

# What is the Inverse Kinematics?

This lesson aims at helping users basically learning about the principle of inverse kinematics. The further learning and practical application of the inverse kinematics of robotic arm is available in the folder “7. Inverse Kinematics Lesson”

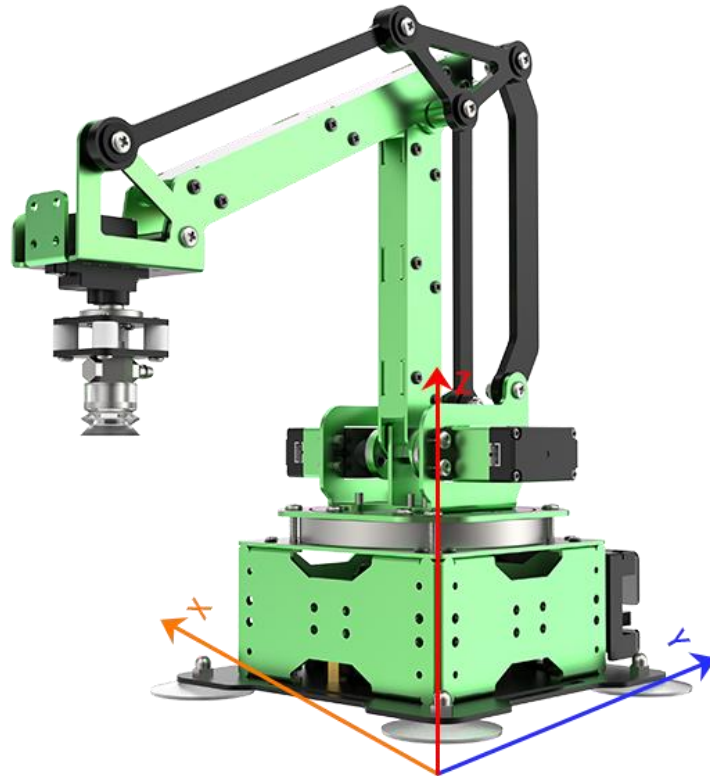
## 1. What is it?

Forward kinematics refers to process of obtaining position and velocity of end effector, given the known joint angles and angular velocities. In other word, the position information of end effector can be obtained when then joint angle and linkages parameters are known.

Inverse Kinematics is the inverse function or algorithm of Forward Kinematics. According to the position and post of the end effector along with linkages parameters, the joint position can be calculated, i.e., Given the robot's end-effector positions, inverse kinematics can determine an appropriate joint configuration.

## 2. Establish Coordinate System

A coordinate system must be established to describe the motion of an object. MaxArm uses x-y-z axes coordinate system (unit:mm) and takes the the base centre of robotic arm as original point (0,0,0), as the figure shown below.



The correspondence relationship between the movement orientation of end effector and the values of x-y-z axes is shown below (user per se as reference):

Coordinate axis	control orientation
x	Control the end effector of robotic arm to move left or right (As the x value is positive, it moves to the right. As the x value is negative, it moves to the left. )
y	Control the end effector of robotic arm to move forward and backward. (As the y value is negative,it moves backward. As the y value is positive, it moves forward.)
z	Control the end effector of robotic arm to move up and down (As the z value is negative, it moves up. As the z value is positive, it moves down.)

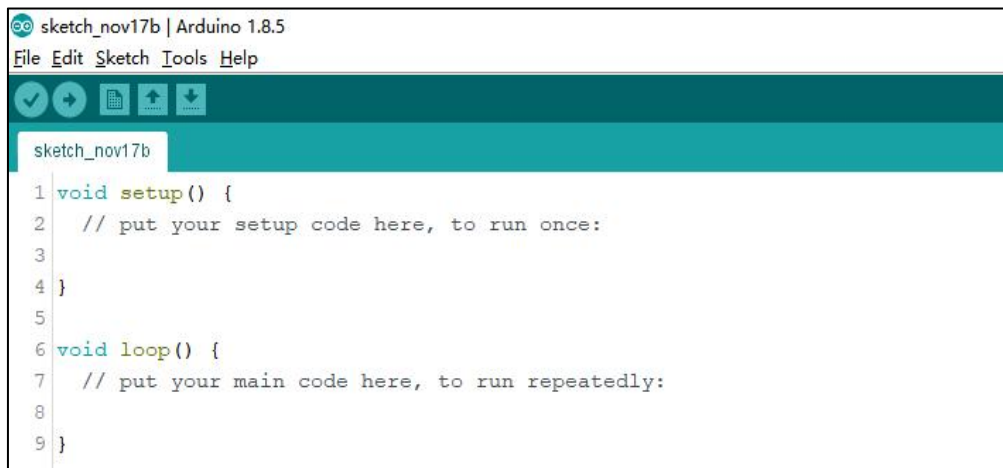
### 3. Project Operation

After learning about the principle and spatial concept of inverse kinematics, the control method of inverse kinematics can be mastered by a simple routine. You can follow the steps below to run the game.

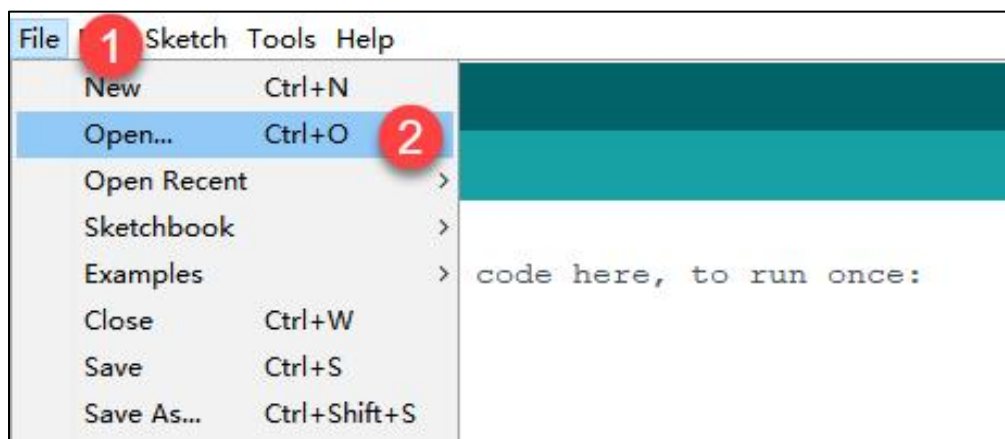
1) Install and connect Arduino. (Please refer to the tutorial in folder “4.

Underlying Files Learning/ Arduino Development/ Lesson 1 Set Development Environment”)

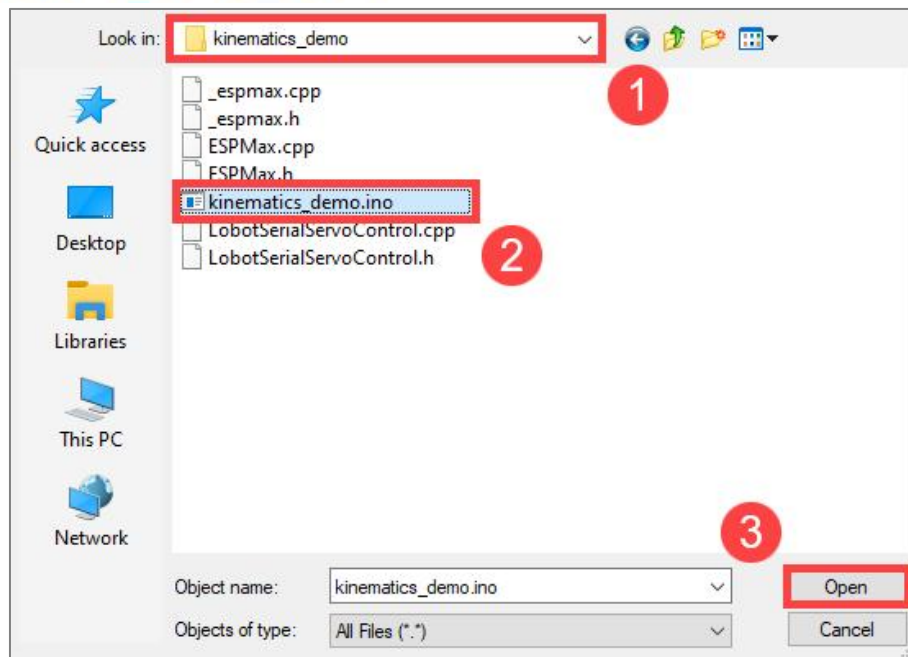
2) Double click to open Arduino IDE



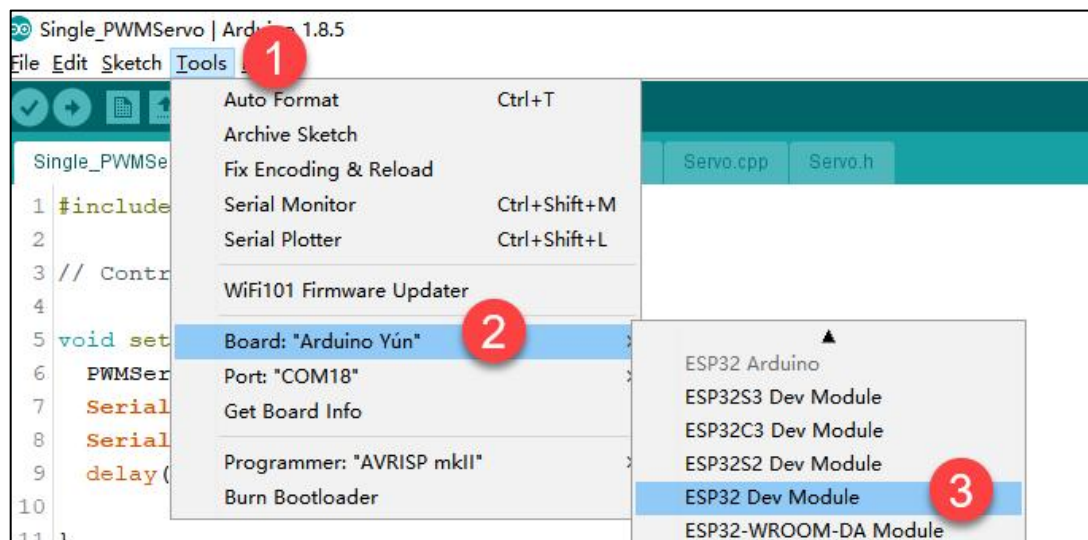
3) Click “File->Open”.



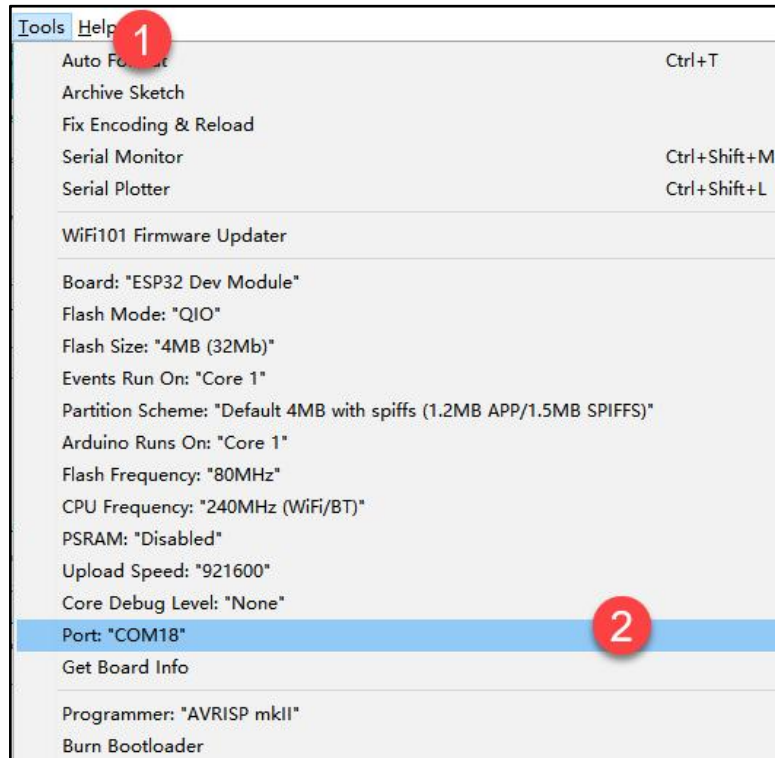
4) Select the program “kinematics\_demo.ino” in the folder “5.MaxArm Hardware Basics Learning/ Arduino Development/ Game Programs/ Program Files/ kinematics\_demo”, and click “Open”.



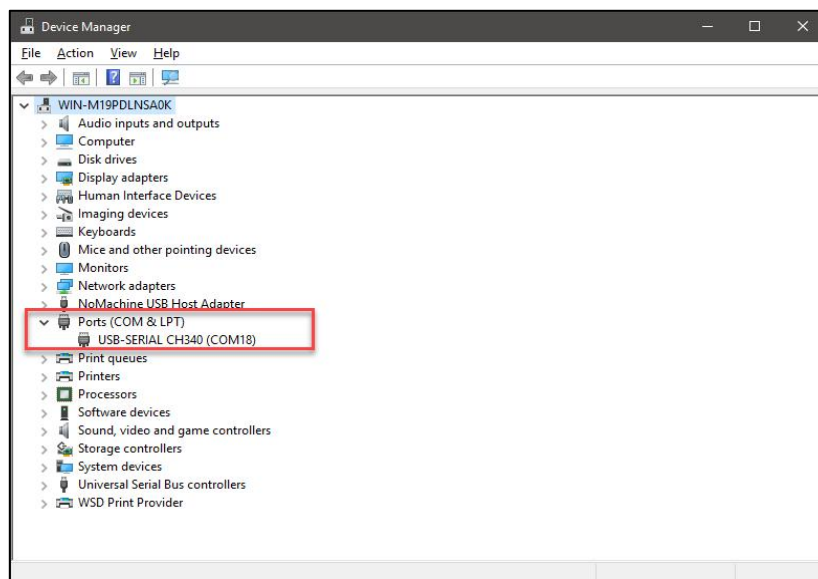
5) Check the board model. Click “Tools->Board” and select “ESP 32 Dev Module”. (If the model of development board has been configured when setting the development environment, you can skip this step.)



6) Select the corresponding port of ESP32 controller in “Tools->Port”. (Here take the port “COM5” as example. Please select the port based on your computer. If COM1 appears, please do not select because it is the system communication port but not the actual port of the development port.)




7) If you are not sure the port number, please open the “This PC” and click “Properties->Device Manager” in turns to check the corresponding port number (the device is with CH340).




8) After selecting, confirm the board “ESP32 Dev Module” in the lower right corner and the port number “COM5” (it is an example here, please refer to the actual situation).

ESP32 Dev Module, Disabled, Default 4MB with spiiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, Core 1, Core 1, None on COM18

9) Then click on  icon to verify the program. If no error, the status area will display “Compiling->Compile complete” in turn. After compiling, the information such as the current used bytes, and occupied program storage space will be displayed.

Done compiling.  
Sketch uses 247733 bytes (18%) of program storage space. Maximum is 1310720 bytes.  
Global variables use 16584 bytes (5%) of dynamic memory, leaving 311096 bytes for local variables. Maximum is 327680 bytes.

10) After compiling, click on  icon to upload the program to the development board. The status area will display “Compiling->Uploading->Complete” in turn. After uploading, the status area will stop printing the uploading information.

Done uploading.  
Leaving...  
Hard resetting via RTS pin... 175%

## 4. Project Analysis

The complete program is as follow:

```

1 #include "ESPMMax.h"
2 #include "_espmax.h"
3
4
5
6 void setup(){
7     ESPMax_init();
8     go_home(2000);
9     Serial.begin(9600);
10    Serial.println("start...");
11 }
12
13 bool start_en = true;
14 void loop(){
15     if(start_en){
16         float x,y,z;
17         float pos[3];
18         // The xyz position of the robot arm at the initial position
19         x = 0;
20         y = -(L1 + L3 + L4);
21         z = (L0 + L2);
22         // The serial port prints the xyz position and the unit is mm

```

```

23     Serial.print(x);
24     Serial.print(" ");
25     Serial.print(y);
26     Serial.print(" ");
27     Serial.println(z);
28
29     // The initial position of robot arm is at the edge of the movement space, so robot arm needs to move down first. Otherwise, it can not move on x and y axes
30     // set_position(pos,t), pos=(x,y,z);
31     //x: the x-axis coordinate; y: the y-axis coordinate; z: the z-axis coordinate; t: the mvoment time (The longer the time, the slower the speed)
32
33     pos[0] = x; pos[1] = y; pos[2] = z-100;
34     set_position(pos,2000); // Robot arm moves down to 100mm
35     delay(2000);
36     pos[0] = x; pos[1] = y; pos[2] = z;
37     set_position(pos,2000); // Robot arm returns to the initial posture
38     delay(1000);
39
40     start_en = false;
41 }
42 else{
43     delay(500);
44 }
45 }

```

## 4.1 Import function library

Before the robotic arm starts to move, the encapsulation library and underlying library of inverse kinematics need to be imported.

```

1 #include "ESPMMax.h"
2 #include "_espmax.h"

```

## 4.2 Calculate the initial position of robotic arm

According to the linkage parameters of L0-L4 defined in kinematics underlying library.



```
kinematics_demo$ ESPMax.cpp ESPMax.h LobotSerialServoControl.cpp LobotSerialServoControl.h _espmax.cpp _espmax.h
1 #ifndef _ESPMAX_H
2 #define _ESPMAX_H
3
4 #define L0      84.4
5 #define L1      8.14
6 #define L2     128.4
7 #define L3     138.0
8 #define L4      16.8
```

Calculate the initial position of the end effector. (Use the L0-L4 values to get  $x=0$ ,  $y=162.94$ ,  $z=212.8$ )

```
18 // The xyz position of the robot arm at the initial position
19 x = 0;
20 y = -(L1 + L3 + L4);
21 z = (L0 + L2);
```

### 4.3 Control robotic arm

Use the function `set_position()` to control the end effector to move.

Take the code “`set_position(pos,2000)`” as example.

```
33 pos[0] = x; pos[1] = y; pos[2] = z-100;
34 set_position(pos,2000); // Robot arm moves down to 100mm
35 delay(2000);
36 pos[0] = x; pos[1] = y; pos[2] = z;
37 set_position(pos,2000); // Robot arm returns to the initial posture
38 delay(1000);
```

The first parameter “pos” is a set of values representing the position values of the end effector on x-y-z axis.

Among them, “pos[0]” represents the x-axis value of the initial position of the end-effector.

“pos[1]” represents the the y-axis value of the initial position of the end-effector.

“pos[2]” represents the end-effector moves down to 100mm. And the position of the end effector can be set by modifying the x,y and z values.

For example, if want to control the end-effector to move 200mm to the left. (its position relative to the original moves to 200mm to the left), set x value plus 200. If want to move to 200 to the right, set x-200.

If want to directly move to the set position, for example, move to 200mm on x axis, you just need to set  $x=200$ .



The second parameter "2000" is the running time and the unit is ms.

## 5. Inverse kinematics library analysis

The path to the inverse kinematics library: Appendix/8. Controller Underlying Files/ Arduino Development/ espmax.py

此电脑 > DATA (D:) > MaxArm > 附录 > 8.主板底层文件 > Arduino开发			
名称	修改日期	类型	大小
_espmax.cpp	2022/9/5 16:37	CPP 文件	4 KB
_espmax.h	2022/3/15 14:58	H 文件	1 KB
Buzzer.cpp	2022/3/15 14:58	CPP 文件	1 KB
Buzzer.h	2022/3/15 14:58	H 文件	1 KB
ESP32PWMServo.cpp	2022/9/6 12:08	CPP 文件	2 KB
ESP32PWMServo.h	2022/9/6 12:11	H 文件	1 KB
<b>ESPMax.cpp</b>	2022/9/12 20:30	CPP 文件	3 KB
ESPMax.h	2022/3/15 16:03	H 文件	1 KB
LobotSerialServoControl.cpp	2022/9/12 20:21	CPP 文件	9 KB
LobotSerialServoControl.h	2022/9/1 17:27	H 文件	3 KB
Servo.cpp	2022/3/24 18:59	CPP 文件	3 KB
Servo.h	2022/3/24 19:06	H 文件	2 KB

### 5.1 Import head file and define pin

```

1  #include "ESPMax.h"
2  #include "_espmax.h"
3  #include "LobotSerialServoControl.h"

```

Import the inverse kinematics and servo head files.

```

5  #define SERVO_SERIAL_RX    35
6  #define SERVO_SERIAL_TX    12
7  #define receiveEnablePin  13
8  #define transmitEnablePin 14
9  HardwareSerial HardwareSerial(2);
10 LobotSerialServoControl BusServo(HardwareSerial, receiveEnablePin,
    transmitEnablePin);

```

Define the serial communication pin.

## 5.2 Initialization

```
12 float ORIGIN[3] = { 0, -(L1 + L3 + L4), (L0 + L2) };
13 float positions[3];
14
15 void ESPMax_init() {
16     BusServo.OnInit();
17     HardwareSerial.begin(115200, SERIAL_8N1, SERVO_SERIAL_RX,
18                          SERVO_SERIAL_TX);
19 }
```

ORIGIN[3] is the initial position of the end effector calculating from the linkage length.

The function ESPMax\_init() is used for initialization

The function BusServo.OnInit() is the servo configuration initialization.

HardwareSerial.begin() is serial communication configuration. "115200" is the baud rate. "SERIAL\_8N1" refers to the working mode. "SERVO\_SERIAL\_RX" is the pin number of RX port. "SERVO\_SERIAL\_TX" is the pin number of TX port.

## 5.3 Control a single servo

```
20 int set_servo_in_range(int servo_id, int p, int duration) {
21     if(servo_id == 3 & p < 470) p = 470;
22     if(servo_id == 2 & p > 700) p = 700;
23     BusServo.LobotSerialServoMove(servo_id, p, duration);
24     return int(1);
25 }
```

The function set\_servo\_in\_range() is used to control the movement of a single servo, and limit the position of servo ID2 and ID3. The ID3 Servo can not be less than 470 impulse and the No.4 can not be less than 700. The parameter "servo\_id" is servo ID number and the parameter "p" is servo impulse. The parameter "duration" is the running time.

The function BusServo.LobotSerialServoMove() in BusServo library controls a single servo to move.

## 5.4 Calculate servo pulse

```
27 float* position_to_pulses(float pos[3], float* pul){
28     float angles[3];
29     inverse(pos,angles);
30     deg_to_pulse(angles,pul);
31     return pul;
32 }
```

The function `position_to_pulses()` is used to calculate the servo pulse. The parameter “position” is position coordinate. “angles” is servo angle. “pulse” is servo pulse. Then the value of servo pulse will be returned.

The function `inverse()` is used to calculate the servo angle according to the coordinates.

The function `deg_to_pulse()` is used to calculate the servo pulse according to the servo angle.

## 5.5 Calculate robotic arm position

```
34 float* pulses_to_position(float pul[3], float* pos){
35     float joints[3];
36     pulse_to_deg(pul,joints);
37     forward(joints,pos);
38     return pos;
39 }
```

The function `pulses_to_position()` is used to calculate the coordinate of robotic arm position. The parameter “pul[3]” is servo pulse. The coordinate of robotic arm is calculated according to the servo pulses, and then the coordinate value is returned.

## 5.6 The movement of robotic arm

```
41 int set_position(float pos[3], int duration){
42     float x = pos[0];
43     float y = pos[1];
44     float z = pos[2];
45     if(z > 255) z = 255;
46     if(sqrt(x*x + y*y) < 50) return int(0);
47     float angles[3];
48     inverse(pos,angles);
49     float pul[3];
50     deg_to_pulse(angles,pul);
51     for(int i=0; i<3; i++){
52         positions[i] = pul[i];
53         BusServo.LobotSerialServoMove(i+1,pul[i],duration);
54         delay(2);
55     }
56     return int(1);
57 }
```

The function `set_position()` is used to control the robotic arm to move, and add the position limit. The parameter “position” is the position coordinate and the “duration” is the running time.

Use judgement statement to limit the robotic arm position. The coordinate of z-axis can not be greater than 225. The root of the sum of the squares of the x and y axes coordinates should be greater than 50, which means the end effector should be outside the circle with the coordinate origin as the center and the radius of 50. The unit is millimeter.

Then use “for” to control the ID1, ID2 and ID3 servos to rotate.

## 5.7 Back to the initial position

```
83 void go_home(int duration){
84     set_position(ORIGIN, duration);
85 }
```

The function `go_home()` is used to get robotic arm back to the initial position.

The parameter “duration” is the running time and the parameter `self.set_position()` is to control robotic arm to move. The parameter “ORIGIN” is the coordinate of the initial position set in program.

## 5.8 Read position coordinate

```
93 float* read_position(float* pos){  
94     float pul[3];  
95     for(int i=0; i<3; i++){  
96         pul[i] = BusServo.LobotSerialServoReadPosition(i+1);  
97     }  
98     pulses_to_position(pul,pos);  
99 }
```

Use the function `self.bus_servo.get_position()` to get the pulse value of servo ID1, ID2, and ID3.

Get the x,y,z position coordinate by calculating the robotic arm position function `pulses_to_position()`. Then the x, y and z values are obtained.