机械零件定位问题的探究

**摘要**

本文针对机械零件加工过程中的定位问题，基于中心定位模型、最小面积外接矩形模型、广度优先搜索法、十字扫描定点法，对单个、多个零件进行了定位，并对模型的通用性进行了探究。

针对问题一，提出两种数学模型进行求解。1.中心定位模型：首先对图像进行均值处理，补充图像缺失的点，使之便于连通块的计算，再进行广度优先搜索，统计每个连通块中白色像素点的个数。用MATLAB计算出每个圆形轮廓的圆心，通过圆心坐标之间的关系以及连通块中像素点的个数，分辨出各连通块对应的圆。零件的位置用大圆圆心坐标表示。再利用大圆和顶部小圆圆心连线的斜率计算出零件相对于标准位置逆时针旋转的角度为。2.最小面积外接矩形模型：首先利用MATLAB编程求得零件面积最小的外接矩形，统计长边上白色像素点的个数，确定与圆相切的长边，利用几何关系和坐标之间的关系，求得大圆圆心坐标为。再利用长边的倾斜角求得零件相对于标准位置逆时针旋转的角度为。两模型得到的结果在误差范围内相等。我们对两种数学模型，各运行十次，取平均值作为该算法的运行时间方法一为，方法二为。除此之外，还引入时间复杂度的概念来评判两种模型算法的运行效率，分析得到中心定位模型的时间复杂度为，最小面积外接矩形模型的时间复杂度为，得出最小面积外接矩形模型更为高效的结论。

针对问题二 ，选择沿用问题一中心定位模型的方法。首先需要确定哪几个连通块位于同一个零件上。对此，分别计算圆心轮廓的圆心坐标，求得每个小圆圆心和两个大圆圆心之间的距离，小圆和离自身最近的大圆属于同一零件。再通过计算大圆圆心和零件顶部小圆圆心连线的斜率求得零件相对于标准位置逆时针旋转的角度。最后得到图片右上方零件的中心坐标为，相对于标准位置逆时针旋转的角度为；左下方零件的中心坐标为，相对于标准位置逆时针旋转的角度为。

针对问题三，建立的两种模型的通用性分别进行讨论。对于中心定位模型，首先探究对连通块形状的要求。发现该模型能够适用于圆形和存在平行且相等边的多边形，如，平行四边形、边数为偶数的正多边形。再对连通块的数量进行探究，发现适用于零件连通块数量不小于两个的情况，当零件数量等于两个时，要求零件外轮廓的中心与内部连通块的中心不相重合。对于最小面积外接矩形模型，从矩形各条边上的白色像素点数量的不同分布情况进行讨论分析。发现本模型适用于最小外接矩形至少有一条边上的白色像素点数量区别于其余边，以及矩形两组邻边上白色像素点数量分别相等的情况。

**关键词:** 中心定位模型 最小面积外接矩形模型 连通块 零件定位

# 问题的重述

## 问题的背景

在“智能制造工程”的背景下，智能制造成为了一个热点问题。本题目讨论研究机械加工零件在自动化生产线加工过程中零件位置识别问题。在题目提供的3个图像数据附件中，DATA1为机械手抓取零件后所放的标准位置；DATA2为机械手抓取零件前需要计算机自动识别出的单个零件图像轮廓数据；DATA3为机械手抓取零件前需要计算机自动识别出的多个零件图像轮廓数据。拍摄相机的位置、焦距等属性保持不变，零件放置于同一高度的平面。

## 问题的提出

（1）根据DATA2，建立模型，计算出给定零件的位置坐标，尝试找到多种模型或算法，评价各种算法的计算速度是否快速高效，并给出每种方法的识别时间。

（2）根据已给出数据DATA3，建立模型，识别多个零件显示在同一图像时不同零件的位置。

（3）讨论分析能否运用上述方法或模型识别其他形状的零件。建议自己寻找或设定某种特定形状的零件轮廓图进行识别验证，分析所提供模型或方法的通用性。

# 模型的假设

（1）忽略相机镜头畸变对零件位置坐标计算产生的误差。

（2）忽略零件的高度对计算零件坐标产生的影响。

（3）假设在零件的标准位置时，零件的对称轴与轴平行。

（4）像素点的坐标值均取整数。

（5）假设零件内部圆形轮廓均为正圆。

（6）假设各零件规格相同。

（7）假设多个零件之间只会存在边缘的接触，不存在零件的重叠。

（8）忽略在上下0.5%以内的误差。

# 符号说明

|  |  |
| --- | --- |
| 符号 | 说明 |
|  | 代表零件 |
|  | 零件内部大圆的 |
|  | 圆左侧小圆的圆心 |
|  | 圆上方小圆的圆心 |
|  | 圆右侧小圆的圆心 |
|  | 直线的倾斜角 |
|  | 零件相对标准位置旋转的角度 |
|  | 直线的斜率 |
|  | 圆心之间的距离 |
|  | 最小外接矩形的顶点 |
|  | 线段的中点 |

# 模型的建立与求解

## 问题一

### 问题一的分析

问题一要求我们建立模型计算出给定零件的坐标位置，提供多种模型或算法，评价是否高效，给出识别的时间。

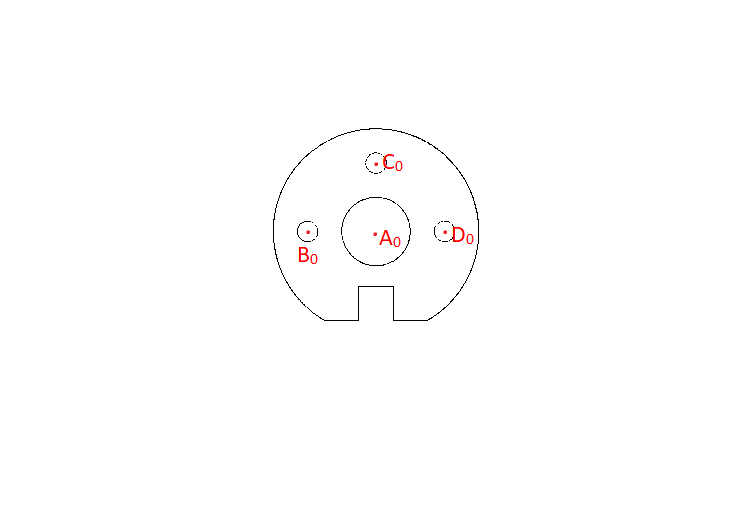
中心定位模型：首先对图像进行均值处理，补充图像中缺失的点；再进行广度优先搜索，识别出图像中的连通块，通过各块的特征找到零件内部三个小圆和一个大圆轮廓所在的连通块；计算大圆圆心的坐标确定零件的位置，再通过小圆的圆心坐标确定零件旋转的角度。

最小面积外接矩形模型：利用MATLAB编程求得零件最小外接矩形，确定零件与圆相切的最长边，利用几何关系求得圆心坐标。再通过长边的斜率求得零件旋转的角度。

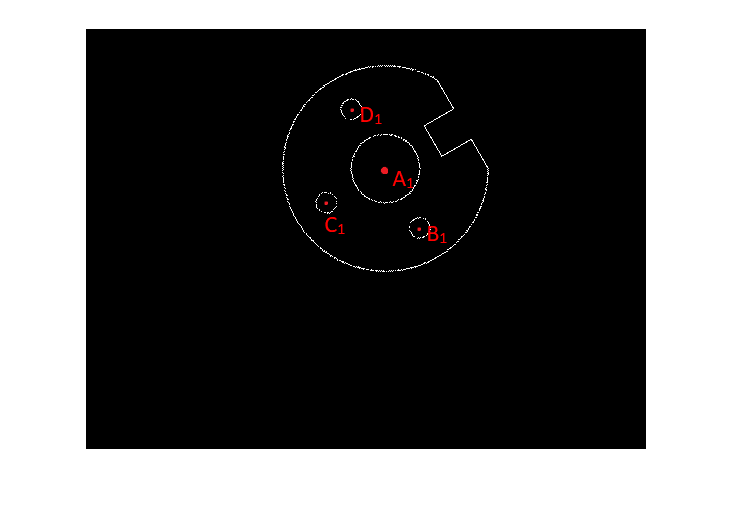
### 建模前的准备

·符号准备

利用MATLAB将DATA1、DATA2数据文件转化为零件的图像轮廓，为了方便表述，将一些特征点用符号标明，如图 4‑1、图 4‑2所示。



**P0**



**P1**

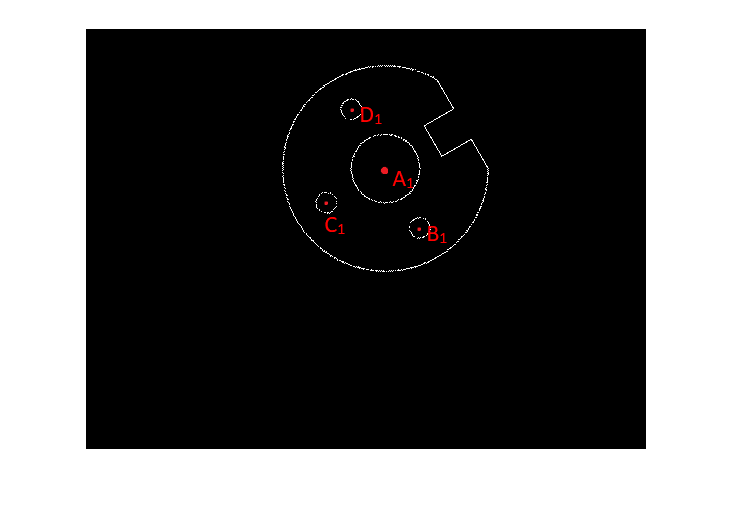
图 4‑1 DATA1识别图像

图 4‑2 DATA2识别图像

其中点表示零件内部大圆的圆心，点分别表示如图所示的三个小圆的圆心。

·坐标系准备

再以图片的左上角为原点，以正方形像素的边长为单位长度，建立平面直角坐标系，如图 4‑3所示。



y

x

图 4‑3平面直角坐标系的建立

由此，将零件的位置用圆心A的坐标和零件相对于标准位置（如图 4‑1所示）逆时针旋转的角度来表示。

·名词的解释

广度优先搜索又称BFS，属于一种盲目搜索法。其特性是不考虑算法搜索的结果，彻底地搜索整张图。BFS的基本思想是：首先从起始点出发，对于，访问每一个的未访问过的邻接点，然后依次访问的所有未访问过的临接点，以此类推，直至图中所有的顶点都被访问过为止。

在图中，一部分相互连通的区域称为连通块，基于此，我们定义任意两点之间的曼哈顿距离为2的像素点即为连通。利用BFS算法思想，遍历全图的点作为起始点，寻找该起始点所在的连通块中的像素点，并做标号和数量统计，算法实现参见附录2。

### 中心定位模型

·定位轮廓

首先对图像进行均值处理，补充图像轮廓上缺失的点，使图中的白色像素点相互连接，形成连通块，便于对轮廓的定位，处理前后图像的对比效果如图 4‑4图 4‑5所示。

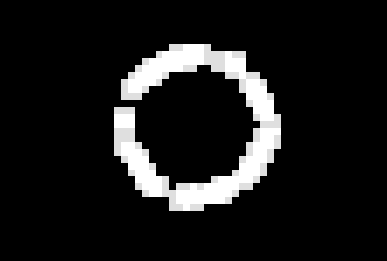
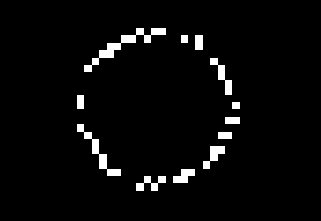


图 4‑4处理前DATA2部分图像

图 4‑5均值处理后DATA2部分图像

再对均值化处理后的图像，进行广度优先搜索，求得图像中各个连通块中白色像素点的个数，如表 4‑1所示。

表 4‑1DATA2各连通块白色像素点个数

|  |  |
| --- | --- |
| 连通块 | 个数 |
| 1 | 2626 |
| 2 | 788 |
| 3 | 221 |
| 4 | 220 |
| 5 | 220 |

分析上表数据，连通块3、4、5包含的白色像素点个数相差不大，连通块1包含的数量明显高于其他连通块，连通块2包含的数量处于上述两部分之间。由此判断，连通块1表示零件的外部轮廓。连通块2表示圆的轮廓，连通块3、4、5表示圆的轮廓（顺序未知）。

·定位圆心

对于一个圆形轮廓的连通块，利用十字扫描定点的方法确定圆心坐标。首先识别出连通块最上方的像素点和最下方的像素点。在的范围内进行遍历，假设当时左右两像素点之间的距离最大，则该圆的圆心坐标为

用上述方法求得2、3、4、5三个连通块圆心的坐标，如表 4‑2所示。

表 4‑2连通块圆心坐标

|  |  |
| --- | --- |
| 连通块 | 圆心坐标 |
| 2 | （299,140） |
| 3 | （333,200） |
| 4 | （266,80） |
| 5 | （243,173） |

点为点的中点，则有线段的中点坐标公式，

|  |  |  |
| --- | --- | --- |
|  |  | ( 1 ) |

观察上述坐标点，计算 3、4两连通块圆心的中点为，由于像素点坐标为整数，因此可以视作中点即点。因此点的坐标为、，的顺序未知，点坐标为。

·旋转角度的计算

直线的倾斜角为，

|  |  |  |
| --- | --- | --- |
|  |  | ( 2 ) |

倾斜角和零件旋转角度的示意图，如图 4‑6所示。

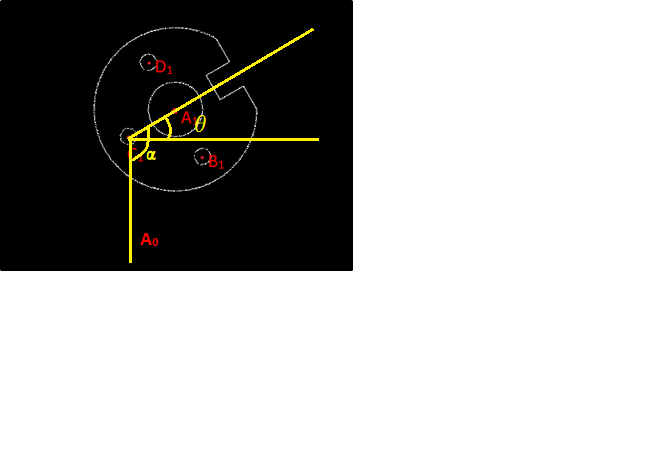


图 4‑6旋转角示意图

因此零件的逆时针旋转角度，

|  |  |  |
| --- | --- | --- |
|  |  | ( 3 ) |

计算得到直线的倾斜角为，零件相对于标准位置逆时针旋转角度的值为。

综上，零件中心的坐标为，相对于标准位置逆时针旋转角度为。

### 最小面积外接矩形模型

·定位外轮廓圆圆心

首先利用MATLAB编程，求得零件面积最小的外接矩形，如图 4‑7所示。

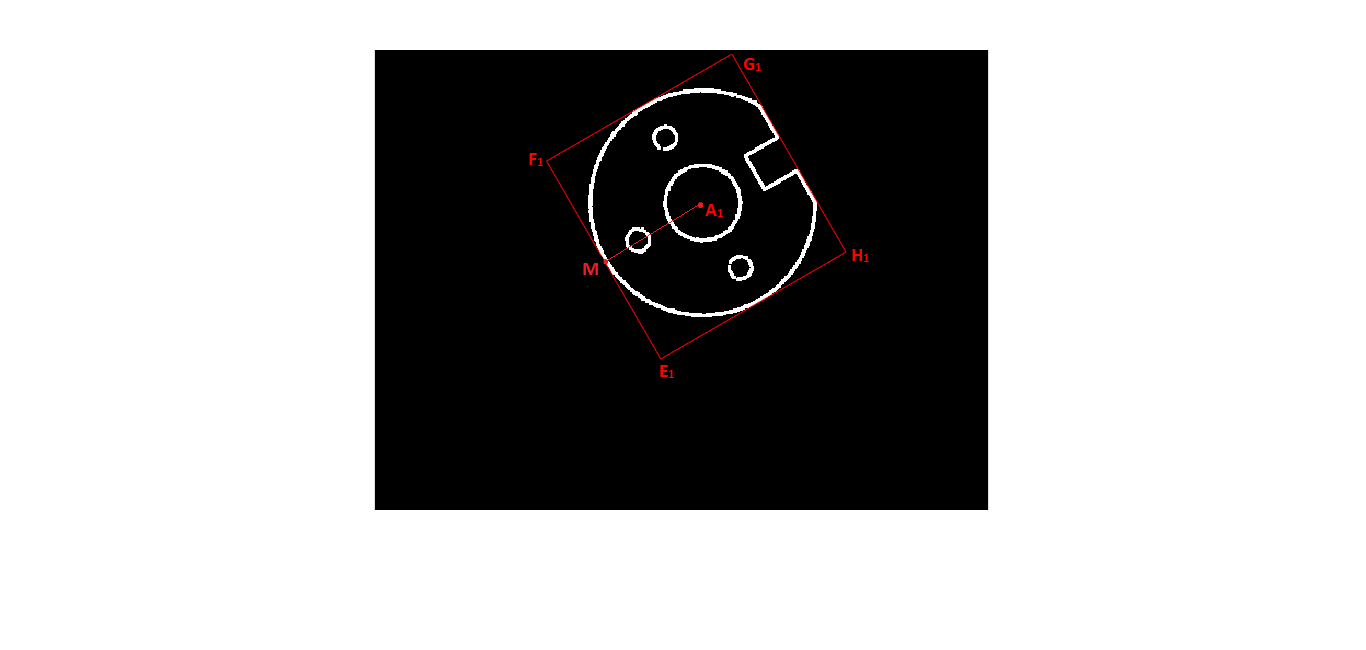
****

图 4‑7最小面积外接矩形

计算矩形的边长，找到两条较长边，用MATLAB计算两条长边上白色像素点的个数，像素点较少的一条即边。设点坐标为，点坐标为，则点横纵坐标，

|  |  |  |
| --- | --- | --- |
|  |  | ( 4 ) |

下面求零件外轮廓圆圆心点的坐标。由于线段与线段长度相等且相互垂直，通过、、三个点的几何关系，构造如图 4‑8所示的全等三角形。

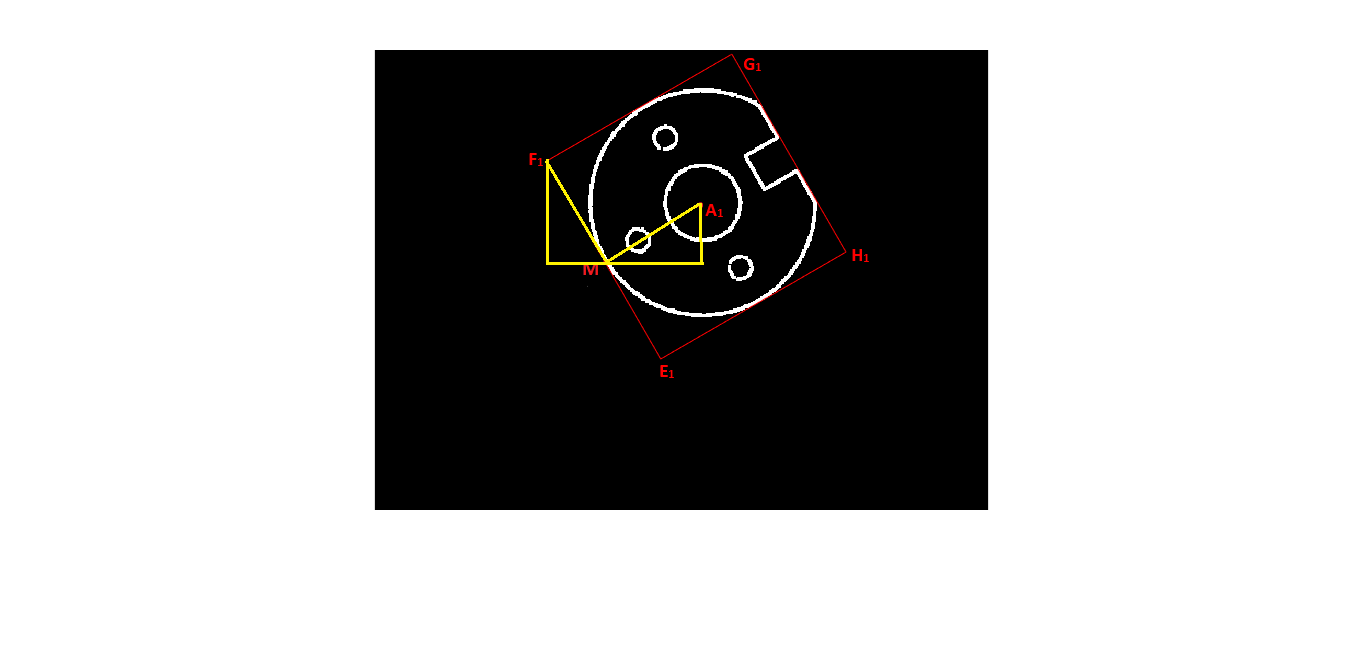


图 4‑8几何关系

外轮廓圆圆心的坐标，

|  |  |  |
| --- | --- | --- |
|  |  | ( 5 ) |

·同心圆的证明与圆心坐标的计算

由于零件的位置是用零件内部大圆圆心的坐标来表示的，因此下面证明外轮廓圆心和零件内部大圆圆心重合，即证明该零件是一个同心圆。

对位于标准位置的零件进行广度优先搜索，得到各连通块包含的白色像素点如表 4‑3所示。

表 4‑3各连通块包含像素点数量

|  |  |
| --- | --- |
| 连通块 | 像素点数量 |
| 1 | 2436 |
| 2 | 751 |
| 3 | 214 |
| 4 | 205 |
| 5 | 203 |

分析上述数据，连通块1包含的像素点明显多于其他连通块，因此为零件外轮廓连通块。连通块2包含的像素点数量多于3,4,5连通块，少于连通块1，因此为零件内部大圆。

运用方法一中的遍历法定位外轮廓圆的圆心和零件内部大圆的圆心，分别用红色十字和绿色十字表示，定位效果图如图 4‑9所示。

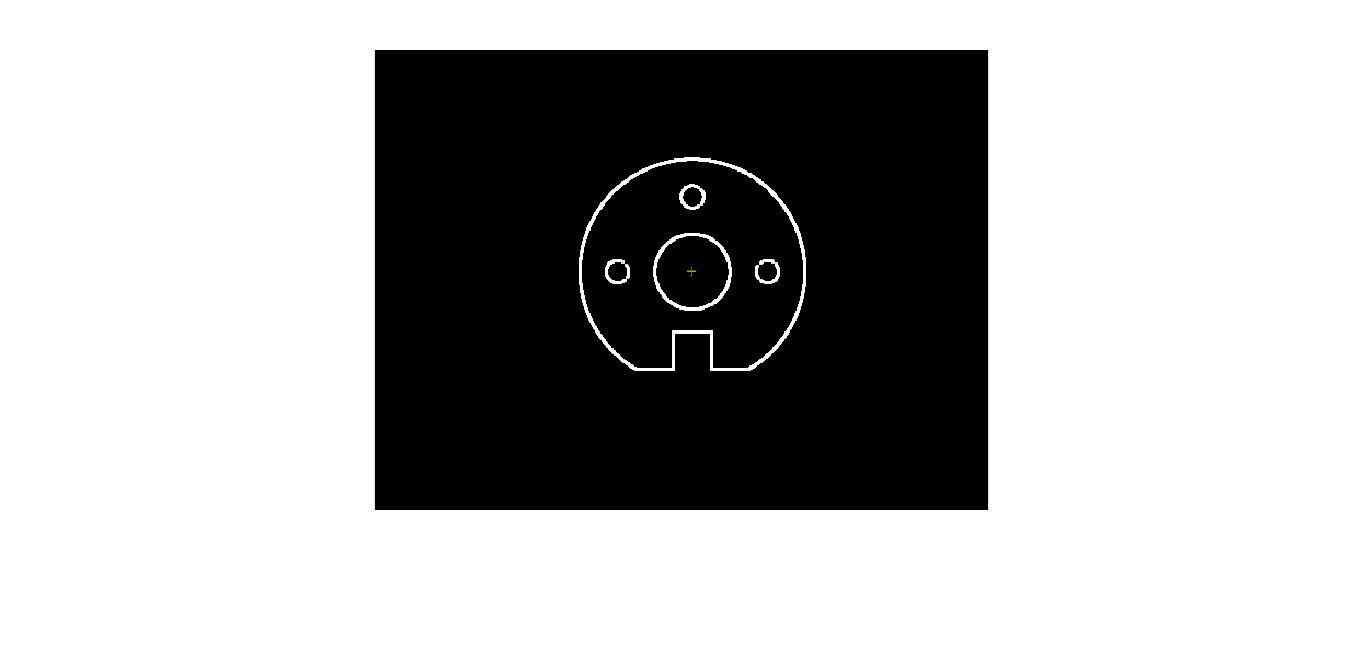


图 4‑9两圆心定位效果图

从图中可以用肉眼观察到两圆心的位置重合，计算两圆心的具体坐标分别为两坐标在误差范围内相等，因此确定该零件为同心圆。

通过MATLAB计算得到点坐标为，点的坐标为，代入公式（4）计算得点坐标为。再将点和点的坐标代入公式（3）得到圆心点的坐标为。

·旋转角度的计算

通过、点坐标求得直线的斜率为，

|  |  |  |
| --- | --- | --- |
|  |  | ( 6 ) |

则直线的倾斜角为，

|  |  |  |
| --- | --- | --- |
|  |  | ( 7 ) |

倾斜角和零件旋转角度的关系如图 4‑10所示。

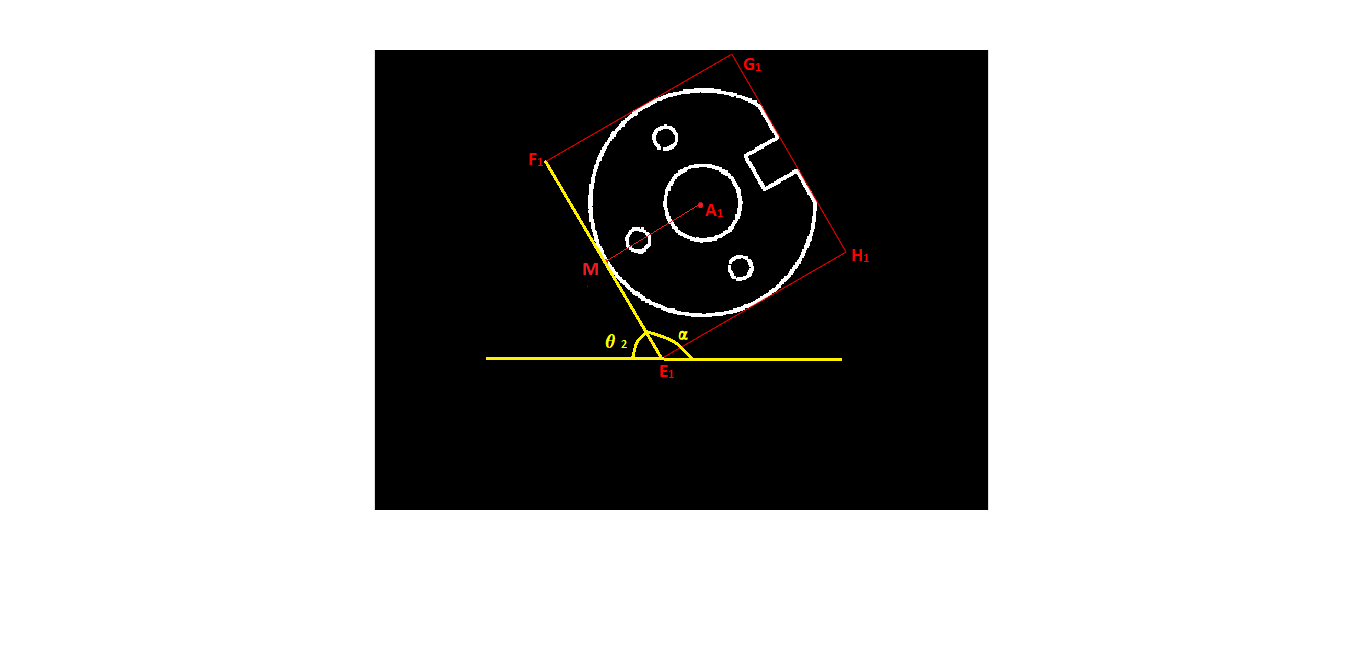


图 4‑10倾斜角和零件旋转角度的关系图

因此，零件旋转的角度为

|  |  |  |
| --- | --- | --- |
|  |  | ( 8 ) |

通过公式（6）求得直线的斜率为，代入公式（8）计算的零件的旋转角度为。

综上，对点的坐标取整得零件中心的坐标为，相对于标准位置逆时针旋转角度为。

### 效率的评价

为了能够估算模型算法需要的运行时间，定量分析评价两种算法的运行效率，我们引入了时间复杂度这一概念。

在计算机科学中，算法的时间复杂度是一个函数，它定性描述该类算法的运行时间。[1]时间复杂度可以用来评判同一问题下，不同模型算法的效率。

方法一建立的中心定位模型，总体可分为两部分：第一部分是连通块的求解，第二部分是各个圆心定位和角度的计算。

方法二建立的最小外接面积矩形法模型，同样也可以分为两部分：第一部分是确定最小面积矩形，求解矩形的角的坐标，第二部分是建立一次函数进行圆心定位和角度的计算。

方法一当中，求解连通块的方法，本质上是一种广度优先搜索（BFS）的变形，本题二值图像的存储方式是二维数组，存储大小为。我们可以将其扩充为的图像，看作是个点的无向完全图。当用二维数组表示邻接矩阵作为图的存储结构时，查找每个点的邻接点所需要的时间为，为图中顶点个数。

方法二当中，求解最小外接面积矩形法，其使用的旋转卡壳法时间复杂度为，为多边形的边数。

在的值相等情况下，由上时间复杂度可知，方法一的时间消耗是以平方级增长，方法二的时间消耗是以线性对数级增长，方法二的运行效率更好。

MATLAB每次运算时间主要由算法的复杂程度决定，还会受到电脑状态的影响，为了精确测量每个模型的运算时间，将每个算法进行十次运算，取运算时间的平均值，如表 4‑4所示。

表 4‑4模型所需时间

|  |  |  |
| --- | --- | --- |
| 模型时间  序号 | 圆心定位模型 | 最小外接矩形模型 |
| 1 | 0.010853 | 0.0025203 |
| 2 | 0.0086882 | 0.0020378 |
| 3 | 0.0089596 | 0.0021854 |
| 4 | 0.0095654 | 0.0024201 |
| 5 | 0.011453 | 0.0021359 |
| 6 | 0.010749 | 0.0022332 |
| 7 | 0.011296 | 0.0021227 |
| 8 | 0.010705 | 0.0021082 |
| 9 | 0.012262 | 0.002054 |
| 10 | 0.012007 | 0.0019285 |
| 平均时间 | 0.01065382 | 0.00217461 |

由上表数据，圆心定位模型平均时间为,最小外接矩形模型为

将原有图像放大至100倍后，运行效率如表 4‑5所示。

表 4‑5图像放大后各模型运行效率

|  |  |  |
| --- | --- | --- |
| 模型时间  序号 | 中心定位模型 | 最小外接矩形模型 |
| 1 | 0.67625 | 0.036681 |
| 2 | 0.66244 | 0.0305 |
| 3 | 0.61935 | 0.031613 |
| 4 | 0.66639 | 0.036545 |
| 5 | 0.63401 | 0.029164 |
| 平均时间 | 0.651688 | 0.0329006 |

可以发现，方法二的运行效率是方法一的近20倍。

## 问题二

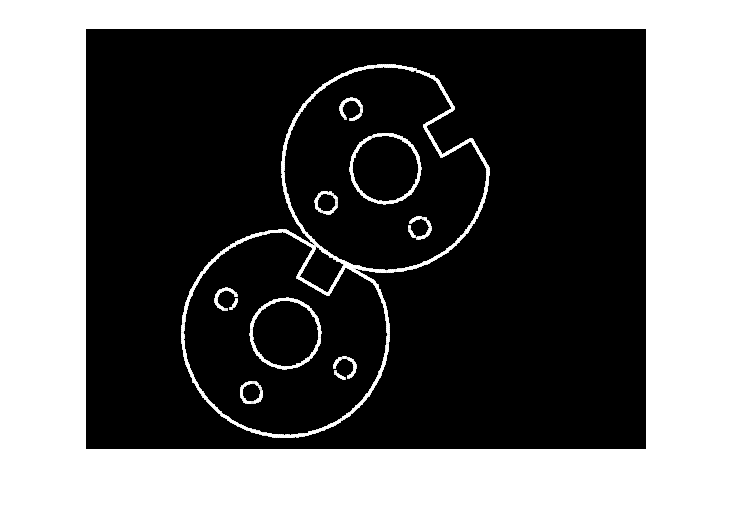
### 问题的分析

问题二需要识别多个零件的位置，首先需要识别哪几个圆位于同一个零件上。分别计算出每个小圆圆心和大圆圆心之间的距离，每个小圆和离自身最近的大圆属于同一个零件。再延续问题一中心定位模型的思路，通过确定各个零件内部四个圆的圆心坐标，得出各个零件的位置即旋转角度。

### 零件间的区分

首先将DATA3数据文件进行均值处理，补充缺失的点，使之便于连通块的计算，利用MATLAB绘制出图像轮廓，并建立平面直角坐标系，如图 4‑11所示。

x



y

图 4‑11 DATA3均值处理图

对图形进行广度优先搜索，求得各连通块所包含的白色像素点的个数，表 4‑6所示。

表 4‑6 DATA3各连通块白色像素点个数

|  |  |
| --- | --- |
| 连通块 | 个数 |
| 1 | 5236 |
| 2 | 789 |
| 3 | 788 |
| 4 | 221 |
| 5 | 220 |
| 6 | 220 |
| 7 | 220 |
| 8 | 219 |
| 9 | 215 |

分析上表数据，连通块1包含的白色像素点的个数远大于其他连通块，连通块2、3白色像素点的数量相差不大，且高于除1外的其他六个连通块。由此可得，连通块1属于两个相邻零件的外部轮廓，连通块2和3表示零件内部两个大圆的轮廓，连通块4-9表示两个零件内部六个小圆的轮廓。

利用问题一中心定位模型中的遍历法，定位各圆的圆心坐标，如表 4‑7所示。

表 4‑7 DATA3各连通块圆心坐标

|  |  |
| --- | --- |
| 连通块 | 圆心坐标 |
| 2 | (201,305) |
| 3 | (299,140) |
| 4 | (333,200) |
| 5 | (266,80) |
| 6 | (243,173) |
| 7 | (141,272) |
| 8 | (166,365) |
| 9 | (258,338) |

分别计算各小圆的圆心和两个大圆圆心之间的距离，

|  |  |  |
| --- | --- | --- |
|  |  | ( 9 ) |

其中，表示小圆连通块，表示零件编号，表示小圆圆心和零件内部大圆圆心之间的距离，。

小圆与离它最近的大圆属于同一个零件，即

|  |  |  |
| --- | --- | --- |
|  |  | ( 10 ) |

其中，分别表示两个大圆连通块——连通块2、连通块3，所在的零件。

将数据代入计算得到各小圆圆心和大圆圆心之间的距离如表 4‑8所示。

表 4‑8小圆圆心和大圆圆心距离

|  |  |  |
| --- | --- | --- |
| 小圆连通块 | 与连通块2圆心距 | 与连通块3圆心距 |
| 4 | 168.6683 | 68.96376 |
| 5 | 234.2008 | 68.47627 |
| 6 | 138.5208 | 65 |
| 7 | 68.47627 | 205.8835 |
| 8 | 69.46222 | 261.3695 |
| 9 | 65.8635 | 202.2004 |

由上表数据可知，7,8,9与2四个连通块属于零件；4,5,6与3四个连通块属于零件。

### 旋转角度的确定

根据中点坐标公式(1)，找到零件内部小圆的圆心坐标（顺序未知），从而得到圆的圆心坐标。具体数据如表 4‑9所示。

表 4‑9各零件坐标数据

|  |  |  |
| --- | --- | --- |
| 零件 | P2 | P3 |
| A点坐标 | (201,305) | (299,140) |
| B、D点坐标 | (141,272)(258,338) | (333,200)(266,80) |
|
| C点坐标 | (166,365) | (243,173) |

再通过点和点的坐标，利用公式（2）求得直线的倾斜角，代入公式（3）求得零件逆时针旋转的角度。

将零件特征点的坐标数据代入公式计算得到，相对于标准位置逆时针旋转的角度为，零件相对于标准位置逆时针旋转的角度为。

### 零件位置的描述

综上所述，零件中心的坐标为，相对于标准位置逆时针旋转的角度为；零件中心的坐标为，相对于标准位置逆时针旋转的角度为。

## 问题三

### 问题三的分析

问题三需要讨论之前建立的模型是否试用于其他形状的零件，即讨论模型的普适性。我们取不同形状特征的零件利用问题一、二中建立的两种模型进行识别分析，最后进行总结，探究每种模型的能够识别的零件特点。

### 中心定位模型的探究

中心定位模型的适用对于零件图像连通块的数量和形状均有要求，首先对零件内部轮廓的形状的要求进行探究。

#### 连通块形状的要求

·矩形

从相对简单的矩形开始。确定零件内部矩形的标准位置如图 4‑12所示。

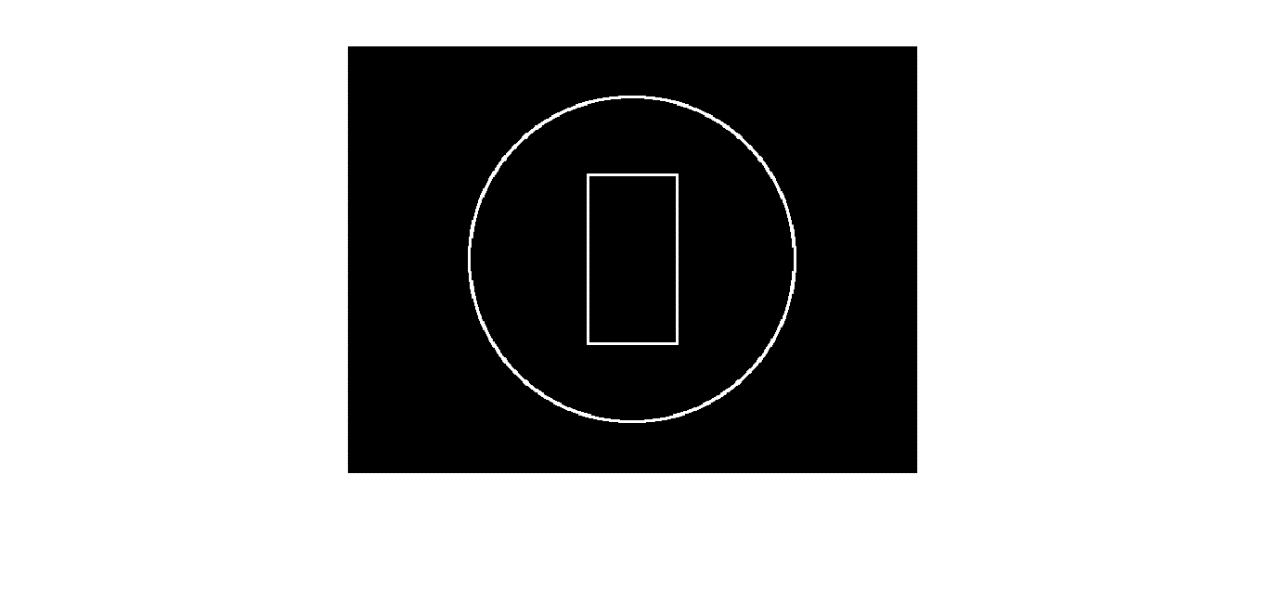


图 4‑12矩形标准位置

利用问题一中的中心定位模型，识别出图像中的连通块，由于零件内部矩形和外部圆形轮廓之间的周长存在差异，由此找到矩形所对应连通块。利用十字扫描定点法能够较为容易地求得标准位置矩形的中心坐标为

将零件绕着矩形的中心点旋转任意角度，如图 4‑13所示。

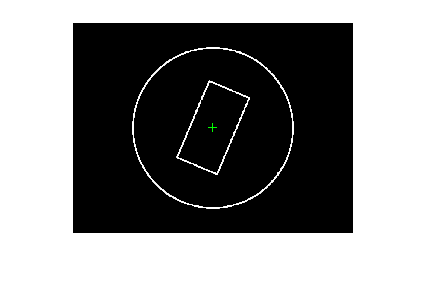


图 4‑13零件旋转后的位置

在确定表示矩形的连通块后，利用十字扫描定点法求矩阵的中心点。首先找到矩形连通块最上方和最下方的像素点，在两点纵坐标的范围内进行扫描。由于在两平行线所夹线段的长度保持不变，因此取到上下两个距离发生改变的两个临界点，用中点公式取两个临界点的中点，即矩形的中心点。中心点位置如上图绿色十字所示。

利用MATLAB求得图片的数据后，将数据代入模型计算，得到矩形中心点的坐标为，在误差范围内求解准确。因此中心定位模型对矩形使用。

·平行四边形

确定平行四边形的标准位置，如图 4‑14所示。

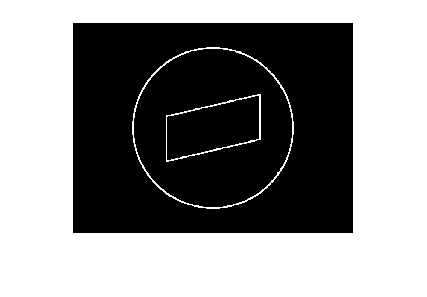


图 4‑14平行四边形标准位置

利用中心定位模型求得标准位置平行四边形的中心坐标为。将零件绕平行四边形的中点旋转任意的角度，得到零件的图像如图 4‑15所示。

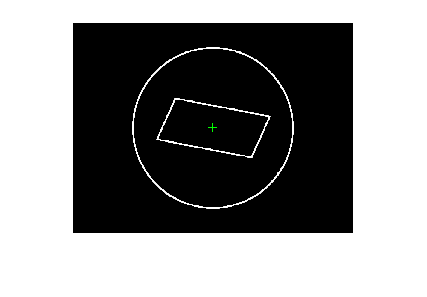


图 4‑15零件旋转后的位置

首先确定平行四边形所在的连通块。由于平行四边形的边两两平行，因此能够通过与矩形相同的方法，取线段长度发生变化的临界点，求两点的中点坐标即平行四边形的中心坐标。中心点的具体位置如上图绿色十字所示。

将图像数据代入模型求解得到，旋转后平行四边形的中心坐标为，与标准位置的中心坐标保持一致，因此中心定位模型对平行四边形也能进行准确的求解。

·正六边形

首先确定正六边形的标准位置，如图 4‑16所示。

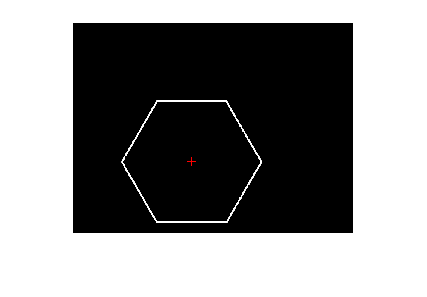


图 4‑16正六边形标准位置

用中点定位模型求正六边形的中心，如上图红色十字所示，中心坐标为。

将正六边形绕中心点旋转任意角度，得到图形如图 4‑17所示。

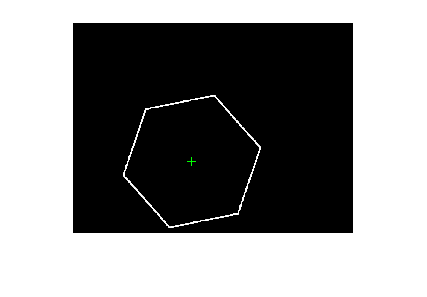


图 4‑17零件旋转后的位置

由于正六边形同样存在相互平行且相等的边，因此同样可以采用与矩形相同的十字扫描定点法求中心点的坐标。求得中心点如上图绿色十字所示，中心点坐标为。与在标准位置时的中心坐标在误差范围内相等，因此本模型对正六边形也能较好地使用。

·适用形状的总结

结合对上述三种不同形状图形的分析，能够得出该模型适用于存在平行且相等边的多边形，如，平行四边形、边数为偶数的正多边形，根据问题一的求解，发现该模同样适用于圆形。

### 连通块数量的要求

·1-2个连通块

在满足形状要求的前提下，对于只有一个连通块的零件，中心定位模型能够找到其中心，但是无法通过一个中心坐标确定零件旋转的角度。因此，此模型不能适用于1个连通块时的情况。

对于有两个连通块的零件，若两个连通块的中心不重合，则可以通过两连通块中心连线的斜率计算零件旋转的角度；若两连通块中心重合则与一个连通块的情况相同，无法确定零件旋转的角度。

·3个连通块

对于三个连通块的情况，假设零件处于标准位置时如图 4‑18所示。

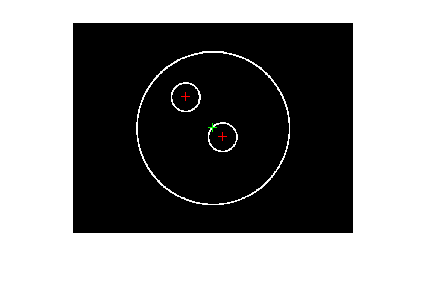


图 4‑18三个连通块零件标准位置

计算可得标准位置零件外轮廓的中心坐标，以及内部两个连通块的中心坐标。用绘图软件将零件绕着外轮廓中心点逆时针旋转 。得到零件图，如图 4‑19所示。

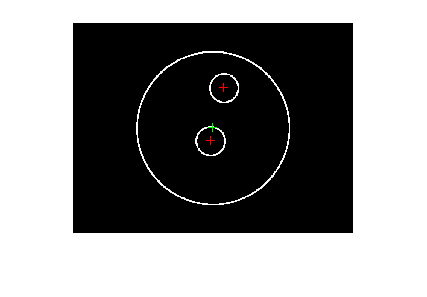


图 4‑19旋转后的零件图

用广度优先搜索统计各连通块白色像素点的数量，区分出零件外部轮廓连通块和内部连通块。

利用十字扫描求点法求旋转后零件各连通块的圆心坐标。

标准位置以及所求位置的坐标数据如表 4‑10所示。

表 4‑10连通块坐标

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 坐标 | 标准位置 | | 所求位置 | |
| x | y | x | y |
| 外轮廓中点 | 280 | 210 | 280 | 210 |
| 内部连通块中点 | 225 | 148 | 302 | 130 |
| 299 | 228 | 275 | 236 |

计算零件内部两连通块中心点连线在标准位置以及所求位置的斜率，计算零件的旋转角，

|  |  |  |
| --- | --- | --- |
|  |  | ( ) |

将数据代入计算得零件旋转的角度为，与预先设置的在误差范围内相等。因此本模型适用于3个连通块的情况。

·4个以上连通块

对于四个以上连通块的情况，假设零件的标准位置如图 4‑20所示，

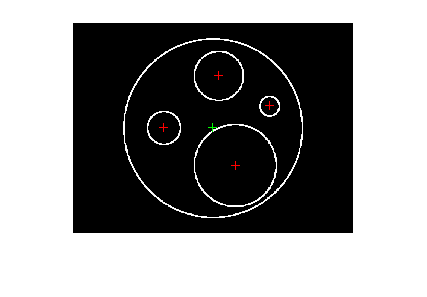


图 4‑20四个连通块标准位置

将零件绕着外轮廓中心点旋转，得到零件图如图 4‑21所示。

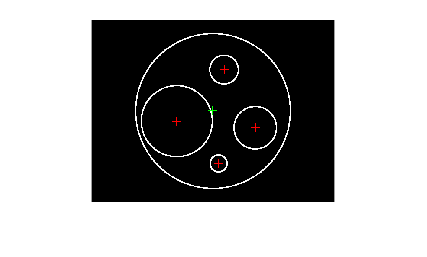


图 4‑21旋转后的零件图

假设零件内部各连通块所包含的白色像素点数量不能对连通块进行较好的区分。利用遍历算法，分别求出标准位置和旋转后位置零件内部各连通块中心之间的距离，比较两组数据，找出旋转后和标准位置零件内部连通块中心点的对应关系。通过求任意两点连线在标准位置和旋转后位置的斜率，得到零件旋转的角度。

旋转前后零件的坐标数据如表 4‑11所示。

表 4‑11旋转前后零件的坐标数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 坐标 | 标准位置 | | 所求位置 | |
| x | y | x | y |
| 外轮廓中点 | 280 | 210 | 280 | 210 |
| 内部连通块中点 | 292 | 106 | 306 | 115 |
| 393 | 166 | 197 | 234 |
| 182 | 210 | 378 | 249 |
| 325 | 285 | 293 | 331 |

分别求同一位置下零件内部连通快中心和其他连通块中心之间的距离找到标准位置和所求位置对应的两组点，计算两个点所确定线段斜率的变化求得零件旋转的角度。

代入数据计算得到旋转角度为，在误差范围内与先前设定的角度相等。

·适用连通块数量的总结

中心定位模型适用于零件连通块数量不小于两个的情况，其中当零件数量等于两个时，要求零件外轮廓的中心与内部连通块的中心不相重合。

### 最小面积外接矩形模型的探究

由于最小面积外接矩形将零件的进行了唯一的框定，零件的位置可以转化为矩形中心点的坐标和矩形旋转的角度。假设某不规则零件旋转前后的图像如下图所示，

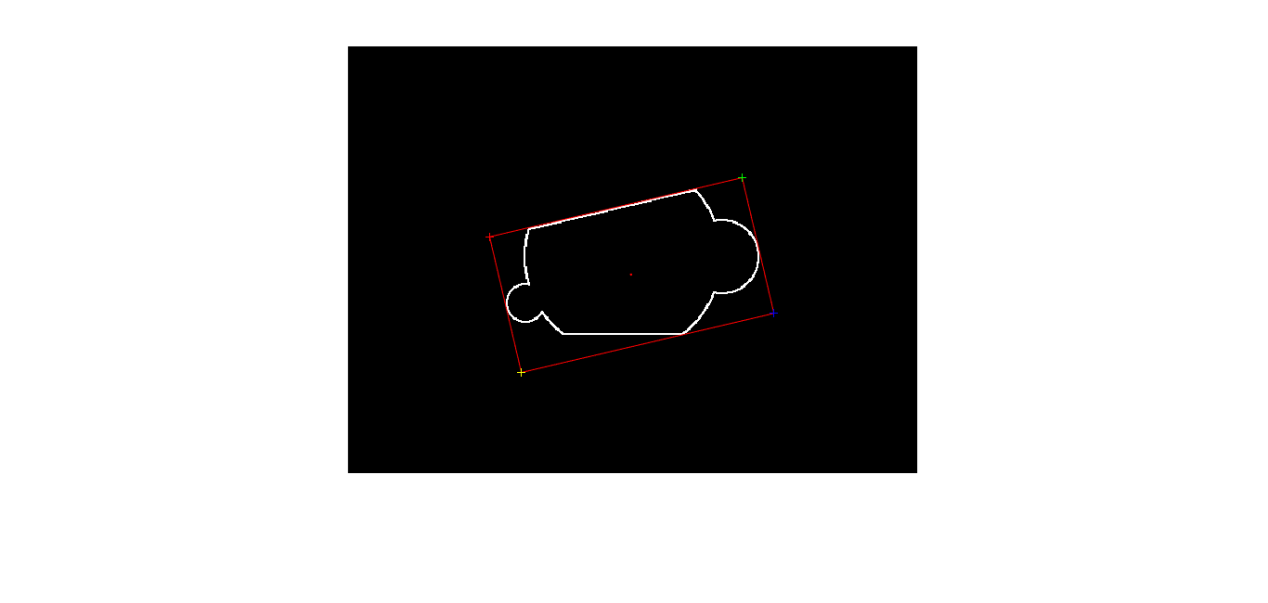
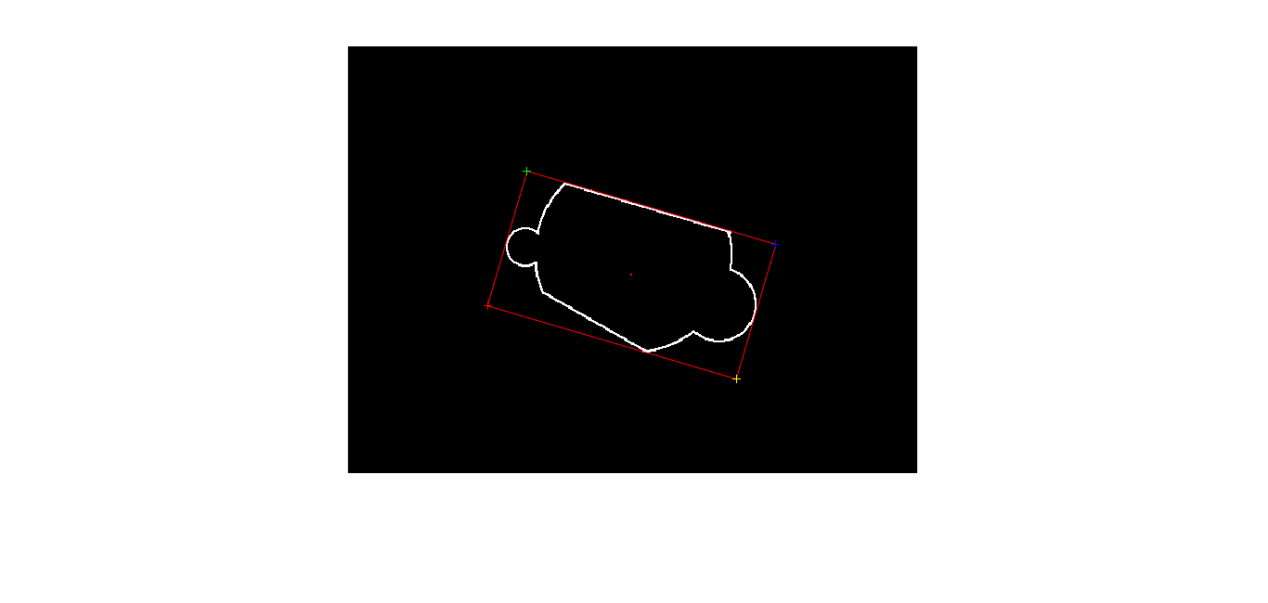


图 4‑23所求位置图像

图 4‑22标准位置图像

对于矩阵的中心点，能够通过求对角线交点的方式进行求解；对于旋转角的计算，需要识别出矩形旋转前后对应的边，通过边的斜率计算零件旋转的角度。因此本模型能否适用取决于能否识别出矩形旋转前后对应的边。若图像上零件的轮廓用白色像素点表示，通过计算矩形每条边上白色像素点的个数识别旋转前后相同的边。

将每条边上像素点的数量用字母表示，下面对边上像素点个数分布的所有情况进行讨论分析：

（1）型

当矩形四条边上像素点数量相等时，无法将旋转前后的边进行对应，因此本模型不适用于此情况下的求解。

（2）型

当矩形两组邻边上的像素点数量分别相等时，能够确定邻边组成直角的顶点坐标，通过标准位置和所需计算位置对角线的斜率，求得零件旋转的角度。因此本模型适用于这种情况。

（3）型

当矩形两组对边上的像素点分别相等，则零件绕矩形中心点旋转前后的状态体现在外接矩形上是相同的。因此不能旋转后的边和标准位置对应起来，模型不适用于这种情况。

（4）型、型、型

这三种情况下，至少有一条边和矩形其余边上的像素点数量不同，因此可以通过这条边上像素点的数量确定标准位置上对应的边。通过计算这条边的斜率求得零件旋转的角度。因此模型使用于这三种情况。

综上，本模型适用于最小外接矩形至少有一条边上的白色像素点数量区别于其余边，以及矩形两组邻边上白色像素点数量分别相等的情况。

# 模型的评价与推广

## 模型的优点

（1）问题一中建立了多种不同的数学模型，从多条思路对问题进行了分析求解，各方法求得的答案相近，求解准确。

（2）利用了识别图中连通块的方法对零件内部各圆进行定位，较为准确地求出了各个连通块圆心的坐标。

（3）在问题一的最小面积外接矩形模型中，将不规则零件的位置和旋转角度巧妙地转化为求最小面积外接矩形的位置和相对于标准位置旋转的角度，简化了模型的求解。

## 模型的缺点

（1）利用中心定位模型对问题一进行计算时效率不高，程序运行的时间较长。

（2）中心定位模型只对特定形状的连通块适用，具有一定的局限性。

（3）最小面积外接矩形模型，对矩形边上轮廓像素点数量的分布有要求，对于无法通过像素点数量确定与标准位置对应边的情况具有局限性。

## 模型的推广

本文建立的中心定位模型和最小面积外接矩形模型各有各的特点，均能够较为准确地对零件进行定位。如果将两者相结合，就能够对现实生活中绝大多数情况下的零件进行定位。本模型为实际生活中零件的定位提供了新的思路，具有一定的现实意义。

# 参考文献

[1]维基百科编者. 时间复杂度[G/OL]. 维基百科, 2018(20180622)[2018-06-22]. <https://zh.wikipedia.org/w/index.php?title=%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6&oldid=50080081>.

[2]刘玉珍,刘润涛.简单多边形的最小外接矩形算法[J].哈尔滨理工大学学报,2008(02):5-7.

[3]金文华,何涛,刘晓平,唐卫清,唐荣锡.基于有序简单多边形的平面点集凸包快速求取算法[J].计算机学报,1998(06):533-539.

**附录**

**附录1**

**BFS时间测算**

clc;

clear;

load out2.txt;

a=out2;

dx=[1;1;1;0;0;-1;-1;-1;2;0;0;-2];

dy=[1;-1;0;1;-1;1;-1;0;0;2;-2;0];

for i=1:420

for j=1:560

v(i,j)=0;

end

end

tic;

point=[];% 1为序号 2为个数

for i=1:20

point(i,1)=i;

end

id=0;

for i=1:420

for j=1:560

if a(i,j)==1&&v(i,j)==0

id=id+1;

tail=1;

head=0;

queue(tail,1)=i;

queue(tail,2)=j;

while head~=tail

head=head+1;

for k=1:16

cur\_x=queue(head,1)+dx(k,1);%移动x

cur\_y=queue(head,2)+dy(k,1);%移动y

if a(cur\_x,cur\_y)~=1||v(cur\_x,cur\_y)~=0

continue ;%可以访问 已经访问

end

if cur\_x<0||cur\_x>420||cur\_y<0||cur\_y>560

continue ;%当前坐标是否超出界线

end

v(cur\_x,cur\_y)=id;

tail=tail+1;

queue(tail,1)=cur\_x;

queue(tail,2)=cur\_y;

end

end

end

end

end

cnt=[zeros(20,1)];

for i=1:420

for j=1:560

if v(i,j)~=0

cur=v(i,j);

cnt(cur)=cnt(cur)+1;

end

end

end

toc;

disp(['运行时间: ',num2str(toc)]);

**附录2**

**BFS求连通块**

#include<iostream>

#include<cmath>

#include<ctime>

#include<iomanip>

#include<algorithm>

using namespace std;

int v[500][600];

int a[500][600];

int dx[30]={0,1,1,1,0,0,-1,-1,-1,2,0,0,-2};

int dy[30]={0,1,-1,0,1,-1,1,-1,0,0,2,-2,0};

int x,y;

struct node

{

int cnt;

int num;

}ans[20];

double cur;

bool cmp(node,node);

void DFS(int,int,int);

void local\_base\_point(int);

int main ()

{

ios::sync\_with\_stdio(false);

freopen("spare.txt","r",stdin);

// freopen("out.txt","w",stdout);

for (int i=1 ; i<=420 ; i++ )

for (int j=1 ; j<=560 ; j++ )

{

cin >> cur;

// cout << cur << '\n' ;

if (cur==0)

a[i][j]=0 ;

else

{

a[i][j]=1 ;

}

}

int start=clock();

int id=0;

for (int i=1 ; i<=20 ; i++ )

ans[i].num = i ;

for (int i=1 ; i<=420 ; i++ )

for (int j=1 ; j<=560 ; j++ )

if (a[i][j]==1&&v[i][j]==0)

DFS(i,j,++id);

for (int i=1 ; i<=420 ; i++ )

for (int j=1 ; j<=560 ; j++ )

if (v[i][j]!=0)

ans[v[i][j]].cnt++ ;//连通块个数计算

sort(ans+1,ans+20+1,cmp);//降序排序

for (int i=1 ; i<=10 ; i++ )

cout << ans[i].num << ':' << ans[i].cnt << '\n' ;//输出结果展示

for (int i=1 ; i<=10 ; i++ )

{

local\_base\_point(i);//求圆心

cout << "id:" << i << ' ' << "x:" << x << ' ' << "y:" << y << '\n' ;

}

return 0 ;

}

bool cmp(node x,node y)

{

return x.cnt>y.cnt;

}

void DFS(int x,int y,int id)

{

if (x<0||x>420||y<0||y>560)

return ;//出界格子

if (v[x][y]!=0||a[x][y]!=1)

return ;//是否可行、是否访问

v[x][y]=id ;//标记编号

for (int i=1 ; i<=16 ; i++ )

DFS(x+dx[i],y+dy[i],id);

}

void local\_base\_point(int id)

{

int head = 0 ;

int tail = 0 ;

int X[700][3] ;//420行

int distance\_X[700];//head tail 距离

int distance\_Y[700];//head tail 距离

int MAX = 0 ;

for (int i=1 ; i<=420 ; i++ )

{

head = 0 ;

tail = 0 ;

for (int j=1 ; j<=560 ; j++ )

{

if (v[i][j]==id)

if (a[i][j]==1)

{

head = j ;

break ;

}

}

for (int j=560 ; j>=1 ; j-- )

{

if (v[i][j]==id)

if (a[i][j]==1)

{

tail = j ;

break ;

}

}

X[i][1] = head ;

X[i][2] = tail ;

}

for (int i=1 ; i<=420 ; i++ )

distance\_X[i] = X[i][2]-X[i][1] ;//求head tail 距离

MAX = 0 ;

//定位坐标轴x

for (int i=1 ; i<=420 ; i++ )

{

if (MAX<distance\_X[i])

{

MAX = distance\_X[i] ;

head = i ;

tail = i ;

}

if (MAX==distance\_X[i])

tail = i ;

}

x = floor((tail+head)/2);

//-------------------------------------------------------------

int Y[700][3] ; //560列

for (int i=1 ; i<=560 ; i++ )

{

head = 0 ;

tail = 0 ;

for (int j=1 ; j<=420 ; j++ )

{

if (v[j][i]==id)

if (a[j][i]==1)

{

head = j ;

break ;

}

}

for (int j=420 ; j>=1 ; j-- )

{

if (v[j][i]==id)

if (a[j][i]==1)

{

tail = j ;

break ;

}

}

Y[i][1] = head ;

Y[i][2] = tail ;

}

for (int i=1 ; i<=560 ; i++ )

distance\_Y[i] = Y[i][2]-Y[i][1] ;//求head tail 距离

MAX = 0 ;

//定位坐标轴y

for (int i=1 ; i<=560 ; i++ )

{

if (MAX<distance\_Y[i])

{

MAX = distance\_Y[i] ;

head = i ;

tail = i ;

}

if (MAX==distance\_Y[i])

tail = i ;

}

y = floor((tail+head)/2);

}

**附录3**

**最小面积矩形**

clc;

clear;

out2=imread('C:\Users\leafpigbirds\Desktop\不规则图形.bmp');

tic;

% t1=clock;

[r c]=find(out2==1);

[rectx,recty,area,perimeter]=minboundrect(c,r,'a');

figure(1);

imshow(out2);

line(rectx(:),recty(:),'color','r');

hold on;

plot(rectx(1,1),recty(1,1),'r+',rectx(2,1),recty(2,1),'g+');

[wid hei]=minboxing(rectx(1:end-1),recty(1:end-1));

%Slope斜率判断方位

point=[];%1为x,2为y

for i=1:4

point(i,1)=rectx(i,1);

point(i,2)=recty(i,1);

end

%1-2

dis(1,1)=sqrt((point(1,1)-point(2,1))^2+(point(1,2)-point(2,2))^2);

%2-3

dis(2,1)=sqrt((point(2,1)-point(3,1))^2+(point(2,2)-point(3,2))^2);

%3-4

dis(3,1)=sqrt((point(3,1)-point(4,1))^2+(point(3,2)-point(4,2))^2);

%4-1

dis(4,1)=sqrt((point(4,1)-point(1,1))^2+(point(4,2)-point(1,2))^2);

%斜率计算

cnt1=0;

cnt2=0;

cur=0;%当前位置

k12=(point(1,2)-point(2,2))/(point(1,1)-point(2,1));

k23=(point(2,2)-point(3,2))/(point(2,1)-point(3,1));

b12=-k12\*rectx(1,1)+recty(1,1);

b34=-k12\*rectx(3,1)+recty(3,1);

for i=floor(rectx(1,1)):floor(rectx(2,1))

cur=k12\*i+b12;

% hold on;

% plot(i,floor(cur),'b.',i,floor(cur+2),'b.');

if out2(floor(cur),i)==1

cnt1=cnt1+1;

end

if out2(floor(cur+2),i)==1

cnt1=cnt1+1;

end

if out2(floor(cur-2),i)==1

cnt1=cnt1+1;

end

end

for i=floor(rectx(3,1)):-1:floor(rectx(4,1))

cur=k12\*i+b34;

% hold on;

% plot(i,floor(cur),'b.',i,floor(cur+2),'b.');

if out2(floor(cur),i)==1

cnt2=cnt2+1;

end

if out2(floor(cur+2),i)==1

cnt2=cnt2+1;

end

if out2(floor(cur-2),i)==1

cnt2=cnt2+1;

end

end

mid\_x=floor((rectx(3,1)+rectx(4,1))/2);

mid\_y=floor((recty(3,1)+recty(4,1))/2);

% hold on;

% plot(mid\_x,mid\_y,'b\*');

Long=-point(4,2)+mid\_y;

Short=point(4,1)-mid\_x;

Ans\_x=mid\_x+Long;

Ans\_y=mid\_y+Short;

toc;

disp(['运行时间: ',num2str(toc)]);