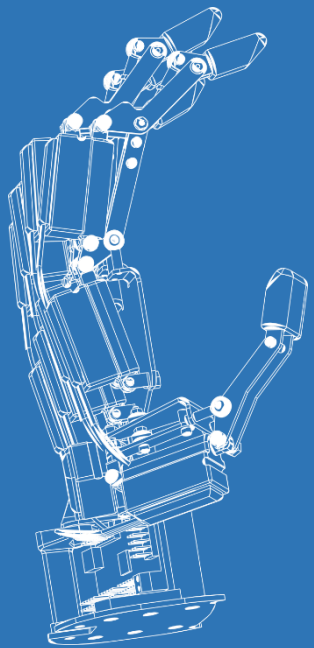


AR10 Hand

10 Degrees of Freedom Humanoid Hand

Getting Started Guide

AR10 Humanoid Robotic Hand

ACTIVE8 ROBOTS

Collaborative Robot Specialists

Contents

Contents

Introduction

The AR10 Robot Hand features 10 degrees of freedom (DOF) that are servo actuated and controlled using a programmable microcontroller. It is designed for use in a teaching, research and lab environment. Manufactured from a hybrid construction, it balances strength and weight. It is an ideal platform to carry out research in the field of robotics. Its capability can be expanded by adding sensors or combining the hand with a robot arm.

The latest up to date files can be downloaded from the AR10 Robotic Hand page, or found on the included USB stick:
http://www.active8robots.com/robots/ar10-robotic-hand/

Contents

AR10 Robotic Hand

USB A - USB mini-B lead

USB stick

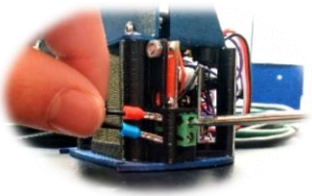
Getting Started

Getting Started

Connecting the hand

On the opposite side to the thumb you will see a mini USB plug. Connect the provided USB lead to the AR10 Robot Hand and your workstation.

On same side there will be the power inputs labelled positive and negative (positive being on top (see picture). Connect a 7.5V – 30V DC power supply to the terminal and tighten the screws as shown.



Controlling the Hand

The hand can be controlled in a number of ways. See below for the relevant pages:

Windows:

Linux:

Installing Pololu Maestro Control Center on Linux

Prerequisites

Libusb will need to be installed. This can be done from a command window in linux with the command:

```
sudo apt-get install libusb-1.0-0-dev mono-runtime
libmono-winforms2.0-cil
```

Installing

Unzip the *maestro-linux-150116.tar* by running "tar -xzvf" followed by the name of the file.
After following the instructions in README.txt, you can run the programs by executing `MaestroControlCenter` and `UscCmd`.

USB Configuration

Open the following folder:

```
cd /etc/udev/rules.d/
gksu gedit
```

You may then be asked to install an application.
Here copy the contents of *99-pololu.rules* to the newly opened document and save it in the current folder with the name *99-pololu.rules*
Then run the following code with the AR10 Hand unplugged from the workstation:

```
sudo udevadm control --reload-rules
```

Running the Program

The following code will run the programs:

```
./MaestroControlCenter
./UscCmd
```

If an error message saying "cannot execute binary file" appears, use the mono command to run the program

```
mono ./UscCmd
```

The *Getting Started section* then explains how to load the calibration file and control the servos.

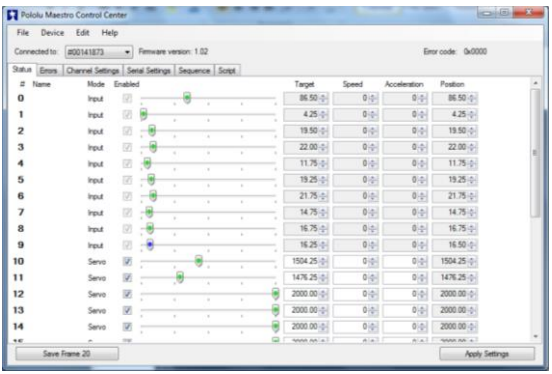
Note: for Windows install the *maestro-windows* file then follow the instructions below

Settings File

Load the *AR10v4_settings* settings file. This should be pre-loaded, but it is good practice to load it up the first time you use the hand. Go to File > Open Settings File
This loads the specific servo limits for each joint. Click *Apply* Settings on the bottom right. This will write the settings to the Maestro servo board. These should not be changed as it will result in joints colliding and may cause damage to the hand.

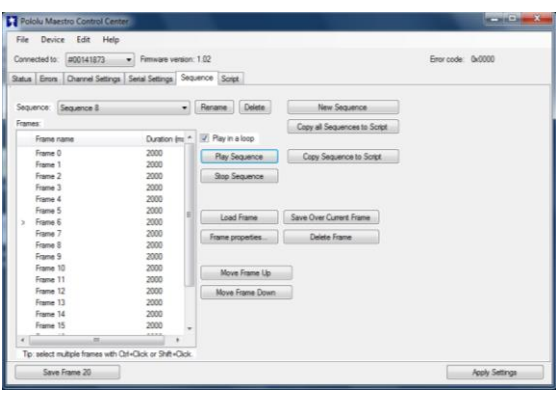
Enabling Servos

Enable the servos in the *Status* tab by ticking the checkboxes. The servos can now be controlled and monitored from this tab using the sliders.



Demo Program

A demo sequence is loaded into the *Sequence 8vi test* file. Load this in the same way and open the S*equence* tab. You should see a sequence displayed. By clicking the *Play Sequence* button, the hand will run through its whole range of motions.



Maestro Scripting

The Sequencer

Simple sequences can be created with the Sequence tab. Each frame denotes a separate position. These sequences can be copied to the Pololu Maestro board in the following way:

Copy Sequence to Script stores the script on the Pololu Maestro to a looped version of the sequence.

Scripts

The *Script* tab enables you to write more complex programs for the AR10 Robotic Hand.

The green *run script* button will process the script one instruction at a time until a *quit* instruction or error is reached. A small pink triangle will jump along the code showing which line is being executed.

The blue *Step Script* button is a useful tool for running through the code step by step.

Run script on startup option in the Script tab enables the script to be run when the hand is powered up. This negates the need for a connection to a PC.

Control Structures

The Maestro Scripting Language Basics

`begin…repeat` is a useful infinite loop to use
`while` loop consumes the top number of the stack and end the loop if the number is a zero. These loops however come after the argument. E.g.

```
1  5 # start with a 5 on the stack
2  begin
3    dup  # copy the number on the stack - the copy will be consumed by WHILE
4    while # jump to the end if the count reaches 0
5    4500 1 servo
6    500 delay
7    2000 1 servo
8    1500 delay
9    1 minus # subtract 1 from the number of times remaining
10 repeat
```

Subroutines

These are useful to call if you need to often return to a specific position such as returning the hand to its open position.

```
sub open
  4500 1 servo
  4500 2 servo
  return
```

By using the command *open* the hand will then return to its open positions. (nb: only 2 servos are being used).

Using Inputs

The potentiometer feedback can be used as an input to a program. External inputs connected to the remaining four pins of the Pololu Maestro can be used. For example: potentiometers, force sensors or EMG muscle activity sensors.

Depending on the value of an input an analogue position for one or several servos can be set correspondingly. This can be scaled to match the range of the servo. For example:

```
  # Sets servo 0 to a position based on an analogue input.
  begin
    1 get_position  # get the value of the input, 0-1023
    2 times 2000 plus # scale it to 2000-4046, approximately 1-2 ms
    5 servo     # set servo 5 based to the value
  repeat
```

A discrete position can also be set depending on the input value:

```
# Set the servo to 2000, 3000, or 4000 depending on an analog input.
begin
  1 get_position  # get the value of the input, 0-1023
  dup 200 less_than
  if
    2000   # go to 2000 for values 0-199
  else
    dup 500 less_than
    if
      6000 # servo to position 3000 for values 200-499
    else
      8000 # servo to position 4000 for values 500-1023
    endif
  endif
  5 servo
  drop  # remove the original copy of the input value
repeat
```

Note. When the input value is close to the limit denoted in the program. E.g. 200 or 800, noise in the digital to analogue conversion may lead to the servo jumping back and forth at random. The use of hysteresis is a way to overcome this:

```
# Set the servo to 2000, 3000, or 4000 depending on input, with hysteresis.
begin
  2000 0 200 servo_range
  3000 200 500 servo_range
  4000 500 1023 servo_range
repeat

# usage: <pos> <low> <high> servo_range
# If the input is in the range specified by low and high,
# keeps servo 5 at pos until the input (inp) moves out of this
# range, with hysteresis.
sub servo_range
  inp 2 pick less_than logical_not  # >= low
  inp 2 pick greater_than logical_not # <= high
  logical_and
  if
    begin
      inp 2 pick 10 minus less_than logical_not  # >= low - 10
      inp 2 pick 10 plus greater_than logical_not # <= high + 10
      logical_and
    while
      2 pick 5 servo
    repeat
  endif
  drop drop drop
  return

sub pot
  1 get_position
  return
```

Here a range of +/- 10 is used for making a transition. The value has to reach 10 above or below the threshold before the servo will make the transition. This value can be changed to overcome noise.

More information on scripting can be found in the *Pololu Maestro User Manual*.

Maestro Language

These are useful functions to call when using the Pololu Maestro scripting language.

keyword	stack effect	description
BEGIN	none	marks the beginning of a loop
ENDIF	none	ends a conditional block IF…ENDIF
ELSE	none	begins the alternative block in IF…ELSE…ENDIF
GOTO *label*	none	goes to the label *label* (define it with *label*:)
IF	-1	enters the conditional block if the argument is true (non-zero) in IF…ENDIF or IF…ELSE…ENDIF
REPEAT	none	marks the end of a loop
SUB *name*	none	defines a subroutine *name*
WHILE	-1	jumps to the end of a loop if the argument is false (zero)
QUIT	none	stops the script
RETURN	none	ends a subroutine
DELAY	-1	delays by the given number of milliseconds
SPEED	-2	Sets the *speed* of the channel specified by the top element to the value in the second element
ACCELERATION	-2	Sets the *acceleration* of the channel specified by the top element to the value in the second element
GET_POSITION	-1,+1	Get the *position* of the channel specified by the top element
SERVO	-2	Sets the *target* of the channel specified by the top element f the value in the second element, in the units of 0.25µs

Using Python

A Python class has been developed, enabling the AR10 Robotic Hand to be programmed in Python.

The AR10 Python Class can be used to communicate with the Pololu Maestro board and control the servos.
It includes several defined functions, as listed in the tables on pages 13 and 14.

USB Dual Port

Ensuring USB Dual Port. This should be set by default, however if not it can be found in the *Serial Settings* tab of the Pololu Maestro control Center

Permissions

The following will ensure you have permissions to control the USB port

```
Sudo adduser MY_USER_NAME dialout
```

You also need to ensure the ttyACM0 is read/writeable by using the following code:

```
Cd /dev
Sudo chmod 666 ttyACM0
```

This will need to be run again every time the hand is connected to the workstation.

Libusb

Installing the 'libusb' Pyhton module will enable communication to the USB when running the Python code.

This can be done with the following code

```
        sudo apt-get install libusb-1.0-0-dev mono-runtime
libmono-winforms2.0-cil
```

Limb Limits

These are set in the settings file written to the Pololu Maestro board. These should not be changed as it will result in joints colliding and may cause damage to the hand.

Calibration Files

There are two calibration files included:

AR10_calibrate.py
AR10_check_calibration.py

Running the AR10_calibrate.py files will run the hand through its range of motions and check the position it is receiving on the feedback channel and calibrate the hand accordingly.

```
Joint = 0 y intercept = 8155.78362826 slope = -6.34587274929
Joint = 1 y intercept = 8295.09535433 slope = -6.92578464618
Joint = 2 y intercept = 8256.94844543 slope = -6.64925888806
```

The above will be displayed on the console and run through all the joints. These will then be written into *calibration_file*. This should be kept in the same directory as the Python calibration scripts.

Running the AR10_check_calibration.py will run the hand through its range of motions and display the *target position* set for each joint, and the *position* read and the error. It is normal to have small errors.

It should look like this:

```
Joint = 0 set position = 4544 position = 4442 error = 102
Joint = 0 set position = 5000 position = 4911 error = 89
Joint = 0 set position = 5500 position = 5419 error = 81
```

The code will continue to then run through all the joints.

Writing your own code

The AR10_class needs to be imported. Functions such as those listed on page 13 and 14 can then be used, such as:

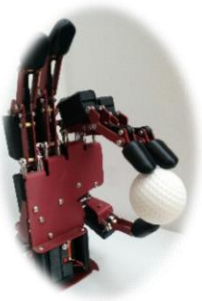
```
1 import AR10_class
3
4 hand = AR10_class.AR10()
5
6 hand.move(0, 5500)
```

Demo Program

Demo Program

A demo program called AR10_hand.py has been included. It is a useful reference for using different functions. Make sure the AR10_class.py is in the same root folder.

It includes the use of joint moves, finger moves, setting speed and acceleration as well as pre programmed grips for holding a tennis ball or a golf ball.



Python Commands

This is a list of functions that can be called when the AR10_class is referenced.

keyword	description
get_read_position(self, channel)	Have servo outputs reached their targets? This is useful only if Speed and/or Acceleration have been set on one or more of the channels.
get_position(self, channel)	As above but returns True or False.
get_moving_state(self)	As above but for the moving state of the joints.
run_script(self, subNumber)	Run a Maestro Script subroutine in the currently active script. Scripts can have multiple subroutines, which get numbered sequentially from 0 on up. Code your Maestro subroutine to either infinitely loop, or just end (return is not valid).
stop_script(self)	Stops the current Maestro script.
move(self, joint, target)	Moves the defined joint to the target position.
wait_for_hand(self)	Waits for joints to stop moving.

Python Commands

Keyword	Description
open_hand(self)	Opens the hand.
close_hand(self)	Closes the hand.
close(self)	Closes the USB serial port.
change_speed(self, speed)	Changes the speed setting of the servos.
set_speed(self, channel)	Set the speed of an individual channel. A speed of 1 will take 1 minute, a speed of 60 would take 1 second and a speed of 0 is unlimited.
change_acceleration(self, acceleration)	Changes the acceleration of the servos. This provides soft starts and finishes when servo motors are set to target positions.
set_acceleration(self, channel, acceleration)	Sets the acceleration of individual channels. Valid values are from 0 to 255. 0=unlimited, 1 is slowest start. A value of 1 will take the servo about 3s to move between 1ms to 2ms range.
set_target(self, channel, target)	Sets a particular channel to a specific target value. The range of values is from 4200 to 7950.
joint_to_channel(self, joint)	Converts a joint number to a channel number.
get_set_position(self, joint)	Get the current position of the device on the specified channel . The result is returned in a measure of quarter-microseconds, which mirrors the Target parameter of set target. This is not reading the true servo position, but the last target position sent to the servo. If the Speed is set to below the top speed of the servo, then the position result will align well with the actual servo position, assuming it is not stalled or slowed.

ROS Setup

Requirements

Ubuntu 14.04
ROS Indigo – For installation instructions please visit wiki.ros.org/indigo/Installation/Ubuntu

Other versions of Ubuntu and ROS may be compatible however at this given time, certain features may not function as expected.

Prerequisites

An Understanding of Linux based systems
An Understanding of ROS, Recommended Tutorials : wiki.ros.org/ROS/Tutorials
USB permissions to communicate with the hand, can be found in the Maestro control center setup guide on page 6

Installation

Download the latest version of the AR10 ROS Package from: www.active-robots.com/ar10-humanoid-robotic-hand

Unzip the file and place the folders ar10, ar10_description and ar10_moveit to the src folder in the ROS workspace.

The .py files in the scripts folder need to be made into executables so they can be run. This is done using the following command from the ROS workspace:

```
chmod +x src/ar10/scripts/*.py
```

The ROS package should now be ready to use.

Using ROS

Whenever a new terminal is opened in linux the setup.bash file must be sourced in order to gain access to the ROS commands. This can be done from the ROS workspace using the command :

```
source devel/setup.bash
```

More infomation on ROS commands can be found at wiki.ros.org

ROS Example

ROS requires a roscore to be run in order for nodes to communicate with each other. This can be run from your ROS workspace once the setup.bash file has been sourced using the following command:

```
roscore
```

With the roscore running, it is possible to run other executables in separate terminals. Further information on the scripts and how to run them can be found by opening the files in a text editor such as Gedit.

Example

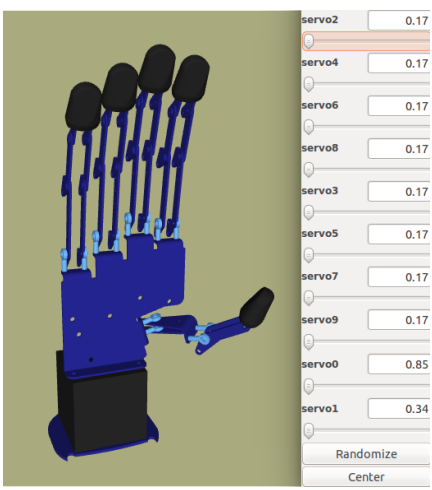
The scripts ar10_rviz_control_node.py uses commands from Rviz to send to the AR10 hand. It can be opened from the ROS workspace using the following command:

```
rosrun ar10 ar10_rviz_control_node.py
```

The Rviz interface can now be opened in a separate terminal from the ROS workspace:

```
roslaunch src/ar10_description/launch/display.launch
model:=src/ar10_description/urdf/ar10.urdf
```

The URDF model should now be open in the workstation. The joints of the model can be controlled using the joint_state_publisher GUI. Providing that ar10_rviz_controll_node.py is still open, the AR10 hand should also be controllable using the joint_state_publisher GUI.



The Hardware

DC-DC Step Down Regulator

The regulator powers the linear actuators and their microcontrollers. The DC-DC regulator will accept voltages from 7.5V to 30V DC, with a power supply of at least 1.5A. It is pre-set to output 6V DC to the servos with up to 3 amps, and does not need adjusting.

Powering the Pololu Chipset without USB

Should you wish to power the Maestro servo controller from the DC-DC step down regulator instead of USB then place a jumper lead across the pair of pins marked "VSRV=VIN" on the Maestro PCB (marked by an orange box on the Maestro pin-out diagram).

Servos

Each degree of freedom is actuated by a 6V linear actuator which incorporates a linear potentiometer and controller. This allows it to operate as a linear servo. The servos are specifically modified Firgelli PQ series linear actuators, each configured with a 100:1 gearbox.

They are not back-drivable. Care should be taken to adhere to the spec requirements of the motors when using the hand, specifically the 20% duty cycle of the actuators.
The actuator datasheet can be found on the Active Robots AR10 Humanoid Robotic Hand webpage.

The servos are connected to the Maestro board with a 4-wire ribbon cable . They operate as 3-wire servos, with the fourth wire providing feedback (see table).

They are not back-drivable. Care should be taken to adhere to the spec requirements of the motors when using the hand, specifically the 20% duty cycle of the actuators.
The actuator datasheet can be found on the Active Robots AR10 Humanoid Robotic Hand webpage.

The Hardware

Servos

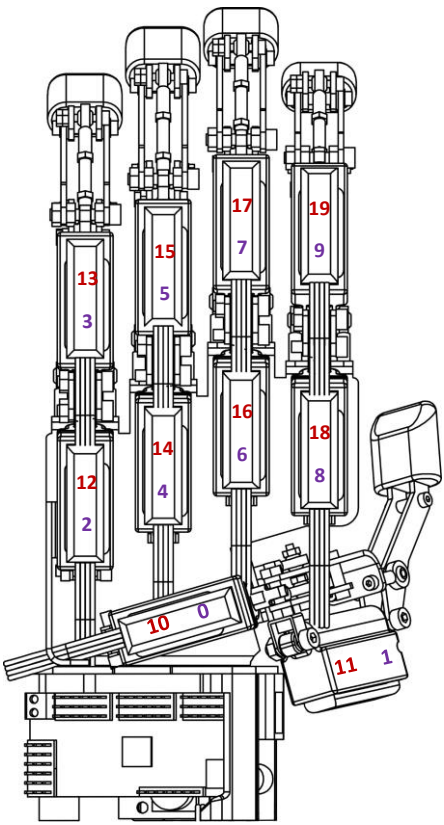
The servos are connected to the Maestro board with a 4-wire ribbon cable . They operate as 3-wire servos, with the fourth wire providing feedback (see table).

Servo Wire	Designation
Black	0 Volts DC
Red	+6 Volts DC
White	Servo Control
Purple	Potentiometer Feedback

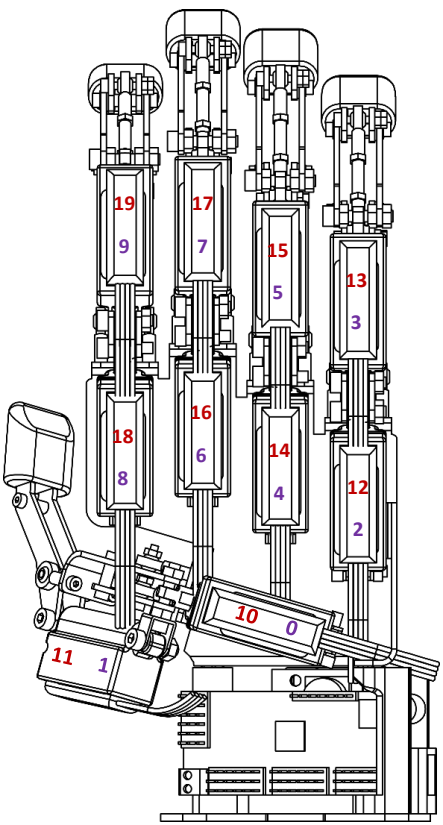
Each finger has two motors. The lower motor is marked on the wiring harness using a cable marker according to the diagram opposite.
The Maestro is configured with a settings file that designates the required channels as analogue inputs or servo outputs as shown in the diagram overleaf.

The Hardware

Control Channel	Servo Control & Feedback Channel
Red	Purple



Left Hand



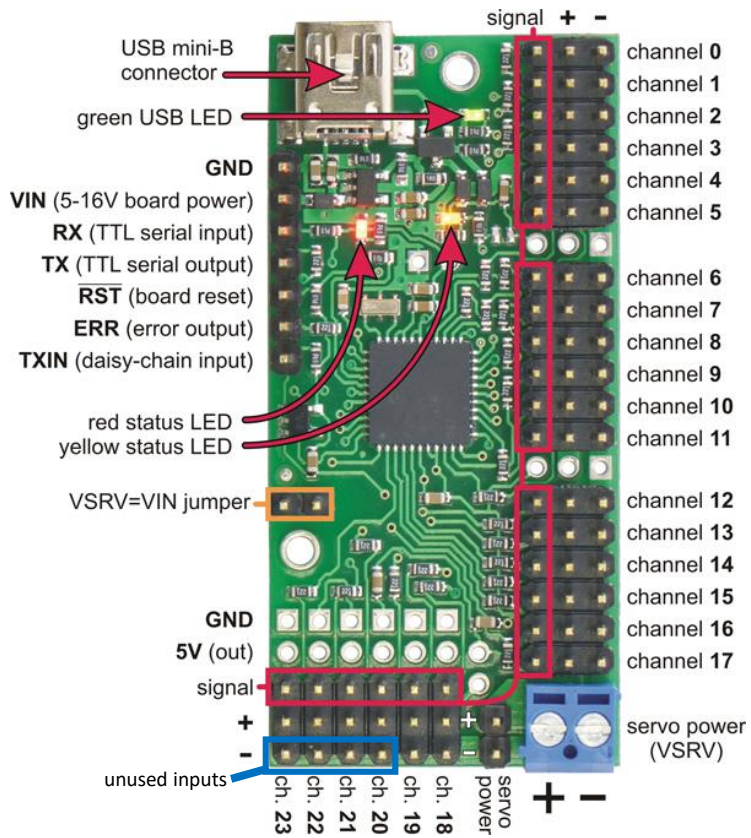
Right Hand

The Hardware

The Controller

The hand uses a Pololu Maestro Mini 24 channel servo controller to drive the actuators and receive feedback. There are 4 extra channels on the Pololu Maestro not being used to control the hand. Extra sensors or actuators can be added to these, such as force sensors, or EMG sensors to measure muscle activity and control the hand. See the assembly manual for a detailed view of the daughter board that is attached to the Pololu board.

Pololu Maestro Mini 24 Pin-Out



www.active8robots.com

Active Robots Ltd
Unit 10A, New Rock Industrial Estate
Chilcompton
Radstock
BA3 4JE
United Kingdom
Active Robots Technical Support:
Email: support@active-robots.com
Phone: 01761 234376

Registered: England & Wales
Company number: 4693628
VAT Reg: GB 810 7979 13

Document Version 2.4 November 2016
Active Robots Ltd

Made in the UK