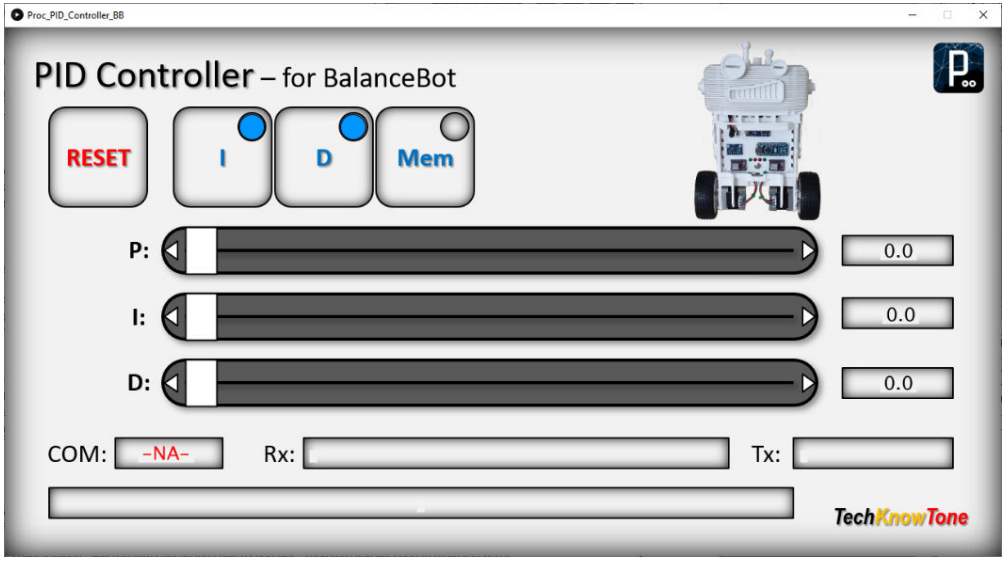
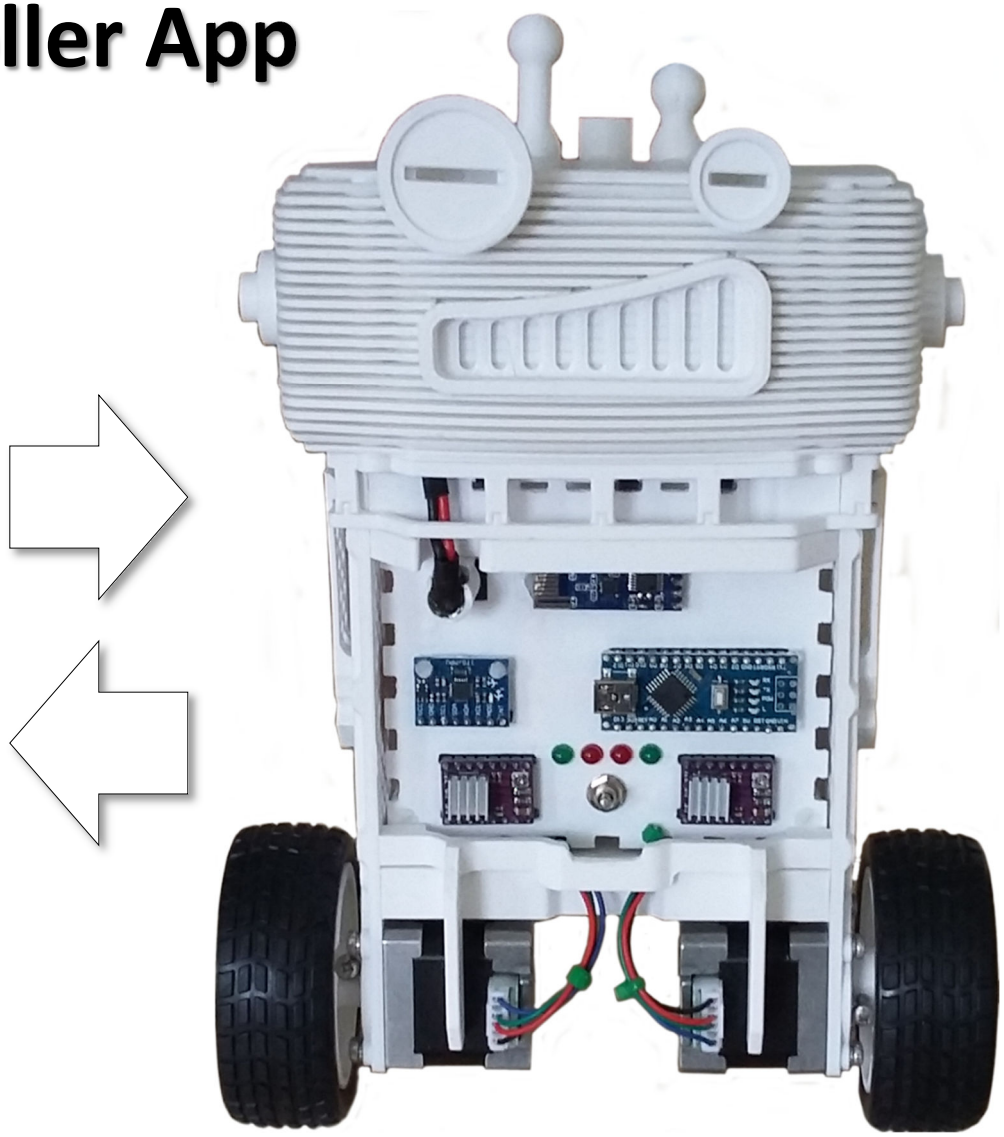
BalanceBot

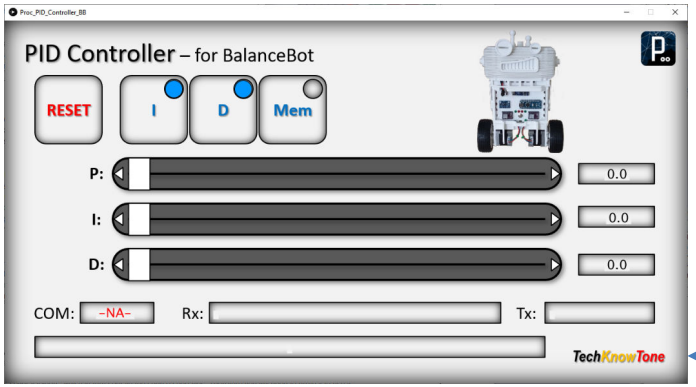
PID Controller App



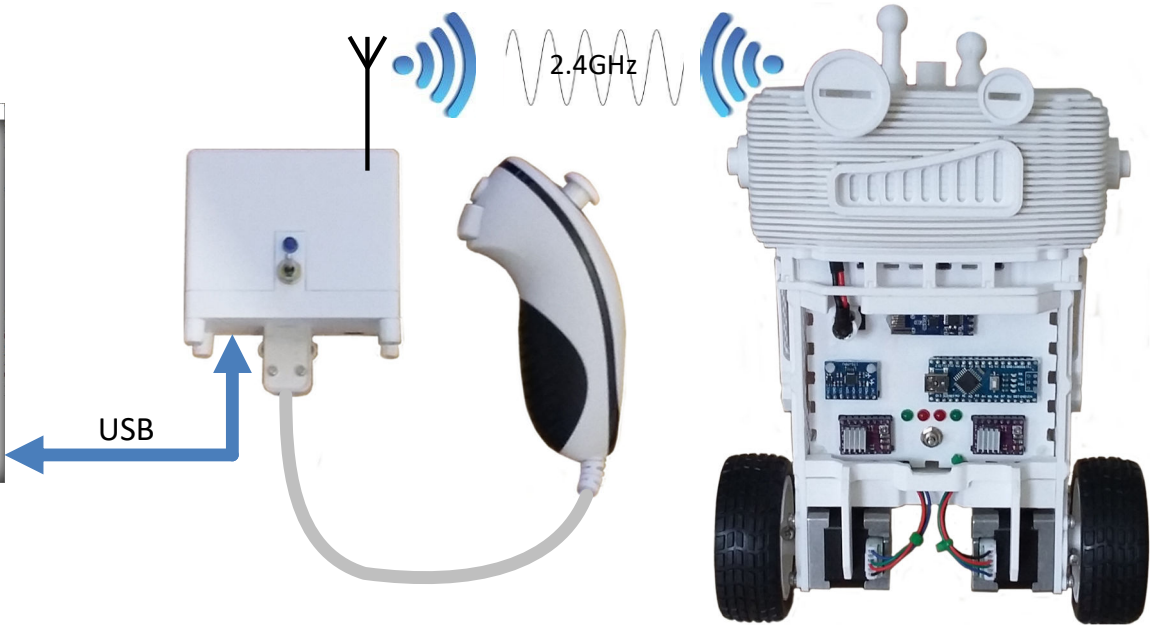


Issue: 1.0 Released: 01/09/2022 TechKnowTone

Overview



Windows App



The diagram above explains how the Windows based PID Controller app connects to the BalanceBot functionally, using a USB link between your PC and the NANO micro in the wireless transceiver, and then over the 2.4GHz link. Data flow is bi-directional.
The motor control system in the BalanceBot is known as a PID controller, which needs to be tuned to achieve self-balance with given components. Look up the term PID controller on the internet to learn more.

Essentially the P, I and D gain values need to be adjusted in order to achieve a responsive and stable behaviour from the controller.
The PID Controller app enables you to simply move sliders on the PC screen and adjust these variables in real time, whilst the robot is moving. Without the app, changing values in code and re-programming the NANO in the robot would be very tedious and prove to be extremely difficult to arrive at the correct values.
The app was written in C++ using 'Processing' which uses a coding environment very similar to Arduino, so you might like to experiment with that environment.

The app essentially sends very simple text commands to the code in the robot, which it then interprets as instructions. You will see these commands appear in the Tx field of the apps form, and the robots response to these commands appears in the Rx field.

The serial data sent over the link is conveyed at 9600 baud.

The following pages explains the key features of the PID Controller interface.

PID Controller

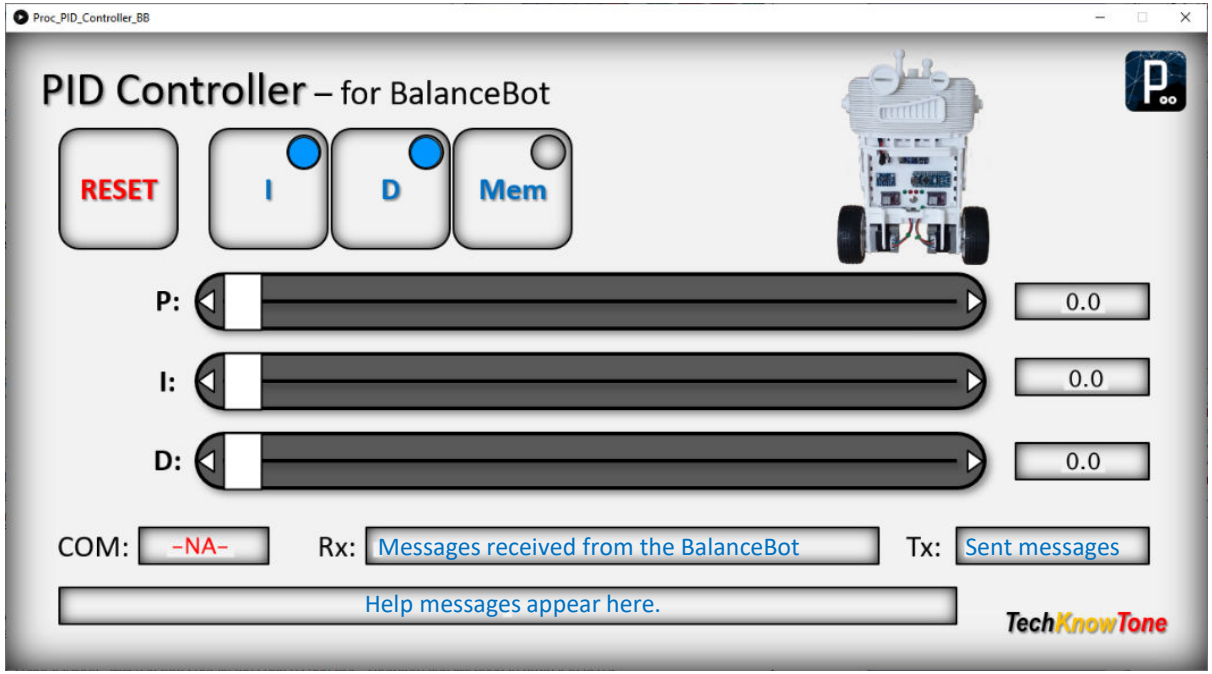
RESET restarts the app and sends a message to the robot to reset also.

I button toggles the PID I-gain ON/OFF.

D button toggles the PID I-gain ON/OFF.

MEM button recalls stored values if clicked briefly. A long press on this button will store values.

COM field indicates USB port status. **Red** is disconnected.



Slider values are displayed in the corresponding number fields at the edge of the form.

Rx field displays text messages received from the robot.

Tx field displays text messages sent as commands to the robot. Clicking on this field will cause the robot to send its PID values, and the controllers sliders will be updates accordingly.

When you launch the app, it will try to connect to the NANO in the wireless transceiver. If it is successful the COM port number should appear in the COM field in black text. If unsuccessful the COM field will display -NA- in red, and the app can't function.

A left mouse click on this field at any time will tell the app to re-connect to an available COM port. A right mouse click on the field will tell the app to disconnect; which is useful if you want to briefly connect to your robot using the Arduino IDE to upload code.

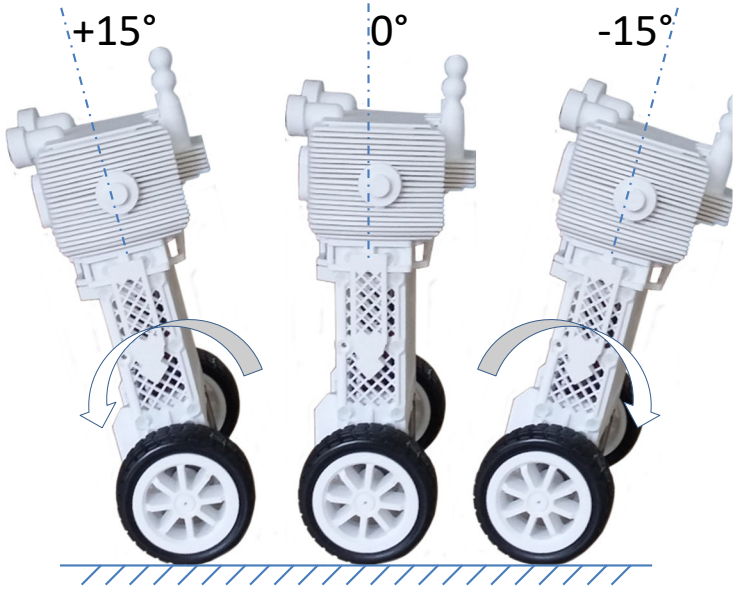
The bottom field on the form displays help information about any area of the form that is under the mouse pointer. So move the mouse around and see what it displays.

Note that if you move the sliders too quickly you can corrupt the data sent to the robot, causing strange effects, so move them slowly.

Click on the Tx field to request the robots actual gain values and update the apps sliders to match.

There are later versions of this PID controller on my web site, which support graphing functions. But as this early robot works at 9600 baud, that is not possible.

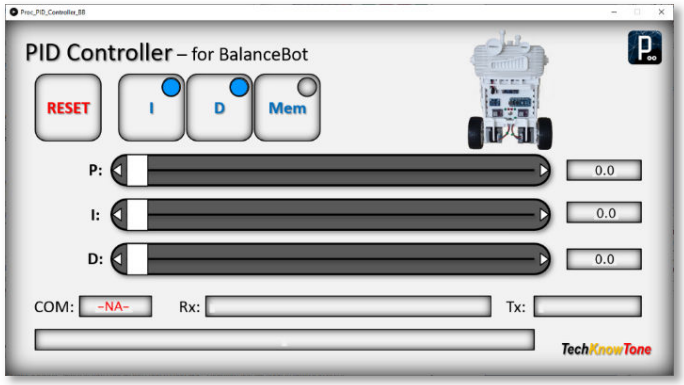
PID Controller Tuning



The MPU6050 motion sensor enables the BalanceBot to self-balance by providing data, at a high rate, which is converted into a vertical angle and compared with a setpoint variable. The resulting difference (error) is used to drive the wheels. When the robot is stationary, the setpoint is set to 0°. If the control system senses that the robot is falling forwards, it will drive forwards to minimise the error. Similarly it will drive backwards if it senses that it is falling backwards. The aim of the PID controller is to maintain equilibrium about the desired setpoint angle.

The PID controller applies three gain settings to the angular error signal. A **P**roportional gain, which simply multiplies the error signal by a gain factor. An **I**ntegral gain, which effectively accumulates small errors, such that if the error is small near the setpoint target, it will become larger over time. A **D**erivative gain, which in effect provides a braking function, to avoid overshoot and instability.

To tune the BalanceBot's PID controller, run the app with your PC connected to the Wii transceiver serial port. This enables the PID controller app to send control messages directly to your BalanceBot over Wi-Fi in a hands-off fashion. Start with all PID gain variables set to zero.



[1] Initiate the balance state by lying the robot on its back, after it has performed self calibration of the gyro; then take it to the vertical ACTIVE state. With all variables at zero the motors should not respond. [2] now slowly increase the P-gain value, whilst physically moving the robot forwards and backwards. You should start to feel drive from the motors assisting your movements. [3] increasing the P-gain will take you to a point where the robot almost self-balances, but more gain beyond that point makes the whole system go unstable. [4] repeat, and note the highest P-gain value possible, just before instability kicks in. Then back off the P-gain by about 33%. [5] now with P-gain set, slowly increase the I-gain value. You should find a point at which self-balance occurs; but again increasing I-gain further will cause instability. Note the best I-gain value for self balance.

[6] you can now bring the D-gain variable into play. Increasing D-gain should allow you to have higher values of I-gain and P-gain, before instability occurs. Higher values should mean that the robot effectively stiffens up about the setpoint angle, and doesn't wander or vibrate at that point.

This method is empirical, as there is no steadfast way of arriving at the PID gain values. Once you have determined the three gain numbers, they can then be entered into the robots code, in the table of definitions near the start.