MG400 QUICKSTART GUIDE

By In-Position Technologies

Contents

Getting Started 3
Hardware Required 3
Downloading the Software 4
Connecting to the MG400 5
Configuring Your Hardware 5
Configuring Your IP-Address 7
Connecting to the MG400 with DobotStudio2020 11
Enabling the Robot 12
Using the Unlock Button to move the MG400 13
Using the Jog Panel to Move the MG400 14
MG400 IO 17
MG400 IO Wiring Example 18
Programming the MG400 19
Basics 19
Saving Points 20
Different Move Types 21
Blockly 22
Porting Blockly Program to Script 25
Blockly Example 1: Basic Moves 27
Blockly Example 2: Using IO #1 28
Blockly Example 3: Modified Parameters in Move 29
Script 30
Script Example 1: Basic Moves 33
Script Example 2: Basic IO Operations 33
Script Example 3: Custom Function 34
Script Example 4: TCP/IP Client 35
Script Example 5: Variable palletizing 36
Troubleshooting & Documents 40
Dobot Connectivity Guide for MG400 40
Hardware User Guide 40
DobotStudio2020 Dobot User Guide 40
MG400 Spec Sheet 40
Clearing MG400 Errors 40
LED Functionality 41
Using Sync with IO Commands 41
Contacts 42
Phone 42
Email 42

Getting Started

Hardware Required

To operate the robot, you'll need:

- MG400 Robot

- MG400 power adapter

- MG400 E-Stop

- Ethernet cable (included cable preferred)

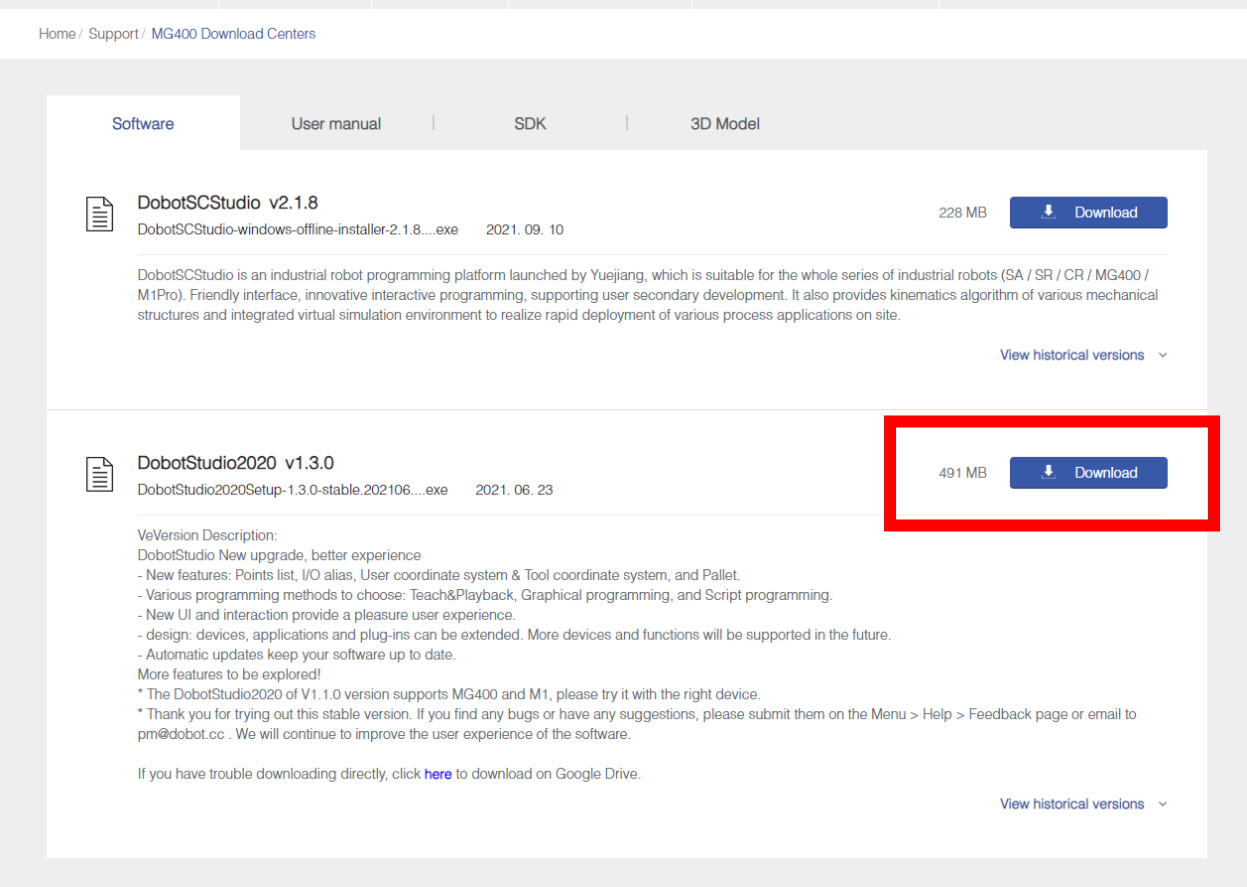
- A PC running windows

- If your PC doesn't have an Ethernet port, you'll need a USB to Ethernet adapter

See Connecting to the MG400 for details on cable connections.

Downloading the Software

You'll need the MG400 software before getting connected. Download it off the [Dobot official manufacturer's] site or contact your distributor for the most recent version.
The software you need is called "DobotStudio2020"

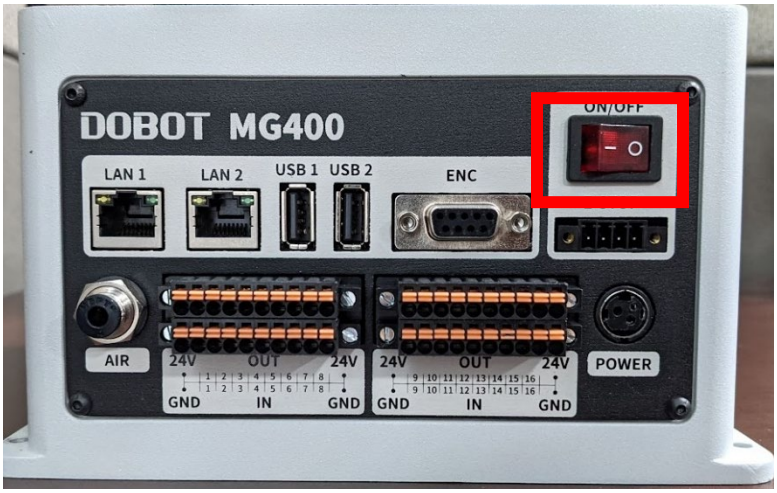


While DobotSCStudio functions with the MG400, we recommend using Dobotstudio2020.

Connecting to the MG400

Configuring Your Hardware

Before making any connections, be sure your MG400 power switch is set to “off” (O), like shown here:

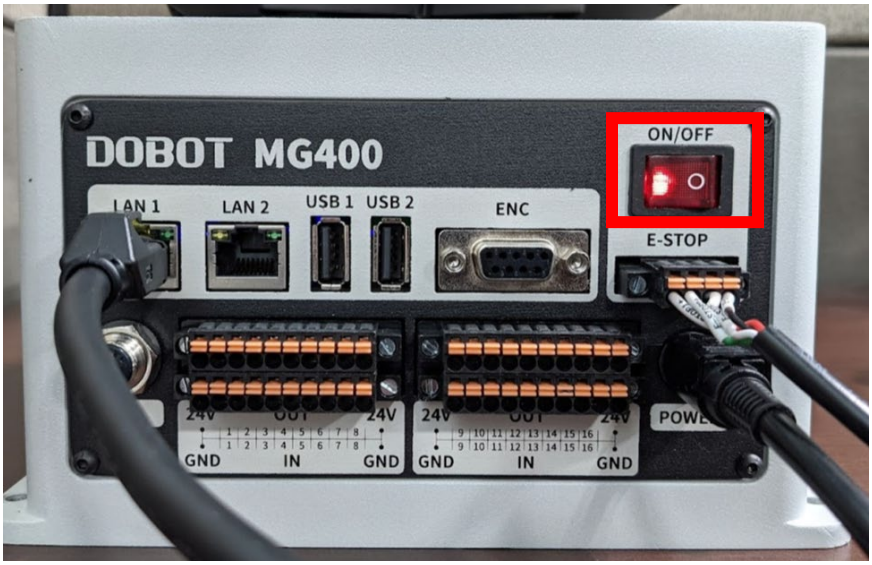


Make the following connections to the MG400:

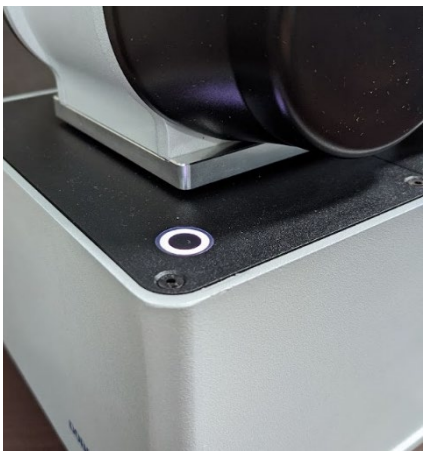
- Plug the power connector into a standard US wall outlet
- Plug the DC output of the power connector into the robot port labeled “POWER”
- Plug the E-Stop connector into the port labeled “E-STOP”
- Plug your Ethernet cable into the port labeled “LAN 1” and plug the other end into your PC

The result should look like this:

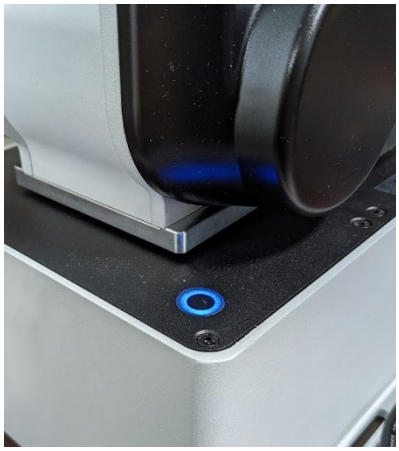
Next, flip the ON/OFF switch to the ON position. The switch LED should turn on, as shown here:



The LED on the front side of the robot base will now be blinking white to signify that the robot is booting up. Once the robot is ready for a connection, the LED will be a solid blue:



Robot Booting

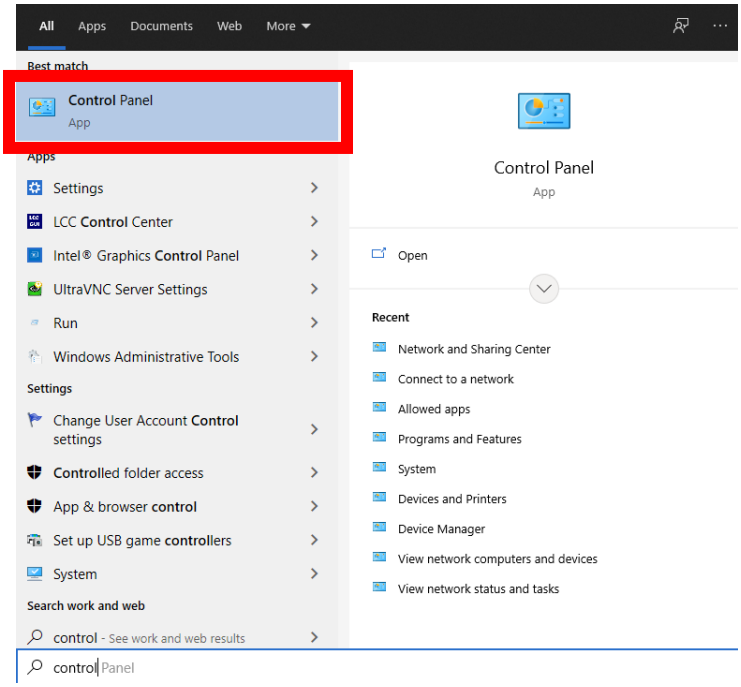


Robot Ready

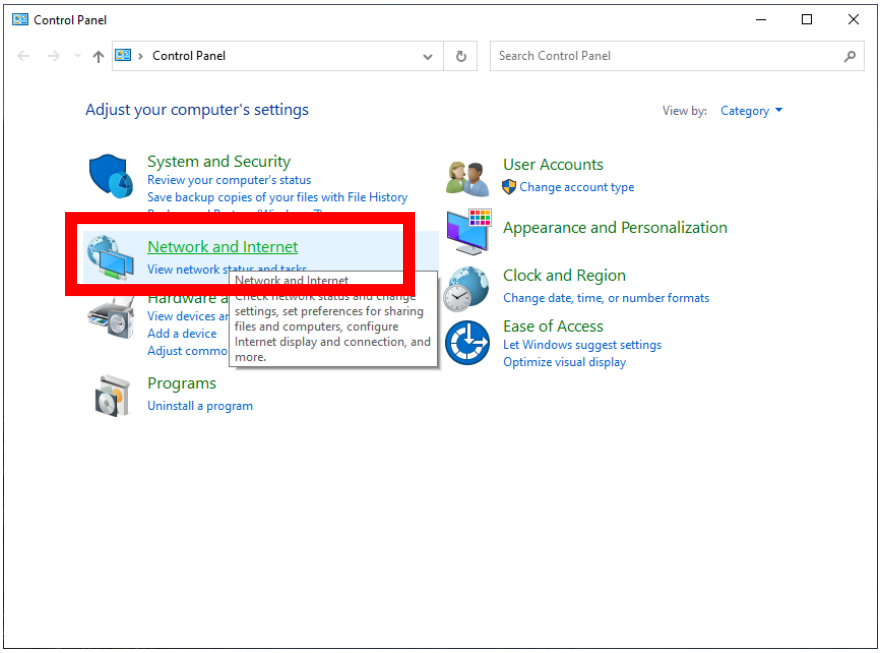
Configuring Your IP-Address

You'll need to configure the IP-Address of the MG400 to be able to connect to it with your computer.[1]

Go to the control panel:

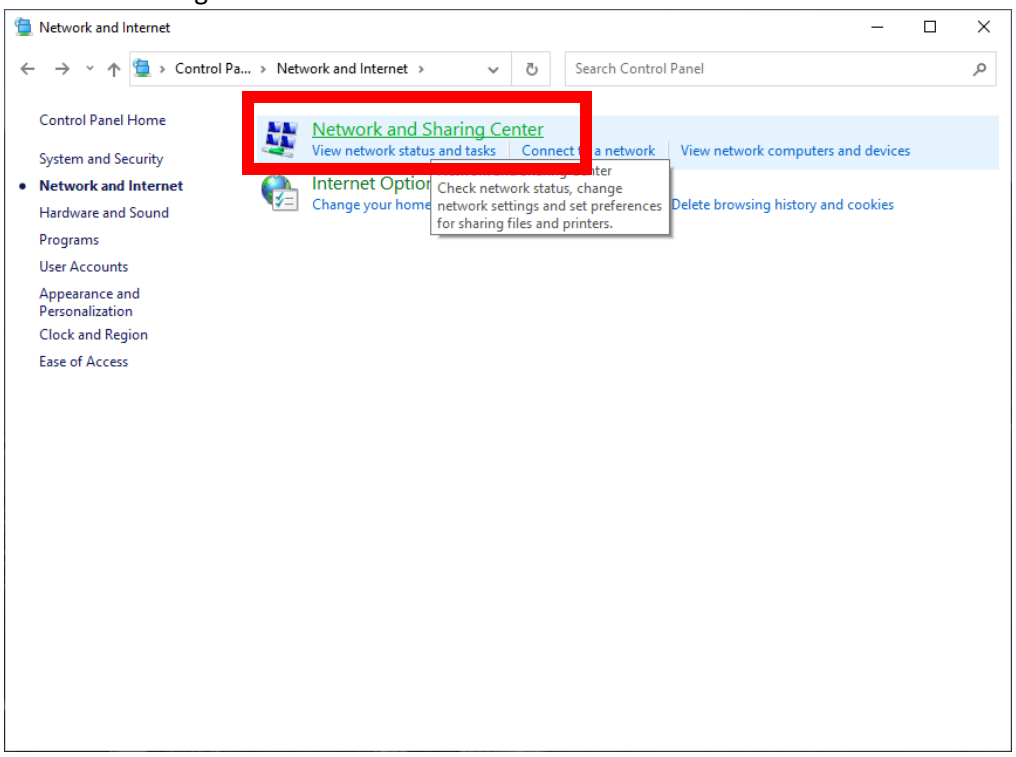


Click "Network and Internet"

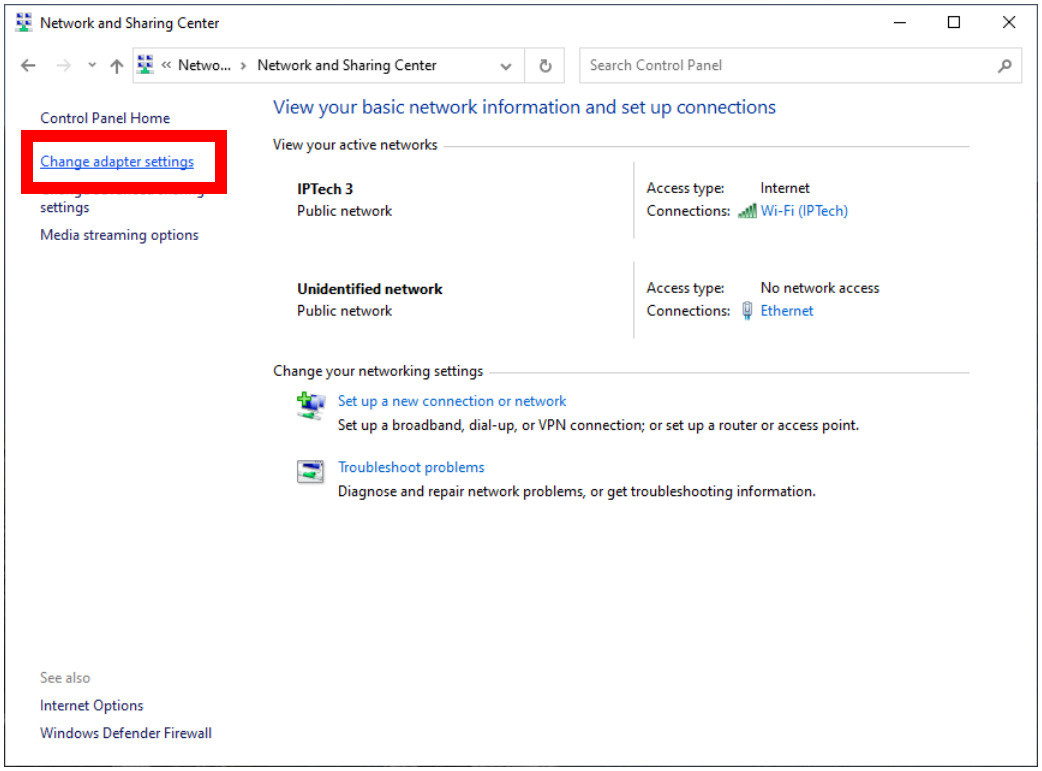


[1] For more information on why IP-Addresses are necessary, see here: https://www.paessler.com/it-explained/ip-address

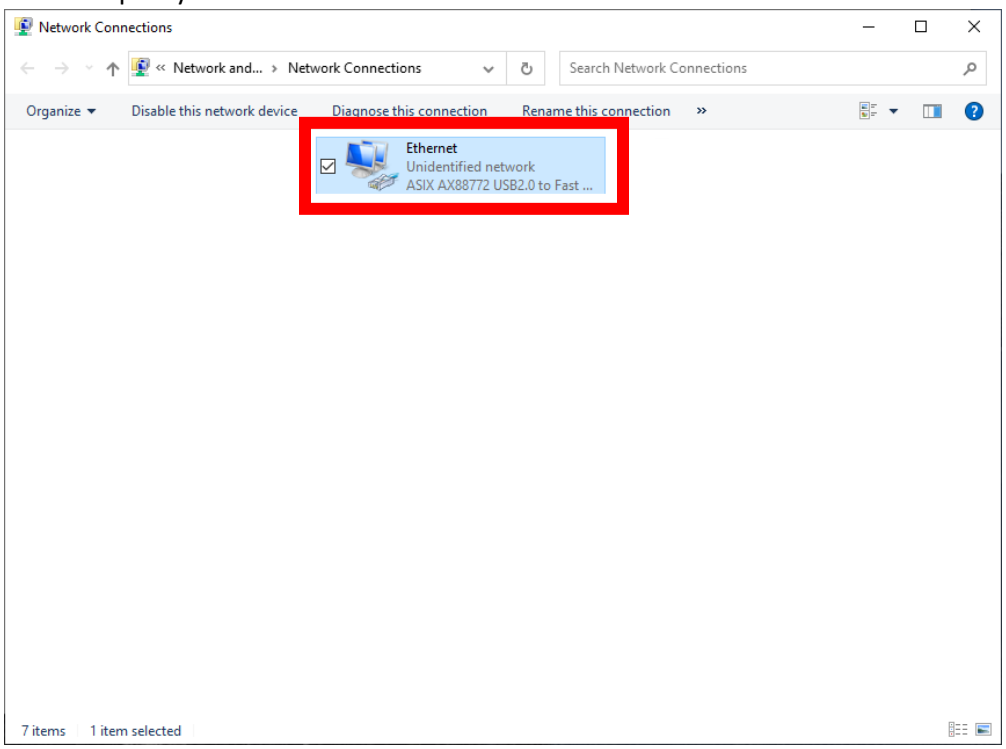
Click “Network and Sharing Center”:



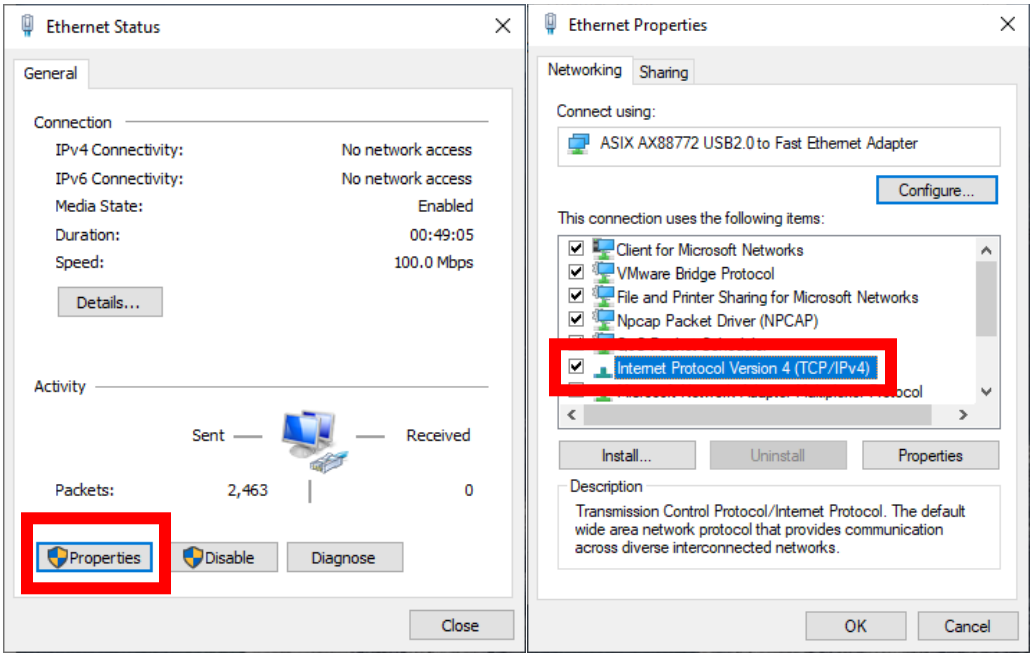
Click “Change adapter settings”:



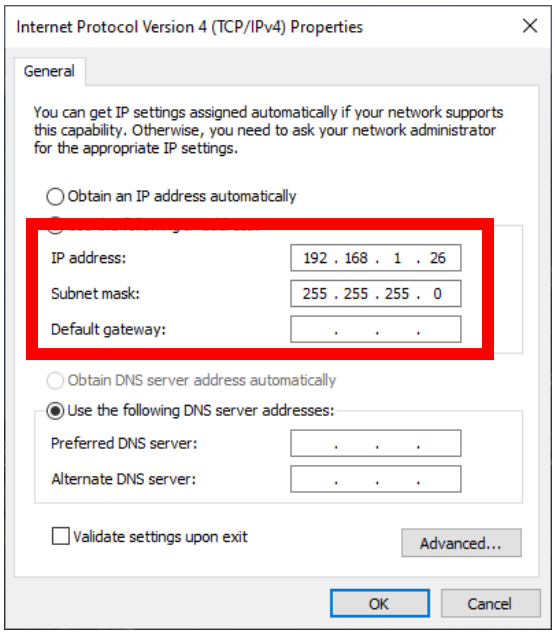
Double click the adapter you'd like to use:



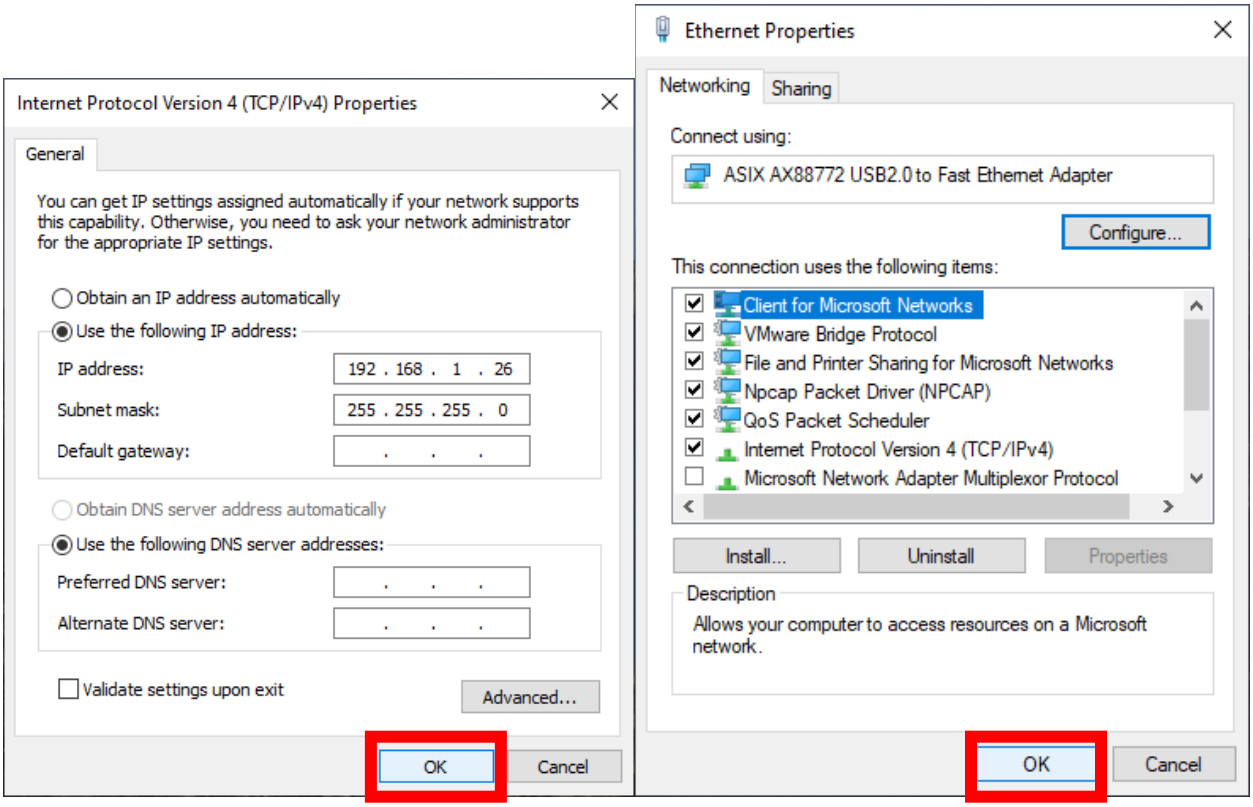
Click "Properties" and then double-click "Internet Protocol Version 4 (TCP/IPv4)"



The LAN1 port of the MG400 is always set to the IP address 192.168.1.6 with subnet 255.255.255.0. Set your IP-Address to 192.168.1.x where x is any number between 0 and 255 that is not 6. As an example, I used this one:



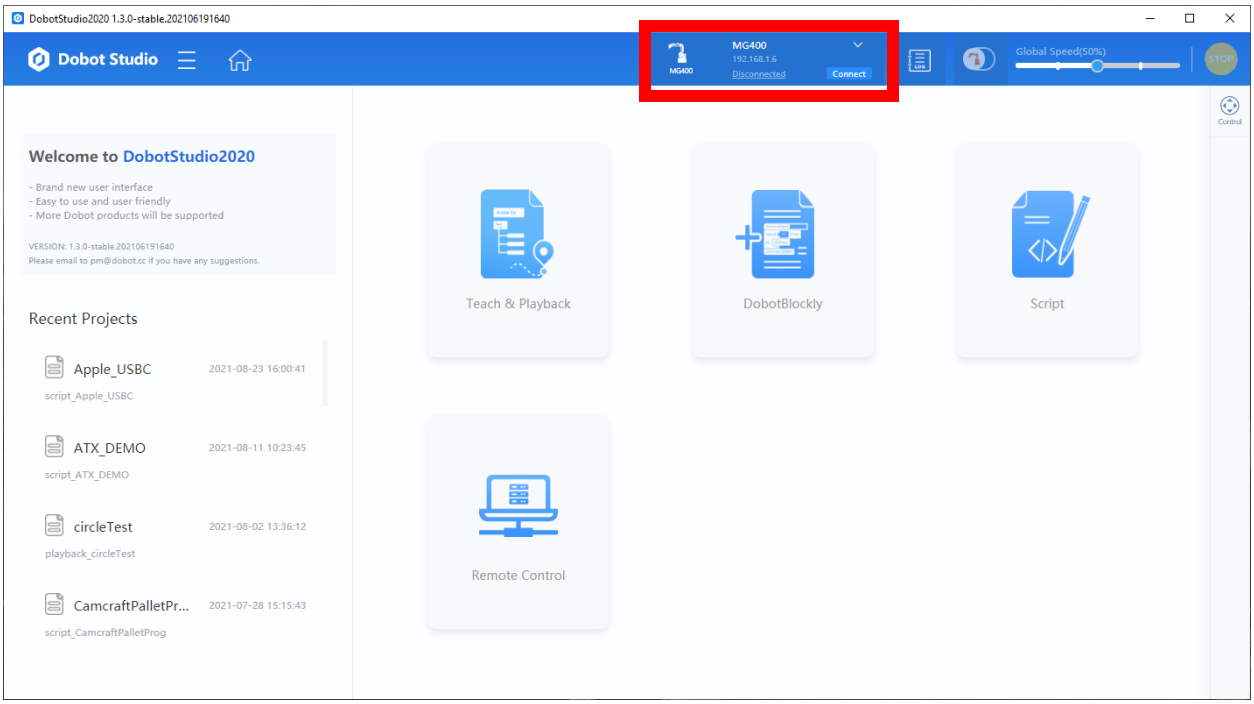
Leave the rest of the information blank and then hit okay to exit the IPv4 prompt. Hit okay a second time to exit the Ethernet properties prompt.



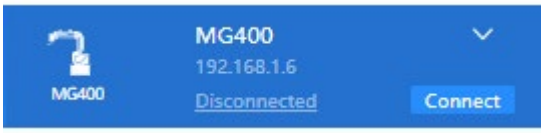
You are now ready to connect to the MG400.

Connecting to the MG400 with DobotStudio2020

Open DobotStudio2020. The screen should look like this:



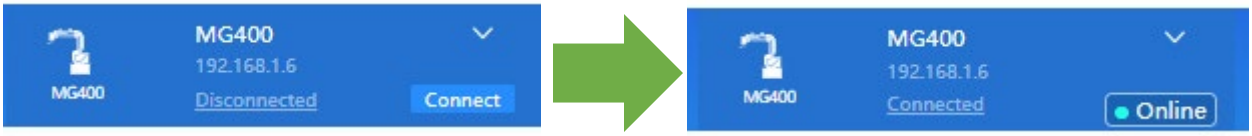
The search bar should have the MG400 pre-populated:



If this is not the case:

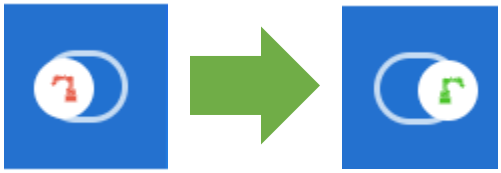
- Check that IP-Address is correct (See Configuring Your IP-Address)
- Check that the robot is powered on and shows a blue solid light (See Configuring Your Hardware)
- Try the Dobot connectivity guide (See Dobot Connectivity Guide for MG400)
- If the above options all fail, get in touch with your Dobot distributor for additional support

If you are able to connect, hit connect. The robot should now be online.



Enabling the Robot

In robotics, the robot is unable to move until it has been enabled. Before enabling the MG400, Joints 2 and 3 are mechanically locked and Joints 1 and 4 can be rotated freely. This state is considered the “disabled state.” After enabling the MG400 by left-clicking the button shown below, the MG400 is enabled. The robot’s joints are now active and cannot be moved by hand.



Disabled Enabled

Upon starting up and trying to enable for the first time, the MG400 will ask you for the payload information.

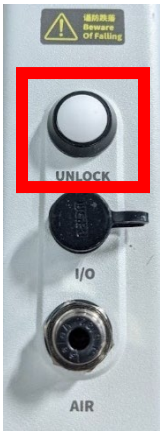
The definition of “Payload” is the net weight that you have attached to the robot. The MG400 can handle a maximum payload of 750g.

If you have not attached anything to the end of arm tool (like in the picture below), ensure that the payload is set to 0g. Knowing this weight helps the robot determine when it has collided with something while moving. The offset values are the x and y offsets of the center of gravity of your tool. You may leave these offsets blank most of the time.

If you attach something to the robot in the future, be sure to change the payload in the MG400 settings.

After enabling the robot, you are ready to begin motion. The front LED should now be green. If an error appears and the robot remains disabled, that means something is preventing the robot from booting. See Clearing MG400 Errors.

Using the Unlock Button to move the MG400

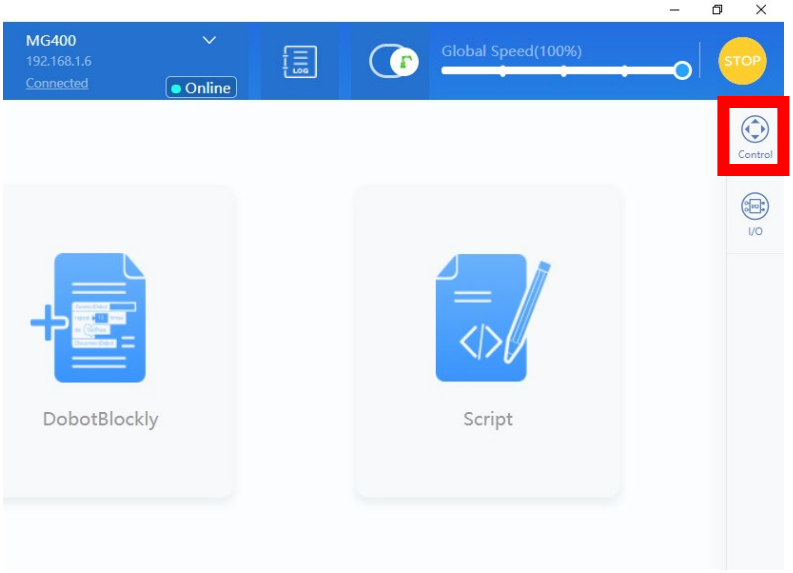


A great perk of the MG400 is the built-in unlock button. The unlock button allows the user to “drag and drop” the robot. This allows for fast programming and development compared to traditional methods.

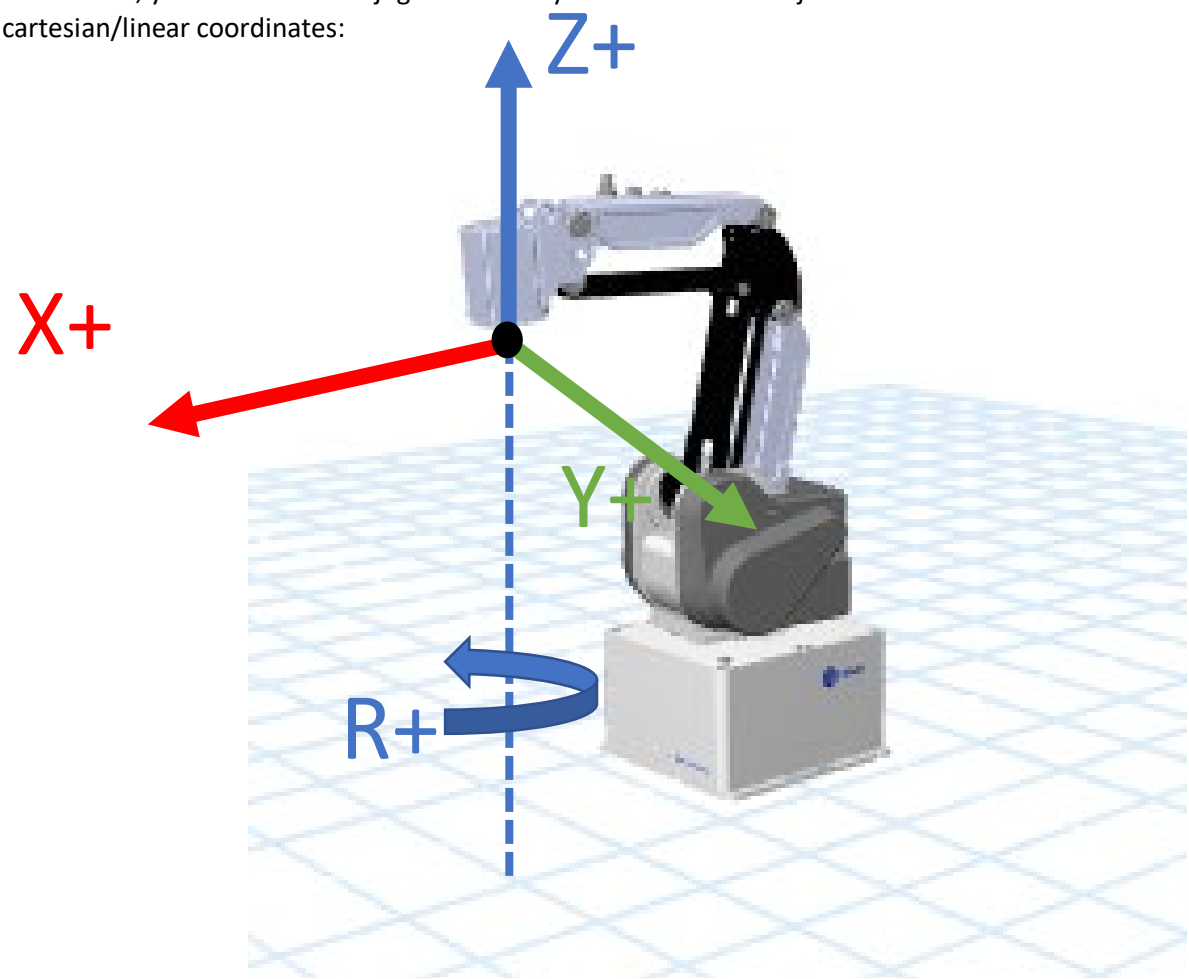
To use the unlock function, simply click the button on the GM400 arm. The green light will change to blinking blue. You can now re-position the robot freely. Be sure to click the “unlock” button again before commanding motion.

Using the Jog Panel to Move the MG400

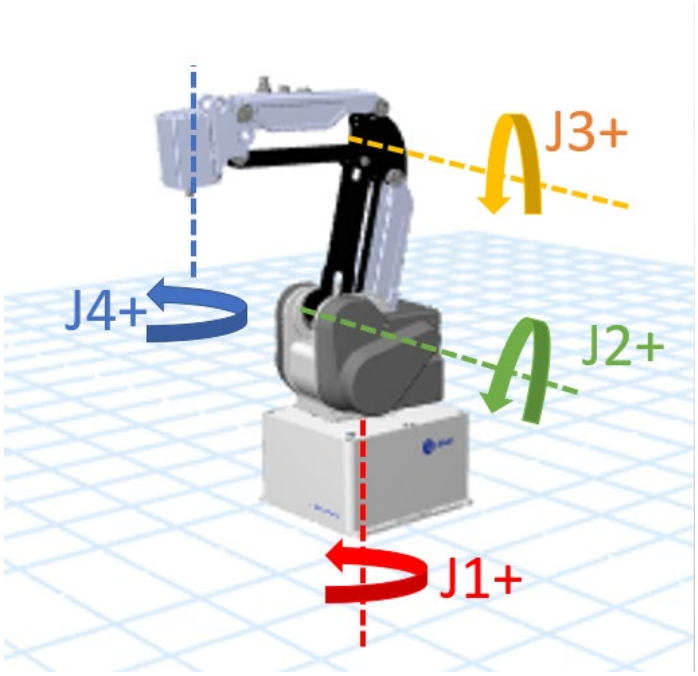
For a more repeatable way to move the robot, the user can use the jog window to position the robot electronically. To open the jog window, click “control” on the right side of DobotStudio2020:



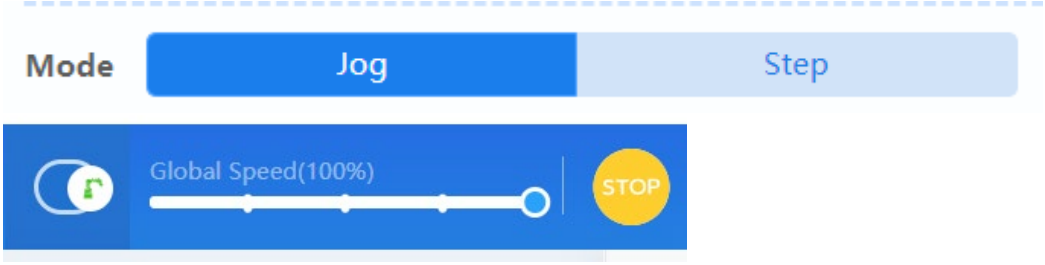
From there, you will be able to jog the robot by either cartesian or joint moves. Here are the cartesian/linear coordinates:



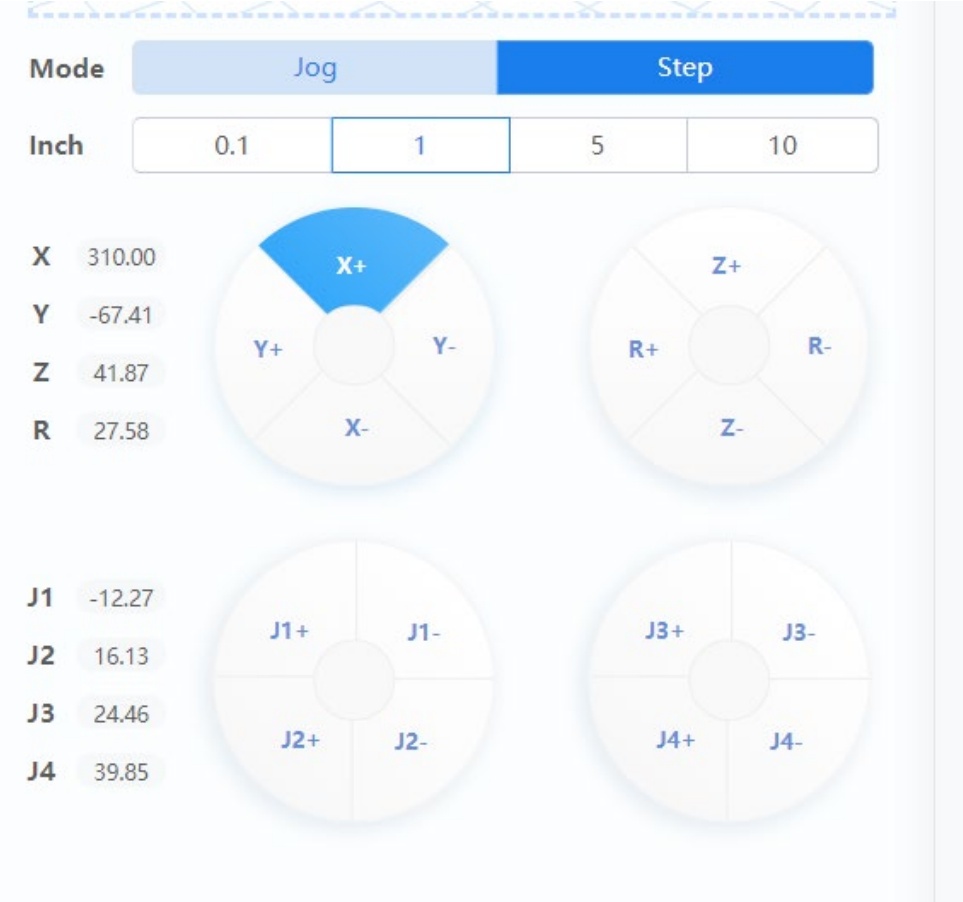
Here are the joint coordinates:



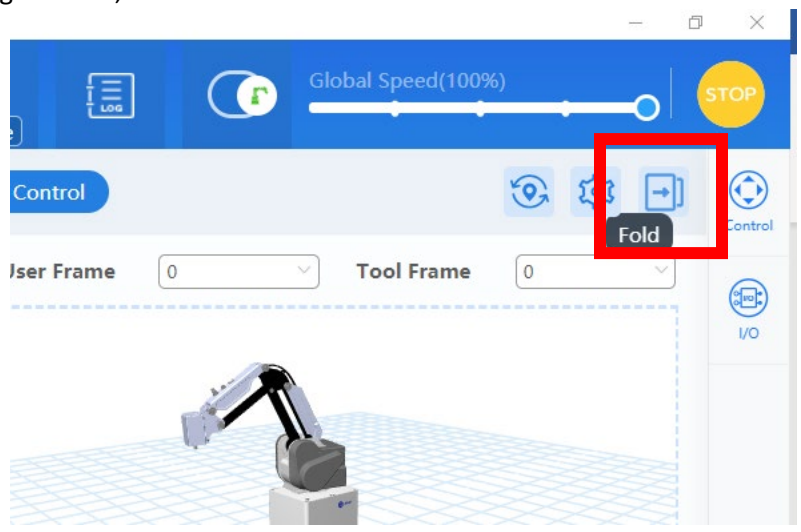
In “Jog” mode, the robot will jog in the selected direction until the button is released. The speed of the robot is controlled by the speed slider on the top of the screen.



In “Step” mode, the robot will increment the selected distance in mm whenever the user taps a button. For example, the following button configuration would drive the robot 1mm in the x-direction when clicked:

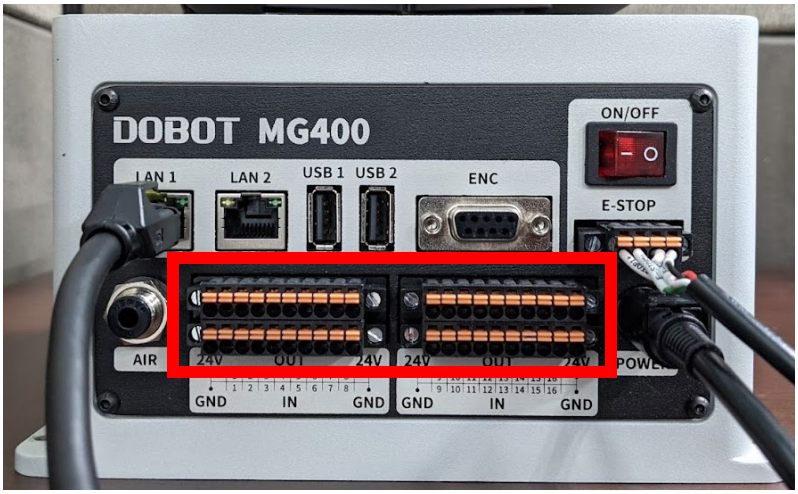


To get rid of the jog window, hit the “fold” button:



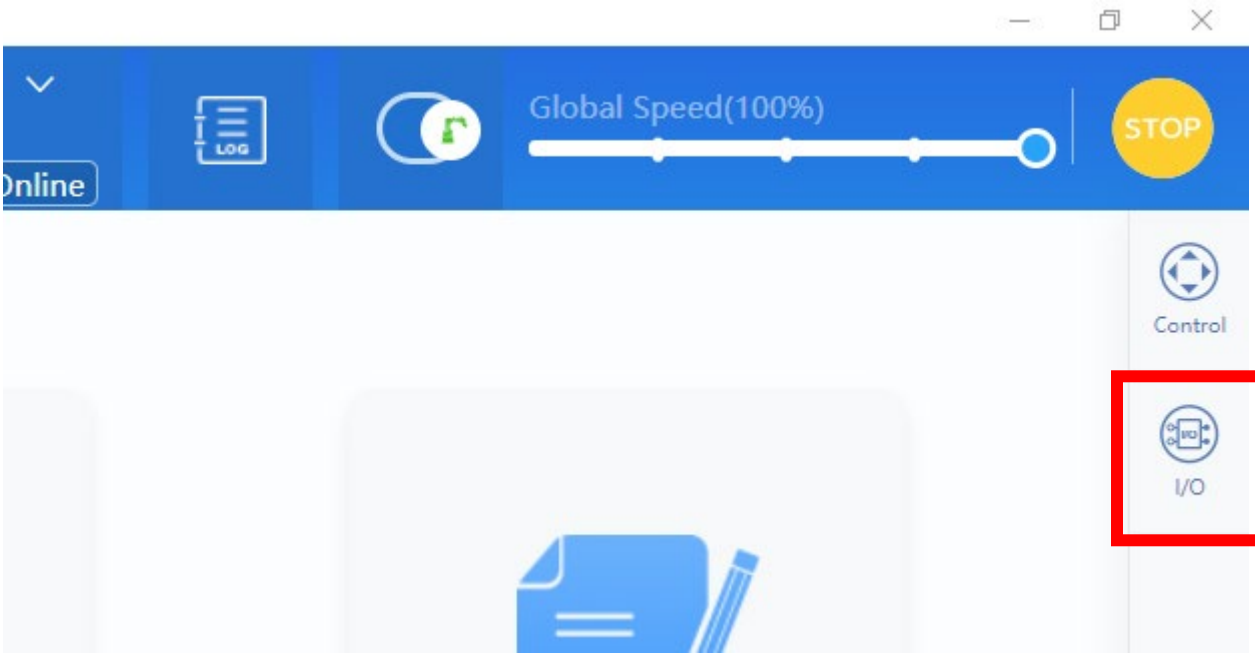
MG400 IO

The MG400 has 16 digital inputs and 16 digital outputs that function on 24VDC that are visible on the back of the robot base. The MG400 also has 2 digital inputs and 2 digital outputs on the arm of the robot for easy access to robot tools. Dobot provides the connector for this IO with the MG400.



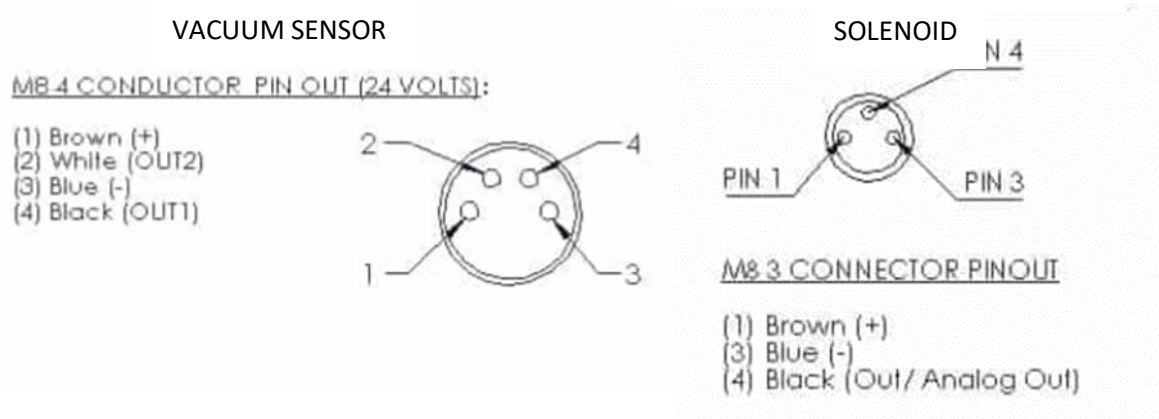
To use the rear-IO, compress the orange tab and insert a wire. Release the tab and the wire should be firmly clamped in the IO slot. We recommend zip-tieing your automation setup so the IO wires don't have any strain.

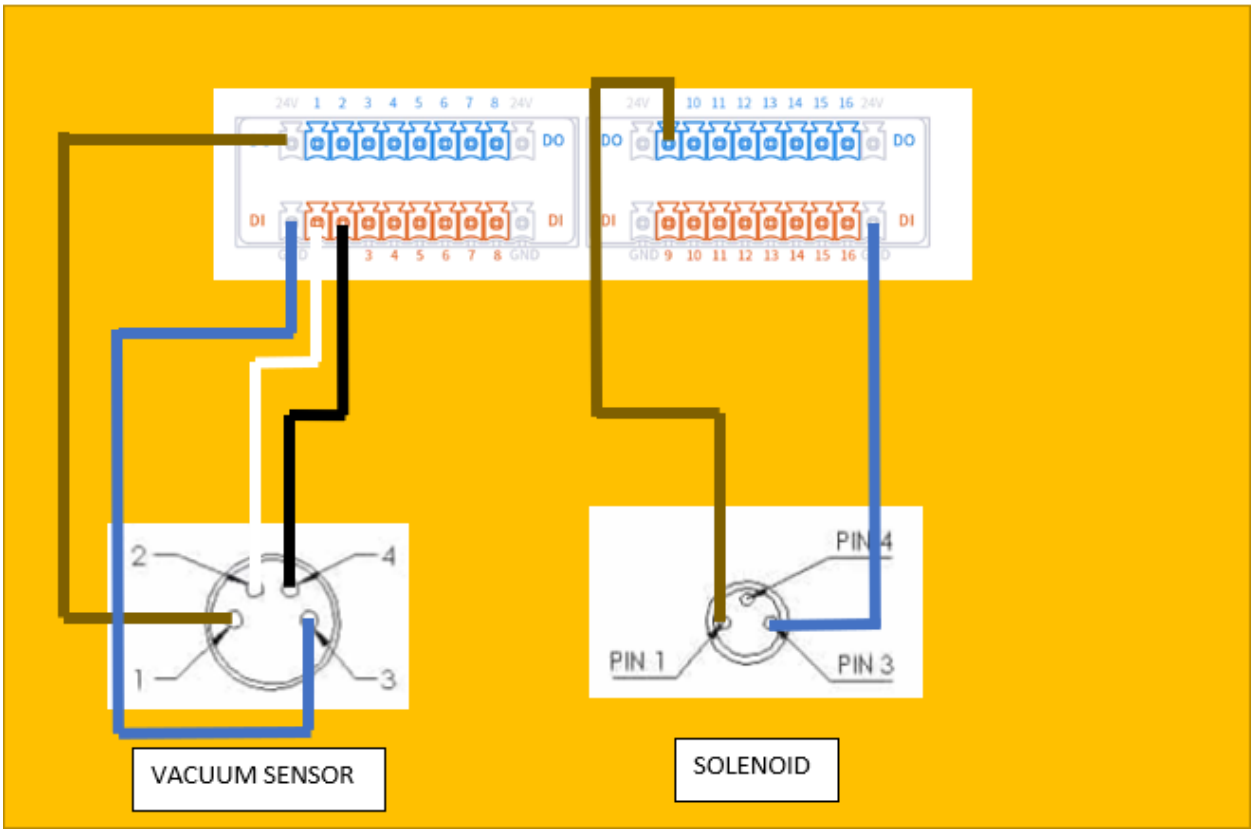
To view IO status and toggle IO easily, use the "I/O" tab:



MG400 IO Wiring Example

Here is an example on how to wire two 24VDC sensors with the following pinouts:



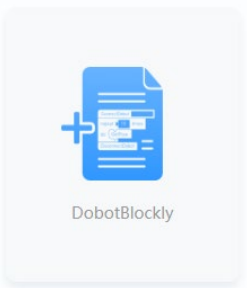


Note: your sensors may have different wiring. Incorrect wiring could damage the MG400.

Programming the MG400

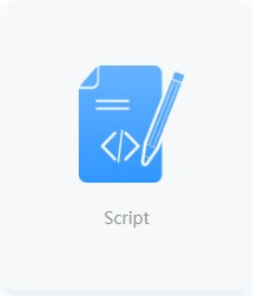
Basics

There are many ways to program the MG400. In this guide, we will cover Blockly and Script. Blockly is a visual, block-based programming method. Script is programming in Lua, a high-level programming language similar to Python. Generally, Blockly is friendlier to new users and quick applications while Script is preferred by more experienced users and is suited towards



Blockly

- Visual block-building
- Also known as “Scratch”
- Suited towards simple applications



Script

- Simple structured-text programming in Lua
  - Similar to Python
- Suited towards more advanced applications/more experienced users

For a first-time application, it is recommended to start with Blockly. You may port your project to a script project (see Porting Blockly Program to Script).

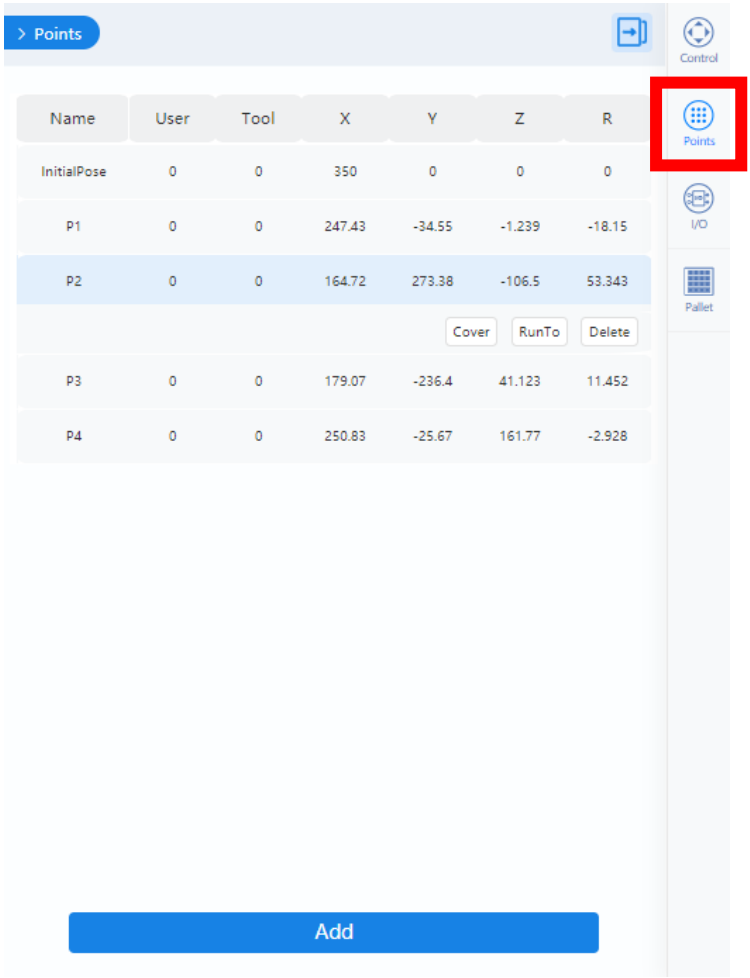
Saving Points

Saving points is the same in Blockly and in Script and is usually done before writing any code.
Open the points list in the top right of the screen. Hitting “add” will take the robot’s current cartesian position and save it as a point.

You can rename points by double-clicking the name P1, P2, etc.

You can tweak a point by double-clicking an X,Y,Z, or R value.

By clicking a point, you are given the options to “Cover”, “RunTo”, or “Delete”. Cover will overwrite the selected point with the current position of the robot. RunTo will move the robot to the selected point. Delete will remove the point.



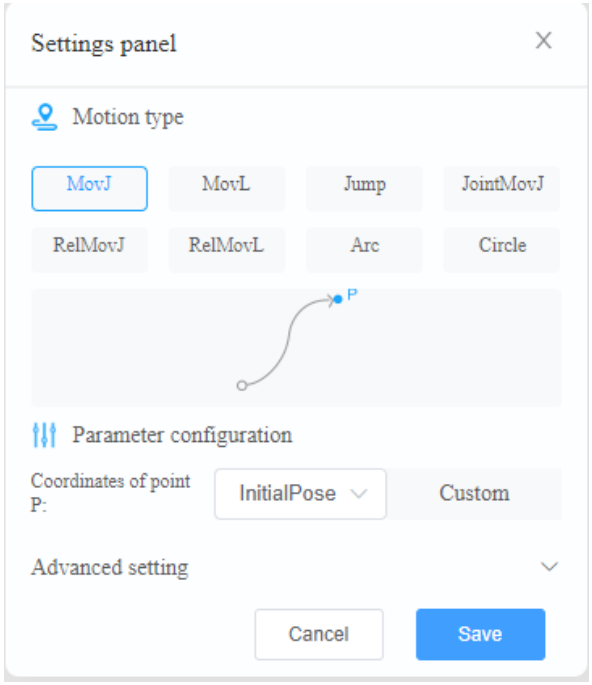
Different Move Types

In Dobot Studio, you'll find several move types. Here is a list of their functions and where to use them:

- MovJ
  - Joint-based trajectory move to point P
  - Fastest, most efficient move

- MovL
  - Linear Trajectory to point P
  - Good for insertions, picking parts, paths

- Jump
  - Builds in a routine that moves linearly up a height and down to point P
  - List of parameters in settings



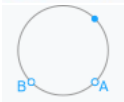
- JointMovJ
  - Move to joint position instead of cartesian position
  - Generally not used

- RelMovJ
  - MovJ to an offset X,Y,Z,R position instead of to a fixed point
  - Joint Trajectory

- RelMovL
  - MovL to an offset X,Y,Z,R position instead of to a fixed point
  - Linear Trajectory

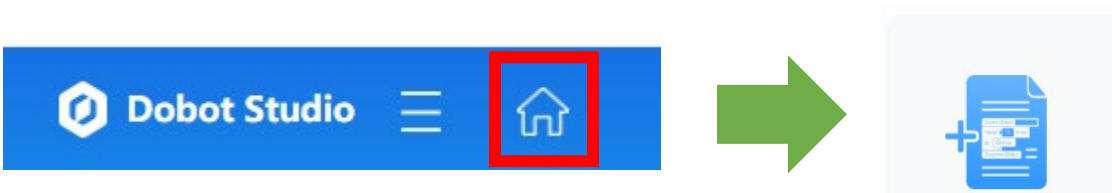
- Arc
  - Partial circular trajectory

- Circle
  - Full circle trajectory

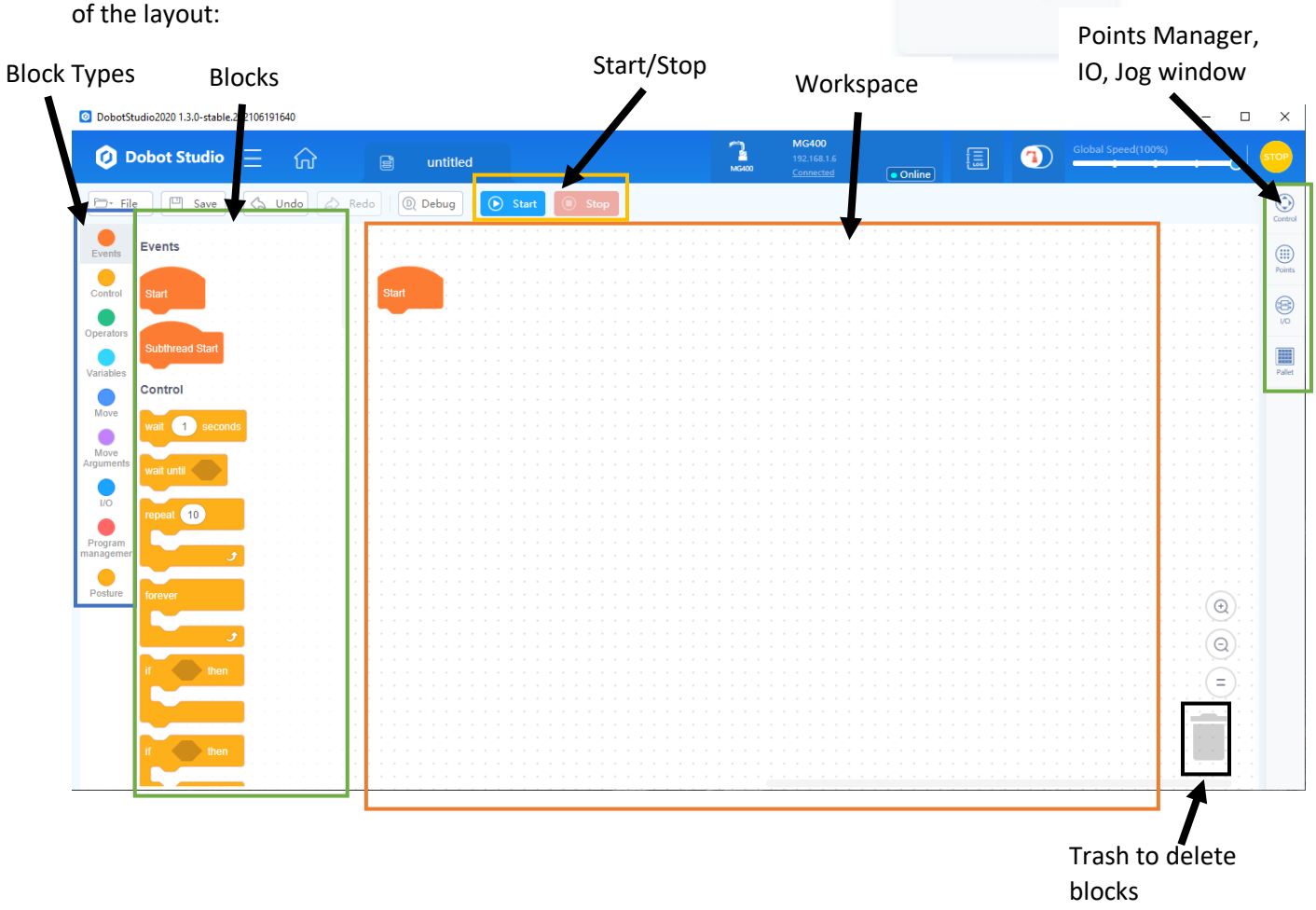


Blockly

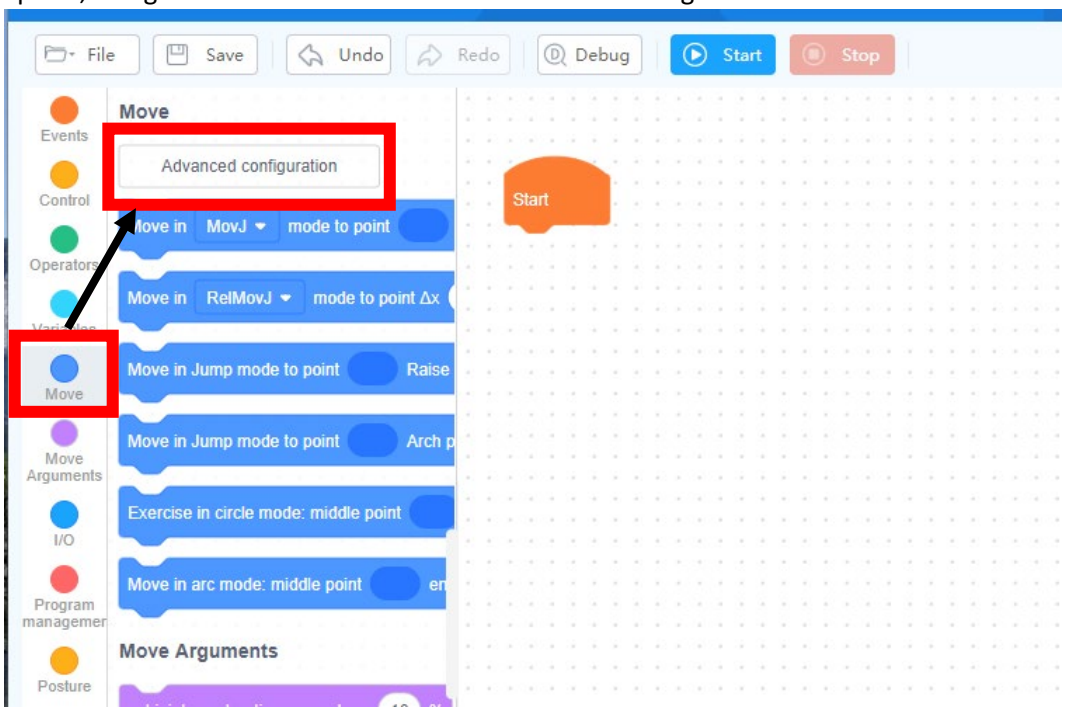
Open up a new Blockly program by clicking the “Homepage” button and then clicking the “DobotBlockly” button



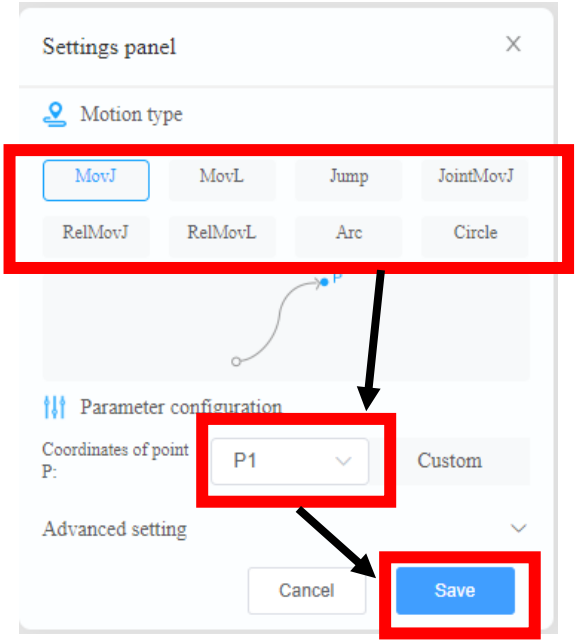
This will open up the Blockly Programming Interface. These are the key points of the layout:



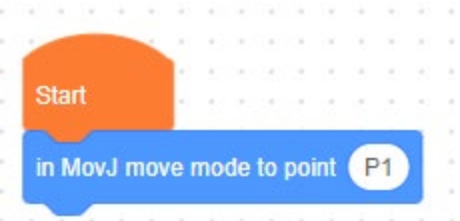
Start your program by adding some points in your points list (See Saving Points).
To add a point, navigate to “Move” and then hit “Advanced configuration”:



This will open the move panel. Select your type of move and hit “Save” (for more info, see Different Move Types)



Drag the block into a position in your blockly workspace. Note that the blocks need to “click” together.

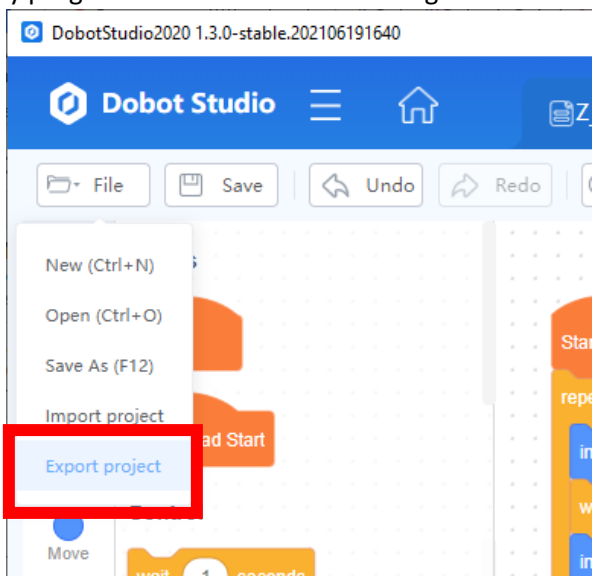


To copy blocks, select a block, hit “ctrl + x”, and “ctrl + v” to paste. From there, you can edit the point names. You can also right click and hit “duplicate” to make copies.
From there, you can make simple programs. For example, the program shown here will move to points 1, 2, 3, and 4.

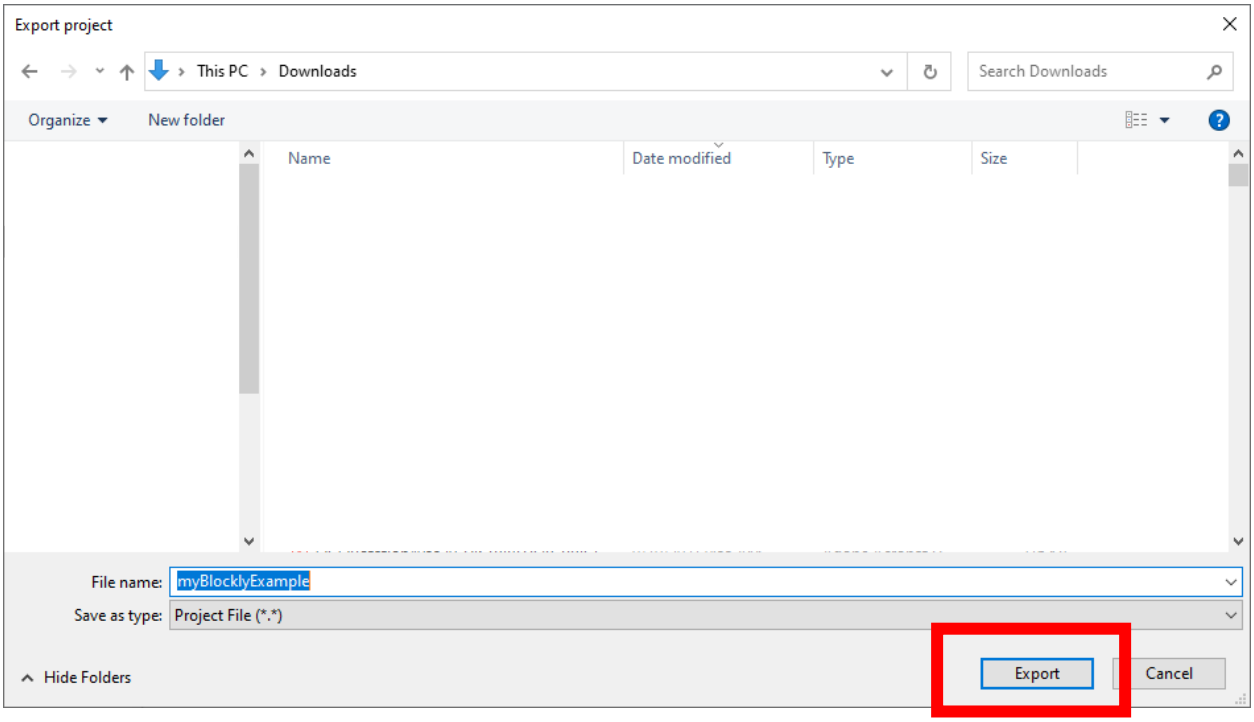
Porting Blockly Program to Script

Often times, it's easier to start in Blockly and transfer to script once the basic program is there.

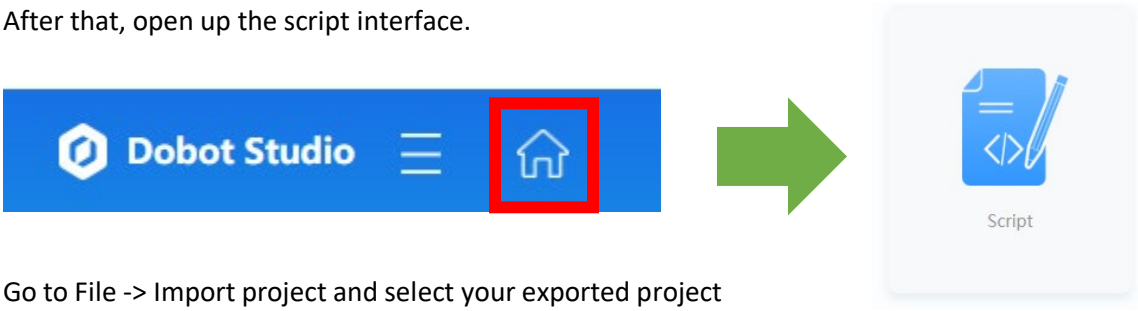
Start by opening your blockly program. Be sure it is saved. Navigate to File -> Export Project



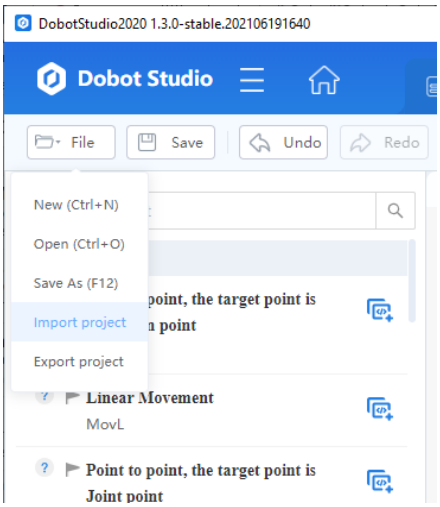
Name the file and save it somewhere you will remember. Then hit "Export"

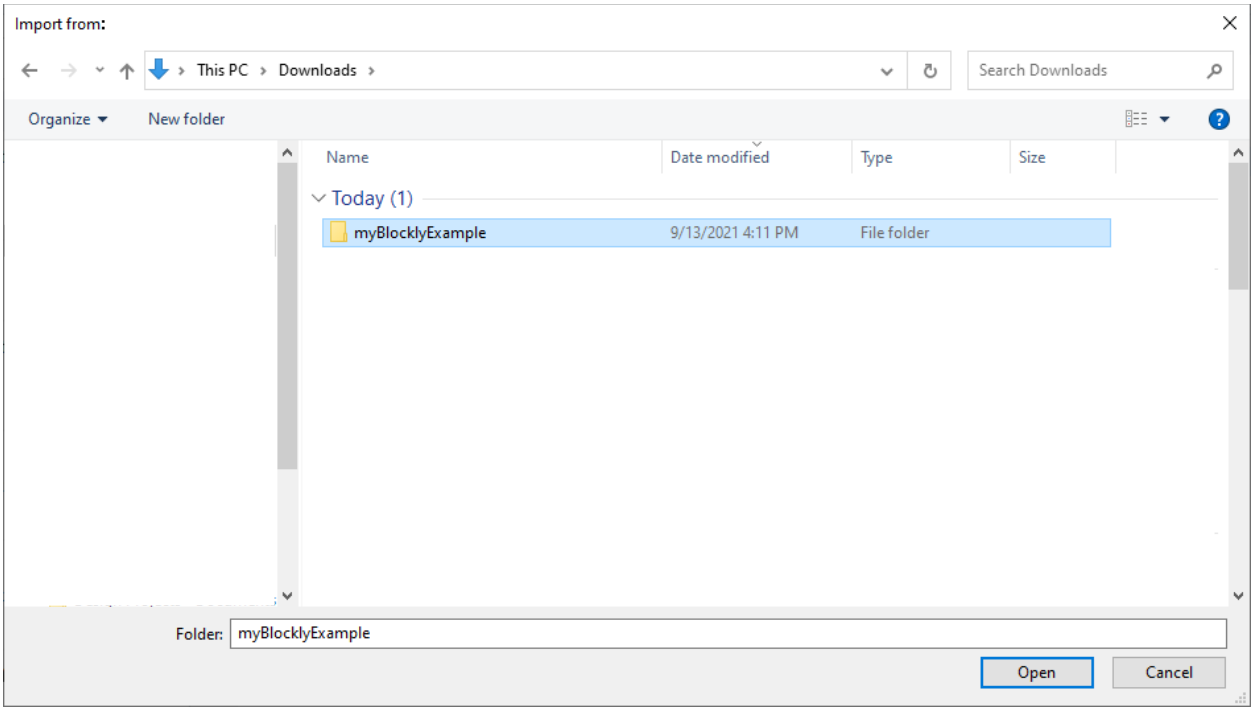


After that, open up the script interface.



Go to File -> Import project and select your exported project

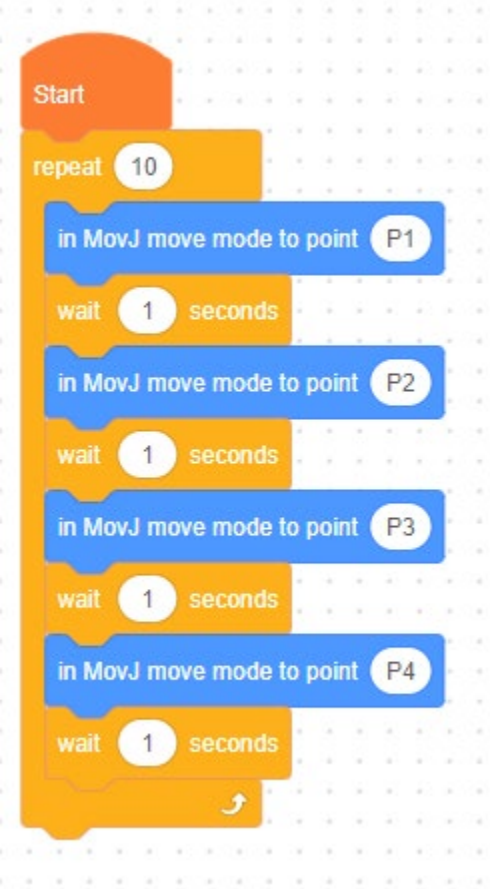




Your blockly program will now be displayed as script.

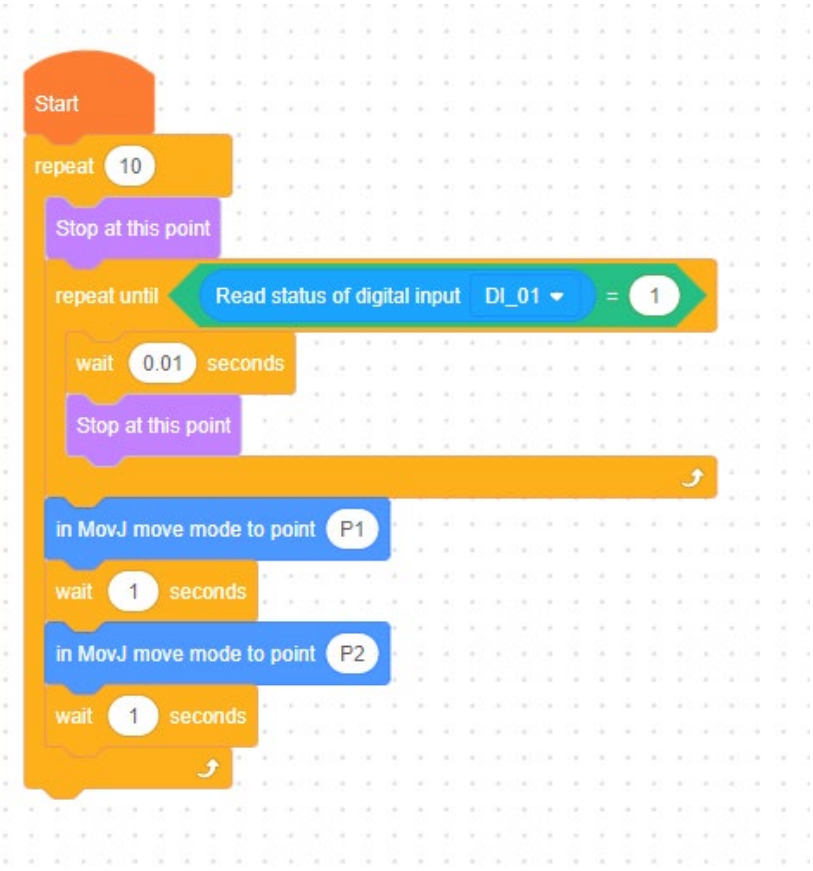
Blockly Example 1: Basic Moves

This program will run to points 1, 2, 3, and 4. At each point, the MG400 will wait 1 second to proceed. This process will repeat 10 times before ending.



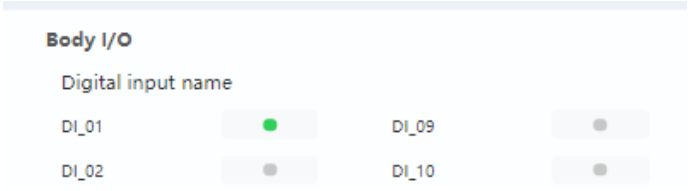
Blockly Example 2: Using IO #1

This program will execute a loop 10 times. Each time, it will wait until Digital Input 1 is high. After the signal goes high, it will execute 10 times.



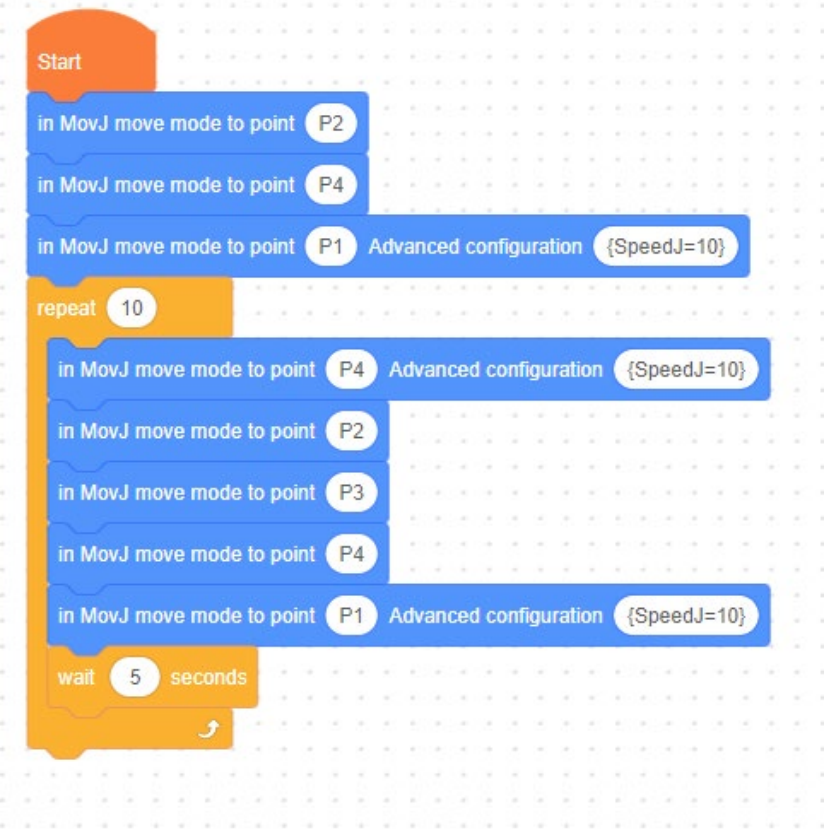
Without the "Stop at this point" command, the program will not work. For more information on why the Sync function is necessary, see Using Sync with IO Commands.

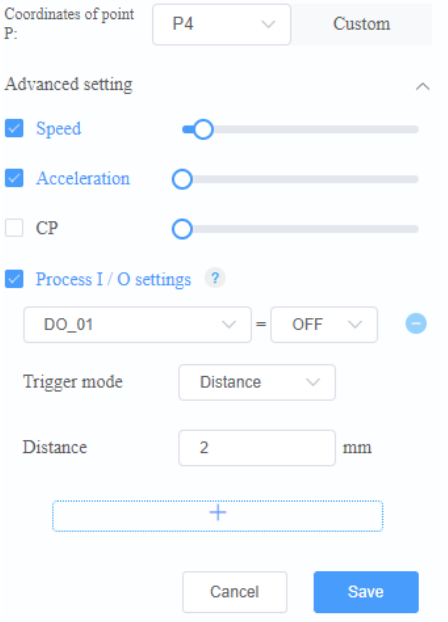
Remember you can monitor this with the IO panel to help debug: (right side of studio)



Blockly Example 3: Modified Parameters in Move

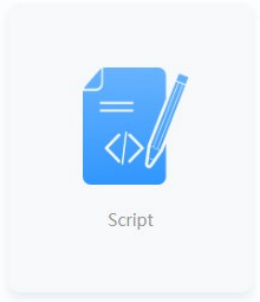
By right-clicking a move and hitting “edit”, you may make special configurations for that move. For example, you can have slower velocity and acceleration on an insertion move. Or you can Have an IO toggle a certain distance into a move.





Script

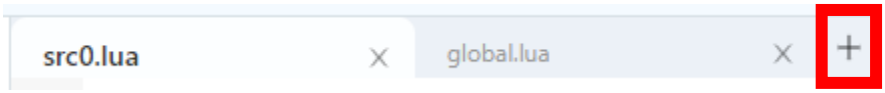
Open up a new Script program by clicking the “Homepage” button and then clicking the “Script” button



This scripting language is called Lua. It is similar to Python and should be familiar to those who know structured text programming. Aside from the Dobot functions, everything in Lua is supported here.[2]

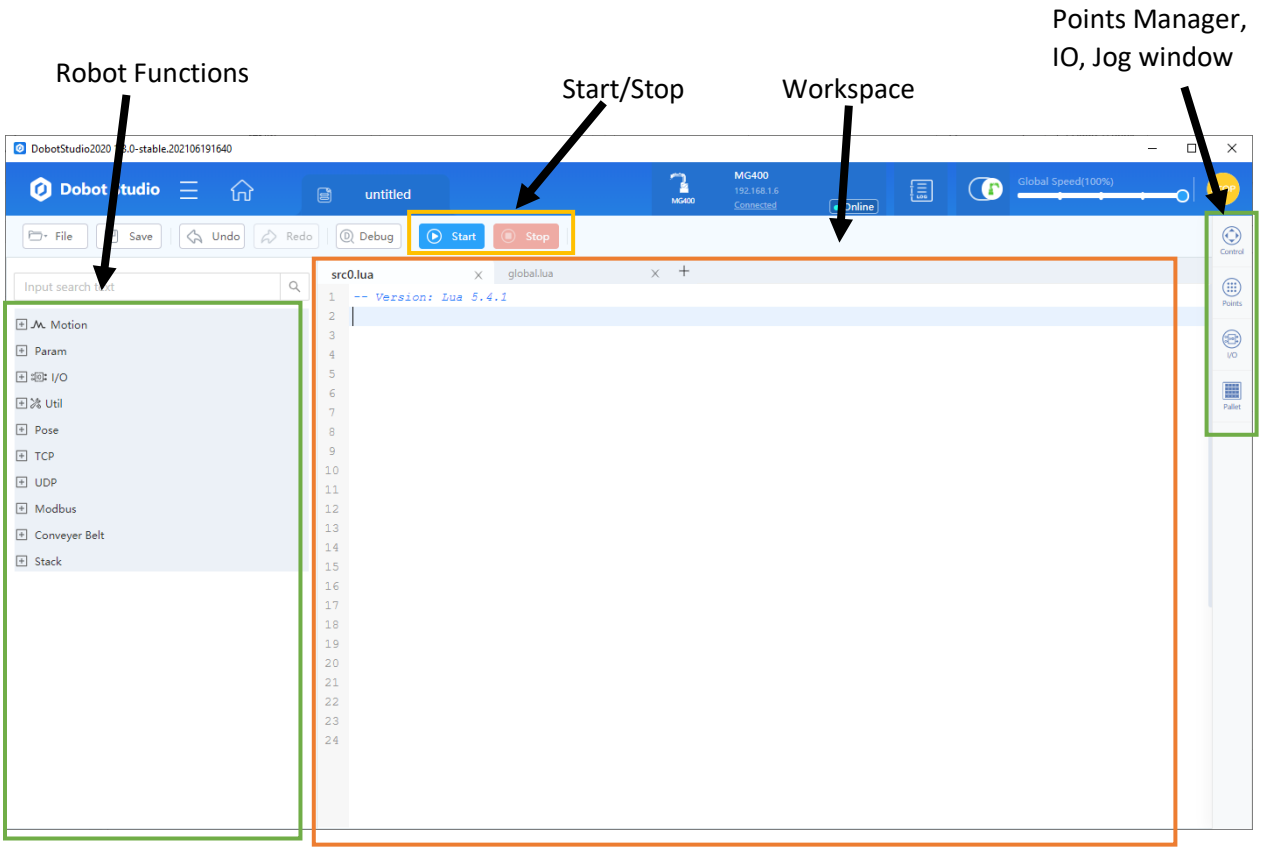
There are two files that open on bootup: scr0.lua and global.lua.

- scr0.lua
  - Used for motion commands
  - Used for sequential code
- global.lua
  - Used for global variables
  - Used for user functions
- scr1.lua, scr2.lua, etc.
  - User may create as many parallel threads as they desire by hitting the +

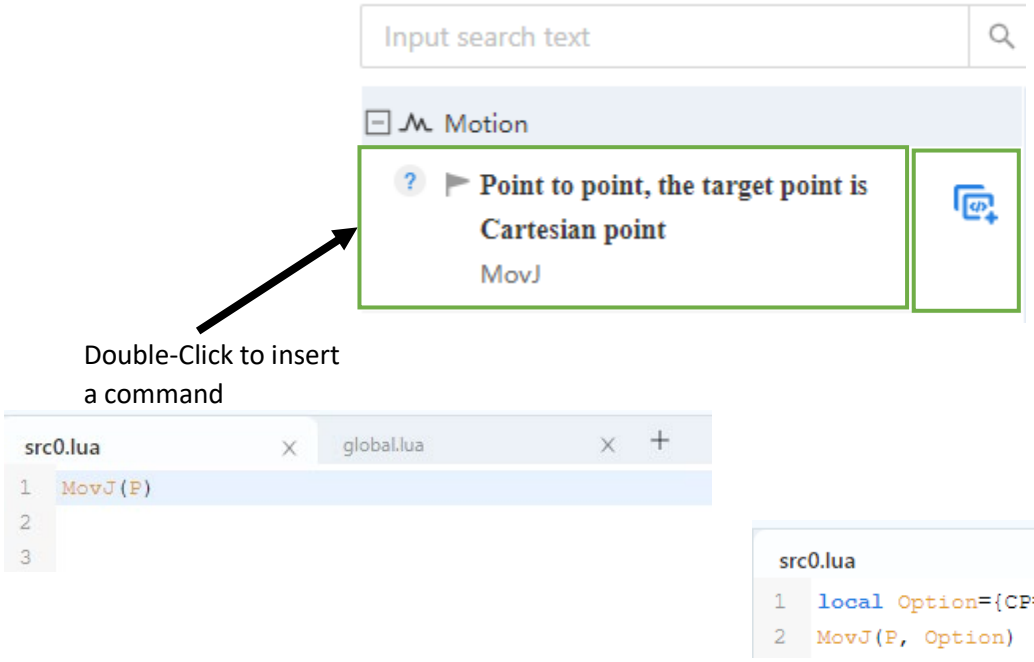


  - Recommended for monitoring IO, TCP/IP communications, etc.
  - It is recommended to avoid motion commands in parallel threads. This could lead to erratic behavior

[2] If you are unfamiliar with structured-text programming, there is a great Lua guide here: https://www.tutorialspoint.com/lua/index.htm


Robot Functions
Start/Stop
Workspace
Points Manager,
IO, Jog window
untitled
MG400
Connected
Online
Global Speed(100%)
File
Save
Undo
Redo
Debug
Start
Stop
Input search text
Motion
Param
I/O
Util
Pose
TCP
UDP
Modbus
Conveyer Belt
Stack
src0.lua
global.lua
-- Version: Lua 5.4.1
Control
Points
I/O
Pallet

Users may use the built-in robot functions on the left to quickly deploy code without worrying about Syntax:



"--" signifies a comment in Lua

Script Example 1: Basic Moves

This script runs to P1 and P2 with normal parameters. Afterwards, it goes to P3 with 50% speed and 20% velocity with a CP = 1.

src0.lua

```
MovJ(P1) --move to P1 with a joint trajectory
MovL(P2) --move to P2 with a linear trajectory
MovJ(P3, {CP=1, SpeedJ=50, AccJ=20}) --move to P1 with a joint trajectory at 50% speed
```

global.lua

```
--empty
```

Script Example 2: Basic IO Operations

This script iterates 10 times in a loop. It starts by moving to P3 and then will move to P1 is DI1 is high or P2 if DI1 is not high.

src0.lua

```
for count = 1, 10 do
  MovJ(P3)
  Sync()
  if (DI(1))==1 then
    MovJ(P1)
  else
    MovJ(P2)
  end
  Wait(1000)
end
```

global.lua

```
--empty
```

Script Example 3: Custom Function

If you have repetitive code, use a custom function to save on time and programming efficiency. The below example uses a custom function named “partInsertion()”. That function is defined in global.lua and called in scr0.lua during execution.

src0.lua

```
while true do
  MovJ(P1) --clearpoint
  MovJ(P2) --process point 1
  partInsertion()
  MovJ(P3) --process point 2
  partInsertion()
  MovJ(P4) --process point 3
  partInsertion()
  Wait(1500)
end
```

global.lua

```
function partInsertion()
  RelMovL({0, 0, -10,0},{CP=0, SpeedL=20, AccL=20}) --move 10mm downwards at 20% speed
  Sync() --Sync() before IO commmand
  DO(1,ON) --toggle DIO on
  Wait(1500) -- wait 1500ms (1.5s)
  Sync() --Sync() before IO commmand
  DO(1,OFF) --toggle DIO off
  RelMovL({0,0,10,0}) --move back up
end
```

Script Example 4: TCP/IP Client

You can have the MG400 run a program on-board

src0.lua

```
-- Version: Lua 5.4.1
--Robot code:
--- VAR CONFIG
local ip="192.168.1.6" -- IP address of the robot as a server
local port=6001 -- Server port
local err=0
local socket=0

--- PROGRAM
err, socket = TCPCreate(true, ip, port)
if err == 0 then
        err = TCPStart(socket, 0)
        if err == 0 then
                local RecBuf
                while true do
                        TCPWrite(socket, "tcp server test") -- Server sends data
                        err, RecBuf = TCPRead(socket,10,"string") -- Server receives
                        if err == 0 then --if you received a message…
                                Go(P1) --Start to run motion commands
                                Go(P2)
                                print(RecBuf.buf)
                        else
                                print("Read error ".. err)
                                break
                        end
                        Wait(100)
                end
                else
                print("Create failed ".. err)
        end
        TCPDestroy(socket)
else
        print("Create failed ".. err)
end
```

global.lua

```
--empty
```

This may be run with the python script shown here:

MG400_simpleSocket.py

Script Example 5: Variable palletizing

This script takes in variables for X,Y,Z, and R positions of two pallets sitting in any X-Y plane. This example program is of a pick + camera inspect + place sequence.

src0.lua

```
-- VARIABLES (TO BE POPULATED BY USER)
--IO items
cameraSignal_in = 5 --set this to the digital input of the camera
cameraSignal_out = 3 --set this to the digital output trigger from the robot to the
camera
gripperSignal_out = 2 --set this to the digital output for the gripper

--input pallet parameters (this is where you pick up parts)
x_input_startingPoint = 300 --x coordinate of robot when above the first pallet slot
y_input_startingPoint = 100 --y coordinate of robot when above the first pallet slot
x_input_pallet = 10 --x distance between spots in pallet, defined in mm
y_input_pallet = -25 --y distance between spots in pallet, defined in mm
z_input_pallet = 10 --defined in mm, same for all input pallet points
r_input_pallet = 90 --defined in degrees, same for all input pallet positions
x_input_num_iterations = 4
y_input_num_iterations = 3 --this is an 8x3 pallet where we have 8 rows (x shifts), 3
columns (y shifts)
num_parts_input_pallet = x_input_num_iterations * y_input_num_iterations --calculate
total input items

--output pallet parameters (this is where you put good parts)
x_output_startingPoint = 250 --x coordinate of robot when above the first pallet slot
y_output_startingPoint = -70 --y coordinate of robot when above the first pallet slot
x_output_pallet = -15 --x distance between spots in pallet, defined in mm
y_output_pallet = 26 --y distance between spots in pallet, defined in mm
z_output_pallet = -10 --defined in mm, same for all output pallet points
r_output_pallet = 0 --defined in degrees, same for all output pallet positions
x_output_num_iterations = 3
y_output_num_iterations = 4 --this is an 8x3 pallet where we have 8 rows (x shifts), 3
columns (y shifts)
num_parts_output_pallet = x_output_num_iterations * y_output_num_iterations --
calculate total input items

--reset counters
x_input_iter = 0
y_input_iter = 0
x_output_iter = 0
y_output_iter = 0
input_pallet_finished = 0
output_pallet_finished = 0

-- PROGRAM VARIABLES (DO NOT MODIFY)
--set inital pick and place point
pickPoint = {armOrientation="right",
coordinate={x_input_startingPoint+x_input_iter*x_input_pallet,
```

```
y_input_startingPoint+y_input_iter*y_input_pallet, z_input_pallet, r_input_pallet,
0.000000, 0.000000},tool=0, user=0}
placePoint={armOrientation="right",
coordinate={x_output_startingPoint+x_output_iter*x_output_pallet,
y_output_startingPoint+y_output_iter*y_output_pallet, z_output_pallet,
r_output_pallet, 0.000000, 0.000000},tool=0, user=0}

-- PROGRAM (USER MAY MODIFY)
Go(clearPoint) --this is my clear point
print('Starting Program')
while (input_pallet_finished == 0) and (output_pallet_finished == 0) do
  --pick part
  MovJ(pickPoint)
  inputPalletIter()
  pickPart()

  --INSPECT PART
  MovJ(cameraPoint)
  DO(cameraSignal_out,ON)
  Sync()
  Wait(500)
  DO(cameraSignal_out,OFF)
  Sync()

  --place part if good
  if DI(cameraSignal_in)==OFF then --put your camera's "good part" digital input here
    MovJ(placePoint)
    outputPalletIter()
    placePart()
  else
    MovJ(dumpPoint)
    placePart()
  end
end

--clearpoint
Go(clearPoint)
```

global.lua

```
-- Version: Lua 5.4.1

function pickPart()
  local Offset = {0, 0, -10, 0}
  local Option={CP=1, SpeedL=50, AccL=20}
  RelMovL(Offset, Option)
  Sync()
  DO(gripperSignal_out, ON)
  Wait(500)
  Offset = {0, 0, 10, 0}
  RelMovL(Offset, Option)
  Sync()
```

```
end

function placePart()
  local Offset = {0, 0, -10, 0}
  local Option={CP=1, SpeedL=50, AccL=20}
  RelMovL(Offset, Option)
  Sync()
  DO(gripperSignal_out, OFF)
  Wait(500)
  Option={CP=1, SpeedL=100, AccL=75}
  Offset = {0, 0, 10, 0}
  RelMovL(Offset, Option)
  Sync()
end

function inputPalletIter()
  --this funciton iterates the input pallet

  if (x_input_iter+1) < x_input_num_iterations then
    --if x row isn't finished, keep 1 incriment in the x
    x_input_iter = x_input_iter + 1
  elseif (y_input_iter+1) < y_input_num_iterations then
    --else if y column isn't finished, x iter set to 0, incriment 1 in the y
    print('GOT HERE')
    x_input_iter = 0
    y_input_iter = y_input_iter + 1
  else
    --else input pallet is done
    print('INPUT PALLET DONE')
    x_input_iter = 0
    y_input_iter = 0
    input_pallet_finished = 1
  end
  pickPoint = {armOrientation="right",
coordinate={x_input_startingPoint+x_input_iter*x_input_pallet,
y_input_startingPoint+y_input_iter*y_input_pallet, z_input_pallet, r_input_pallet,
0.000000, 0.000000},tool=0, user=0}
  Sync()
end

function outputPalletIter()
  --this funciton iterates the output pallet

  if (x_output_iter+1) < x_output_num_iterations then
    --if x row isn't finished, keep 1 incriment in the x
    x_output_iter = x_output_iter + 1
  elseif (y_output_iter+1) < y_output_num_iterations then
    --else if y column isn't finished, x iter set to 0, incriment 1 in the y
    print('GOT HERE')
    x_output_iter = 0
    y_output_iter = y_output_iter + 1
  else
    --else input pallet is done
```

```
    print('OUTPUT PALLET DONE')
    x_output_iter = 0
    y_output_iter = 0
    output_pallet_finished = 1
  end
  placePoint={armOrientation="right",
coordinate={x_output_startingPoint+x_output_iter*x_output_pallet,
y_output_startingPoint+y_output_iter*y_output_pallet, z_output_pallet,
r_output_pallet, 0.000000, 0.000000},tool=0, user=0}
  Sync()
end
```

Troubleshooting & Documents

Dobot Connectivity Guide for MG400

How to connect
MG400 using software

Hardware User Guide

Dobot MG400
Hardware User Guide

DobotStudio2020 Dobot User Guide

DobotStudio2020
User Guide V1.1.1(2020)

MG400 Specs

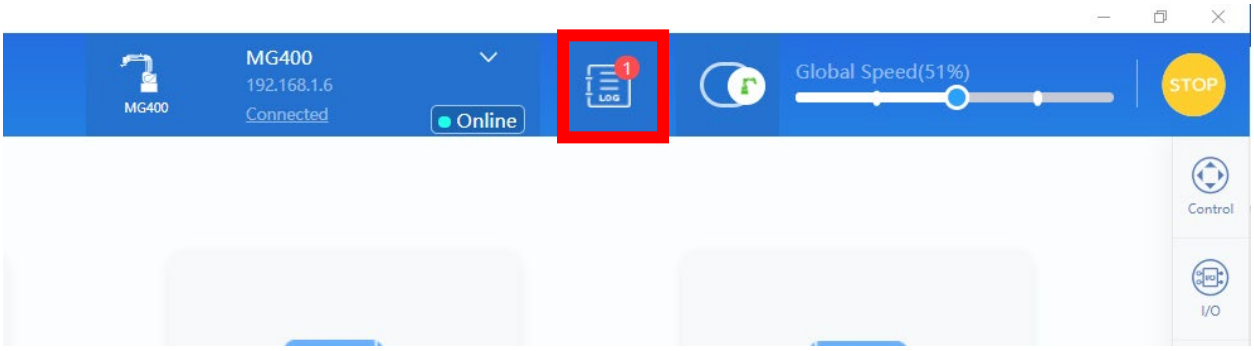
MG400 specs.pdf

Clearing MG400 Errors

The MG400 will error out when the user tries to perform one of many forbidden actions or the robot cannot perform a given task. You can tell the robot is in an error state without a PC because the front LED will be blinking red. Common cases where a user can see an error include:

- Attempting to enable in an invalid position
- A collision was detected
- The robot is violating the parallelogram limits (too close to the base)
- The user attempted to command the robot to hit itself or move outside the max reach

An error is indicated by the log in the top right of DobotStudio2020:



To clear the error, press the log and view what the error was. It may require you to resolve the issue before hitting “clear alarm.”

LED Functionality

The LED on the front right of the MG400 base can glow several colors. This tells us what state the robot is in without needing a computer.

Color	State	Meaning
White	Blinking	Booting
Blue	Solid	Ready, disabled
Blue	Blinking	Unlock button pressed – free to drag robot
Green	Solid	Enabled, Ready
Green	Blinking	Executing Program
Red	Blinking	Error

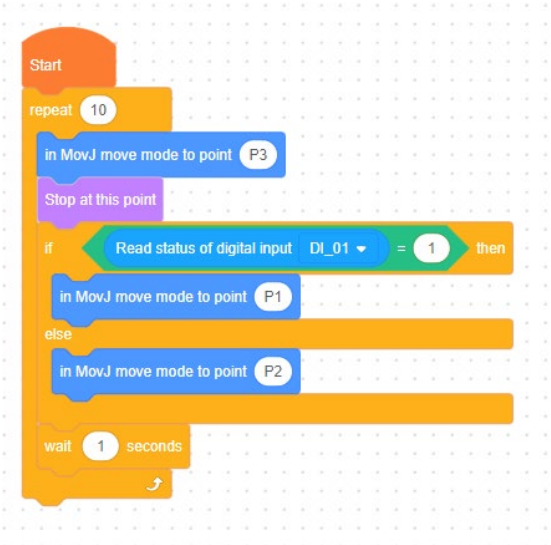
Using Sync with IO Commands

The Sync() function ("Stop at this point" in blockly) is essential for the use of MG400 DIO.

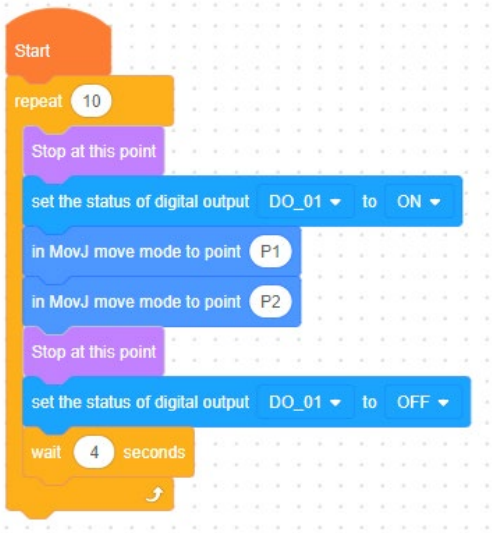
The reason for this is queuing. Similar to other robots, the MG400 compiles all commands very quickly and then executes them in real time. The reading of inputs is generally queued immediately and not in sync with the real-time execution, like we generally want IO to be.

When reading from an IO or writing to an IO, use a Sync() or Stop at this point block before the IO command, like shown below:

Reading from an IO



Writing to an IO



Contacts

Don’t hesitate to reach out for help, questions, and sales. You may reach IP-Tech Dobot support in any of the following ways:

Phone

Customer Service/Sales: 877-478-3241

Email

Help email: help@iptech1.com