1 背景介绍

机器视觉是一个发展中的学科方向，其理论体系还在不断完善，以普通视觉模型为对象的视觉检测系统在工业应用中的研究具有重要的意义。对于工业产品的检测而言，立体视觉检测算法主要包括：摄像机标定；图像的预处理及特征提取；特征匹配；尺寸计算。

利用边缘检查的尺寸检查是图像传感器的最新应用趋势。图像传感器可以将检查对象在平面上表现出来，通过边缘检测, 测算位置、宽度、角度等。

所谓边缘是指图像内明亮部位与阴暗部分的边缘，边缘检测是通过视觉系统来检测这种浓淡变化的边缘。边缘是图像的一个基本特征，对边缘的检测一直是图像处理技术中非常重要的问题。

传统的边缘检测算子，通常对噪声比较敏感，且检测一般为像素级，精度较低。而在许多实际应用中，要求检测出的图像边缘达到亚像素级。例如，在计算机视觉测量领域，被测件边缘点的精度往往直接影响到整个测量结果的精度，因此，研究图像的亚像素边缘检测算法有着重要的实际意义。国内外很多学者对该问题进行了广泛的研究，提出了很多亚像素边缘检测方法。

亚像素边缘检测方法是指在硬件条件一定的情况下，用软件的方法来提高边缘的定位精度，使边缘能定位到像素内部更精确的位置。目前学术界对亚像素边缘检测尚无统一的定义，一般可以理解为一种提高边缘定位精度的方法，或者是一种可以使分辨率小于一个像素的图像处理技术。

2 原理

使用了一种基于 *Zernike*矩和梯度方向插值的亚像素边缘检测的方法来实现物体尺寸的测量，原理如下。

2.1 Zernike 矩

图像的 *Zernike*矩可以表示为 [1][2]

$$Z_{nm} = \sum_{x}\sum_{y} f(x,y)V_{nm}^{*}(\rho,\theta) \tag{1}$$

其中 V_{nm} 为 $Zernike$ 多项式，可以表示为

$$V_{nm} = R_{nm}(\rho)e^{im\theta}, |\rho| \leq 1 \tag{2}$$

其中，$R_{nm}(\rho)$ 为径向多项式，其中当 $n-m$ 为偶数时 $R_{nm}(\rho)=0$，当 $n-m$ 为奇数时，可表示为

$$R_{nm}(\rho) = \sum_{k=0}^{(n-|m|)/2} \frac{(-1)^k(n-k)!}{k!(\frac{n+|m|}{2}-k)!(\frac{n-|m|}{2}-k)!}\rho^{n-2k} \tag{3}$$

通过式1我们可以计算出图像 $f(x,y)$ 各阶的 $Zernike$ 矩。

2.2 基于 Zernike 矩的亚像素边缘检测

为了方便讨论，建立单位圆边缘模型，如图1所示，该模型中的边缘为理想阶跃型边缘，其中 h 为背景灰度值，k 为阶跃边缘的灰度值，ϕ 为边缘的方向，l 为圆心到边缘的距离。接下来，我们使用式1计算该模型的各个参数，如下所示 [3]

$$k = \frac{3Z_{11}^{'}}{2(1-l_2^2)^{3/2}} \tag{4}$$

$$h = \frac{1}{\pi}\left[Z_{00} - \frac{k\pi}{2} + k\arcsin(l_2) + kl_2\sqrt{1-l_2^2}\right] \tag{5}$$

$$l = \frac{1}{2}\left[\sqrt{\frac{5Z_{40}^{'}+3Z_{20}^{'}}{8Z_{20}}} + \sqrt{\frac{5Z_{11}^{'}+Z_{11}^{'}}{6Z_{11}}}\right] \tag{6}$$

$$\phi = \arctan\left[\frac{Im[Z_{n1}]}{Re[Z_{n1}]}\right] \tag{7}$$

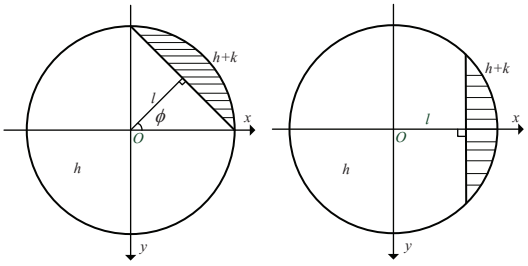

Figure 1: 理想阶跃边缘模型

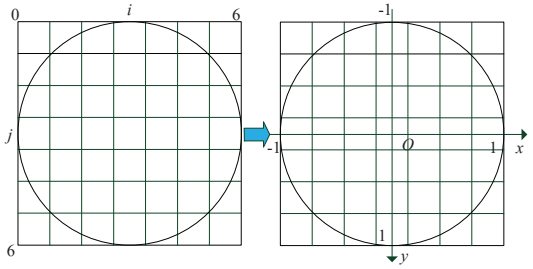

Figure 2: 模板效应

如图2所示，考虑到将边缘 7*7 邻域视为如图1所示单位圆内部的的像素，即将产生位置的模板效应，边缘的亚像素坐标为

$$Subpixel = \begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} x + \dfrac{lN}{2}\cos(\phi) \\ \\ y + \dfrac{lN}{2}\sin(\phi) \end{bmatrix} \tag{8}$$

其中，(x_s, y_s) 为亚像素级边缘位置，(x, y) 为像素级边缘位置，而 N 为邻域尺寸。

2.3 梯度方向插值优化边缘检测

使用 7*7$Zernike$ 模板来检测亚像素边缘，虽然速度快，但是检测到的亚像素点数量少，并且，对复杂边缘无法定位出完整的边缘轮廓，边缘连接性差。为了最大化边缘信息，使用双线性插值法对边缘邻域进行方向性伸展，设 x 方向的伸展倍数为 x_t，设 y 方向的伸展倍数为 y_t，则有

$$x_t = \lfloor C_t \frac{\partial f(x,y)}{\partial x} \rfloor, y_t = \lfloor C_t \frac{\partial f(x,y)}{\partial y} \rfloor \tag{9}$$

其中 C_t 为一常数。现在，边缘亚像素坐标可以改写为

$$Subpixel = \begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} x/x_t + \dfrac{lN}{2x_t}\cos(\phi) \\ \\ y/y_t + \dfrac{lN}{2y_t}\sin(\phi) \end{bmatrix} \tag{10}$$

2.4 拟合亚像素点

将检测到的亚像素点拟合到任意曲线或直线上，即可实现尺寸测量，值得注意的是，这个时候得到的是像素距离，要得到实际的长度还需要知道每个像素代表多少长度，这个需要在实际测量中结合镜头高度和实际距离，像素个数，其他环境因素来考虑和测定，在得到该数据后，可用于该环境下的物体尺寸测量。

3 系统设计

以下介绍本小组队该系统的具体设计细节，从平台与环境的搭建，上位机设计，核心算法设计三方面来讲，根据不同小组成员分工的不同，将会有侧重，本人负责核心算法的设计，遂该部分在本课程设计报告上会有较为详细的叙述。

3.1 平台与环境的搭建

此电子信息系统由硬件与软件构成，硬件部分为 NVIDIA JETSON TK1 开发板，如图3

Figure 3: NVIDIA® Jetson TK1

NVIDIA® Jetson TK1 开发组件能够为嵌入式应用而释放 GPU 的强大能力，满足开发者所需，具体配置见表1。该平台围绕 NVIDIA® Tegra® K1 SOC 而打造，采用与世界各地计算机相同的 NVIDIA® Kepler™ 计算核心。它能够为你提供一个全功能 NVIDIA® CUDA® 平台，让你能够快速开发和部署用于计算机视觉、机器人学以及医学等领域的计算密集型系统。NVIDIA® 可提供整个 BSP 和软件包，其中包括 CUDA®、OpenGL 4.4 以及由 Tegra® 加速的 OpenCV。凭借全套的开发和动态分析工具以及对摄像头和其它外设的原生支持，NVIDIA® 为开发者带来了理想的解决方案，可帮助打造嵌入式的未来。

Table 1: NVIDIA® Jetson TK1 配置一览

Tegra K1 SOC
NVIDIA® Kepler® GPU、192 个 CUDA 核心
NVIDIA® 4-Plus-1™ 四核 ARM® Cortex™-A15 CPU
2 GB x16 内存、64 位宽度
16 GB 4.51 eMMC 内存
1 个 USB 3.0 端口、A
1 个 USB 2.0 端口、Micro AB
1 个半迷你 PCIE 插槽
1 个完整尺寸 SD/MMC 连接器
1 个 RTL8111GS Realtek 千兆位以太网局域网
1 个 SATA 数据端口
1 个完整尺寸 HDMI 端口
1 个 RS232 串行端口
SPI 4 兆字节引导闪存
1 个带 Mic In 和 Line Out 的 ALC5639 Realtek 音频编解码器
以下信号可通过扩展端口获得:DP/LVDS, Touch SPI 1x4 + 1x1 CSI-2, GPIOs, UART, HSIC, I^2C

在本课程设计中，在 NVIDIA® Jetson TK1 上移植了 *ubuntu*14.04，*Opencv*2.4.10 库，*GTK*2.0 库（用于人机界面程序编写）。具体配置方法这里不做赘述，参考负责这部分的组员的详细描述。

3.2 上位机设计

3.2.1 配置和外观

为了实现人机交互，帮助算法的实现，使用 GTK2.0 编写可视化上位机界面程序，具体的配置清单如表2所示，上位机的功能如表格3所示。

Table 2: 上位机配置清单

运行环境	ubuntu14 (基于 Cortex®-A15 芯片)
编程语言	C/C++
第三方库及组件	GTK2.0，OpenCV2.4.10
开发环境	Qt Creator 与 make 工程管理器
编译工具链	NVIDIA®-ARM® 编译工具链
程序结构	模块化结构

Table 3: 上位机功能清单

编号	功能描述
1	可打开/关闭摄像头
2	可通过摄像头捕获图片为目标图片
3	可从文件系统内选择图片并载入为目标图片
4	可以检测目标图片中圆形轮廓的半径和圆心
5	可以检测目标图片中平行直线的间距
6	检测算法的参数可自由调整

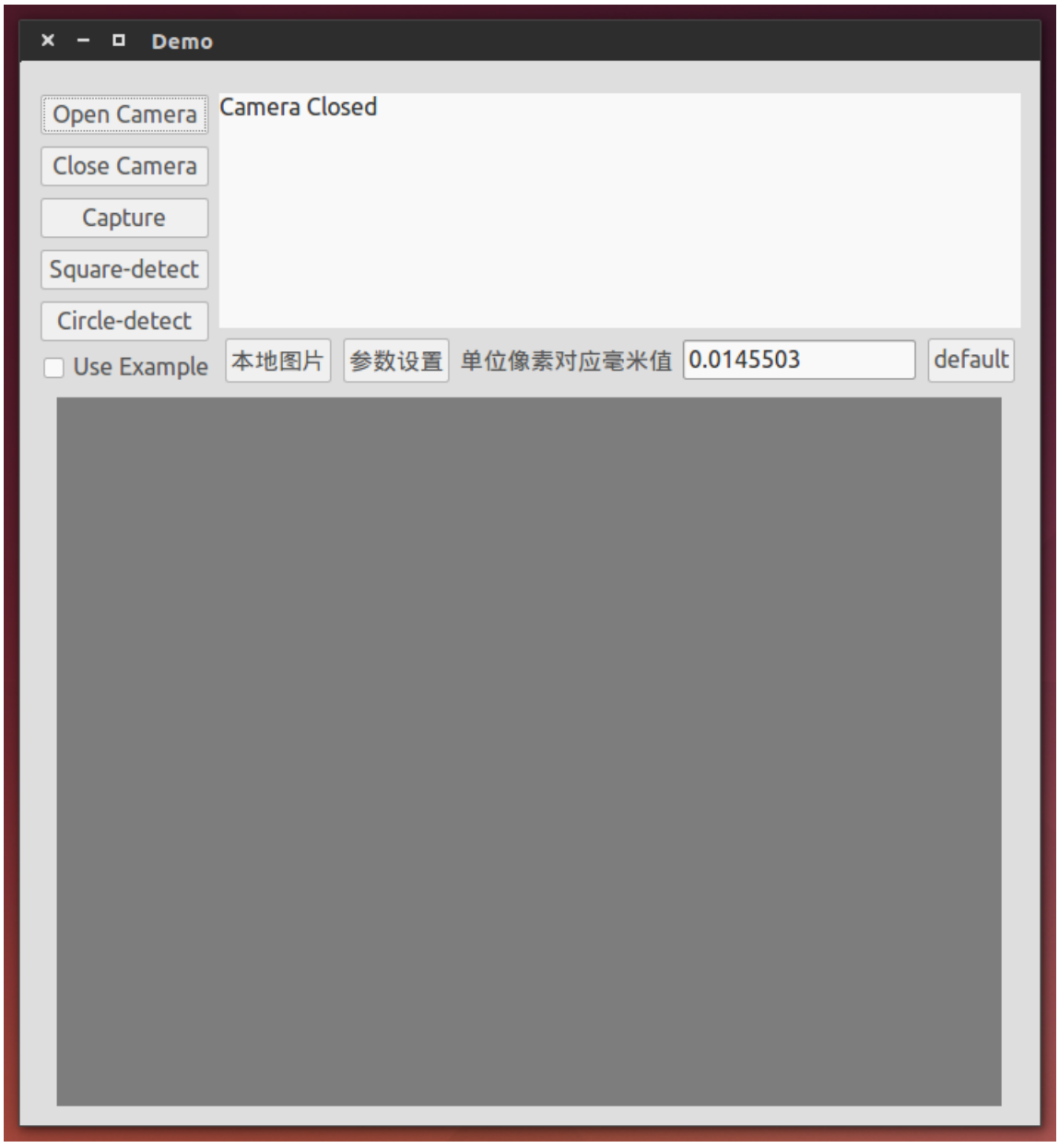

Figure 4: 上位机主界面

Figure 5: 上位机子窗口

3.2.2 操作指南

如图4所示，程序的主界面由上下两部分构成。上半部分由各类按键，信息显示文本区等构成，下半部分为待测量图片/测量结果图片显示区域。点击 Open Camera 按键可以打开摄像头，摄像头会每隔 30ms 拍摄一幅图片，并显示。点击 Close Camera 按键会关闭摄像头。在开启摄像头后，点击 Capture 按键会截获当前拍摄的图片作为待测量的目标图片。点击“本地图片”按键会弹出文件选择对话框，可以在其中选择本地图片载入为目标图片。点击“参数设置”按键会弹出参数设置界面如图5，可以在此界面内修改核心算法依赖的各项参数。勾选 Use Example 会使程序进入示例模式，此模式下，按下 Square-dectect 和 Circle-dectect 按键会自动选取当前目录下的示例图片进行测量并在主界面上方白色文本区域显示测量结果。当待测量图片区不为纯灰色背景时，点击 Square-dectect 按键会进入平行线间距检测流程，并弹出平行线选取窗体，此时，用鼠标框选窗体内图片中的两条平行线再按下 Esc 按键便会开始平行线间距的测量，测量结果将在白色文本区域显示。当待测量图片区域不为纯灰色背景时，点击 Circle-dectect 按键会进入圆尺寸测量流程，并弹出圆选取窗体，此时，用鼠标框选一个圆选取窗体图片中的圆形，再按下 Esc 按键便会开始圆尺寸测量，测量结果（半径，圆心位置）会呈现在白色文本区域

3.3 核心算法设计

算法使用 $OpenCV2.4.10(C++)$ 编写，使用 $VisualStudio2017$ 调试。算法的整体设计思路为：输入图像和算法配置参数，处理，输出测量结果。将功能包装成函数，以方便外部调用，实现接收待测图像，处理，返回结果的功能，如图6所示。根据结构6编写两个函数，一个用于矩形尺寸测量，一个用于圆形测量。

Figure 6: 算法整体结构

如图6所示，这里输入为图像与配置参数，具体输入参数配置一览见表4，我编写矩形和圆形测量两个函数的输入参数都是一样的；输出测量结果为一结构体 (*struct* 类型)，包含所需要输出的测量结果，详细结构体成员项目见表5。其中标记图像为算法将拟合的到的直线（圆）标记到图像后的图像，这里作为输出是因为方便调用的时候用于可视化的显示检测结果。

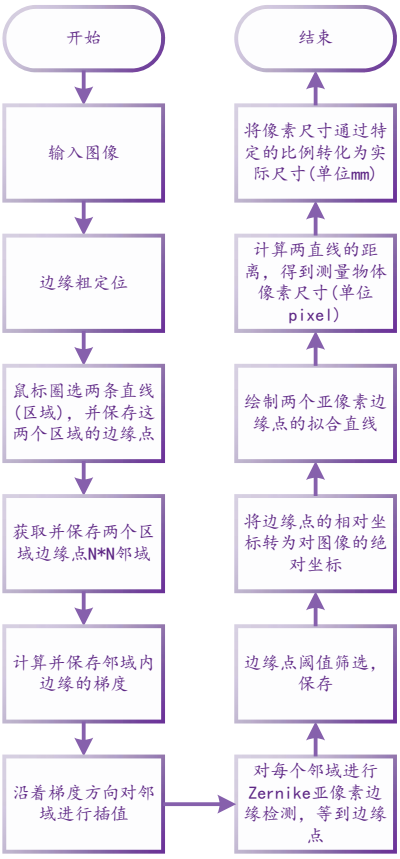

Figure 7: 矩形测量算法流程

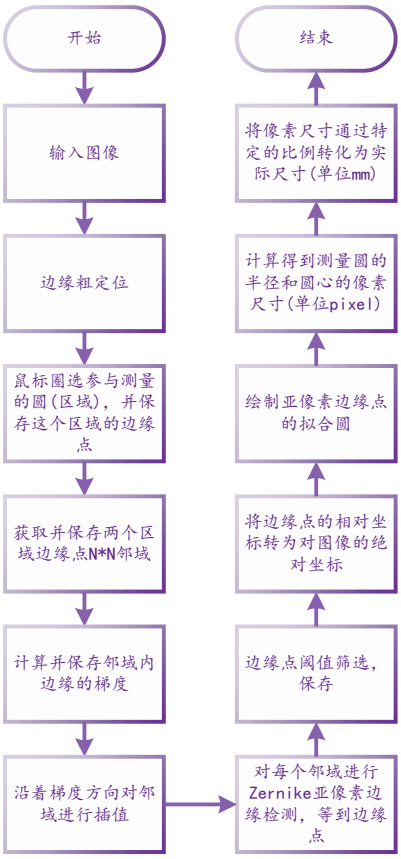

Figure 8: 圆形测量算法流程

Table 4: 输入参数配置表

参数	类型	说明	默认值
matRoi	Mat	输入图像	
rto	double	毫米/像素值 (mm/pixels)	0.0145503
drawColor	int	区域选择矩形框颜色（灰度 0-255）	255
ifmedianBlur	bool	输入图像是否需要中值滤波	Yes
cannyL[4]	int	Canny 粗边缘检测低阈值	150
cannyH	int	Canny 粗边缘检测高阈值	200
cannyCore	int	Canny 核大小（奇数）	3
nbsize	int	边缘点邻域大小（奇数）	7
Vhtime	Point2i	垂直/水平边缘插值缩放系数	Point2i(4, 4)
ZerBgrL	int	亚像素边缘背景低阈值	10
ZerBgrH	int	亚像素边缘背景高阈值	50

Table 5: 函数返回的结构体成员说明

	矩形测量函数	圆测量函数
结构体成员	两直线距离 (单位:pixels)	圆心位置（单位:pixels）
	两直线距离 (单位:mm)	圆心位置（单位:mm）
	毫米/像素值 (单位:mm/pixels)	毫米/像素值 (单位:mm/pixels)
	标记图像 (类型:Mat)	标记图像 (类型:Mat)

如表5所示，每个返回的结构体中都有“距离”这个成员变量，“距离”既有像素距离，又有实际距离，具体的转换方法为

$$l_r = l_p * MP \tag{11}$$

其中，l_r 为实际长度，l_p 为像素距离，MP 为毫米/像素值 (单位:mm/pixels)，这个值在不同的测量环境和相机配置中是不同的，需要在测量的时候先确定。

函数的主体流程如图7和图8所示。

最后的函数下代码所示，其中 *ZerResult*，*ZerResultCir*即为表5所描述的输出结构体。相关参数已经进行宏定义，这里不做赘述，详情见附录代码。

```
ZerResult CalDistanceSquare(Mat& matRoi, double rto = MMPERPIXEL,
                            int drawColor = WHITE, bool ifmedianBlur=YESMEDIANBLUR,
                            int cannyL = 150, int cannyH = 200, int cannyCore = 3,
                            int nbsize = N, Point2i VHtime = Point2i(4, 4),
                            int ZerBgrL = 10, int ZerBgrH = 50);
ZerResultCir CalDistanceCircle(Mat& matRoi, double rto = MMPERPIXEL,
                            int drawColor = WHITE, bool ifmedianBlur = YESMEDIANBLUR,
                            int cannyL = 150, int cannyH = 200, int cannyCore = 3,
                            int nbsize = N, Point2i VHtime = Point2i(4, 4),
                            int ZerBgrL = 10, int ZerBgrH = 50);
```

4 实验

4.1 平台与配置

实验平台：NVIDIA® Jetson TK1 板，*Opencv*2.4.10，*GTK*2.0

4.2 实验结果与分析

以下分为两部分阐述实验结果，分别是矩形测量实验结果和圆形测量实验结果。这里先不打开摄像头，假设相机已经标定，即毫米/像素值 =0.0145503，待检测图片是一系列已知实际尺寸的圆形和方形图像。如图9所示。实验的操作过程涉及上位机的操作，具体操作方法参见第 3.2 节所述上位机设计的内容。算法的参数遵从表4所示，使用默认参数进行实验，由于参数众多，不同参数可能产生不同的效果，实际应用中，可以反复调节，直到合适为止。

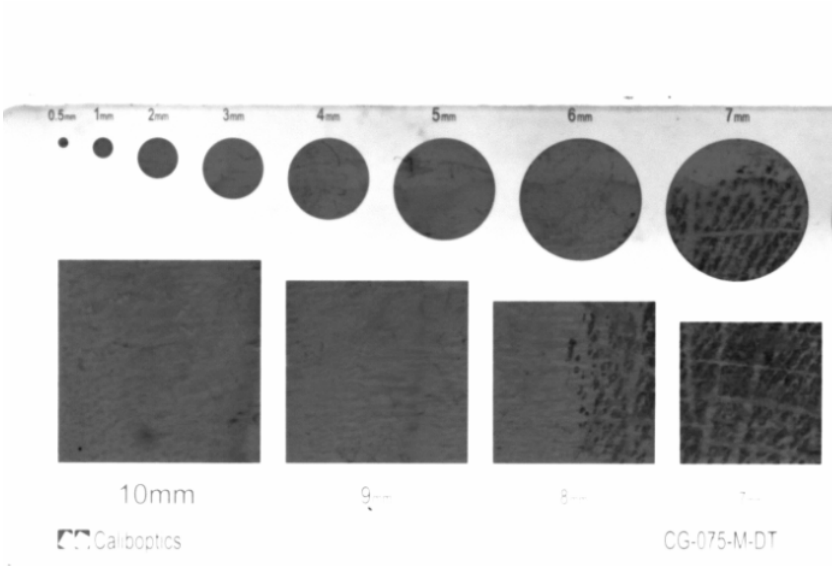

Figure 9: 待测标准图

4.2.1 矩形测量实验结果

测定图9中 4 个矩形的边长（从左到右实际边长为 10mm，9mm，8mm，7mm），多次测量，得到平均误差，结果如表6

Table 6: 矩形测量实验结果

实际边长 (单位:mm)	测量边长 (单位:mm)			平均误差 (单位:mm)
	一	二	三	
10	9.99286	9.9928	9.99282	0.00719
9	8.99718	8.99704	8.99708	0.00289
8	8.00154	8.00153	8.00173	0.0016
7	7.00176	7.00185	7.00176	0.0017

可以看到，随着测量物体边长的变大，误差随之增大，估计是误差的累计效应，误差稳定在 0.009mm。实际操作可以继续调整参数，加以优化。

4.2.2 圆形测量实验结果

测定图9中 4 个圆形的半径和（分别选取半径为 3mm，2.5mm，2mm，1.5mm 的 4 个圆），多次测量，得到平均误差，结果如表7。

可以看到，同圆形尺寸测量的结果，随着半径的增大，误差逐渐增大，估计是误差的累计效应，整体误差稳定在 0.03mm 以内。这里多次测量的结果仍然一样，这是因为程序设计导致的，在一开始我们会使用矩形框框选需要测量的圆，只要圆被圈住，计算的结果就一致。

Table 7: 圆形测量实验结果

实际半径 (单位:mm)	测量半径 (单位:mm)			平均误差 (单位:mm)
	一	二	三	
3	3.0216	3.0216	3.0216	0.0216
2.5	2.5222	2.5222	2.5222	0.0222
2	2.0194	2.0194	2.0194	0.0194
1.5	1.5135	1.5135	1.5135	0.0135

5 总结

在该课程设计项目中，使用 NVIDIA® Jetson TK1 板，移植了 *ubuntu*14.04，*Opencv*2.4.10 库，*GTK*2.0 库，并以此作为实验环境和测试平台。核心算法使用 *C++* 编写，为了在嵌入式板平台下可以进行人机交互，使用 *GTK*2.0 库编写上位机程序。使用标定图进行测试，经过对比和分析，物体测量精度基本符合预期，通过亚像素的边缘定位和梯度插值技术，将测量精度提高了一个档次，值得期待的是，本实验只对一组输入配置参数进行测试，在实际应用中，还可以进一步调整参数，优化算法，提高精度。

References

[1] Ghosal S, Mehrotra R. Orthogonal moment operators for subpixel edge detection. Pattern recognition, 1993, 26(2): 295-306.

[2] Hwang S K, Kim W Y. A novel approach to the fast computation of Zernike moments[J]. Pattern Recognition, 2006, 39(11):2065-2076.

[3] Gao S Y. Improved Algorithm about Subpixel Edge Detection of Image Based on Zernike Orthogonal Moments[J]. Acta Automatica Sinica, 2008, 34(9):1163-1168.

[4] Canny J. A Computational Approach to Edge Detection. IEEE Computer Society, 1986.