

# IOCard USBStepper User manual.

**Author: Manuel Vélez**  
**Translation: Manuel Hernández-Peña**

[www.opencockpits.com](http://www.opencockpits.com)

**ver 1.0**

## **INTRO**

IOCard USBStepper has been designed to automatically handle with unipolar and bipolar stepper motors.

With this card we can easily control any stepper motor. This type of motors can be used to simulate and build analogue gauges.

This card uses an USB connector and the controller is managed using IOCP protocol.

## **SPECIFICATIONS**

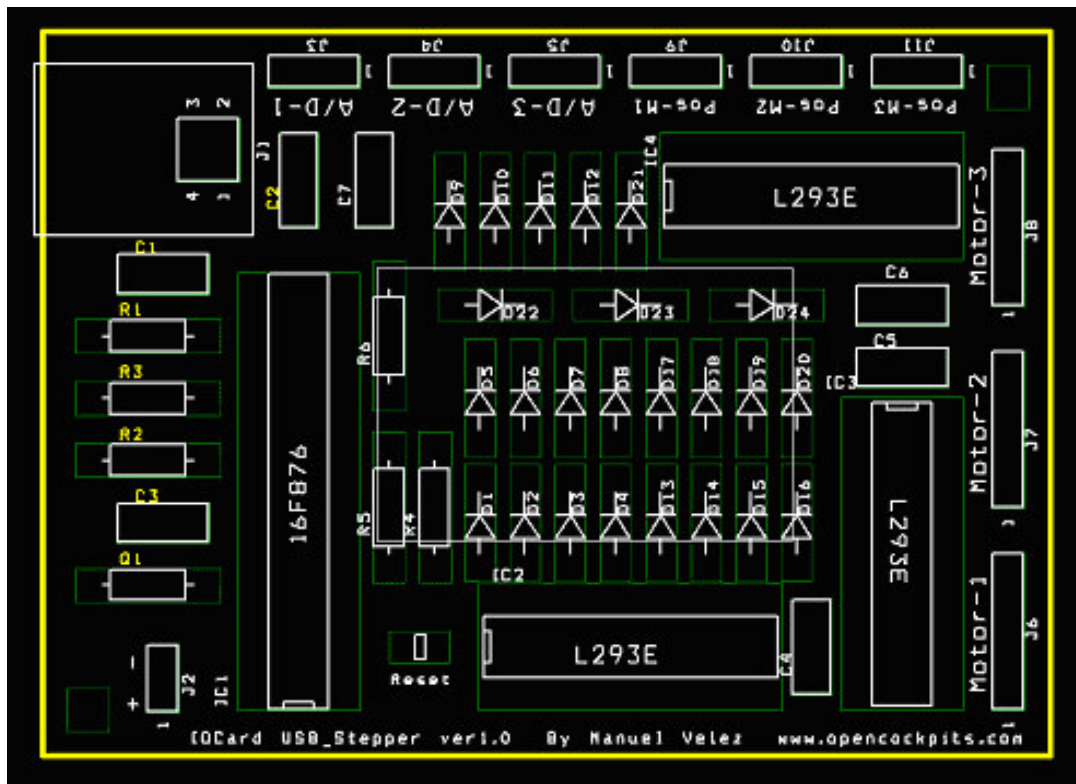
USBSteper:

- Has an USB connection
- Controls up to 3 stepper motors
- Controls unipolar and bipolar motors
- Has connectors for position sensors
- Has 3 analogue inputs (axles)
- Each motor can be fed with up to 36 vcc and 1 ampere.

## **COMPONENTS**

C1, C4,C5,C6,C7	= Condenser 0,1 mF
C2,C3	= Condenser 22Pf
D1 a D24	= Diodes 1N4007
IC1	= Microcontroller 16C745
IC2,IC3,IC4	= Integrated circuit L293E
J1	= USB connector
J2	= 2 pins connector (feeding)
J3,J4,J5,J9,J10,J11	= 3 pins connector
J6,J7,J8	= 5 pins connector
Q1	= Quartz oscillator 6 MHZ
R1	= Resistor 100R
R2,R4,R5,R6	= Resistor 10k
R3	= Resistor 1K5
SW1	= 2 pins connector (reset)

## CONNECTORS



- J1 = USB connector.
- J2 = Reset connector.
- J3, J4 y J5 = Potentiometers (analogue inputs).
- J6, J7, J8 = Stepper motors connectors.
- J9, J10, J11 = Position sensors connector.

## STEPPER MOTORS



Stepper motors are mainly used when precise movements are needed.

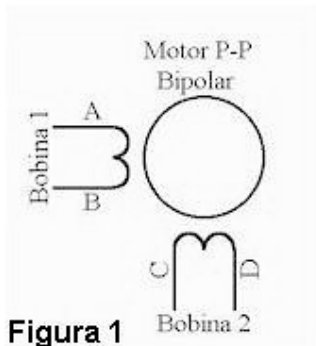
The main feature of these type of motors is the fact that we can move one step for each pulse we apply. We can find stepper motors with  $90^\circ$  per step and

others with  $1.8^\circ$  per step. In the first case, with 4 steps it makes a complete turn, and in the second we'll need 200 steps.

These motors consist in one rotor with different permanent magnets, and one stator with excitation coils.

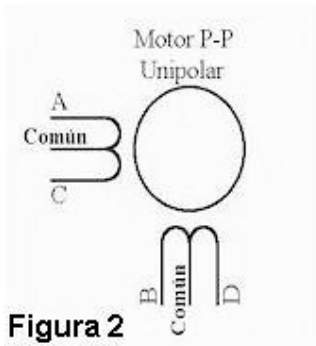
Depending on how the coils are built, there are two types of stepper motors:

**Bipolar:** we can find 4 different cables. The bipolar stepper scheme is:



**Figura 1**

**Unipolar:** we can find 5 or 6 cables, depending on the inner connections. The unipolar stepper scheme is:



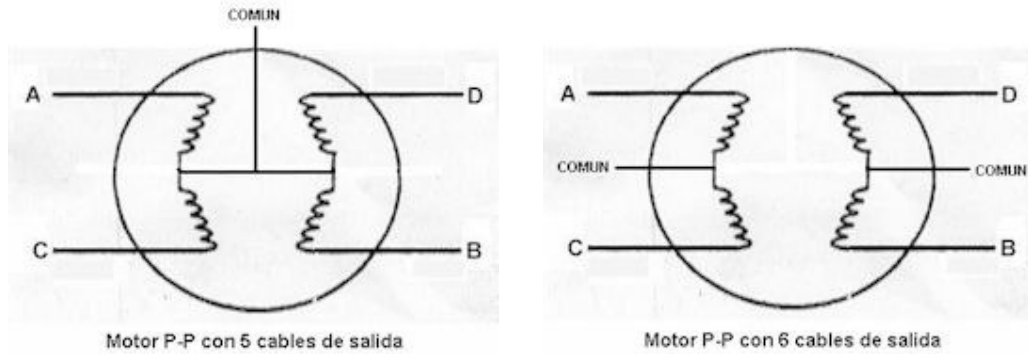
**Figura 2**

USBStepper card can manage both type of motors.

Due to the fact that stepper motors are mechanical devices that have to move loads (this implies torques), the length and frequency of the applied pulses is something to have into account. The motor has to reach its goal before another order is given to it. If the frequency is too high, the motor can react as follows:

- With no movement.
- Can start vibrating but without turning.
- Can turn with errors.
- Can turn in the opposite way.

When we are working with unipolar stepper motors, and we don't have any specification sheet, we can identify each cable as follows:



As shown in the pictures, in the case of 6 cables, two of them are “common”, so the first step is to put them together.

Now we will use a multimeter to check the resistance between pairs of cables. The resistance between the “common” cable and any other cable will be the half that will exist between non-common cables.

In the case of bipolar stepper motors, the cables identification is easier. Using a multimeter, we can locate the cables that connects each coil. We can find the polarity by testing: if the motor doesn’t work we’ll just change the connection in one of the coils. If the motor turns in the wrong direction we will change both connections.

### **Basically:**

- A motor with 5 cables is unipolar.
- A motor with 6 cables is unipolar, with 2 common cables with the same color.
- A motor with only 4 cables is bipolar.

## **POSITION SENSOR AND USBSTEPPER OPERATION**

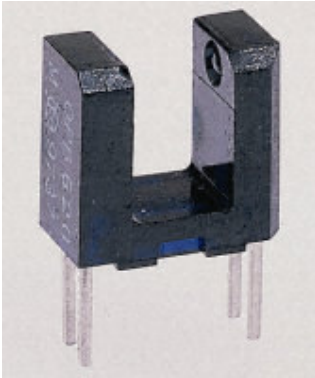
The optic position sensor is used to let the card know the axle position when the motor turns.

The first thing the card does is to turn the motor until the reference passes two times by the sensor. When the first time, a counter is set to zero and from this point, steps are counted until the reference passes by the sensor again. This way the card knows how many steps are in each 360° turn, and the time the motor gets to complete a turn.

Now the card can calculate any position based on the initial one by simply calculating the number of steps required.

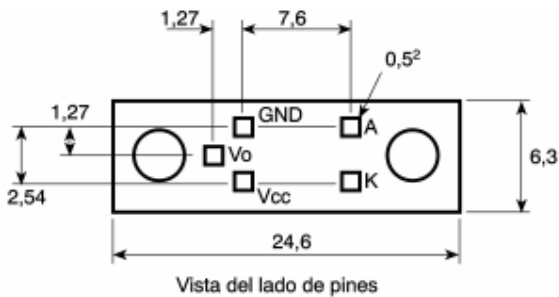
The optic sensors used must give TTL voltages (0 and +5 Volts).

We recommend those ones that include the electronic to give these TTL voltages.



The sensor consists of a light transmitter and a light receiver. When the light doesn't reach the receiver, the voltage changes from 0 to +5 volts.

The general scheme of these type of sensors is:



As shown in the picture, the sensor has 5 pins: 2 for the transmitting led and 3 for receiver feed and output signal.

In this case we should link GND pins in one side, and VCC in the other, installing a limiting resistor (330 or 470 ohms) for the led.

VO will change when something interrupts the light. We can use a needle that passes between the transmitter and the receiver.

The sensor is connected to the card by using different connectors: J9 (in the case of motor 1), J10 (in the case of motor 2), J11 (in the case of motor 3).

In these connectors, pin number 1 is GND, pin 2 is V0 and pin 3 is +5 volts (feeding).

In the picture, K is GND and A is +5Volts.

An example of these type of sensor can be found at [www.amidata.es](http://www.amidata.es) (or .com) reference 304-560.



## **CONNECTING MOTORS**

Each motor is connected to the card using the following connectors: J6 for motor 1, J7 for motor 2 and J8 for motor 3.

Depending on the type of motor, it will be connected in a different way:

**Unipolars:** pins 1, 2, 3 and 4 are connected to the non-common cables for coils 1 and 2. The common cable is connected to pin 5.

**Bipolars:** pins 1 and 3 are connected to cables A and B in coil 1, and pins 2 and 4 to cables C and D in coil 2.

Card is fed by J2 connector. This voltage can be from 3 to 36 volts, but will be **the same** for the 3 motors.

In the case of a high consumption we can limit it to avoid overheating; we can put a limiting resistor for each coil. These resistor must be of an adequate power.

If the motor is big and L293E integrated circuits overheat, we can install a refrigeration device on them.

## **SOFTWARE CONNECTION**

The link between card and simulator is made using IOCP.

Data must be modified before sending it to the card, so we will use SIOC. With a simple script, we will connect the card to SIOC, and then SIOC to the simulator (FSimulator or Xplane) by FSUIPC or IOCP.

If only SIOC is going to be used, is not necessary to activate IOCards module because this card works independently from the rest of IOCards.

The process works as shown in this diagram:



As we don't need to connect IOCards to SIOC we will modify SIOC.INI:

IOCard\_disable=Yes

In the same way, we will disable the protocol we are not using:

If we are not using the connection to simulator by IOCP, we will set:  
*IOCPclient\_disable=Yes*

If we are not using the connection to simulator by **FSUIPC**, we will set:  
*FSUipcdisable=Yes*

Card controller will start just after SIOC.

## **CARD CONFIGURATION**

Card is configured with the file *IOCStepper.ini*

This file has different options:

*MUSB=No* , we will set YES in case we have more than one card connected to the same PC. We'll have to set the device number too.

*deviceUSB=2048* , will inform the controller about the device number. If we don't know the device number, we can connect the card writing down the number shown by the controller. This number is different for each USB port.

*USB\_AD=0* , number of potentiometers we will connect to the card. 0 if we are not using them.

*IOCP\_host=localhost* , will set the IOCP server address where to connect.

*IOCP\_port=8092* , will set the IOCP server port where to connect.

The IOCP server (normally SIOC server) can be located in any PC in our network. So we can install our cards at any PC.

*Speed\_M1=5* , will set motor M1 delay (same for motors M2 and M3). The lower this values is, the higher the motor speed will be. You should not set a speed higher than the motor nominal speed.

*Steeps\_M1=0* , 0 if we want the card calculates the number of steps for a 360° turn. If we already know this value, we can set it here avoiding possible errors in the automatic calculation (same for motors M2 and M3).

*HSteeps\_M1=yes* , YES will enable the use of half steps. NO will force the use of complete steps (same for motors M2 and M3).

*Max\_Steeps\_M1=5* . This is an important value that indicates the maximum number of step orders in each 1/10 sec. If we set a high value, we can overrun the internal buffer in which the pending orders are stores, so we will lose information (position). If the value is too low, the motor speed will be low. A normal value for this parameter is 3-5. Si ponemos un número muy alto

podemos desbordar el buffer interno de la placa donde va acumulando las ordenes de pasos a completar y por tanto se perderá la posición, si ponemos un número excesivamente bajo bajará la velocidad de posicionamiento. Normalmente ente 3 y 5 son buenos valores (same for motors M2 and M3).

*Decimal\_Factor\_M1=10* , this is the factor that will divide the value given by IOCP server. If we set 1, range will be 0-359, if we set 10, range will be de 0-3590, and so on. For example, if we set 10 and IOCP give us 3435 this would mean set the motor position to 343,5° (same for motors M2 and M3).

*IOCP\_Ini\_Var=0* , first IOCP variable where the data is collected and sent. The variables map is as follows:

VAR = Data motor 1  
VAR+1= Data motor 2  
VAR+2= Data motor 3  
VAR+3= Data axle 1  
VAR+4= Data axle 2  
VAR+5= Data axle 3

If we set 10 as initial value, IOCP variables where the card will connect will be 10,11,12,13,14,15



When the software starts, it shows the connection address, if the module is connected to an IOCP server, if an USB card is located and running, and the device number of this card.

On the other hand, the number of steps by 360° turn are shown for each motor connected to the card.

## CONNECTING TO SIMULATOR

Connection to simulator is generally made by SIOC, so the first thing to do is defining the IOCP variable (or FSUIPC) offset where the data is available.

Then the data must be modified. At the end we will have a value between 0 and 360 to be sent to the IOCP variable.

The controller will indicate the shorter way to reach the position, so the direction of turn is automatically selected.

Normally, 0 degrees in the motor axle doesn't correspond with 0 degrees in the compass given by the IOCP variable (or FSUIPC offset). We can adjust the position using SIOC.

This is a SIOC example to manage a compass axle.

```
Var 0000 // Motor1

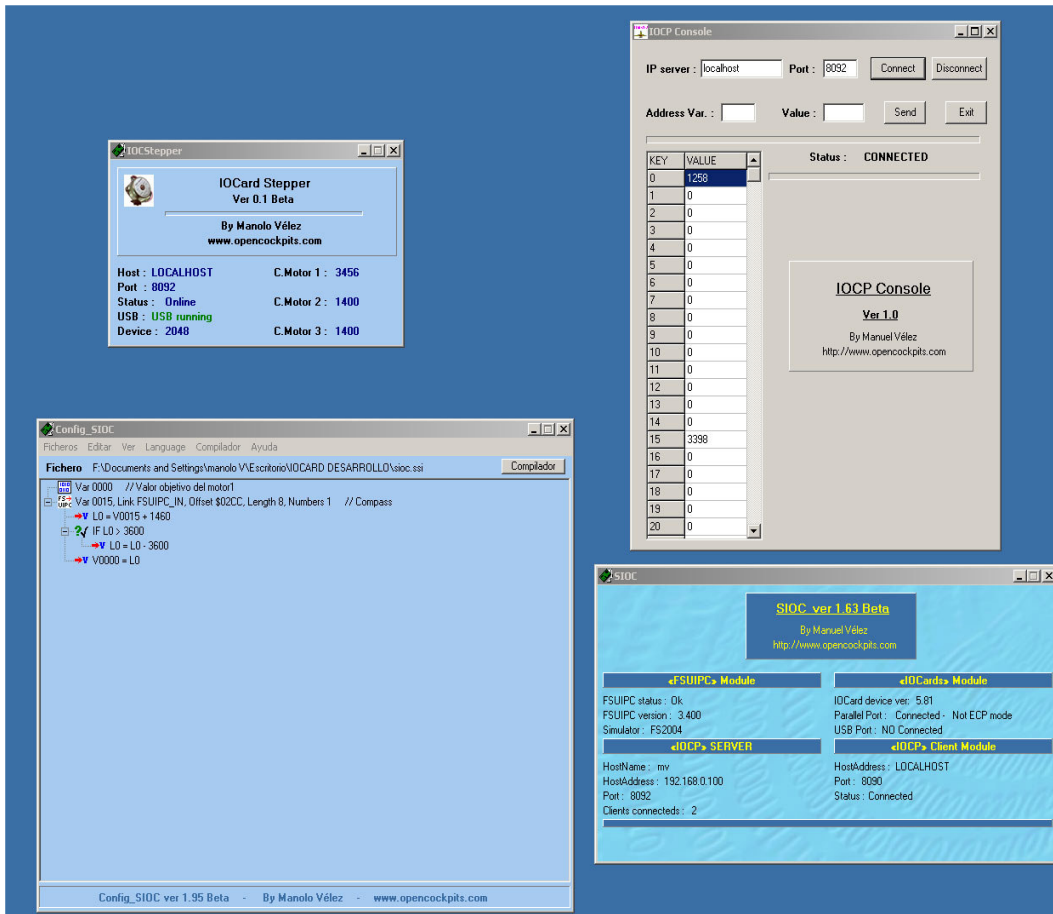
Var 0015, Link FSUIPC_IN, Offset $02CC, Length 8, Numbers 1 // Compass
{
  L0 = V0015 + 1460 // we add 146 degrees.
                    // in this position is where our compass shows North
  IF L0 > 3600 // We don't want to exceed 360°
  {
    L0 = L0 - 3600 // We correct the value in this case.
  }
  V0000 = L0 // We send the value to the motor controller.
}
```

For the controller, 0 and 360 is just the same. We don't want to have higher values than 360.

Offset \$02CC is the one that gives the compass position.

Once the SIOC program is saved, we start SIOC, then connect the card to any USB port and start IOCStepper.exe... The motor will start the selfcalibration process.

If the motor doesn't stop, the optic sensor is not working properly.



We can start IOCP console. We can set the motor position setting a value from 0 to 359 in the corresponding variable. We can also check what our program is sending in real time.

## TECHNICAL CONSIDERATIONS

The movement smoothness depends on the number of steps in the final axle. For example, a standard motor with low resolution has 48 steps per 360° (each step is 7,5 degrees).

We can use a gearbox. If we use a gearbox with a ratio 1:36, 36 complete turns of the motor will make the final axle turn 360° (the card is capable to move the motor with half steps, so everything is multiplied x2).

$48 \cdot 36 \cdot 2 = 3456$  half steps per 360° in the final axle. This means  $360^\circ / 3456 = 0,1^\circ$  per half step.

This number (3456) must be shown in the card controller, because this value has been automatically calculated during the self-calibrating process.

If we use the previous motor without the gearbox, we'll have a resolution of 7,5 degrees per step. In this case, if we say "go to 3°" and the motor is in the 0°, it won't move. If we say "go to 5°", it will go to 7,5°.

We have to choose a good resolution by using an adequate gearbox. By using a gearbox with a factor 4 or 5, and activating the half step, we will have a good enough resolution. The other way is to use motor with higher resolution (1,5° per step).

Another issue is the speed of turning. We are limited by the motor characteristics. If we turn the motor too fast, we will get errors. If we use a high ratio gearbox, the final axle will turn too slow for our requirements.