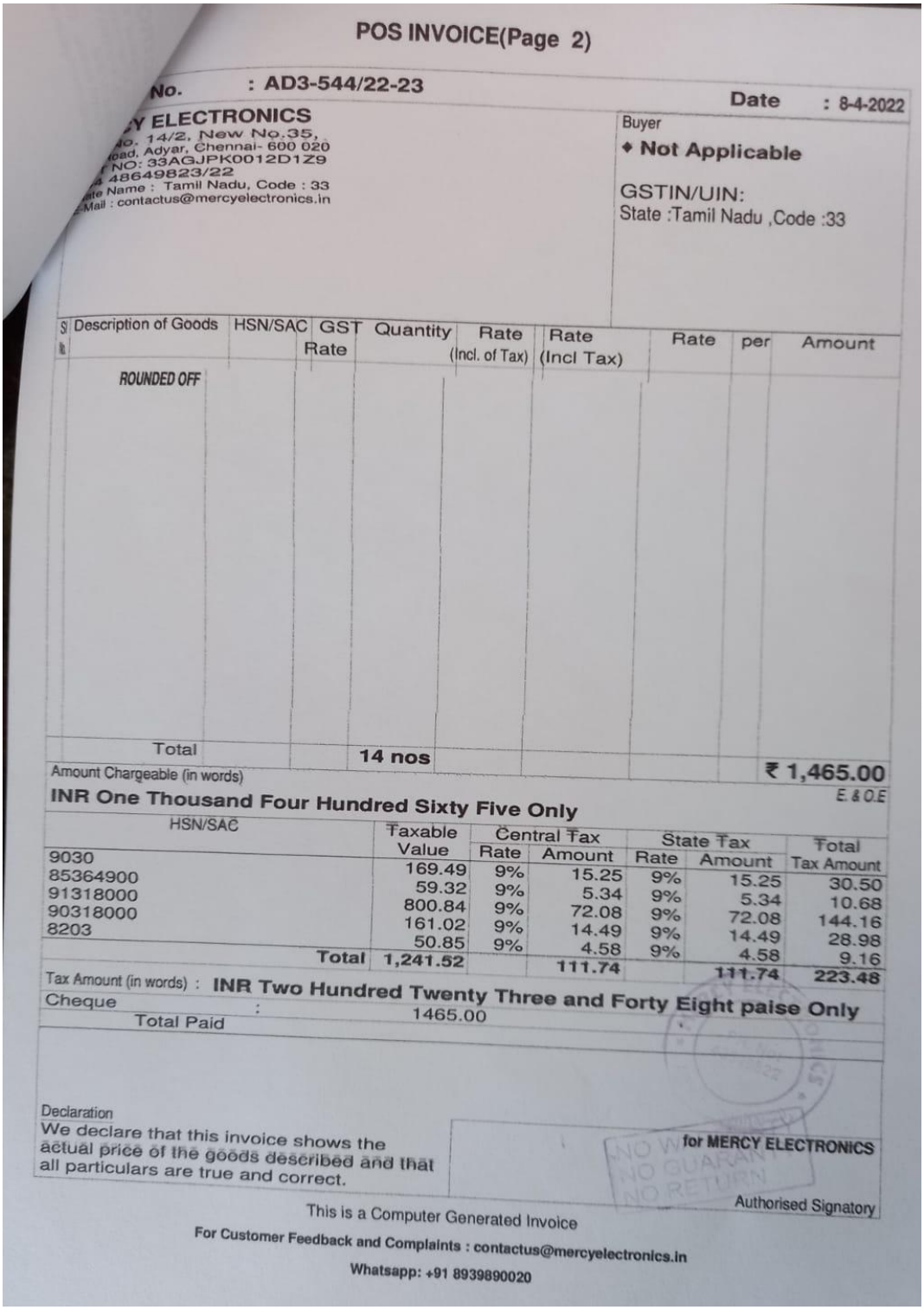
Sl	Description of Goods	HSN/SAC	GST Rate	Quantity	Rate (Incl. of Tax)	Rate (Incl Tax)	Rate	per	Amount
1	SEW Multi Meter SEW 830 D	9030	18 %	1 nos	200.00	200.00	169.49	nos	169.49
2	P.Kit Bread Board Small	85364900	18 %	1 nos	70.00	70.00	59.32	nos	59.32
3	P.Kit Arduino Uno Chi.	91318000	18 %	1 nos	650.00	650.00	550.85	nos	550.85
4	P.Kit Robo Wheel 7 * 1 Bio Heavy	91318000	18 %	2 nos	40.00	40.00	33.90	nos	67.80
5	P.Kit Robo Bio 60 Rpm	91318000	18 %	2 nos	90.00	90.00	76.27	nos	152.54
6	P.Kit Sen. Module Gyro (MBU 6050)	90318000	18 %	1 nos	190.00	190.00	161.02	nos	161.02
7	P.Kit Jum. Cable M to M	91318000	18 %	5 nos	7.00	7.00	5.93	nos	29.65
8	PYE Wire Stripper and Cutter 950	8203	18 %	1 nos	60.00	60.00	50.85	nos	50.85
									1,241.52
	OUTPUT CGST								111.74
	OUTPUT SGST								111.74
	OUTPUT IGST								

continued ...

This is a Computer Generated Invoice

For Customer Feedback and Complaints : contactus@mercyelectronics.in

Whatsapp: +91 8939890020



POS INVOICE(Page 2)

No. : AD3-544/22-23 Date : 8-4-2022

Y ELECTRONICS
o. 14/2, New No.35,
oad, Adyar, Chennai- 600 020
NO: 33AGJPK0012D1Z9
48649823/22
ate Name : Tamil Nadu, Code : 33
Mail : contactus@mercyelectronics.in

Buyer
♦ **Not Applicable**
GSTIN/UIN:
State :Tamil Nadu ,Code :33

Sl No.	Description of Goods	HSN/SAC	GST Rate	Quantity	Rate (Incl. of Tax)	Rate (Incl Tax)	Rate	per	Amount
	ROUNDED OFF								
	Total			14 nos					₹ 1,465.00

Amount Chargeable (in words) E. & O.E
INR One Thousand Four Hundred Sixty Five Only

HSN/SAC	Taxable Value	Central Tax		State Tax		Total Tax Amount
		Rate	Amount	Rate	Amount	
9030	169.49	9%	15.25	9%	15.25	30.50
85364900	59.32	9%	5.34	9%	5.34	10.68
91318000	800.84	9%	72.08	9%	72.08	144.16
90318000	161.02	9%	14.49	9%	14.49	28.98
8203	50.85	9%	4.58	9%	4.58	9.16
Total	**1,241.52**		**111.74**		**111.74**	**223.48**

Tax Amount (in words) : **INR Two Hundred Twenty Three and Forty Eight paise Only**

Cheque : 1465.00
Total Paid

Declaration
We declare that this invoice shows the actual price of the goods described and that all particulars are true and correct.

for MERCY ELECTRONICS
Authorised Signatory

This is a Computer Generated Invoice

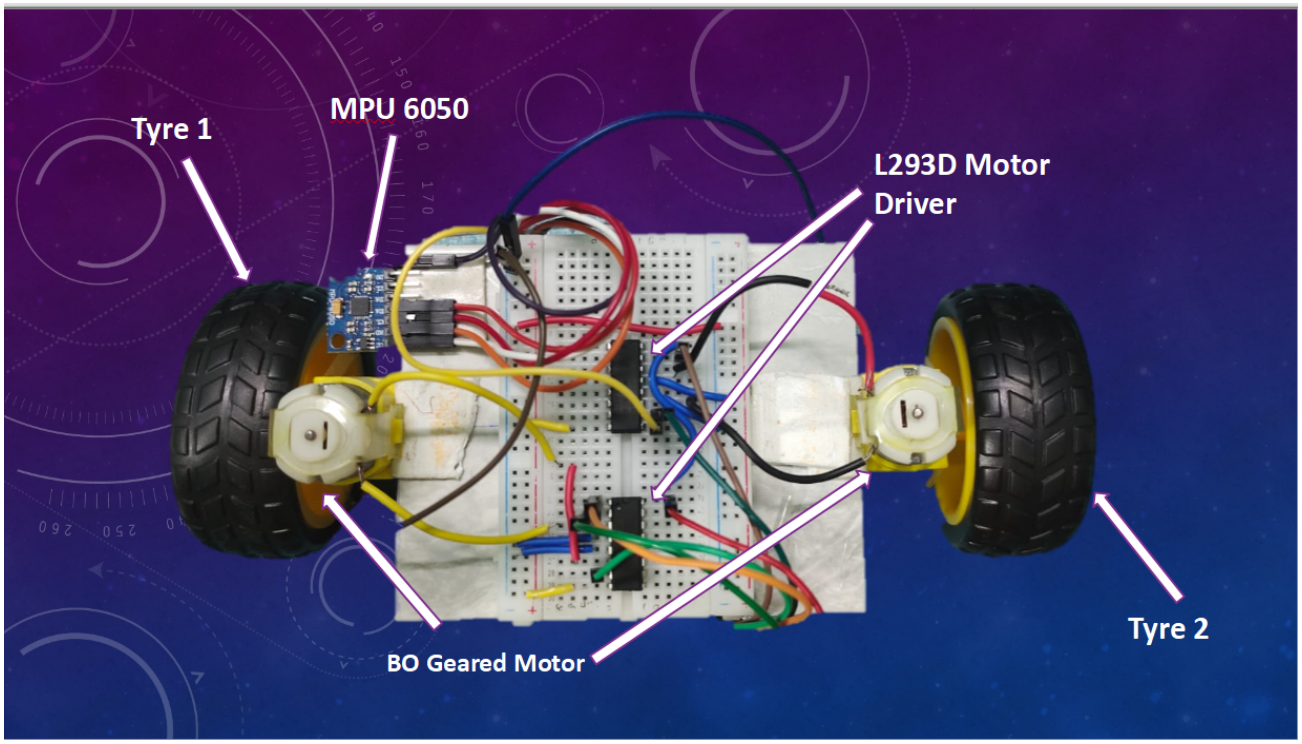
For Customer Feedback and Complaints : contactus@mercyelectronics.in

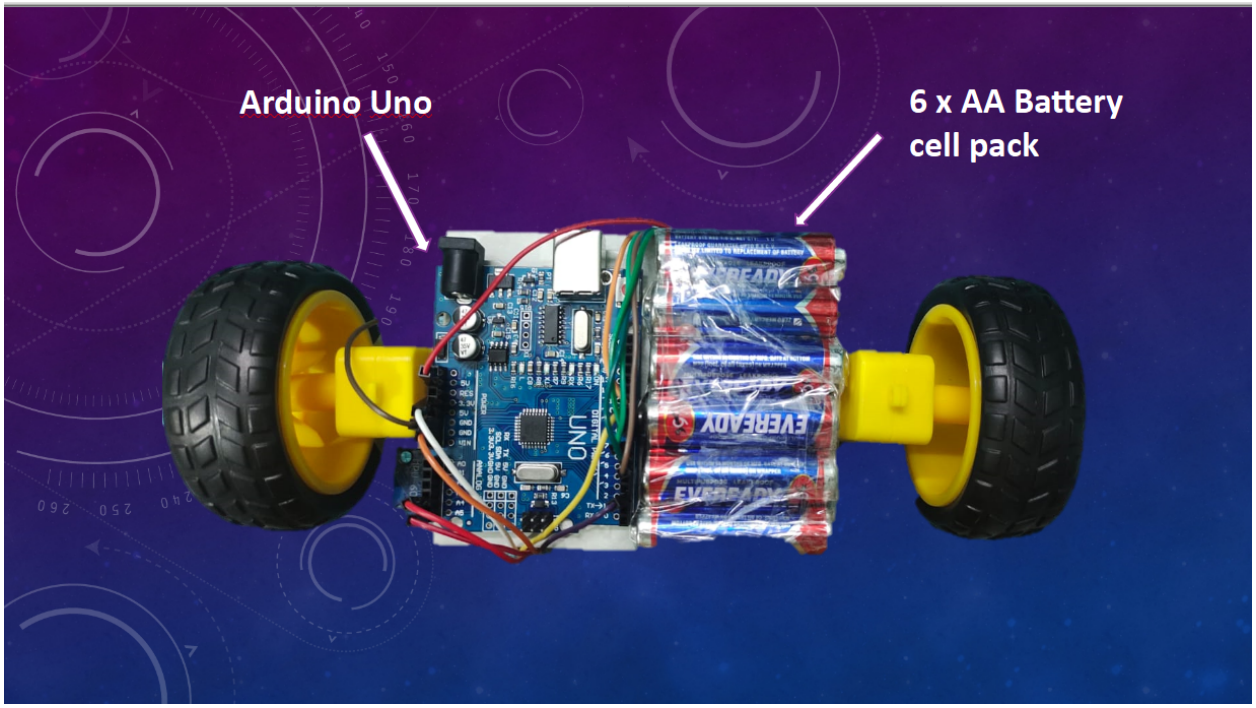
Whatsapp: +91 8939890020

Total Cost : 1577

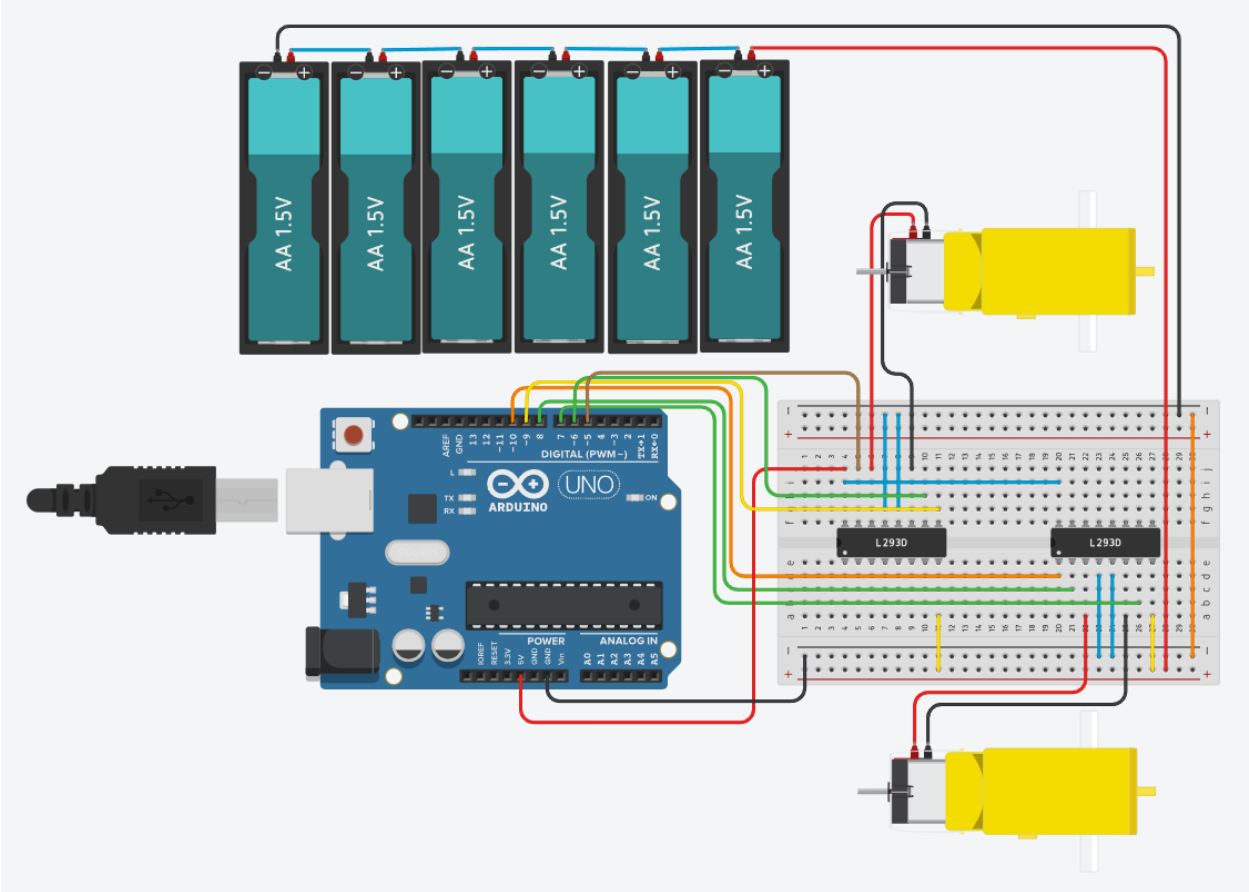
Mechanical design of the system

Mechanical design of this system is very much similar to the design of an inverted pendulum which is kept stable using a control system. An inverted pendulum is a pendulum that has its center of mass above its pivot point. It is unstable and falls over without any external help.It can be suspended stably in this inverted position using a control system which monitors angle of pole and actuates the necessary rotation to keep it stable.




Arduino Uno
6 x AA Battery cell pack
UNO
EVEREADY

Electronic circuit design



[Circuit Diagram in Tinkercad]

MPU 6050 Pin Connections:

VCC - 5V

GND - GND

SCL - A5

SDA - A4

INT - 2

Controller Design & Tuning Method

PID controller(Proportional Integral Derivative controller) is used in the design to balance the robot. PID circuit has 3 constants which determine the correction output of the controller. 3 constants are - Kp(Proportionality constant), Ki(Integral constant), Kd(Derivative constant).

With the proportional mode, the size of the controller output is proportional to the size of the error.

With the derivative mode, the controller output is proportional to the rate of change of error with respect to time.

With the integral mode, the rate of change of controller output is proportional to the error signal.

PID Control Algorithm

Combining all three modes of control (proportional, integral and derivative) gives a controller known as a three-mode controller or PID controller. The equation describing its action can be written as

$$\text{controller output} = K_{\mathrm{P}}e + K_{\mathrm{I}}\int e\,\mathrm{d}t + K_{\mathrm{D}}\frac{\mathrm{d}e}{\mathrm{d}t}$$

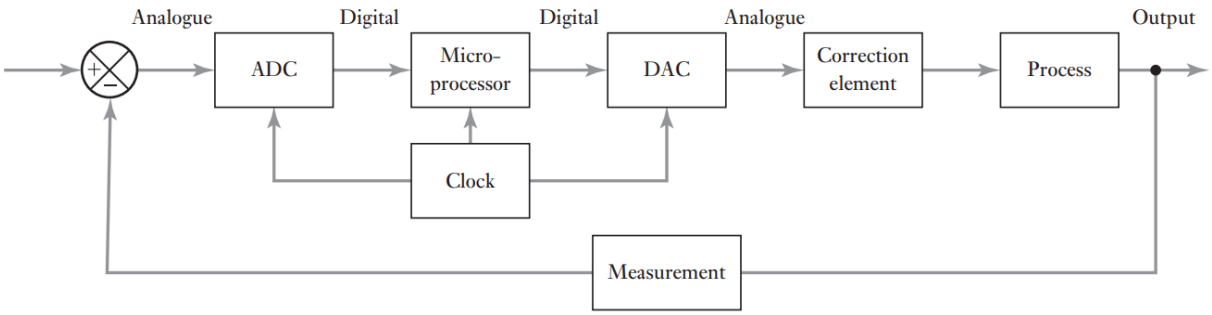
where KP is the proportionality constant, KI the integral constant and KD the derivative constant. Taking the Laplace transform gives

$$\text{controller output}\,(s) = K_{\mathrm{P}}E(s) + \frac{1}{s}K_{\mathrm{I}}E(s) + sK_{\mathrm{D}}(s)$$

And so

$$\text{transfer function} = K_{\mathrm{P}}e + \frac{1}{s}K_{\mathrm{I}} + sK_{\mathrm{D}} = K_{\mathrm{P}}\left(1 + \frac{1}{T_{\mathrm{I}}s} + T_{\mathrm{D}}s\right)$$

Digital Controllers



The above figure shows the basis of a direct digital control system that can be used with a continuous process; the term direct digital control is used when the digital controller, basically a microprocessor, is in control of the closedloop control system. The controller receives inputs from sensors, executes control programs and provides the output to the correction elements. Such controllers require inputs which are digital, process the information in digital form and give an output in digital form.

Since many control systems have analogue measurements an analogue-to-digital converter (ADC) is used for the inputs. A clock supplies a pulse at regular time intervals and dictates when samples of the controlled variable are taken by the ADC. These samples are then converted to digital signals which are compared by the microprocessor with the set point value to give the error signal. The microprocessor can then initiate a control mode to process the error signal and give a digital output.

The control mode used by the microprocessor is determined by the program of instructions used by the microprocessor for processing the digital signals, i.e. the software. The digital output, generally after processing by a digital to-analog converter (DAC) since correcting elements generally require analog signals, can be used to initiate the correcting action.

A digital controller basically operates the following cycle of events:

1. samples the measured value;
2. compares it with the set value and establishes the error;
3. carries out calculations based on the error value and stored values of previous inputs and outputs to obtain the output signal;
4. sends the output signal to the DAC;
5. waits until the next sample time before repeating the cycle.

Microprocessors as controllers have the advantage over analog controllers that the form of the controlling action, e.g. proportional or three-mode, can be altered by purely a change in the computer software. No change in hardware or electrical wiring is required. Indeed the control strategy can be altered by the computer program during the control action in response to the developing situation.

They also have other advantages. With analog control, separate controllers are required for each process being controlled. With a microprocessor many separate processes can be controlled by sampling processes with a multiplexer. Digital control gives better accuracy than analog control because the amplifiers and other components used with analog systems change their characteristics with time and temperature and so show drift, while digital control because it operates on signals in only the on/off mode, does not suffer from drift in the same way.

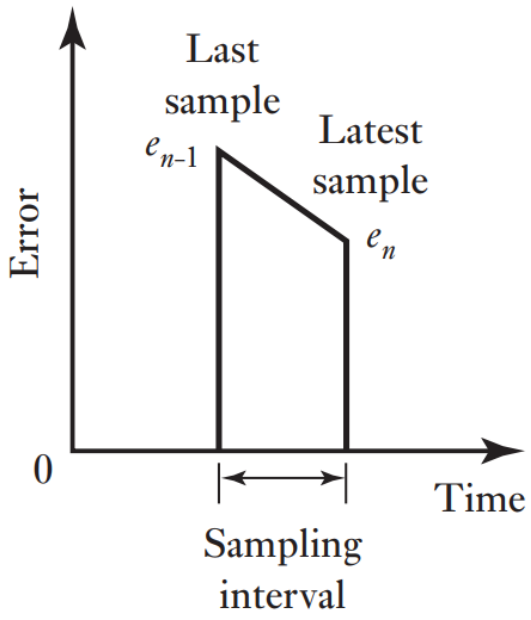
Implementing Control modes

In order to produce a digital controller which will give a particular mode of control it is necessary to produce a suitable program for the controller. The program has to indicate how the digital error signal at a particular instant is to be processed in order to arrive at the required output for the following correction element. The processing can involve the present input together with previous inputs and previous outputs. The program is thus asking the controller to carry out a difference equation.

The transfer function for a PID analog controller is

$$\text{transfer function} = K_{\mathrm{P}} + \frac{1}{s}K_{\mathrm{I}} + sK_{\mathrm{D}}$$

Multiplication by s is equivalent to differentiation. We can, however, consider the gradient of the time response for the error signal at the present instant of time as being (latest sample of the error e_n minus the last sample of the error e_{n-1})/(sampling interval Ts) .



Division by s is equivalent to integration. We can, however, consider the integral of the error at the end of a sampling period as being the area under the error–time graph during the last sampling interval plus the sum of the areas under the graph for all previous samples (Intprev). If the sampling period is short relative to the times involved then the area during the last sampling interval is approximately 1/ 2* (e_n + e_{n-1})/ Ts . Thus we can write for the controller output x_n at a particular instant the equivalent of the transfer function as

$$x_n = K_P e_n + K_I \left(\frac{(e_n + e_{n-1})T_s}{2} + \text{Int}_{\text{prev}} \right) + K_D \frac{e_n - e_{n-1}}{T_s}$$

We can rearrange this equation to give

$$x_n = Ae_n + Be_{n-1} + C(\text{Int}_{\text{prev}})$$

where $A = K_P + 0.5K_I T_s + K_D/T_s$, $B = 0.5K_I T_s - K_D/T_s$ and $C = K_I$.

The program for PID control thus becomes:

1. Set the values of K_P, K_I and K_D.
2. Set the initial values of e_{n-1}, Int_{prev} and the sample time Ts.
3. Reset the sample interval timer.

4. Input the error e_n.
5. Calculate yn using the above equation.
6. Update, ready for the next calculation, the value of the previous area to $Int_{prev} + 0.5(e_n + e_{n-1})Ts$.
7. Update, ready for the next calculation, the value of the error by setting e_{n-1} equal to e_n.
8. Wait for the sampling interval to elapse.
9. Go to step 3 and repeat the loop.

Controller Tuning

The PID parameters were fixed after an iterative trial and error process by testing iterations varying one each of the parameters, until the bot balanced within a short interval time (adapted quickly).

PID Constant values used:

These are the parameters we finalised on after thorough testing.

1. Kp=3500
2. Ki=50
3. Kd=600

Arduino Programming (with code)

We've chosen our digital microcontroller as the Arduino Uno and have coded the same. We've uploaded our code to our project Github repository, the same can be accessed with the link below:

Code

Results

We've built a self balancing robot with the given conditions for the physical aspect of the bot, along with the use of PID control algorithm for the software part.
The overview involves controlling the balance (" no acceleration ") for the bot, by using the reading from the MPU 6050 ("Accelerometer and Gyroscope Sensor") in a closed control loop, where the value is compared with the required value ("0 in our case"). The microcontroller ("Arduino Uno") uses the input reading from the sensor to control the actuator (" Motors") via the motor driver (" Motor driver - L293D "), which in turn changes the acceleration and gyroscope sensor values in the closed loop using PID control.

Conclusion

Thus, we've achieved our objective by building a self balancing robot, which balances when destabilised under a satisfactory time period by itself using PID control algorithms in a closed loop feedback system using a digital controller, sensors, analog actuators and motor drivers. This was a iterative learning process as well as a practical experience. Figuring out the solution for such a hands-on problem statement with a team was a great experience for all of us. We thank you for giving such an opportunity.

Team member contributions

ME20B202 VENNU MITESH REDDY (Batch C) : Coding, Testing, Procurement
ME20B142 RAGAVENDIRAN N (Batch C) : Coding, Testing, Procurement
ME20B140 PUTUMBAKA KARTHIKEYA (Batch C) : Chasis, Assembly, Circuits
ME20B143 RAGHAV JANGID (Batch C) : Chasis, Assembly, Circuits

Photos & Video

We've attached the photos and video of our self balancing robot in the link below:

Photos & Videos

—----------x x x—----------