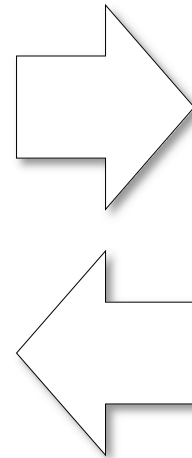
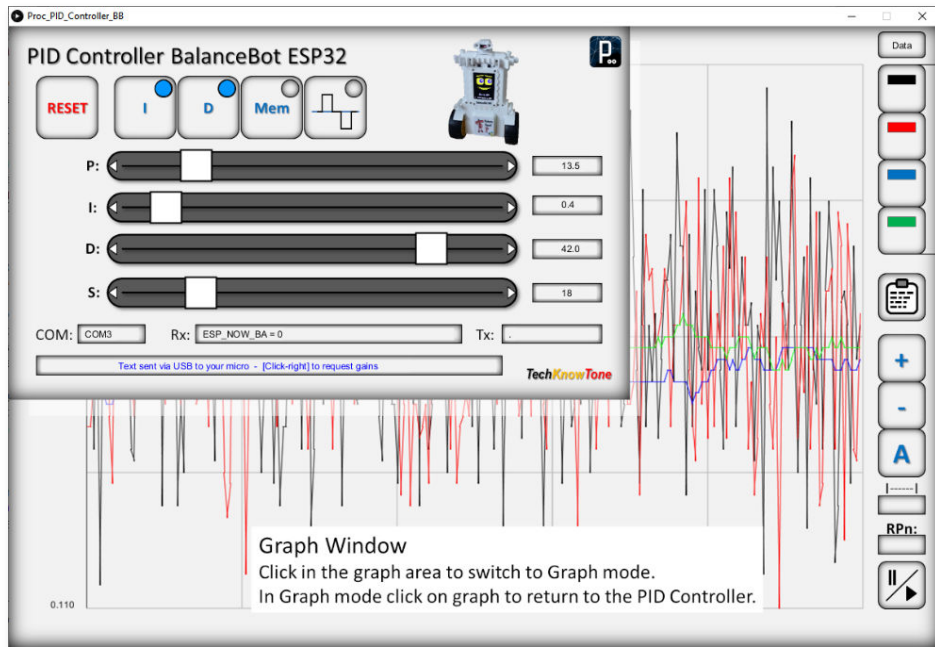
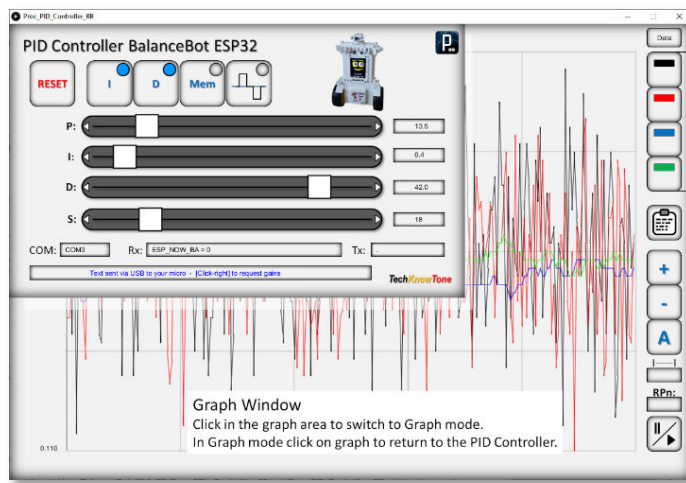


BalanceBot Mk1

PID Controller App



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 WEMOS micro in the wireless transceiver, and then over the 2.4GHz link. Data flow is bi-directional. The control system in the BalanceBot is known as a PID controller, and it needs to be tuned to achieve 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 ESP32 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 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 115200 baud, as a lot of data is sent from the robot in graphing mode. In this mode up to four different values can be broadcast, and which four it is, is determined by print statements in the robots code. The simple text commands allows you to tell the robot which set of four values you want to receive for plotting on the graph, and you can switch between them at will. The names of the four variables are also displayed above the graph area. For example you can look at the gyro sensors, and then switch to looking at the PWM values sent to the motors. The following pages explains the key features of the PID Controller and graphing interface.


PID Controller

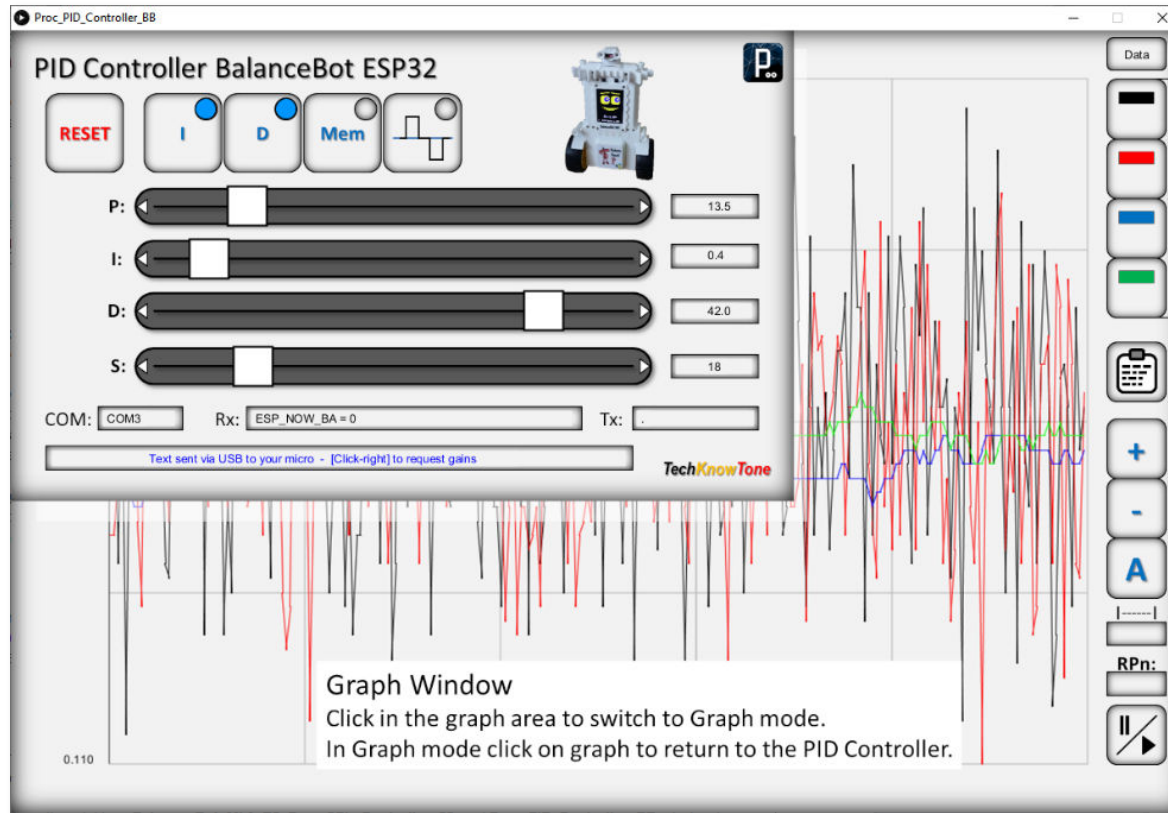
RESET restarts the app and sends a message to the robot to reset also.

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.

 this button tells the robot to apply cyclic step functions to the X and Y gyro axis's for test purposes.



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 controller form sliders will be updates accordingly.

When you launch the app it will try to connect to the WEMOS 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. Click on the Tx field to get the robots actual values and update the sliders to match.

The large graph area of this window, behind the PID Controller form is the graphing tool, and clicking anywhere in this region will switch the app to graphing mode.

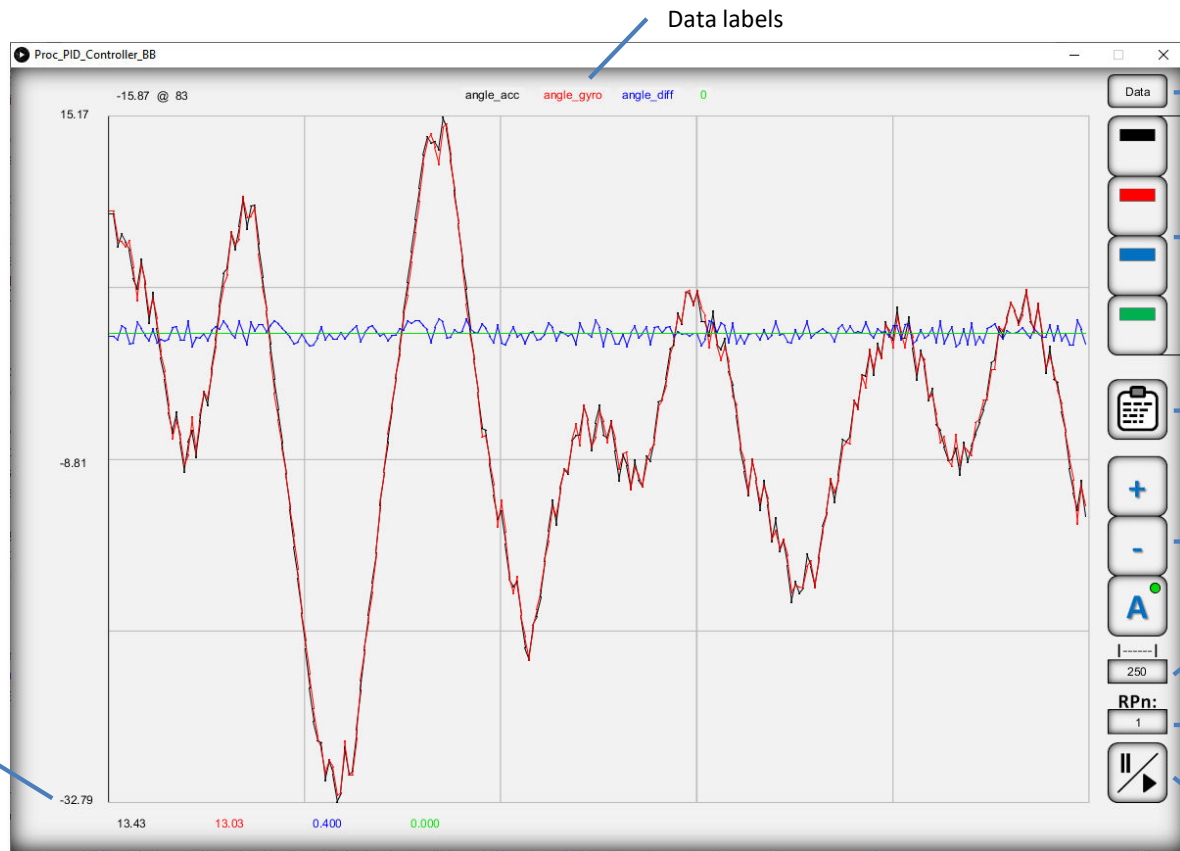
In graphing mode you can return to the PID Controller form at any time, simply by clicking in the centre graph region.

Graph Tool

In automatic mode the limits on the vertical axis scale are automatically set to match the maximum and minimum values displayed in the graph.

Note that the baud rate limit of the transceiver means that we can not see every event occurring within the robot. But it does give us a good idea as to what is going on.

You can click on the graph limits in manual mode to adjust their values and range.



Toggle between graphing and data listing modes.

These four buttons toggle ON/OFF the graph data, if data is being sent.

Copy data to Windows clipboard.

These three buttons increase, decrease or set the vertical scale to automatic.

This field shows/determines the number of points plotted on the X axis.

This RPN field shows/determines which data is being requested from the robot.

The pause/play button controls the scrolling of the graph.

This tool will plot up to four values received from the robot as a continuously scrolling display. The RPN value determines which data the robot will send, and clicking on that field will change it.

Data can be plotted in fixed scale or auto-scaling mode, both are useful depending on the type of data you are wanting to observe. Values are sent twice as a data integrity check.

If you pause the graph you can then move the mouse pointer over the area to observe the values stored in the underlying data. The four values at the bottom relate to the data at a given point in time, whereas the value at the top shows the mouse pointers position in terms of data value y stored value.

You can toggle the plots ON/OFF by clicking on the respective coloured button. If less than four values are being sent the corresponding buttons will be inactive.

By reducing the number of values plotted on the horizontal axis you can in effect speed up the scrolling effect.

To exit graphing mode and return to the PID Controller form, simply click anywhere in the graph area.

You can edit the code in the robot under the **X_Debug** tab to change or add print statements of your choice, taking care to follow the conventions I have used.

Note that if the transceiver power setting is too high data corruption can occur when the robot is too close to the Wii transceiver.

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 sense 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 controller is to maintain equilibrium about the 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 code in the table of definitions.

