

机器人移动抓取

——第二组创新实验报告



小组成员 张梁育 曾浩洲 张雯琪
怀谦益 方胡彪

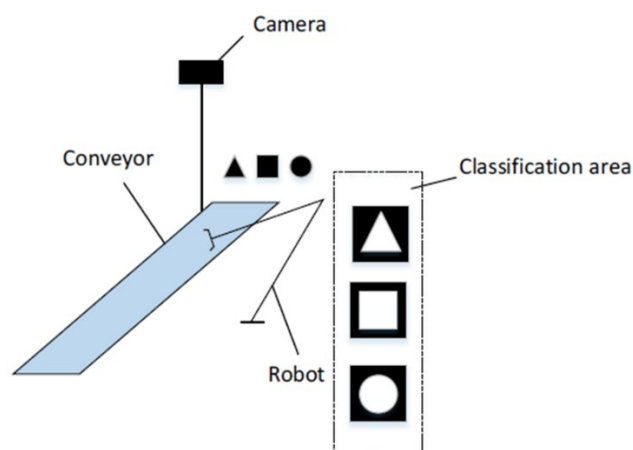
一、 实验目的与实验内容

1.1 实验目的

- 1) 了解学习计算机视觉相关内容，掌握运用形状识别、颜色识别、物体定位等算法。
- 2) 深入掌握机械臂 DH 参数表构件、正逆运动学以及轨迹规划知识。
- 3) 了解学习机械臂末端速度跟踪算法，实现对运动物体的短暂跟踪

1.2 实验内容

机器人移动抓取创新作业要求我们（1）首先通过相机识别不同的木块，（2）然后控制机械臂在木块传送过程中抓取物块并放到指定位置。为此我们小组从识别和抓取两个大方向入手，在识别方面，我们希望通过相机识别物体的颜色或者形状来区分不同物块，同时也要获得物块在传送带上的位置，最后把物块的颜色形状信息和位置信息通过 **topic** 形式发出；在抓取方面，我们希望机械臂在接受到相机发出的 **topic** 之后开始动作，通过速度跟踪算法，让机械臂保持爪子张开，跟踪运动物体一段时间，然后将物体抓起，并且按照物体颜色、形状的不同放置到桌子不同位置上。



二、 实验原理

2.1 物块识别与定位方案

物块识别与定位方案通过计算机视觉系统采集传送带图像信息并适当处理，进行目标检测与实时坐标位置的获取并进行信息传递。

1) 图像预处理

图像预处理通过改善图像数据，淡化图像处理过程中非必要处理或分析的信息以增强分析的特征信息。通过降低镜头、光照、噪声等的影响，使图像处于更适合特征提取的状态。

在本次实验中,通过调用 OpenCV 的 medianBlur() 函数,采用中值滤波来平滑图像。中值滤波是基于排序统计理论中的一种能有效抑制噪声的非线性信号处理技术,中值滤波的基本原理是把数字图像或数字序列中一点的值用该点的一个领域中各点值的中值替代,让周围的像素值接近真实值,从而消除孤立的噪声点。将板内像素按照像素值的大小进行排序,生成单调的二维数据序列。

中值滤波先将所有像素排列并求中值,然后用中值代替中心位置像素,其表达式为:

$$g(x,y) = \text{median} \sum_{(x,y) \in s_{xy}} f(x,y)$$

2) 目标检测:

常用目标特征采集算法有形状检测和颜色检测两类。

形状检测算法通过 OpenCV 中的 findContours() 函数提取轮廓,采用多边形逼近法,通过对轮廓外形无限逼近,删除非关键点、得到轮廓的关键点,不断逼近轮廓真实形状,从而筛选出角点信息,并根据角点的个数来进行形状的判断。

颜色检测算法主要在 HSV 颜色空间中进行,HSV(色调 Hue,饱和度 Saturation,亮度 Value)是根据颜色的直观特性创建的一种颜色空间,也称六角锥体模型。之所以选择 HSV,是因为 H 代表的色调基本上可以确定某种颜色,再结合饱和度和亮度信息判断大于某一个阈值。而 RGB 由三个分量构成,需要判断每种分量的贡献比例。即 HSV 空间的识别的范围更广,更方便。通过调用 OpenCV 中的 inRange() 函数在 hsv 图片中搜索在给定颜色阈值内的像素点,返回一个原图像尺寸的掩膜,对各颜色对应掩膜进行轮廓提取,所含特征点最多的掩膜信息即为目标颜色信息。

在实际操作中,由于本次实验仅考虑单目相机成像系统,若采用基于角点个数的形状识别算法,由于部分形状木块的正视图和侧视图所得角点个数不同,在物块移动过程中会传入错误参数影响识别结果,因此本次实验采用颜色识别算法进行物块识别,通过相机识别指定颜色,并提取该颜色所在轮廓,通过指定形状如矩形来拟合轮廓,获取物块在相机坐标系中的坐标信息。

3) 目标定位:

对目标进行定位需要得到其在图像像素坐标系中的坐标和相机坐标系、机械臂坐标系(即世界坐标系)之间的转换关系。图像像素坐标系与图像固定,以矩阵形式存储,其 X_f, Y_f 平行于相机坐标系的 X、Y 轴,世界坐标系通过旋转平移得到相机坐标系(相机立体标定过程中内外参的转换计算),相机坐标系通过选取合适的相机模型能和图像像素坐标系建立比例关系,从而得到世界坐标下的目标坐标。

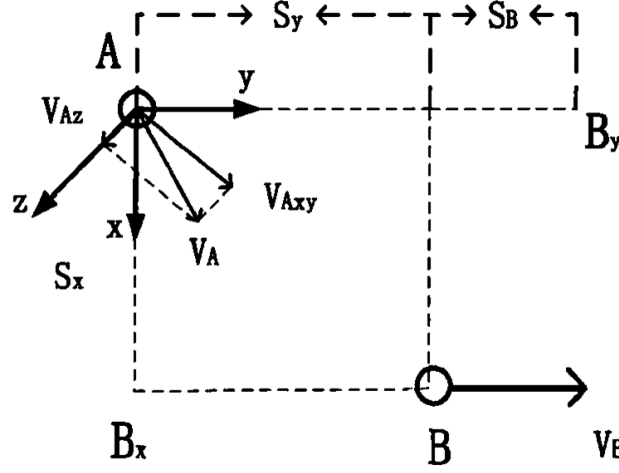
在实际操作中,由于本次实验仅考虑单目相机成像系统,无法较好实现三维立体标定。在相机系统和机械臂系统的坐标系转换中,将相机坐标系视为机械臂坐标系中围绕 Z 轴的二维平面,指定 Y 轴位置,采用线性拟合的方法将相机坐标系中 X 轴的位置信息转换到机械臂坐标系,从而获取物块的实时坐标。

2.2 机械臂末端轨迹跟踪算法

2.2.1 分解速度跟踪算法

为了实现机械臂能够较快地进入跟随状态,防止移动物体跑出机械臂工作空间时仍未实

现抓取，我们根据参考论文，把机械臂末端简化为质点 A，移动物体简化为质点 B，选择分解速度规划算法来完成机械臂末端对物体的跟踪。速度方向分解的定义如下：



如上图所示，定义为 V_A 质点 A 的速度； a_A 为质点 A 的加速度， V_B 为质点 B 的速度， a_B 为质点 B 的加速度，以 A 为原点，先由 \overrightarrow{AB} 和 V_B 定义出 $x-y$ 平面，定义：

$$S_y = \begin{cases} \frac{\overrightarrow{AB} \cdot \overrightarrow{V_B}}{|\overrightarrow{AB}| \cdot |\overrightarrow{V_B}|} \cdot |\overrightarrow{AB}| \cdot \frac{V_B}{|V_B|} & , |V_B| \neq 0 \\ \frac{\overrightarrow{AB} \cdot \overrightarrow{a_B}}{|\overrightarrow{AB}| \cdot |\overrightarrow{a_B}|} \cdot |\overrightarrow{AB}| \cdot \frac{a_B}{|a_B|} & , |V_B| = 0 \end{cases}$$

$$S_x = \overrightarrow{AB} - S_y$$

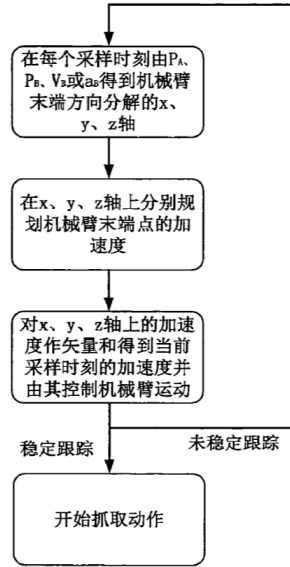
并设定 y 轴与 S_y 同向，定义 x 轴与 S_x 同向，若 $|S_x| = 0$ ，则定义 x 轴为与 y 轴垂直的任意方向，z 轴的方向则可由右手定则得到。由 x, y, z 轴的定义，将 V_A 分别投影到 x, y, z 轴得到 V_{Ax} , V_{Ay} , V_{Az} ， a_A 分别投影到 x, y, z 轴得到 a_{Ax} , a_{Ay} , a_{Az} 。

质点 A 跟踪质点 B 的问题可通过在每个采样时间内计算质点 A 的加速度指令实现，并且由于采样时间通常都比较小，因此质点 B 在每个采样时间内的速度改变可以忽略不计，于是可假设质点 B 在每个采样时间内都以恒定的速度运动，由此可将跟踪问题分解为以下三个子问题解决：

在限制最大速度与最大加速度的情况下，

1. 设计质点 A 在 x 轴上的加速度 a_{Ax} 以满足使质点 A 以最短的时间沿 x 轴移动 S_x ，且此时 $V_{Ax} = 0$ 的目标
2. 设计质点 A 在 y 轴上的加速度 a_{Ay} 以满足使质点 A 以最短的时间沿 y 轴移动 $S_y + S_B$ ，且此时 $V_{Ay} = V_B$ 的目标，其中 S_B 为质点 B 在此过程中的位移
3. 设计质点 A 在 z 轴上的加速度 a_{Az} 以满足使质点 A 在 z 轴方向上的速度 V_{Az} 以最短的时间变为 0 的目标，以保证由 \overrightarrow{AB} 和 V_B 构成的 $x-y$ 平面保持不变。

所以质点 A 跟踪质点 B 可以化简为一维跟踪问题，根据不同情况设计质点 A 在 x、y、z 轴上每个采样时间的加速度即可。应用分解速度抓取过程中的流程图如下所示：



同时计算 x 、 y 、 z 轴上加速度的伪代码分别如下：

● X 轴方向上的加速度计算

x 轴总是与 V_B 垂直，因此若要使 $V_A = V_B$ ，在稳定跟踪的状态下 $V_{Ax} = 0$ ，具体的算法设计如下：

x 轴方向上的加速度控制

1. 计算表 2.1 的情况 1 中由 $V_{Ax} \cdot Ref_x$ 加速至 $V_{x,max}$ ， $t = 0$ 且最后减速至 0 时的位移 S_{x1}
2. If $S_{x1} \cdot Ref_x < S_x \cdot Ref_x$ ，则 $a_{Ax} = a_{x,max} \cdot Ref_x$
3. Else 如果 $V_{Ax} \cdot Ref_x \geq 0$ ，则计算情况 3 中减速至 0 时的位移 S_{x2} ；如果 $V_{Ax} \cdot Ref_x < 0$ ，则计算情况 4 中加速至 0 时的位移 S_{x2}
4. If $S_{x2} \cdot Ref_x < S_x \cdot Ref_x$ ，则 $a_{Ax} = a_{x,max} \cdot Ref_x$
5. Else 则 $a_{Ax} = -a_{x,max} \cdot Ref_x$
6. End 4 If
7. End 2 If
8. If $V_{Ax} \cdot a_{Ax} > 0 \wedge |V_{Ax} + a_{Ax} \cdot t_{sample}| \geq V_{x,max}$ 或 $|V_{Ax}| = |S_x| = 0$ ，则 $a_{Ax} = 0$

$$\text{其中 } Ref_x = \begin{cases} \frac{S_x}{|S_x|} & |S_x| \neq 0 \\ \frac{V_{Ax}}{|V_{Ax}|} & |S_x| = 0 \end{cases}$$

● Y 轴方向上的加速度计算

y 轴总是与 V_B 共线，因此若要使 $V_A = V_B$ ， V_{Ay} 必须在稳定跟踪时等于 V_B ，具体算法设计如下：

y 轴方向上的加速度控制

1. 计算情况 1 中由 $V_{Ay} \cdot Ref_y$ 加速至 $V_{y,max}$ ， $t = 0$ 且最后减速至 $V_B \cdot Ref_y$ 时 A 的位移 S_{y1} 和该过程的时间 t_1 以及在 t_1 时间后以速度 V_B 作匀速运动的 B 的位移 S_{B1}
2. If $S_{y1} \cdot Ref_y < (S_y + S_{B1}) \cdot Ref_y$ ，则 $a_{Ay} = a_{y,max} \cdot Ref_y$
3. Else 如果 $V_{Ay} \cdot Ref_y \geq V_B \cdot Ref_y$ ，则计算情况 3 中减速至 $V_B \cdot Ref_y$ 时的位移 S_{y2} 和该过程的时间 t_2 以及在 t_2 时间后以速度 V_B 作匀速运动的 B 的位移 S_{B2} ；如果 $V_{Ay} \cdot Ref_y < V_B \cdot Ref_y$ ，则计算情况 4 中加速至 $V_B \cdot Ref_y$ 时的位移 S_{y2} 和该过程的时间 t_2 以及在 t_2 时间后以速度 V_B 作匀速运动的 B 的位移 S_{B2}

```

4.      If  $S_{y2} \cdot Ref_y < (S_y + S_{B2}) \cdot Ref_y$ , 则  $a_{Ay} = a_{y\_max} \cdot Ref_y$ 
5.      Else 则  $a_{Ay} = -a_{y\_max} \cdot Ref_y$ 
6.      End 4 If
7.  End 2 If
8.  If  $V_{Ay} \cdot a_{Ay} > 0 \wedge |V_{Ay} + a_{Ay} \cdot t_{sample}| \geq V_{y\_max}$  或  $V_{Ay} = V_B \wedge |S_y| = 0$ , 则
       $a_{Ay} = 0$ 

```

$$\text{其中 } Ref_y = \begin{cases} \frac{S_y}{|S_y|} & |S_y| \neq 0 \\ \frac{V_B}{|V_B|} & |S_y| = 0 \end{cases}$$

● Z 轴方向上的加速度计算

由 x、y、z 轴的定义，稳定跟踪的必要条件是质点 A 只在 x-y 平面内运动，即 A 在 z 轴方向的速度分量 $V_{Az} = 0$ ，因此若 $|V_{Az}| \neq 0$ ，则设计质点 A 在 z 轴方向的加速度分量 a_{Az} 使其速度分量 V_{Az} 减速为 0。具体算法如下：

z 轴方向上的加速度控制

```

1.  If  $|V_{Az}| \neq 0$ ，计算  $V_{Az}$  以最大加速度减速至 0 时的时间  $t_z$ 
2.      If  $t_z > t_{sample}$ ，则  $a_{Az} = -a_{z\_max} \cdot \frac{V_{Az}}{|V_{Az}|}$ 
3.      Else  $a_{Az} = -\frac{V_{Az}}{t_{sample}}$ 
4.      End 2 If
5.  Else  $a_{Az} = \vec{0}$ 
6.  End 1 If

```

在我们合成上面三个方向的加速度，即可得到机械臂末端每个时刻的加速度，进而通过积分得到速度和位置信息，然后通过逆运动学分解得到各个舵机上的速度和角度信息。

2.2.2 部分代码介绍

我们抓取控制的算法分为了 `controller.py`，`func.py`，`para.py` 三个部分。`controller.py` 为主程序，包含了程序主循环以及框架；`func.py` 中都是一些打包好的函数；`para.py` 为一些运行参数。这里我们解释一下 `func.py` 中各函数所实现的功能：

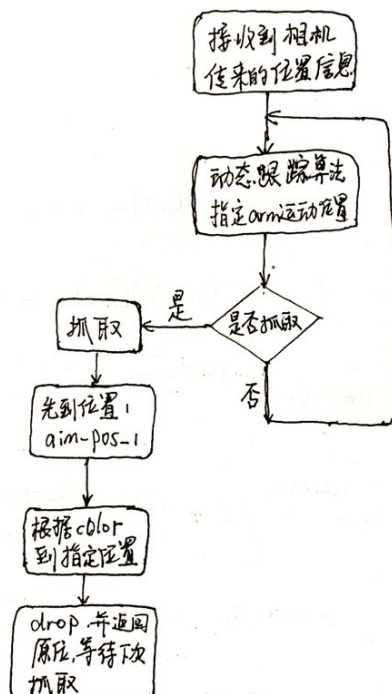
- 1) `def move(p, v, ts)`: 已知速度 v 与时间 ts ，得到目标运动终点位置。
- 2) `def projection(v, p)`: 得到向量 v 在向量 p 上的投影
- 3) `def forward(angle)`: 正向运动学，已知各轴角度，返回末端位姿 `pose`
- 4) `def inverse(pose)`: 逆向运动学，已知末端位姿 `pose`，返回所有可能的角度解 `angle` (list)
- 5) `def judge(angle_0, angle_1)`: 判断逆运动学解中，转动角度最少的情况作为最终选用的解
- 6) `def talker_pos(q, pub, catch)`: 将角度位置信息发送给机械臂
- 7) `def angle_range(angle)`: 将所有角度限制在 $(-180, 180]$ ，方便计算与比较
- 8) `def dynamic_tracking(p_a, p_b, v_a, v_b)`: 速度分解的运动跟踪算法
- 9) `def to_aim(p, angle, pub, catch)`: 指定机械臂末端到达指定位置，方便放置物体与回归原位
- 10) `def angle2motor(angle, catch)`: 将角度数据转换为发送给电机的数据
- 11) `def do_catch(p_a, p_b)`: 判断机械臂与物体位置是否达到可抓取范围，我们设定为小

于 0.001

12) `def drop(color)`: 根据不同颜色指定物块不同的投放地点

● controller.py 中代码要点介绍

下图所示为 controller.py 的算法流程图



1. 我们创建了 `cv_position_controller` 节点，当相机读取到位置信息后就会朝该节点发送信息，然后主程序接收到信息，执行 `callback` 函数。
2. 由于在该算法的跟踪过程中机械臂末端存在一定的震荡，所以我们判断是否进行抓取并不是对每一个循环都进行判断，而是当二者之间的距离第一次达到指定范围内便开始抓取动作，就算之后抓取过程中二者之间的距离稍微超出了指定范围也不影响。实物的夹爪长度其实远远大于震荡的偏差，经过实物测试，这种方法是可行的。
3. 我们为了体现跟随过程，指定让机械臂先跟随物体 1s，然后留有 1s 抓取时间（夹爪闭合的时间），再让机械臂运动到指定位置。
4. 在最开始实物实现时我们发现，如果直接将物块从抓取位置送到指定放置位置，路径上很容易将已经放置好的位置撞开。所以我们选取了一个高度较高的中间位置(`aim_pos_1 [-0.2, 0.2, 0.3]`)。机械臂抓到物块后，都要先运动到该位置，然后再将不同物块放置到指定位置，这样就不会发生碰撞。

● 逆运动学解部分 `def inverse(pose)`

该 `fobot` 机械臂与 `anno` 机械臂的大体结构相同，只是 DH 参数表方面的一些区别。但由于我们希望找到所有解，故采用的是解析解每个角度的方法，这需要我们对每个角度的计算公式重新推导。

1. 我们已知 `fobot` 机械臂的 DH 参数表，首先写出各坐标系之间的位姿变换矩阵

$${}^0T_1 = \begin{bmatrix} \cos \theta_1 & 0 & \sin \theta_1 & 0 \\ \sin \theta_1 & 0 & -\cos \theta_1 & 0 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^1T_2 = \begin{bmatrix} \cos \theta_2 & -\sin \theta_2 & 0 & l_2 \cos \theta_2 \\ \sin \theta_2 & \cos \theta_2 & 0 & l_2 \sin \theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^2T_3 = \begin{bmatrix} \cos \theta_3 & 0 & \sin \theta_3 & l_3 \cos \theta_3 \\ \sin \theta_3 & 0 & -\cos \theta_3 & l_3 \sin \theta_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^3T_4 = \begin{bmatrix} \cos \theta_4 & 0 & \sin \theta_4 & 0 \\ \sin \theta_4 & 0 & -\cos \theta_4 & 0 \\ 0 & 1 & 0 & l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^4T_5 = \begin{bmatrix} \cos \theta_5 & 0 & \sin \theta_5 & 0 \\ \sin \theta_5 & 0 & -\cos \theta_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^5T_6 = \begin{bmatrix} \cos \theta_6 & 0 & -\sin \theta_6 & 0 \\ \sin \theta_6 & 0 & \cos \theta_6 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^6T_7 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -d_6 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

其中 $l_1 = 0.241$, $l_2 = 0.18$, $l_3 = 0.027$, $l_4 = 0.18296$, $d_6 = 0.06185$

2. 我们已知终点的位置(fx, fy, fz)与 rpy 姿态角, 可以直接写出总的位姿变换矩阵

$${}^0T_7 = \begin{bmatrix} \cos p \cos y & \sin r \sin p \cos y - \cos r \cos y & \sin r \sin y + \cos r \sin p \cos y & fx \\ \cos p \sin y & \cos r \cos y + \sin r \sin p \sin y & \cos r \sin p \sin y - \cos y \sin r & fy \\ -\sin p & \cos p \sin r & \cos r \cos p & fz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

所以我们需要做的就是求解如下的矩阵方程:

$${}^0T_7 = {}^0T_1 {}^1T_2 {}^2T_3 {}^3T_4 {}^4T_5 {}^5T_6 {}^6T_7 = {}^0T_3 {}^3T_6 {}^6T_7$$

3. 首先求得

$$T_{30} = \begin{bmatrix} \cos(t2 + t3) \cos(t1), & \sin(t1), & \sin(t2 + t3) \cos(t1), & \cos(t1) * (13 \cos(t2 + t3) + 12 \cos(t2)) \\ \cos(t2 + t3) \sin(t1), & -\cos(t1), & \sin(t2 + t3) \sin(t1), & \sin(t1) * (13 \cos(t2 + t3) + 12 \cos(t2)) \\ \sin(t2 + t3), & 0, & -\cos(t2 + t3), & 11 + 13 \sin(t2 + t3) + 12 \sin(t2) \\ 0, & 0, & 0, & 1 \end{bmatrix}$$

$${}^3T_6 = {}^3T_4 {}^4T_5 {}^5T_6 = \begin{bmatrix} \sin \theta_4 \sin \theta_6 + \cos \theta_4 \cos \theta_5 \cos \theta_6 & -\cos \theta_4 \sin \theta_5 & \cos \theta_6 \sin \theta_4 - \cos \theta_4 \cos \theta_5 \sin \theta_6 & 0 \\ \cos \theta_5 \cos \theta_6 \sin \theta_4 - \cos \theta_4 \sin \theta_6 & -\sin \theta_4 \sin \theta_5 & -\cos \theta_4 \cos \theta_6 - \cos \theta_5 \sin \theta_4 \sin \theta_6 & 0 \\ \cos \theta_6 \sin \theta_5 & \cos \theta_5 & -\sin \theta_5 \sin \theta_6 & l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^0T_6 = {}^0T_7 {}^6T_7^{-1} = \begin{bmatrix} & & dx \\ & {}^0R_6 & dy \\ & & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}^0T_3 {}^3T_6$$

化简上式可得如下表达式:

$$\begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} = \begin{bmatrix} \cos(t1) * (13 * \cos(t2 + t3) + 12 * \cos(t2)) + 14 * \sin(t2 + t3) * \cos(t1) \\ \sin(t1) * (13 * \cos(t2 + t3) + 12 * \cos(t2)) + 14 * \sin(t2 + t3) * \sin(t1) \\ 11 - 14 * \cos(t2 + t3) + 13 * \sin(t2 + t3) + 12 * \sin(t2) \end{bmatrix}$$

其中 dx 、 dy 、 dz 均已知，直接求解 θ_1 、 θ_2 、 θ_3 得到如下结论：

$\theta_1 = \text{atan2}(d_y, d_x)$ ，该处最多有两个解 $\theta_1 \in [\text{atan2}(d_y, d_x), \text{atan2}(d_y, d_x) + \pi]$ 而 θ_2 、 θ_3 则可按如下手稿进行推导得到

$$\begin{cases} dx = \cos \theta_1 [l_3 \cos(\theta_2 + \theta_3) + l_2 \cos \theta_2 + l_4 \sin(\theta_2 + \theta_3)] \\ dy = \sin \theta_1 [l_3 \cos(\theta_2 + \theta_3) + l_2 \cos \theta_2 + l_4 \sin(\theta_2 + \theta_3)] \\ dz = l_1 - l_4 \cos(\theta_2 + \theta_3) + l_3 \sin(\theta_2 + \theta_3) + l_2 \sin \theta_2 \end{cases}$$

$$\Rightarrow \tan \theta_1 = \frac{dy}{dx} \quad \theta_1 = \tan^{-1} \left(\frac{dy}{dx} \right)$$

$$dx \cos \theta_1 + dy \sin \theta_1 = l_3 \cos(\theta_2 + \theta_3) + l_2 \cos \theta_2 + l_4 \sin(\theta_2 + \theta_3) = q_1$$

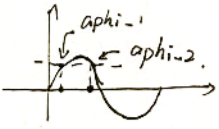
$$dz - l_1 = -l_4 \cos(\theta_2 + \theta_3) + l_3 \sin(\theta_2 + \theta_3) + l_2 \sin \theta_2 = q_2$$

令 q_1, q_2 已知，令 $\theta_2 + \theta_3 = A$ ， $\theta_2 = B$

$$\Rightarrow \text{消去 } B \text{ 得: } (l_3^2 - l_2^2 + l_4^2 + q_1^2 + q_2^2) + \frac{(-2l_3l_4 - 2l_4q_1)}{AA} \sin A + \frac{(2l_4q_2 - l_3^2)}{AA} \cos A = 0$$

CC. AA

利用辅助角公式，当 $AA > 0$ 时，(小于0就两边同乘-1)

$$\sin(A + \varphi) = -\frac{CC}{\sqrt{AA^2 + BB^2}}, \quad \varphi = \tan^{-1} \left(\frac{BB}{AA} \right)$$


即可求得 A ，进而解出 B

又 $A = \theta_2 + \theta_3$ ， $B = \theta_2$ ，易得 θ_2, θ_3 的值。

4. 已经得到 θ_1 、 θ_2 、 θ_3 的值后，我们便可求出 0T_3 ，进而通过 ${}^3T_6 = {}^0T_3^{-1} {}^0T_6$ 求出 3T_6

令

$${}^3T_6 = \begin{bmatrix} s_{11} & s_{12} & s_{13} & 0 \\ s_{21} & s_{22} & s_{23} & 0 \\ s_{31} & s_{32} & s_{33} & l_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

与上面的 3T_6 表达式对比，我们得出如下结论：

$$\begin{aligned} \theta_4 &= a \tan 2(s_{22}, s_{12}) \\ \theta_5 &= a \tan 2(\sqrt{s_{12}^2 + s_{22}^2}, s_{32}) \\ \theta_6 &= a \tan 2(s_{33}, -s_{31}) \end{aligned}$$

5. 解的验证

值得注意的是，我们要考虑到反正切函数的多解型，所以我们在使用反正切函数得到一个角度时，还要将其加上180的角度考虑是否有其他的可能解。另外在求得4、5、6三个角

时，我们并没有用到全部的条件，所以得到的角度可以采用另外的等式去验证。

● 运动跟踪算法 `def dynamic_tracking(p_a,p_b,v_a,v_b)`

python 代码大部分按照伪代码即可实现，其中有一些要点：

1. 本算法涉及到许多向量的运算，我们使用 python 的 **numpy** 包可以很方便的实现许多功能。`np.linalg.norm` 可以计算向量的模；`np.cross` 可以计算两向量的叉积；`np.inner` 可以计算两向量的内积
- 2、可以使用单位向量 `ref_x,ref_y,ref_z` 方便我们的运算。这里我们要注意 S_y 或者 S_x 为 0 的情况。当 S_y 为 0 时，我们选用 V_B 的方向作为 S_y 正方向；当 S_x 为 0 时，我们选用任意与 S_y 垂直的方向作为 S_x 正方向，本代码中我们采用

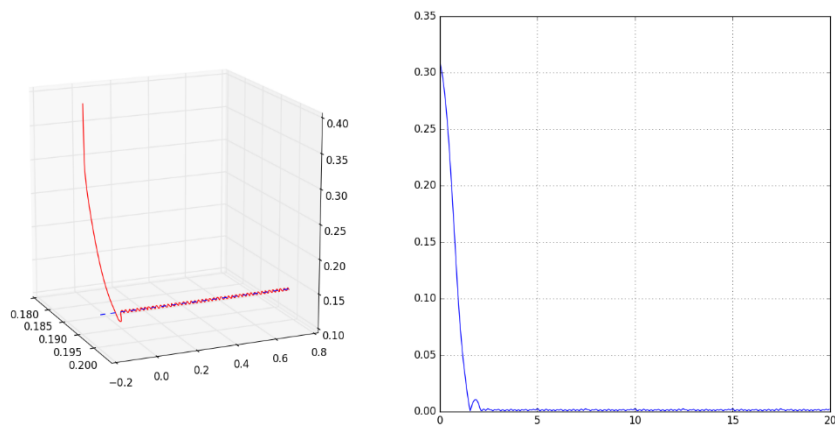
```
1. ref_x = np.cross(s_y, s_y+1) / np.linalg.norm(np.cross(s_y, s_y+1))
```

我们也可以使用如下 python 库对我们的跟踪情况进行可视化的观察：

```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

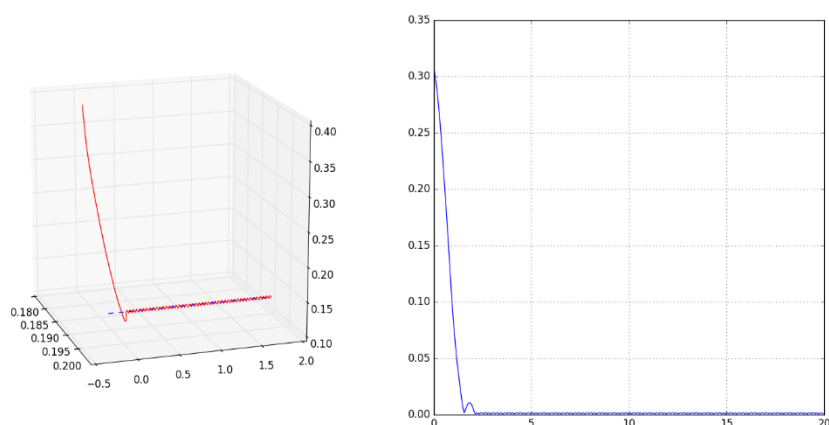
```
1. fig = plt.figure(figsize=[18, 8])
2. ax1 = fig.add_subplot(121, projection='3d')
3. ax2 = fig.add_subplot(122)
4.
5. ax1.plot3D(pa_result[0],pa_result[1],pa_result[2], 'r')
6. ax1.plot3D(pb_result[0],pb_result[1],pb_result[2], 'b--')
7. ax2.plot(np.arange(0, t_max, ts), error_result)
8. ax2.grid()
9.
10. plt.show()
```

当 $v = 0.0483$ 时（此时对应传送带速为 45）



左侧红线为机械臂运动路径，蓝线为目标物体运动路径；右侧为二者之间随时间变化的距离。可以看到该算法能很快地使二者保持一定误差范围内的跟随状态。但是可以看到该跟随并不是完全稳定的，存在连续不断的震荡。通过我们多次结合实物实验发现该微小的震荡对实物并没有什么影响。反而如果我们增大发送频率来减小该震荡时，机械臂末端由于接收位置信息过于密集，运动误差反而增大。

当 $v = 0.1$ 时



可以看到在较高的速度时，也能实现跟随效果。

2.3 相机与机械臂的通讯方案

在实验中，我们通过调用 OpenCV 库读取相机拍摄到的内容并进行识别。为了将其整合进机械臂控制程序，我们使用了 ROS 的 topic 功能作为通讯方式，将图像识别与机械臂控制隔离为两个节点，分别完成各自的工作。这样的好处是：一方面我们对代码进行了解耦，方便调试处理各个功能；另一方面借助 ROS 的功能，可以通过网络分离图像识别节点，在实际应用中，控制机械臂的工控机可以只负责控制机械臂，对计算机处理性能要求较高的图像识别可以由其他服务器完成。

具体实现方式如下：

在机械臂控制代码中，对视觉处理节点的 topic 进行订阅

```
1. sub=rospy.Subscriber('cv_position_publisher', Float64MultiArray, queue_size=100, callback=go)
```

在全局变量中，定义 running 变量。当运动开始时，设置为 True；结束后设置为 False，相当于对视觉节点传入的信息做一次简单的去抖动。通过接收视觉节点传来的位置并进行解算，可以让机械臂抓取识别到的物体。

在具体操作过程中，我们遇到了 OpenCV 库不兼容的情况。本次实验中我们使用了旧版 (kinetic) 的 ROS 程序，其自带的 OpenCV 版本较低，功能缺失。由于在其他控制部分中没有使用到 OpenCV 这个库，我们将新版的二进制文件替换掉了 ROS 版本的二进制文件，使代码成功运行。

2.4 机械爪抓取与物块摆放方案

对于机械爪的抓取，我们小组通过对机械臂实际情况的考虑，最终采用从上往下垂直抓取的抓取方式。从上往下垂直抓取的方式能够保证机械臂有一个较大的抓取范围（为杆①、②在限位条件下的最大圆域），同时该抓取方案有相当好的一致性，这是我们采用此方案的最大原因，只要物块在机械臂的可抓取范围内，那么我们只要保证机械爪下落位置准确、下

降的高度合适，从理论上讲就能有一个较好的抓取效果。



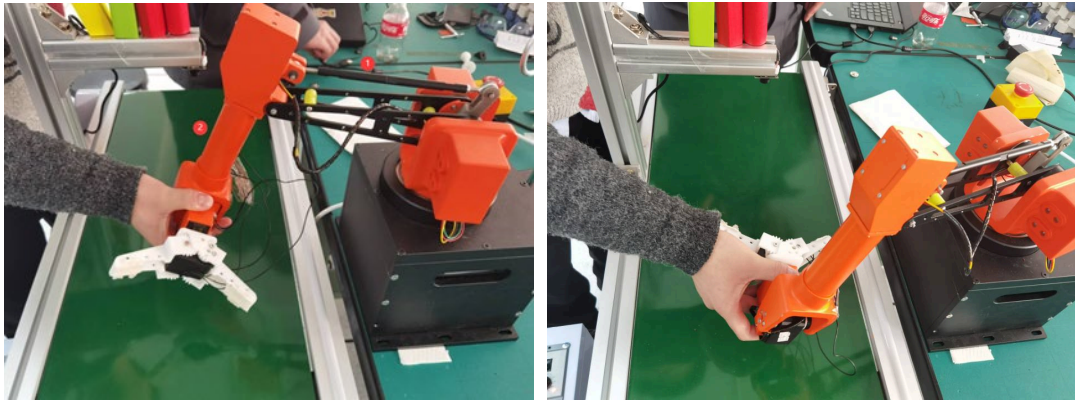
图 机械臂垂直抓取方式

不过对机械爪下落位置的准确性要求高，也是该抓取方案的一个最大的劣势。我们一开始对物体定位的准确性较差，所以垂直抓取的过程发生了机械爪在传送带运动方向上与物体未对齐，导致机械爪与物体发生碰撞，物体倾倒的问题，也发生了机械爪在传送带宽度方向上与物体未对齐，导致机械爪抓了个空，具体下图所示：



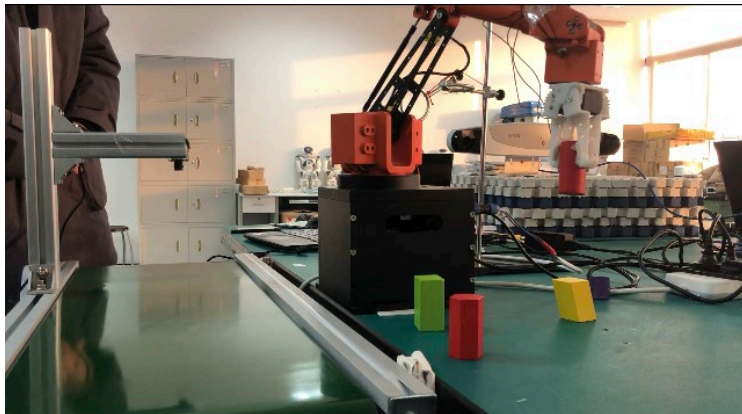
除了垂直抓取的方案之外，还有机械爪横向抓取的方案。机械爪横向抓取时机械爪的抓取范围（不是物体的可抓取区域）远远大于垂直抓取，更容易抓住物体，对物体定位的准确度要求较低，但是由于机械臂物理条件的限制，我们没有采用这种方案。





观察上面 4 幅图可以发现，首先，采用横向抓取方式的抓取范围过小，可以发现机械爪最近的位置已经超过我们摄像头的中心，这主要是由于杆①③长度的限制，如果我们的传送带离机械臂远一些，则有可能采用上两图示例的横向抓取方案；而下两图所显示的问题，则是机械爪无法水平抓取物体，当杆①达到最远位置时，只能转动杆②，而此时则会导致机械爪无法保持水平，无法很好地抓取物体。

而物体的摆放，我们按照物体的不同颜色在桌子上确定四个摆放位置 $P_1 \sim P_4$ ，让机械爪在抓到物体之后，首先到达 $P_1 \sim P_4$ 四点构成的四边形的中心上空，然后在按照相应颜色放到相应的位置。这样一来，能够非常有效地解决机械爪在放置物块时的碰撞问题。我们的碰撞问题，指的是我们的物块摆放位置有靠里和靠外，如果我们首先放置了靠外的物体，那么在放置靠里的物体时，则会与已经摆放好的物体发生碰撞。最后至于为什么没有采用在机械臂侧边一字排开的摆放方案，则主要是因为机械臂能够达到的最远范围有点小，如果在四种颜色的物块全都放在侧边，则过于拥挤，所以我们把部分物体摆放位置放到了机械臂靠后的位置。具体如下图所示：



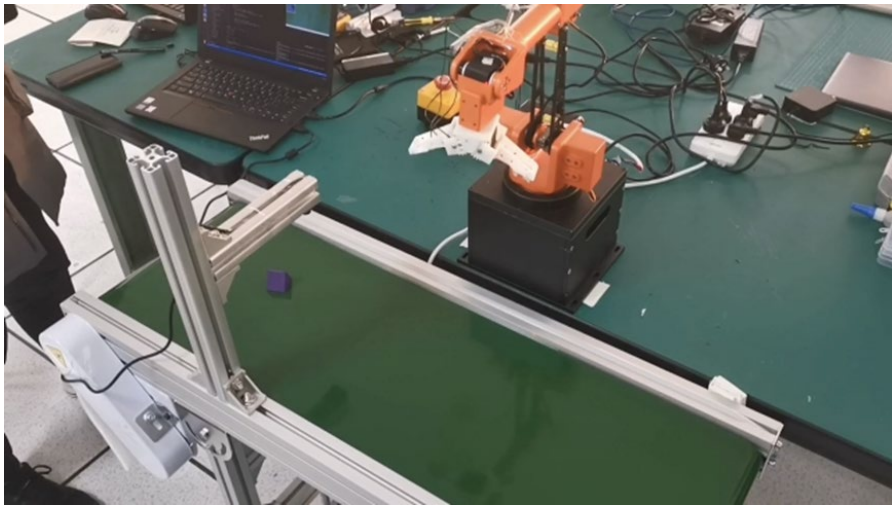
综上所述，我们采用机械爪垂直抓取的方法以及通过一个中间点的摆放物块的方式。

三、 实验步骤

3.1 硬件设备的选择与安装

机器人移动抓取项目的硬件设备部分主要分为机械臂与机械爪、摄像头、传送带、支架、物块，具体示意见实验内容中图片。其中，机械臂朝向向外，放在平坦桌面上；调节支架高度，使传送带表面与桌面平齐，将传送带靠紧桌面边缘，传送带输送物块的方向（轴向）与

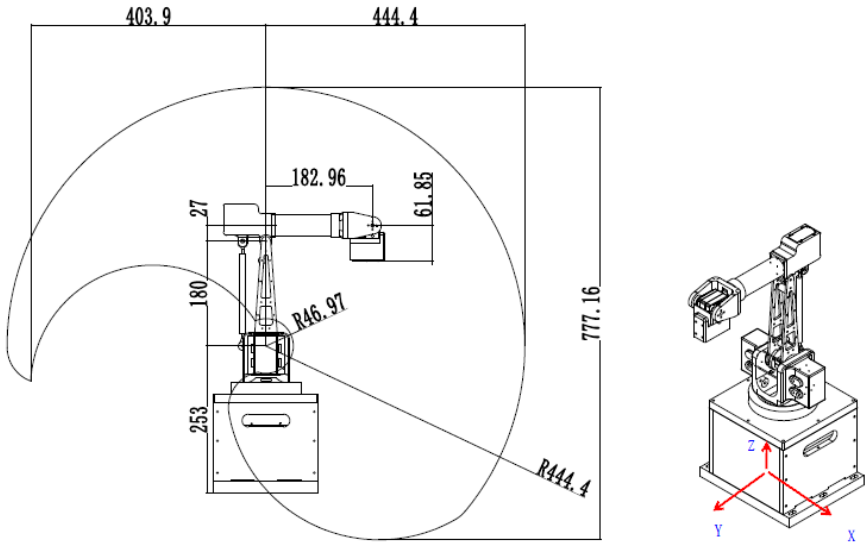
机械臂的朝向（径向）相垂直；调节支架轴向位置、径向延伸长度和高度，使摄像头恰好位于传送带中心正上方，距离传送带表面的高度合适，距离机械臂的轴向距离合适，以恰好不拍摄到传送带的边框为宜。整体布局如下图所示：



摄像头与机械臂坐标原点距离、机械坐标原点和传动带运动方向垂直方向间的距离等等，这些由于我们的开环系统的原因，全部都需要得到控制。

1) 机械臂与机械爪：

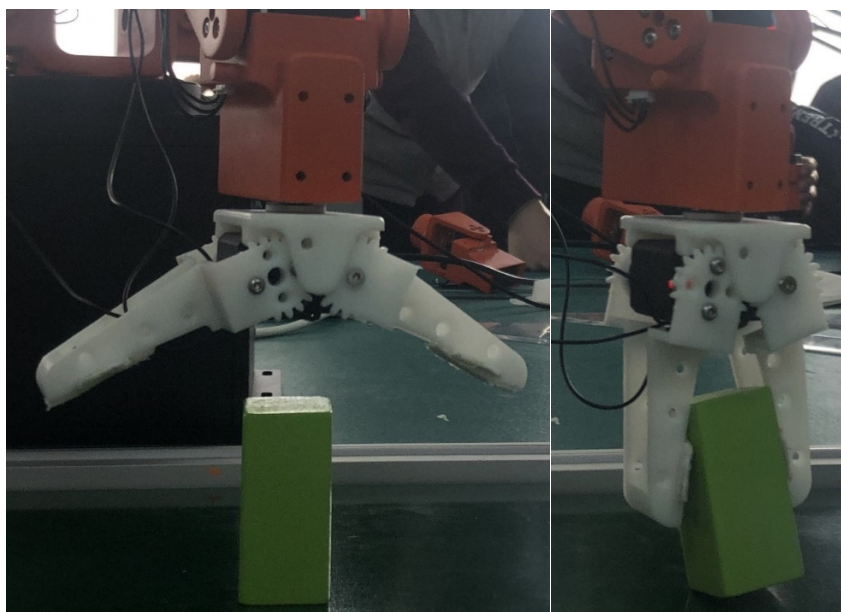
机械臂负责在正确的时间点将机械爪运动到抓取点处。机械臂为由助教提供的标准 6DOF 机械臂，尺寸参数如下所示：



其中机械臂的 2 号关节的电机布置在 1 号关节两侧，通过连杆结构驱动 2 号关节的俯仰运动。

由于机械臂存在工作空间、移动速度和精度限制，所以需要考虑好机械臂与其他模块之间的空间位置关系，比如：传送带应当布置在机械臂工作空间内，物块跟踪段和抓取点最好始终位于机械臂运动的高精度区内。

机械爪负责抓取物块。机械爪为单舵机控制，齿轮啮合传动的双指式机械爪，手指长度 10cm，结构简单，抓取方便，抓取力大。机械爪结构如下图所示：



2) 摄像头:

摄像头负责识别传送带上输送的物块,并确定物块的移动方向和移动速度,将物块的空间位置信息发送给机械臂用于移动抓取。摄像头为淘宝购买的 USB 摄像头模组,支持安卓 UVC 和 USB2.0 协议,长宽为 20mmX20mm,镜头长度 20mm,焦距 2.6mm,720P/100 万像素,拍摄角度 120°,无畸变,拍摄速率 30 帧/s。

如下图所示:



3) 传送带:

传送带负责以稳定的速度输送物块到识别点和抓取点。传送带包括三部分,电机、链和皮带传动机构、支架。传送带长度 100cm,皮带宽度 30cm,可调高度 75-100cm,电机速比 1: 30。

如下图所示:



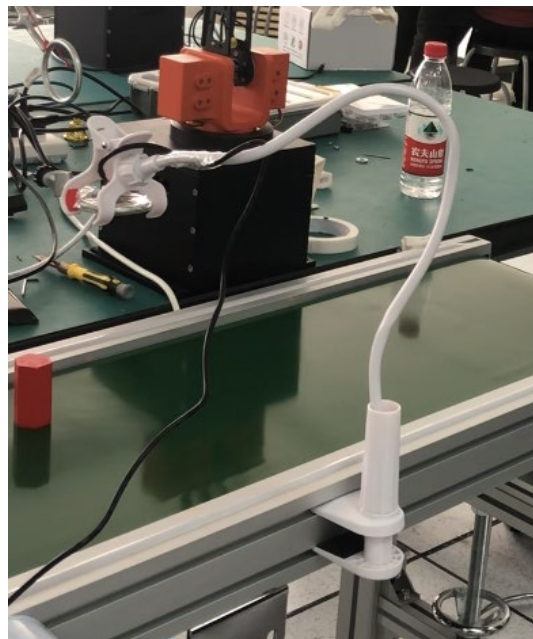
常规高度75cm—100cm

4) 支架：

支架负责将摄像头固定在合适的空间位置，方便摄像头识别物块并确定物块的空间位置。

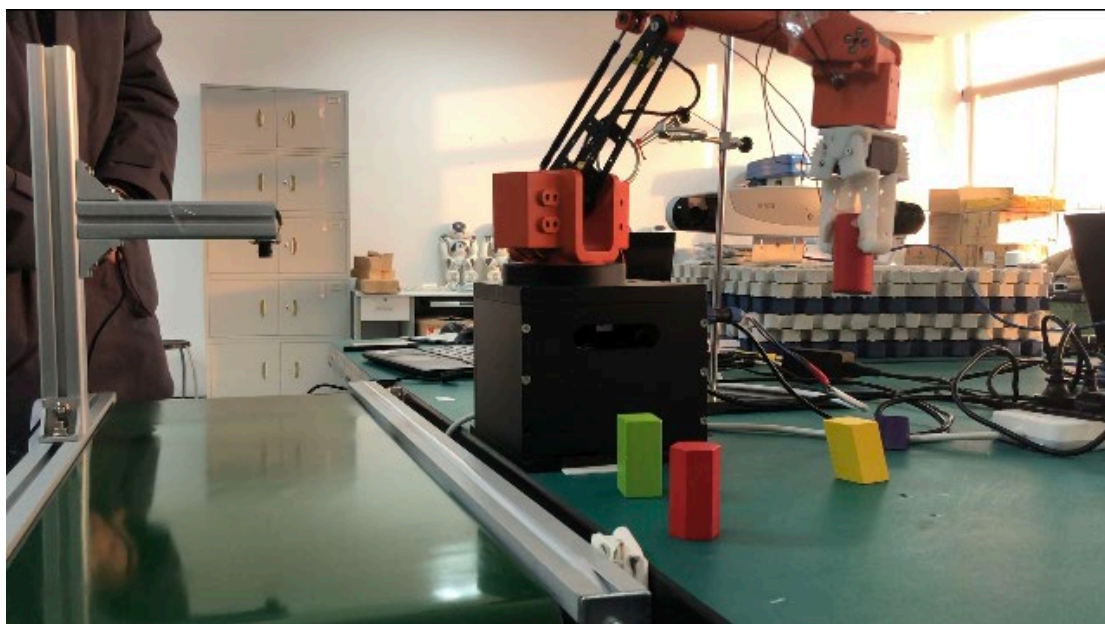
我们有两种直接设计方案：

一是直接用手机支架充当摄像头支架。如下图所示：



这是支架铝型材未到时的权宜之计，经测试发现，手机支架的可调性能很好，可以调整到任意合适的位置，但是其刚度不够，容易因测试时的磕碰而改变摄像头相对于机械臂的空间位置，导致移动抓取误差增大，而且传送带运行过程中的轻微震动也会沿支架传递给末端的摄像头，并产生放大效应，引起摄像头的更剧烈的震动。

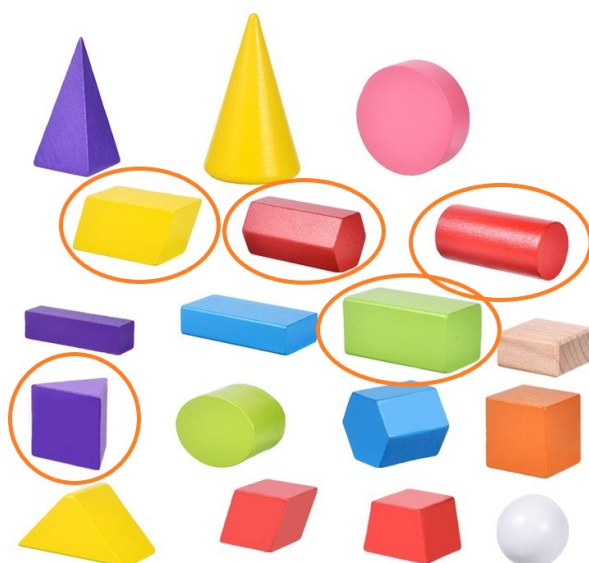
二是采用铝型材。如下图所示：



铝型材为标准规格 40mmX40mm，以螺栓螺母固定，摄像头固定十分牢固，传送带震动传递小，物块定位精度高。

5) 物块：

物块为淘宝购买，分多种不同形状和颜色，可以满足分类抓取的需求。如下图所示：



考虑到摄像头是正上方识别和定位，机械爪是自上而下竖直抓取，所以我们挑选了上述圈内的 5 种物块，既满足了多种不同形状和颜色的要求，也方便物块种类识别，还方便机械爪抓取。

3.2 物块识别遇到的问题与改进

4) HSV 颜色空间识别精度不够

使用 HSV 颜色空间的缺陷在于其受光线等外界条件的影响较大，同时给定的颜色阈值范围不够精确，容易出现颜色无法识别或者蓝色和绿色、红色和紫色混淆等情况。

	红	黄	绿	蓝	紫
Hmin	0/156	26	35	100	125
Hmax	10/180	34	77	124	155
Smin	43	43	43	43	43
Smax	255	255	255	255	255
Vmin	46	46	46	46	46
Vmax	255	255	255	255	255

	红	黄	绿	蓝	紫
Hmin	0	26	35	100	125
Hmax	10	34	77	124	155
Smin	127	110	110	43	43
Smax	255	255	255	255	255
Vmin	128	106	206	46	46
Vmax	255	255	255	255	255

如上图所示，左图为常用 HSV 颜色阈值，右图为通过实际测量各个拍摄角度和环境下图像的 HSV 信息修改后得到的阈值信息，能有效提高颜色识别精度。

5) 轮廓提取和拟合图像抖动过大

在拟合提取出颜色轮廓信息时，考虑到物块形状包括矩形、三角形、圆形，我们一开始采用的是用最小外接圆来拟合轮廓，在实际应用的过程中发现其圆心和半径抖动明显，造成较大位置偏差，改用矩形轮廓拟合能有效增强稳定性。

6) 信息实时传送重复

通过矩形轮廓拟合物块后，所得矩形轮廓的中心点即为需要传送的位置信息，考虑到相机实时读取物块信息，当有效位置信息发送后需立即停止读取，避免重复发送。改进方法是设置标志位，在位置信息发送后跳过之后 5s 内相机所读取的帧的信息，即可避免重复发送信息。

3.3 机械臂跟随抓取遇到的问题与改进

1、位置信息发送频率

根据速度分解的运动跟踪算法，理论上讲频率越高跟踪的精度也就越高。但是实际上我们在往机械臂上发送位置信息时，频率过高会导致机械臂震动，从而运动不准确。综合考虑跟踪算法精度以及实物运动效果，我们最终选取较为合适的频率为 20Hz。

2、设置中间点防止物体摆放时发生碰撞

每当我们抓取到物体后，先将其移动到高度较高的一个位置，然后再根据不同的颜色将其放置到指定位置。这是由于抓取物体的位置都较低，靠近传送带，如果这时直接将物块从传送带移到指定放置位置，路径较低，很容易碰到已经摆放好的物块。所以我们设置较高的中间点，避免这种碰撞。

3、夹爪抓取姿态

我们这次实验采用的夹爪还是比较宽大的，在一定的误差范围内，能够比较轻松抓取我们指定形状的物体。我们统一设定夹爪的朝向是从上往下抓，这样避免了机械臂末端形状对抓取的影响。

4、机械臂工作空间有限

因为机械臂的工作空间有限，所以我们设置的机械臂跟随物体的时间不宜过长，如果到达机械臂最远限位，则 ROS 中会报错，机械臂会卡死在最远位置，并且可能会一直振动，具体情况见附件视频资料。

3.4 开环抓取存在的问题以及实现闭环抓取会遇到的困难

1) 开环抓取存在的问题：

开环抓取：

摄像头识别传送带上物块的类型（形状、颜色）和移动速度、移动方向信息并传送给控制程序，控制程序根据事先给定的摄像头相对机械臂的位置、机械臂本身的 DH 参数，解算相应的机械臂关节控制信号，控制机械臂运动到物块处，先跟随运动一小段时间，随后机械爪抓取物块并放置到指定分类的位置上。

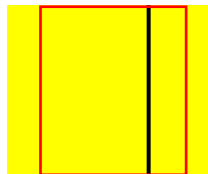
存在的问题：

1. 摄像头对物块的类型（形状、颜色）和移动速度、移动方向识别不准确，发送给控制程序的信息存在误差。

受环境光强度干扰，可能会出现物块类型识别错误（形状识别错误/颜色识别错误）的问题，导致最后机械爪没能成功抓取物块，或是虽然成功抓取但是将物块放置到了错误的位置。对于如下图所示的黄色斜方物块：



可能会出现这样的识别结果：

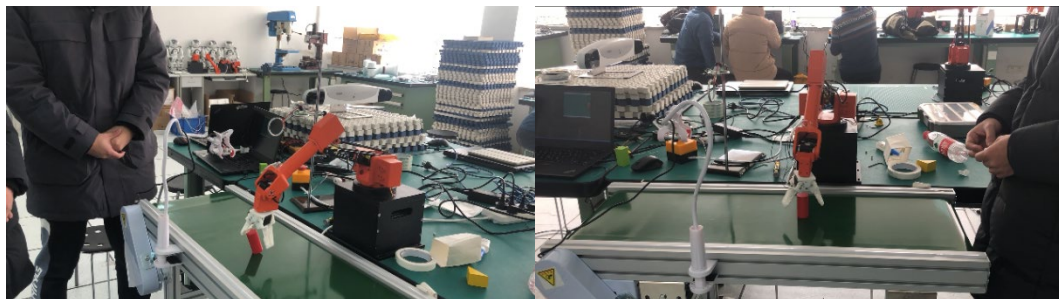


注：此为上视图，红框为摄像头识别后确定的物块轮廓，这意味着识别到的物块中心并不在物块的实际中心，机械爪抓取可能产生偏差。

2. 事先给定的摄像头相对机械臂的位置不准确，导致机械爪抓取时固定的位置偏差。

由于摄像头和机械臂分别处于两个不同的坐标系内，摄像头识别确定的物块位置信息需要换算到机械臂坐标系下，才能作为控制机械臂运动的目标参数。一旦事先给定的摄像头相对机械臂的位置不准确，就会导致机械爪抓取时固定的位置偏差。

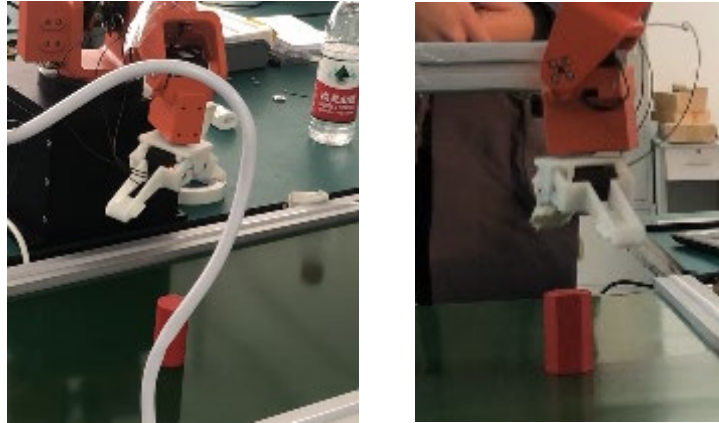
如下图所示：



垂直抓取的过程中发生了机械爪在传送带运动方向上与物体未对齐，导致机械爪与物体发生碰撞，物体倾倒的问题，也发生了机械爪在传送带宽度方向上与物体未对齐，导致机械爪抓了个空的问题。

3. 机械臂本身的 DH 参数不准确，导致机械爪抓取时的位置偏差

如果机械臂的实际 DH 参数与理论的 DH 参数不相符，会导致控制程序发送的控制指令反映到实际的机械臂上效果会发生变化。由于机械臂的 DH 参数已经由图纸给定，误差相对较小，所以 DH 参数误差主要集中在由我们自行装配的机械爪上。由于机械爪安装时的偏差，可能会导致机械爪存在初始角度偏差，如下图所示：



机械爪绕 Z 轴方向（竖直方向）存在初始角度偏差，这会导致机械爪抓取物块时存在一定的夹角，可能会导致物块在两爪之间侧滑甚至脱落而抓取失败。为了避免这种情况发生，我们在机械爪的内表面上贴上了胶条，显著增大了摩擦力，在硬件层面上提高了抓取成功率。

2) 实现闭环抓取会遇到的困难：

闭环抓取：

机械臂先根据摄像头传来的物块位置和速度信息运动到指定的位置，然后在摄像头的跟踪拍摄下，闭环调整机械爪的位置，最终使机械爪成功抓取物块。

闭环抓取能够实现很高的运动精确度和抓取成功率，但是受实际条件所限，我们并没有采用闭环抓取方案，而是仍然采用开环抓取。原因如下所述：

1. 摄像头的限制：

我们使用单个摄像头，无法获得物块的深度信息。并且如果摄像头摆放位置过高，则会导致摄像头将传送带的边框拍摄进去，可能出现错误识别的情况。所以我们将摄像头摆放于较低的高度上，只用于识别物块的水平面位置。这就导致摄像头的拍摄范围很小，无法实现跟踪拍摄。要同时拍摄并识别物块和机械臂，摄像头的拍摄范围就太小了，不能满足要求。

2. 机械臂的限制：

要实现负反馈闭环控制，就要先保证机械臂本身较高的运动精度。而机械臂本身存在较严重的间隙特性，运动速度不快，范围不大，精度不高，所以要实现精确的负反馈闭环控制较为困难。而且传送带是始终处于移动状态中的，受限于上述的摄像头拍摄范围限制，机械臂必须有较短的调整时间，收敛到较高的精度，而不能跟踪物块很长距离，这就限制了负反馈闭环控制的应用。

综上所述，我们采用了开环抓取方案，没有使用闭环抓取方案。

四、 实验创新与改进

4.1 机械臂抓取节奏

我们小组的机械臂移动抓取过程主要分为三个步骤，首先是通过放置在传送带前端的摄

像头识别物块，在物块到达摄像头正下方时，给机械臂发送物块位置和颜色信息的 topic；然后机械臂接收到信号之后，从初始位置开始运动到物体上方，跟随物体移动1s左右时间之后抓取物体；最后机械臂抓取物体到达放置位置上空，根据不同颜色放到不同的位置。

- **摄像头两次传输信息的时间间隔**

为了避免摄像头发送无效信息，我们的摄像头在发送一次 topic 之后会跳约5s的帧，然后才继续等待下一个物块的进入，所以这也就限制住了机械臂抓取物体的节奏。这边可以调整摄像头的空白时间，可以在满足机械臂走完一个流程的时间基础上，尽量减少摄像头的空白时间。

- **机械臂运动速度可进一步提高，跟随时间可适当减短**

我们小组的机械臂运动速度与传送带的移动速度相匹配，在各个运动过程中机械臂的速度设定也是保持一致的，但是该款机械臂各关节的运动速度可以更快。我们可以对不同过程设定不同的机械臂运动速度，比如在机械臂从初始位置运动快速移动到物块上方，然后按照传送带速度跟随短暂时间，最后迅速抓取，放置到相应位置。这样一种变速的过程，完成整个运动过程所花费的时间应该会小于恒定速度的过程。同时现阶段我们的跟随时间为 1s 左右，在机械臂准确到达物块上方的情况下，该跟随时间可以适当减小，进一步加快机械臂节奏。

- **摆放物体时的轨迹可进行合理规划，减少无效路径**

在这三段过程中，无效时间最长的是摆放物体所消耗的时间。由于我们确定了摆放物体时的中间点，所以机械臂在抓到物体之后，都会先移动到中间点，然后再进行摆放物体，机械臂的轨迹较长。我们可以进一步对物体的摆放位置和机械臂轨迹进行规划，比如去掉中间点，采取让机械臂绕过已摆放物体的策略，这样一来，部分物体则可以从抓取点直接放到摆放位置，而剩下物体的轨迹也只需要通过中间点的设置绕开已摆放物体即可，可以加快机械臂抓取物体节奏。

五、 实验结果

我们小组最终完成了不同颜色物体在传送带上的移动抓取，不过由于是开环系统，在传送带速度改变后，我们需要调整相应的摄像头扫到物体时传给机械臂的参数。从总体上来讲，实验效果较好，我们的机械臂能够实现具体如下功能：

1. 抓取红色（六棱柱和圆柱）、绿色（四棱柱）、黄色（斜棱柱）、紫色（低矮三棱柱）这四种不同颜色和形状的物体
2. 在参数准确的情况下，机械臂末端能够跟随移动物体约 1s
3. 抓取物体后，将不同颜色物体放到不同位置

具体实验结果见视频附件。

六、 小组成员分工

我们小组的所有成员移动抓取创新实验中都非常靠谱，大佬带飞，大家一起完成了机械臂的抓取方案设计与实验过程，其中张梁育完成机械臂的控制、实现机械臂与计算机、相机与计算机的通信，调整电机等等，代码不会就找大哥；曾浩洲主要负责机械臂正逆运动学、跟踪算法以及抓取过程的实现，是组里的全能战士；张雯琪完成了相机的识别，是负责采购

的大姐头；方胡彪和怀谦益搭建传送带等硬件环境，编写和汇总实验报告，协助完成物体定位等，哪里缺人补哪里的万金油。

七、 其他贡献说明

我们小组完善了 fobot 机械臂在 ROS 环境下的控制代码，或许算一个小贡献。

7.1 机械臂控制代码的改进

机械臂控制部分使用了 SCServo 的串口通信函数，并用 ROS topic 进行了封装，同时发布机械臂当前的状态信息。

主函数中发布原始位置信息转换后的位置信息，同时有一个接收运动指令的 Subscriber。

```
1. ros::Publisher pos_pub = n.advertise<std_msgs::Float64MultiArray>("/fobot_raw_joint_position_publisher", 100);
2. ros::Publisher position_pub = n.advertise<std_msgs::Float64MultiArray>("/fobot_joint_position_publisher", 100);
3. ros::Subscriber pos_sub = n.subscribe("/fobot_joint_position_controller", 1000, PositionCallback);
4. ros::Rate loop_rate(MSG_FREQ);
```

在主循环中，两个 Publisher 循环发送机械臂当前的位置信息，供其他节点使用。修改 MSG_FREQ 宏可以改变消息发送的频率。接收位置信息的函数只负责存储当前的期望运动目标，而运动控制也在主函数中执行，避免多个消息同时接收而产生冲突。

```
1. while (ros::ok())
2. {
3.     loop++;
4.     ROS_INFO_STREAM_ONCE("Feedback raw: Loop " << loop);
5.     for (int i = 0; i < IDN; i++)
6.     {
7.         if (s.FeedBack(i) != -1)
8.         {
9.             msg.data.at(i * 2) = s.ReadPos(-1);
10.            msg.data.at(i * 2 + 1) = s.ReadSpeed(-1);
11.        }
12.        else
13.        {
14.            ROS_ERROR_STREAM_ONCE("FEEDBACK_READ_ERROR: Loop " << loop << ", Joint " << i);
15.        }
```

```

16.     }
17.
18.     for (size_t i = 0; i < 6; i++)
19.     {
20.         calculated_msg.data.at(i) = ((msg.data.at(i) - BiasPos[i]) / SMSPOSTRAN
S / Dirction[i]);
21.     }
22.     calculated_msg.data.at(2) = ((msg.data.at(1) + msg.data.at(2) - BiasPos[2])
/ SMSPOSTRANS);
23.     calculated_msg.data.at(6) = msg.data.at(6);
24.
25.     pos_pub.publish(msg);
26.     position_pub.publish(calculated_msg);
27.
28.     //position limit judge
29.     if (ExecuteFlag) {
30.         for (int i = 0; i < IDN; i++)
31.         {
32.             if ((posres[i] > ThresholdPosUp[i]) || (posres[i] < ThresholdPosDow
n[i]))
33.             {
34.                 std::cout << "position " << i << " prores " << posres[i] << std
::endl;
35.                 ExecuteFlag = false;
36.                 std::cout << "position limit Err" << i << std::endl;
37.             }
38.         }
39.     }
40.     //drop to motor
41.     if (ExecuteFlag == true)
42.     {
43.         std::cout << "write" << std::endl;
44.         for (int i = 0; i < IDN; i++) {
45.             s.WritePosEx(i, posres[i], spd[i], acc[i]);
46.         }
47.     }
48.
49.     ros::spinOnce();
50.
51.     loop_rate.sleep();
52. }

```

八、 参考文献

- 【1】叶昕宇. 机械臂的移动物体抓取方法研究[D], 2019
- 【2】https://blog.csdn.net/A_Z666666/article/details/81324288
- 【3】<https://m.jb51.net/article/158269.htm>