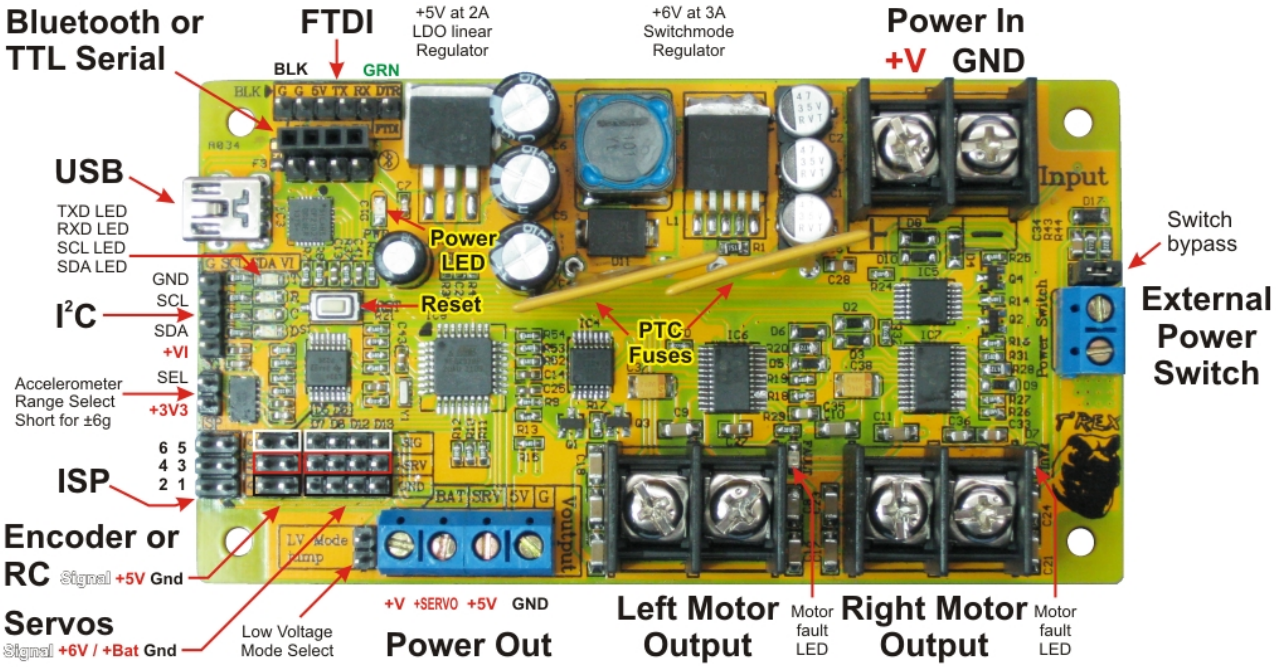
# T'REX ROBOT CONTROLLER




**Introduction:**

The T'REX controller from DAGU is an Arduino compatible robot controller designed to power and control servos and brushed motors. Supplied sample code allows the controller to connect to a radio control receiver, external controller via I²C or optional bluetooth interface. A sample app allows the controller to be controlled by an Android phone or tablet.

The ATmega328P controller comes with the Arduino bootloader and customer software can be uploaded via the built in USB interface, ISP or FTDI sockets.

The Dual H bridges are rated for stall currents of 40A per motor and average currents of 18A per motor. Factory calibrated hall-effect sensors measure current draw of each motor. Each motor has independent variable electronic braking. Self-resetting PTC fuses prevent damage from stalled motors.

The built in switch mode power supply allows the controller to work over a wide range of voltages from 6V – 30V and delivers 3A of current at 6V for driving servos. The 5V LDO regulator can deliver in excess of 2A if required for powering an external controller.

A built in 3-axis accelerometer provides robots with angle, acceleration and impact data.

# Contents

**Power:**
The T'REX controller is rated for a maximum voltage of 30V and can handle currents in excess of 40A per motor. **Note:** Although rated for 30V, a 24V SLA battery will typically be 28.4V when fully charged and may exceed 30V when fast charging so make sure that your system will not exceed 30V or disconnect the batteries when charging.

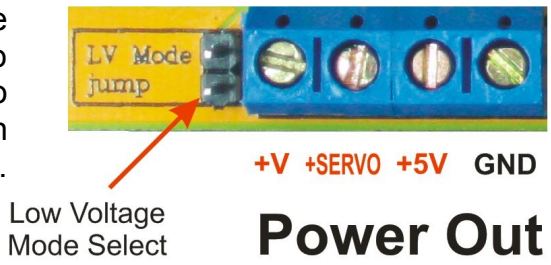
A high current FET is used for switching power allowing low current power switches to be used.

The T'REX controller includes a battery monitor on A3. The sample code will shut down if the battery voltage falls below 5.5V but this value can be changed in I2C mode or by editing the sample code.

**Heat Dissipation:**
Heat generation is minimised by using FET's with a much higher current rating than required. This also allows the controller to handle short-term overloads without damage.

The 4-layer PCB uses the 2 middle layers to conduct heat throughout the PCB making the entire PCB into one large heat sink. If total current draw from the battery exceeds 20A continuous then a heatsink or small fan will be required.

**Low Voltage Mode:**
Use low voltage mode when using low voltage batteries such as 6V SLA, 7.2V NiMh and 7.4V LiPo batteries. Short the pins on the LV mode jumper to bypass the switch mode regulator and power both servos and the 5V regulator directly from the battery. The 5V regulator will get hotter in this mode.




**High Voltage Mode:**
When using batteries with a nominal voltage higher than 8V leave the LV mode jumper pins disconnected. The switch mode regulator efficiently reduces battery voltage to 6V for powering servos and the 5V regulator. In this mode, the maximum current for servos and the 5V regulator combined is 3A.

The 5V regulator can deliver 2A for powering external controllers such as the Raspberry Pi, Beagle Bone or other Arduino controllers.

**Dual H Bridges:**
The H bridges power motors directly from the battery and allow software to control the speed, direction and electronic braking of each motor individually. The PWM frequency has several software selectable settings from 30.5Hz to 31250Hz. The default frequency is 122Hz as not all motors can operate efficiently at higher frequencies.

Factory calibrated, hall-effect, current monitoring sensors allows software to deal with dangerous stall conditions where high currents could otherwise damage the motors or exceed the safe limit of the battery.

Self-resetting PTC fuses provide additional protection in extreme circumstances. The fuses can handle short-term overloads and surges when a motor starts or reverses suddenly without tripping. The slow response time of these fuses allow the software time to handle the situation. The PTC fuses have a maximum rating of 40A each.

**Encoders:**
In I²C mode encoders can be connected to digital pins D5 and D6 (make sure servo positions 4 & 5 are set to 0 to disable servo pulses). The sample code will monitor these pins and count all state changes. If the motor is going forward the count will increase. If the motor is going in reverse the count will decrease. Applying the brake when motor speed is 0 will reset the count for that motor.

**Servos:**
The T'REX controller can drive up to 4 servos at 6V in high voltage mode or battery voltage (6V – 7.4V) in low voltage mode. An additional 2 servos can be powered at 5V if encoders are not being used.

**3-Axis Accelerometer:**
The T'REX controller includes a 3-axis accelerometer on analog inputs A0-A2. The sample code provided uses this sensor to determine direction and magnitude of impacts. Raw sensor data is also available to external controllers for determining the angle and acceleration of the robot. The default sensitivity setting for the accelerometer is ±1.5g but shorting the SG pins with a jumper will change the sensitivity to ±6g.

**I²C Interface:**
The T'REX controller has an I²C interface with automatic voltage translation for logic voltages from 1.8V to 5V. This allows the controller to be slaved to a wide range of external controllers.

The sample code supplied allows an external controller to take full control of motors and servos as well as reading the voltage, current, and 3-axis acceleration sensors.

The default I²C slave address is 0x07 (7 decimal) however this can be changed through the I2C interface with the new address being stored in the EEPROM so it will not be lost when power is disconnected. The default clock speed is 100kHz but is software selectable between 100kHz and 400kHz.

**Bluetooth:**
The T'REX controller allows a DAGU blue tooth module to plug directly into the controller so that it can be controlled by a mobile phone. This allows customers to write mobile phone apps that combine the mobile phone's GPS, Compass and Camera with the sensors on the controller for a mobile phone controlled robot.
**Note:** The blue tooth module should be disconnected when uploading new code using the USB or FTDI interfaces as they share the same serial interface.

**Uploading Code using either USB, ISP or FTDI:**
The T'REX controller can be powered and programmed by USB, ISP and FTDI interfaces. The controller can be powered by these interfaces for the purpose of uploading new code although motor and accelerometer functions will be disabled.

**Note:** although the ISP socket can be used for burning the bootloader, overwriting the bootloader and uploading code directly it is not suitable for interfacing other ISP devices as some of it's pins are dedicated to motor control functions. Battery power should be turned off when uploading new code to prevent unexpected movement from the robot.

**Sample Code:**
The T'REX robot controller comes pre-loaded with the "Arduino Nano w/ 328" bootloader and sample code that lets you use the controller right out of the box. The sample code was written and can be edited using the Arduino IDE 1.04. This software can be downloaded here: http://arduino.cc/en/Main/Software

The T'REX controller uses the CP2102 USB interface from Silicon Labs. Drivers for Windows, Macintosh and Linux can be downloaded here:
http://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx

**Primary Controller:**
The T'REX can be used as the primary controller for a robot. As the sample code was written with the Arduino IDE it can be easily modified or replaced to suit your needs.

As the only spare I/O pins are normally used for servos and encoders the best way to add more sensors and devices is via the I²C interface.

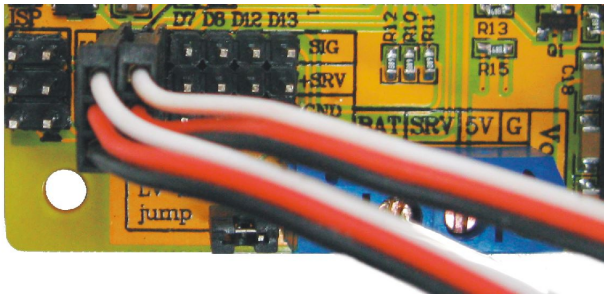
Advanced users may wish to incorporate the GPS, compass, camera and WiFi capabilities of a mobile phone using the optional bluetooth module.

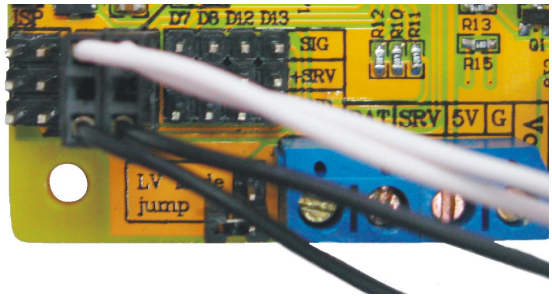
**Radio Control Mode:**
RC mode is the simplest method of control with the T'REX controller behaving as a motor controller and BEC. Two 3-wire female-to-female cables are provided for connecting your RC receiver to the T'REX controller.

**Note:** the supplied cables will power the receiver from +5V when connected as shown. If you wish to power the receiver and additional servos from a different voltage then remove the red wire from the cables to prevent damage.

The sample code automatically looks for RC control signals when power is first applied. This means your radio transmitter should be turned on before you turn on the robot. The motors will quietly beep 3 times to indicate RC mode is selected. If the signal is lost while in RC mode then the motors will shut down until the signal returns.

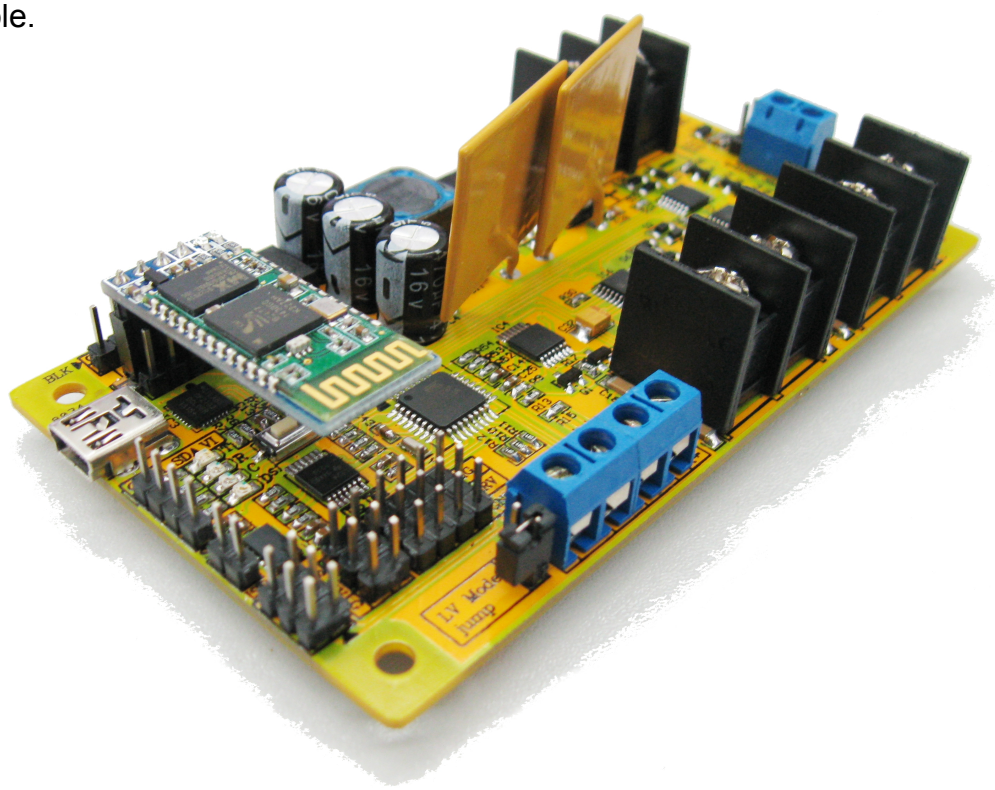
Connect the receiver throttle channel to D5 and the steering channel to D6. The receiver should be powered from +5V to prevent damage to the T'REX controller.




As the T'REX tank chassis and most other robot chassis's use skid steering, the sample code mixes the steering and throttle channels by default to produce left and right motor speeds. This can be changed by editing the code in the tab - **[RCmode]**.

**Bluetooth Mode:**
If no RC signals are detected on power up the sample code will then check if the DAGU bluetooth module is connected. This small, low cost module plugs directly into the PCB. If necessary it can be mounted away from the PCB for better signal reception using the supplied 4-wire cable.



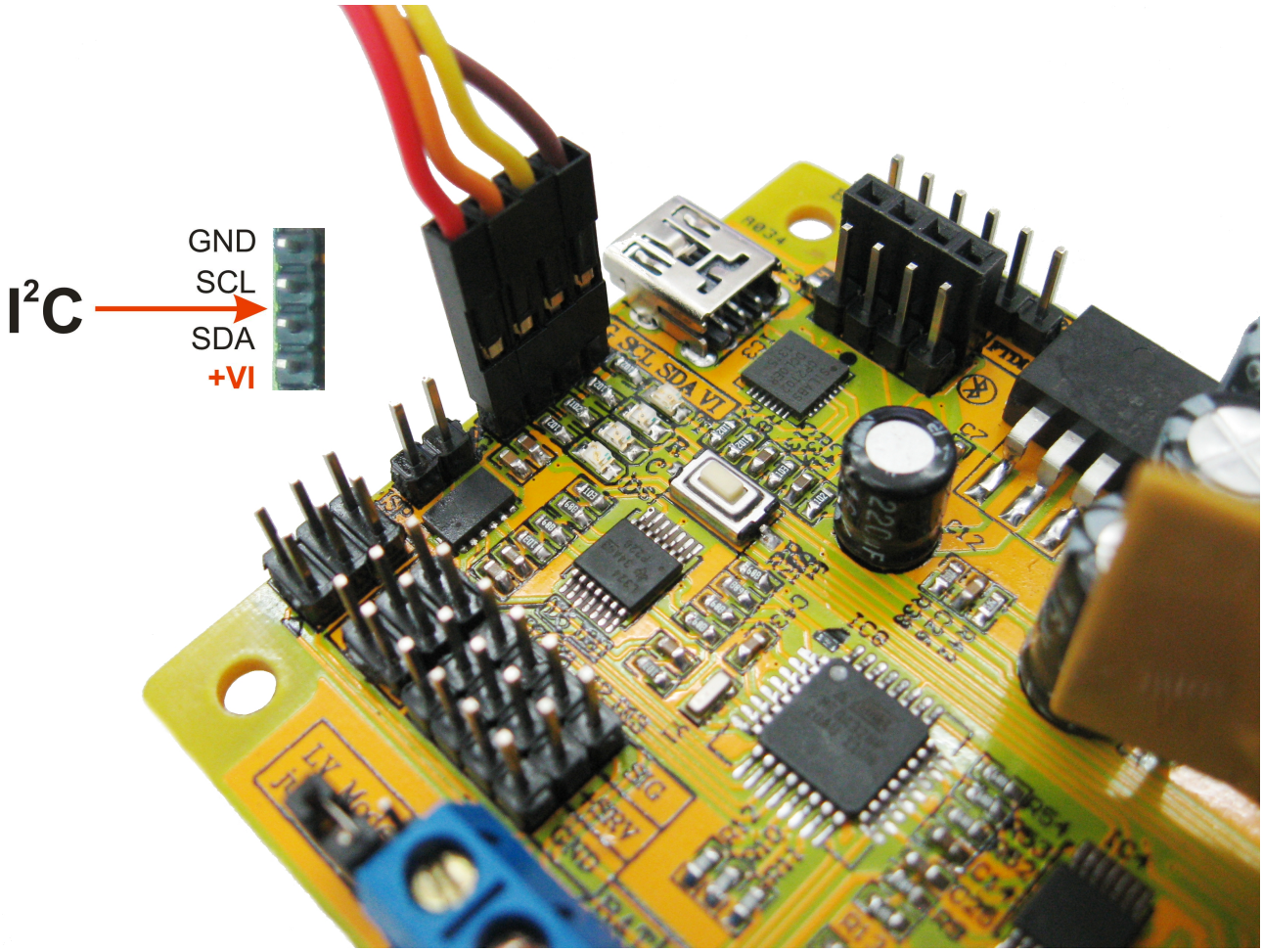
If the bluetooth module is detected the sample code will automatically set the baud rate to 9600, the name to “T'REX” and the pin number to “1234”.

Wait for at least 6 seconds after turning on the T'REX controller before you start the T'REX Android app. This allows time for the controller to configure the bluetooth module.

Touching the dinosaur head will open an options box to allow you to scan for and select the T'REX bluetooth interface. Once selected, wait a few seconds for the message “T'REX connected” to appear.


You can now use the left and right sliders to control the left and right motors. Releasing the sliders will cause them to automatically return to “stop” position. If the T'REX controller detects an impact then the phone will vibrate.

**I²C Mode:**
This is the preferred method of control when using an external controller. All of the controller functions are available in this mode. One 4-wire female-to-female cable is supplied for I²C interfacing.




Connect the +V, SCL, SDA and Gnd of the external logic to the +VI, SCL, SDA and GND pins of the T'REX I²C interface. The T'REX will automatically translate the voltage. The interface includes 10K pullup resistors although external pullup resistors can be added if required.

When using I²C mode for the first time the T'REX controller joins the I2C I²C bus as a slave device with the address 0x07 (decimal 7). This can be changed using the command data packet and the new address will be stored in the controllers EEPROM.

The I²C clock speed is 100kHz on start up but can be changed to 400kHz with the command data packet.

Command data packets consist of 27 bytes and provide full control over the motors and servos as well as configuration settings.

Status data packets consist of 24 bytes and provide an error flag, battery voltage, motor currents, encoder counts, angle and impact data.

**Command Data Packet:**
When an external controller wants to control the T'REX then it must send 27 bytes of data as listed below:

1. Start byte – must be 0x0F (15 decimal)
2. PWM frequency – a number from 1 to 7 to select motor PWM frequency
3. Left motor speed high byte
4. Left motor speed low byte
5. Left motor brake – 0=brake off 1=brake on
6. Right motor speed high byte
7. Right motor speed low byte
8. Right motor brake – 0=brake off 1=brake on
9. Servo 0 position high byte
10. Servo 0 position low byte
11. Servo 1 position high byte
12. Servo 1 position low byte
13. Servo 2 position high byte
14. Servo 2 position low byte
15. Servo 3 position high byte
16. Servo 3 position low byte
17. Servo 4 position high byte
18. Servo 4 position low byte
19. Servo 5 position high byte
20. Servo 5 position low byte
21. Accelerometer de-vibrate 0-255 default=50
22. Impact sensitivity high byte
23. impact sensitivity low byte
24. lowbat high byte
25. lowbat low byte
26. I²C address 0-127
27. clock frequency – 0=100kHz 1=400kHz

**Start byte:** this is a simple method of error checking. If the start byte is not 0x0F or the Master does not send 27 bytes then the receiving buffer is flushed of potentially corrupt data and bit 0 of the error flag is set. The external controller should request a status update after sending command data to check the error flag.

**PWM frequency:** Timer 2 is used to control the PWM (**P**ulse **W**idth **M**odulation) frequency of the dual H bridge. By default this set to 122Hz.

Many smaller motors cannot work efficiently at high frequencies because their inductive reactance increases with frequency and limits the flow of current. The full range of PWM frequencies are:

1. 31250Hz – Silent but many motors will have poor torque and can overheat
2. 3906Hz – Not recommended, high pitch noise and poor torque
3. 977Hz – Not recommended, too much noise
4. 488Hz – Not recommended, too much noise
5. 244Hz – Not recommended, too much noise
6. **122Hz – Default setting – Good torque, low noise**
7. 31Hz – Best torque but more vibration.

**Motor speed:** This is a value from +255 to -255. Negative values indicate a reverse direction.

**Motor brake:** When this byte is non-zero electronic braking occurs. In brake mode the motor speed controls braking. Negative speed values are treated as positive values when braking so that a motor speed of -255 or +255 will both equal 100% brake.

**Servo positions 0-5:** A maximum of 6 servos can be controlled by the T'REX controller. A value of 1500 is equal to a pulse with a width of 1500uS which will centre the servo. Typically the servo position should be a value between 1000 and 2000 although it will vary depending on the servos used.

Negative values will move the servo the same as positive values except the sense of direction will be reversed. A value of 0 indicates no servo connected and no control pulses will be sent to that pin.

**De-vibrate:** After an impact occurs the chassis may continue to vibrate for a short time and this could lead to false triggering. The de-vibrate variable sets a time period in 2mS units where impact sensing is temporarily disabled. The default setting is 50 which equals a time period of 100mS.

**Impact Sensitivity:** There is always acceleration and vibration when a robot moves. The sample code looks for sudden changes in accelerometer readings over a 2mS period to indicate an impact has occurred. Adjust the sensitivity to prevent false triggering due to vibration or rough terrain. Values greater than 0 are used to change the sensitivity setting. The default is 50.

**Low Battery Voltage:** Some batteries can be damaged if their voltage falls too low. The T'REX controller will automatically shut down if the battery voltage is too low. A value between 550 and 3000 will set the minimum safe voltage between 5.5V and 30V. By selecting a value higher than the battery voltage you can force a shut-down

**I²C address:** The default I²C address is 0x07 (7 decimal) but this can be changed at any time. The address is stored in EEPROM memory so it is not forgotten when the power is turned off.

**I²C clock frequency:** When power is first applied or the controller is reset then the I²C clock frequency will default to 100kHz. By setting this byte to a non-zero value the clock frequency can be changed to 400kHz.

**Note:** After changing the I²C clock frequency the master must change it's clock frequency to match.

**Status Data Packet:**
The I²C master can request a status update from the T'REX controller at any time. The T'REX will return 24 bytes of data as listed below:

1. Start byte – will be 0xF0 (240 decimal)
2. Error flag – 0 indicates the last command packet was received ok
3. Battery voltage high byte
4. Battery voltage low byte
5. Left motor current high byte
6. Left motor current low byte
7. Left encoder count high byte
8. Left encoder count low byte
9. Right motor current high byte
10. Right motor current low byte
11. Right motor encoder count high byte
12. Right motor encoder count low byte
13. Accelerometer X-axis high byte
14. Accelerometer X-axis low byte
15. Accelerometer Y-axis high byte
16. Accelerometer Y-axis low byte
17. Accelerometer Z-axis high byte
18. Accelerometer Z-axis low byte
19. Impact X-axis high byte
20. Impact X-axis low byte
21. Impact Y-axis high byte
22. Impact Y-axis low byte
23. Impact Z-axis high byte
24. Impact Z-axis low byte

**Start byte:** This byte should be 0x0F (15 decimal). If not then clear the buffer, wait at least 2mS and repeat the status requested again.

**Error flag:** A non-zero value indicates the previous command data packet contained errors. Different bits are set depending on the nature of the error.

BIT 0: Start byte not received or incorrect data packet size.
BIT 1: PWM frequency was not 1-7.
BIT 2: Left or right motor speed was not -255 to +255.
BIT 3: One or more servo positions was not -2400 to +2400.
BIT 4: Impact sensitivity not 0-1023.
BIT 5: Low battery was not 550 to 3000 (5.5V to 30V).
BIT 6: I²C slave address was not 0-127.
BIT 7: I²C speed not 0 or 1 (100kHz or 400kHz).

If bits 0 or 2 are set then the motors will shut down to prevent a possible collision.

**Battery voltage:** An integer that is 100x the actual voltage. This gives a precision of 0.01V so that a value of 1527 would equal 15.27V.

**Motor current:** Current drawn by the motor in mA.

**Encoder count:** Optional encoders can be attached to D5 (right) and D6 (left). When the motor speed is positive the encoder counter will increment every time the input changes state. If the speed is negative then the counter will decrement when the input changes state.

Engaging the left brake with the left speed set to 0 will reset the left counter. Engaging the right brake with the right speed set to 0 will reset the right counter.

**Accelerometer readings:** This is the raw data from the accelerometer and can be used to determine the angle and acceleration of the robot.

**Impact readings:** The sample code reads the accelerometer every 2mS and compares the old readings to the new readings. A significant change in values indicates an impact has occurred.

When an impact is detected the relative change in the X, Y and Z axes are given and further impact detection is disabled for 100mS to prevent false triggering from chassis vibration. Values of 0 indicate no impact has been detected. Sensitivity and de-vibrate settings can be adjusted in the command data packet.

## T'REX Robot Controller Specifications:

### Processor:

| | |
|---|---|
| MCU: | ATmega328P |
| Clock speed: | 16MHz |
| Logic voltage: | 5V |
| FLASH: | 32K |
| SRAM: | 2K |
| EEPROM: | 1K |
| Bootloader: | Arduino Nano w/ ATmega 328 |

### Power supply:

| | |
|---|---|
| Supply voltage: | 6V – 30V |
| Power switching FET maximum current: | 20A continuous – 110A peak |
| Switch mode regulator: | 6V, 3A maximum, 52kHz |
| +5V output current: | 2A continuous – 3A peak |

### Dual H bridge:

| | |
|---|---|
| Voltage: | Motors powered directly from battery |
| Current per motor: | 9A continuous – 40A peak |
| Current sensor: | Sensitivity 100mV / Amp |
| Fuse type: | 30V / 9A self-resetting PTC |
| Fuse resistance: | 0.005Ω – 0.020Ω |
| Fuse hold current (23°C): | 9A continuous |
| Fuse trip current (23°C): | 18A minimum |
| Fuse maximum current (23°C): | 40A maximum |
| Electronic braking current: | 85A maximum |

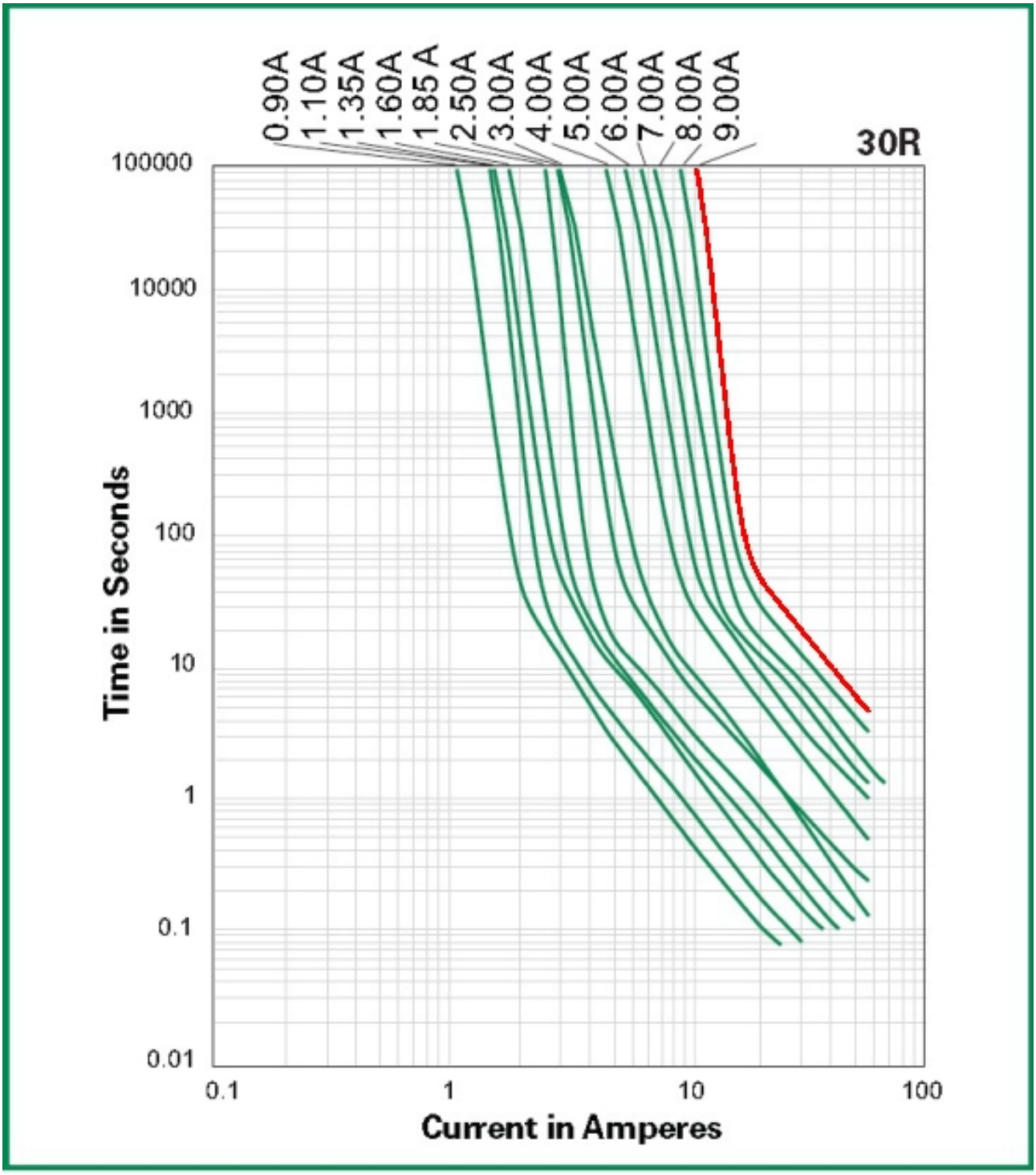
### Accelerometer:

| | |
|---|---|
| Type: | MMA7361L – Freescale Semiconductor |
| Voltage: | 3.3V |
| Number of axes: | 3 |
| Sensitivity: | ±1.5g or ±6g |
| Output: | Analog (amplified from 3.3V to 5V scale) |

### USB interface:

| | |
|---|---|
| Type: | CP2102 – Silicon Labs |
| Drivers for Windows, Macintosh and Linux: | http://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx |

Typical PTC fuse response time.




The average time current curves and Temperature Rerating curve performance is affected by a number or variables, and these curves provided as guidance only. Customer must verify the performance in their application.