

- [Board index](#) < [Pololu Robotics and Electronics Product Support](#) < [Pololu Servo Controllers](#)
- [Change font size](#)
- [Print view](#)
- [Advanced search](#)
- [FAQ](#)
- [Register](#)
- [Login](#)

## Mini Maestro 12 and Python Problems

[Post a reply](#)

3 posts • Page **1** of **1**

### Mini Maestro 12 and Python Problems

by [dabo](#) » Fri Jun 22, 2012 3:16 pm

Hello,

Using the following python code, I'm able to move the (SM-S4303R) Continuous Rotation Servo with the Mini Maestro 12 USB connected to a Win7 AMD64 computer running Python 2.3.7 and PySerial, without any problems.

However running the exact same code on on another Win7 (Intel) with Python 2.3.7 and PySerial, using the same Servo and maestro, does NOT work? The controller seems to receive the data and I don't get any errors but the servo does not move.

On both machines I can use the servo with the Maestro Control Center, this is why I believe it is most likely a Python related issue.

Here is the code I'm running on both machines: (I'm adjusting for the different COM ports, and USB Dual Port mode is enabled)

Code: [Select all](#) [Expand](#)

```
#!/usr/bin/python
#####
# Filename:
#   Device.py
#####
# Project Authors:
#   Juhapekka Piironen
#   Brian Wu
#
# Changes:
#   June 14, 2010 by Juhapekka Piironen - changes committed to svn
#       - added comments for the device commands according to the manual from Pololu
#       - added latest draft code for rotating base servo (Parallax Continuous Rotating Servo)
#       - note! you should be able to clear error flags with .get_errors function according to the manual
#       - renamed CameraDriver to LegacyCameraDriver as Brian Wu has done better one
#       - integrated batch of changes provided by Brian Wu
#
#   June 11, 2010 by Brian Wu - Changes committed thru email
#       - Decoupling the implementation from the program
#
#   April 19, 2010 by Juhapekka Piironen
#       - Initial Release
#
# Email:
#   juhapekka.piironen@gmail.com
#
# License:
#   GNU/GPLv3
#
# Description:
#   A python-wrapper for Pololu Micro Maestro 6-Channel USB Servo Controller
#
#####
# /\! Notes /\!
# You will have to enable _USB Dual Port_ mode from the _Pololu Maestro Control Center_.
#
#####
# Device Documentation is available @ http://www.pololu.com/docs/pdf/0J40/maestro.pdf
#####
# (C) 2010 Juhapekka Piironen
#       Brian Wu
#####
import serial
```

```

import time

def log(*msgline):
    for msg in msgline:
        print msg,
    print

class Device(object):
    def __init__(self, con_port="COM6", ser_port="COM7", timeout=1): #/dev/ttyACM0 and /dev/ttyACM1 for Linux
        #####
        # lets introduce and init the main variables
        self.con = None
        self.ser = None
        self.isInitialized = False

        #####
        # lets connect the TTL Port
        try:
            self.con = serial.Serial(con_port, timeout=timeout)
            self.con.close()
            self.con.open()
            log("Link to Command Port -", con_port, "- successful")

        except serial.SerialException, e:
            print e
            log("Link to Command Port -", con_port, "- failed")

        if self.con:
            #####
            #If your Maestro's serial mode is "UART, detect baud rate", you must first send it the baud rate indication byte
            #the RX line before sending any commands. The 0xAA baud rate indication byte can be the first byte of a Pololu
            #command.
            #http://www.pololu.com/docs/pdf/0J40/maestro.pdf - page 35
            self.con.write(chr(0xAA))
            self.con.flush()
            log("Baud rate indication byte 0xAA sent!")

            #####
            # lets connect the TTL Port
            try:
                self.ser = serial.Serial(ser_port, timeout=timeout)
                self.ser.close()
                self.ser.open()
                log("Link to TTL Port -", ser_port, "- successful")

            except serial.SerialException, e:
                print e
                log("Link to TTL Port -", ser_port, "- failed!")

        self.isInitialized = (self.con!=None and self.ser!=None)
        if (self.isInitialized):
            err_flags = self.get_errors()
            log("Device error flags read (" + err_flags + ") and cleared")
            log("Device initialized:", self.isInitialized)

        #####
        ## common write function for handling all write related tasks
        def write(self, *data):
            if not self.isInitialized: log("Not initialized"); return
            if not self.ser.writable():
                log("Device not writable")
                return
            for d in data:
                self.ser.write(chr(d))
            self.ser.flush()

        #####
        ## Go Home
        # Compact protocol: 0xA2
        # --
        # This command sends all servos and outputs to their home positions, just as if an error had occurred. For servos and
        # outputs set to "Ignore", the position will be unchanged.
        # --
        # Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf

```

```

def go_home(self):
    if not self.isInitialized: log("Not initialized"); return
    self.write(0xA2)

#####
## Set Target
# Compact protocol: 0x84, channel number, target low bits, target high bits
# --
# The lower 7 bits of the third data byte represent bits 0-6 of the target (the lower 7 bits), while the lower 7 bits of
the
# fourth data byte represent bits 7-13 of the target. The target is a non-negative integer.
# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def set_target(self,servo,value):
    if not self.isInitialized: log("Not initialized"); return
    highbits,lowbits = divmod(value,32)
    self.write(0x84,servo,lowbits << 2,highbits)

#####
## Set Speed
# Compact protocol: 0x87, channel number, speed low bits, speed high bits
# --
# This command limits the speed at which a servo channel's output value changes. The speed limit is given in units of
(0.25 us)/(10 ms)
# --
# For example, the command 0x87, 0x05, 0x0C, 0x01 sets
# the speed of servo channel 5 to a value of 140, which corresponds to a speed of 3.5 us/ms. What this means is that if
# you send a Set Target command to adjust the target from, say, 1000 us to 1350 us, it will take 100 ms to make that
# adjustment. A speed of 0 makes the speed unlimited, so that setting the target will immediately affect the position.
Note
# that the actual speed at which your servo moves is also limited by the design of the servo itself, the supply voltage,
and
# mechanical loads; this parameter will not help your servo go faster than what it is physically capable of.
# --
# At the minimum speed setting of 1, the servo output takes 40 seconds to move from 1 to 2 ms.
# The speed setting has no effect on channels configured as inputs or digital outputs.
# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def set_speed(self,servo,speed):
    if not self.isInitialized: log("Not initialized"); return
    highbits,lowbits = divmod(speed,32)
    self.write(0x87,servo,lowbits << 2,highbits)

#####
## Set Acceleration
# Compact protocol: 0x89, channel number, acceleration low bits, acceleration high bits
# --
# This command limits the acceleration of a servo channel's output. The acceleration limit is a value from 0 to 255 in
units of (0.25 us)/(10 ms)/(80 ms),
# --
# A value of 0 corresponds to no acceleration limit. An acceleration limit causes the speed of a servo to slowly ramp up
until it reaches the maximum speed, then
# to ramp down again as position approaches target, resulting in a relatively smooth motion from one point to another.
# With acceleration and speed limits, only a few target settings are required to make natural-looking motions that would
# otherwise be quite complicated to produce.
# --
# At the minimum acceleration setting of 1, the servo output takes about 3 seconds to move smoothly from a target of 1
ms to a target of 2 ms.
# The acceleration setting has no effect on channels configured as inputs or digital outputs.
# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def set_acceleration(self,servo,acceleration):
    if not self.isInitialized: log("Not initialized"); return
    highbits,lowbits = divmod(acceleration,32)
    self.write(0x89,servo,lowbits << 2,highbits)

#####
## Get Position
# Compact protocol: 0x90, channel number
# Response: position low 8 bits, position high 8 bits
# --
# This command allows the device communicating with the Maestro to get the position value of a channel. The position
# is sent as a two-byte response immediately after the command is received.

```

```

# --
# If the specified channel is configured as a servo, this position value represents the current pulse width that the
Maestro
# is transmitting on the channel, reflecting the effects of any previous commands, speed and acceleration limits, or
scripts
# running on the Maestro.
# --
# If the channel is configured as a digital output, a position value less than 6000 means the Maestro is driving the
line low,
# while a position value of 6000 or greater means the Maestro is driving the line high.
# --
# If the channel is configured as an input, the position represents the voltage measured on the channel. The inputs on
# channels 0-11 are analog: their values range from 0 to 1023, representing voltages from 0 to 5 V. The inputs on
channels
# 12-23 are digital: their values are either exactly 0 or exactly 1023.
# --
# Note that the formatting of the position in this command differs from the target/speed/acceleration formatting in the
# other commands. Since there is no restriction on the high bit, the position is formatted as a standard little-endian
two-
# byte unsigned integer. For example, a position of 2567 corresponds to a response 0x07, 0x0A.
# --
# Note that the position value returned by this command is equal to four times the number displayed in the Position box
# in the Status tab of the Maestro Control Center.
# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def get_position(self,servo):
    if not self.isInitialized: log("Not initialized"); return None
    self.write(0x90,servo)
    data = self.ser.read(2)
    if data:
        return (ord(data[0])+(ord(data[1])<<8))/4
    else:
        return None

#####

## Get Moving State
# Compact protocol: 0x93
# Response: 0x00 if no servos are moving, 0x01 if servos are moving
# --
# This command is used to determine whether the servo outputs have reached their targets or are still changing, limited
# by speed or acceleration settings. Using this command together with the Set Target command, you can initiate several
# servo movements and wait for all the movements to finish before moving on to the next step of your program.
# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def get_moving_state(self):
    if not self.isInitialized: log("Not initialized"); return None
    self.write(0x93)
    data = self.ser.read(1)
    if data:
        return ord(data[0])
    else:
        return None

#####

## Get Errors
# Compact protocol: 0xA1
# --
# Response: error bits 0-7, error bits 8-15
# --
# Use this command to examine the errors that the Maestro has detected.
# --
# The error register is sent as a two-byte response immediately after the command is received,
# then all the error bits are cleared. For most applications using serial control, it is a good idea to check errors
continuously
# and take appropriate action if errors occur.
# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def get_errors(self):
    if not self.isInitialized: log("Not initialized"); return None
    self.write(0xA1)
    data = self.ser.read(2)
    if data:
        return ord(data[0])+(ord(data[1])<<8)

```

```

        else:
            return None

#####
    ## a helper function for Set Target
    def wait_until_at_target(self):
        while (self.get_moving_state()):
            time.sleep(0.1)

#####
    ## Lets close and clean when we are done
    def __del__(self):
        if (self.ser):
            self.ser.close()
        if (self.con):
            self.con.close()
        del(self.ser)
        del(self.con)

#####

servo = Device("COM4", "COM5")
servo.set_target(0, 1000)
time.sleep(0.2)
servo.set_target(0, 0)

```

Any thoughts on why it is not working? Anyone have a similar issue?

Any help is greatly appreciated 😊

[dabo](#)

**Posts:** 2

**Joined:** Fri Jun 22, 2012 3:01 pm

[Top](#)

---

### [Re: Mini Maestro 12 and Python Problems](#)

by [dabo](#) » Sat Jun 23, 2012 5:55 am

Solution: Apparently PySerial does not automatically set the Baudrate to 9600 on some machines.

A simple "*self.con.baudrate = 9600*" solved the problem.

[dabo](#)

**Posts:** 2

**Joined:** Fri Jun 22, 2012 3:01 pm

[Top](#)

---

### [Re: Mini Maestro 12 and Python Problems](#)

by [jonnycowboy](#) » Sat Aug 25, 2012 9:05 am

Hi all,  
in case you are running on Python 3 I've made some mods to the code above so it will run and not generate any errors.

Code: [Select all](#) [Expand](#)

```

#!/usr/bin/python
#####
# Filename:
#     Device.py
#####

```

```

# Project Authors:
#   Juhapekka Piiroinen
#   Brian Wu
#
# Changes:
#   June 14, 2010 by Juhapekka Piiroinen - changes committed to svn
#       - added comments for the device commands according to the manual from Pololu
#       - added latest draft code for rotating base servo (Parallax Continuous Rotating Servo)
#       - note! you should be able to clear error flags with .get_errors function according to the manual
#       - renamed CameraDriver to LegacyCameraDriver as Brian Wu has done better one
#       - integrated batch of changes provided by Brian Wu
#
#   June 11, 2010 by Brian Wu - Changes committed thru email
#       - Decoupling the implementation from the program
#
#   April 19, 2010 by Juhapekka Piiroinen
#       - Initial Release
#
# Email:
#   juhapekka.piiroinen@gmail.com
#
# License:
#   GNU/GPLv3
#
# Description:
#   A python-wrapper for Pololu Micro Maestro 6-Channel USB Servo Controller
#
#####
# /\ Notes /\
# You will have to enable _USB Dual Port_ mode from the _Pololu Maestro Control Center_.
#
#####
# Device Documentation is available @ http://www.pololu.com/docs/pdf/0J40/maestro.pdf
#####
# (C) 2010 Juhapekka Piiroinen
#       Brian Wu
#####
import serial
import time

def log(*msgline):
    for msg in msgline:
        print (msg),
    print ()

class Device(object):
    def __init__(self, con_port="COM4", ser_port="COM5", timeout=1): #/dev/ttyACM0 and /dev/ttyACM1 for Linux
        #####
        # lets introduce and init the main variables
        self.con = None
        self.ser = None
        self.isInitialized = False

        #####
        # lets connect the TTL Port
        try:
            self.con = serial.Serial(con_port, timeout=timeout)
            self.con.baudrate = 9600
            self.con.close()
            self.con.open()
            log("Link to Command Port -", con_port, "- successful")

        except (serial.SerialException, e):
            print (e)
            log("Link to Command Port -", con_port, "- failed")

        if self.con:
            #####
            #If your Maestro's serial mode is "UART, detect baud rate", you must first send it the baud rate indication byte
            0xAA on

            #the RX line before sending any commands. The 0xAA baud rate indication byte can be the first byte of a Pololu
            protocol

            #command.
            #http://www.pololu.com/docs/pdf/0J40/maestro.pdf - page 35
            self.con.write(str.encode(chr(0xAA)))
            self.con.flush()
            log("Baud rate indication byte 0xAA sent!")

```

```
#####
# lets connect the TTL Port
try:
    self.ser = serial.Serial(ser_port,timeout=timeout)
    self.ser.close()
    self.ser.open()
    log("Link to TTL Port -", ser_port, "- successful")
except (serial.serialutil.SerialException, e):
    print (e)
    log("Link to TTL Port -", ser_port, "- failed!")

self.isInitialized = (self.con!=None and self.ser!=None)
if (self.isInitialized):
    err_flags = self.get_errors()
    log("Device error flags read (" ,err_flags," ) and cleared")
log("Device initialized:",self.isInitialized)

#####
## common write function for handling all write related tasks
def write(self,*data):
    if not self.isInitialized: log("Not initialized"); return
    if not self.ser.writable():
        log("Device not writable")
        return
    for d in data:
        self.ser.write(str.encode(chr(d)))
    self.ser.flush()

#####
## Go Home
# Compact protocol: 0xA2
# --
# This command sends all servos and outputs to their home positions, just as if an error had occurred. For servos and
# outputs set to "Ignore", the position will be unchanged.
# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def go_home(self):
    if not self.isInitialized: log("Not initialized"); return
    self.write(0xA2)

#####
## Set Target
# Compact protocol: 0x84, channel number, target low bits, target high bits
# --
# The lower 7 bits of the third data byte represent bits 0-6 of the target (the lower 7 bits), while the lower 7 bits of
the
# fourth data byte represent bits 7-13 of the target. The target is a non-negative integer.
# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def set_target(self,servo,value):
    if not self.isInitialized: log("Not initialized"); return
    highbits,lowbits = divmod(value,32)
    self.write(0x84,servo,lowbits << 2,highbits)

#####
## Set Speed
# Compact protocol: 0x87, channel number, speed low bits, speed high bits
# --
# This command limits the speed at which a servo channel's output value changes. The speed limit is given in units of
(0.25 us)/(10 ms)
# --
# For example, the command 0x87, 0x05, 0x0C, 0x01 sets
# the speed of servo channel 5 to a value of 140, which corresponds to a speed of 3.5 us/ms. What this means is that if
# you send a Set Target command to adjust the target from, say, 1000 us to 1350 us, it will take 100 ms to make that
# adjustment. A speed of 0 makes the speed unlimited, so that setting the target will immediately affect the position.
Note
# that the actual speed at which your servo moves is also limited by the design of the servo itself, the supply voltage,
and
# mechanical loads; this parameter will not help your servo go faster than what it is physically capable of.
# --
# At the minimum speed setting of 1, the servo output takes 40 seconds to move from 1 to 2 ms.
# The speed setting has no effect on channels configured as inputs or digital outputs.
```

```

# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def set_speed(self,servo,speed):
    if not self.isInitialized: log("Not initialized"); return
    highbits,lowbits = divmod(speed,32)
    self.write(0x87,servo,lowbits << 2,highbits)

#####
## Set Acceleration
# Compact protocol: 0x89, channel number, acceleration low bits, acceleration high bits
# --
# This command limits the acceleration of a servo channel's output. The acceleration limit is a value from 0 to 255 in
units of (0.25 us)/(10 ms)/(80 ms),
# --
# A value of 0 corresponds to no acceleration limit. An acceleration limit causes the speed of a servo to slowly ramp up
until it reaches the maximum speed, then
# to ramp down again as position approaches target, resulting in a relatively smooth motion from one point to another.
# With acceleration and speed limits, only a few target settings are required to make natural-looking motions that would
# otherwise be quite complicated to produce.
# --
# At the minimum acceleration setting of 1, the servo output takes about 3 seconds to move smoothly from a target of 1
ms to a target of 2 ms.
# The acceleration setting has no effect on channels configured as inputs or digital outputs.
# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def set_acceleration(self,servo,acceleration):
    if not self.isInitialized: log("Not initialized"); return
    highbits,lowbits = divmod(acceleration,32)
    self.write(0x89,servo,lowbits << 2,highbits)

#####
## Get Position
# Compact protocol: 0x90, channel number
# Response: position low 8 bits, position high 8 bits
# --
# This command allows the device communicating with the Maestro to get the position value of a channel. The position
# is sent as a two-byte response immediately after the command is received.
# --
# If the specified channel is configured as a servo, this position value represents the current pulse width that the
Maestro
# is transmitting on the channel, reflecting the effects of any previous commands, speed and acceleration limits, or
scripts
# running on the Maestro.
# --
# If the channel is configured as a digital output, a position value less than 6000 means the Maestro is driving the
line low,
# while a position value of 6000 or greater means the Maestro is driving the line high.
# --
# If the channel is configured as an input, the position represents the voltage measured on the channel. The inputs on
# channels 0-11 are analog: their values range from 0 to 1023, representing voltages from 0 to 5 V. The inputs on
channels
# 12-23 are digital: their values are either exactly 0 or exactly 1023.
# --
# Note that the formatting of the position in this command differs from the target/speed/acceleration formatting in the
# other commands. Since there is no restriction on the high bit, the position is formatted as a standard little-endian
two-
# byte unsigned integer. For example, a position of 2567 corresponds to a response 0x07, 0x0A.
# --
# Note that the position value returned by this command is equal to four times the number displayed in the Position box
# in the Status tab of the Maestro Control Center.
# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def get_position(self,servo):
    if not self.isInitialized: log("Not initialized"); return None
    self.write(0x90,servo)
    data = self.ser.read(2)
    if data:
        return data #(ord(data[0])+(ord(data[1])<<8))/4
        #commented out "ORD": bytes objects in Python 3 are already treated as arrays as integers so the
extra conversion using ord() isn't needed
    else:
        return None

#####

```



```

## Get Moving State
# Compact protocol: 0x93
# Response: 0x00 if no servos are moving, 0x01 if servos are moving
# --
# This command is used to determine whether the servo outputs have reached their targets or are still changing, limited
# by speed or acceleration settings. Using this command together with the Set Target command, you can initiate several
# servo movements and wait for all the movements to finish before moving on to the next step of your program.
# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def get_moving_state(self):
    if not self.isInitialized: log("Not initialized"); return None
    self.write(0x93)
    data = self.ser.read(1)
    if data:
        return data #ord(data[0])
        #commented out "ORD": bytes objects in Python 3 are already treated as arrays as integers so the
extra conversion using ord() isn't needed
    else:
        return None

#####

## Get Errors
# Compact protocol: 0xA1
# --
# Response: error bits 0-7, error bits 8-15
# --
# Use this command to examine the errors that the Maestro has detected.
# --
# The error register is sent as a two-byte response immediately after the command is received,
# then all the error bits are cleared. For most applications using serial control, it is a good idea to check errors
continuously
# and take appropriate action if errors occur.
# --
# Source: http://www.pololu.com/docs/pdf/0J40/maestro.pdf
def get_errors(self):
    if not self.isInitialized: log("Not initialized"); return None
    self.write(0xA1)
    data = self.ser.read(2)
    if data:
        return data #ord(data[0])+(ord(data[1])<<8)
        #commented out "ORD": bytes objects in Python 3 are already treated as arrays as integers so the
extra conversion using ord() isn't needed
    else:
        return None

#####

## a helper function for Set Target
def wait_until_at_target(self):
    while (self.get_moving_state()):
        time.sleep(0.1)

#####

## Lets close and clean when we are done
def __del__(self):
    if (self.ser):
        self.ser.close()
    if (self.con):
        self.con.close()
    del(self.ser)
    del(self.con)

#####

servo = Device("COM4","COM5")
servo.set_target(0,1000)
time.sleep(0.2)
servo.set_target(0,0)

```

thanks!  
Jon

PS Dabo, do you know where this came from?  
thanks!

[jonnycowboy](#)

**Posts:** 1

**Joined:** Sat Aug 25, 2012 9:02 am

[Top](#)

Display posts from previous:  Sort by

[Post a reply](#)

3 posts • Page **1** of **1**

[Return to Pololu Servo Controllers](#)

Jump to:

## Who is online

Users browsing this forum: Google [Bot] and 6 guests

- [Board index](#)
- [The team](#) • [Delete all board cookies](#) • All times are UTC - 8 hours

Powered by [phpBB®](#) Forum Software © phpBB Group

[Home](#) | [Forum](#) | [Blog](#) | [Support](#) | [Ordering Information](#) | [Wish Lists](#) | [Distributors](#) | [BIG Order Form](#) | [About](#) | [Contact](#)

© 2001–2013 Pololu Corporation



[My Account](#) | [Wish Lists](#) | [BIG Order Form](#) | [Shopping Cart](#)

[US toll free: 1-877-7-POLOLU ~ \(702\) 262-6648](#)

[Same-day shipping, worldwide](#)

[Catalog](#) [Forum](#) [Blog](#) [Support](#) [Ordering](#) [Distributors](#) [About](#) [Contact](#)

## [Feedback](#)

[Comments or questions?](#) (opens in new window)

## [Services](#)

- [Custom Laser Cutting](#)
- [SMT Stencils](#)

## [Products](#)

### [New Products](#)

### [Specials!](#)

## [Robot Kits](#)

- [Robot Kits with Soldering](#)
- [Robot Kits without Soldering](#)
- [Tamiya Robot Kits](#)
- [Chassis](#)

## [Electronics](#)

- [Programmable Controllers](#)
- [Motion Control Modules](#)

- [Sensors](#)
- [Regulators and Power Supplies](#)
- [Cables and Wire](#)
- [Connectors](#)
- [Electronics Prototyping](#)
- [Signal Adapters and Extenders](#)
- [Switches, Buttons, and Relays](#)
- [Computer Interface](#)
- [RC Interface](#)
- [Electronics Kits](#)
- [LEDs](#)
- [Audio](#)
- [Discrete Components](#)
- [Displays](#)
- [Batteries](#)
- [Battery Holders](#)
- [Wireless](#)

### **Mechanical Components**

- [Motors and Gearboxes](#)
- [RC Servos](#)
- [Wheels, Tracks, and Ball Casters](#)
- [Hardware](#)
- [Chassis](#)
- [Tamiya Products](#)


### **Tools**

### **Shirts**

### **Combination Deals**

### **Discontinued Items**

## **Community**

- [Pololu Blog](#)
- [Engage Your Brain](#)
- [Community Projects](#)
-  [Feeds](#)