RC Servomotor and ultrasonic range detection expansion board

Features

- 5 independent outputs for RC servo control.
- 2 sockets for ultrasonic range detection module (Parallax Ping range sensor compatible)
- Wide range of servo operation: from 0.3 to 2.7 ms
- Generates 6V voltage to power RC servos using a 5V input power supply
- Multiplexed activation of each servo reduces the current peak, allowing powering from USB.
- DV0 bus compatible, easy managed by a DV0 header for a generic Linux system or DVO shield for Raspberry Pi board
- Easy to use library functions available in Java, C/C++ and Python

- Extended low position pulse time: 0.3ms
- Extended high position pulse time: 2.7ms
- Maximum current from 5V power input (with five RC servos connected): 600mA
- Power supply: 4.5 to 5.5 V
- Control bus: DV0 standard
- Board consumption (without Parallax Ping sensor connected): 22mA @5V
- Command latency from user application to IO_Servo servo outputs: 15ms

Technical specifications

- RC servo outputs period: 20ms
- Nominal low position pulse time: 1ms
- Nominal center position pulse time: 1.5ms
- Nominal high position pulse time: 2ms

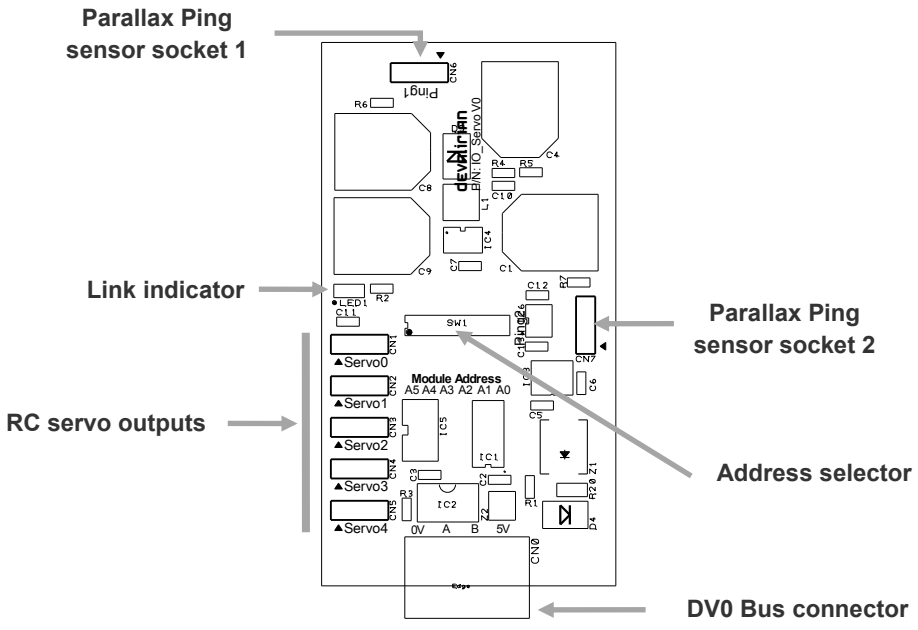


Fig. 1 Simplified connectors diagram

Table of contents

Revision history:

V1.0 Jan-2015

1. FUNCTIONAL OVERWIEW

The IO_Servo board continuously listen to the DV0 bus serial lines for incoming commands from the Header. This commands can be either a setting for an output, or an enquiring for the actual value of an input or a request for the IO_Servo manifest. Commands from the header starts with the address of the target and every board in the bus compares the command address with the actual value of the address selector.

CAUTION:

> Care must be taken to ensure that there are no duplicate address board in the bus, because the response collision will hung the boards involved and an unpredictable behavior of the rest of the boards is very likely.

At power up, the IO_Servo board blinks the Link Indicator, reporting that it is powered but not yet addressed by the Header. When the first command that mach its address is received, the Blink Indicator remains continuously lighted.

1.1 Address selection

Address is selected by setting the individual switches of the Address Selector, at it is shown in the Figure 2.

Fig. 2 Address Selector

The switches involved in the address selection are those labeled as A0 to A5, coded as a binary number where A0 is the least significant bit and A5 the most significant bit. The switch meaning is “1” if it is “ON” and “0” otherwise. Factory values are all “OFF”, that means “Normal Analog Input” and “Address = 0”. Keep in mind that zero is not a valid address value and will be ignored.

The binary switches codification is illustrated in the Figure 3.

1.2 DV0 Bus connection

This receptacle connector contains receives power and data from the Header. An extracting vertical socket with screws is supplied with the IO_Servo board to facilitate bus cabling (AWG 16..24, cross section 1.5 to 0.2 mm)

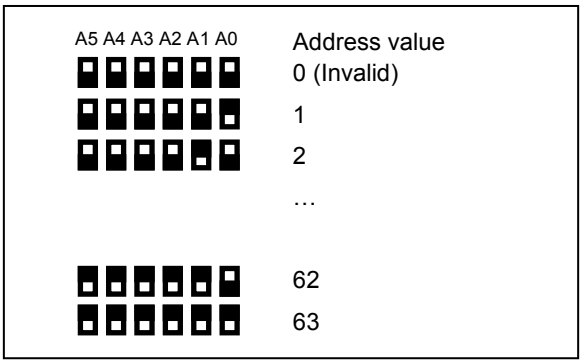


Fig. 3 Address coding example

Vertical socket technical data:

- Socket reference: 20020008-D041B01LF from FCI.
- Solid/Stranded wire: AWG 16 to AWG 26
- Wire cross section: 1.5mm to 0.2mm
- Data signal A and B ESD, EFT and Surge protection: IEC 61000-4-2 (ESD), IEC 61000-4-4 (EFT) and IEC 61000-4-5 (Surge)

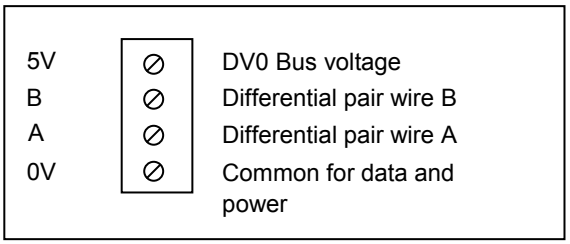


Fig. 4 DVO Bus connector pin out top view

Connect differential pair A to all A inputs of connected IO boards, and pair B to all B inputs as well. Also, connect the common data pin (0V) to all IO boards. The output marked as 5V is the DV0 bus voltage output.

1.3 Manifest

The manifest of IO_Servo, as it is seen by the API function *GetManifest* is:

```
Board name     : IO_Servo
Registers
PWM Outputs    : 5
```

2. ELECTRICAL CHARACTERISTICS

This chapter explains the electrical characteristics of each IO_Servo input and output.

2.1 Absolute maximum ratings

Absolute maximum ratings for the IO_Servo board are listed below. Exposure to these maximum rating conditions for extended periods may affect device reliability. Functional operation of the device at these, or any other conditions above the parameters indicated in the operation listings of this specification, is not assured.

Operating ambient temperature……....-0°C to +70°C
Storage ambient temperature .....-55°C to +125°C
Storage ambient humidity……...35% to 85%
Operating ambient humidity……...35% to 85%
Voltage at 5V DV0 connector……..5.5V

2.2 RC Servo outputs

Figure 5 shows the pinout for the RC servo connectors 0 to 4, market at the board as Servo0 to Servo4. Note the triangle-shaped mark that indicates the negative (-) pin.

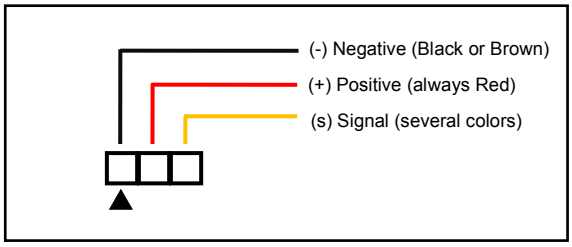


Fig. 5 Standard RC Servo connector pinout

Figure 6 shows the most popular RC servo brand cabling colors. Note that the positive cable is always red, and the negative one is always black with the unique exception of JR model. Also, their relative position is the same used by the IO_Servo board. Only some old models placed the negative at the center

CAUTION:

RC servos are not protected against wrong polarity connection. Be careful when connecting the RC servo to the IO_Servo plugs and place the socket so that the cable black (or brown for JR servos) is in the pin with the triangle-shaped mark.

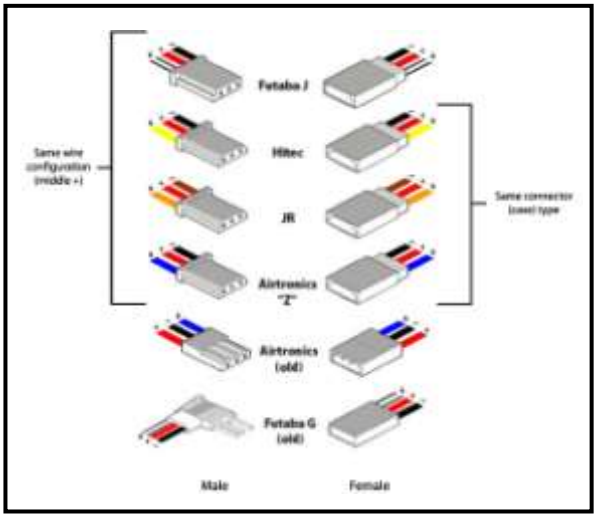


Fig. 6 Most common RC servo pinouts

2.2.1 API functions

The following table summarizes the API functions related to the RC servo outputs. See 3.1 for more detailed information.

Function name	Description
`SetPWMValue`	Set the output pulse with in steps of 0.01 percent. The parameter *Value* can be in the range of 1 (minimum pulse with) to 10000 (100%, maximum pulse with) The center position of servo is obtained giving 5000 for *Value*
`SetPWMLimits`	Parameter *Period (always 20000*, *TimeForZero (from 300 to 1000)* and *TimeFor100 (from 2000 to 2700).* *This function is useful for RC servos that allow extended command limits (from 0.3 ms up to 2.7ms)* Default values are 20000, 1000 and 2000, that generates the standard RC pulse with of 1.5ms at center, 1ms at minimum position and 2ms at maximum position in a period of 20ms

2.3 Parallax Ping sensors

The IO_Servo board supplies two plugs for connecting two Parallax Pig range sensor. Figure 6 shows the connection required for the Parallax sensor (GND, 5V and Signal)

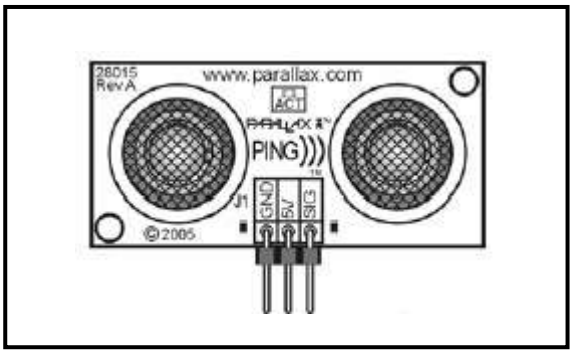


Fig. 6 Parallax range sensor

Figure 7 shows the pinout of the two connectors for Parallax sensor that provides the IO_Servo board. They are marked as Ping1 and Ping2. Note that the triangle shaped mark of this connectors signals the GND ping. **Be careful: Parallax sensor are not protected against polarity inversion.**

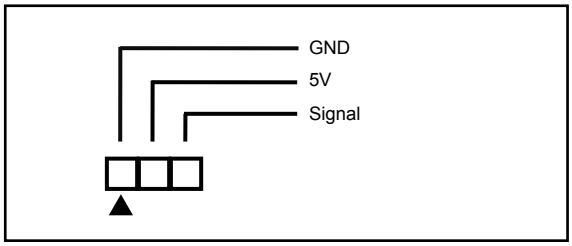


Fig. 7 Parallax pinout connector at IO_Servo board

2.3.1 API Functions

User application control these two sensors using Register functions. Writing in the general purpose register 1 (GP1) the user application controls how the Parallax is polled. If a 1 is written into GP1, the IO_Servo boards tells the Parallax modules to generate one single ping, while writing a 2 on this registers force the Parallax to measure continuously (well, every 40ms).

Reading GP1 the user application can obtain the last measure yield by the Parallax sensor connected on the connector Ping1, while reading GP2 gives the same measure from the Parallax connected on connector Ping2.

The following table summarizes the API functions related to the Parallax connector. Note that the functions allows up to 6 general purpose register, but only GP1 and GP2 are used by IO_Servo.

Function name	Description
SetRegister	Set the value of 6 general purpose register. Each register is 16bits long. Only GP1 is used to set de polling mode: 1 implies one-shot and 2 implies continuously polling. Ping generation is interleaved in order to avoid interferences between the two modules.
GetRegister	Get the value of 6 general purpose register. Each register is 16bits long. GP1 and GP2 returns the detected range in millimeters. 0 means that a ping is in progress Minimum polling time: 40ms

Caution: if no sensor is connected, the result of GP1 and GP2 are unpredictable.

3. SOFTWARE

3.1 *DV0* API functions related to IO_Servo

The DV0 API contains a set of functions intended to manage the IO_Servo board that can be found for Java, C++ and Python languages.

Some examples of those functions have been illustrated in the sections 2. This section is intended as a formal declaration of parameters and return values

WARNING:

This data sheet summarizes the features of DV0 API functions related to IO_Servo board. It is not intended to be a comprehensive reference source. For more information, refer to the DV0 API Reference Guide for C++, Java and Python at www.devalirian.com, in the Technical Information section

The steps for managing the IO_Servo control board programmatically are:

- Create an instance of the class DV0
- Call the function Open
- If successful, call IO_Servo functions related.

In the following sections, an example of each language will be given for creating instance, for calling Open function and for calling related functions. Also, more complete examples can be found at the end of this chapter.

Table 5 summarize the functions related to IO_Servo. Note that only the name of the function is listed in the table below. This is because the parameters and return values are slightly different among the three implementation (Java, C++ and Python).

Table 1. IO_Servo related functions

Name	Description
Open	Initial and mandatory function to connect with dv0_manager
SetPWMValue	Set the duty cycle of a PWM output
SetPWMLimits	Set the PWM period, as well as minimum an maximum pulse value
SetRegisters	Sets register GP1 for starting measure of Parallax sensor
GetRegisters	Reads GP1 and GP2 from the board in order to get the measured range

3.1.1 Creating an instance of DV0 class

Java, C++ and Python are all object oriented languages and it is useful to encapsulate all the accessing to the API throughout one object. All subsequent managing functions will pass through this variable. Let's assume that this variable will be called *dv0*. For C++ or Java, the creation of this variable is thoroughly:

```
DV0 dv0; // C++ or Java
```

And so is for Python, but slightly different (actually, It is not a class)

```
import dv0
```

3.1.2 Connecting to *dv0_manager*

The connection to *dv0_manager* is made through the function Open.

Function Open generic description

```
int Open(IpAddress, Port, Login, Password);
```

Open connection with the dv0_manager. Mandatory for all other members. Blocking.

Parameters:

IpAddress is the IP where dv0_manager listens to incoming connections. By default is "localhost" that suits perfectly is this application runs in the same machine that dv0_manager does. If a remote connection is desired, call dv0_manager with the argument -ip_address followed by the IP address of the internet interface of your board (use ifconfig to know it) and pass this value to the current IpAddress parameter.
Port is the UDP port where dv0_manager listens to incoming connections. By default is 6900 but DV0_manager can listen to whatever port selected by command line argument -port. In this case, give the same value to current Port parameter. Useful only if another application had catch this port.
Login and **Password** are required if the dv0_manager daemon is called with -login <username> argument. Current Login parameter must match with <username> and a valid Password must be entered

Return values

Returns 0	if connection is successful
DV_NOT_CONNECTION	if dv0_manager does not respond. Check IpAddress and Port
DV_INVALID_LOGIN	if dv0_manager refuses login or password

C++ Syntax

```
int Open(char *IpAddress = NULL, int Port = 0, char *Login = NULL, char *Password = NULL);
```

Java Syntax

```
int Open(String IpAddress, int Port, String Login, String Password);
```

Python Syntax

```
def Open(ipAddress, port, login, password):
```

Example: Connect to a *dv0_manager* running in the same Raspberry that the current program, using de default port and with no login required. The *dv0_manager* call can be:

```
/home/pi/dv0/dv0_manager -force_date
```

From a C++ user program, then:

```
DV0 dv0;
…
int r = dv0.Open(NULL, 0, NULL, NULL)
if (r == 0) {
…
} else DisplayError(r);
```

From a Java user program, then:

```
DV0 dv0;
…
int r = dv0.Open(“”, 0, “”, “”)
if (r == 0) {
…
} else DisplayError(r);
```

From a Python program, there are no so much differences:

```
import dv0
res = dv0.Open('',0,'','')
if (res == 0):
    print ("Connected to dv0_manager")
```

```
else:
    DisplayError(res)
```

Example: Connect to a *dv0_manager* running in a Raspberry connected to internet through the interface 10.0.0.2, and the current program running somewhere, login name 'pi' required (with default password 'raspberry') , using de default port. The *dv0_manager* call can be:

```
/home/pi/dv0/dv0_manager –ip_address 10.0.0.2 –force_date –login pi
```

From a C++ and Java user program, then

```
DV0 dv0;
…
int r = dv0.Open(”10.0.0.2”, 0, ”pi”, ”raspberry”)
if (r == 0) {
…
} else DisplayError(r);
```

From a Python program, there are no so much differences:

```
import dv0
res = dv0.Open('10.0.0.2',0,'pi','raspberry')
if (res == 0):
    print ("Connected to dv0_manager")
else:
    DisplayError(res)
```

3.1.3 Summary of API functions related to IO_Servo

3.1.3.1 Functions for RC servo managing

Function *SetPWMValue* generic description

```
int SetPWMValue(int Board, int Output, int Value);
```

This function controls the position of the servo connected at this Output

Parameters:

```
Pre-conditions : "Board" matches with address micro switches in board
Output from 0 to 4
Value from 1 (minimum pulse with, 0 degrees position) to 10000 (100%, maximum pulse with, 180
degrees position) Default value is 5000 (neutral servo position or 90 degrees)
```

Return codes

```
Returns 0 if successful or
DV_NOT_CONNECTED      if previous Open call had failed or connection has been canceled
DV_INVALID_BOARD      if that Board doesn't exist or it is not responding
DV_NOT_SUPPORTED      if there is not such output in that board
```

C++ Syntax

```
int SetPWMValue(int Board, int Output, int Value);
```

Java Syntax

```
int SetPWMValue(int Board, int Output, int Value);
```

Python Syntax

```
def SetPWMValue(Board, Output, Value):
```

Function *SetPWMLimits* description

```
int SetPWMLimits(int Board, int Output, int Period, int TimeForZero, int TimeFor100);
```

```
Define the limits and behavior of RC PWM
For example, typical RC Model servo expects a 20ms period pulse with a minimum value of 1ms
(servo to left) and a maximum of 2ms (servo to right) which means:
    SetPWMValue(board, output, 20000, 1000, 2000)
and to center the servo, just order a 50% setting:
    SetPWMValue(board, output, 5000) or whatever other value in between
Default values are Period = 20000, TimeForZero = 1000 and TimeFor100 = 2000, which fits the
standard limits for RC servos.
However, there are servos that accept extended limits (bellow 0 and behind 180). This
function extends the range of SetPWMValue
Changing limits force the PWM value to 5000 so that the servo goes to its neutral position.
```

Parameters:

```
Pre-conditions : "Board" matches with address micro switches in board
Output from 0 to 5 (connectors Servo0 to Servo4)
TimeForZero and TimeFor100 in micro seconds (from 300 to 2700 µs)

Example for extended RC servos (0.3ms to 2.7ms, neutral always at 1ms):
SetPWMLimits(Board, Output, 20000, 300, 2700)
```

Return codes

```
Returns 0 if successful or
DV_NOT_CONNECTED      if previous Open call had failed or connection has been canceled
DV_INVALID_BOARD      if that Board doesn't exist or it is not responding
DV_NOT_SUPPORTED      if there is not such output in that board
```

C++ Syntax

```
int SetPWMLimits(int Board, int Output, int Period, int TimeForZero, int TimeFor100);
```

Java Syntax

```
int SetPWMLimits(int Board, int Output, int Period, int TimeForZero, int TimeFor100);
```

Python Syntax

```
def SetPWMLimits(Board, Output, Period, TimeForZero, TimeFor100):
```

3.1.3.2 Functions for Parallax sensor managing

Function *SetRegisters* description

```
int SetRegisters(int Board, int GP1, int GP2, int GP3, int GP4, int GP5, int GP6);
```

Set the value of 6 general purpose register. Each register is 16bits long.
Only GP1 is used to set de polling mode: 1 implies one-shot and 2 implies continuously polling (for both modules)
Ping generation is interleaved in order to avoid interferences between the two modules

Parameters:

Pre-conditions : "Board" matches with address micro switches in board

Return codes

Returns 0 if successful or
DV_NOT_CONNECTED if previous Open call had failed or connection has been canceled
DV_INVALID_BOARD if that Board doesn't exist or it is not responding
DV_NOT_SUPPORTED if there is not such output in that board

C++ Syntax

```
int SetRegisters(int Board, int GP1, int GP2, int GP3, int GP4, int GP5, int GP6);
```

Java Syntax

```
int SetRegisters(int Board, int GP1, int GP2, int GP3, int GP4, int GP5, int GP6);
```

Python Syntax

```
def SetRegisters( Board,  GP1,  GP2,  GP3,  GP4,  GP5,  GP6):
```

Function *SetPWMLimits* description

```
int GetRegisters(int Board, int *GP1, int *GP2, int *GP3, int *GP4, int *GP5, int *GP6);
```

Get the last range measured by the module on the connector Ping1 (in GP1) and the module on the connector Ping2 (in GP2).
If their value is zero means that a measure is not finished.
Returned value is expressed in millimeters
A new measure is available every 40ms.

Parameters:

Pre-conditions : "Board" matches with address micro switches in board

Return codes

Returns 0 if successful or
DV_NOT_CONNECTED if previous Open call had failed or connection has been canceled
DV_INVALID_BOARD if that Board doesn't exist or it is not responding
DV_NOT_SUPPORTED if there is not such output in that board

C++ Syntax

```
int GetRegisters(int Board, int *GP1, int *GP2, int *GP3, int *GP4, int *GP5, int *GP6);
```

Java Syntax

```
int SetRegisters(int Board, int GP[]);
```

Python Syntax

```
def GetRegisters( Board ):
# Returns a 2-elements tuple containing an array of six integers an integer (return value)
return [0,0,0,0,0,0], Result
```

3.1.4 API examples

In this section, a complete example of how to manage Header1 related functions are described both in C++ and Python languages. Java example is omitted because it is so close to C++ that only adds confusion. However, there is an example of how to program an Android APP with the DV0 API, that is not included in this section for the benefit of simplicity but it can be found at www.devalirian.com, inside the Technical Information section.

3.1.4.1. C++ ExampleServo.cpp

```
#include <iostream>
#include <stdio.h>
using namespace std;
#include "DV0.h"

char *ServAddress       = NULL;
char *LoginName = NULL;
char *Password          = NULL;
int  Port = 0;
DV0  dv0;

void DisplayError(int r);
void DoTest(void);
int  input (char *text);

int main() {
        int r;
        // Trying to connect
        r = dv0.Open(ServAddress, Port, LoginName, Password);
        if (r == 0) {
                cout << "Connection to dv0_manager successful " << endl;
                DoTest();
        } else DisplayError(r);
        return r;
}

void DoTest(void) {
        int board, mode, output, value, res, option, period, timeForZero, timeFor100;
        int Input, v1, v2, foo;
        // Print menu
        cout << "1-Set RC Output" << endl;
        cout << "2-Set RC Limits" << endl;
        cout << "3-Set Parallax mode" << endl;
        cout << "4-Get Parallax measure" << endl;
        cout << "5-Get Manifest" << endl;
        cout << "0-Exit" << endl;
        // Get user option
        cin >> option;
        // And execute
        switch (option) {
                case 1:
                        // ---------------------------------------- Set PWM value example
                        board = input ("Enter board address:");
                        output= input ("Enter output:");
                        value = input ("Enter servo position (0 to 10000) :");
                        res = dv0.SetPWMValue(board, output, value);
                        DisplayError(res);
                        break;
                case 2:
                        // ---------------------------------------- Set PWM limits example
                        board = input ("Enter board address:");
                        output= input ("Enter output:");
                        period= 20000;
                        timeForZero= input ("Enter time for zero(us):");
                        timeFor100 = input ("Enter time for 100%(us):");
                        res = dv0.SetPWMLimits(board, output, period, timeForZero, timeFor100);
                        DisplayError(res);
                        break;
                case 3:
                        // ---------------------------------------- Set registers
                        board = input ("Enter board address:");
                        mode = input ("Enter scan mode (1 single, 2 continuously :");
                        res = dv0.SetRegisters(board, mode, 0, 0, 0, 0, 0);
```

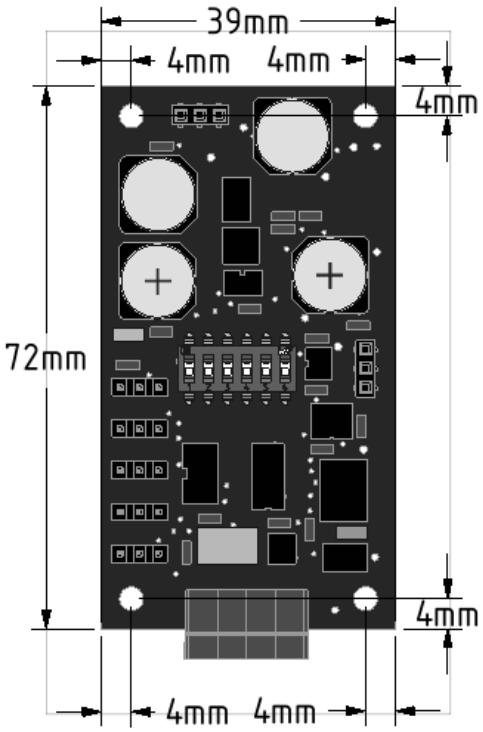
```
                        DisplayError(res);
                        break;
                case 4:
                        // ---------------------------------------- Get registers
                        board = input ("Enter board address:");
                        res = dv0.GetRegisters(board, group, &v1, &v2, &foo, &foo, &foo, &foo);
                        if (res == DV0_OK)
                                cout << "Distance Ping1: " << v1 << " mm" <<  endl;
                                cout << "Distance Ping2: " << v2 << " mm" <<  endl;
                        else
                                DisplayError(res);
                        break;
                case 5:
                        // ---------------------------------------- Get Manifest example
                        board = input ("Enter board address:");
                        char description [756];
                        res = dv0.GetBoardManifest(board, description);
                        if (res == DV0_OK)
                                cout << "Value: " << description << endl;
                        else
                                DisplayError(res);
                        break;
        }
}

int input (char *text) { int option;
        cout << text;
        cin >> option;
        return option;
}

void DisplayError(int error) {
        switch (error) {
                case DV_NOT_CONNECTION:
                        cout << "dv0_manager does not respond. Check IpAddress and Port" << endl; break;
                case DV_INVALID_LOGIN:  cout << "dv0_manager refuses login or password" << endl; break;
                case DV_NOT_CONNECTED:  cout << "DVO header board is not connected" << endl; break;
                case DV_INVALID_BOARD:  cout << "Invalid board address" << endl; break;
                case DV_NOT_SUPPORTED:  cout << "Invalid peripheral" << endl; break;
                case DV0_OK: break;
                default: cout << "Unexpected error code" << endl; break;
        }
}
```

3.1.4.2. Python ExampleServo.py

```
import sys
import dv0

def DisplayError(error):
    if error == 0:
        return
    if error == dv0.DV_NOT_CONNECTION:
        print ("dv0_manager does not respond. Check IpAddress and Port")
    elif error == dv0.DV_INVALID_LOGIN:
        print ("dv0_manager refuses login or password")
    elif error == dv0.DV_NOT_CONNECTED:
        print ("DVO header board is not connected")
    elif error == dv0.DV_INVALID_BOARD:
        print ("Invalid board address")
    elif error == dv0.DV_NOT_SUPPORTED:
        print ("Invalid peripheral")
    else:
        print ("Unexpected error code: ", error)

# Assuming that the Raspberry is not this computer and connected as 10.0.0.2
# Also, dv0_manager is called like "dv0_manager -ip_address 10.0.0.2 -login pi"
res = dv0.Open('10.0.0.2', 6900, 'pi', 'raspberry')
# Assuming that the Raspberry is this computer
# and dv0_manager is called just like "dv0_manager"
# res = dv0.Open('',0,'','');

looping = True
if (res == 0):
    print ("Connected to dv0_manager")
else:
    DisplayError(res)
    looping = False
while looping:
    print ("1-Set RC Output")
    print ("2-Set RC Limits")
    print ("3-Set Parallax mode")
    print ("4-Get Parallax measure")
    print ("5-Get Manifest")
    print ("0-Exit")
    option = int(input("Enter option:"))
    if option == 1:
        # ----------------------------------------- Set PWM value example
        board = int(input ("Enter board address:"))
        output= int(input ("Enter servo output:"))
        value = int(input ("Enter servo position (1 to 10000) :"))
        res = dv0.SetPWMValue(board, output, value)
        DisplayError(res)
    elif option == 2:
        # ----------------------------------------- Set PWM limits example
        board = int(input ("Enter board address:"))
        output= int(input ("Enter output:"))
        period= 20000
        timeForZero= int(input ("Enter time for zero(us):"))
        timeFor100 = int(input ("Enter time for 100%(us):"))
        res = dv0.SetPWMLimits(board, output, period, timeForZero, timeFor100)
        DisplayError(res)
```

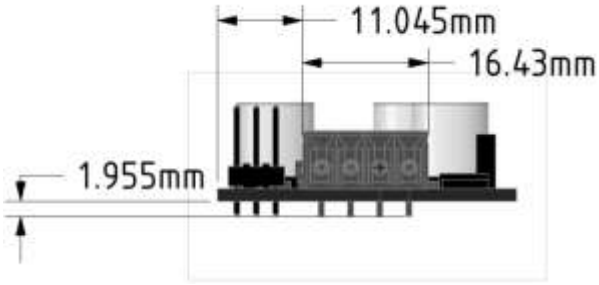
```
elif option == 3:
    # ----------------------------------------- Set Registers
    board = int(input ("Enter board address:"))
    mode  = int(input ("Enter scan mode (1 single, 2 countinuously):"))
    value, res = dv0.SetRegisters(board, mode, 0, 0, 0 ,0, 0)
    DisplayError(res)
elif option == 4:
    # ----------------------------------------- Get Registers
    board = int(input ("Enter board address:"))
    value, res = dv0.GetGroupInput(board, group)
    if (res == dv0.DV0_OK):
        print ("Distance Ping1: %d mm" % value[0])
        print ("Distance Ping2: %d mm" % value[1])
    else:
        DisplayError(res)
elif option == 5:
    # ----------------------------------------- Get Manifest example
    board = int(input ("Enter board address:"))
    description, res = dv0.GetBoardManifest(board);
    if (res == dv0.DV0_OK):
        print (description)
    else:
        DisplayError(res)
elif option == 0:
    looping = False
```

4. MECHANICAL DRAWINGS



Board dimensions and hole positions (top view)

Board dimensions (left view)



Board dimensions (front view)

5. ACCESSORIES

The IO_Servo board comes with the following accessories:

- One DV0 socket. Model 20020004-D041B01LF

A DIN RAIL adapter is available at www.devalirian.com

IMPORTANT NOTICE

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

deValirian MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE**.**

deValirian disclaims all liability arising from this information and its use. Use of ***deValirian*** devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless ***deValirian*** from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any ***deValirian*** intellectual property rights.

Header1 board is not designed to be radiation tolerant

Please be sure to implement in your equipment using the safety measures to guard against the possibility of physical injury, file or any other damaged cause in event of the failure of IO_Servo board. deValirian shall bear no responsibility whatsoever for you're your use of IO_Servo board outside the prescribed scope or not in accordance with this manual.

Reproduction of significant portions of ***deValirian*** information in ***deValirian*** data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. ***deValirian*** is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Web Site: www.devalirian.com

Mail info: info@devalirian.com