Contents

1 Overview 1
2 Core Code 1
3 Servo_Service Function 3
4 Streamline Programming 4
5 Thank You for Support! 5

1 Overview

This article is to help users better understand the code for the **SunFounder Crawling Quadruped Robot Kit for Arduino**. In this article, the core code of the sketch, manipulator model of each leg, and proof of the model as well as the corresponding code for the proof will be presented in detail. When you've thoroughly understood these, you can write your own code for the robot! For example, you may write a sketch to make the robot swing the legs when walking, or sway a bit, walk in a bigger pace, dance more steps, etc. Sound amazing? Get started to learn and you can make it!

2 Core Code

This article focuses on how to transform the coordinates of the end of each leg into the rotational angle of each servo. First check the functions void cartesian_to_polar(volatile float &alpha, volatile float &beta, volatile float &gamma, volatile float x, volatile float y, and volatile float z). These are the core of the code for the quadruped robot, which is to transform the coordinates of the legs into the servo rotational angles.

Parameters: alpha, beta, gamma, the address that stores the output angle.

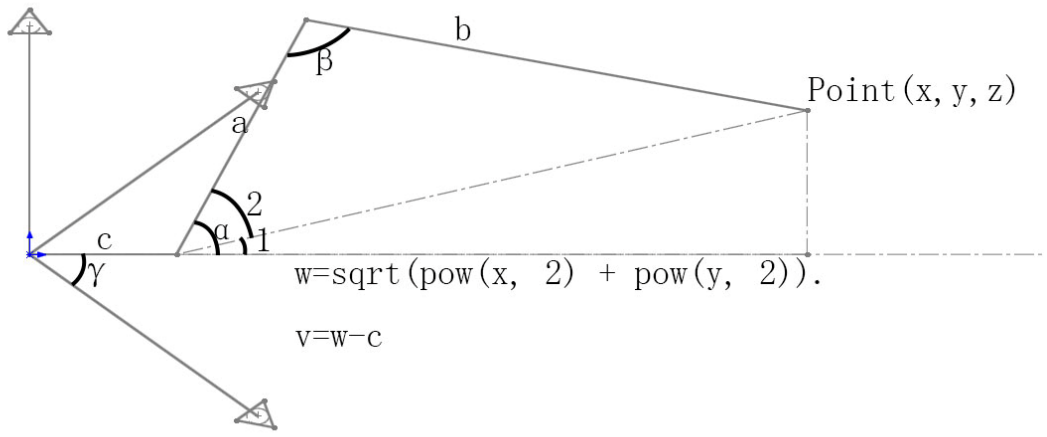
Parameters: x, y, z, the coordinates of the position of the leg end.

The source code of cartesian_to_polar is as follows:

```
/*
 - trans site from cartesian to polar
```

```
 - mathematical model 2/2
 * ---------------------------------------------------------------------------*/
void cartesian_to_polar(volatile float &alpha, volatile float &beta, volatile float
        &gamma, volatile float x, volatile float y, volatile float z)
{
        //calculate w-z degree
        float v, w;
        w = (x >= 0 ? 1 : -1)*(sqrt(pow(x, 2) + pow(y, 2)));
        v = w - length_c;
        alpha = atan2(z, v) + acos((pow(length_a, 2) - pow(length_b, 2) +
        pow(v, 2) + pow(z, 2)) / 2 / length_a / sqrt(pow(v, 2) + pow(z, 2)));
       beta = acos((pow(length_a, 2) + pow(length_b, 2) - pow(v, 2) - pow(z,
        2)) / 2 / length_a / length_b);
        //calculate x-y-z degree
        gamma = (w >= 0) ? atan2(y, x) : atan2(-y, -x);
        //trans degree pi->180
        alpha = alpha / pi * 180;
        beta = beta / pi * 180;
        gamma = gamma / pi * 180;
}
```

First build a 3D model for a certain leg. The coordinate direction should be consistent with that on the calibration chart, as shown below:



Here we'll only analyze the first quadrant of the leg end: given the end position Point (x,y,z) and segment a, b, c (the length of each segment of the leg), to calculate the rotational angle of the servo α, β, γ. Within, π/2≤α≤π/2, 0≤β

≤π, -π/2≤γ≤π/2. In this way, transform these into a basic mathematic model. The proof of the model:

w= $\sqrt{x^2 + y^2}$.

V=w-c.

With the law of cosines, cos a = $\frac{b^2+c^2-a^2}{2*b*c}$, the result of ∠2 can be calculated.

∠2=arc cos $\frac{a^2+(z^2+v^2)-b^2}{2*a*\sqrt{z+v^2}.}$

∴ ∠α=∠1+∠2=arc tan(z/v)+ arc cos $\frac{a^2+(z^2+v^2)-b^2}{2*a*\sqrt{z+v^2}.}$。

The program should be:

```
alpha = atan2(z, v) + acos((pow(length_a, 2) - pow(length_b, 2) + pow(v, 2) + pow(z, 2)) / 2 / length_a / sqrt(pow(v, 2) + pow(z, 2)));
```

Similarly, ∠β= arc cos $\frac{a^2+b^2-(z^2+v^2)}{2*a*b}$.

The program should be:

```
beta = acos((pow(length_a, 2) + pow(length_b, 2) - pow(v, 2) - pow(z, 2)) / 2 / length_a / length_b);
```

Similarly, ∠γ=arc tan (y/x).

The program should be (here only analyze the case for the leg end in the first quadrant):

```
gamma = (w >= 0) ? atan2(y, x) : atan2(-y, -x);
```

Hereto all the transformation from coordinates of the leg end into the servo rotational angle is done.
Each leg has its own coordinate system, which is calculated independently.

3 Servo_Service Function

After the function cartesian_to_polar is done in the sketch, immediately call the function void polar_to_servo(int leg, float alpha, float beta, float gamma) to adjust the servo rotational angle to the set angle. These two functions will be called one by one in the 50HZ service function void servo_service(void). It is a critical function and you need to pay much attention here.

4 Streamline Programming

After you've understood the core code and the working sequence, review the code:

```
/* Installation and Adjustment ---------------------------------------------*/
#define INSTALL     //uncomment only this to install the robot
//#define ADJUST    //uncomment only this to adjust the servos
//#define VERIFY    //uncomment only this to verify the adjustment
```

Activate the INSTALL command line and then add a for() loop in setup.

```
void setup()
{
#ifdef INSTALL
    //initialize all servos
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            servo[i][j].attach(servo_pin[i][j]);
            delay(100);
        }
    }
    while (1);
```

Here set the shaft of the each servo in the center position so as to minimize the error during the installation. After servos are installed, run the calibration program to check whether all the servo are in the center position. Activate ADJUST line and start the calibration:

```
/* Installation and Adjustment ---------------------------------------------*/
//#define INSTALL   //uncomment only this to install the robot
#define ADJUST      //uncomment only this to adjust the servos
//#define VERIFY    //uncomment only this to verify the adjustment
```

The program still waits in the loop in setup. Set a set of calibration coordinates manually. Then obtain the real coordinates via the calibration chart provided in the kit and a ruler (also an acrylic one included), and then modify the default real coordinates in the sketch.

```
const float real_site[4][3] = { { 115, 68, 42 }, { 105, 66, 60 },
{ 92, 70, 56 }, { 92, 70, 56 } };
```

Activate VERIFY and store the coordinates just obtained. Calculate the error and add it every time the servo rotates, so the accuracy of each segment moving can be ensured.

When all the calibration above mentioned is done, comment the three lines under Installation and Adjustment. After initialization, enter the loop. Here the servo service program runs in the frequency of 50Hz.

During this period, the main function waits for the remote control commands, so the robot moves accordingly under different command, while the service function is executed all the time, constantly determines whether there is a new target position, and drives the servo to rotate to the position by the functions cartesian_to_polar and polar_to_servo. Thus, when you push the joystick of the remote control, the corresponding command sent can be executed.

5 Thank You for Support!

After all the reading, you may hopefully be able to solve the problem encountered in coding and gain a lot from the kit. If you have any questions, welcome to post in Forum section on our website www.sunfounder.com!

Thanks again for supporting SunFounder!