浙江大学

六自由度机械手控制及末端装置设计

课程：机电系统实验

班级：机械工程 1601

学号：3160101483

姓名：范耀威

指导教师：高宇

日期：2020 年 1 月 6 日

目录

一、任务要求

任务的主要内容可以概括为：控制机械手将工件从放置区搬运到检测区并检测工件是否合格。需要完成的工作包括：

1）设计合适的工位以放置工件和固定检测板；

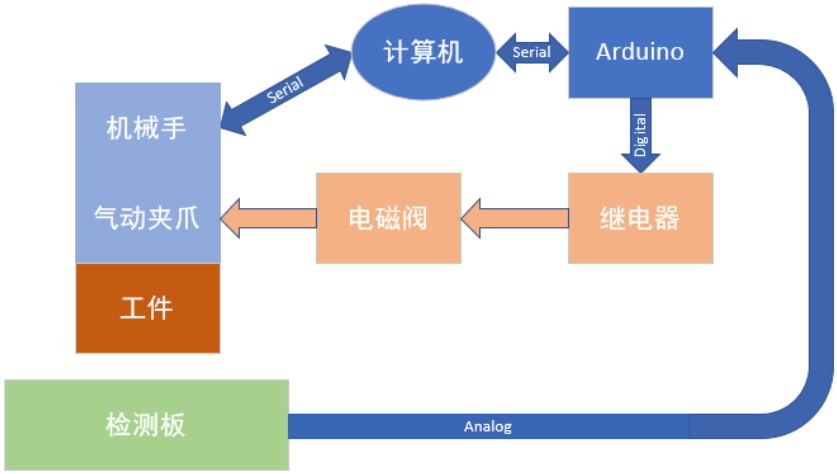
2）选择和设计合理的机械手末端抓手并搭建控制电路；

3）实现机械手的驱动，并完成工件的检测：从放置工位夹取工件并搬运至检测区，将工件转动一个角度后检测一次，多次转动和检测后得到一系列检测数据。

二、方案设计

1 系统架构

为使系统具有较高的可靠性和拓展性，我们采用图所示的架构。该架构中，计算机作为主控端，通过串口向机械手和 Arduino 发送指令并接收反馈数据，实现机械手、气动夹手和检测板的交替控制。



系统架构

2 软件平台

ROS（Robot Operating System，下文简称“ROS”）是一个适用于机器人的开源的元操作系统。它提供了操作系统应有的服务，包括硬件抽象，底层设备控制，常用函数的实现，进程间消息传递，以及包管理。它也提供用于获取、编译、编写、和跨计算机运行代码所需的工具和库函数。

ROS 的主要目标是为机器人研究和开发提供代码复用的支持。ROS 是一个分布式的

进程（也就是“节点”）框架，这些进程被封装在易于被分享和发布的程序包和功能包中。ROS 也支持一种类似于代码储存库的联合系统，这个系统也可以实现工程的协作及发布。这个设计可以使一个工程的开发和实现从文件系统到用户接口完全独立决策（不受 ROS 限制）。同时，所有的工程都可以被 ROS 的基础工具整合在一起。

基于以上优点，我们考虑采用 ROS 架构来实现目标任务。

3 节点布置

为实现相对稳定的分布式控制，需要合理安排 ROS 节点。

在 PC 机上，除了需要开启 ROS Master 节点之外，还需要运行一个负责与机械手进行通讯并管理任务流程的节点，在 Arduino 上则需要运行一个负责 IO 输入输出任务的节点，节点之间互相通讯。

4 机械手驱动

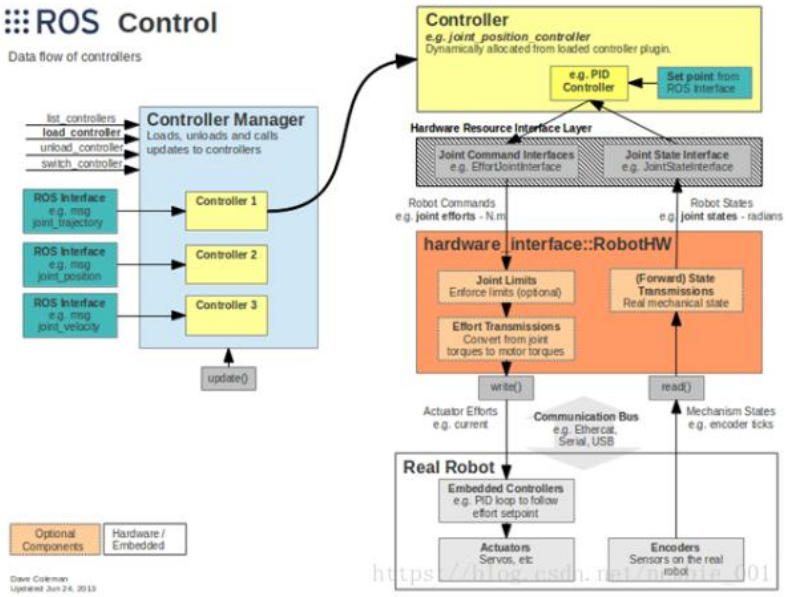
方案中最重要的部分是实现机械手的驱动。原定的驱动方案为：借助 Moveit！平台实现机械手末端的轨迹规划和控制。

在了解 Moveit！平台和 ROS 的控制结构后，我们发现驱动机械手需要做以下工作：

1）建立带气动夹手的机械手 SolidWorks 模型，由该模型建立相应的 URDF 文件；

2）编写 write()和 read()函数，实现机械手的硬件抽象；

3）借助 Moveit！接口实现机械手末端的轨迹规划和控制。



ROS 控制结构

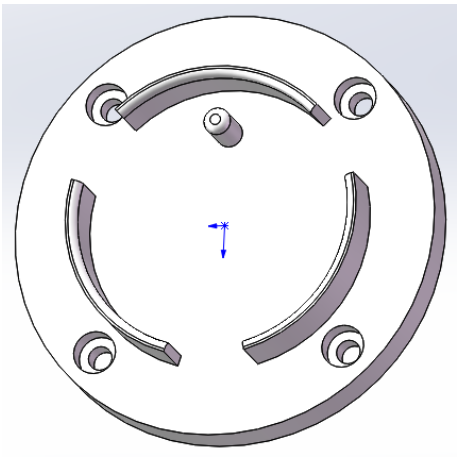
考虑到以上工作的复杂性，我们跳过了 Moveit！平台，采用简单的串口通信来达到驱动机械手的目的，大大减少了工作量。

三、实现细节

1 硬件实现

1.1 工件放置工位

采用 3D 打印的方式制作工件放置工位。



工件放置工位 3D 模型

工件放置工位实物

1.2 检测工位

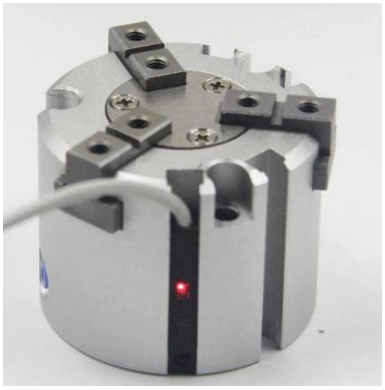
检测工位主要由检测板组成，通过长铜柱固定在工作台上，见工作台图。

1.3 气动夹手

1.3.1 气动夹手选型与设计

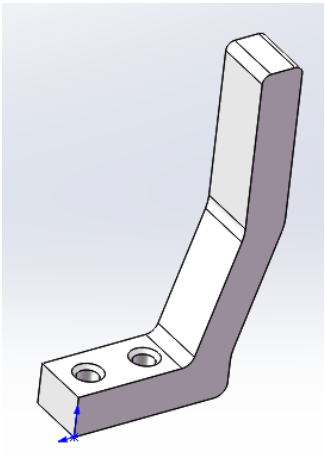
考虑到工件需要被牢牢夹住、工件的中心轴需要与第六轴重合，且夹手的整体体积不宜过大，我们采用以下方案。

1）我们选定执行机构为气动三爪卡盘。结合实际工况，我们选用 MHS3 型气动三爪卡盘，其缸径为 20mm。气动卡盘的配套装置有继电器、电磁阀、气管、快插转接头等，需要一同配齐。



气动三爪卡盘

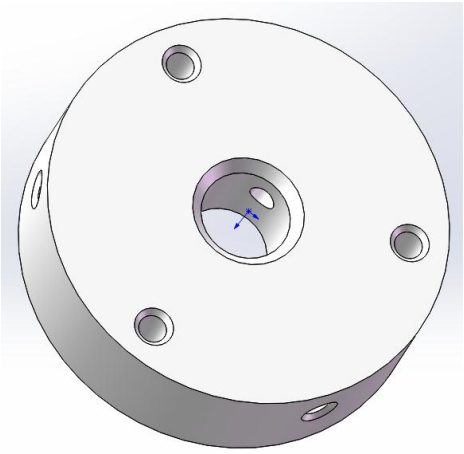
2）每条爪指内侧面设计成与工件等径的圆弧面，爪指 3D 打印成形。



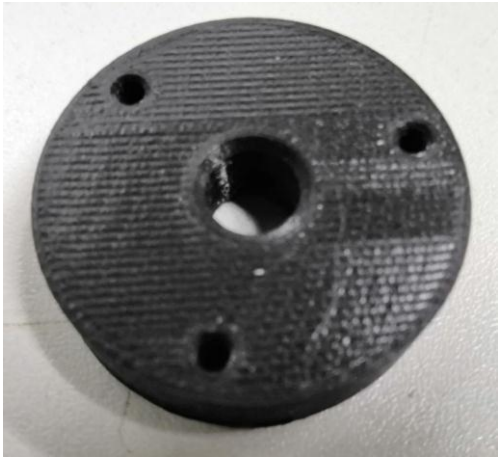
爪指 3D 模型

爪指实物

3）三爪卡盘联接件 3D 打印成形。

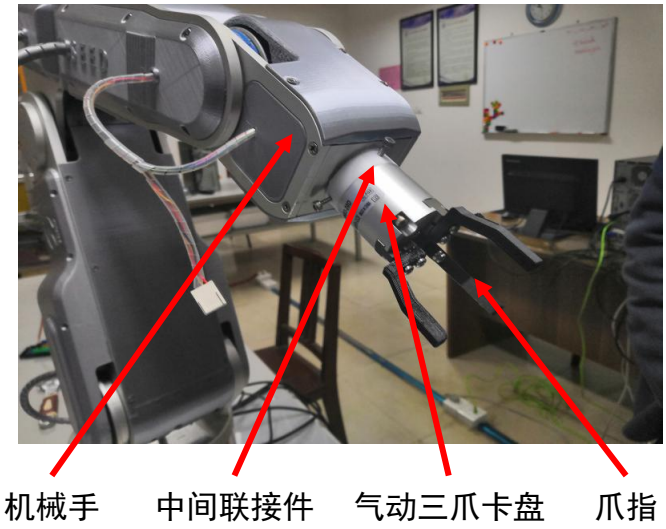


三爪卡盘联接件 3D 模型



三爪卡盘联接件实物

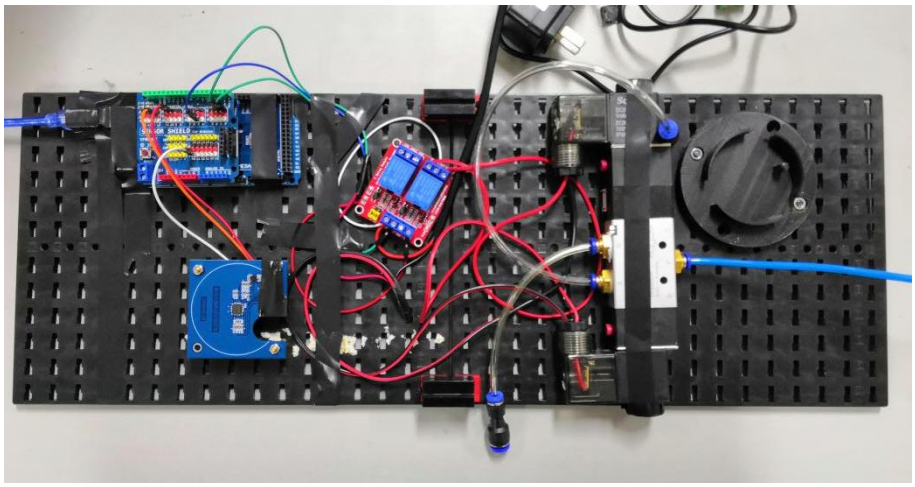
1.3.2 整体装配效果



整体装配效果图

1.4 工作台

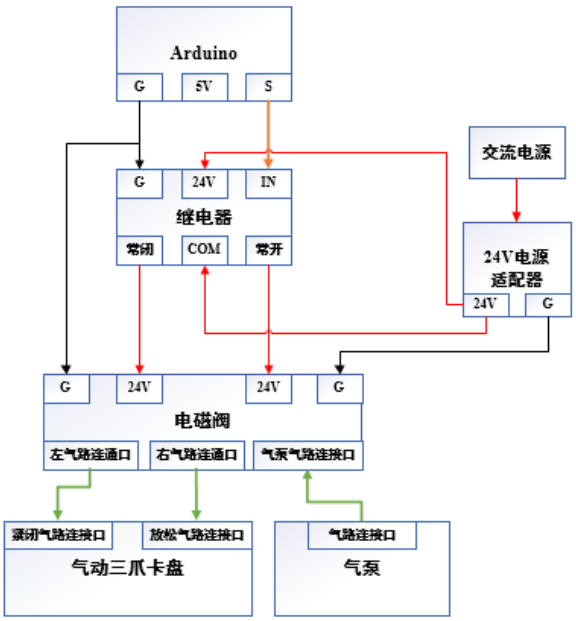
由于机械手所在的桌面没有任何可用于定位的结构，所以需要另外设计整体定位基台使得工件放置工位和检测工位相对于机械手基座的位置确定。我们用慧鱼组件搭建了这样一个工作台。



工作台

1.5 电路、气路连接

电路和气路连接如下图所示，涉及到的器件有 Arduino Mega2560、继电器、气动三爪卡盘、电磁阀、气泵、电源适配器等。



电路、气路连接图

2 软件实现

软件具体实现分为两个部分，一个是 PC 端的节点，负责与机械手进行通讯并管理任务流程，另一个是 Arduino 端的节点，负责管理 IO。

2.1 PC 端节点

2.1.1 机械手控制

机械手采用串口控制，用到了 ROS 的 serial 功能包。主要涉及到一条运动指令——轴角度插补指令，和一种状态反馈信息——坐标值状态信息。

轴角度插补指令：

a[0]	a[1]	a[2]	a[3] –a[6]	a[7] –a[10]	a[11] –a[14]	a[15] –a[18]	a[19] –a[22]	a[23] –a[26]	a[27] –a[30]	a[39] –a[42]	a[43] –a[46]	a[47]
0xee	'3'	0–6	a0	a1	a2	w0	w1	aw	PWM	N/A	speed	0xef
帧头	指令 a	指令 b	浮点数 1 mm	浮点数 2 mm	浮点数 3 mm	浮点数 4 度	浮点数 5 度	浮点数 6 度	浮点数 7 1us/步	浮点数 10 0	浮点数 11 度/秒	帧尾
a[31]–a[34]是外部轴 E0 角度值，a[35]–a[38]是外部轴 E1 角度值												

坐标值状态信息：

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
0xce	x_h	x_l	y_h	y_l	z_h	z_l	0	0xcf
帧头	d0	d1	d2	d3	d4	d5	指令标志	帧尾

PC 机向机械手发送轴角度插补指令，该指令包含了通过示教得到的目标点位置的关节坐标信息 a_0、a_1、a_2、w_1、w_2、aw 和转动角速度信息 speed。机械手接收到运动指令后，结合当前关节坐标，插补得到运动轨迹，并沿着这条轨迹运动到目标位置。在机械手运动过程中，PC 机等待，并不断读取机械手反馈的坐标值状态信息，从中提取机械手当前的坐标位置，当获得的坐标值等于目标坐标值时，PC 机停止等待，进行下一个任务。

结构体 axis_angle_msg 包含了轴角度插补指令所需要的信息：

```
struct axis_angle_msg
{
    float angle1;
    float angle2;
    float angle3;
    float angle4;
    float angle5;
    float angle6;
    float speed;
    axis_angle_msg();
    axis_angle_msg(float a1,float a2,float a3,float a4,float a5,floa
t a6,float s);
};
```

指令类 order，包含运动指令初始化接口、控制指令初始化接口（本项目中没有用到控制指令）和指令发送接口：

```
class order
{
public:
    void move_order_init(axis_angle_msg msg);
    void control_order_init();
    size_t send(serial::Serial &sp);
private:
    void trans(float angle,uint8_t a[]);
    uint8_t order[48];
};
```

目标点位置的关节空间坐标表示和操作空间坐标表示需要提前通过示教得到。为了防止机械手最后运动到的位置与目标值有误差，设定一个极小的容许误差 T_：

```
bool is_right(position_msg &pos_msg,int pos_choose)
{
    if(pos_choose==1)
        return (abs(pos_msg.x-origin_pos.x)+abs(pos_msg.y-origin_pos
.y)+abs(pos_msg.z-origin_pos.z))<T_;
    if(pos_choose==2)
        return (abs(pos_msg.x-wp_up_pos.x)+abs(pos_msg.y-wp_up_pos.y
)+abs(pos_msg.z-wp_up_pos.z))<T_;
    if(pos_choose==3)
        return (abs(pos_msg.x-wp_down_pos.x)+abs(pos_msg.y-wp_down_p
os.y)+abs(pos_msg.z-wp_down_pos.z))<T_;
    if(pos_choose==4)
        return (abs(pos_msg.x-detect_pos.x)+abs(pos_msg.y-detect_pos
.y)+abs(pos_msg.z-detect_pos.z))<T_;
    return false;
}
```

以下是使机械手运动到原点并等待运动结束的整个过程，其中，wait 函数中的参数“1”代指储存的第一个目标点（原点）：

```
myorder.move_order_init(origin);
myorder.send(sp);
wait(sp,mystate,1);//运动至原点
```

2.1.2 话题的订阅与发布——与 Arduino 进行通讯

PC 与 Arduino 之间的 ROS 通讯依赖于 rosserial_arduino。PC 端节点与 Arduino 端节点通过话题进行消息传递，在 PC 端创建一个话题发布器 chatter_pub 用于发布指令信息 chatter_pub，创建一个话题订阅器 chatter_sub 用于订阅动作完成信息 chatter_sub：

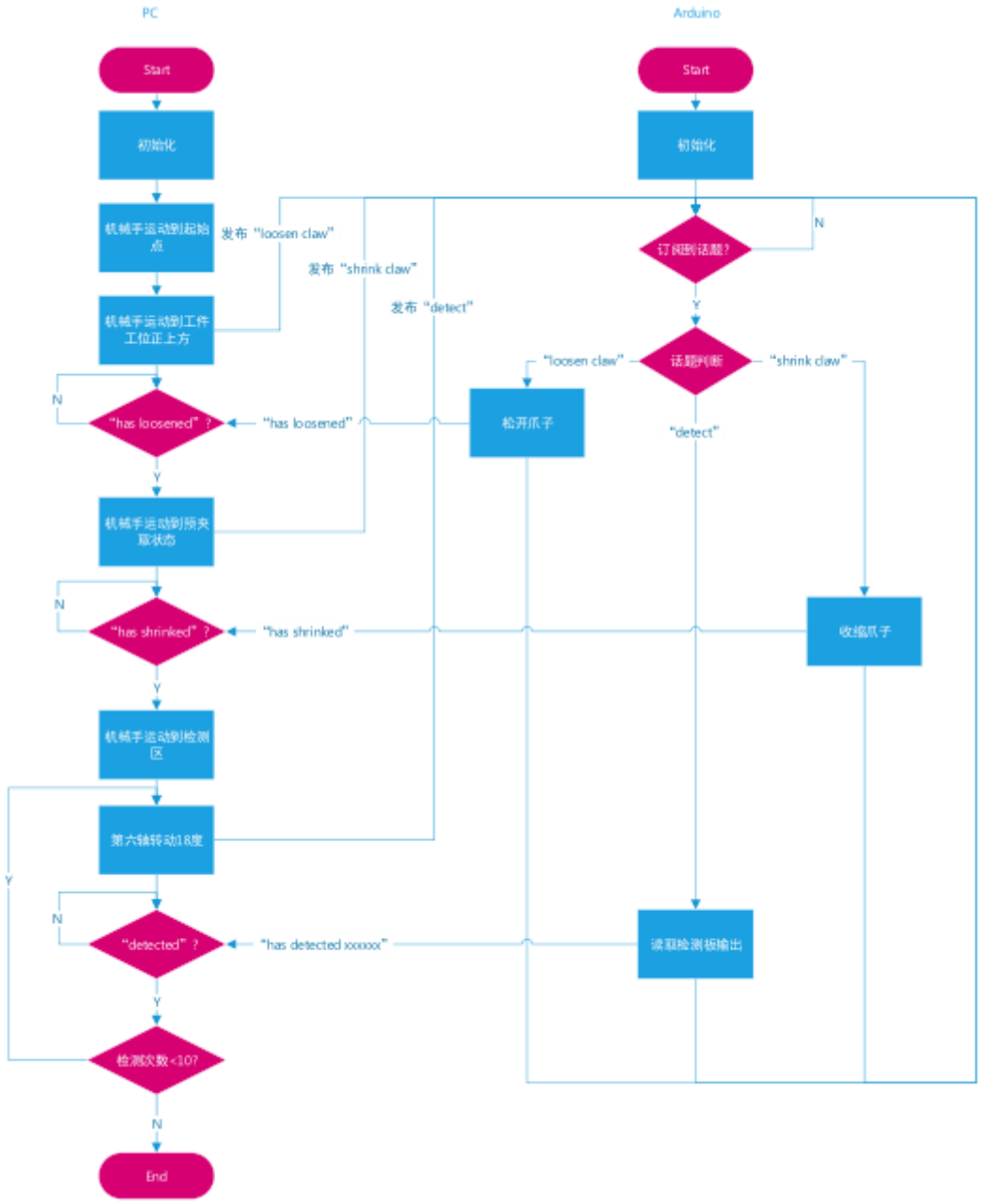
```
    ros::Publisher  chatter_pub = n.advertise<std_msgs::String>("chatter
_pub", 100);
    ros::Subscriber chatter_sub = n.subscribe<std_msgs::String>("chatter
_sub", 100, Callback);
```

发布和订阅的话题内容具体如下：

```
Publish:
    "loosen claw"      ————>        松爪
    "shrink claw"      ————>        收爪
    "detect"           ————>        检测一次
Subscrib:
    "has loosened"             ————>     已松爪
    "has shrinked"             ————>     已收爪
    "has detected xxxxxx"      ————>     已检测一次 + 六位检测值
```

PC 端发布执行任务话题后等待应答，Arduino 端订阅话题后执行相应的任务并产生应答，收到应答后 PC 端进行下一个任务。

2.1.3 任务流程


PC
Arduino
Start
初始化
机械手运动到起始点
发布 "loosen claw"
发布 "shrink claw"
发布 "detect"
机械手运动到工件工位正上方
N
"has loosened" ?
"has loosened"
Y
机械手运动到预夹取状态
N
"has shrinked" ?
"has shrinked"
Y
机械手运动到检测区
第六轴转动18度
Y
N
"detected" ?
"has detected xxxxxx"
Y
检测次数<10?
N
End
Start
初始化
订阅话题?
N
Y
话题判断
"loosen claw"
"shrink claw"
"detect"
松开爪子
收缩爪子
读取检测板输出

2.2 Arduino 端节点

2.2.1 话题的订阅与发布——与 Arduino 进行通讯

Arduino 端节点创建一个话题订阅器 sub 用于订阅 PC 端节点发布的话题 chatter_pub，创建一个话题发布器用于发布 PC 端需要订阅的话题 chatter_sub：

```
ros::Publisher pub("chatter_sub", &str_msg);
ros::Subscriber<std_msgs::String> sub("chatter_pub", messageCb );
```

发布和订阅的话题内容具体如下：

```
Subscrib:
    "loosen claw"      ————>        松爪
    "shrink claw"      ————>        收爪
    "detect"           ————>        检测一次
Publish:
    "has loosened"              ————>    已松爪
    "has shrinked"              ————>    已收爪
    "has detected xxxxxx"       ————>    已检测一次 + 六位检测值
```

2.2.2 气动夹手控制

气动夹手用 Arduino 数字口控制：

```
pinMode(CLAW,OUTPUT);
void loosen_claws(void)
{
  digitalWrite(CLAW,LOW);
  delay(1000);
}

void shrink_claws(void)
{
  digitalWrite(CLAW,HIGH);
  delay(1000);
}
```

2.2.3 检测板读数

检测板读数利用 Arduino 的模拟口，最大读数范围为 0~1023。以下过程为：从对应的模拟口读出检测板读数，并将数据组织成“has detected xxxxxx”的格式：

```
pinMode(DETECT,INPUT);
void detecting(char msg[])
{
  char temp_str[10];
  int digit;
  digit=analogRead(DETECT);
  strcpy(msg,"has detected ");
  sprintf(temp_str,"%-6d",digit);
  for(int i=0;i<7;i++)
    msg[13+i]=temp_str[i];
}
```

四、系统调试

调试工作的主要内容是目标点的示教。主要包括原点、工件放置工位上方点、工件预抓取点和检测点：

```
axis_angle_msg origin     ( -2.90,   30.33,   100.44,   -0.50,   40.
94,       0,   7);
axis_angle_msg wp_up_msg  ( 28.24,   65.55,    108.7,    0.49,   66.
55,   -5.73,   7);
axis_angle_msg wp_down_msg( 28.24,   68.35,   110.39,       0,   67.
24,   -5.74,   7);
axis_angle_msg detect_msg (-23.26,   67.59,   111.11,       0,   67.
01,   -5.74,   7);

position_msg origin_pos  (4467, -155,  2398);//pos_choose:1
position_msg wp_up_pos   (4283, 2277,  109 );//pos_choose:2
position_msg wp_down_pos (4302, 2287,  -55 );//pos_choose:3
position_msg detect_pos  (4584, -1946, 54 );//pos_choose:4
```

五、缺点与不足

项目虽然达到了预期效果，但仍有许多不足之处：

1）尚未结合 Moveit！平台，难以实现复杂的轨迹规划任务。虽然在 ROS 平台上实现了目标任务，但并没有利用该平台上丰富的资源，如可视化工具、轨迹规划工具、仿真工具等等；

2）机械手在复位后的重定位精度不高，而检测时工件与检测板之间需要有一个比较准确的相对位置，否则很难获得可靠数据，这对检测结果影响很大。要解决这个问题一方面可以通过提升机械手的精度或者机械手不停机，另一方面则是考虑加入视觉校正的部分；

3）程序没有处理异常的功能，如果夹取不当或者运动过程中遇到障碍，会造成难以预料的后果。

六、心得感悟

在本项目中，我主要学习了 ROS serial 功能包和 rosserial_arduino，并利用它们完成了一些简单的工作。项目本身的难度不大，在团队四人的密切配合下，我们比较顺利地完成了任务。比较遗憾的是由于时间精力所限，没有结合 Moveit！，在之后的学习过程中，如果有机会一定要动手实现一下。