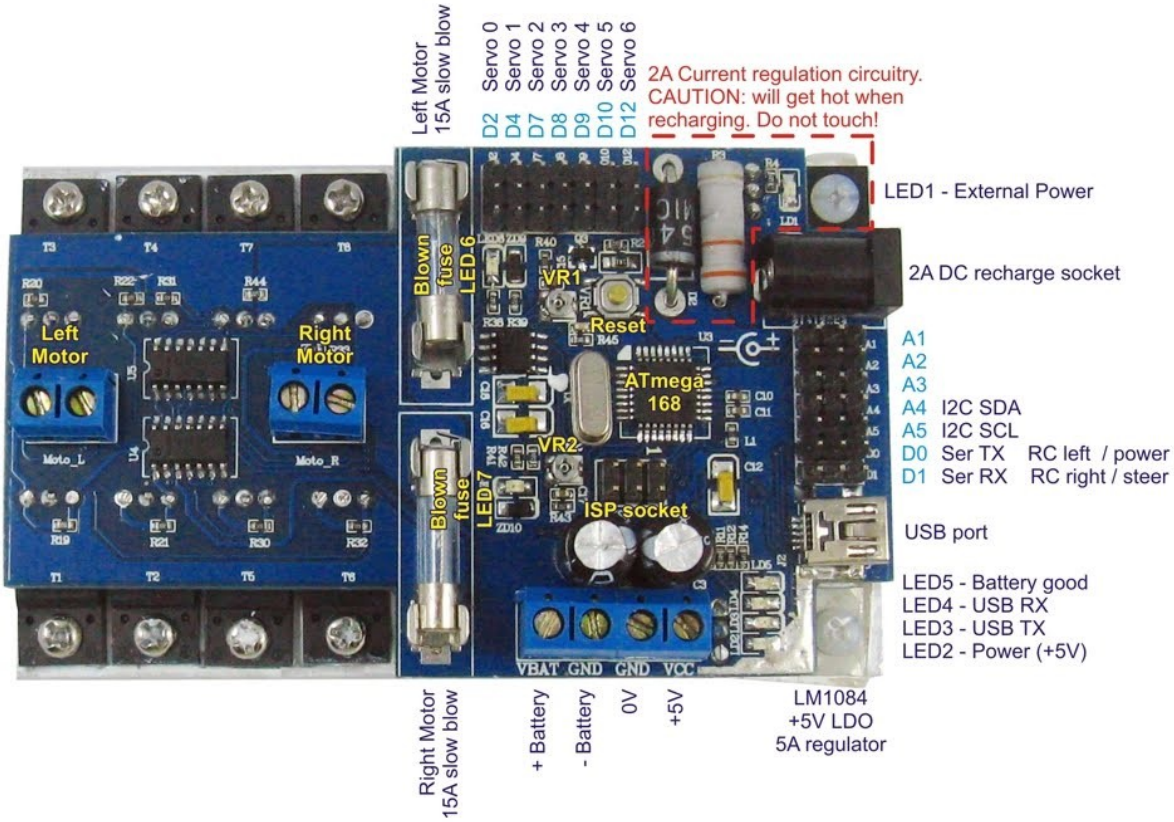
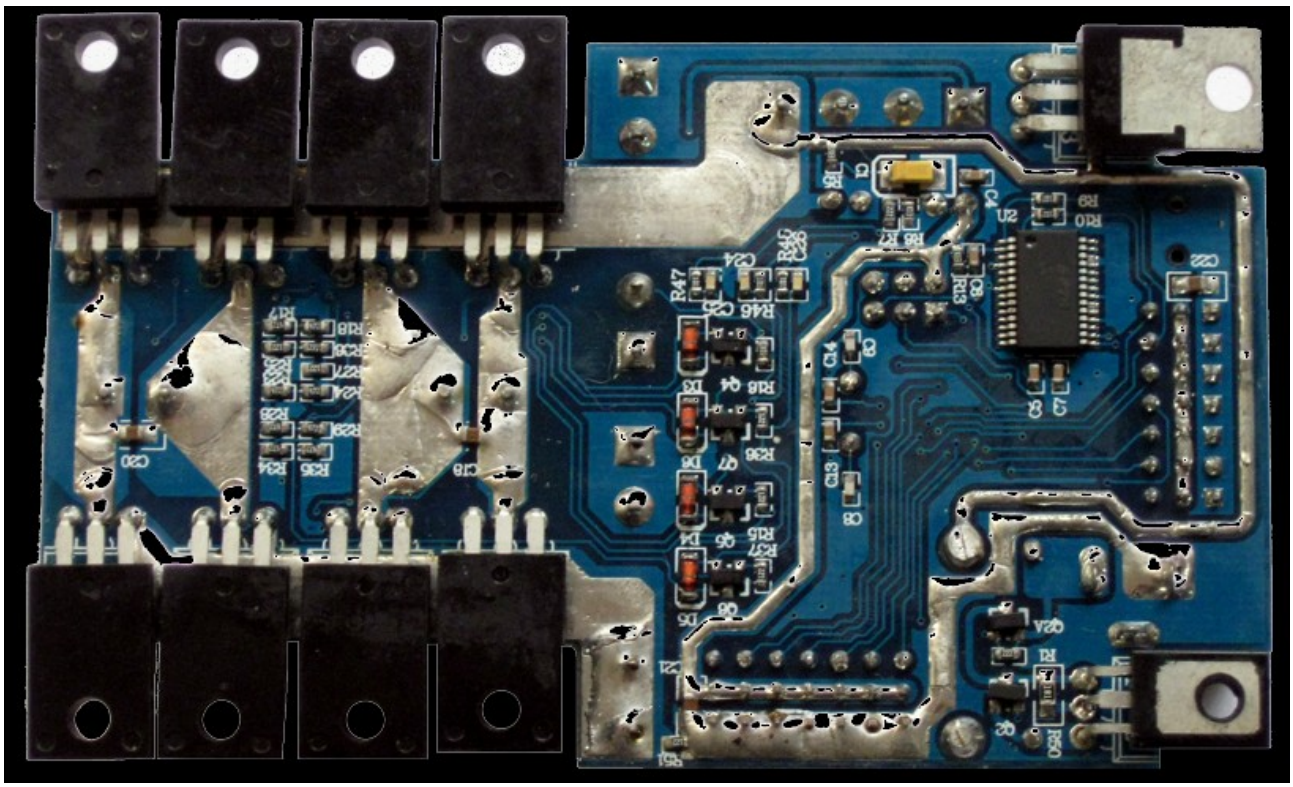
Understanding the Wild Thumper controller

The **"Wild Thumper"** robot controller uses the ATmega168 processor and is software compatible with the Arduino Nano 168. Unlike other Arduino compatible boards this controller includes features especially designed for larger / faster robots such as Sumo Bots.

- Dual FET "H" bridges. Each "H" bridge is rated for 15A continuous with current sensing and fuse protection. Maximum frequency 24KHz.
- Pulse width modulated electronic braking allowing fast and heavy robots to stop smoothly or quickly if required.
- Processor controlled 2A, current limited trickle charging circuit for use with SLA, NiMh and NiCd batteries.
- LDO, 5V, 5A regulator allowing several servos to be powered directly from the controller.
- Large heatsink which can be fitted with a 40mm CPU fan if required.
- USB interface and ISP socket.





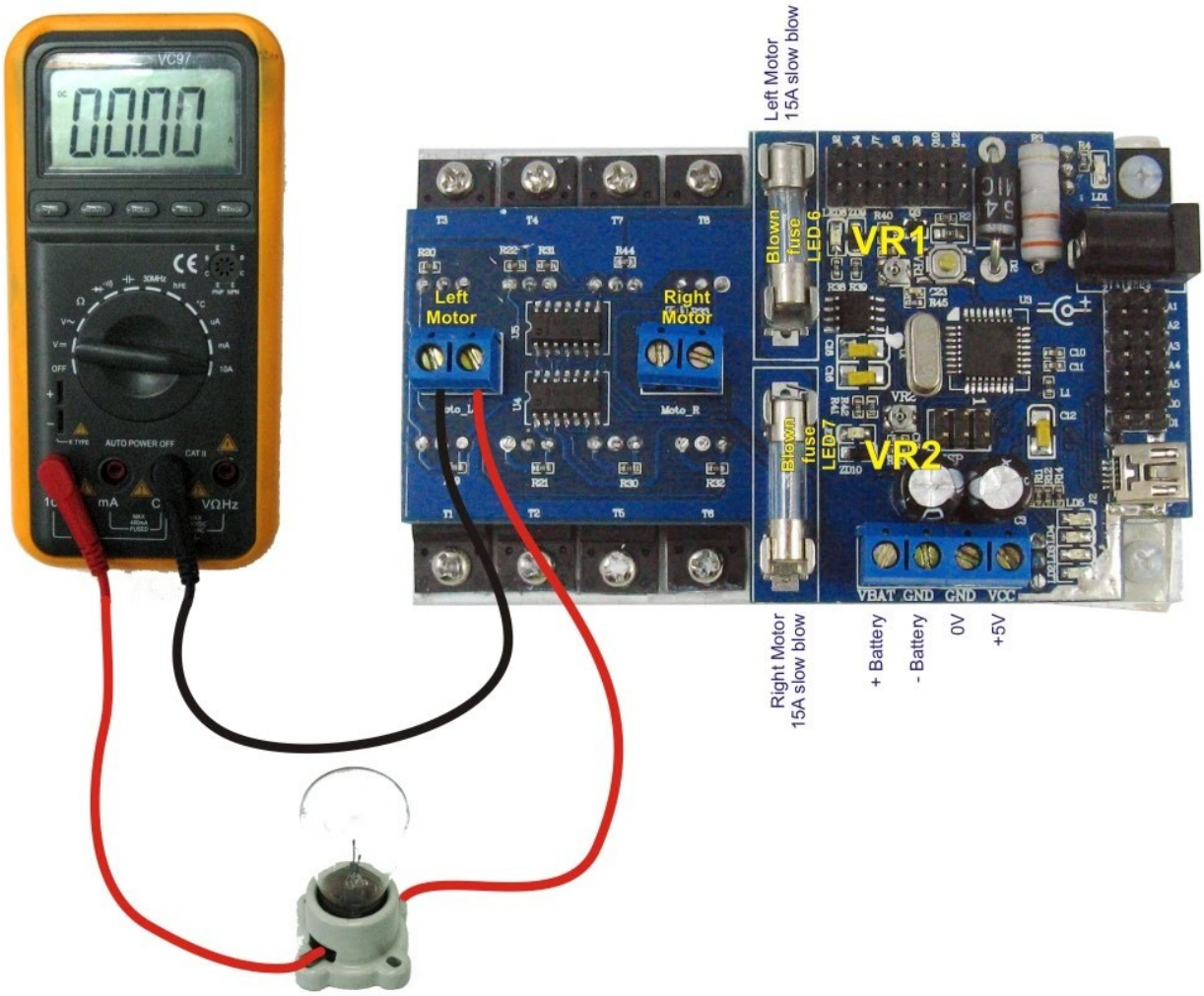
FET "H" bridges:
FET stands for **F**ield **E**ffect **T**ransistor. FET's are far more efficient than normal transistors and can have very low "on" resistances of just a few hundredth's of an Ohm. This allows them to handle high currents without getting too hot. The "H" bridge consist of 4x FETs that control power to the motor. The name "H" bridge comes about because in a schematic diagram the motor and transistors form a "H" pattern.

To control the speed of a motor, the "H" bridge pulses full power to the motor very quickly using PWM (**P**ulse **W**idth **M**odulation). The standard PWM frequencies used by the Arduino system are 488Hz and 976Hz (according to my digital oscilloscope) . Some customers may wish to change the frequencies to above 20KHz to reduce motor noise. It should be noted that higher frequencies increase the inductive reactance of the motor windings which limits the motors current draw. This results in a loss of torque. Changing the frequency is not recommended as a mistake in the settings can result in frequencies beyond the 24KHz limit which can damage the "H" bridge.

WARNING: Some commands and libraries such as the "tone()" command require use of the timers and can affect or disable the "H" bridges. For this reason it may be prefered to slave this controller to another processor. The servo function will **not** affect the "H" bridges unless you try to initialise more than 12 servos.

Current sensing:
Current sensing allows the processor to monitor how much current is flowing through the motors. This is very important for larger robots where the stall current of their motors can exceed the limits of the motor controller. The controller measures the voltage across the fuses. Because fuses are not all the same the current sensing circuit must be callibrated to suit the fuse. This works best if the fuse is soldered into the fuse holder. By using "slow blow" fuses and having the controller shut down the motor as soon as the current exceeds a safe limit. If a fuse does blow then a red LED near the fuse will light to show that it has blown.

Calibrating the current sensors:
Current sensor callibration is critical if your motors have stall currents greater than 15A. When used with a Wild Thumper robot chassis, the supplied slow blow fuses are ideal. If your stall currents can exceed 20A then you should use fast blow fuses. This provides a backup should your software fail to shut the motors down quick enough.



Calibrating the current sensors will require a Multimeter with a 10A range and a stable test load such as a light globe from a car or a high wattage resistor e.g. 10 ohm, 5W or higher.
Connect the controller to a battery or DC supply by the battery terminals. Connect to your computer with the USB cable. Load and run the “Wild_Thumper_Diagnostic.pde” program.
Click on the “Serial Monitor” icon and set the baud rate to 9600. You will see the load being measured by the processor.
If you remove the fuses then you will see the current readings jump above 700. This is the highest possible reading.
Replace the fuses and adjust VR1 or VR2 depending on which motor output you are calibrating.

Example:
Your multimeter is connected to the left motor and is reading 940mA.You adjust VR1 until the processor reads 47.
The left motor is now calibrated with a resolution of approximately 20mA (47x20mA=940mA).
Your motors have a stall current of 12Amps. To protect you motors you decide 11A is the safe limit.
Set “Leftmaxamps” in the “Constants.h” tab to 550 (11000mA / 20mA).
Set “overloadtime” to the number of mS required before the motor can restart after an overload occurs. The default is 100mS (1/10 of a second).

Now as soon as your motor draws more than 11A the controller will shut down the power for the overload time.
This protects your fuse and motor but still allows the robot to recover quickly.

Electronic braking:
One advantage of this controller over many others is that it has electronic braking (DC brushed motors only). For small light weight robots this feature is not normally required. For large/heavy robots or very fast robots, electronic braking is a must to avoid collision.

Under normal conditions when you turn off power to a motor it's rotor will continue to spin due to the momentum of the moving parts and the robot. At this point the motor behaves like a generator and produces voltage in the reverse direction to which the power was originally applied. When you short the terminals of the motor together a heavy current flows through the rotor generating a "back EMF" effectively trying to spin the rotor in the opposite direction.

The faster the rotor is spinning the greater the braking effect is. The controller uses PWM to control how hard the motor brakes. This is useful when you want your robot to stop smoothly rather than slamming to a halt and possibly falling or flipping over.

Processor controlled battery charger:
This feature should not be used with LiPo batteries as the batteries may catch fire or explode if incorrectly charged!!

The controller has the ability to measure the batteries voltage and switch between fast charge (2A maximum) and standby (aproximately 50mA). This allows the controller to charge Sealed Lead Acid (SLA) as well as Nickel Metal Hydride (NiMh) and the older Nickel Cadnium (NiCd) batteries. In the sample code, the processor shuts down the motors if the battery voltage gets too low and waits for the battery level to rise above the batteries nominal voltage.

For example if we use a 7.2V NiMh battery then once the voltage gets below 6V (the LDO regulator may not be reliable below 5.5V), the controller shuts down the motors. The software then enables the charging circuit and waits for the battery voltage to rise above 7.2V indicating that DC power has been applied to the charging socket. The sample code then monitors the change in the battery voltage. If the voltage peaks (does not increase after 15minutes) then the battery is assumed to be charged and the controller switches the charging circuit to standby.

This feature requires the DC input to be 3V - 6V higher than the voltage of the battery being charged. The DC supply must be rated for at least 2A continuous. If your charging voltage is more than 4V higher than the nominal battery voltage then a small CPU fan should be fitted to the heatsink.

NOTE: This controller was originally designed to work with the Wild Thumper robot chassis ans a 7.2V NiMh battery. While higher battery voltages can be used this will cause the 5V regulator to get very hot if heavily loaded. The battery charging regulator can also get very hot if the charging voltage is more than 3-4V higher than the batteries nominal voltage. A 40mm **CPU cooling fan** should be fitted to the center of the heatsink in these situations.

Recommended batteries and voltages are:

Battery Voltage	Battery Type	Charger Voltage no fan	Charger Voltage with fan	Comment
6V	SLA / NiMh	9V	12V	As 6V is close to the 5V regulators dropout voltage the 5V supply should not be heavily loaded for best reliability.
7.2V - 9.6V	NiMh		12V	These batteries can be charge at 12V without a fan but the heatsink can get quite hot. A fan is recommended.
12V	SLA / NiMh	15V	18V	
14.4V	NiMh	18V		Charging voltages higher than 18V are not recommended.

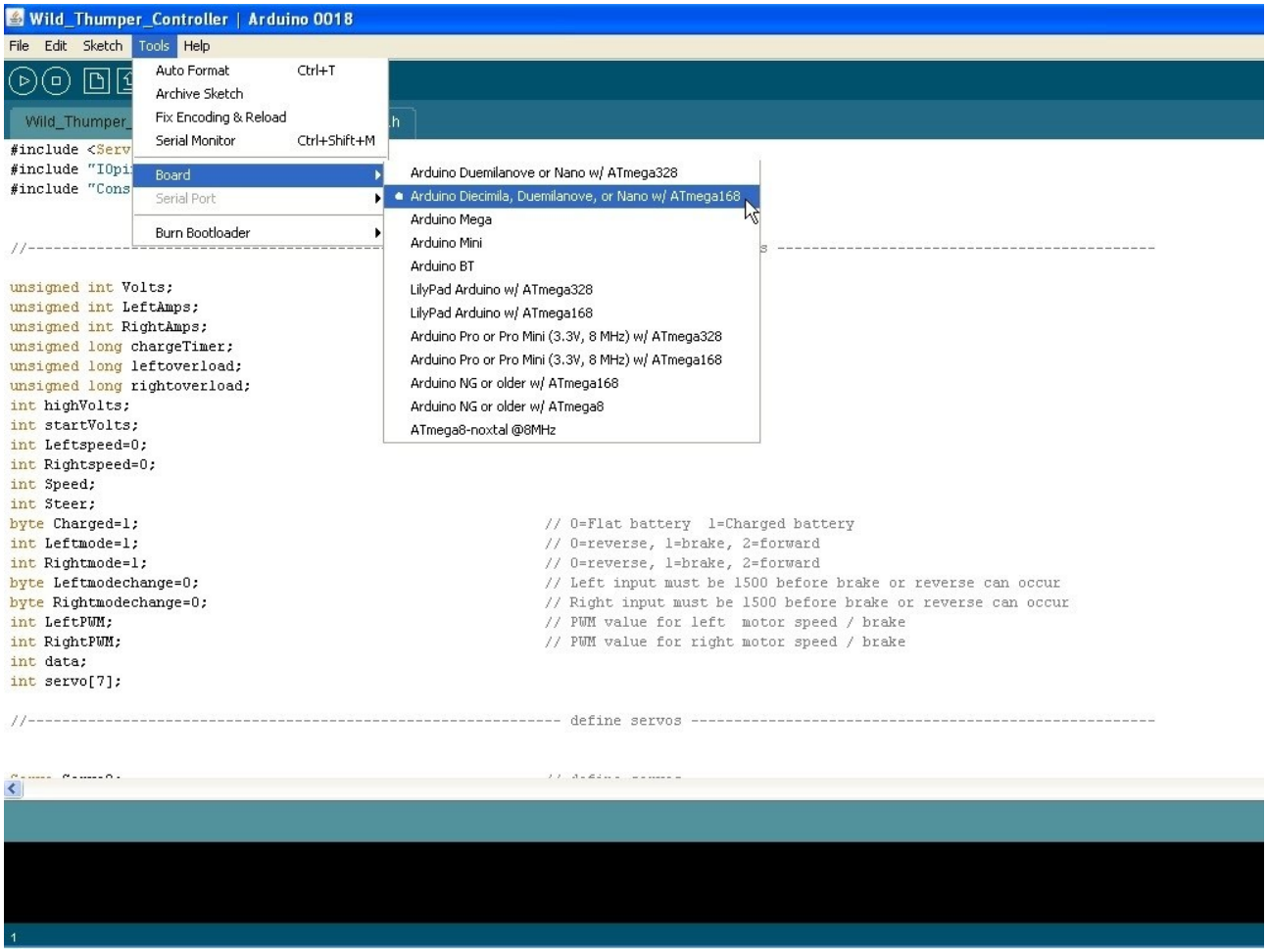
Spare IO pins, Servos and Sensors:
The controller has spare digital and analog pins available. These are terminated in 3pin male headers with Vcc (5V) in the center and ground (0V) as the pin closest to the edge of the board. The 5V regulator has a maximum rating of 5A which allows miniature and standard servos to be powerd directly from the controller. 5V sensors can also be powered directly from ther board.
NOTE: some sensors may have a 3 pin connector with the power pins in a different order. Check that Vcc is the center pin to prevent damage to your sensor.

It should be noted that all spare pins can be used as digital inputs or outputs. A1 - A5 are digital IO pins D15 to D19. If a pin is configured as an input (power up default setting) then writing a logical "1" to that pin will enable an internal 20K pullup resistor. This can be useful when connecting simple switches. A logical "0" will disable it making that input high impedance (open circuit). D9 and D10 can be used for PWM outputs if the Servo() function is not used.

Programming:
The sample code has been written using the Arduino IDE and the processor is loaded with the Arduino Bootloader. When using the Arduino IDE to program the controller, select Arduino Nano with ATmega168 as your board type. You will see three tabs at the top of the program labelled “Wild_Thumper_Controller”, “Constants.h” and “Iopins.h”.

The “Wild_Thumper_Controller” tab contains the main program. This can be changed to suit your needs allowing the controller to be the brains as well as the brawn of your robot.
The “Constants.h” tab stores values such as communications mode, baud rate, minimum battery voltage, maximum allowable current draw etc.
The “I/Opins.h” tab tells the program which of the processors I/O pins are used for what function. Take care when modifying the contents of this tab as many functions are “hard wired” on the PCB.

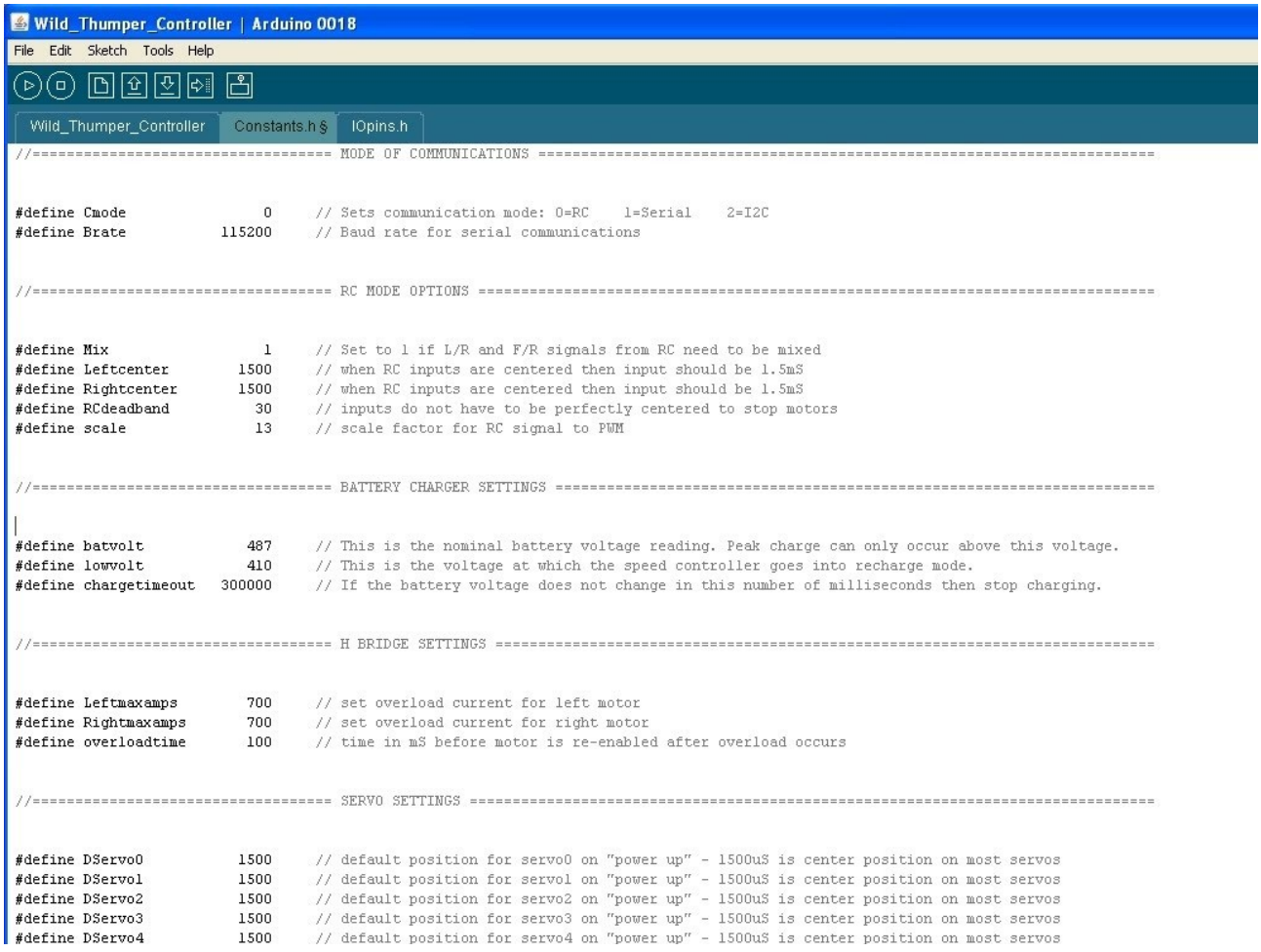
NOTE: The sample code was originally written using version 18 of the Arduino IDE. Newer versions of the Arduino IDE have utilised the definitions A0, A1, etc. For this reason you will need to delete or rename these definitions in the "IOpins.h" tab of the program to prevent compilation errors.



Control and Communications:
The default settings in the sample code allow the controller to be controlled using a standard RC receiver. In this mode pulsewidths from the receiver are measured and converted to speed and direction for the two motors. The default pins for RC mode are D0 and D1. As these are also the serial communications pins used for programming. You must disconnect cables connected to these pins when uploading a new program or usinging the serial port for diagnostics. If you have spare I/O pins then you can redefine "RCleft" and "RCright" in the "IOpins.h" tab of the sample code. Other servos can be plugged directly into the RC receiver to create a fully radio controlled robot.

Other modes of opperation include working as a stand alone unit, processing sensor data and controlling your robot autonomously or slaved to another processor either directly or wirelessly. These modes can be selected in the "Constants.h" tab of the sample code by changing the value of "Cmode". You can also select the baud rate for serial communications if that mode is chosen.

I2C communications is also supported in the hardware using A4 and A5 but it is up to the user to write their own code. I2C communications requires pullup resistors, 20K internal pullup resistors can be enabled by using the digitalWrite() command to write a logical 1 to D18 and D19 (A4 and A5) while they are configured as inputs (input is the default setting on power up).




```
//================================== MODE OF COMMUNICATIONS ==========================================================================


#define Cmode                0     // Sets communication mode: 0=RC    1=Serial    2=I2C
#define Brate           115200     // Baud rate for serial communications


//================================== RC MODE OPTIONS =================================================================================


#define Mix                  1     // Set to 1 if L/R and F/R signals from RC need to be mixed
#define Leftcenter        1500     // when RC inputs are centered then input should be 1.5mS
#define Rightcenter       1500     // when RC inputs are centered then input should be 1.5mS
#define RCdeadband          30     // inputs do not have to be perfectly centered to stop motors
#define scale               13     // scale factor for RC signal to PWM


//================================== BATTERY CHARGER SETTINGS ========================================================================

#define batvolt            487     // This is the nominal battery voltage reading. Peak charge can only occur above this voltage.
#define lowvolt            410     // This is the voltage at which the speed controller goes into recharge mode.
#define chargetimeout   300000     // If the battery voltage does not change in this number of milliseconds then stop charging.


//================================== H BRIDGE SETTINGS ===============================================================================


#define Leftmaxamps        700     // set overload current for left motor
#define Rightmaxamps       700     // set overload current for right motor
#define overloadtime       100     // time in mS before motor is re-enabled after overload occurs


//================================== SERVO SETTINGS ==================================================================================


#define DServo0           1500     // default position for servo0 on "power up" - 1500uS is center position on most servos
#define DServo1           1500     // default position for servo1 on "power up" - 1500uS is center position on most servos
#define DServo2           1500     // default position for servo2 on "power up" - 1500uS is center position on most servos
#define DServo3           1500     // default position for servo3 on "power up" - 1500uS is center position on most servos
#define DServo4           1500     // default position for servo4 on "power up" - 1500uS is center position on most servos
```