浙江大学

本科生毕业论文（设计）

题目 基于Kinect深度信息的植物无损测量

姓名与学号 3100100423 马轲

指导教师 蒋焕煜

年级与专业 2010生物系统工程

所在学院 生物系统工程与食品科学学院

目录

摘要

植物结构性参数的无损测量在生长监测、科学建模和品质控制等方面具有重大意义。本研究利用微软 Kinect 传感器作为彩色和三维位置数据源，结合多种点云处理技术获取了植株的完整三维点云，并在此基础上导出了植株的多种结构性参数，达到了无损测量的目标。本研究将整个无损测量过程分为植物三维点云的获取、校正、预处理、配准、后处理和测量六个步骤，并详细讨论了每个步骤的基本原理、实现方法和改进意见。本研究提出的无损测量流程对硬件和软件环境要求较低，步骤简便，易于实现，测量结果误差较小，且能够完成一些物理测量难以获取的指标的测量，具有一定的实践意义。

关键词 Kinect，植物，无损测量，结构性参数，点云

Abstract

The non-destructive measurement of plant structural parameters is of great importance in fields such as growth monitoring, scientific modelling and quality control. This research utilizes a Microsoft Kinect sensor to acquire color and 3D position data and combines multiple point cloud processing techniques to reconstruct the full 3D point clouds of the plants. It also extracts several plant structural parameters based on the clouds, so as to achieve the goal of non-destructive measurement. The whole process is divided into six steps including acquisition, adjustment, pre-processing, registration, post-processing and measurement. The basic principles, implementations and improvements of each step are demonstrated in detail. The non-destructive measurement process proposed by this research has relatively low hardware and software requirements, convenient steps and easy implementations; the measurement error is reasonable, and some parameters that can hardly be achieved by physical measurements can be measured. Thus, this non-destructive measurement process has some practical significances.

Keywords Kinect, Plant, Non-destructive Measurement, Structural Parameter, Point Cloud

第 1 章 绪论

1.1 课题背景

1.1.1 植物无损测量的意义和方法

近年来，随着传感技术、信息技术等学科的迅速发展，数字农业已成为农业工程的一个重要的研究领域，作为当代农业发展的新方向展现出广阔的应用前景。数字农业即是在传统基础学科的指导下，结合多种高新技术，在农业生产过程中对作物及环境进行监测和调控，从而达到降低成本、提高品质、改善生态等一系列目标。

作为生产监测的重要一环，植物结构性参数的测量在科学研究和农业生产中都具有十分重要的意义。植物生长过程复杂，通过对多种结构性参数的量化可以直观地反映出植物的生长情况，为进一步研究提供数据支持。事实上，植物结构性参数的测量在生长监测、科学建模、品质控制等诸多方面均有应用。

长期以来，植物的生长参数测量以接触性和破坏性测量为主，虽然简便易行，但弊端同样显而易见。这类测量往往难以大规模进行，得到的数据具有片面性；并且难以实时进行监测，数据的观测过程不连续；测量过程还可能会对植物造成不可预料的影响。植物无损测量则有效解决了这些问题，更加适应于未来对植物生长过程进行大范围动态监测的需要。

20 世纪 90 年代以来，农作物遥感监测成为遥感技术应用的一个重要方面。但这种技术常受到气候条件影响精度不高，且运行周期较长不能做到实时监测。在监测范围上来讲，遥感监测只适用于宏观农作物检测，无法对单个植株进行快速、准确、全面的测量与分析。此外，遥感监测通常需要地面上的校准工作，必须通过其他的测量方法实现。

机器视觉是利用计算机模拟视觉功能的新兴技术，它能够利用图像重建现实世界的物理对象，从而有效地避免遥感测量的各种弊端。传统的基于机器视觉的植物无损测量主要通过 CCD 摄像机、激光扫描仪、双目设备等实现。

CCD 摄像机能够获取植物的二维彩色图像。通过采用不同敏感波段的摄像机，还能够反映出植物的一些肉眼不可见的生理状况。这类设备已经获得了广泛的应用，设备和相关算法都比较成熟。其缺点在于只能获得二维的平面数据，不能够全面地反映植物的三维结构信息；由于植物本身外观的多样性和背景环境的复杂性，在图像分割、识别等方面也常常会出现问题。

激光扫描仪、双目设备则能够获取具有深度数据的三维图像。前者通过激光飞行时间计算目标与设备间的距离，而后者则以三角测量原理计算深度。深度数据的获取使得对植物进行三维重建成为可能，更为全面的结构性参数测量也就具备了数据来源。双目设备获取的深度图像通常是分片而稀疏的；激光扫描仪的深度图像虽然较细密，但分辨率低且多噪声。此外，这两类设备价格昂贵，难以在农业领域获得广泛应用。

1.1.2 微软 Kinect 传感器简介

Kinect 传感器本是微软公司于 2010 年推出的 XBOX360 家用游戏机的体感控制器，它可以实时同步捕捉彩色图像和深度图像，同时提取出人类骨架数据，从而允许人们不用触摸控制器即可进行游戏。由于其相对低廉的价格和优秀的设备特性，人们很快发现了它丰富的应用场合，并开发了一些第三方的驱动程序利用 Kinect 传感器进行开发。微软公司同样看到了这一趋势，在 2012 年 2 月推出了专门用于在 Windows 平台上的进行应用开发的 Kinect for Windows 传感器，以及 Kinect for Windows SDK 软件开发工具包。

图 1.1.1. Kinect 传感器外观图

图 1.1.2. Kinect 红外投影机发出的激光散斑

Kinect 传感器的核心部件是正面的红外投影机（左）、红外摄像头（右）和彩色摄像头（中），如图 1.1.1 所示。红外投影机和红外摄像头共同用于深度图像的获取。红外投影机采用一种光编码技术，发射出的普通红外激光束通过不均匀介

质形成随机衍射斑点，称为激光散斑，如图 1.1.2。这种激光散斑在不同位置的平面上具有不同图案，摄像头捕获红外图像后利用存储在内部的参考图案与图像进行相关运算，即可确定每个点的深度。它能够提供最大分辨率 640×480、每秒 30 帧的深度图像，如图 1.1.3 所示。彩色摄像头即是 CCD 摄像头，能够提供分辨率 1280×960、每秒 12 帧或分辨率 640×480、每秒 30 帧的 RGB 三通道彩色图像，如图 1.1.4 所示。两个摄像头在 Kinect 传感器内部即进行了配准过程，使得彩色图像和深度图像的像素一一对应。

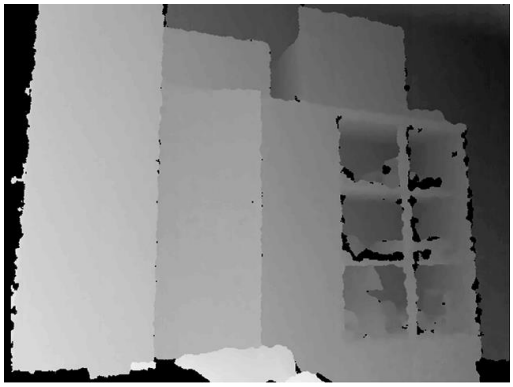


图 1.1.3. Kinect 获取的深度图像

图 1.1.4. Kinect 获取的彩色图像

对于 Kinect for Windows 传感器而言，可视距离为 0.8~4.0 米，采用近景模式则为 0.4~3.5 米；可视角度为水平方向 57°、垂直方向 43°，并且可以在垂直方向上倾斜±28°[1]。

与其他能够获取三维数据的成像设备相比，Kinect 传感器获得的深度图像不仅像素细密、分辨率和帧率高，而且能够同时获得准确对齐的彩色图像。此外，Kinect 传感器作为一款消耗类电子产品，具有价格低廉、可获得性好的特点。当然，Kinect 传感器也具有一些固有的缺点，例如深度数据噪声较大、物体边缘处往往存在大量无效数据等。

1.1.3 基于 Kinect 的植物无损测量研究现状

将 Kinect 传感器用于植物无损测量的研究大致起始于 2011 年，在近几年中得到快速发展，已有不少研究成果见诸学术期刊。而由于 Kinect 传感器在中国上市较晚，国内对于 Kinect 传感器在植物无损测量方面的研究也相对落后，但同样出现了一批具有启示意义的研究成果。

瑞典和西班牙的 Wallenberg 等提出了利用 Kinect 获取的彩色图像和深度图像进行植物叶片分割的方法，该研究采用通道编码技术和超顺磁聚类方法，有效地从整个植株图像中划分出单个叶片[2]。华南农业大学的江晓庆等提出了利用 Kinect 获取植物深度图像及点云的一种过程，但并未在获取点云后做进一步的研究[3]。日本的 Yamamoto 等提出了利用 Kinect 在一种循环式可动工作台系统中监测草莓生长信息的方法，该研究中全面尝试了对草莓植株的株高、宽度、体积、叶面积、叶倾角和颜色等指标的测量[4]。北京师范大学和桂林科技大学的 Chen Yiming 等探索了利用 Kinect 无损测量玉米单个叶片叶面积和叶倾角的方法，研究利用不规则三角网进行三维曲面重建，同时获得了叶面积和叶倾角的测量结果[5]。美国的 Azzari 等尝试将 Kinect 用于植物的冠层特性描述，研究采用两种模式对植物点云进行捕获，并根据两类点云的特征采用不同方法研究了植株的基圆直径、高度、体积等指标[6]。

当前对基于 Kinect 的植物无损测量的研究，从研究对象上分，可以分为对植株整体的结构性参数的测量和对植株的部分（通常是叶片）的参数进行测量；而从研究指标上分，通常包括尺寸测量（包括直径、围长、株高等）、叶面积和叶倾角测量、体积估算等，还包括对植株进行三维重建和叶片分割的研究。

同时必须承认，当前的相关研究也存在着一些不足。各种测量方法都缺乏大范围的验证，准确性、稳定性、效率等多方面均有待加强；Kinect 传感器也存在着一些内在的不足，例如在日光下深度图像质量较差，深度测量范围较窄，物体边缘处存在盲区等等。这些问题都期望通过未来的研究得到解决，同时，未来的研究也应当尝试更多的测量参数和更深层次的分析，也应当把相关领域的优秀成果引入对植物无损测量的研究中来。

1.2 研究内容和目标

本课题主要包含三个方面的研究：测量系统的构建与点云的获取，植物三维点云的处理，植物结构性参数的测量。

测量系统的构建与点云的获取。对于植物点云的获取，需要硬件和软件相互配合支持，构成测量系统。测量系统硬件部分的设计需要考虑测量对象的性质，也要考虑 Kinect 传感器的固有特性，包括视场角、测量范围、测量环境等等。为

了获得较为完整的植物点云，测量系统必须能够从多个角度对植物进行拍摄，因而需要一些可移动设备的辅助。软件部分则需要利用 Kinect 提供的编程接口读取深度数据流，再通过坐标变换将深度图像中的每个点转换为三维空间的坐标点，从而构成三维点云。

植物三维点云的处理。获取的植物多角度三维点云不能直接用于测量，还需要经过一定的处理。这些处理不仅包括将多角度点云合成为单个完整点云的操作，还包括一些为了提升点云质量、提高测量精度、缩短运算时间等的处理。因为任务步骤较多、流程较为复杂，根据情况会分成多个子任务进行。

植物结构性参数的测量。在得到植物的较为完整的三维点云后，就可以通过一定的计算测量出植物的结构性参数。但能够测量哪些参数、以及参数测量的精度，还与获取的植物点云的质量有关。可测量的参数大致可以分为两类，一类关注植株整体，另一类则关注植株的某些重要部位（如叶片），本课题主要关注前者。此外，还需要对植株进行一定的人工物理测量，用于与无损测量的结果进行对比。

1.3 研究思路和方法

1.3.1 主要流程

根据上文所述，本课题的研究内容主要包含三个部分的任务，其中植物三维点云的处理部分还可以进一步分为多个子任务。因此，我把研究过程中划分为六个具有一定先后顺序、又彼此紧密联系的部分，分别为：植物三维点云的获取、校正、预处理、配准、后处理和测量。具体流程见图 1.3.1。

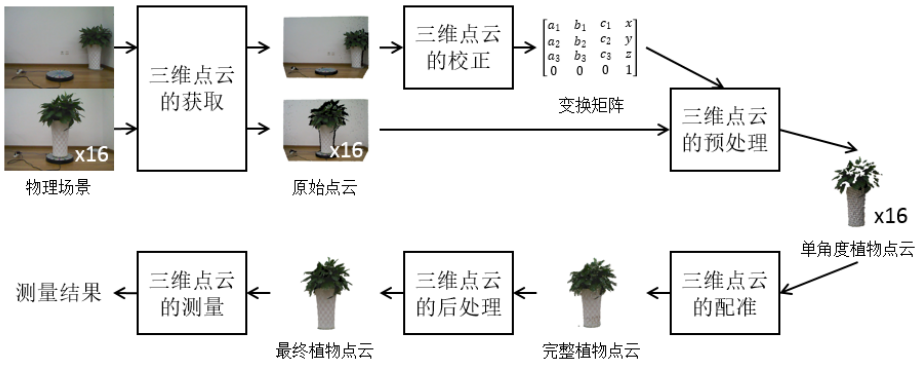


图 1.3.1. 研究主要流程

测量系统硬件部分主要包括 Kinect 设备和转台，软件部分则包括上述六个部分对应的程序。首先利用三维点云获取软件获取转台空载时的点云数据，用于后面的校正过程；再将待测植物放置在转台上，旋转转台并获取包含多角度植物点云数据的多幅原始点云。利用转台空载的点云数据计算出转台中心和平面法向，进而计算出校正点云倾斜的变换矩阵。结合变换矩阵信息对原始点云进行预处理，得到去除背景点云和离群点、矫正了倾斜并进行了姿态推测（粗配准）的单角度植物点云。接着对多幅单角度植物点云进行配准，并合成为单个完整植物点云。随后经过一定的后处理，去除多余点并进行平滑以提高点云质量。最后利用三维点云测量软件对植物点云进行多钟结构性参数的测量。

1.3.2 开发环境

本课题所使用的 Kinect 传感器为微软 Kinect for Windows 传感器。主要的编程和执行工作在装有 Windows 8.1 操作系统的 PC 上运行，其处理器为 Intel Core i5 M450（双核 2.40GHz）。

程序开发语言选用 C++。现代 C++语言可以看做有三部分构成：低级语言、现代高级语言特性和标准库[7]。低级语言使程序能够高效运行，适用于图形运算等计算密集型的工作；现代高级语言特性使得编程更为轻松，大型程序的组织更为有序；标准库免去了从零开始编写一些常用算法和数据结构的麻烦。本课题选用 C++作为编程语言的原因有二：一是其高效和易用性，二是有很多用于处理图形图像的开源库均为其设计。

集成开发环境选用微软 Visual Studio 2010。Visual Studio 提供了完整的编写、编译、调试 C++程序的环境，以及包含自动完成、格式化元素等在内的一系列方便开发者的功能。选用这一版本而非最新版本的原因在于，2008 版本和 2010 版本能够对下述的点云库 PCL 提供更好的支持。

主要使用的开源库为点云库 PCL 1.7.1。PCL 实现了大量点云相关的通用算法和高校数据结构，涉及点云获取、滤波、分割、配准、检索、特征提取、识别、追踪、曲面重建、可视化等[8]。由于 PCL 每个版本都有许多新特性加入，因而这里选择了较新的版本。PCL 还需要一些依赖库的支持，包括 Kinect 传感器等一系列自然人机交互设备的接口 OpenNI、用于共享指针和线程操作的 Boost、用于矩阵和向量数据操作的 Eigen、用于 3D 点云渲染和可视化的 VTK 等。

1.3.3 实验对象

本研究对于待测植株的选择具有一定的要求：一是植株形态方面，待测植株如果茎秆较细或叶片很小，这些部位就不容易得到有效的深度数据，因而获取这类植株的点云质量就会比较差。而植株叶片较大较密的情况，就比较适合本研究。二是植株大小方面，如前所述，Kinect 传感器具有一定的可视距离，也就是说待测植物不能够放置于太靠近或太远离 Kinect 传感器的位置。如果植株太小，Kinect 获取的点云密度也会比较小，影响了点云质量和测量精度；如果植株太大，Kinect 可能无法拍摄到完整的植株侧面。

图 1.3.2. 待测植株 A

图 1.3.3. 待测植株 B

图 1.3.4. 待测植株 C

课题所选用的待测植株为三个盆栽植物。它们总的来讲均符合上述对待测植株的基本要求，又具有不同的特点。待测植物 A 叶片较大，植株体积也较大，形态相对简单，如图 1.3.2 所示；待测植物 B 叶片较小，但叶片数量众多，植株体积相对植株 A 略大，形态相对复杂，如图 1.3.3 所示；待测植物 C 叶片也比较小，而植株体积也远远小于前两个植株，同时具有冠层在一个方向上跨度较大、另一个较小的特点，如图 1.3.4 所示。

1.4 本文章节安排

根据上述的研究流程，本文共分八个章节。其中第二章至第七章分别对应研究流程中的七个部分，每个部分详细介绍了研究中用到的一些基本原理并阐述了具体的实现方法，展示并分析了实验结果，最后提出了一些改进的意见。第八章为上述六章的总结，总结了本研究的工作并提出了继续研究的方向。

第 2 章 植物三维点云的获取

2.1 基本原理

2.1.1 主要工作思路

这一部分的主要工作在于获取包含多个角度的植物点云在内的多幅物理场景三维点云，涉及到硬件和软件两个层面。在硬件层面，需要让 Kinect 传感器能够拍摄到多角度的植物点云，因而需要运动设备的支持，这里选用电动转台使待测植株能够 360°转动。此外，为了下一章中植物点云校正的需要，在搭建测量系统硬件时还需要考虑一定的参考标记或图案。在软件层面，需要让用户能够实时看到 Kinect 捕获到的物理场景三维点云，同时能够快捷地保存成为点云文件。同样为了方便点云校正，程序也应当能够实时显示和保存彩色图像。

2.1.2 Kinect 彩色和深度图像数据

Kinect 传感器获取的原始彩色图像是 1280×1024 分辨率的 Bayer 格式数据。Bayer 数据采样时奇数扫描行输出“RGRG…”，偶数行输出“GBGB…”，而对于每个像素来说，其 RGB 信号由自身的某颜色信号和相邻像素的其他颜色信号构成。这种采样方式基本不损失图像质量，但可以将采样频率降低 60%以上[1]。但由于高分辨率的图像数据量较大，受传输速率限制，帧率较低；Kinect 传感器可以预先对彩色图像进行下采样，减少数据量，从而保证每秒 30 帧的帧率。在本研究中，就使用了分辨率为 640×480、帧率为 30fps 的 32bit RGB 图像。

在 Kinect 传感器获取的深度图像中，每个像素都包含了深度信息，这种深度信息是从红外摄像头的视角看，特定坐标处距离摄像头平面最近的物体到该平面的距离，单位是毫米。深度图像中，每个像素占用 2 字节，其中高 13 位记录深度数据，低 3 位记录用户 ID[1]。深度数据若为 0，则表示此坐标处的深度数据不可获得。由于本研究中仅使用深度数据，骨骼跟踪引擎关闭，用户 ID 的 3 位数据始终为 0。深度图像同样支持多个分辨率，本研究中采用分辨率为 640×480、帧率为 30fps 的 16bit 深度图像。

在深度图像中，其 X、Y 坐标表示像素，而 Z 坐标表示深度（米），单位并不相同，也不能与物理场景中的实际位置对应，需要进行坐标转换。经转换后的图像位于摄像机坐标系下，原点为红外摄像头中心，X、Y 轴与图像 X、Y 轴平行，Z 轴为红外摄像头光轴，与图像平面垂直，各坐标轴的单位均为米[1]。但很多时候 Kinect 传感器被放置在非水平表面上或仰角没有处于零位，此时摄像机坐标系的 Y 轴不与水平面垂直（或 Y 轴不与重力方向平行），因而后面的校正过程是必要的。除此之外，为了使彩色图像像素点与深度图像像素点进行对应，还需要额外的对齐工作。

从程序中获取 Kinect 的彩色图像和深度图像数据流通常有两种方式，即轮询方式和事件方式[1]。事件方式可以在彩色图像或深度图像准备就绪时自动触发，响应比较及时，编程更为优雅，因而本研究中选用这种方式。

2.1.3 PCL 中点和点云数据

上述经过坐标变换和对齐操作的彩色和深度图像，其中的每个像素点其实已经对应了物理场景中的实际位置，可以认为是一个点云了。但还需要将其存入 PCL 通用的数据结构中进行后续处理。

PCL 中内定义了大量的点类型。在本研究中，需要记录每个点的三维空间坐标以及 RGB 三通道颜色，因此选用了 PointXYZRGBA 数据结构。这一数据结构由两个联合构成。第一个联合用于表示坐标，由两个成员构成：第一个成员是大小为 4 的单精度浮点数数组 data[4]；第二个成员是包含 x、y、z 三个单精度浮点数的结构。可以通过 data[0/1/2]或 x/y/z 两种方式来存取点的坐标，而 data[3]的存在主要是为了存储对齐的需要，同时还可以用作一些运算的标志。第二个联合用于表示颜色，由两个成员构成：第一个成员是大小为 4 的单精度浮点数数组 data_c[4]；第二个成员是包含一个无符号整型数 rgba 的结构。这种构成方式同样是为了存储对齐的需要，满足 SSE 指令运行条件，提高运行效率[9]。此外还有一种包含坐标和颜色信息的点类型 PointXYZRGB，它使用一个单精度浮点数 rgb 来存储颜色，但这种点类型已过时。

PCL 中定义的点云类型为模板类型 PointCloud。它包含一些常用的字段：宽度 width，整型数，对于有序点云，如 Kinect 捕获的原始点云，它反映每一行有多少个点；对于无序点云，它反映点云中共有多少点。高度 height，整型数，对

于有序点云，它反映每一列有多少个点；对于无序点云，它始终为 1。点 points，点的容器向量，包含点云所存储的所有点。密集的 is_dense，布尔型，表明点云中没有坐标为 Inf / NaN 值的点[10]。

2.1.4 PCD 文件格式

在程序中获取了 Kinect 传感器捕获的点云数据后，还需要将点云数据以一种合适的格式存储到文件中。PCD 文件是 PCL 存储点云的专用格式。目前的 PCL 版本使用的 PCD 文件版本为 0.7。

PCD 文件的文件头包含所存储点云的基本信息，全部以 ASCII 编码。其中的每个字段占一行，行尾使用换行符“\n”分隔。基本字段包括：版本 VERSION，为 0.7；字段 FIELDS，指定每个点包含哪些字段，这里为 x y z rgb；大小 SIZE，指定每个字段的大小，字段的大小与其类型有关，这里 4 个字段均为单精度浮点数或无符号整型数，因此大小为 4 4 4 4；类型 TYPE，指定每个字段的类型，这里为 F F F U；宽度 WIDTH 和高度 HEIGHT，定义如上节所述；点数 POINTS，指定点云包含的总点数；数据类型 DATA，可以是文本“ascii”或二进制“binary”[8]。

PCD 文件的文件体紧跟文件头。如果是文本形式，则每个点的数据占一行，每个字段间用空格隔开；如果是二进制格式，则是程序 PointCloud 数据结构的 points 点元素容器向量的一个完整拷贝[11]。二进制的 PCD 文件一方面占用空间比较小，另一方面读取和存储都比较快速，因而在本研究中使用这种 PCD 文件。

2.2 实现方法

2.2.1 程序的使用规范

本程序可执行文件名为 *grab.exe*，没有运行参数。

程序开启后，会出现三个窗口。命令行窗口显示程序的基本信息，当保存彩色图像和三维点云到文件后，会在此显示保存的序号，如图 2.2.1。屏幕左上方的窗口显示实时彩色图像，其分辨率为 640×480；可以通过上下或左右拖拽调整彩色图像的亮度和对比度；如图 2.2.2。右边的窗口显示实时三维点云，它是 640×480 的有序点云；左下角可以显示帧率，由于计算机性能限制，帧率往往达不

到 30fps；可以通过拖拽来改变观察视角；如图 2.2.3。在彩色图像窗口或三维点云窗口中按下空格键，可以分别保存彩色图像和三维点云到 *image?.png* 和 *cloud?.pcd*，其中的问号表示序号，会随着每次保存而递增。

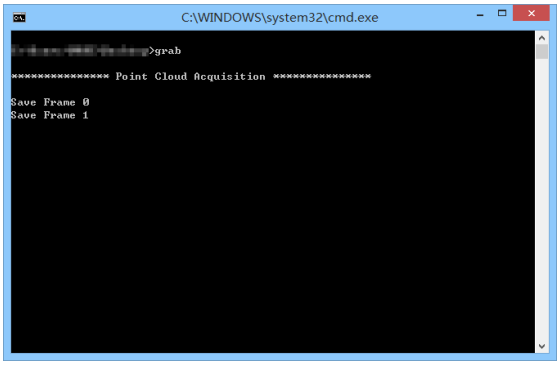


图 2.2.1. 命令行窗口

图 2.2.2. 彩色图像窗口

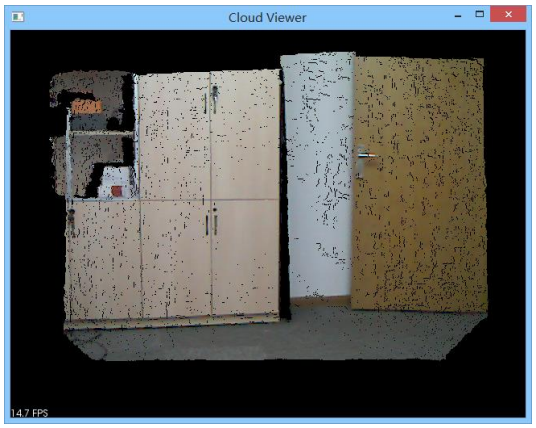


图 2.2.3. 三维点云窗口

2.2.2 程序的具体实现

本程序源文件名为 *grab.cpp*。

程序首先创建 IO 模组中的 OpenNIGrabber 对象，用于捕获 Kinect 数据，并将彩色图像回掉函数和三维点云回掉函数与其绑定。这两个回掉函数结构相同，均在开始时锁定互斥锁防止中断，并将获取的数据指针赋给两个全局的数据指针供主循环调用，之后解锁互斥锁。随后创建 Visualization 模组中的 ImageViewer 对象和 PCLVisualizer 对象，用于显示彩色图像和三维点云，其中 PCLVisualizer 对象还需要设定初始观察视角。这两个对象还要绑定键盘响应回掉函数，此函数判断按下的键是否为空格键，若是则更新序号并设定保存标志。进入主循环之前还需要初始化保存文件的相关变量，并打开 OpenNIGrabber 对象的数据流。在主

循环中，分别处理彩色图像数据和三维点云数据。对于彩色图像数据，应先锁定互斥锁，若没有初始化过，则应先初始化一些存储变量和窗口大小，否则直接显示图像，如果保存标志为真还应将彩色图像保存到文件，之后解锁互斥锁。对于三维点云数据，与前者基本一致，不再赘述。处理完两种数据后，刷新两个窗口。主循环结束条件为两个窗口中任何一个被关闭，此后应关闭 OpenNIGrabber 对象的数据流。整个流程见图 2.2.4。

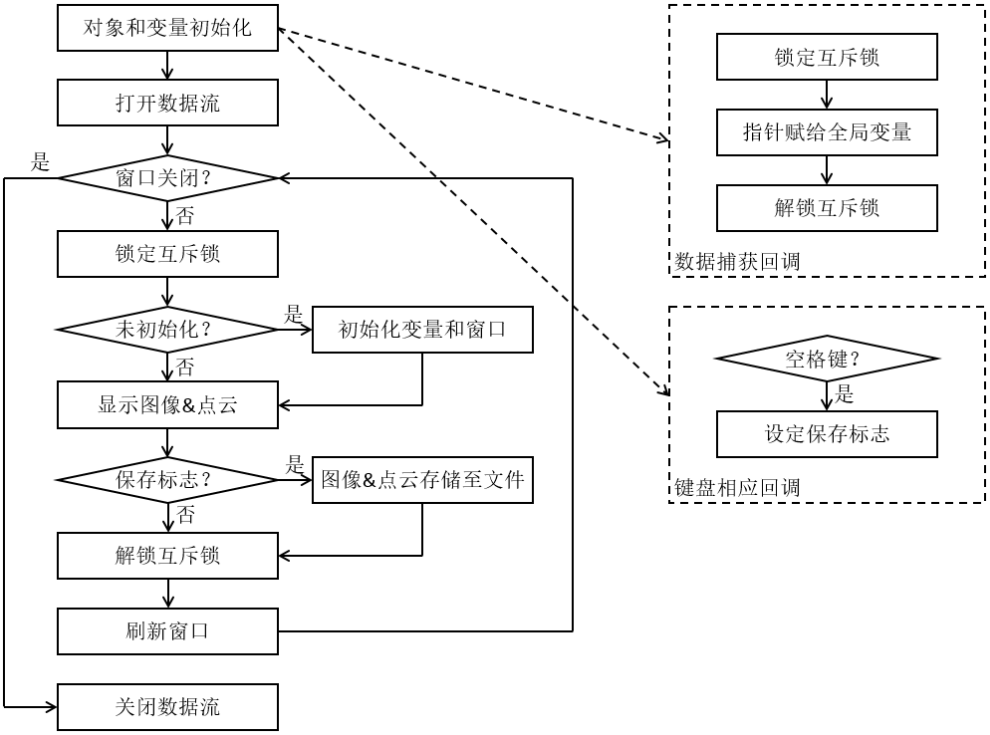


图 2.2.4. 点云获取程序流程图

2.2.3 装置搭建和拍摄过程

用于植物三维点云获取的拍摄装置由 Kinect 传感器和电动转台构成。拍摄的场景应较为空旷，便于分割出植物点云。转台平置于地面上，Kinect 传感器放置在距离转台一定距离的位置处，并调整 Kinect 传感器的姿态使转台大约位于视野的中部。Kinect 传感器与转台之间的距离有一定的要求：考虑 Kinect 传感器的可视距离为 0.8~4.0 米，它与转台之间的距离应当控制在比这略小的范围内，如 1.0~3.0 米的范围内；同时待测植株与转台应当完整的包含在 Kinect 传感器的视野内，同时尽可能的靠近传感器。如果待测植物较高大，则对 Kinect 传感器的放置高度和仰角也有一定要求：Kinect 传感器应尽可能放置在植株的中央高度处，

以尽可能完整的得到植株的上部和下部的点云，减少遮挡的影响；Kinect 传感器光轴与转台平面的角度也不能过小，通常光轴应略偏下，保证能够获取到转台平面大部分点的有效数据，用于后面的校正操作。根据上述几个原则，可以确定 Kinect 传感器放置的大致位置，如图 2.2.1 所示。

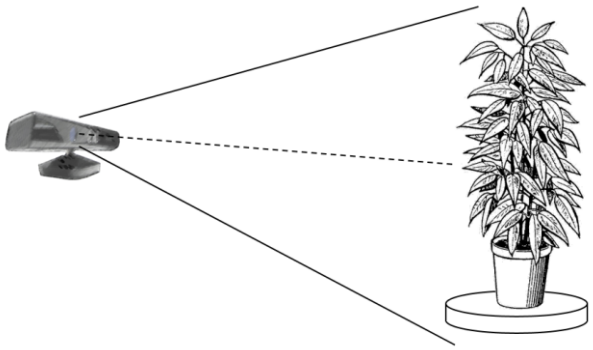


图 2.2.5. 拍摄装置示意图

本研究中，要拍摄每个植株 16 个不同角度侧面的点云，因而要使转台大约每转过 22.5°时停止一次，进行拍摄。为了达到这一目的，需要在转台上均匀的做一些标记，以便在转动中记录行程。本研究中，过转台中心平直均匀的贴了 8 条胶带作为标记，如图 2.2.2 中的红色和蓝色胶带。其他三个红色点标是用于点云校正的参考标记，将在后文中详述。工作中，在转台周围的地面做一个固定标记，转台每转动至标记线末端靠近该固定标记时，即可认为转台转过了大约 22.5°，此时可以停止转台并进行拍摄。

图 2.2.6. 标记过的转台

根据上述方法，首先拍摄一次转台空载时的场景点云，用于点云校正；再将待测植株放置在转台上，令转台转动，每隔约 22.5°拍摄一次。这样共可获得 17 个点云文件及其对应的彩色图像，分别命名为 cloud[0-16].pcd 和 image[0-16].pcd，其中转台空载的一组序号为 0。

2.3 结果分析与讨论

2.3.1 实验结果

对三个待测植株各拍摄了包括转台空载场景和 16 个角度植株场景的 17 组图像和点云。拍摄程序需要 2~3 秒进行初始化，存储每组彩色图像和三维点云耗时少于 1 秒；拍摄过程人工进行，整个过程耗时 5 分钟以内。部分结果如图 2.3.1-12 所示。

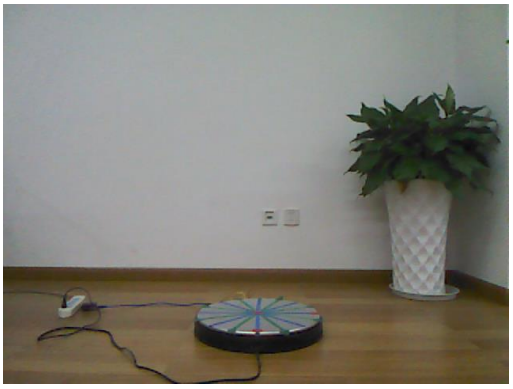


图 2.3.1. 植株 A 空载彩色图像

图 2.3.2. 植株 A 空载三维点云

图 2.3.3. 植株 A 正面彩色图像

图 2.3.4. 植株 A 正面三维点云

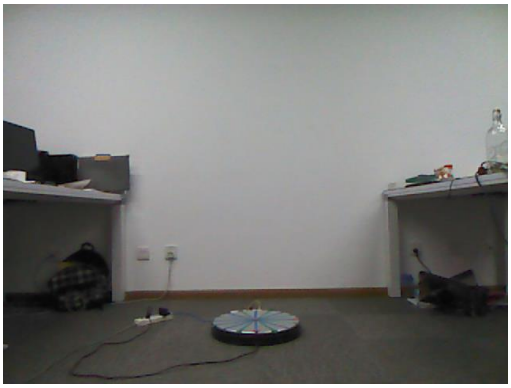


图 2.3.5. 植株 B 空载彩色图像

图 2.3.6. 植株 B 空载三维点云

图 2.3.7. 植株 B 正面彩色图像

图 2.3.8. 植株 B 正面三维点云

图 2.3.9. 植株 C 空载彩色图像

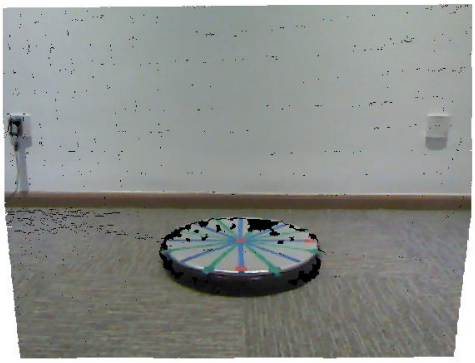


图 2.3.10. 植株 C 空载三维点云

图 2.3.11. 植株 C 正面彩色图像

图 2.3.12. 植株 C 正面三维点云

2.3.2 结果分析

由上图可以看出，所得到的结果基本满足要求。但同时存在着一些问题值得引起注意：

Kinect 传感器获取的彩色数据点和深度数据点不完全对齐。现象即为在三维点云中，植株冠层叶片边缘的颜色为背景的颜色（灰白色），而非叶片中心的颜色（绿色）。

Kinect 传感器获取的点云数据在物体边缘处存在大量无效点。可以看出在三维点云中，植株边缘存在着较厚的黑色缝隙，就是由无效点构成的。这些无效点的形成原因在于，在物体边缘处 Kinect 发射的激光散斑图案不完整，不能运算出正确的深度数据。这种现象会导致对于叶片较稀疏的植株，一些在内部的叶片虽然可以观察到，却无法得到有效的对应坐标点，如植株 B。

由于视角原因，植株上方的无效点较多。上部由于与 Kinect 传感器光轴夹角较大，遮挡较多，相对距离也较远，因而无效点较多。这导致了最终的植物点云上方较为稀疏。

距离较远或 Kinect 传感器光轴与转台平面夹角较小时，转台表面无效点较多。典型的例子如植株 B 的空载点云。转台表面的点云质量不好，会极大地影响后面的校正过程。

2.3.3 工作展望

可能的改进方案包括以下几个方面：

当前为手动控制转台的转动与停止，粗略的估计转动角度，可考虑改良转台设计，自动每隔一定角度停止，并于点云获取程序相连接自动拍摄。可以利用单片机配合光电编码器实现。

拍摄中每转动转台都会引起植株枝叶的震动，需要一定时间待其静止才可以拍摄，可考虑改为固定植株而绕着植株转动 Kinect 传感器。这样会需要一个圆形轨道来实现，同时占地较大。当然也可以直接使用多台 Kinect 传感器来构造阵列获取多角度点云。

针对无效点较多的问题，可考虑在保存一幅点云时实际对多幅点云数据进行融合。经观察发现，Kinect 传感器获取三维点云的过程并不是非常稳定的，有些点在一些帧中是有效的，在有些帧中却是无效的；同时 Kinect 传感器获取的点数据还存在一定的噪声。对多幅点云进行融合，可以利用求取平均值有效地抑制噪声，同时还可以大大减少无效点。

植株上方点较为稀疏，可以通过增加一个俯视拍摄的点云来得到解决。这需要一个较高的支架将 Kinect 传感器放置于植株的正上方。

第 3 章 植物三维点云的校正

3.1 基本原理

3.1.1 主要工作思路

植物三维点云的校正其实是对拍摄系统的标定，通过计算 Kinect 传感器与转台之间的相对位置关系，来校正三维点云的倾斜情况。这一部分的主要工作在于首先确定转台平面的中心的法向量，进而计算出对整个点云从摄像机坐标系变换到转台坐标系的变换矩阵。对于转台平面相关参数的确定需要一些人为输入，本研究中采用最简单的不共线三点确定平面的原理进行相关工作。

3.1.2 平面法向量

三维空间中，平面法向量是垂直于该平面的三维向量。平面的法向量不具有唯一性：从方向来说，指向平面两边的垂直向量均为平面的法向量；从大小来说，若向量 $\boldsymbol{n}$ 为某平面的法向量，则向量 $k\boldsymbol{n}$ 也是该平面的法向量。在本研究中，为了避免多义性，规定转台平面的法向量为垂直于转台平面并指向上方的单位向量。

在已知平面上不共线三点的情况下，平面的法向量可以通过由三点构成的两个向量的叉积来求得。设三点分别为 $O(x_0, y_0, z_0)$、$A(x_1, y_1, z_1)$和 $B(x_2, y_2, z_2)$，则可构成两向量如：

$$\boldsymbol{OA} = (x_1 - x_0)\,\boldsymbol{i} + (y_1 - y_0)\,\boldsymbol{j} + (z_1 - z_0)\,\boldsymbol{k} = x_{OA}\,\boldsymbol{i} + y_{OA}\,\boldsymbol{j} + z_{OA}\,\boldsymbol{k} \tag{3.1.1}$$

$$\boldsymbol{OB} = (x_2 - x_0)\,\boldsymbol{i} + (y_2 - y_0)\,\boldsymbol{j} + (z_2 - z_0)\,\boldsymbol{k} = x_{OB}\,\boldsymbol{i} + y_{OB}\,\boldsymbol{j} + z_{OB}\,\boldsymbol{k} \tag{3.1.2}$$

则该平面的一个法向量为：

$$\begin{aligned}\boldsymbol{n} &= \boldsymbol{OA} \times \boldsymbol{OB} \\ &= (y_{OA} z_{OB} - y_{OB} z_{OA})\,\boldsymbol{i} - (x_{OA} z_{OB} - x_{OB} z_{OA})\,\boldsymbol{j} + (x_{OA} y_{OB} - x_{OB} y_{OA})\,\boldsymbol{k}\end{aligned} \tag{3.1.3}$$

由此得到平面的一个法向量经过一些处理可得到符合要求的特定法向量。首先对法向量进行单位化，即用法向量的每个系数除以向量的模得到新的系数。之后对法向量的方向进行调整，由于在摄像机坐标系下，竖直向上的方向约为 y 轴负方向，因而若法向量中 $\boldsymbol{j}$ 的系数为负，则无需调整，否则三个系数均取相反数。

3.1.3 点云的几何变换

点云的几何变换其实是一个对坐标空间进行线性变换的过程。将三维空间中的点坐标*(x, y, z)*包装在一个四维向量*[x y z 1]*^{T}里。利用一个 4×4 的方阵与向量相乘来达成各种几何变换，即：

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.1.4}$$

这一个 4×4 的变换矩阵可以分成 4 块，每一块均能产生不同的几何变换效果[12]。

第一块为方阵左上的 3×3 矩阵，能够产生比例、对称和旋转等效果。其中比较重要的是绕某一坐标轴的旋转变换，在右手坐标系下旋转角度的正负以右手法则确定。若绕 x 轴旋转 θ 角，则此矩阵为：

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{3.1.5}$$

若绕 y 轴旋转 θ 角，则此矩阵为：

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.1.6}$$

若绕 z 轴旋转 θ 角，则此矩阵为：

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1.7}$$

第二块为方阵右上角的 3×1 矩阵，能够产生沿三个坐标轴的平移变换，即：

$$\begin{bmatrix} a_{03} \\ a_{13} \\ a_{23} \end{bmatrix} = \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} \tag{3.1.8}$$

第三块为方阵左下角的 1×3 矩阵，能够产生透视变换。在刚体变换中不需要使用透视变换，因此该矩阵中各元素常为 0，即：

$$[a_{30} \quad a_{31} \quad a_{32}] = [0 \quad 0 \quad 0] \tag{3.1.9}$$

第四块为方阵右下角的 1×1 矩阵，能够产生比例缩放变换。在刚体变幻中不需要比例缩放变换，因此该矩阵中的元素常为 1，即：

$$[a_{33}] = [1] \tag{3.1.10}$$

3.2 实现方法

3.2.1 程序的使用规范

本程序可执行文件名为 *calib.exe*，有多个运行参数：输入文件名，必要参数，由“*.pcd”指定，指定转盘空载的场景点云文件，读取用于校正的点云数据；转台中心点坐标，必要参数，由“-c X_O, Y_O”指定，指定转台中心点在对应彩色图像上的坐标，决定了校正变换后的坐标原点；参考点 A 和参考点 B 坐标，必要参数，由“-a X_A, Y_A”和“-b X_B, Y_B”指定，指定另外两个校正标记点在对应彩色图像上的坐标，用于计算转台平面的法向量；输出文件名，可选参数，由“-o *”指定，决定输出的变换矩阵文件名，默认为“calib”。完整的运行命令如下：

```
calib.exe *.pcd -c XO,YO -a XA,YA -b XB,YB [-o *]
```

程序开启后，会出现一个命令行窗口，如图 3.2.1。首先显示了输入点云文件名，以及利用给定的中心点和参考点的彩色图像坐标找出的三维点云中对应点的坐标，并计算了转台平面的法向量。利用这些数据计算变换矩阵，并显示了旋转矩阵和平移向量，以及变换后的中心点和参考点的坐标。最后显示将变换矩阵数据保存到了文件中。

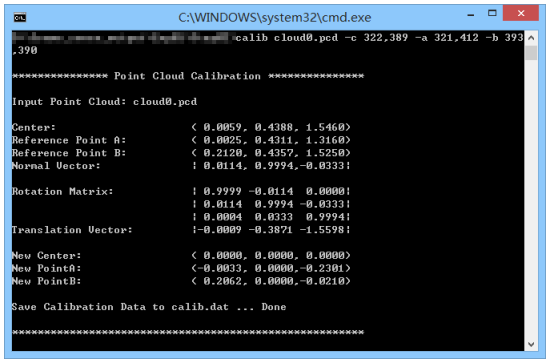


图 3.2.1. 命令行窗口

3.2.2 程序的具体实现

本程序源文件名为 *calib.cpp*。

程序首先利用 Console 模组分析命令行执行参数，把相关参数存入变量，如果必要参数缺失则报错退出。利用 IO 模组读入 PCD 点云文件。对于中心点和参考点，利用其彩色图像坐标来读取三维点云的对应点，由于原始点云为有序点云，可以通过 X + Y × Width 的序号来找到对应点。之后利用三个点的三维坐标求转台平面法向量，计算方法如前所述。由于得到的法向量为单位向量，可以很方便的求出法向量与三个坐标轴的夹角，进而构造旋转矩阵。构造旋转矩阵可以分两步进行：先绕 X 轴旋转，使法向量旋转至 X-Y 平面上；再绕 Z 轴旋转，使法向量与 Y 轴平行。两个旋转矩阵相乘可以得到最终的旋转矩阵。读取旋转变换后的中心点坐标，对各个坐标取反即可得到平移向量。点云的几何变换可以通过 Registration 模组的 transformPointCloud 函数完成。最后将旋转矩阵的 9 个元素和平移向量的 3 个元素以二进制格式存入文件。整个流程见图 3.2.2。

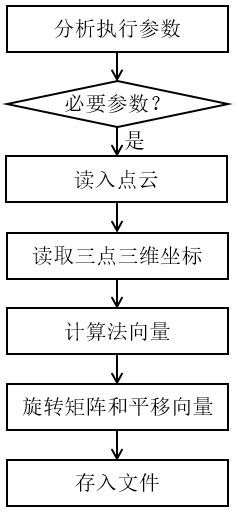


图 3.2.2. 点云校正程序流程图

3.2.3 校正参考标记

本研究采用的校正参考标记如图 2.2.6 所示。使用红色胶带粘贴在转台表面上，构成约 3cm×3cm 大小的方形标记。其中一个标记粘贴在转台中心位置，另外两个标记与转台中心成 90° 粘贴在转台边缘处，且两个标记到中心的距离大约 20cm。之所以使用较大的标记，是由于 Kinect 传感器获取的彩色图像分辨率

有限，且传感器光轴与转台平面的夹角不是很大，标记过小可能会导致在彩色图像中难以分辨。

在拍摄时，需要拍摄转台空载的场景点云和对应的彩色图像，用于校正操作。此时要注意三个校正参考标记点位置处的数据点必须是有效的。为此，应转动转台，使两个转台边缘标记点位于靠近 Kinect 的位置。在本研究中，使一个边缘标记点处于 Kinect 与转台中心的连线上（X 坐标与中心标记点相近），另一个边缘标记点处于转台中心点的右侧（Z 坐标与中心标记点相近），如图 3.2.3 所示。

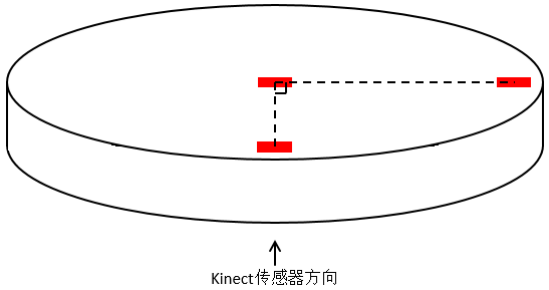


图 3.2.3. 转台空载场景拍摄时的转台状态

在校正时，首先需要人工指定中心标记点和两个边缘标记点在彩色图像上的坐标，并作为运行参数传入校正程序。由于标记点相对较大，在彩色图像上是一片红色的区域，应当近似选用区域的中心点。将最后算出的变换矩阵信息存入 calib.dat 文件，用于后面的点云预处理。

3.3 结果分析与讨论

3.3.1 实验结果

对三个植株的转台空载点云提取校正信息。校正程序的执行时间少于 1 秒。

对于植株 A 的转台空载彩色图像，选取中心点 C 为(322, 389)，边缘点 A 为(321, 412)，边缘点 B 为(393, 390)，如图 3.3.1 所示（局部，放大 6 倍）。

在原始点云中：中心点 C 坐标为(0.0059, 0.4388, 1.5460)，边缘点 A 为(0.0025, 0.4311, 1.3160)，边缘点 B 为(0.2120, 0.4357, 1.5250)；转台平面法向量为[0.0114 0.9994 -0.0333]。

在校正点云中：中心点 C 坐标为(0.0000, 0.0000, 0.0000)，边缘点 A 为(-0.0033, 0.0000, -0.2301)，边缘点 B 为(0.2062, 0.0000, -0.0210)；转台平面法向量为[0.0000 1.0000 0.0000]。

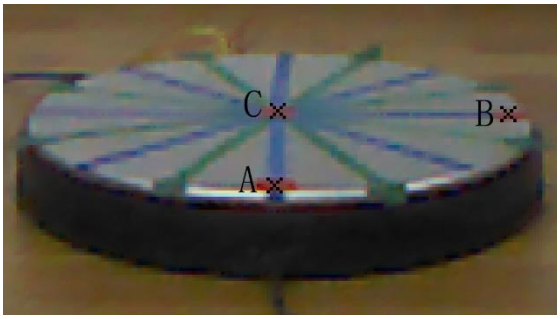


图 3.3.1. 植株 A 转台空载彩色图像局部放大

对于植株 B 的转台空载彩色图像，选取中心点 C 为(320, 399)，边缘点 A 为(321, 412)，边缘点 B 为(370, 398)，如图 3.3.2 所示（局部，放大 6 倍）。

在原始点云中：中心点 C 坐标为(0.0021, 0.6541, 2.1530)，边缘点 A 为(0.0057, 0.6545, 1.9920)，边缘点 B 为(0.2071, 0.6500, 2.1530)；转台平面法向量为[0.0200 0.9998 0.0030]。

在校正点云中：中心点 C 坐标为(0.0000, 0.0000, 0.0000)，边缘点 A 为(0.0036, -0.0000, -0.1610)，边缘点 B 为(0.2051, 0.0000, 0.0000)；转台平面法向量为[0.0000 1.0000 0.0000]。

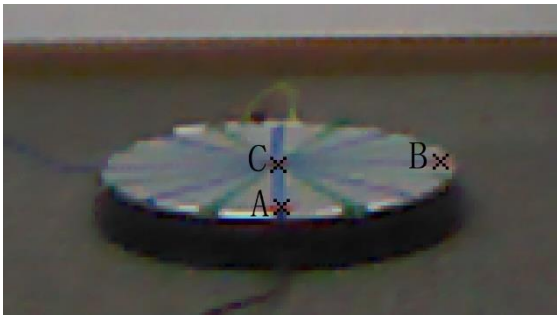


图 3.3.2. 植株 B 转台空载彩色图像局部放大

对于植株 C 的转台空载彩色图像，选取中心点 C 为(317, 313)，边缘点 A 为(317, 348)，边缘点 B 为(406, 313)，如图 3.3.3 所示（局部，放大 6 倍）。

在原始点云中：中心点 C 坐标为(-0.0058, 0.1700, 1.2140)，边缘点 A 为(-0.0048, 0.2102, 1.0170)，边缘点 B 为(0.2000, 0.1700, 1.2140)；转台平面法向量为[-0.0000 0.9798 0.2000]。

在校正点云中：中心点 C 坐标为(0.0000, 0.0000, 0.0000)，边缘点 A 为(0.0009, 0.0000, -0.2011)，边缘点 B 为(0.2058, 0.0000, 0.0000)；转台平面法向量为[0.0000 1.0000 0.0000]。

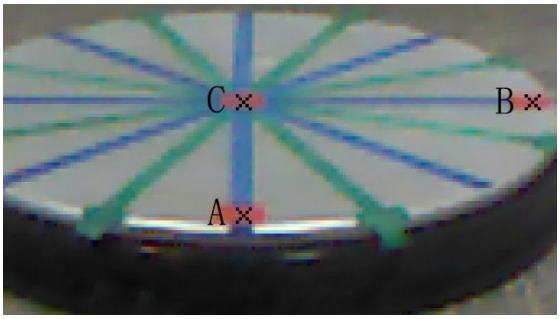


图 3.3.3. 植株 C 转台空载彩色图像局部放大

3.3.2 结果分析

结果显示，经过校正操作，各个点云的转台平面法向量均与重力方向平行，基本达到了预定的校正效果。但同时也发现了一些问题：

Kinect 传感器对深度数据的测量准确性会极大地影响校正的准确性。在本研究中，校正完全依赖于选取的三个校正标记点。如果 Kinect 传感器对这三个点的深度数据测量存在较大的误差，这种误差就会直接反应在校正效果和未来的测量准确度上。如在拍摄植株 B 的转台空载场景时，就发现由于转台距离 Kinect 传感器较远，转台表面的点质量不太好，且无效点较多。而校正结果也显示出了这一问题，A 点与 C 点的物理距离应为 20cm 左右，但在点云中距离仅为 16cm 左右，说明了从 Kinect 获得的原始数据存在误差，这也会影响后续测量过程的准确性。此外，各组结果的 AC、BC 点间距离都不尽相同，说明 Kinect 传感器获得的点云确实存在一定的随机误差或是畸变。

在彩色图像中选取参考标记点依赖于人工判断。在本研究中，参考标记点的选取相对宽松。仅有中心点关系到校正后转台坐标系的原点，需要相对准确，其余两个参考点只需要满足在转台表面且不共线的要求即可。但人工判断的参考点可能会出现失误，如选取的参考点不在转台表面上，而在转台侧面，则会引起很大的误差。

3.3.3 工作展望

可能的改进方案包括以下几个方面：

对于在彩色图像中选取参考标记点的操作，可以使用图像处理技术自动完成。可能的实现方法是根据一定的颜色阈值找出可能的标记点区域，迭代收缩直到整个图像上仅存在三个连通区域，在找到三个连通区域的中心点即可。

对于过分依赖三个参考点的问题，可以考虑使用转台表面的所有有效点来估算法线。可能的实现方法是首先利用滤波或分割的方法分离出转台表面的点，再对这些点进行平面拟合来计算法线。

针对 Kinect 深度数据的误差或无效点较多的问题，可以采用上一章所述的一次性采集多幅点云进行数据融合的方法进行抑制。

为了在校正点云的同时抑制镜头畸变，还可以采用棋盘格标定板作为校正参考图案。

第 4 章 植物三维点云的预处理

4.1 基本原理

4.1.1 主要工作思路

这一部分的主要工作在于对包含多角度植物点云的原始场景点云进行一些预处理操作，为后面的配准工作做准备。具体的任务包括利用前面得到的变换矩阵数据对点云进行校正；再将点云从转台坐标系（右手系，X-Y 平面竖直，Z 轴正方向指向外）变换到更为符合人们习惯的世界坐标系（右手系，X-Y 平面水平，Z 轴正方向指向上），如图 4.1.1；还要利用直通滤波将不属于植株的背景数据点去除；并去除一些由于噪声或其他原因造成的离群点；最后进行对多角度点云集中的各个点云进行姿态推测（粗配准）。以植株 A 的第 8 角度点云（背面点云）为例，这一部分的流程如图 4.1.2。

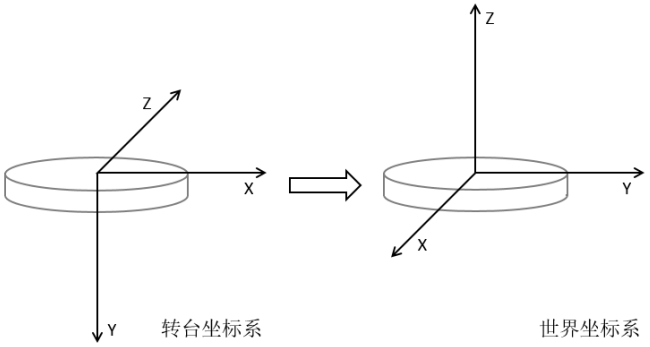


图 4.1.1. 转台坐标系与世界坐标系示意图

图 4.1.2. 三维点云预处理流程

4.1.2 直通滤波

直通滤波是一种最简单的滤波器。它允许某个字段在一定限制范围内的点通过，而将其余的点滤除。同时，它还能去除点云中的无效数据点。

在本研究中，使用直通滤波来去除背景点而仅保留属于植株的点。由于拍摄场景较为空旷，植株和转台周围少有其他物体，因此可以用一个箱型范围将植株隔离出来，即通过直通滤波对三个坐标进行滤波。又因为在世界坐标系下，坐标原点为转台中心点，Z 轴即为植株的中轴，则可以对称地划定 X、Y 坐标的通过范围，单向地划定 Z 坐标的通过范围，如图 4.1.3 所示。

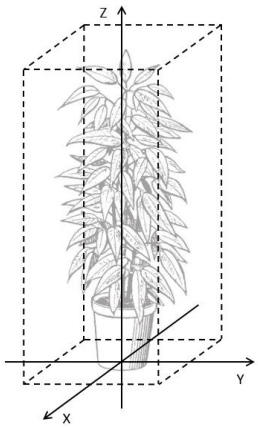


图 4.1.3. 世界坐标系下直通滤波通过范围示意

4.1.3 统计性离群点去除

统计性离群点去除利用点的近邻统计特性来滤除离群点。它两次遍历整个点云。第一次遍历计算每个点到其 K 个近邻点间的平均距离。假设这些距离符合高斯分布，其分布函数形状由均值和标准差决定，则可以认为距离超出标准范围的点为离群点。因而要计算这些距离的均值和标准差，用于确定距离阈值——均值 + 标准差乘数阈值 × 标准差。在第二次遍历时，距近邻点平均距离超过该距离阈值的被判为离群点而排除，反之则保留[13]。

在本研究中，为了防止在去除离群点的同时删去大量的有用点，通常将标准差乘数阈值设置为 3.0，对应正态分布中的 99.87%的概率保留。

4.1.4 姿态推测（粗配准）

利用转台旋转植株拍摄的多角度点云，每一个植株点云在坐标系中都处在同一个位置，不利于配准时寻找对应点，因而直接配准效果较差。因此，应当事先进行粗略的姿态推测，也就是使每个植株点云都处在它应当存在的位置附近，再进行精细配准的效果会更好。

在本研究中，由于植株点云处于世界坐标系下，坐标系的 Z 轴即近似等于植株的中轴，姿态推测就只需要使各个植株点云绕 Z 轴旋转一定的角度即可。根据前述的点云获取流程可知，第一个植株点云不需要进行旋转，之后的每个植株点云都需要递增旋转 22.5°。由于之后还需要进行精细配准工作，姿态推测的结果可以不必太准确，即相邻的两个点云的重叠部分之间允许存在一定的偏移，因此这一工作也被成为粗配准。

4.2 实现方法

4.2.1 程序的使用规范

本程序可执行文件名为 *preproc.exe*，有多个运行参数：输入变换矩阵文件名，必要参数，由“*.dat”指定，指定点云校正中生成的转换矩阵数据；输入点云文件名，必要参数，由“*.pcd”指定，指定需要进行预处理的原始点云文件，可以输入多个进行批处理；直通滤波参数，可选参数，由“-p R_X, R_Y, R_Z”指定，指定分割植株点云的包围箱体的大小，默认均为 1.0；统计性离群点去除参数，可选参数，由“-s K, Mul”指定，指定计算近邻点平均距离时使用的近邻点数和计算距离阈值时的标准差乘数阈值，默认分别为 30 和 1.0；姿态推测步进角，可选参数，由“-r Deg”指定，指定为粗配准而进行的递增旋转的步进角度，默认为 0.0；输出文件前缀，可选参数，由“-o *”指定，决定输出的点云文件名的前缀（输出文件名由前缀加序号构成），默认为“proc_cloud”。完整的运行命令如下：

```
preproc.exe *.dat *.pcd [*.pcd …]
[-p RX,RY,RZ] [-s K,Mul] [-r Deg] [-o *]
```

程序开启后，会出现一个命令行窗口，如图 4.2.1。首先显示了输入点云文件名，接着显示了程序主要进行的五步操作以及相关的参数，包括旋转矩阵和平移向量、直通滤波通过范围、统计性离群点去除参数和粗配准旋转步进角。此后开始分别处理每一个点云的数据，并在处理结束之后保存到文件中。

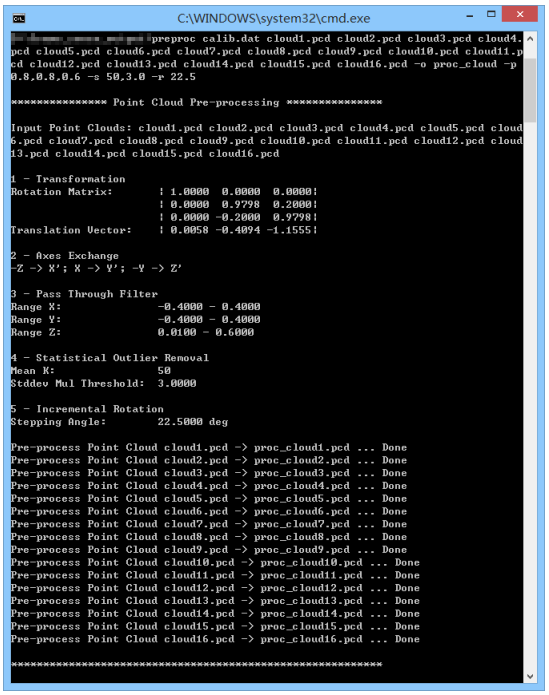


图 4.2.1. 命令行窗口

4.2.2 程序的具体实现

本程序源文件名为 *preproc.cpp*。

程序首先利用 Console 模组分析命令行执行参数，把相关参数存入变量，如果必要参数缺失则报错退出。读入校正用的变换矩阵数据，利用 IO 模组读入所有输入 PCD 点云文件，然后利用循环分别处理每个点云。首先利用 Registration 模组的 transformPointCloud 函数将点云从摄像机坐标系变换到转台坐标系，接着遍历整个点云，使每个点的 X 坐标为原 Z 坐标的相反数、Y 坐标为原 X 坐标、Z 坐标为原 Y 坐标的相反数，这样就使点云处于世界坐标系下了。为了分离出植株点云，利用三个 Filters 模组的 PassTrough 直通滤波对象分别选通 X 坐标的$-R_X/2 \sim R_X/2$ 范围、Y 坐标的$-R_Y/2 \sim R_Y/2$ 范围和 Z 坐标的 $0.01 \sim R_Z$ 范围。之所以滤除了 Z 坐标的 0~0.01 范围，是由于 Kinect 深度数据噪声的存在，校正变换后仍有一些转台表面的点的 Z 坐标处于这个范围内。之后利用 Filters 模组的 StatisticalOutlierRemoval 统计性离群点去除对象根据给定参数去除离群点。最后再次利用 transformPointCloud 函数递增旋转点云，并将点云存入 PCD 文件。整个流程见图 4.2.2。

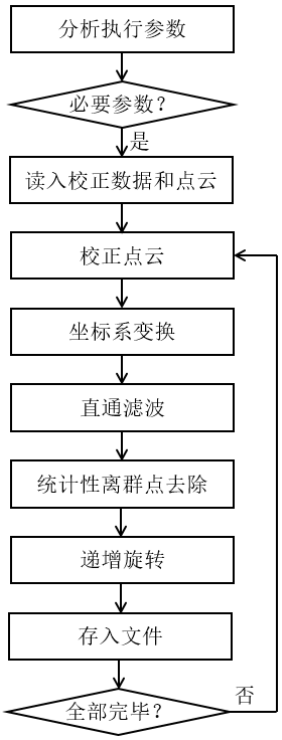


图 4.2.2. 点云预处理程序流程图

4.3 结果分析与讨论

4.3.1 实验结果

对每个植株的 16 个角度的点云进行了预处理。预处理程序对每个点云的处理时间少于 1 秒，与点云中的点数密切相关。将每个植株的第一个点云（正面）和全部点云（以不同颜色表示不同点云）的可视化效果展示如下：

对于植株 A，直通滤波参数 1.2、1.2、1.2，统计性离群点去除参数 50、3.0，其余保持默认。16 个角度的点云总处理时间约为 8 秒。

图 4.3.1. 植株 A 正面预处理点云

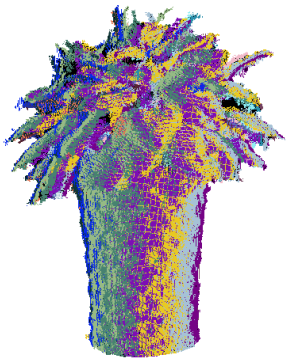


图 4.3.2. 植株 A 全部预处理点云

对于植株 B，直通滤波参数 1.5、1.5、1.5，统计性离群点去除参数 50、3.0，其余保持默认。16 个角度的点云总处理时间约为 9 秒。

图 4.3.3. 植株 B 正面预处理点云

图 4.3.4. 植株 B 全部预处理点云

对于植株 C，直通滤波参数 0.8、0.8、0.6，统计性离群点去除参数 50、3.0，其余保持默认。16 个角度的点云总处理时间约为 4 秒。

图 4.3.5. 植株 C 正面预处理点云

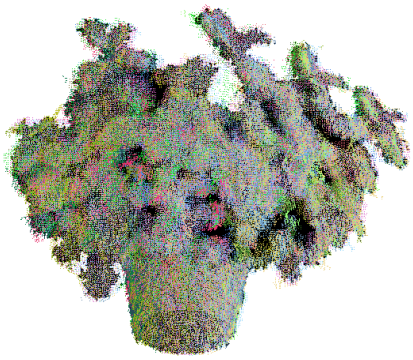


图 4.3.6. 植株 C 全部预处理点云

4.3.2 结果分析

由结果可以得知，预处理程序基本达成了预定的处理效果。

但不难看出，经过了直通滤波和统计性离群点去除，仍有一些属于转台表面的点没有被去除，这一现象在植株 B 的点云中尤为明显。这可能与植株 B 的校正效果相对较差有关。

此外，预处理的效果与滤波器的参数设置有很大关系，尤其是统计性离群点去除，对其近邻点数和标准差乘数阈值进行合理设置可以去除绝大多数离群点。但这种设置依赖于人工试验和经验。

4.3.3 工作展望

可能的改进方案包括以下几个方面：

将植株点云进行分离的操作不再使用需要指定参数的直通滤波，而使用自动化的编组、聚类或分割算法来实现。植株点云与背景点云具有不同的特点，例如植株点云在世界坐标系下位于中央位置、颜色相对背景较为均匀等。

去除离群点时的滤波器参数不再使用统一的指定的参数，而是自适应地设置滤波参数。可以根据一些条件或指标来决定最适宜的参数，例如滤除的点数不能过多、每个点的一定半径内都必须存在一定数量的近邻点等。

第 5 章 植物三维点云的配准

5.1 基本原理

5.1.1 主要工作思路

这一部分的主要工作在于对不同角度的经过粗配准的多个点云进行精细配准，减小乃至消除相邻点云之间的偏移，最后合成为单个完整的植株点云。这里主要利用成对配准的方法，将相邻的每两个点云之间进行配准。由于配准误差的存在是不可避免的，采用了一定的配准策略来使误差对配准结果的影响减小。

5.1.2 成对配准

成对配准的结果是一个刚性变换矩阵，它表示为了把一个点云（来源）完美地对齐另一个点云（目标）所需要进行的旋转和平移[14]。成对配准的常用步骤如图 5.1.1 所示，它表示算法的一次迭代过程。

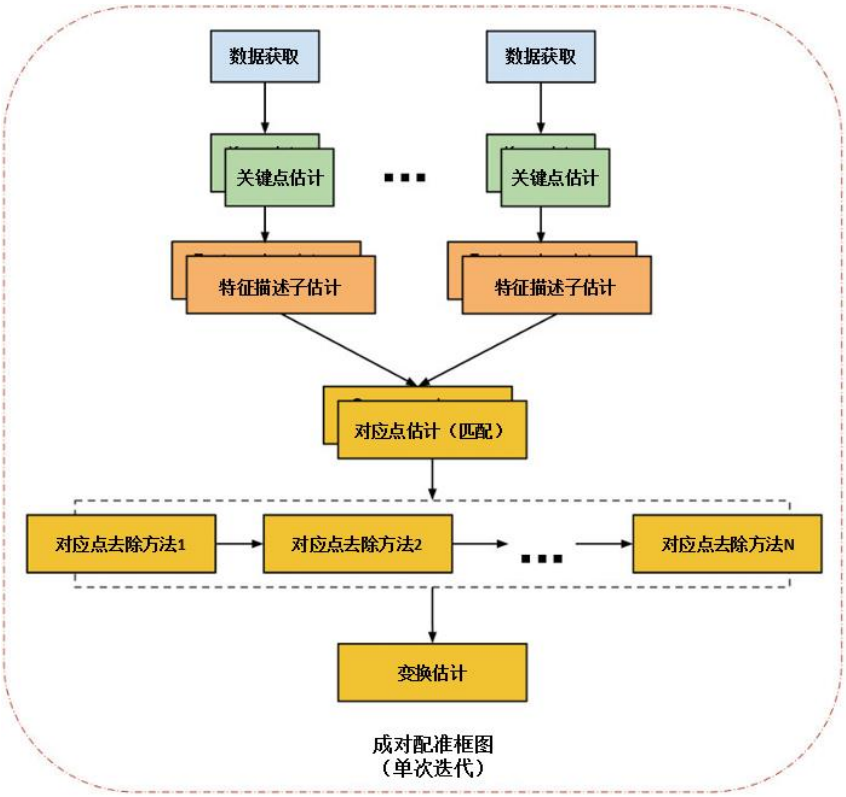


图 5.1.1. 成对配准单次迭代流程图

首先从两个点云找出能够最好地代表场景的兴趣点（即关键点）。这一步是为了减小进行对应点估计时的数据量，因为如果将包含 N 个点的整个点云作为输入，则会有 N^2 个可能的对应关系。本研究简单地使用向下采样的方法来提取关键点。

然后对每一个关键点，计算出一种特征描述子。将特征描述子的信息包装成向量，作为点之间相互比较的依据。在本研究中，使用每个点的曲率作为特征描述子。

从两个点云的特征描述子连同 X、Y、Z 坐标，根据特征和位置的相似性估计一系列对应点。寻找对应点是为了找到两个点云之间的重叠部分，根据方法的不同可以分为点匹配 / 特征匹配以及单向特征点估计 / 相互特征点估计。

由于数据往往是有噪声的，并不是所有的对应点都是有效的，所以应当去除不利于配准过程的错误对应点。这可以通过使用随机采样一致性，或者削减对应点数量而仅使用一定比例的对应点来达成。

最后，利用剩余的有效对应点估计从来源点云到目标点云的变换。这需要先利用对应点确定误差度量，再估计刚性变换来最小化误差度量，不断迭代来满足某种收敛准则。

在本研究中，前述的三个步骤由非线性迭代最近点算法进行。

5.1.3 体素网格向下采样

体素是一种非常直观的表达三维模型的方法，它把三维空间均匀分成三维网格，从而创建所谓的体积图像。图像中每个网格中的元素被称为体积像素，简称体素，也即是三维版本的像素。

对点云进行体素网格向下采样的方法非常直观：首先对输入的点云数据创建一个三维体素网格，也即把三维空间均匀地分成许多小箱体。然后，在每个体素中，将所有点使用它们的重心来代替，这样就达到了向下采样的效果。这种方法虽然直接用体素的中心来近似慢一些，但可以更准确地表现表面[15]。

在本研究中使用体素网格向下采样主要是为了在减小输入配准算法的数据量的同时，最大程度的保留点云的原始信息。

5.1.4 表面法线估计

确定一个表面在一个点处的法线的问题可以近似为估计表面的切平面的法线，这使得法线估计问题变成最小二乘平面拟合的问题。因此，表面法线估计的解法就成了对某个点的近邻点建立的协方差矩阵的特征向量和特征值的分析（即主成分分析）。对每个点 $\boldsymbol{p_i}$，协方差矩阵 $\boldsymbol{C}$ 如下：

$$\boldsymbol{C} = \frac{1}{k}\sum_{i=1}^{k}(\boldsymbol{p_i} - \overline{\boldsymbol{p}}) \cdot (\boldsymbol{p_i} - \overline{\boldsymbol{p}})^T \tag{5.1.1}$$

$$\boldsymbol{C} \cdot \boldsymbol{v_j} = \lambda_j \cdot \boldsymbol{v_j}, \quad j \in \{0, 1, 2\} \tag{5.1.2}$$

其中，k 是需要考虑的 $\boldsymbol{p_i}$ 的近邻点个数；$\overline{\boldsymbol{p}}$ 是近邻点的三维质心；λ_j 是协方差矩阵的第 j 个特征值；$\boldsymbol{v_j}$ 是第 j 个特征向量。

由于无法通过数学方法决定法向量的符号，通过主成分分析得到的法向量方向是不明确的，在整个点云中并不一致。为了解决这一问题，可以首先确定一个视点 $\boldsymbol{v_p}$，并且使所有的法向量都一致的朝向视点，即满足下式：

$$\boldsymbol{n_i} \cdot (\boldsymbol{v_p} - \boldsymbol{p_i}) > 0 \tag{5.1.3}$$

表面曲率通过特征值与协方差矩阵之间的关系进行估计，如下式：

$$\sigma = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{5.1.4}$$

一个点处的表面法线需要用其近邻点来估计，因此选择合适的近邻范围就十分重要[16]。如果近邻范围过小，法线估计收噪声的影响较大；如果近邻范围过大，则近邻点集可能包含相近的其他表面的点，从而使得法线估计失真。

在本研究中使用表面发现估计来计算每个点处的表面曲率，作为配准时判别对应点的依据之一。

5.1.5 迭代最近点配准

迭代最近点是一种用来最小化两个点云之间偏移的算法，常用来从不同的扫描中重建三维表面。算法主要包括四个步骤：寻找对应点、去除错误的对应点、利用正确的对应点估计变换和迭代[17]。

该算法中寻找对应点主要依靠“最近”这一指标，即对于来源点集中的某个点，选取目标点集中在多维空间中距其最近的点为对应点。在本研究中，除了 X、Y、Z 三维坐标，还加上了曲率作为第四维坐标，使得算法寻找对应点时在四维空间中进行搜索。

该算法中去除错误的对应点采用随机采样一致性。它首先随机选取一个样本子集，利用最小二乘估计对其计算模型参数，再计算所有样本点与模型的偏差，将偏差与阈值进行比较以判断是合群点还是离群点，并不断迭代最终得到合群点个数最多的模型参数作为最佳模型。这种方法可以排除离群点的干扰，达到全局最优[18]。

该算法中估计变换有多种方式：一种是基于奇异值分解确定对应点之间变换关系的方法，是一种线性方法。奇异值分解是方阵特征值分解在任意矩阵上的推广。另一种是利用非线性的 Levenberg-Marquardt 优化直接最小化拟合误差，它不会显著地增加运算时间，却增加了算法的鲁棒性且更不依赖于初始的姿态推测[19]。在本研究中即利用非线性的迭代最近点算法。

算法通过不断地迭代来找到最优变换。有多种条件可以终止迭代过程：其一是达到一定的迭代次数；其二是两次迭代的变换矩阵之差小于一定值；其三是对应点对的欧式平方误差之和小于一定值。

5.2 实现方法

5.2.1 程序的使用规范

本程序可执行文件名为 *register.exe*，有多个运行参数：输入文件名，必要参数，由“*.pcd”指定，指定需要配准的点云文件，必须有两个点云，前者为目标点云（不进行变换），后者为来源点云（进行变换）；向下采样参数，可选参数，由“-v Size”指定，指定配准前向下采样的体素大小，默认为 0.05；法线估计参数，可选参数，由“-n K”指定，指定法线估计时使用的近邻点个数，默认为 30；配准参数，可选参数，由“-r Dist, Iter”指定，指定配准过程中最大对应点距离和迭代次数，默认为 0.1 和 30；输出文件名，可选参数，由“-o *”指定，决定输出的点云文件名，默认为“reg_cloud”。完整的运行命令如下：

```
register.exe *.pcd *.pcd
[-v Size] [-n K] [-r Dist,Iter] [-o *]
```

程序开启后，会出现两个窗口。命令行窗口显示程序的基本信息，如图 5.2.1。首先显示了输入的来源点云和目标点云的文件名和点数，接着显示了与配准相关的运行参数。之后进入配准阶段，可以实时显示配准进度，完成后会显示配准结果的适合程度。最后会显示将点云存入了 PCD 文件中。屏幕左上方的窗口显示点云的可视化结果，如图 5.2.2。在程序开启时，会在窗口左半边显示输入的点云，其中目标点云为红色、来源点云为绿色，可以通过拖拽操作改变视角。按下键盘的“Q”键将会开始配准过程，窗口右半边会实时显示当前的配准结果。配准结束后，窗口左右侧会同时显示输入点云和配准点云，供用户查看配准结果，再次按“Q”键退出程序。

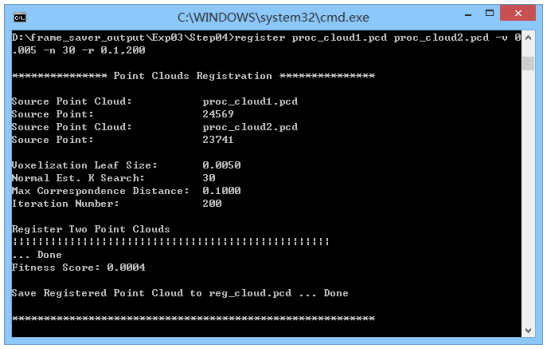


图 5.2.1. 命令行窗口

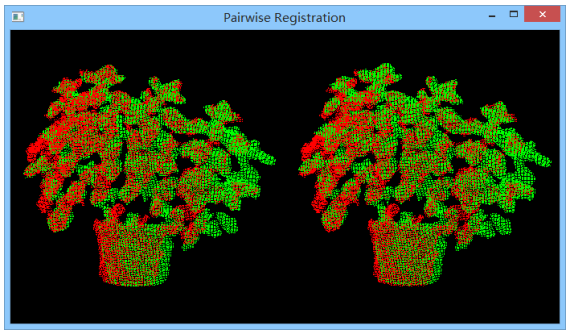


图 5.2.2. 点云可视化窗口

5.2.2 程序的具体实现

本程序源文件名为 *register.cpp*。

程序需要额外定义一个类来表达配准过程中使用的点（X，Y，Z，曲率），这个类继承于 PointRepresentation 类，但要将维度字段 nr_dimensions_设为 4，并重写虚函数 copyToFloatArray。

程序首先利用 Console 模组分析命令行执行参数，把相关参数存入变量，如果必要参数缺失则报错退出。利用 IO 模组读入 PCD 点云文件。初始化 Visualization 模组的 PCLVisualizer 点云可视化对象，并在左侧显示原始点云，颜色利用 PointCloudColorHandlerCustom 对象进行控制。此后开始处理点云，首先利用 Filters 模组的 VoxelGrid 对象进行体素网格向下采样，再利用 Features 模组的 NormalEstimation 对象进行表面法线估计来计算各点曲率，设置相关参数。由于是无序点云，近邻搜索时选用 Search 模组的 KdTree 对象来完成。在配准前，新建一个自定义点表达的对象，并设定四个维度权值均为 1.0。初始化 Registration 模组的 IterativeClosestPointNonLinear 对象进行非线性迭代最近点配准，设置变换矩阵之差阈值为 10^{-8}，设置点表达为上述自定义点表达的对象，并设置最大对应点距离。为了能够实时显示配准结果，设置最大迭代次数为 1，并自行定义一个循环，在循环中进行配准并在可视化窗口右侧显示结果。如果变换矩阵之差小于阈值，则缩小最大对应点距离。最后将配准的两个点云合成为一个，并写入文件，再在可视化窗口右侧显示最终结果。整个流程见图 5.2.3。

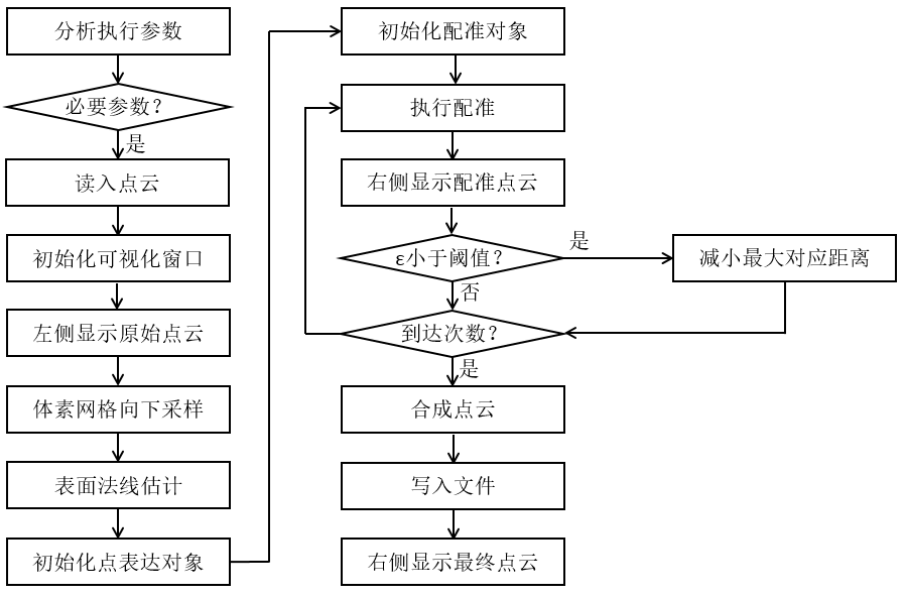


图 5.2.3. 点云配准程序流程图

5.2.3 配准策略

在常见的递增成对配准中，由于配准误差的存在，随着配准的进行误差逐渐积累，往往会导致呈环状的数个点云中第一个点云与最后一个点云之间存在较大的偏移，如图 5.2.4 所示。这种问题会极大地影响点云测量的准确性，因此本研究中采取了一定的配准策略来防止这类问题的发生。

图 5.2.4. 递增成对配准中的首尾偏移

该配准策略的整体思路在于分层次进行逐对的点云配准。在第一层次，输入点云为经过预处理的 16 个单角度点云，命名为点云 1、点云 2、……、点云 16。将点云 1 与点云 2、点云 4 与点云 3、……、点云 13 与点云 14、点云 16 与点云 15 配准，得到点云 1-2、点云 3-4、……、点云 13-14、点云 15-16。之所以每对点云配准时都调换一次方向，是为了使配准结果的点云基本保持在原位置附近。在第二层次，同理，将点云 1-2 与点云 3-4、点云 7-8 与点云 5-6、……、点云 15-16 与点云 13-14 配准，得到点云 1-4、点云 5-8、……、点云 13-16。在第三层次，将点云 1-4 与点云 5-8、点云 13-16 与点云 9-12 配准，得到点云 1-8、点云 9-16。在第四层次，将点云 1-8 与点云 9-16 配准即得到最终的配准点云。整个配准策略如图 5.2.5 所示。

这种配准策略虽然增加了成对配准的次数，但却使配准误差的累积次数从 N-1 次降低为 $\log_2 N$ 次，其中 N 为单角度点云的个数。因此，利用这种配准策略进行环状的多个单角度点云的配准，得到的配准结果远远优于递增成对配准，也防止了点云首尾偏移过大的问题。

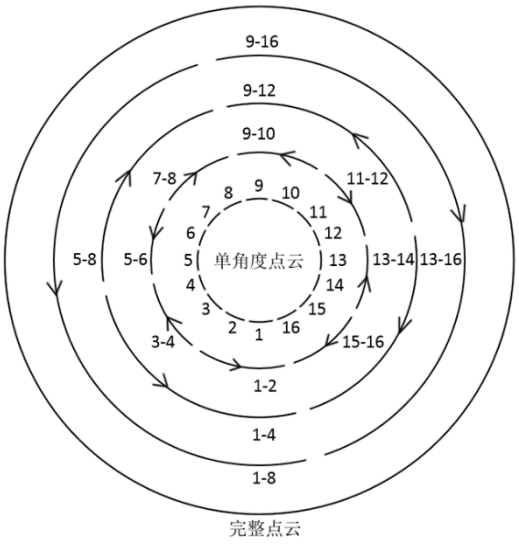


图 5.2.5. 点云配准策略

5.3 结果分析与讨论

5.3.1 实验结果

对三个植株的 16 个经过预处理的单角度点云按照上述配准策略进行了配准，并得到了三个完整的植株点云。点云两两配准的消耗时间相对较长，取决于两个点云中的点数以及相关的配准参数。根据观察，配准效果良好，并且每一次配准的适合程度都在 0.001 以下。

对于植株 A 点云的配准，设置向下采样体素大小为 0.01，法线估计近邻点数为 30，最大对应点距离为 0.05，迭代次数为 300。第一层次中点云两两配准耗时约 14 秒；第二层次 23 秒；第三层次 37 秒；第四层次 56 秒；总计耗时约 5.5 分钟。配准结果如图 5.3.1。

图 5.3.1. 植株 A 配准的完整点云

图 5.3.2. 植株 B 配准的完整点云

对于植株 B 点云的配准，设置向下采样体素大小为 0.01，法线估计近邻点数为 30，最大对应点距离为 0.05，迭代次数为 300。第一层次中点云两两配准耗时约 16 秒；第二层次 24 秒；第三层次 48 秒；第四层次 60 秒；总计耗时约 6.3 分钟。配准结果如图 5.3.2。

对于植株 C 点云的配准，设置向下采样体素大小为 0.005，法线估计近邻点数为 30，最大对应点距离为 0.01，迭代次数为 300。第一层次中点云两两配准耗时约 14 秒；第二层次 19 秒；第三层次 36 秒；第四层次 45 秒；总计耗时约 5.1 分钟。配准结果如图 5.3.3。

图 5.3.3. 植株 C 配准的完整点云

5.3.2 结果分析

观察配准结果，可知得到的植株完整点云基本满足了要求，并且不存在相邻单角度点云之间的明显偏移。但仍有一些值得改进的地方，如相邻的单角度点云之间仍存在一定的肉眼可见的偏移、合适的配准参数需要多次试验、配准过程容易陷入局部极值等。

此外，配准操作的时间过长也是一个需要解决的问题。需要注意的是，本研究中所采用的配准参数并非最优化的配准参数，而仅仅是采用了一种能够获得良好配准效果的参数，并没有考虑到配准时间的问题。通过进一步的实验，应当可以找到能够大大缩短配准时间、而不太影响配准效果的最优化配准参数。

5.3.3 工作展望

可能的改进方案包括以下几个方面：

关于关键点的提取，本研究采用的方法相当于仍然将整个点云的全部信息作为输入，其中可能包含一些噪声等等影响配准结果，可以考虑使用其他的关键点提取算法提取点云关键点用于配准。例如，考虑深度图像的边界的归一化对齐径向特征（NARF）关键点提取[20]、需要点的强度信息的尺度不变性特征转换（SIFT）关键点提取[21]等。

关于特征描述子，还有许多除表面曲率外的选择，可能能够获得更好的效果。包括 NARF 描述子和需要法线信息的快速点特征直方图（FPFH）描述子等。

关于对应点估计，可以考虑使用双向估计取交集的方法减少错误的对应点数。

在这种类型的点云配准中，最终误差很大程度上来自于配准误差的逐步积累。因此，一些基于图理论的全局配准算法可以有效的解决这一问题，它们可以在成对配准的基础上进行适度的松弛，使得全局误差最小。这些算法包括显式环封闭探索（ELCH）算法[22]、全局一致性扫描匹配（LUM）算法[23]等。

第 6 章 植物三维点云的后处理

6.1 基本原理

6.1.1 主要工作思路

这一部分的主要工作在于对配准得到的植株完整点云进行后处理，以提升点云测量的准确度。具体工作包括：对点云进行半径离群点去除，剔除影响测量结果的离群点；再利用体素网格向下采样减少点数并使点云密度更为均匀；最后进行移动最小二乘平滑，减小随机噪声对点云测量的影响。以植株 A 的点云为例，这一部分的流程如图 6.1.1。

图 6.1.1. 三维点云后处理流程

6.1.2 半径离群点去除

半径离群点去除以点的近邻点数为依据进行离群点剔除。算法首先遍历整个点云，对于每一个点，检索在一定半径内的近邻点数。如果这个点的近邻点数太少，就被认为是离群点，将会被剔除。

在本研究中，由于前面讨论过的一些原因，点云中仍存在一些转台表面的点，应予以去除，否则会极大地干扰点云测量的结果。由于它们分布较为分散，与主要的植株点云距离较远，比较适合使用半径离群点去除进行剔除。

6.1.3 移动最小二乘平滑

移动最小二乘平滑的方法是基于点集隐式定义了表面这一思想而产生的。算法对整个点云中的所有点进行了两个步骤的处理：第一步，利用一个点本身和它周围一定数量的点，进行最小二乘表面拟合。这个表面既可以是点集所隐式定义的表面的切平面，如同法线估计时一样，也可以是由多项式所定义的高阶曲面。但多项式阶数过高也会导致平滑效果不足和计算缓慢等问题。第二步，将这个点

投影到拟合的最小二乘表面上。这样，就完成了移动最小二乘平滑的过程。通过这种表面拟合的思想，还可以进行点云的向上采样，从而形成更为密集更为均匀的点云[24]。

在本研究中，使用移动最小二乘进行平滑的原因在于：一方面，Kinect 获取的深度数据具有随机噪声，导致同一个平面上的点也可能会因噪声的存在而变得高低起伏；另一方面，由于完整的植株点云是依靠点云配准获得的，不可避免的存在配准误差，会出现一些同一个表面却有两层点的情况。移动最小二乘平滑可以有效地解决这些问题，有利于提升点云测量准确性。

6.2 实现方法

6.2.1 程序的使用规范

本程序可执行文件名为 *postproc.exe*，有多个运行参数：输入文件名，必要参数，由“*.pcd”指定，指定需要后处理的点云文件；离群点去除参数，可选参数，由“-r R_{out}, N_{out}”指定，指定半径离群点去除的搜索半径和最小近邻点数，若不指定该参数则不进行操作；向下采样参数，可选参数，由“-v Size”指定，指定体素网格向下采样的体素大小，若不指定该参数则不进行操作；平滑参数，可选参数，由“-m R, Order”指定，指定移动最小二乘平滑的平滑半径和多项式阶数，若不指定该参数则不进行操作；输出文件名，可选参数，由“-o *”指定，决定输出的点云文件名，默认为“final_cloud”。完整的运行命令如下：

```
postproc.exe *.pcd
[-r Rout,Nout] [-v Size] [-m R,Order] [-o *]
```

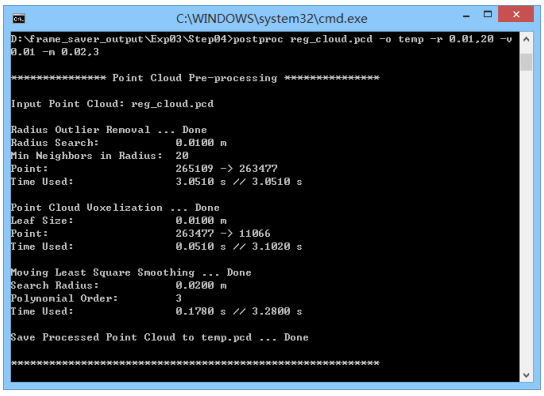


图 6.2.1. 命令行窗口

程序开启后，会出现一个命令行窗口，如图 6.2.1。首先显示了输入点云文件名，接着显示进行的操作及相关参数。由于一些操作（如平滑）耗时较多，增加了显示所耗时间的功能。最后显示将点云存入 PCD 文件中。

6.2.2 程序的具体实现

本程序源文件名为 *postproc.cpp*。

程序首先利用 Console 模组分析命令行执行参数，把相关参数存入变量，如果必要参数缺失则报错退出。利用 IO 模组读入输入 PCD 点云文件。如果半径离群点去除被启用，则初始化 Filters 模组的 RadiusOutlierRemoval 对象并设定相关参数，对点云进行滤波。如果体素网格向下采样被启用，则初始化 Filters 模组的 VoxelGrid 对象并设定相关参数，对点云点数进行削减。如果移动最小二乘平滑被启用，则初始化 Surface 模组的 MovingLeastSquare 对象并设定相关参数，其中搜索方式选用 Search 模组的 KdTree 对象，向上采样方式选用“NONE”。最后将处理后的点云存入文件。此外，在程序的开端还要初始化 Common 模组的 StopWatch 对象，并在每个操作进行前将其重置，在操作进行后读取所耗时间。整个流程见图 6.2.2。

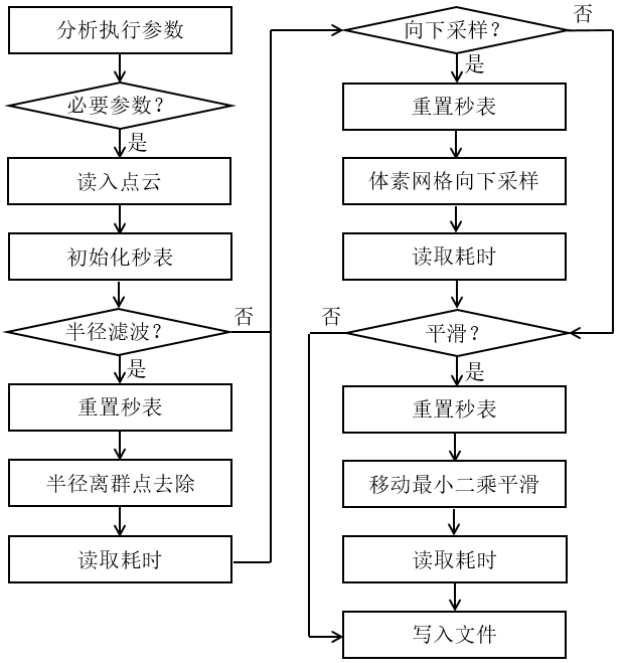


图 4.2.2. 点云后处理程序流程图

6.3 结果分析与讨论

6.3.1 实验结果

对三个植株的完整点云进行了后处理。点云后处理需要消耗一定的时间，这主要决定于所使用的后处理参数。

对于植株 A 点云的后处理，设置半径离群点去除的搜索半径为 0.01，近邻点阈值为 30，向下采样的体素大小为 0.002，移动最小二乘平滑的平滑半径为 0.03，多项式阶数为 3。总处理时间 123 秒，其中平滑操作消耗时间 120 秒。后处理结果如图 6.3.1。

对于植株 B 点云的后处理，设置半径离群点去除的搜索半径为 0.02，近邻点阈值为 40，向下采样的体素大小为 0.002，移动最小二乘平滑的平滑半径为 0.03，多项式阶数为 3。总处理时间 128 秒，其中平滑操作消耗时间 119 秒。后处理结果如图 6.3.1。

图 6.3.1. 植株 A 经后处理的点云

图 6.3.2. 植株 B 经后处理的点云

图 6.3.3. 植株 C 经后处理的点云

对于植株 C 点云的后处理，设置半径离群点去除的搜索半径为 0.01，近邻点阈值为 20，向下采样的体素大小为 0.002，移动最小二乘平滑的平滑半径为 0.02，多项式阶数为 3。总处理时间 48 秒，其中平滑操作消耗时间 45 秒。后处理结果如图 6.3.3。

6.3.2 结果分析

可以看出，经过后处理的点云质量比配准后的点云好很多，但仍然存在着一些问题：

半径离群点去除的搜索半径和阈值需要通过多次试验来确定。如果设置的不合理，很有可能无法去除全部离群点，或是将大量有效点也一并去除。

移动最小二乘平滑的平滑半径的选择存在困境。如果平滑半径选择较小，就会受到数据点噪声的影响，导致平滑效果不佳，平滑后的点云依然凹凸不平；如果平滑半径过大，就会改变原始点云的形状。观察本研究的结果可以发现，平滑后的点云与原点云相比有着不同程度的变形，比较明显的就是有所收缩。这显然是会在一定程度上影响点云测量结果的。

6.3.3 工作展望

可能的改进方案包括以下几个方面：

在配准前的预处理时彻底解决离群点问题，而在后处理过程中进行更为保守的离群点剔除，甚至不进行离群点剔除。事实上，在经过配准得到的植株完整点云中，有用的数据点是占绝大部分的，对其进行离群点剔除就不可避免地删去了一些有用的点。因此，在配准前通过合适的算法彻底剔除全部离群点，可以避免两次离群点剔除带来的有用点损失。

通过选择合适的移动最小二乘平滑参数，或是利用其他平滑算法，在尽量不改变点云形状的同时平滑点云。因为本研究的目的在于点云的测量，形状变化对测量影响非常大，在点云的处理过程中是一定要避免的。移动最小二乘平滑还具有一些变种，对算法的鲁棒性等进行了一些优化，可以利用它们得到更好的平滑效果。

第 7 章 植物三维点云的测量

7.1 基本原理

7.1.1 主要工作思路

这一部分的主要工作在于对植株点云进行测量。由于本研究关注的是整个植株的外形参数，而非植株的个别部位的属性，因此仅从宏观上进行测量，不再做进一步的点云分割。对植株的测量主要从基部、冠层和整体三个方面入手：

基部测量依赖于距植株底部 1cm 处的横截面，测量参数包括最大长度、宽度、平均直径、平均费雷特直径、周长和面积。基部参数的测量有利于确定植株的占地情况，在植株密度不太大的时候此数据可用于规划植株布置。

冠层测量依赖于整个植株的垂直投影面，测量参数同基部测量。冠层参数的测量代表了植物的遮荫情况，一方面它代表了植株的生长状况，一方面也可以用于合理安排植株种植密度。

整体测量则包括株高和体积。它直接反映的植株的长势情况。

7.1.2 单调链凸包算法

对于基部和冠层的测量，都是基于对一个平面上的点集所隐式定义的形状的测量。由于原始数据是离散的无序点集，必须找出哪些点处于形状的边界，以及它们是通过何种顺序进行连接的，才能进行各种计算。因此，必须使用凸包算法求出点集的凸包。

一个点集的凸包是指能够满足点集内所有点都在其内或边界上这一条件的最小的凸多边形，它可以看作由点集中处于边界上的点按一定顺序连接而成。凸包算法是几何算法研究中的一个热门问题，在本研究中选择实现了单调链凸包算法，也称 Andrew 算法。它的时间复杂度为 $O(n \log n)$。

算法首先对点集按照 X 坐标升序进行排序，如果相等则按照 Y 坐标升序，之后再分别构造下凸包和上凸包两条单调链：从最左侧的点开始，沿逆时针顺序构造下凸包，直到最右侧的点；之后再沿逆时针顺序构造上凸包。具体的构造方法是：正序遍历排序后的点集，对于每一个点，在链中含有至少两个点且当前点

与链中的最后两个点不构成逆时针转折时，删除链中的最后一个点，循环执行直到不符合条件，再把当前点加入链中。这样就得到了下凸包。上凸包同理，只是遍历的顺序为逆序[25]。如图 7.1.1 所示。

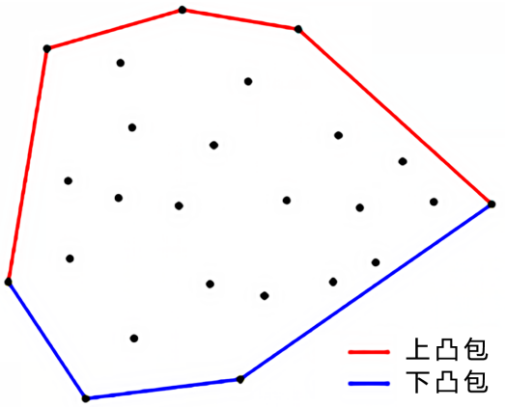


图 7.1.1. 单调链凸包算法

7.1.3 多边形周长计算

由于生成的凸包是具有顺序的点列表，周长计算的问题就简化成了相邻点间距离求和的问题。在二维欧氏空间中，两点间距离的计算公式如下：

$$L_{AB} = \sqrt{(X_A - X_B)^2 + (Y_A - Y_B)^2} \tag{7.1.1}$$

7.1.4 多边形直径计算

在本研究中用到的多边形直径包括最大长度 / 宽度、平均直径和平均费雷特直径三种，它们都可以认为是费雷特直径的特殊情况或由其计算产生。

费雷特直径是一个物体沿着一个特定方向的量度。在广义上，它可以被定义为两个平行的平面在垂直于该方向上夹住物体的距离，因如同利用卡尺测量物体的大小，也被称为卡尺距离。在二维中，费雷特直径定义中的物体退化为平面图形、平行平面退化为平行线。

最大长度是一个图形的最大的费雷特直径，而宽度则是在垂直于最大长度的方向上的费雷特直径。平均直径是一个图形的最大费雷特直径与最小费雷特直径的平均值。平均费雷特直径是在所有方向上的费雷特直径的平均值。

这三种直径可以在一个流程中同时计算出来。具体方法是：将图形从 0° 到 90° 每隔一定的步进角进行旋转，在每次旋转后，测量 X 方向上和 Y 方向上的跨度，作为两个相互垂直的费雷特直径。则最大长度为其中的最大值，宽度为与

最大值在同一个转角下得到的另一个值，平均直径为最大值与最小值的平均值，平均费雷特直径为所有值的平均值。

关于图形的旋转，在此处为多边形的旋转，事实上就是构成多边形的每个顶点的坐标系旋转。这与前述的点云几何变换类似，只是在二维空间下进行。公式如下：

$$x' = x\cos\theta - y\sin\theta \tag{7.1.2}$$

$$y' = x\sin\theta + y\cos\theta \tag{7.1.3}$$

其中，x、y 为原始坐标；x'、y'为旋转后的坐标；θ为旋转角度，逆时针为正。

此外，根据柯西定理，对于二维的凸多边形，平均费雷特直径等于多边形的周长除以 π。此规律对于凹多边形不使用。

7.1.5 多边形面积计算

多边形面积的计算是基于将多边形分割为多个三角形的思路得到的。任意多边形的面积等于多边形上相邻的每两个顶点与坐标原点构成的三角形的有向面积之和，其中，有向面积是指三角形的三个顶点若按逆时针顺序排列则面积为正，反之为负。如图 7.1.2，多边形面积可看作由 3 个正向三角形 OBC、OCD、ODE 和 2 个负向三角形 OAB、OEA 面积相加而成。

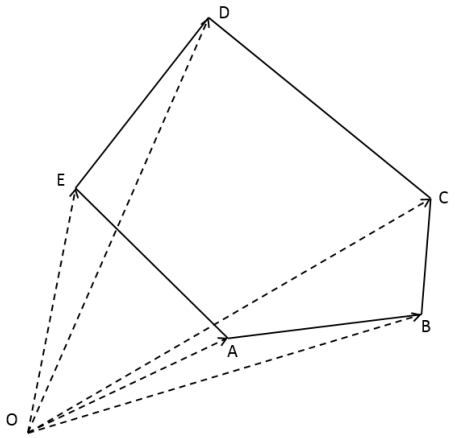


图 7.1.2. 任意多边形面积计算示意

因此，任意多边形的面积计算公式如下：

$$S = \frac{1}{2}\left\{\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} + \cdots + \begin{vmatrix} x_n & y_n \\ x_1 & y_1 \end{vmatrix}\right\}$$
$$= \frac{1}{2}\sum_{k=1}^{n}(x_k y_{k+1} - x_{k+1} y_k) \quad (7.1.4)$$

7.1.6 体积估算

在本研究中，对于植株点云的体积估算采用微元法的思想。即以一个非常小的距离为间距将植株点云的总高度分成许多小份，在每个高度处求横截面的点集凸包面积，再用面积乘以间距累加起来就得到了体积的近似值。

7.2 实现方法

7.2.1 程序的使用规范

本程序可执行文件名为 *measure.exe*，有一个运行参数：输入文件名，必要参数，由“*.pcd”指定，指定需要测量的植株点云文件。完整的运行命令如下：

```
measure.exe *.pcd
```

程序开启后，会出现一个命令行窗口，如图 7.2.1。首先显示了输入点云文件名及点数。之后进行基部参数计算，计算完成后会显示相关结果及所耗时间。同样地，之后会进行冠层参数计算和整体参数计算。

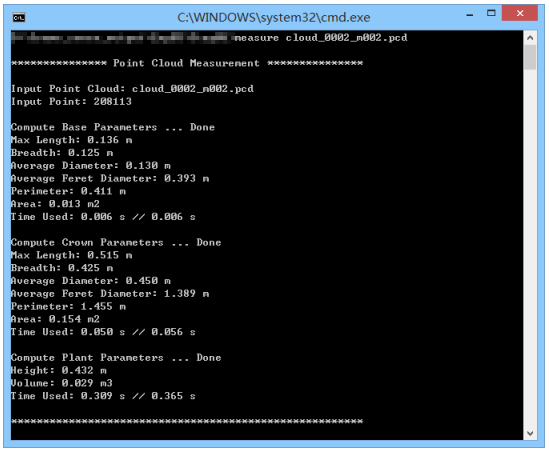


图 7.2.1. 命令行窗口

7.2.2 程序的具体实现

本程序源文件名为 *measure.cpp*。

程序首先利用 Console 模组分析命令行执行参数，把相关参数存入变量，如果必要参数缺失则报错退出。利用 IO 模组读入输入 PCD 点云文件。在测量基部参数时，首先使用 Filters 模组的 PassThrough 对象对点云截取 Z 坐标的 0.005~0.015 范围，作为基部的点集。再使用 copyPointCloud 函数将 PointXYZRGBA 类型的点云复制到 PointXY 类型的点云中，舍弃了 Z 坐标和颜色信息，即是投影到了平面上。对复制后的点云求凸包，并利用求得的凸包计算各种参数。在测量冠层参数时，不需要截取部分点，而是将全部点直接投影的平面上，再进行后续计算。在测量整体参数时，首先遍历整个点云，找到 Z 坐标最大的点，其 Z 坐标即为植株的高度。再以 0.01 为步长遍历植株的高度，在每个高度下，截取 Z 坐标±0.005 范围的点集，投影至平面，求凸包及其面积，将面积乘以 0.01 的步长累加起来，遍历结束后得到体积估计值。此外，在程序的开端还要初始化 Common 模组的 StopWatch 对象，并在每个操作进行前将其重置，在操作进行后读取所耗时间。整个流程见图 7.2.2。

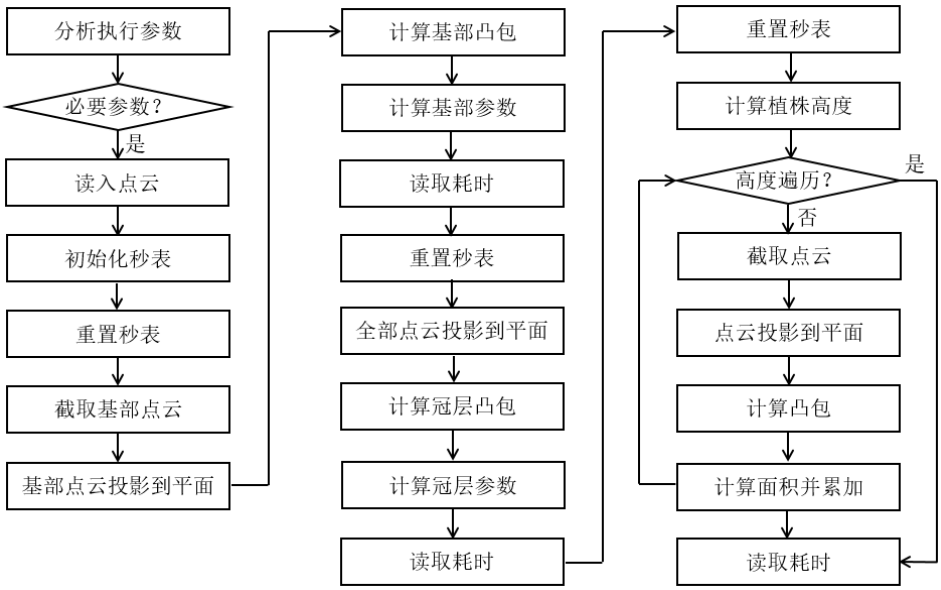


图 7.2.2. 点云测量程序流程图

7.3 结果分析与讨论

7.3.1 实验结果

对三个植株的经过后处理的最终点云进行测量。测量程序的执行时间均在 3 秒以内，因输入点云的点数而异。结果如下表所示：

表 7.3.1 植株 A、B、C 结构性参数的无损测量结果

类型	参数	植株 A	植株 B	植株 C
基部	最大长度 / m	0.260	0.252	0.136
	宽度 / m	0.242	0.242	0.125
	平均直径 / m	0.250	0.242	0.130
	平均费雷特直径 / m	0.251	0.242	0.131
	周长 / m	0.789	0.761	0.411
	面积 / m^2	0.049	0.045	0.013
冠层	最大长度 / m	0.773	1.135	0.515
	宽度 / m	0.763	1.073	0.425
	平均直径 / m	0.751	1.060	0.450
	平均费雷特直径 / m	0.754	1.077	0.463
	周长 / m	2.368	3.383	1.455
	面积 / m^2	0.431	0.844	0.154
整体	高度 / m	1.040	1.446	0.432
	体积 / m^3	0.160	0.528	0.029

对于三个植株的基部周长、冠层最大长度、植株高度三个指标进行物理测量，用于验证无损测量的结果。其他指标由于难以进行物理测量，本研究中不予涉及。结果如下表所示：

表 7.3.2 植株 A、B、C 部分结构性参数的无损测量与物理测量结果对比

植株	参数	无损测量结果	物理测量结果	相对误差
植株 A	基部周长 / m	0.789	0.76	3.82%
	冠层最大长度 / m	0.773	0.78	-0.90%
	植株高度 / m	1.040	1.03	4.00%
植株 B	基部周长 / m	0.761	0.72	5.69%
	冠层最大长度 / m	1.135	1.09	4.13%
	植株高度 / m	1.446	1.41	2.55%
植株 C	基部周长 / m	0.411	0.40	2.50%
	冠层最大长度 / m	0.515	0.50	3.00%
	植株高度 / m	0.432	0.43	0.47%

7.3.2 结果分析

上述的结果对比显示，无损测量可以满足基本的测量需求，但与物理测量的结果仍有一定的偏差。由于植物这一测量对象的特殊性，准确的物理测量也相当困难，因此上述结果中的物理测量数据应当也存在一定的测量误差，尤其是冠层

最大长度这一参数。但需要承认的是，无损测量结果的准确度仍有很大的改进空间，下面将进行分析。

造成无损测量误差的因素有很多，无损测量流程中的任何一个环节都有可能带来或大或小的误差，许多已经在前面的章节中进行了分析，这里只分析测量程序可能带来的测量误差：

在截取某一高度处横截面的点云时，所使用的 Z 坐标范围较大。在通过前述过程获取的点云中，点的排列不是严格整齐的，即便是大致位于同一个高度处的点，其 Z 坐标也会存在上下浮动。由于不知道点云的初始密度，只能以一个相对较大的 Z 坐标范围进行截取，才能保证能够截取到足够多的、能够完整定义截面形状的点集。在本研究中，采用±0.005 作为这一范围。但这也同时带了一个问题：在变化较为剧烈的部分，Z 坐标相距 0.01 的两个横截面可能存在很大的不同，如果以这样的范围进行截取，则相当于求取了它们并集，并不能准确表达某一高度处横截面的实际情况。此外，在计算植株体积时，同样采用 0.01 作为步长，这对于微元来说稍稍显大，会导致体积计算产生较大误差。

图 7.3.1 植株 A 高度为 0.7 附近的平面点集

凸包并不能准确代表平面点集所隐式定义的形状。由于植物冠层是由大量叶片构成的，其中具有很多空隙，因而截面的形状更有可能是一个凹多边形，而非凸多边形。例如图 7.3.1，这是植株 A 高度为 0.7 附近的平面点集。因此，利用植株截面点集的凸包来近似截面的形状，会导致周长的计算偏小，而面积的计算偏大。对于植株的基部来说，截面形状多为近似于圆形的凸多边形，这一问题则不是产生误差的主要原因。如在本研究中，植株的基部均为圆台状的花盆。

7.3.3 工作展望

可能的改进方案包括以下几个方面：

在测量前先采用一定的算法计算点云的大致密度，再根据点云密度确定截取横截面点云的 Z 坐标范围。计算点云密度的方法可以是对于点云中的每一点进行近邻搜索，计算该点到数个近邻点的平均距离，再对所有平均距离求平均值，可以作为点云中相邻两点的平均距离。

在近似平面点集所定义的形状时，计算点集的凹包而非凸包。凹包有时被认为是能够包围住所有点的多边形中面积最小的一个，但有时面积最小的多边形边缘不够平滑，并不是最为理想的凹包。这里存在一个取舍的问题，计算凹包的算法也因此比凸包算法复杂得多。但仍然有一些算法可以较为可靠地计算它，可以考虑应用在未来的研究中[26]。

此外，还可以考虑先对点云进行三角化表面重建，再进行相关测量。这将会使某高度处横截面的形状更为确定、体积测量更为准确。

由于获取的植株点云比较完整，除了本研究中所涉及的几个之外，还有更多的参数可以从中提取得到。例如，利用点云分割算法将植株的叶片分离出来，测量叶片的相关参数如叶面积、叶倾角等等。

第 8 章 结论

8.1 研究总结

植物结构性参数的无损测量在生长监测、科学建模、品质控制等方面意义重大。本研究利用微软的 Kinect 传感器作为彩色和三维位置数据的获取源，结合多种点云处理技术获取了植株的完整三维点云，并在此基础上导出了植株的多种结构性参数，达到了无损测量的目标。整个无损测量流程对硬件和软件环境要求较低，步骤简便，易于实现，测量结果误差较小，且能够完成一些物理测量难以进行的指标的测量，具有一定的实践意义。

本研究主要完成了以下几方面的研究工作：

1）植物三维点云的获取。按照一定的要求搭建测量装置；利用 Kinect 传感器获取彩色图像和三维点云数据，实时显示并根据需要随时保存到文件中；通过固定的流程获取植株的多角度点云。

2）植物三维点云的校正。布置校正参考标记，并获取其三维点云用于校正；指定彩色图像中的参考点坐标，并转换为三维位置；计算法向量和校正用的转换矩阵；将变换矩阵数据存入二进制文件。

3）植物三维点云的预处理。对原始场景点云利用变换矩阵数据进行校正，并变换到世界坐标系；利用直通滤波限定通过范围去除背景点并进行离群点剔除；进行姿态估计为点云配准做准备。

4）植物三维点云的配准。对每两个单角度点云进行迭代最近点配准，并通过一些方法提升配准速度和准确性；通过一定的配准策略减小累积配准误差的影响。

5）植物三维点云的后处理。对配准后的完整点云滤除个别离群点；向下采样保证后续运算速度和点云均匀性；平滑处理提升测量准确性。

6）植物三维点云的测量。对植株基部、冠层和整体三个方面进行包括高度、直径、周长、面积、体积等指标在内的多种结构性参数的测量。

8.2 后续工作

虽然本研究已经基本达成本研究在总体上来说还存在一些方面可进行后续研究：

1）整个植物无损测量流程的自动化。当前的研究将整个无损测量流程分割为相互关联的六个步骤，对每个步骤编写了独立的程序进行实现。而在生产实践中，这样的做法效率低下。理想的情况是，通过自动化的拍摄装置快速获取植株多角度点云，自适应地设定各种参数进行点云的处理，并将植株点云的测量结果以可视化的形式反馈给用户，从而提升植物无损测量软件的易用性。

2）实现植株的实时无损测量。在当前的研究中，点云处理的某些步骤耗时较长，不能满足实时测量的要求，例如点云配准和点云平滑。可以通过使用更具有效率的点云处理算法，或合理地设置各类算法的参数来提高点云的处理速度；抑或是深入研究每个处理过程对于测量准确性的影响，从而剔除一些对准确性影响不大的操作来换取处理速度。

参考文献

[1] 余涛. Kinect 应用开发实战：用最自然的方式与机器对话 [M]. 北京: 机械工业出版社, 2012.

[2] Wallenberg M, Felsberg M, Forssén P E, et al. Leaf Segmentation Using the Kinect [C]// Proceedings of SSBA 2011 Symposium on Image Analysis. 2011.

[3] 江晓庆, 肖德琴, 张波, 等. 基于 Kinect 的农作物长势深度图像实时获取算法 [J]. 广东农业科学, 2012, 39(23): 195-199.

[4] S. Yamamoto, S. Hayashi, S. Saito, et al. Measurement of Growth Information of a Strawberry Plant Using a Natural Interaction Device [C]// ASABE Annual International Meeting. ASABE, 2012.

[5] Chen Y, Zhang W, Yan K, et al. Extracting Corn Geometric Structural Parameters Using Kinect [C]// IEEE International Geoscience and Remote Sensing Symposium (IGARSS). IEEE, 2012: 6673-6676.

[6] Azzari G, Goulden M L, Rusu R B. Rapid Characterization of Vegetation Structure with a Microsoft Kinect Sensor [J]. Sensors, 2013, 13(2): 2384-2398.

[7] Lippman S B, Lajoie J, Moo B E. C++ Primer 英文版（第 5 版） [M]. 北京: 电子工业出版社, 2013.

[8] 朱德海. 点云库 PCL 学习教程 [M]. 北京: 北京航空航天大学出版社, 2012.

[9] Rusu R B. Adding Your Own Custom PointT Point Type [EB/OL]. http://www.pointclouds.org/documentation/tutorials/adding_custom_ptype.php, 2014-05-21.

[10] Rusu R B. Getting Started / Basic Structures [EB/OL]. http://www.pointclouds.org/documentation/tutorials/basic_structures.php, 2014-05-21.

[11] Rusu R B. The PCD (Point Cloud Data) File Format [EB/OL]. http://www.pointclouds.org/documentation/tutorials/pcd_file_format.php, 2014-05-21.

[12] Shirley P, Ashikhmin M, Marschner S. Fundamentals of Computer Graphics Third Edition [M]. Boca Raton: CRC Press, 2009.

[13] Rusu R B, Marton Z C, Blodow N, et al. Towards 3D Point Cloud Based Object Maps for Household Environments [J]. Robotics and Autonomous Systems, 2008, 56(11): 927-941.

[14] Holz D, Rusu R B, Sprickerhof J. The PCL Registration API [EB/OL]. http://www.pointclouds.org/documentation/tutorials/registration_api.php, 2014-05-23.

[15] Rusu R B. Downsampling a PointCloud Using a VoxelGrid Filter [EB/OL]. http://www.pointclouds.org/documentation/tutorials/voxel_grid.php, 2014-05-23.

[16] Rusu R B. Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments [D]. Muenchen: Computer Science department, Technische Universitaet Muenchen, 2009.

[17] Besl P J, McKay N D. A Method for Registration of 3-D Shapes [J]. IEEE Transcations on Pattern Analysis and Machine Intelligence, 1992, 14(2): 239-256.

[18] Fischler M A, Bolles R C. Random Sample Consensus: a Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography [J]. Communications of the ACM, 1981, 24(6): 381-395.

[19] Fitzgibbon A W. Robust Registration of 2D and 3D Point Sets [J]. Image and Vision Computing, 2003, 21(13): 1145-1153.

[20] Steder B, Rusu R B, Konolige K, et al. Point Feature Extraction on 3D Range Scans Taking into Account Object Boundaries [C]// IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2011: 2601-2608.

[21] Lowe D G. Distinctive Image Features from Scale-Invariant Keypoints [J]. International Journal of Computer Vision, 2004, 60(2): 91-110.

[22] Sprickerhof J, Nüchter A, Lingemann K, et al. An Explicit Loop Closing Technique for 6D SLAM [C]// ECMR. 2009: 229-234.

[23] Lu F, Milios E. Globally Consistent Range Scan Alignment for Environment Mapping [J]. Autonomous Robots, 1997, 4(4): 333-349.

[24] Alexa M, Behr J, Cohen-Or D, et al. Computing and Rendering Point Set Surfaces [J]. IEEE Transactions on Visualization and Computer Graphics, 2003, 9(1): 3-15.

[25] Andrew A M. Another Efficient Algorithm for Convex Hulls in Two Dimensions [J]. Information Processing Letters, 1979, 9(5): 216-219.

[26] Moreira A, Santos M Y. Concave Hull: A K-Nearest Neighbours Approach for the Computation of the Region Occupied by a Set of Points [C]// Proceedings of the International Conference on Computer Graphics Theory and Applications (GRAPP). INSTICC Press, 2007: 61-68.

致谢

本论文是在导师蒋焕煜教授的关怀和指导下完成的。从论文的选题、开题，到项目的准备、实施，再到结果的分析和论文的撰写，导师的关心与帮助都起到了不可磨灭的作用，在此谨向导师致以诚挚的感谢和崇高的敬意。导师正直的为人和严谨的治学态度，鼓舞和激励着我在学术道路上不断前行。

本论文的完成离不开吴茜学姐的帮助。她在研究过程给予的适时督促和检查使得论文得以如期完成。在研究陷入僵局时，她也提出了许多建设性的意见帮助我走出困境。此外还为论文的修改提供了大量有用的建议。

感谢浙江大学生物系统工程与食品科学学院生物系统工程系的其他老师在大学四年中所传授的知识和给予的教导，使我的知识面得到了很大的拓展，也锻炼了我多方面的能力。特别要感谢应义斌教授对于职业规划和人生追求方面的关怀与指导，使我明确了出国继续深造的目标。

感谢新加坡科技设计大学的陈陆捷副教授在交流期间给予的帮助和指导。陈老师通过科研训练带我初窥了计算机图形学的奥妙，并在留学申请方面给予了我很多帮助。

感谢浙江大学国际设计研究院的老师们传授给我当代的设计思维和设计方法。特别要感谢张东亮教授为本研究提供了实验场地和部分器材，以及他在第二专业的毕业设计中给予的无私指导。

感谢在大学四年中陪伴我的同学们、朋友们以及同事们，是他们使我的大学生活变得多姿多态。尤其要感谢与我在同一个教室一同学习、一同工作、一同娱乐的周奥博、石磊、熊钊、李腾、魏凡丁、周家惠、叶稚韵、刘子榆、甘茜珺等同学，以及与我在同一个寝室共同生活、容忍我的缺点的室友李琛、张俊敏、马壮同学。

最后，还要感谢一直在精神和经济等方方面面支持我的家人，他们是我在学术的道路上越走越远的最大后盾。

马轲

二〇一四年五月于浙大紫金港

附录 程序源代码

grab.cpp

```
#include <pcl/io/openni_grabber.h>
#include <pcl/visualization/pcl_visualizer.h>
#include <pcl/visualization/image_viewer.h>
#include <pcl/io/pcd_io.h>
#include <pcl/io/png_io.h>

boost::mutex cld_mutex, img_mutex;

pcl::PointCloud<pcl::PointXYZRGBA>::ConstPtr g_cloud;
boost::shared_ptr<openni_wrapper::Image> g_image;

int frames_saved;
bool save_cloud;
bool save_image;

struct EventHelper
{
    // 点云获取回调
    void
        cloud_cb (const pcl::PointCloud<pcl::PointXYZRGBA>::ConstPtr & cloud)
    {
        cld_mutex.lock ();
        g_cloud = cloud;
        cld_mutex.unlock ();
    }

    // 图像获取回调
    void
        image_cb (const boost::shared_ptr<openni_wrapper::Image>& image)
    {
        img_mutex.lock ();
        g_image = image;
        img_mutex.unlock ();
    }
};

// 键盘响应回调
void
    keyboardEventOccurred(const pcl::visualization::KeyboardEvent& event,
        void* nothing)
{
    // 若按下空格键，则保存点云和图像
    if (event.getKeySym() == "space" && event.keyDown())
    {
        cout << "Save Frame " << frames_saved + 1 << "\n";
        frames_saved++;
        save_cloud = true;
        save_image = true;
    }
}

int
    main (int argc, char** argv)
{
```

```
// 点云图像获取对象初始化
pcl::Grabber* interface = new pcl::OpenNIGrabber ();
EventHelper event_helper;
boost::function<void(const pcl::PointCloud<pcl::PointXYZRGBA>::ConstPtr&)>
    cloud_cb = boost::bind (&EventHelper::cloud_cb, &event_helper, _1);
interface->registerCallback (cloud_cb);
boost::function<void (const boost::shared_ptr<openni_wrapper::Image>&)>
    image_cb = boost::bind (&EventHelper::image_cb, &event_helper, _1);
interface->registerCallback (image_cb);

// 点云可视化对象初始化
boost::shared_ptr<pcl::visualization::PCLVisualizer> cld;
cld.reset (new pcl::visualization::PCLVisualizer ("Cloud Viewer"));
cld->registerKeyboardCallback(keyboardEventOccurred);
cld->setCameraPosition (0.0, 0.0, -2.0, 0.0, 0.0, 1.0, 0.0, -1.0, 0.0);

// 图像可视化对象初始化
boost::shared_ptr<pcl::visualization::ImageViewer> img;
img.reset (new pcl::visualization::ImageViewer ("Image Viewer"));
img->registerKeyboardCallback(keyboardEventOccurred);

// 保存文件状态初始化
frames_saved = -1;
save_cloud = false;

printf ("\n*************** Point Cloud Acquisition **************\n");
printf ("\n");

// 获取开始
interface->start ();

// 主循环
bool cld_init = false;
bool img_init = false;
unsigned char* rgb_data = 0;
unsigned rgb_data_size = 0;
while (!cld->wasStopped () && !img->wasStopped ())
{
    // 显示点云
    if (g_cloud && cld_mutex.try_lock ())
    {
        // 初始化
        if (!cld_init)
        {
            cld->getRenderWindow ()->SetSize (g_cloud->width,
                g_cloud->height);
            cld->getRenderWindow ()->SetPosition (g_cloud->width, 0);
            cld_init = true;
        }

        // 显示点云
        cld->removePointCloud ("Cloud");
        cld->addPointCloud (g_cloud, "Cloud");

        // 保存点云
        if (save_cloud)
        {
            std::stringstream out;
            out << frames_saved;
            std::string cloud_name = "cloud" + out.str() + ".pcd";
            pcl::io::savePCDFile( cloud_name, *g_cloud, true );
            save_cloud = false;
        }
```

```
            // 解除互斥锁
            cld_mutex.unlock ();
        }

        // 显示图像
        if (g_image && img_mutex.try_lock ())
        {
            // 初始化
            if (!img_init)
            {
                rgb_data_size = g_image->getWidth () * g_image->getHeight ();
                rgb_data = new unsigned char [rgb_data_size * 3];
                img->setSize (g_image->getWidth (), g_image->getHeight ());
                img_init = true;
            }

            // 显示图像
            g_image->fillRGB (g_image->getWidth (), g_image->getHeight (),
                rgb_data);
            img->showRGBImage (rgb_data, g_image->getWidth (),
                g_image->getHeight ());

            // 保存图像
            if (save_image)
            {
                std::stringstream out;
                out << frames_saved;
                std::string image_name = "image" + out.str() + ".png";
                pcl::io::saveCharPNGFile (image_name, rgb_data,
                    g_image->getWidth(), g_image->getHeight(), 3);
                save_image = false;
            }

            // 解除互斥锁
            img_mutex.unlock ();
        }

        // 刷新可视化窗口
        cld->spinOnce ();
        img->spinOnce ();
    }

    // 获取停止
    interface->stop ();

    printf ("\n*********************************************************\n");
    printf ("\n");
}
```

calib.cpp

```
#include <pcl/io/pcd_io.h>
#include <pcl/registration/transforms.h>
#include <pcl/console/parse.h>

int
    main (int argc, char** argv)
{
    // PCD文件
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr source_cloud (new
        pcl::PointCloud<pcl::PointXYZRGBA> ());
    std::vector<int> pcd_indices = pcl::console::parse_file_extension_argument (argc,
        argv, "pcd");
    bool loadfail = true;
    if (pcd_indices.size () == 1)
    {
        if (pcl::io::loadPCDFile (argv[pcd_indices[0]], *source_cloud) == 0)
            loadfail = false;
    }
    if (loadfail)
    {
        printf ("Need one PCD File \"*.pcd\" as an input argument!\n");
        return -1;
    }

    // DAT文件
    std::string dat_filename = "calib";
    pcl::console::parse_argument (argc, argv, "-o", dat_filename);

    // 点云图像长宽
    int width = source_cloud->width;
    int height = source_cloud->height;

    // 中心点
    int centerX = -1;
    int centerY = -1;
    pcl::console::parse_2x_arguments (argc, argv, "-c", centerX, centerY);
    if (centerX < 0 || centerX >= width || centerY < 0 || centerY >= height) {
        printf ("Need valid Center Coordinate \"-c X,Y\" as an input argument!\n");
        return -1;
    }
    pcl::PointXYZRGBA center = source_cloud->points[centerX + centerY * width];

    // 参考点A
    int pointAX = -1;
    int pointAY = -1;
    pcl::console::parse_2x_arguments (argc, argv, "-a", pointAX, pointAY);
    if (pointAX < 0 || pointAX >= width || pointAY < 0 || pointAY >= height) {
        printf ("Need valid Reference Point A Coordinate \"-a X,Y\" as an input
            argument!\n");
        return -1;
    }
    pcl::PointXYZRGBA pointA = source_cloud->points[pointAX + pointAY * width];

    // 参考点B
    int pointBX = -1;
    int pointBY = -1;
    pcl::console::parse_2x_arguments (argc, argv, "-b", pointBX, pointBY);
    if (pointBX < 0 || pointBX >= width || pointBY < 0 || pointBY >= height) {
        printf ("Need valid Reference Point B Coordinate \"-b X,Y\" as an input
            argument!\n");
        return -1;
```

```
}
pcl::PointXYZRGBA pointB = source_cloud->points[pointBX + pointBY * width];

printf ("\n*************** Point Cloud Calibration ***************\n");
printf ("\n");

printf ("Input Point Cloud: %s\n", argv[pcd_indices[0]]);
printf ("\n");

// 求法向向量
float vectorAX = pointA.x - center.x;
float vectorAY = pointA.y - center.y;
float vectorAZ = pointA.z - center.z;
float vectorBX = pointB.x - center.x;
float vectorBY = pointB.y - center.y;
float vectorBZ = pointB.z - center.z;
float normalX = vectorAY * vectorBZ - vectorAZ * vectorBY;
float normalY = vectorAZ * vectorBX - vectorAX * vectorBZ;
float normalZ = vectorAX * vectorBY - vectorAY * vectorBX;
float normalMod = sqrt(normalX * normalX + normalY * normalY + normalZ * normalZ);
normalX = normalX / normalMod;
normalY = normalY / normalMod;
normalZ = normalZ / normalMod;
if (normalY < 0) {
    normalX = -normalX;
    normalY = -normalY;
    normalZ = -normalZ;
}

printf ("Center:            (%7.4f,%7.4f,%7.4f)\n", center.x, center.y, center.z);
printf ("Reference Point A:      (%7.4f,%7.4f,%7.4f)\n", pointA.x, pointA.y,
    pointA.z);
printf ("Reference Point B:      (%7.4f,%7.4f,%7.4f)\n", pointB.x, pointB.y,
    pointB.z);
printf ("Normal Vector:        |%7.4f,%7.4f,%7.4f|\n", normalX, normalY, normalZ);
printf ("\n");

// 旋转矩阵
Eigen::Matrix4f rotX = Eigen::Matrix4f::Identity();
float sqrtY2Z2 = sqrt(normalZ * normalZ + normalY * normalY);
rotX (1, 1) = normalY / sqrtY2Z2;
rotX (1, 2) = normalZ / sqrtY2Z2;
rotX (2, 1) = -normalZ / sqrtY2Z2;
rotX (2, 2) = normalY / sqrtY2Z2;
Eigen::Matrix4f rotZ = Eigen::Matrix4f::Identity();
float sqrtX2Y2 = sqrt(normalX * normalX + normalY * normalY);
rotZ (0, 0) = normalY / sqrtX2Y2;
rotZ (0, 1) = -normalX / sqrtX2Y2;
rotZ (1, 0) = normalX / sqrtX2Y2;
rotZ (1, 1) = normalY / sqrtX2Y2;
Eigen::Matrix4f rotmat = rotX * rotZ;

// 旋转
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr rotated_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA> ());
pcl::transformPointCloud (*source_cloud, *rotated_cloud, rotmat);

// 平移矩阵
center = rotated_cloud->points[centerX + centerY * width];
Eigen::Matrix4f transmat = Eigen::Matrix4f::Identity();
transmat (0, 3) = -center.x;
transmat (1, 3) = -center.y;
transmat (2, 3) = -center.z;
```

```
    // 平移
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr translated_cloud (new
        pcl::PointCloud<pcl::PointXYZRGBA> ());
    pcl::transformPointCloud (*rotated_cloud, *translated_cloud, transmat);

    printf ("Rotation Matrix:       |%7.4f %7.4f %7.4f|\n", rotmat (0,0), rotmat (0,1),
        rotmat (0,2));
    printf ("                       |%7.4f %7.4f %7.4f|\n", rotmat (1,0), rotmat (1,1), rotmat
        (1,2));
    printf ("                       |%7.4f %7.4f %7.4f|\n", rotmat (2,0), rotmat (2,1), rotmat
        (2,2));
    printf ("Translation Vector:     |%7.4f %7.4f %7.4f|\n", transmat (0,3), transmat
        (1,3), transmat (2,3));
    printf ("\n");

    // 新的转台中心点和边缘点
    center = translated_cloud->points[centerX + centerY * width];
    pointA = translated_cloud->points[pointAX + pointAY * width];
    pointB = translated_cloud->points[pointBX + pointBY * width];

    printf ("New Center:          (%7.4f,%7.4f,%7.4f)\n", center.x, center.y,
        center.z);
    printf ("New PointA:          (%7.4f,%7.4f,%7.4f)\n", pointA.x, pointA.y,
        pointA.z);
    printf ("New PointB:          (%7.4f,%7.4f,%7.4f)\n", pointB.x, pointB.y,
        pointB.z);
    printf ("\n");

    printf ("Save Calibration Data to %s.dat", dat_filename.c_str ());

    // 将矩阵写入文件 calib.dat
    std::ofstream fout;
    fout.open(dat_filename + ".dat", std::ios::binary);
    for (int i = 0; i < 3; ++i)
        for (int j = 0; j < 3; ++j)
            fout.write ((char*)& rotmat (i, j), sizeof(float));
    for (int i = 0; i < 3; ++i)
        fout.write ((char*)& transmat (i, 3), sizeof(float));
    fout.close();

    printf (" ... Done");
    printf ("\n");

    printf ("\n*********************************************************\n");
    printf ("\n");
}
```

preproc.cpp

```
#include <pcl/console/parse.h>
#include <pcl/io/pcd_io.h>
#include <pcl/registration/transforms.h>
#include <pcl/filters/passthrough.h>
#include <pcl/filters/statistical_outlier_removal.h>

int
    main (int argc, char** argv)
{
    // 校准数据文件
    std::ifstream fin;
    std::vector<int> dat_indices = pcl::console::parse_file_extension_argument (argc,
        argv, "dat");
    bool loadfail = true;
    if (dat_indices.size () == 1)
    {
        fin.open(argv[dat_indices[0]], std::ios::binary);
        if (fin.is_open ())
            loadfail = false;
    }
    if (loadfail)
    {
        printf ("Need one DAT File \"*.dat\" as an input argument!\n");
        return -1;
    }
    Eigen::Matrix4f rotmat = Eigen::Matrix4f::Identity();
    for (int i = 0; i < 3; ++i)
        for (int j = 0; j < 3; ++j)
            fin.read ((char*)& rotmat (i, j), sizeof(float));
    Eigen::Matrix4f transmat = Eigen::Matrix4f::Identity();
    for (int i = 0; i < 3; ++i)
        fin.read ((char*)& transmat (i, 3), sizeof(float));
    fin.close();

    // 输入点云文件
    std::vector<std::pair<std::string, pcl::PointCloud<pcl::PointXYZRGBA>::Ptr>>
        clouds;
    std::vector<int> pcd_indices = pcl::console::parse_file_extension_argument (argc,
        argv, "pcd");
    for (int i = 0; i < pcd_indices.size (); ++i)
    {
        pcl::PointCloud<pcl::PointXYZRGBA>::Ptr pc (new
            pcl::PointCloud<pcl::PointXYZRGBA>);
        if (pcl::io::loadPCDFile (argv[pcd_indices[i]], *pc) < 0) {
            printf ("Failed to load PCD File %s!\n", argv[pcd_indices[i]]);
            continue;
        }
        clouds.push_back (std::pair<std::string,
            pcl::PointCloud<pcl::PointXYZRGBA>::Ptr> (argv[pcd_indices[i]], pc));
    }
    if (clouds.size () < 1)
    {
        printf ("Need at least one PCD File \"*.pcd\" as an input argument!\n");
        return -1;
    }

    // 输出点云前缀
    std::string out_prefix = "proc_cloud";
    pcl::console::parse_argument (argc, argv, "-o", out_prefix);

    // 直通滤波器参数
```

```
float rangeX = 1.0f;
float rangeY = 1.0f;
float rangeZ = 1.0f;
pcl::console::parse_3x_arguments (argc, argv, "-p", rangeX, rangeY, rangeZ);

// 统计滤波器参数
float tempK = 30.0f;
float stddevMulThresh = 1.0f;
pcl::console::parse_2x_arguments (argc, argv, "-s", tempK, stddevMulThresh);
int meanK = (int) tempK;

// 旋转步进角
float currDeg = 0.0f;
float deltaDeg = 0.0f;
pcl::console::parse_argument (argc, argv, "-r", deltaDeg);

printf ("\n*************** Point Cloud Pre-processing ***************\n");
printf ("\n");

printf ("Input Point Clouds:");
for (int i = 0; i < clouds.size (); ++i)
    printf (" %s", clouds[i].first.c_str ());
printf ("\n\n");

printf ("1 - Transformation\n");
printf ("Rotation Matrix:    |%7.4f %7.4f %7.4f|\n", rotmat (0,0), rotmat (0,1),
    rotmat (0,2));
printf ("                    |%7.4f %7.4f %7.4f|\n", rotmat (1,0), rotmat (1,1), rotmat
    (1,2));
printf ("                    |%7.4f %7.4f %7.4f|\n", rotmat (2,0), rotmat (2,1), rotmat
    (2,2));
printf ("Translation Vector:  |%7.4f %7.4f %7.4f|\n", transmat (0,3), transmat
    (1,3), transmat (2,3));
printf ("\n");

printf ("2 - Axes Exchange\n");
printf ("-Z -> X'; X -> Y'; -Y -> Z'\n");
printf ("\n");

printf ("3 - Pass Through Filter\n");
printf ("Range X:         %.4f - %.4f\n", -rangeX / 2, rangeX / 2);
printf ("Range Y:         %.4f - %.4f\n", -rangeY / 2, rangeY / 2);
printf ("Range Z:         %.4f - %.4f\n", 0.01, rangeZ);
printf ("\n");

printf ("4 - Statistical Outlier Removal\n");
printf ("Mean K:          %d\n", meanK);
printf ("Stddev Mul Threshold: %.4f\n", stddevMulThresh);
printf ("\n");

printf ("5 - Incremental Rotation\n");
printf ("Stepping Angle:     %.4f deg\n", deltaDeg);
printf ("\n");

// 分别处理每个点云
for (int i = 0; i < clouds.size (); ++i)
{
    std::stringstream out_filename;
    out_filename << out_prefix << i + 1 << ".pcd";
    printf ("Pre-process Point Cloud %s -> %s", clouds[i].first.c_str (),
        out_filename.str ().c_str ());

    // 读取点云
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr source_cloud (new
```

```
    pcl::PointCloud<pcl::PointXYZRGBA> ());
source_cloud = clouds[i].second;

// 矩阵变换
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr rotated_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA> ());
pcl::transformPointCloud (*source_cloud, *rotated_cloud, rotmat);
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr translated_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA> ());
pcl::transformPointCloud (*rotated_cloud, *translated_cloud, transmat);

// 直通滤波器X (-Z)
pcl::PassThrough<pcl::PointXYZRGBA> passX;
passX.setInputCloud (translated_cloud);
passX.setFilterFieldName ("z");
passX.setFilterLimits (-rangeX / 2, rangeX / 2);
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr passX_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA>());
passX.filter (*passX_cloud);

// 直通滤波器Y (X)
pcl::PassThrough<pcl::PointXYZRGBA> passY;
passY.setInputCloud (passX_cloud);
passY.setFilterFieldName ("x");
passY.setFilterLimits (-rangeY / 2, rangeY / 2);
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr passY_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA>());
passY.filter (*passY_cloud);

// 直通滤波器Z (-Y)
pcl::PassThrough<pcl::PointXYZRGBA> passZ;
passZ.setInputCloud (passY_cloud);
passZ.setFilterFieldName ("y");
passZ.setFilterLimits (-rangeZ, -0.01);
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr passZ_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA>());
passZ.filter (*passZ_cloud);

// 统计滤波器
pcl::StatisticalOutlierRemoval<pcl::PointXYZRGBA> sor;
sor.setInputCloud(passZ_cloud);
sor.setMeanK(meanK);
sor.setStddevMulThresh(stddevMulThresh);
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr inlier_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA>());
sor.filter(*inlier_cloud);

// 变换坐标轴
for (size_t i = 0; i < inlier_cloud->points.size(); ++i)
{
    pcl::PointXYZRGBA &pt = inlier_cloud->points[i];
    float x = -pt.z;
    float y = pt.x;
    float z = -pt.y;
    pt.x = x;
    pt.y = y;
    pt.z = z;
}

// 步进旋转
Eigen::Matrix4f incrotmat = Eigen::Matrix4f::Identity();
float currRad = M_PI / 180 * currDeg;
incrotmat (0, 0) = cos(currRad);
```

```
            incrotmat (0, 1) = -sin(currRad);
            incrotmat (1, 0) = sin(currRad);
            incrotmat (1, 1) = cos(currRad);
            pcl::PointCloud<pcl::PointXYZRGBA>::Ptr incrrot_cloud (new
                pcl::PointCloud<pcl::PointXYZRGBA> ());
            pcl::transformPointCloud (*inlier_cloud, *incrrot_cloud, incrotmat);
            currDeg += deltaDeg;

            // 输出点云
            pcl::io::savePCDFile (out_filename.str (), *incrrot_cloud, true);

            printf (" ... Done\n");
        }

        printf ("\n*************************************************************\n");
        printf ("\n");
}
```

register.cpp

```
#include <pcl/console/parse.h>
#include <pcl/io/pcd_io.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/features/normal_3d.h>
#include <pcl/registration/icp_nl.h>
#include <pcl/visualization/pcl_visualizer.h>

using pcl::visualization::PointCloudColorHandlerGenericField;
using pcl::visualization::PointCloudColorHandlerCustom;

pcl::visualization::PCLVisualizer *p;
int vp_1, vp_2;

class MyPointRepresentation : public pcl::PointRepresentation <pcl::PointNormal>
{
    using pcl::PointRepresentation<pcl::PointNormal>::nr_dimensions_;

public:

    MyPointRepresentation ()
    {
        nr_dimensions_ = 4;
    }

    virtual void copyToFloatArray (const pcl::PointNormal &p, float * out) const
    {
        out[0] = p.x;
        out[1] = p.y;
        out[2] = p.z;
        out[3] = p.curvature;
    }
};

// 在左边显示原始点云
void showCloudsLeft(const pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud_source, const
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud_target)
{
    p->removePointCloud ("vp1_target");
    p->removePointCloud ("vp1_source");

    PointCloudColorHandlerCustom<pcl::PointXYZRGBA> tgt_h (cloud_target, 0, 255, 0);
    PointCloudColorHandlerCustom<pcl::PointXYZRGBA> src_h (cloud_source, 255, 0, 0);
    p->addPointCloud (cloud_target, tgt_h, "vp1_target", vp_1);
    p->addPointCloud (cloud_source, src_h, "vp1_source", vp_1);

    p->spin();
}

// 在右边显示配准点云
void showCloudsRight(const pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud_source, const
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud_target)
{
    p->removePointCloud ("target");
    p->removePointCloud ("source");

    PointCloudColorHandlerCustom<pcl::PointXYZRGBA> tgt_h (cloud_target, 0, 255, 0);
    PointCloudColorHandlerCustom<pcl::PointXYZRGBA> src_h (cloud_source, 255, 0, 0);
    p->addPointCloud (cloud_target, tgt_h, "target", vp_2);
    p->addPointCloud (cloud_source, src_h, "source", vp_2);

    p->spin();
```

```
}
void showCloudsRight(const pcl::PointCloud<pcl::PointNormal>::Ptr cloud_source, const
    pcl::PointCloud<pcl::PointNormal>::Ptr cloud_target)
{
    p->removePointCloud ("source");
    p->removePointCloud ("target");

    PointCloudColorHandlerCustom<pcl::PointNormal> tgt_color_handler (cloud_target, 0,
        255, 0);
    PointCloudColorHandlerCustom<pcl::PointNormal> src_color_handler (cloud_source,
        255, 0, 0);

    p->addPointCloud (cloud_target, tgt_color_handler, "target", vp_2);
    p->addPointCloud (cloud_source, src_color_handler, "source", vp_2);

    p->spinOnce();
}

// 两个点云间配准
float pairAlign (const pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud_src, const
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud_tgt,
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr output, float leafSize, int searchK, float
    correspondenceDistance, int iterationNumber)
{
    // 向下采样
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr src (new
        pcl::PointCloud<pcl::PointXYZRGBA>);
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr tgt (new
        pcl::PointCloud<pcl::PointXYZRGBA>);
    pcl::VoxelGrid<pcl::PointXYZRGBA> grid;
    grid.setLeafSize (leafSize, leafSize, leafSize);
    grid.setInputCloud (cloud_src);
    grid.filter (*src);
    grid.setInputCloud (cloud_tgt);
    grid.filter (*tgt);

    // 计算法线和曲率
    pcl::PointCloud<pcl::PointNormal>::Ptr points_with_normals_src (new
        pcl::PointCloud<pcl::PointNormal>);
    pcl::PointCloud<pcl::PointNormal>::Ptr points_with_normals_tgt (new
        pcl::PointCloud<pcl::PointNormal>);
    pcl::NormalEstimation<pcl::PointXYZRGBA, pcl::PointNormal> norm_est;
    pcl::search::KdTree<pcl::PointXYZRGBA>::Ptr tree (new
        pcl::search::KdTree<pcl::PointXYZRGBA> ());
    norm_est.setSearchMethod (tree);
    norm_est.setKSearch (searchK);
    norm_est.setInputCloud (src);
    norm_est.compute (*points_with_normals_src);
    pcl::copyPointCloud (*src, *points_with_normals_src);
    norm_est.setInputCloud (tgt);
    norm_est.compute (*points_with_normals_tgt);
    pcl::copyPointCloud (*tgt, *points_with_normals_tgt);

    // 初始化自定义点表达并设定权值
    MyPointRepresentation point_representation;
    float alpha[4] = {1.0, 1.0, 1.0, 1.0};
    point_representation.setRescaleValues (alpha);

    // 配准设置
    pcl::IterativeClosestPointNonLinear<pcl::PointNormal, pcl::PointNormal> reg;
    reg.setTransformationEpsilon (1e-8);
    reg.setPointRepresentation (boost::make_shared<const MyPointRepresentation>
        (point_representation));
```

```
    reg.setMaxCorrespondenceDistance (correspondenceDistance); // 对齐距离，根据实际情况调整
    reg.setInputCloud (points_with_normals_src);
    reg.setInputTarget (points_with_normals_tgt);

    // 循环配准并可视化结果
    Eigen::Matrix4f Ti = Eigen::Matrix4f::Identity (), prev, targetToSource;
    pcl::PointCloud<pcl::PointNormal>::Ptr reg_result = points_with_normals_src;
    reg.setMaximumIterations (1);

    for (int i = 0; i < iterationNumber; ++i)
    {
        // 上次的目标作为本次的源
        points_with_normals_src = reg_result;

        // 配准
        reg.setInputCloud (points_with_normals_src);
        reg.align (*reg_result);

        // 每次递归累积变换矩阵
        Ti = reg.getFinalTransformation () * Ti;

        // 两次变换矩阵之差小于某阈值则减小对齐距离
        if (fabs ((reg.getLastIncrementalTransformation () - prev).sum ()) <
            reg.getTransformationEpsilon ())
            reg.setMaxCorrespondenceDistance (reg.getMaxCorrespondenceDistance () /
                1.01);
        prev = reg.getLastIncrementalTransformation ();

        // 可视化当前状态
        showCloudsRight(points_with_normals_src, points_with_normals_tgt);

        // 显示进度
        if (i % (iterationNumber / 50) == 0)
            printf ("|");
    }

    // 从目标点云到源坐标系的变换
    Eigen::Matrix4f final_transform = Ti.inverse();
    pcl::transformPointCloud (*cloud_tgt, *output, final_transform);

    return reg.getFitnessScore ();
}

int main (int argc, char** argv)
{
    // 输入点云文件
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr source_cloud (new
        pcl::PointCloud<pcl::PointXYZRGBA> ());
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr target_cloud (new
        pcl::PointCloud<pcl::PointXYZRGBA> ());
    std::vector<int> pcd_indices = pcl::console::parse_file_extension_argument (argc,
        argv, "pcd");
    bool loadfail = true;
    if (pcd_indices.size () == 2)
    {
        if (pcl::io::loadPCDFile (argv[pcd_indices[0]], *source_cloud) == 0 &&
            pcl::io::loadPCDFile (argv[pcd_indices[1]], *target_cloud) == 0)
            loadfail = false;
    }
    if (loadfail)
    {
        printf ("Need two PCD Files \"*.pcd *.pcd\" as an input argument!\n");
        return -1;
```

```
}

// 输出点云前缀
std::string out_prefix = "reg_cloud";
pcl::console::parse_argument (argc, argv, "-o", out_prefix);

// 读取下采样参数
float leafSize = 0.05f;
pcl::console::parse_argument (argc, argv, "-v", leafSize);

// 读取法线估计参数
int searchK = 30;
pcl::console::parse_argument (argc, argv, "-n", searchK);

// 读取ICP配准参数
float correspondenceDistance = 0.1;
float tempIterationNumber = 30.0f;
pcl::console::parse_2x_arguments (argc, argv, "-r", correspondenceDistance,
    tempIterationNumber);
int iterationNumber = (int) tempIterationNumber;

printf ("\n*************** Point Clouds Registration ***************\n");
printf ("\n");

printf ("Source Point Cloud:      %s\n", argv[pcd_indices[0]]);
printf ("Source Point:          %d\n", source_cloud->size ());
printf ("Source Point Cloud:      %s\n", argv[pcd_indices[1]]);
printf ("Source Point:          %d\n", target_cloud->size ());
printf ("\n");

printf ("Voxelization Leaf Size:    %.4f\n", leafSize);
printf ("Normal Est. K Search:     %d\n", searchK);
printf ("Max Correspondence Distance: %.4f\n", correspondenceDistance);
printf ("Iteration Number:       %d\n", iterationNumber);
printf ("\n");

// 创建可视化窗口
p = new pcl::visualization::PCLVisualizer ("Pairwise Registration");
p->createViewPort (0.0, 0, 0.5, 1.0, vp_1);
p->createViewPort (0.5, 0, 1.0, 1.0, vp_2);

// 在左侧显示原始点云
showCloudsLeft(source_cloud, target_cloud);

printf ("Register Two Point Clouds\n");

// 执行配准
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr new_target_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA>);
float fitness = pairAlign (source_cloud, target_cloud, new_target_cloud, leafSize,
    searchK, correspondenceDistance, iterationNumber);

printf ("\n");
printf ("... Done\n");
printf("Fitness Score: %.4f\n", fitness);
printf ("\n");

printf ("Save Registered Point Cloud to %s.pcd", out_prefix.c_str ());

// 合成点云
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr full_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA>);
*full_cloud += *source_cloud;
```

```
    *full_cloud += *new_target_cloud;

    // 保存合成点云
    pcl::io::savePCDFile (out_prefix + ".pcd", *full_cloud, true);

    printf (" ... Done\n");

    // 在右侧显示配准点云
    showCloudsRight(source_cloud, new_target_cloud);

    printf ("\n*************************************************************\n");
    printf ("\n");
}
```

postproc.cpp

```
#include <pcl/console/parse.h>
#include <pcl/io/pcd_io.h>
#include <pcl/filters/radius_outlier_removal.h>
#include <pcl/surface/mls.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/common/time.h>

int
    main (int argc, char** argv)
{
    // 输入点云文件
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr source_cloud (new
        pcl::PointCloud<pcl::PointXYZRGBA>);
    std::vector<int> pcd_indices = pcl::console::parse_file_extension_argument (argc,
        argv, "pcd");
    bool loadfail = true;
    if (pcd_indices.size () == 1)
    {
        if (pcl::io::loadPCDFile (argv[pcd_indices[0]], *source_cloud) == 0)
            loadfail = false;
    }
    if (loadfail)
    {
        printf ("Need one PCD File \"*.pcd\" as an input argument!\n");
        return -1;
    }

    // 输出点云前缀
    std::string out_prefix = "final_cloud";
    pcl::console::parse_argument (argc, argv, "-o", out_prefix);

    // 半径离群点滤波器参数
    float radius = 0.01f;
    float tempNeighbors = 10.0f;
    int radius_enable = pcl::console::parse_2x_arguments (argc, argv, "-r", radius,
        tempNeighbors);
    int neighbors = (int) tempNeighbors;

    // 体素化参数
    float leafsize = 0.01f;
    int voxel_enable = pcl::console::parse_argument (argc, argv, "-v", leafsize);

    // 移动最小二乘平滑参数
    float mls_radius = 0.01f;
    float tempOrder = 3.0f;
    int mls_enable = pcl::console::parse_2x_arguments (argc, argv, "-m", mls_radius,
        tempOrder);
    int order = (float) tempOrder;

    // 初始化秒表
    pcl::StopWatch watch;
    double time = 0;
    double total_time = 0;

    printf ("\n*************** Point Cloud Pre-processing ***************\n");
    printf ("\n");

    printf ("Input Point Cloud: %s\n", argv[pcd_indices[0]]);
    printf ("\n");

    // 半径离群点滤波器
```

```
printf ("Radius Outlier Removal");
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr inlier_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA>);
if (radius_enable < 0)
{
    inlier_cloud = source_cloud;
    printf (" ... Disabled\n");
    printf ("\n");
}
else
{
    watch.reset();
    pcl::RadiusOutlierRemoval<pcl::PointXYZRGBA> outrem;
    outrem.setInputCloud(source_cloud);
    outrem.setRadiusSearch(radius);
    outrem.setMinNeighborsInRadius (neighbors);
    outrem.filter (*inlier_cloud);
    time = watch.getTimeSeconds();
    total_time += time;

    printf (" ... Done\n");
    printf ("Radius Search:       %.4f m\n", radius);
    printf ("Min Neighbors in Radius: %d\n", neighbors);
    printf ("Point:            %d -> %d\n", source_cloud->size (),
        inlier_cloud->size ());
    printf ("Time Used:         %.4f s // %.4f s\n", time, total_time);
    printf ("\n");
}

// 体素化
printf ("Point Cloud Voxelization");
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr sampled_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA>);
if (voxel_enable < 0)
{
    sampled_cloud = inlier_cloud;
    printf (" ... Disabled\n");
    printf ("\n");
}
else
{
    watch.reset();
    pcl::VoxelGrid<pcl::PointXYZRGBA> grid;
    grid.setLeafSize (leafsize, leafsize, leafsize);
    grid.setInputCloud (inlier_cloud);
    grid.filter (*sampled_cloud);
    time = watch.getTimeSeconds();
    total_time += time;

    printf (" ... Done\n");
    printf ("Leaf Size:         %.4f m\n", leafsize);
    printf ("Point:            %d -> %d\n", inlier_cloud->size (),
        sampled_cloud->size ());
    printf ("Time Used:         %.4f s // %.4f s\n", time, total_time);
    printf ("\n");
}

// 移动最小二乘平滑
printf ("Moving Least Square Smoothing");
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr smoothed_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA>);
if (mls_enable < 0)
{
    smoothed_cloud = sampled_cloud;
```

```
            printf (" ... Disabled\n");
            printf ("\n");
        }
        else
        {
            watch.reset();
            pcl::search::KdTree<pcl::PointXYZRGBA>::Ptr tree (new
                pcl::search::KdTree<pcl::PointXYZRGBA>);
            pcl::MovingLeastSquares<pcl::PointXYZRGBA, pcl::PointXYZRGBA> mls;
            mls.setComputeNormals (false);
            mls.setSearchMethod (tree);
            mls.setSearchRadius (mls_radius);
            mls.setPolynomialFit (true);
            mls.setPolynomialOrder (order);
            mls.setUpsamplingMethod (pcl::MovingLeastSquares<pcl::PointXYZRGBA,
                pcl::PointXYZRGBA>::UpsamplingMethod::NONE);
            mls.setInputCloud (sampled_cloud);
            mls.process (*smoothed_cloud);
            time = watch.getTimeSeconds();
            total_time += time;

            printf (" ... Done\n");
            printf ("Search Radius:       %.4f m\n", mls_radius);
            printf ("Polynomial Order:     %d\n", order);
            printf ("Time Used:          %.4f s // %.4f s\n", time, total_time);
            printf ("\n");
        }

        // 输出点云
        printf ("Save Processed Point Cloud to %s.pcd", out_prefix.c_str ());
        pcl::io::savePCDFile (out_prefix + ".pcd", *smoothed_cloud, true);
        printf (" ... Done\n");

        printf ("\n************************************************************\n");
        printf ("\n");
}
```

measure.cpp

```
#include <pcl/console/parse.h>
#include <pcl/io/pcd_io.h>
#include <pcl/common/time.h>
#include <pcl/filters/passthrough.h>

// 凸包算法所使用的点类型比较器
bool compare (const pcl::PointXY &a, const pcl::PointXY &b)
{
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

// 凸包算法所使用的方向判断函数
inline float cross (const pcl::PointXY &o, const pcl::PointXY &a, const pcl::PointXY &b)
{
    return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x);
}

// 单调链凸包算法
void convex_hull (pcl::PointCloud<pcl::PointXY>& p)
{
    // 变量初始化
    int n = p.size (), k = 0;
    std::vector<pcl::PointXY, Eigen::aligned_allocator<pcl::PointXY>> h(2*n);

    // 排序
    sort(p.points.begin (), p.points.end(), compare);

    // 计算第一个单调链
    for (int i = 0; i < n; ++i) {
        while (k >= 2 && cross(h[k-2], h[k-1], p.points[i]) <= 0) k--;
        h[k++] = p.points[i];
    }

    // 计算第二个单调链
    for (int i = n-2, t = k+1; i >= 0; i--) {
        while (k >= t && cross(h[k-2], h[k-1], p.points[i]) <= 0) k--;
        h[k++] = p.points[i];
    }

    // 存入点云
    h.resize (k);
    p.resize (k);
    p.points = h;
}

// 计算两点间距离
inline float dist (const pcl::PointXY &a, const pcl::PointXY &b)
{
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

// 计算凸包周长
float comp_perimeter (const pcl::PointCloud<pcl::PointXY>& p)
{
    if (p.size () < 3)
        return 0.0f;

    float perimeter = dist (p.points[p.size () - 1], p.points[0]);
    for (int i = 1; i < p.size (); ++i)
    {
```

```
        perimeter += dist (p.points[i - 1], p.points[i]);
    }
    return perimeter;
}

// 计算两点与原点构成的三角形的面积的两倍
inline float area_2x_origin (const pcl::PointXY &a, const pcl::PointXY &b)
{
    return a.x * b.y - a.y * b.x;
}

// 计算凸包面积
float comp_area (const pcl::PointCloud<pcl::PointXY>& p)
{
    if (p.size () < 3)
        return 0.0f;

    float area_2x = area_2x_origin (p.points[p.size () - 1], p.points[0]);
    for (int i = 1; i < p.size (); ++i)
    {
        area_2x += area_2x_origin (p.points[i - 1], p.points[i]);
    }
    return abs (area_2x) / 2;
}

// 计算凸包相关直径
void comp_diameter (const pcl::PointCloud<pcl::PointXY>& p, float& max_length, float&
breadth, float& average_diameter, float& feret_diameter)
{
    if (p.size () < 3)
    {
        max_length = 0;
        breadth = 0;
        average_diameter = 0;
        feret_diameter = 0;
    }

    max_length = 0;
    breadth = 0;
    average_diameter = 0;
    feret_diameter = 0;
    float min_length = FLT_MAX;
    for (int deg = 0; deg < 90; deg += 1)
    {
        float x_left = FLT_MAX;
        float x_right = FLT_MIN;
        float y_bottom = FLT_MAX;
        float y_top = FLT_MIN;

        float theta = deg * M_PI / 180;
        float cost = cos(theta);
        float sint = sin(theta);

        for (int i = 0; i < p.size (); ++i)
        {
            const pcl::PointXY &pt = p.points[i];
            float xt = pt.x * cost - pt.y * sint;
            float yt = pt.x * sint + pt.y * cost;

            if (xt < x_left) x_left = xt;
            if (xt > x_right) x_right = xt;
            if (yt < y_bottom) y_bottom = yt;
            if (yt > y_top) y_top = yt;
        }
```

```
            float x_range = x_right - x_left;
            float y_range = y_top - y_bottom;

            feret_diameter += x_range + y_range;
            if (x_range > max_length)
            {
                max_length = x_range;
                breadth = y_range;
            }
            if (y_range > max_length)
            {
                max_length = y_range;
                breadth = x_range;
            }
            if (x_range < min_length)
                min_length = x_range;
            if (y_range < min_length)
                min_length = y_range;
        }
        average_diameter = (max_length + min_length) / 2;
        feret_diameter /= 180;
}

int main (int argc, char** argv)
{
        // PCD文件
        pcl::PointCloud<pcl::PointXYZRGBA>::Ptr source_cloud (new
            pcl::PointCloud<pcl::PointXYZRGBA> ());
        std::vector<int> pcd_indices = pcl::console::parse_file_extension_argument (argc,
            argv, "pcd");
        bool loadfail = true;
        if (pcd_indices.size () == 1)
        {
            if (pcl::io::loadPCDFile (argv[pcd_indices[0]], *source_cloud) == 0)
                loadfail = false;
        }
        if (loadfail)
        {
            printf ("Need one PCD File \"*.pcd\" as an input argument!\n");
            return -1;
        }

        //初始化秒表
        pcl::StopWatch watch;
        double time = 0;
        double total_time = 0;

        //输出输出格式
        std::cout.setf(std::ios::fixed);
        std::cout.setf(std::ios::showpoint);
        std::cout.precision(3);

        std::cout << std::endl << "*************** Point Cloud Measurement ***************"
            << std::endl << std::endl;

        std::cout << "Input Point Cloud: " << argv[pcd_indices[0]] << std::endl
                << "Input Point: " << source_cloud->points.size () << std::endl
                << std::endl;

        //测量基部
        std::cout << "Compute Base Parameters";
        watch.reset();
```

```
pcl::PassThrough<pcl::PointXYZRGBA> pass_base;
pass_base.setInputCloud (source_cloud);
pass_base.setFilterFieldName ("z");
pass_base.setFilterLimits (0.005f, 0.015f);
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr base_cloud (new
    pcl::PointCloud<pcl::PointXYZRGBA>);
pass_base.filter (*base_cloud);
pcl::PointCloud<pcl::PointXY>::Ptr base_points (new pcl::PointCloud<pcl::PointXY>);
pcl::copyPointCloud (*base_cloud, *base_points);
convex_hull(*base_points);
float base_max_length;
float base_breadth;
float base_average_diameter;
float base_feret_diameter;
comp_diameter (*base_points, base_max_length, base_breadth, base_average_diameter,
    base_feret_diameter);
float base_perimeter = comp_perimeter (*base_points);
float base_area = comp_area (*base_points);

time = watch.getTimeSeconds();
total_time += time;
std::cout << " ... Done" << std::endl
        << "Max Length: " << base_max_length << " m" << std::endl
        << "Breadth: " << base_breadth << " m" << std::endl
        << "Average Diameter: " << base_average_diameter << " m" << std::endl
        << "Average Feret Diameter: " << base_feret_diameter << " m" << std::endl
        << "Perimeter: " << base_perimeter << " m" << std::endl
        << "Area: " << base_area << " m2" << std::endl
        << "Time Used: " << time << " s // " << total_time << " s" << std::endl
        << std::endl;

// 测量冠层
std::cout << "Compute Crown Parameters";
watch.reset();

pcl::PointCloud<pcl::PointXY>::Ptr crown_points (new
    pcl::PointCloud<pcl::PointXY>);
pcl::copyPointCloud (*source_cloud, *crown_points);
convex_hull(*crown_points);
float crown_max_length;
float crown_breadth;
float crown_average_diameter;
float crown_feret_diameter;
comp_diameter (*crown_points, crown_max_length, crown_breadth,
    crown_average_diameter, crown_feret_diameter);
float crown_perimeter = comp_perimeter (*crown_points);
float crown_area = comp_area (*crown_points);

time = watch.getTimeSeconds();
total_time += time;
std::cout << " ... Done" << std::endl
        << "Max Length: " << crown_max_length << " m" << std::endl
        << "Breadth: " << crown_breadth << " m" << std::endl
        << "Average Diameter: " << crown_average_diameter << " m" << std::endl
        << "Average Feret Diameter: " << crown_feret_diameter << " m" << std::endl
        << "Perimeter: " << crown_perimeter << " m" << std::endl
        << "Area: " << crown_area << " m2" << std::endl
        << "Time Used: " << time << " s // " << total_time << " s" << std::endl
        << std::endl;

// 测量整体
std::cout << "Compute Plant Parameters";
watch.reset();
```

```
    float z_max = 0;
    for (size_t i = 0; i < source_cloud->points.size(); ++i)
    {
        pcl::PointXYZRGBA &pt = source_cloud->points[i];
        if (pt.z > z_max)
            z_max = pt.z;
    }
    float volume = 0;
    for (float zt = 0.005; zt <= z_max - 0.005; zt += 0.01)
    {
        pcl::PassThrough<pcl::PointXYZRGBA> pass;
        pass.setInputCloud (source_cloud);
        pass.setFilterFieldName ("z");
        pass.setFilterLimits (zt - 0.005, zt + 0.005);
        pcl::PointCloud<pcl::PointXYZRGBA>::Ptr plane_cloud (new
            pcl::PointCloud<pcl::PointXYZRGBA>);
        pass.filter (*plane_cloud);
        pcl::PointCloud<pcl::PointXY>::Ptr plane_points (new
            pcl::PointCloud<pcl::PointXY>);
        pcl::copyPointCloud (*plane_cloud, *plane_points);
        convex_hull(*plane_points);
        float area = comp_area (*plane_points);
        volume += area * 0.01;
    }

    time = watch.getTimeSeconds();
    total_time += time;
    std::cout << " ... Done" << std::endl
            << "Height: " << z_max << " m" << std::endl
            << "Volume: " << volume << " m3" << std::endl
            << "Time Used: " << time << " s // " << total_time << " s" << std::endl;

    std::cout << std::endl << "*****************************************************"
        << std::endl << std::endl;
}
```

浙江大学本科生毕业论文（设计）诚信承诺书

1.本人郑重地承诺所呈交的毕业论文（设计），是在指导教师的指导下严格按照学校和学院有关规定完成的。

2.本人在毕业论文（设计）中引用他人的观点和参考资料均加以注释和说明。

3. 本人承诺在毕业论文（设计）选题和研究内容过程中没有抄袭他人研究成果和伪造相关数据等行为。

4. 在毕业论文（设计）中对侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

毕业论文（设计）作者签名：

_____年_____月_____日

本科生毕业论文（设计）任务书

一、题目：基于 Kinect 深度信息的植物无损测量

二、指导教师对毕业论文（设计）的进度安排及任务要求：

2013年9月—10月：　前期研究，查找与课题相关的文献，进行仔细研究和归纳总结，并撰写文献综述和外文翻译。

2013年11月：　课题开题，考虑课题研究要达成的具体目标和可能的实现步骤，撰写开题报告。

2013年12月：　技术学习，学习课题研究中可能用到的相关技术，并进行一些初期的实践工作。

2014年1月—2月：　数据获取，根据开题中提出的方法进行数据获取工作，并在过程中不断完善，获得完整的植物三维数据。

2014年3月—4月：　实验进行，通过一定的方法从获取的数据中进行植物的测量工作，并与物理测量结果进行对比。

2014年5月上旬：　后期完善，对误差进行分析，从可能的误差原因入手对测量流程进行改进。

2014年5月中下旬：　论文撰写，对整个课题研究的流程进行总结，撰写毕业论文并进行修改。

起讫日期　**2013** 年 **9** 月 **1** 日 至 **2014** 年 **5** 月 **25** 日

指导教师（签名）＿＿＿＿职称＿＿＿＿

三、系或研究所审核意见：

负责人（签名）＿＿＿＿＿＿

年　月　日

毕 业 论 文（设计） 考 核

一、指导教师对毕业论文（设计）的评语：

指导教师(签名）

年 月 日

二、答辩小组对毕业论文（设计）的答辩评语及总评成绩：

成绩比例	文献综述 占（10%）	开题报告 占（20%）	外文翻译 占（10%）	毕业论文（设计）质量及答辩 占（60%）	总评成绩
分 值					

答辩小组负责人（签名）

年 月 日

浙江大學

本 科 生 毕 业 论 文（设计）

文献综述和开题报告

姓名与学号 3100100423 马轲

指导教师 蒋焕煜

年级与专业 2010 生物系统工程

所在学院 生物系统工程与食品科学学院

一、题目：基于 Kinect 深度信息的植物无损测量

二、指导教师对文献综述和开题报告的具体内容要求：

指导教师（签名）_____

年　　月　　日

目录

文献综述

摘　要 对植物一些结构性生长参数的测量在科学研究和农业生产中都具有十分重要的意义，而植物无损测量技术具有非接触式和非破坏性的特点，适应未来农业生产中对植物大范围实时动态监测的需要。Kinect 作为自然交互设备的代表，能够以低廉的成本提供配准的彩色图像和深度图像，已被证实可应用于植物无损测量领域。本文概述了基于 Kinect 的植物无损测量的国内外研究现状和主要研究方向及进展，并提出了亟待解决的问题和对未来研究的期望。

关键词 Kinect; 深度图像; 无损测量

1 背景介绍

对植物一些结构性生长参数的测量在科学研究和农业生产中都具有十分重要的意义。测量方法则主要分为接触式、破坏性的测量和非接触式、非破坏性的测量两种，而在后者中，基于机器视觉的植物无损测量展现出极大的发展前景。Kinect 作为自然交互设备的代表，能够以低廉的成本提供配准的彩色图像和深度图像，已被证实可应用于植物无损测量领域。

1.1 植物无损测量

1.1.1 植物无损测量的意义

随着信息技术的迅猛发展，许多传统行业也呈现出崭新的面貌。数字农业作为农业技术与信息技术的交叉领域，已经在当下呈现出了良好的前景，称为现代农业发展的新方向之一。数字农业不仅将继承了地理学、农学、生态学、植物生理学、土壤学等基础学科在农业生产中的指导思想，还结合了全球定位系统、地理信息系统、遥感技术以及计算机技术、通信网络技术、自动化技术等高新技术，从而实现在农业生产过程中对农作物及其相关环境的全方位实时监测与控制，辅以对农业生产中的现象、过程进行模拟等手段，达到优化农业资源配置、降低生产成本、改善生态环境、提升产品品质的目的。

在生产过程中对植物的各种生长参数（本文主要研究结构性参数）进行测量正是数字农业的一个重要的研究领域，在生产实践中具有着重要的意义。植物的生长是一个极为复杂的过程，不仅与植物内部的遗传因素密切相关，同时还极大程度地受到环境因素和随机因素的影响，难以用简单的模型进行描述，其定量化表示也是长期以来的研究热点。生长参数，例如株高、株宽、叶面积等，通常可以直观地反映植物的生长情况，这就为进一步研究提供了基础。这些数据可被广泛应用于生长监测、科学建模、品质控制等诸多方面。

植物的生长参数测量在早期以接触式和破坏性测量为主，即通过将植物部分或整体分离，利用直尺及其他手工测量工具对各种参数进行测量。这类方法虽然在部分参数的测量方面能够得到较为精确的结果，但其弊端也是显而易见的。由于其破坏性，测量难以大范围进行，造成了数据的片面性，同时也为产量带来了影响；接触性通常意味着测量必须人工进行，不能够做到实时动态监测，测量过程中也可能对植物生长造成影响。

植物无损测量技术具有非接触式和非破坏性的特点，适应未来农业生产中对植物大范围实时动态监测的需要，在植物学、农学、生态学、林学等众多的农业领域有着广泛的应用前景。

1.1.2 植物无损测量的方法

自 20 世纪 90 年代以来，农作物遥感监测便成为遥感技术应用的一个重要方面。但基于卫星空间监测的遥感技术常常受到气候条件影响而导致精度不高，由于其运行周期问题也往往不能做到实时监测。更为重要的问题在于，遥感监测只能适用于宏观的大面积农作物检测，而无法对小面积中的单个或数个植物进行快速、准确、全面地测量与分析。再者，遥感监测通常需要地面上的校准工作，这些校准工作又必须通过一些其他的有损或无损测量方法实现。

机器视觉是利用计算机模拟宏观视觉功能的新兴技术，它能够利用图像重建现实世界的物理对象。基于机器视觉的植物无损测量可以有效地避免遥感测量的各种弊端，已被证实在农业生产中有着广阔的前景。传统的基于机器视觉的植物无损测量主要通过 CCD 摄像机、激光扫描仪、双目设备等实现。

CCD 摄像机能够获取植物的二维彩色图像。通过采用不同敏感波段的摄像机，还能够反映出植物的一些肉眼不可见的生理状况。这类设备已经广泛用于机器视觉技术中，无论是设备本身还是相关数据处理算法，都已经过多年的发展。

其缺点在于只能获得二维的平面数据，不能够全面地反映植物的三维结构信息；由于植物本身外观的多样性和背景环境的复杂性，在图像分割、识别等方面也常常会出现问题。

激光扫描仪、双目设备则能够获取具有深度数据的三维图像。前者通过激光飞行时间计算目标与设备间的距离，而后者则以三角测量原理计算深度。从这两类设备可以得到较为完整的深度数据，这使得对植物进行三维重建成为可能，更为全面的结构性参数测量也就具备了数据来源。双目设备获取的深度图像通常是分片而稀疏的，而激光扫描仪的则较密但分辨率低且多噪声。此外，这两类设备价格通常较为昂贵，在目前对于私人用户来说难以承担。

1.2 Kinect 简介

1.2.1 自然交互设备 NID

自然用户界面 NUI 是指一类无形的用户界面。“自然”一词是相对图形用户界面 GUI 而言的，GUI 要求用户必须先学习软件开发者预先设置好的操作，而 NUI 则只需要人们以最自然的交流方式与机器互动。微软开发的 Kinect 就是其中的典型代表，甚至被誉为继键盘、鼠标、触控后的人机交互方式的第四次革命。

Kinect 是微软公司 2010 年推出的 XBOX360 游戏机的体感控制器，它允许人们不用触摸控制器即可进行游戏。它能够提供实时的彩色图像和深度图像，默认以 30 帧每秒的速度提供分辨率为 640×480 的彩色图像和 320×240 的深度图像数据。对于 Kinect for Windows 传感器而言，它的有效视距为 0.8 米到 4.0 米之间，最佳距离在 2.5 米左右，非常适合近距离的深度数据采集[1]。同时，Kinect 作为一款消费性电子产品，具有着较低的价格和功耗，可以有效降低各种植物无损测量系统的构建成本。

1.2.2 Kinect 传感原理

Kinect 传感器主要通过三个部分获取图像数据，即其前面板上的红外投影机（左）、红外摄像头（右）和彩色摄像头（中），见图 1。

对于彩色图像的获取，Kinect 与一般的 CCD 摄像头没有区别。这里不对其工作原理进行详述。

图 1. Kinect 外观图

图 2. Kinect 发出的激光散斑

对于深度图像的获取，Kinect 并没有采取常用的飞行时间或结构光原理，而是采用一种新型的光编码技术。激光投影机中发出的普通的红外激光束，通过不均匀介质产生随机衍射斑点，称为激光散斑，见图 2。这些斑点在距离发射器不同位置的平面上具有不同的图案，对每个位置进行预先标定后，在使用时，红外摄像头拍摄到的散斑图像与存储在设备内的不同距离的参考图进行相关运算，即可确定图像上每个点的深度数据[1]。

Kinect 的彩色摄像头和红外摄像头之间存在着不可避免的位置偏离，因此两者获得的图像也不可能完全重合。这一问题需要依靠图像间的配准过程进行解决。通过上述过程，即可获得像素间具有对应关系的彩色图像和深度图像。

2 国内外研究现状

早在上个世纪，就已经有许多科学家探索了利用机器视觉进行植物生长信息测量的可能性。Seginer 等利用机器视觉跟踪番茄植株的叶尖以检测叶片枯萎情况[2]。Shimizu 和 Heins 利用基于机器视觉的系统测量植物茎的伸长[3]。Lin 等进行了蔬菜幼苗的 3D 建模并以很高的准确率估计了它们的叶面积[4]。Kacira 和 Ling 开发了一种自动化的非接触传感系统，它可以持续地监测植物的健康和生长状况[5]。Kacira 等构建了一个早期检测植物水分胁迫的测量系统[6]。Chien 和 Ling 提出了利用立体图像测量植物生长的方法[7]。

Kinect 传感器在 2010 年 11 月在北美上市，而直至 2011 年 6 月微软官方才发布了可供大众开发者利用的软件开发工具包。此前 Kinect 传感器主要用于视频游戏领域，后来才大量出现了 Kinect 应用在其他领域的报道。将 Kinect 用于植物无损测量的研究同样始于 2011 年，短短两年时间，已经有不少研究成果见诸学术期刊，尽管因为时间较短还未能被广泛用于生产实践。

2.1 国外研究现状

在基于 Kinect 的植物无损测量方面的研究，最早的尝试者来自北欧。

瑞典和西班牙的 Wallenberg 等提出了利用 Kinect 获取的彩色图像和深度图像数据进行植物叶片图像分割的方法[8]。此研究意在利用数据融合的方法，从整个植株图像上划分出单个叶片。研究并没有采用当时并不成熟的官方 Kinect SDK 和驱动，而是使用了的第三方的驱动 libfreenect 库，这使得研究中需要进行一些额外的彩色图像和深度图像的配准工作。研究采用了通道编码技术进行数据融合以及超顺磁聚类方法进行图像分割。研究针对六组图像进行了测试，在进行低通滤波以抑制噪声后，通过分别对彩色数据、深度数据和融合数据进行聚类得到三组图像分割结果。研究同时手工绘制了标准分割图像并定义了用于表征分割效果的一致分数，进而指出融合数据的分割效果最好，深度数据其次，彩色数据最差。

日本的 Yamamoto 等在 2012 年的美国农业工程师协会年会上发表演讲，提出了利用 Kinect 在一种循环式可动工作台系统中监测草莓植株生长信息的方法[9]。此研究目的是通过 Kinect 获取草莓植株生长信息，并依此绘制出系统中的生长状况图。在这种种植系统中，工作台循环式移动从而每天两次经过获取点并得到浇灌，这使得只需将 Kinect 固定在获取点周围即可监测整个系统中的所有草莓植株的生长状况。由于 Kinect 工作时不宜阳光直射，研究仔细选取的试验时间和环境。此研究使用了 OpenNI 作为驱动以及 MvTech 的 Halcon 软件用于数字图像处理。由于 Kinect 的视域不足以一次性拍摄整个工作台，因此研究采用了灰阶匹配算法拼合每个工作台的图像，之后通过划定一定深度范围并根据颜色剔除周围物体而提取出叶片部分图像。随后为图像划分出数个小的感兴趣区域，并在每个感兴趣区域上执行相关算法以求得株高、宽度、体积、叶面积、叶倾角和颜色等信息。研究显示通过 Kinect 数据导出的测量结果与手工测量结果均具有较高的相关性，但 Kinect 测量结果往往偏小，尤其在叶面积方面仅为手工测量结果的一半左右，怀疑由于算法本身存在不足导致。最后根据测量结果绘制了生长状况图，并得出不同品种草莓生长情况存在差异的结论。

美国的 Azzari 等尝试将 Kinect 用于植物的冠层特性描述[10]。此研究意在显示 Kinect 在包括基圆直径、高度、体积等植物结构参数现场测量方面的有用性。研究的各种处理通过 OpenNI 和 PCL 在 C++环境中实现，首先通过 PCL OpenNI 抓取器获取点云，该点云仅保留空间信息而并未利用颜色信息，接着进行噪声抑制、个体植物隔离等前处理，最后计算三方向极差和植物体积。研究通过两种布局来获取植物点云，一是仅能获取俯视不完整点云的“塔模式”，二是能获取相对完整点云的“多角度模式”。针对这两种布局获取的点云的不同特征，分别采用凸包算法和凹包算法计算植物体积。研究首先进行了预先测试，并发现环境光强太强时，Kinect 深度测量结果质量会下降，以及对于叶片较小、背景复杂的植物，配准过程不太完美。现场测量中，基圆直径和高度测量都取得了很好的效果，体积方面则认为椭圆锥能够提供测试植物的形状最优拟合。此外，通过对数函数还导出了植物基圆直径、高度、体积与干生物量之间的异速生长关系，相关性很高。

2.2 国内研究现状

由于 Kinect for Windows 传感器 2012 年才在中国正式上市，国内对于 Kinect 在植物无损测量方面的研究相对落后。但 Kinect 作为新兴技术，对于其应用方面各国间差距并不大，国内也同样涌现出一些具有启示意义的研究成果。

华南农业大学的江晓庆等提出了利用 Kinect 获取植物深度图像及点云的过程[11]。研究以 SensorKinect 作为驱动，OpenNI 和 NITE 作为编程接口，C++语言作为平台，OpenCV 库作为辅助开发工具。研究指出，OpenNI 框架提供了完整的深度图像获取方法，包括对彩色摄像机和红外摄像机的控制和参数设置、两者间的配准和将二维图像数据转化为三维点云世界坐标。研究评价了利用 Kinect 获取植物深度图像的优势，但并未在获取点云数据后做深入研究。

北京师范大学和桂林科技大学的 Chen Yiming 等探索了利用 Kinect 无损测量玉米单个叶片叶面积和叶倾角的方法[12]。研究认为玉米叶片复杂程度中等，即不能被平面有效表达，又具有一些方便三维重建的特征。研究采用 Kinect for Windows SDK 以及 C#编程环境，在利用深度图像得到点云后，仅保留一定深度范围内的点，此时可得到包含玉米叶片大部分表面的点云，且不同叶片间是相互分离的。三维重建工作则由 OpenGL 库和 C++编程环境以不规则三角网 TIN 的方法完成，同时可获得叶片面积和倾角数据。研究还比较了 Kinect 测量结果和手

工测量结果，总体来说 Kinect 测量结果偏低，原因包括叶片遮挡、Kinect 深度数据在物体边界处的缝隙等。

3 研究方向及进展

当前基于 Kinect 的植物无损测量的相关研究主要集中于两个方面：一是对植株整体的结构性参数进行测量，二是对植株的部分（主要是叶片）的参数进行测量。这两个方面之间又存在着交叉。具体可以分为下述研究方向。

3.1 三维重建

植物的冠层结构直接影响太阳光在地表附近的传输状况，是控制初级生产、蒸发蒸腾和植物竞争的主要因素。许多生理生态模型的建立和验证依赖于对植物冠层结构的充分认识，而其中一种途径就是将真正的植物冠层结构以三维模型的方式构建出来。

三维重建需要比较完整的点云数据的支持，由于植物叶片间的相互遮挡，单角度的深度图像不能够提供用于三维重建的完整数据。因此，从多个角度获取植物的深度图像，并通过一定的算法进行对齐、融合，从而获得完整的点云，这是在三维重建前所必须进行的工作。对于多角度深度数据的采集，可选的方案有直接使用多个设备进行拍摄和将一个设备绕植物旋转拍摄两种。

多角度图像的配准过程是植物结构三维重建的一个难点。对表面规则的实体进行多角度深度图像采集、配准和三维重建已经形成了相对完善的算法，而植物的结构较为复杂，叶片之间又具有相似的模式，在配准过程中常常出现错误，这一问题在植物叶片较小、较多时尤为明显。

Azzari 等通过使 Kinect 围绕植物旋转 360 度，同时记录下点云数据流。一个 360 度的完整扫描将会产生 2000 幅深度图像，占用约 2GB。之后通过 Kinect Fusion 技术自动配准流化点云数据，合并为单个点云[10]，见图 3。由于植物叶片间的相互遮蔽，即便进行 X-Y 平面上的 360 度扫描，仍然会有一些叶片的点云不完整。此外，这项工作的数据量十分巨大，需要占用大量的磁盘空间，且需要专用的图形处理单元进行运算。

也有一些研究仅对植株的某一部分进行三维重建，比如叶片。

Chen 等在提取出单个叶片的点云数据后，利用 OpenGL 提供的不规则三角网进行曲面拟合，得到单个玉米叶片的三维模型[12]，见图 4。因为其点云数据的采集来自于俯视，实际上没有叶片背面的信息，因此拟合的曲面也只有上表面具有实际意义。

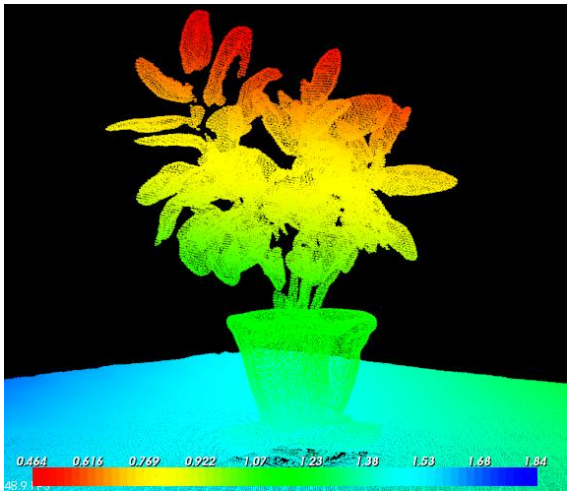


图 3. 多角度深度图像合成的单个点云

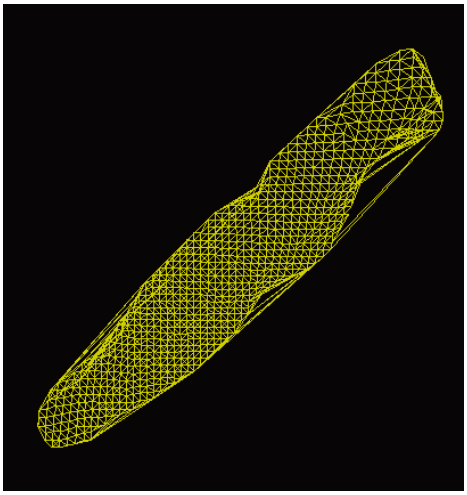


图 4.玉米叶片三维重建模型

3.2 尺寸测量

尺寸测量是 Kinect 的基础应用之一，早在 Kinect 推出之初就有开发商开发了能够测量人体身高的软件。由于当前相关软件包已经日趋完善，用户可以直接将深度图像数据转化为真实世界的三维坐标，这为利用 Kinect 进行尺寸测量提供了方便。

基于 Kinect 的植物尺寸测量主要包含两个方面，即高度测量和宽度（或基圆直径）测量。测量方法主要通过寻找指定点云的点坐标极值来计算相关尺寸。对于俯视来说，植物高度为深度图像中的背景深度值（地面）减去深度最小值（植物顶端），而植物宽度（基圆直径）则植物部分在 X 轴、Y 轴上的跨度的平均值。对于侧视来说，植物高度为植物部分的 Y 轴跨度，而植物宽度则为 X 轴跨度和深度范围的平均值。

Yamamoto 等以俯视测量了工作台上草莓植株群体的高度和宽度。研究首先在拼合后的全工作台深度图像中选取数个感兴趣区域，再对感兴趣区域进行有关运算。植株高度平均值或最大值直接由深度获得，宽度平均值或最大值则使用植物区域的费雷特直径[9]。其结果表明 Kinect 测量结果和手工测量结果相关度高，

回归模型的决定系数对高度和宽度分别达到 0.96 和 0.83；但回归系数则为 0.90 和 0.81，说明 Kinect 测量结果普遍偏小。

Azzari 等以俯视测量了在野外两种植物的高度和基圆直径[10]。测量结果显示测量结果的相对优劣与植物种类具有相关关系，但总体来说，均与手工测量结果较为吻合。对于两种植物的高度和基圆直径的测量，从恒等线计算的标准化均方根误差在 2.7%到 19.1%的范围内波动。

3.3 叶面积和叶倾角测量

叶面积和叶倾角也是植物生长的重要参数，它们能够直观地反映植株的生长状况。利用 Kinect 对单个叶片的面积和倾角测量相对准确；而对整个植株或植株群的叶面积和叶倾角进行测量多以俯视进行，但由于叶片间的相互遮挡，很难得到准确的数据。

Yamamoto 等在感兴趣区域中每隔数个像素进行采样，将采样点转换为世界坐标后，通过相邻采样点构建向量并计算法向量，最后根据法向量的模和角度计算面积和倾角[9]。结果显示，虽然叶面积测量结果与手工测量结果间的决定系数达到了 0.8 以上，但回归系数却只有 0.5 左右，误差较大。

Chen 等在对玉米叶片进行三维重建后，利用得到的三角网格可以方便的得到面积和法向量[12]。其结果显示，Kinect 测量的面积通常比人工测量结果偏小。

3.4 体积估算

体积估算可以通过前述的三维重建、尺寸测量和叶面积测量结果得到。有许多不同的方式用于体积估算，如包算法、实体拟合等。

Yamamoto 等将体积估算简单化，直接使用植物区域的面积乘以植株的平均高度得到[9]。因为此研究的目标对象是工作台上的草莓植株群体，因此这样的估算是合理的。

Azzari 等则利用包算法和实体拟合两种途径估算单个植株的体积[10]。研究中“塔模式”获取的点云仅有俯视，是不完整的，只能使用凸包进行体积估算，但这种估算方式不能正确满足植株冠层大、基部小的特征，因此估算并不准确，见图 5。“多角度模式”获取的点云是完整点云，可以使用更为精确的凹包算法进行估算，这种算法可以准确贴合植物表面，其体积测量准确程度甚至超过手工测量，

见图 6。研究同时将 Kinect 测量结果和多钟实体拟合结果进行比较，得出了最有拟合的实体模型。

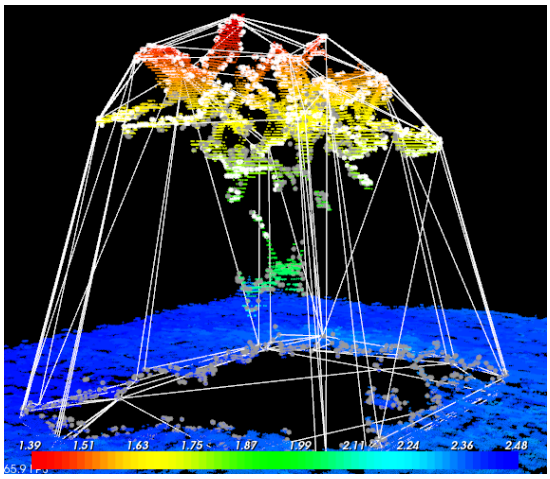


图 5. 凸包算法体积估算

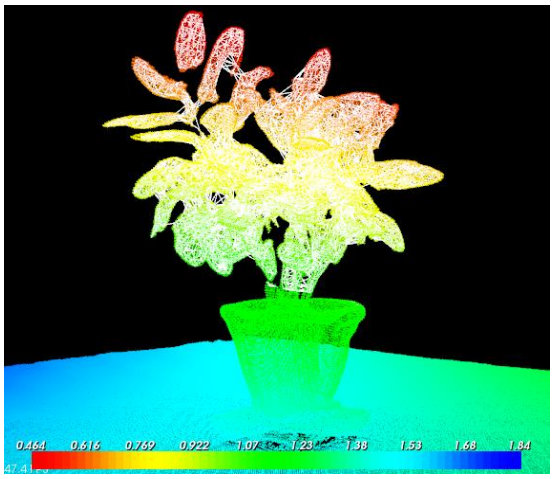


图 6. 凹包算法体积估算

3.5 叶片分割

叶片分割自从机器视觉技术引进农业领域以来一直是一个重要的研究领域。只有通过合适的分割，得到叶片单独的图像，才能进行许多后续分析处理。通常认为人眼进行叶片分割时，叶片在三维空间的分散状况比颜色上的边缘能够提供更多信息。因此利用深度图像进行分割相比利用彩色图像分割更为有效。还有许多研究综合了颜色和深度信息，以追求更好的叶片分割效果。

Wallenberg 等利用通道编码技术将彩色信息和深度信息进行融合，再利用超顺磁聚类算法对图像进行无监督聚类，通过设定合适的参数确定聚类的停止时机，最终得到的聚类结果即成为叶片分块[8]。其结果能够将每个不同的叶片分为不同的区块。研究同时也分别单独利用彩色信息和深度信息重复聚类，将三种方法的结果进行比较。结果显示融合数据的分割结果更接近人工分割结果，产生的错误比较少；而深度数据和彩色数据的分割结果则不尽人意，尤其是彩色数据结果更差。见图 7。

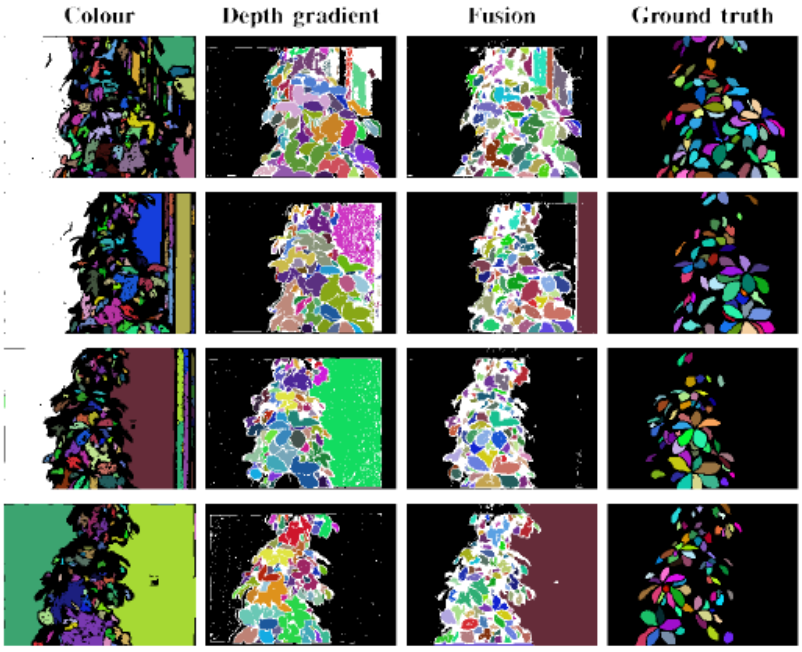


图 7. 利用颜色、深度和融合数据对植物图像进行分割的结果

Chen 等以俯视获取玉米植株的点云后，以较为简便的方法进行叶片分割，即划定一定的深度范围，此范围内能保留叶片表面大部分的点，但删去了靠近茎部的点，从而各个叶片之间产生了明显的间隙，可通过此间隙进行分割[12]，见图 8。

Yamamoto 等并没有将叶片分割精确到每个叶片，而是从包含工作台及周边设施的图像中将叶片部分提取出来。其做法为在深度图像中提取高于背景一定距离的部分，在彩色图像中利用颜色特征排除周边设施的部分，两者取交集即可得到仅包含叶片部分的图像[9]，见图 9。

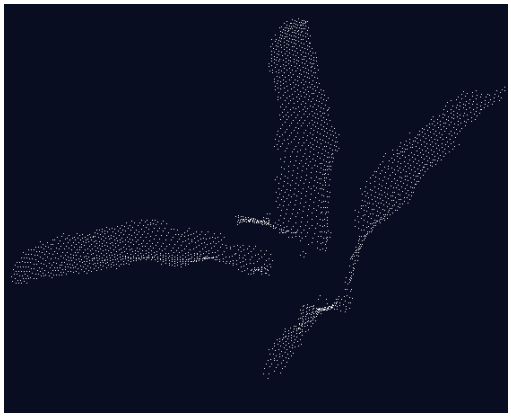


图 8. 玉米叶片的分割

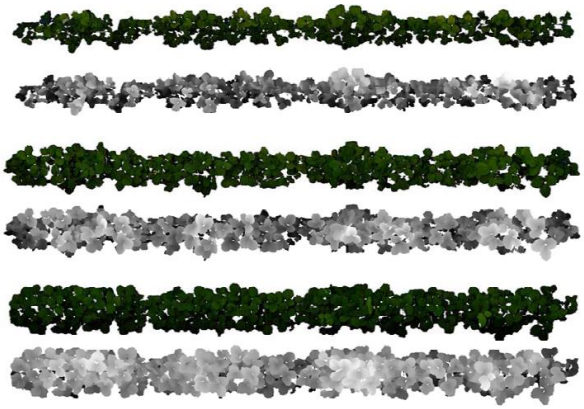


图 9. 草莓植株叶片的分割

4 存在问题

虽然基于 Kinect 进行植物无损测量是一项极具前景的新兴技术，但仍然存在一些问题需要正视并解决。其中包括 Kinect 设备本身的缺陷，也包括一些当前未能有效解决的技术问题。

各种测量方法需要更大范围的验证。当前有关此领域的研究还比较零星，并没有形成一定的规模，因此各类发表的研究成果也只是针对小样本进行了验证，并没有在更大的范围内验证算法的精确性、鲁棒性等。然而大范围的验证对于技术的推广和应用是必须的，这项工作需要耗费大量人力和时间，可能会在此后的数年内逐步完善。

各类算法的准确性方面需要完善。由前述可知，当前的各种 Kinect 测量结果虽然与手工测量结果之间相关性很强，但往往都是有偏差的。这些偏差一部分来自于植物结构的复杂性，一部分则来自于算法本身原理方面存在的问题。因此完善算法必须极力填补后者的漏洞，并使用一些间接方法减少前者的影响。

多角度数据配准的稳定性需要加强。多角度数据的正确配准是形成完整点云、进行三维重构和后续分析的基础。但当前已经发现传统的针对较为规则实体的配准算法并不总适用于具有复杂结构的植物，因此加强配准算法的稳定性就成为一个研究重点。

各种测量算法仍处在比较原始的阶段，未经过优化，因而难以满足实时测量的需要。这点在植物三维重建方面显得尤为突出，因为这项任务所设计的数据量和计算量十分巨大，如果相应的算法优化程度不够，则很难实时得到结果。

Kinect 的深度测量结果在日光下质量较差。已经有许多研究证实，在强日光下 Kinect 发射的激光散斑会受到影响，直接干扰了深度测量结果，造成最后得到的深度图像上出现大量盲区[10]，见图 10。Kinect 的这一特性使之在实地测量方面的应用受到限制，解决方法可能在于人工构建一定的测量环境或是升级 Kinect 的硬件。

Kinect 的测量范围也限制了其应用。当前版本的 Kinect 能够测量的深度范围较窄，一旦超出推荐的测量范围就会出现很大的盲区，使得测量数据变得无效，这限制了测量系统的构建和布局。

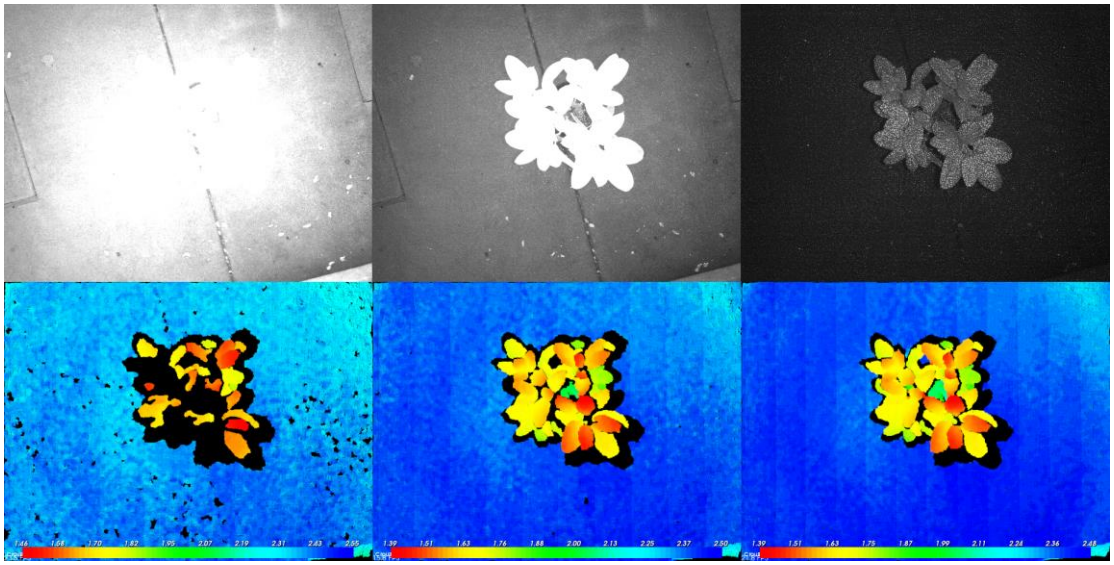


图 10. Kinect 在午后、傍晚和夜间获取的红外原始图像和深度图像

此外，Kinect 的深度图像在物体边缘处往往会出现一个较窄的盲区，也从一定程度上影响了各类测量结果，见图 10 深度图像中的黑色区域。

5 研究展望

基于 Kinect 的植物无损测量还处于方兴未艾的阶段，未来的研究方向可从两个方向入手：一是继续完善当前的各类研究，着力于提高各种算法的准确性、稳定性和速度；二是不断探索新的领域，将 Kinect 深度测量技术应用到植物无损测量的其他方面中去。

当前对植物进行三维重建得到植物三维模型后，仅进行了一些例如尺寸测量、体积估算的表层分析，而并没有进行一些深层次的分析，例如分支结构、各个部分的形状拟合等等。这一领域在当前仍然比较空白，有大量的研究工作可以进行。通过这些深层次的分析，可以不断改进当前的植物形态模型和一些生理生态模型，从而应用到植物生长仿真当中。

由于植物本身结构的复杂性，单一视角的深度图像往往很难得出各种参数的准确测量值，因此构建一类多设备多角度测量系统就显得很有必要。通过多组数据的融合，减少深度数据的多义性，有利于得到更为准确的测量结果。同时系统的构建也有利于克服 Kinect 设备本身在光源要求、视野范围等方面的缺陷。

因为当前相关研究刚刚起步，仍有许多可能性未被尝试，这些可能性中也许蕴藏的新的重大发现。例如，当前的叶片分割技术多采用简单的深度分割或是较为基础的彩色、深度融合数据分割。一些色彩空间的转换或是几何坐标的转换可以应用到数据当中，一些经典的或新的统计分析方法也可以用来进行分析，通过不同的组合很可能能够得到进行叶片分割的更优方案。

当然，前人的许多通过其他方式获取深度图像并进行分析的研究也可以用作参照。

Alenya 等研究了通过深度数据辅助的基于色彩的叶片分割方法[13]。研究通过超顺磁聚类方法对颜色数据进行聚类，并利用平面和曲面两种模型分步进行拟合，以最小化均方根距离决定合适的拟合模型以及聚类终点，再结合深度图像进行区域合并和区域生长，最终得到较为准确的精确单个叶片的分割结果。研究中的深度图像来自与基于飞行时间的激光扫描仪，使用此种分割方法的原因也是基于对深度图像分辨率远小于彩色图像的考量。研究还根据分割出的叶片区域决定感兴趣的叶片，并同时计算出新的相机位置，随后引导安装在机械手末端的相机到达此位置，对感兴趣叶片拍摄更为精细的图像，见图 11。相关的算法和应用对本领域的研究同样具有启示意义。

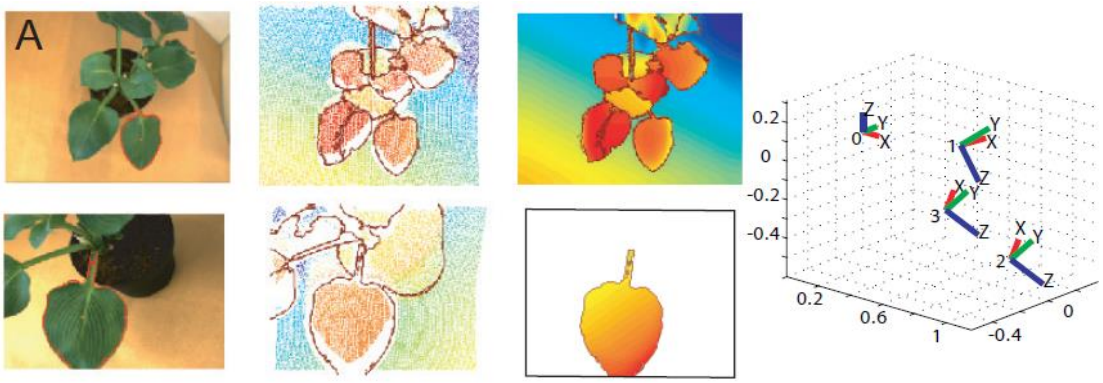


图 11. 分割叶片区域，选定感兴趣叶片，计算新的相机位置，并获取更近的图像

Chene 等研究了通过深度图像对整株植物进行表形的方法[14]。研究提出了一种从俯视深度图像进行图像分割的原创算法，并探索了多种应用。一项应用测量植物叶片的曲率，针对丝兰叶片的特殊形状，研究使用一对向量表示弯折的叶片两部分在空间中的不同朝向，见图 12。另一项应用则测量了玫瑰花丛的叶片方位角，见图 13。未来的研究同样可以探索这些应用在 Kinect 获取的深度图像中的实现。

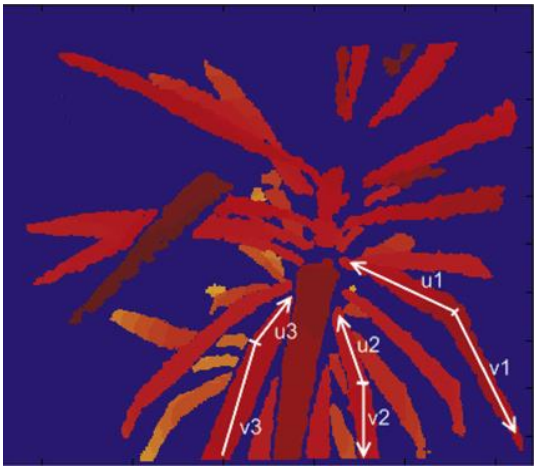


图 12. 计算丝兰叶片的曲率

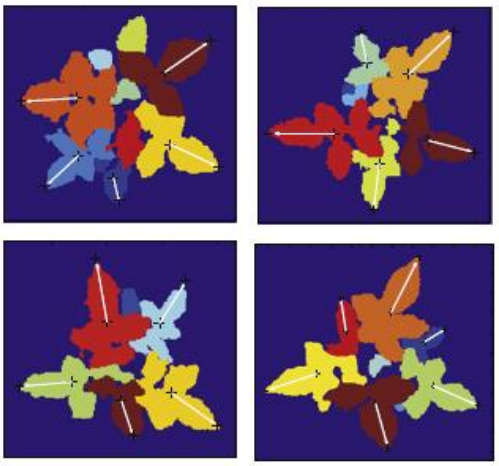


图 13. 计算玫瑰花丛的叶片方位角

Song 等研究了一种结合双目视觉和飞行时间深度图像的叶片面积无损自动测量方法[15]。通过两种设备数据的融合，弥补了相互的缺点，并以一种全局优化策略进行数据平滑并保留间隙，最后依据边缘颜色和间隙判别叶片边缘，从而得到叶片面积，见图 14。Kinect 作为一款新的立体视觉设备同样可以与其他传感器相互配合，弥补数据的不足，得到更为精确的结果。

王奎等提出了一种 Kinect 深度图像快速修复算法，此算法可以有效地填补原始深度图中由于遮挡产生的空洞，以获取整体平滑、边缘清晰的深度图像[16]，见图 15。类似的 Kinect 深度图像处理和分析算法也可以应用到本领域的研究中来。

图 14. 融合数据叶片分割

图 15. Kinect 深度图像修复结果

参考文献

[1] 余涛. Kinect 应用开发实战：用最自然的方式与机器对话 [M]. 北京: 机械工业出版社, 2012.

[2] Seginer I, Elster R T, Goodrum J W, et al. Plant wilt detection by computer-vision tracking of leaf tips [J]. Transactions of the ASAE, 1992, 35(5): 1563-1567.

[3] Shimizu H, Heins R D. Computer-vision-based system for plant growth analysis [J]. Transactions of the ASAE, 1995, 38(3): 959-964.

[4] Lin T T, Liao W C, Chien C F. 3D graphical modeling of vegetable seedlings based on a stereo machine vision system [C]//ASAE Meeting Paper. 2001 (01-3137).

[5] Kacira M, Ling P P. Design and development of an automated and non-contact sensing system for continuous monitoring of plant health and growth [J]. Transactions of the ASAE, 2000, 44(4): 989-996.

[6] Kacira M, Ling P P, Short T H. Machine vision extracted plant movement for early detection of plant water stress [J]. Transactions of the ASAE, 2001, 45(4): 1147-1153.

[7] Chien C F, Lin T T. Non-destructive growth measurement of selected vegetable seedlings using orthogonal images [J]. Transactions of the ASAE, 2005, 48(5): 1953-1961.

[8] Wallenberg M, Felsberg M, Forssén P E, et al. Leaf Segmentation using the Kinect [C]//Proceedings of SSBA 2011 Symposium on Image Analysis. 2011.

[9] S. Yamamoto, S. Hayashi, S. Saito, et al. Measurement of Growth Information of a Strawberry Plant Using a Natural Interaction Device [C]//2012 ASABE Annual International Meeting. ASABE, 2012.

[10] Azzari G, Goulden M L, Rusu R B. Rapid Characterization of Vegetation Structure with a Microsoft Kinect Sensor [J]. Sensors, 2013, 13(2): 2384-2398.

[11] 江晓庆, 肖德琴, 张波, 等. 基于 Kinect 的农作物长势深度图像实时获取算法 [J]. 广东农业科学, 2012, 39(23): 195-199.

[12] Chen Y, Zhang W, Yan K, et al. Extracting corn geometric structural parameters using Kinect [C]//Geoscience and Remote Sensing Symposium (IGARSS), 2012 IEEE International. IEEE, 2012: 6673-6676.

[13] Alenya G, Dellen B, Torras C. 3D modelling of leaves from color and ToF data for robotized plant measuring [C]//Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011: 3408-3414.

[14] Chéné Y, Rousseau D, Lucidarme P, et al. On the use of depth camera for 3D phenotyping of entire plants [J]. Computers and Electronics in Agriculture, 2012, 82: 122-127.

[15] Song Y, Glasbey C A, Polder G. Non-destructive Automatic Leaf Area Measurements by Combining Stereo and Time-of-Flight Images [J].

[16] 王奎, 安平, 张兆杨, 等. Kinect 深度图像快速修复算法 [J]. 上海大学学报 (自然科学版）, 2012, 18(5).

开题报告

1 研究概况

对植物一些结构性生长参数的测量在科学研究和农业生产中都具有十分重要的意义。测量方法则主要分为接触式、破坏性的测量和非接触式、非破坏性的测量两种，而在后者中，基于机器视觉的植物无损测量展现出极大的发展前景。Kinect 作为自然交互设备的代表，能够以低廉的成本提供配准的彩色图像和深度图像，已被证实可应用于植物无损测量领域。

1.1 植物无损测量

1.1.1 植物无损测量的意义

随着信息技术的迅猛发展，许多传统行业也呈现出崭新的面貌。数字农业作为农业技术与信息技术的交叉领域，已经在当下呈现出了良好的前景，称为现代农业发展的新方向之一。数字农业不仅将继承了地理学、农学、生态学、植物生理学、土壤学等基础学科在农业生产中的指导思想，还结合了全球定位系统、地理信息系统、遥感技术以及计算机技术、通信网络技术、自动化技术等高新技术，从而实现在农业生产过程中对农作物及其相关环境的全方位实时监测与控制，辅以对农业生产中的现象、过程进行模拟等手段，达到优化农业资源配置、降低生产成本、改善生态环境、提升产品品质的目的。

在生产过程中对植物的各种生长参数（本文主要研究结构性参数）进行测量正是数字农业的一个重要的研究领域，在生产实践中具有着重要的意义。植物的生长是一个极为复杂的过程，不仅与植物内部的遗传因素密切相关，同时还极大程度地受到环境因素和随机因素的影响，难以用简单的模型进行描述，其定量化表示也是长期以来的研究热点。生长参数，例如株高、株宽、叶面积等，通常可以直观地反映植物的生长情况，这就为进一步研究提供了基础。这些数据可被广泛应用于生长监测、科学建模、品质控制等诸多方面。

植物的生长参数测量在早期以接触式和破坏性测量为主，即通过将植物部分或整体分离，利用直尺及其他手工测量工具对各种参数进行测量。这类方法虽然在部分参数的测量方面能够得到较为精确的结果，但其弊端也是显而易见的。由于其破坏性，测量难以大范围进行，造成了数据的片面性，同时也为产量带来了影响；接触性通常意味着测量必须人工进行，不能够做到实时动态监测，测量过程中也可能对植物生长造成影响。

植物无损测量技术具有非接触式和非破坏性的特点，适应未来农业生产中对植物大范围实时动态监测的需要，在植物学、农学、生态学、林学等众多的农业领域有着广泛的应用前景。

1.1.2 植物无损测量的方法

自 20 世纪 90 年代以来，农作物遥感监测便成为遥感技术应用的一个重要方面。但基于卫星空间监测的遥感技术常常受到气候条件影响而导致精度不高，由于其运行周期问题也往往不能做到实时监测。更为重要的问题在于，遥感监测只能适用于宏观的大面积农作物检测，而无法对小面积中的单个或数个植物进行快速、准确、全面地测量与分析。再者，遥感监测通常需要地面上的校准工作，这些校准工作又必须通过一些其他的有损或无损测量方法实现。

机器视觉是利用计算机模拟宏观视觉功能的新兴技术，它能够利用图像重建现实世界的物理对象。基于机器视觉的植物无损测量可以有效地避免遥感测量的各种弊端，已被证实在农业生产中有着广阔的前景。传统的基于机器视觉的植物无损测量主要通过 CCD 摄像机、激光扫描仪、双目设备等实现。

CCD 摄像机能够获取植物的二维彩色图像。通过采用不同敏感波段的摄像机，还能够反映出植物的一些肉眼不可见的生理状况。这类设备已经广泛用于机器视觉技术中，无论是设备本身还是相关数据处理算法，都已经过多年的发展。其缺点在于只能获得二维的平面数据，不能够全面地反映植物的三维结构信息；由于植物本身外观的多样性和背景环境的复杂性，在图像分割、识别等方面也常常会出现问题。

激光扫描仪、双目设备则能够获取具有深度数据的三维图像。前者通过激光飞行时间计算目标与设备间的距离，而后者则以三角测量原理计算深度。从这两类设备可以得到较为完整的深度数据，这使得对植物进行三维重建成为可能，更为全面的结构性参数测量也就具备了数据来源。双目设备获取的深度图像通常是

分片而稀疏的，而激光扫描仪的则较密但分辨率低且多噪声。此外，这两类设备价格通常较为昂贵，在目前对于私人用户来说难以承担。

1.2 Kinect 简介

1.2.1 自然交互设备 NID

自然用户界面 NUI 是指一类无形的用户界面。“自然”一词是相对图形用户界面 GUI 而言的，GUI 要求用户必须先学习软件开发者预先设置好的操作，而 NUI 则只需要人们以最自然的交流方式与机器互动。微软开发的 Kinect 就是其中的典型代表，甚至被誉为继键盘、鼠标、触控后的人机交互方式的第四次革命。

Kinect 是微软公司 2010 年推出的 XBOX360 游戏机的体感控制器，它允许人们不用触摸控制器即可进行游戏。它能够提供实时的彩色图像和深度图像，默认以 30 帧每秒的速度提供分辨率为 640×480 的彩色图像和 320×240 的深度图像数据。对于 Kinect for Windows 传感器而言，它的有效视距为 0.8 米到 4.0 米之间，最佳距离在 2.5 米左右，非常适合近距离的深度数据采集。同时，Kinect 作为一款消费性电子产品，具有着较低的价格和功耗，可以有效降低各种植物无损测量系统的构建成本。

1.2.2 Kinect 传感原理

Kinect 传感器主要通过三个部分获取图像数据，即其前面板上的红外投影机（左）、红外摄像头（右）和彩色摄像头（中）。

对于彩色图像的获取，Kinect 与一般的 CCD 摄像头没有区别。这里不对其工作原理进行详述。

对于深度图像的获取，Kinect 并没有采取常用的飞行时间或结构光原理，而是采用一种新型的光编码技术。激光发射器中发出的普通的红外激光束，通过不均匀介质产生随机衍射斑点，称为激光散斑。这些斑点在距离发射器不同位置的平面上具有不同的图案，对每个位置进行预先标定后，在使用时，红外摄像机拍摄到的散斑图像与存储在设备内的不同距离的参考图进行相关运算，即可确定图像上每个点的深度数据。

Kinect 的彩色摄像机和红外摄像机之间存在着不可避免的位置偏离，因此两者获得的图像也不可能完全重合。这一问题需要依靠图像间的配准过程进行解决。通过上述过程，即可获得像素间具有对应关系的彩色图像和深度图像。

1.3 研究意义和目的

1.3.1 研究意义

如上所述，对植物的生长参数进行测量具有重要的意义。这些参数包括了株高、宽度、体积、叶面积、叶倾角、方位角等多项指标，它们被称为植物的肢体语言，对其进行实时监测研究在育种工作、生长状态分析、病虫害判断、生产管理决策等方面都具有十分重要的意义。

然而，根据我国目前农业生产的实际情况，植物生长参数的测量还处于比较初级的阶段，即以表观判断为主，辅以接触式、破坏性的手工测量。手工测量的破坏性、繁琐性决定了它难以用于大规模的实时测量，不符合现代农业发展的基本要求。

在一些科研实验田和现代化的农业生产园区中，也使用了基于机器视觉的现代无损测量技术，通常使用的是普通 CCD 摄像机，根据拍摄的彩色图像进行相关计算从而提取植物的生长参数。其弊端在于，彩色图像所包含的信息比较有限，因而这类机器视觉系统所能测量的参数也有所不足，限制了其应用范围。

立体视觉系统可以提供深度图像，有效地弥补了彩色图像信息量不足的缺点。但各种立体视觉系统均存在各种各样的不足，如采样点稀疏、分辨率不高等问题，同时往往价格昂贵，当前很难应用到生产实践中。在国内也鲜见应用实例。

微软 Kinect 传感器作为一款消费性电子产品，具有低廉的价格，同时能够同时提供质量较好的彩色图像和深度图像，十分适合应用在植物的生长参数测量方面，具有着广阔的应用前景。但由于其发布时间较短，学术界对其在植物无损测量方面的研究还比较少，相关的研究成果不多，也没有形成成熟的理论体系。

因此，尽管具备着无限的可能性，基于 Kinect 深度数据的植物无损测量作为一个新兴的研究领域，需要更多的研究者投入精力进行广泛的探索，以填补当前该领域研究的空缺，才能在未来真正地为农业生产实践服务。

1.3.2 研究目的

本研究并不强调对于 Kinect 深度测量技术的理论研究，而更加着重于 Kinect 在植物无损测量领域的应用，以期为填补该领域当前的研究空缺略尽绵薄之力。

研究的主要思路在于通过搭建一定的测量系统，获取植物形态的深度数据，随后结合各类算法对数据进行滤波、融合、变换等前处理，最后利用深度数据分

析并导出一些重要的植物结构性生长参数，并利用人工测量数据进行校验并比较、分析测量结果。

本研究的目的大致有两个方面：一是在前人研究的基础上，对已被探讨过的可利用 Kinect 测量的植物结构参数进行研究，自行编写算法和程序测量结果，以期能对前人的研究成果做出一定的改良；二是利用通过 Kinect 得到的植物三维数据导出前人研究未被提及的、但对农业生产或科学研究具有实际意义的植物结构参数，以期能够开辟本领域研究的新方面。显然后一个方面对研究提出了更高的要求，需要在对植物各种结构性参数具有充分了解的基础上，具备一定的创新能力和实践能力，在全新的问题上进行探索。对这一方面，研究中可以参考利用其他类型的深度数据对植物进行无损测量、却尚未被应用到 Kinect 上的相关研究，从中获得启发。

2 主要研究内容和技术路线

本研究并不是单纯的算法创新或软件开发，更强调整个测量系统的构建及其实践意义。因此本研究既包括系统规划的理论部分，又包括需要实际搭建测量系统、编写测量程序的实践部分，是一个较为综合性的研究项目。

2.1 主要研究内容

本研究的主要内容包括三个方面，这三个方面具有一定的先后顺序，并且紧密相连。

2.1.1 测量系统的设计与构建

在本测量系统中，Kinect 是主要的测量设备，但不排除使用其他辅助设备以进行数据融合的可能性。测量系统的设计必须依据所选取的研究对象进行合理规划，同时考虑到 Kinect 设备的基本参数和测试特性，还与期望测量的植物参数有关。一方面，仅获取单视角的深度数据可以大大简化测量系统的设计，但也导致点云数据不完整，能够导出的植物参数受到限制；另一方面，若期望获得完整的植物点云，则必须获取多角度的点云数据，使得测试系统中必须包含可移动的部分，让 Kinect 设备能够以不同的视角拍摄植物深度图像，同时也对数据的处理提出了更高的要求。为了获取信息更为完整的植物三维点云，本研究倾向于选择多角度拍摄方式。

2.1.2 三维点云数据的获取

通过搭建的测量系统，对选定的研究对象利用 Kinect 采集深度数据，并将深度数据转换为三维世界坐标点云数据。根据期望得到的点云数据类型的不同，这一部分也包含不同的内容。若要获取完整的植物点云数据，则可能要对测量系统中搭载 Kinect 设备的可动部分进行控制，以获得多角度的深度数据；在得到数据后还需要进行多角度深度图像的配准和融合工作；进而通过数据类型和坐标变换转换为三维点云数据。而仅获取单视角点云数据相对来说工作量则小得多。但两种情况都无法避免的工作是对获取的原始深度图像进行一定前处理，包括平滑、异常点排除等。

2.1.3 从三维点云数据导出植物结构参数

在得到一定形态的植物三维点云后，即可通过各种不同的算法提取植物的各种结构参数。能够提取的植物结构参数与前面所获取的植物三维点云类型有关，完整的三维点云包含植物形态的绝大部分信息，因而理论上可以导出绝大部分植物结构参数。如前所述，本研究的这一部分不仅会关注前人研究中已经提及的结构参数，如株高、宽度、叶面积、叶倾角等，还可能关注一些尚未被前人研究探索的结构参数，如方位角等。这一部分可以说是整个研究的核心部分，它直接决定了研究能够得出怎样的结果。当然，在进行这一部分的时候，还需要人工对许多植物结构参数进行测量，以作为标准数据供 Kinect 测量结果进行校验。

2.2 技术路线

本节主要探讨本研究中软件部分的实现细节。本项目软件部分的主要技术路线为：利用 Kinect 驱动程序建立 Kinect 设备与 PC 之间的通信桥梁；利用 Kinect 编程接口获取从 Kinect 传输过来的深度图像，并进行一定的必要处理；利用其他辅助开发软件包协助从深度图像到三维点云数据再到植物结构参数过程的实现；一些从点云数据中提取植物结构参数的工作仍然需要通过一些算法来实现；整个流程都通过一定的语言以编程方式实现，并需要选择适合的集成开发环境。在上述技术路线的每个部分中，均存在一些可选的方案，这需要在项目进行时进行仔细研究选择，甚至在某些情况下可以结合使用。

2.2.1 开发语言及集成开发环境

目前在 Kinect 的开发方面主要使用两种编程语言，即 C#和 C++。

C#是微软推出的一种基于.NET 框架的、面向对象的高级编程语言。C#由 C 语言和 C++派生而来，继承了其强大的性能，同时又以.NET 框架类库作为基础，拥有类似 Visual Basic 的快速开发能力。C#是微软首推的 Kinect 应用程序开发语言，并在 Kinect 开发者工具箱中提供了大量的例子。C#的优点在于开发快速简便，与微软官方提供的 Kinect SDK 结合紧密，而缺点在于对一些开源类库的支持上存在一定的不足。C#最适宜的集成开发环境是微软开发的 Visual Studio。

C++是一种使用非常广泛的电脑程序设计语言。它是一种静态数据类型检查的，支持多范型的通用程序设计语言。C++支持过程化程序设计、数据抽象化、面向对象程序设计、泛型程序设计、基于原则设计等多种程序设计风格。微软官方的 Kinect SDK 和其他很多开源类库包括 OpenNI、OpenCV 和 PCL 等均完整支持 C++，因此 C++非常适合用于 Kinect 开发，合理地运用各种类库也可以节省大量的开发时间。此外，C++程序的运行效率也比较高，适合大规模的运算。缺点是 C++开发相对高度面向对象的 C#来说略显晦涩。支持 C++的集成开发环境也有很多，包括 Visual Studio 和 QT 等。

2.2.2 Kinect 驱动

Kinect 驱动应当首推微软推出的官方驱动。在当前，无论微软 Kinect SDK 还是新版的 OpenNI 均支持该驱动。

另一个驱动则是一个第三方驱动程序 SensorKinect，该驱动已经于一年前停止更新。这里提到它的原因在于，许多现有的 Kinect 程序在基于 OpenNI 的早期版本编写的，在当时 OpenNI 仅支持 SensorKinect 驱动程序。因此如果需要用到这类程序的话，仍然需要这一种驱动。

2.2.3 Kinect 编程接口

主流的 Kinect 开发工具包括微软的 Kinect for Windows SDK（以及开发者工具包）和开源的 OpenNI SDK。

Kinect for Windows SDK 使开发者能够在运行 Windows 的电脑上，创建使用 Kinect 传感技术支持姿势和语音识别的应用程序。该工具更新速度很快，自最初发布以来已经历过多次重大的版本升级，加入了越来越多的常用功能（包括实时进行三维重建的 Kinect Fusion），使 Kinect 的开发日趋简单。

OpenNI（开放自然交互）是一个多语言、跨平台的框架，它定义了编写应用程序，并利用其自然交互的 API。OpenNI API 由一组可用来编写通用自然交互应用的接口组成。OpenNI 的主要目的是要形成一个标准的 API，来搭建视觉和音频传感器及视觉和音频感知中间件两方面之间通信的桥梁。其优点在于开源性，有着许多能实现各种不同功能的第三方插件。

2.2.4 辅助开发软件包

由于对三维数据的处理过程非常复杂，全部手动编写完成并不现实，因此可能需要使用多钟辅助开发软件包。

PCL（点云库）是一个独立的大型的处理二维/三维图像和点云数据的开源工程，由 Willow Garage 公司开发，包含了许多先进算法，比如滤波、特征估计、表面重建、模型拟合和分割等。它可以与 OpenNI 紧密结合，从而直接从 Kinect 获取点云数据。此外 PCL 还包括 Kinfu，它是 Kinect Fusion 的开源实现，同样可以进行三维重建。

OpenCV（开源计算机视觉库）是一个跨平台的计算机视觉库，它轻量级而且高效——由一系列 C 函数和少量 C++ 类构成，同时提供了 Python、Ruby、MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。

2.2.5 植物结构参数提取算法

这一部分是本研究与其他利用 Kinect 获取点云数据、进行三维重建的项目的最大区别，也是本研究直接农业生产实际意义相连接之处。

前人的研究已经对包括株高、宽度、体积、叶面积、叶倾角等在内的一些结构参数的提取算法进行了一些及其有益的探索，具体可参见文献综述部分。而未被探索的结构参数则需要开发新的算法进行提取。

2.3 可行性分析

2.3.1 环境可行性

本研究主要在室内进行，并不涉及污染环境的行为，因此环境上可行。

2.3.2 经济可行性

微软 Kinect 传感器作为消费性电子产品，价格相对低廉，并不会在经济上造成较大的负担。其他可能的开销包括构建测量系统的材料和加工费用，以及研究对象的种植或购买成本。上述开销均在研究经费可接受的范围，因此经济上可行。

2.3.3 政策可行性

本研究不涉及触犯各类法律或法规的行为，因此政策上可行。

2.3.4 技术可行性

利用 Kinect 进行植物无损测量已被多个研究证实有效，因此本研究在原理上是可行的。项目实施者具有多年编程经验，且具有 Kinect 和计算机图形学相关程序开发的经历，能够承担本研究的任务。本研究的相关参考资料可能的来源包括：书籍，主要是相关技术的指导性书籍，包括《C#与.NET 高级程序设计》、《C++编程思想》、《Kinect 应用开发实战》、《点云库 PCL 学习教程》、《学习 OpenCV》等；论文，包括与利用 Kinect 进行植物无损测量直接相关的论文和有关深度图像的论文等；网络资源，许多技术的指导教程和常见问题的解决方案均可以在网络中找到。综上所述，本研究在技术上可行。

3 研究进度安排及预期结果

为了保证本研究的有序进行，这里给出初步的进度安排，但在实际操作过程中可能存在一定的不一致。

3.1 进度安排

（1）2013 年 12 月 1 日——2014 年 1 月 31 日：相关技术的学习和基本实践，包括对 C++语言的学习、Kinect 应用程序的开发实践、PCL 点云库的学习、OpenCV 计算机视觉库的学习等。因为时间较短，学习中可能有所侧重，主要学习本研究中可能用到的内容。

（2）2014 年 1 月 1 日——1 月 31 日：设计并构建测量系统，包括合理设计测量系统的结构、绘制图纸、购买材料并进行加工等。之前还需要确定本研究中所使用的植物对象，并通过种植或购买等方式获得。期间继续进行相关技术的学习。

（3）2014 年 2 月 1 日——2 月 14 日：利用构建的测量系统获取植物（多角度）深度图像。因为可能遇到设计中没有预想到的问题，因此留出了较长的时间用于修改、调试测量系统。

（4）2014 年 2 月 15 日——2 月 28 日：对获取的植物深度图像进行各种前处理，并转换为三维世界坐标的点云。

（5）2014 年 3 月 1 日——4 月 30 日：利用得到的点云数据提取植物的各种结构参数，包括尝试改进以往研究中使用的提取算法和创造新的算法提取其他参数。期间还需要手工测量一些植物的结构参数，用于 Kinect 测量数据的校验。

（6）2014 年 4 月 16 日——5 月 10 日：对得到的各种类型的数据进行分析，研究出现误差等的可能原因，得出一定的结论。时间允许的话仍可以对一些结构参数提取算法进行修正和完善。本阶段的一部分工作可与上一阶段同步进行。

（7）2014 年 5 月 11 日——5 月 20 日：撰写毕业论文。

（8）2014 年 5 月 21 日——5 月 31 日：结合指导老师的意见，对论文的结构、内容等方面进行修改。

（9）2014 年 6 月上旬：进行毕业论文答辩。

3.2 预期结果

预期结果包括研究论文、基于 Kinect 的植物无损测量系统。

论文中应包含研究背景和文献回顾，测量系统硬件构成细节和参数，主要技术流程和实现过程与方法，提取的结构参数、提取算法以及校验结果，对测量误差的分析、讨论与改进意见，对基于 Kinect 的植物无损测量的优劣势分析，对未来研究的展望以及结论等。

基于 Kinect 的植物无损测量系统应包含硬件和软件两部分。硬件部分要求能够按照需要使 Kinect 以一定的角度（或多个角度）获取目标植物的彩色图像和深度图像；软件部分要求能够对采集到的图像进行各种必要的处理，以得到三维点云数据，并通过该数据提取植物的结构参数。

利用微软 Kinect 传感器的植物结构快速特性描述

George Azzari [1,*]，Michael L. Goulden [1] 和 Radu B. Rusu [2]

[1] 地球系统科学系，加州大学欧文分校，欧文，加里弗尼亚州 92697，美国；
电子邮箱：mgoulden@uci.edu

[2] 开源感知公司，柳树路 68 号，门洛帕克，加里弗尼亚州 94025，美国；
电子邮箱：rusu@openperception.org

[*] 通信作者；电子邮箱：gazzari@uci.edu；电话：+1-949-824-9273。

收稿日期：2012 年 11 月 14 日；修改版本：2012 年 12 月 26 日／接受日期：2013 年 1 月 31 日／出版日期：2013 年 2 月 11 日

摘要：植物结构和生物量在控制陆地—大气交换方面的重要性已被广泛认知，但冠层结构的测量却极具挑战、耗费时间，并且常常依赖于破坏性的方法。微软 Kinect 传感器是一个为视频游戏而设计的红外传感器，它输出同步的颜色和深度图像，并且具有实现植物结构快速特性描述的潜能。我们对生长于加利福尼亚州草地上的两个物种比较了从 Kinect 传感器得到的深度图像和对植物结构和大小的人工测量结果。深度图像与对植物大小的水平和垂直人工测量结果吻合的很好。相似的，利用三维凸包方法计算的植物体积也与植物生物量良好相关。Kinect 在有关较短的测量范围和白天光污染的生态观测方面显示出一些局限性。但是，Kinect 的较轻质量、较快获取时间、较低能量要求和费用使之成为一个冠层结构现场快速调查，尤其是对较小的植物的，有前景的工具。

关键词：陆地生态学；现场测量；冠层结构；生物量；LIDAR；微软 Kinect；点云；深度图像；凸包；凹包

1. 简介

植物结构和生物量在控制陆地生态系统功能和陆地—大气交换方面的重要性已被广泛认知。冠层结构影响被树叶所截断的光线，它是一个控制初级生产、蒸发蒸腾和植物竞争的主要因素[1]。植物结构通过反射率、发射率、潜热通量和显热通量的变化影响生态系统—大气的质量和能量交换；这些属性在根本上影响着当地和区域尺度上的气候[2-5]。生物量的积累在当地和全球碳循环中扮演重要的叫角色，同时也决定能源积累并为土壤营养平衡做出贡献[6]。

冠层光转移模型，它是陆地表面模型的一个基本组成，常常依赖于用如箱体、球体、圆柱体或圆锥体等实体形状表现植物的简化冠层（[1]，及其中的参考文献）。生物量通常利用基于

植物形状和体积的近似简化假设的异速生长关系来进行估计。树体结构的真实信息，尤其是如果能够在空间和时间上扩展，将能够改进生物物理冠层模型和生物量估计，并有助于我们理解生态系统功能。

冠层结构的测量是极具挑战性的[6]。从卫星的遥感测量可以提供具有良好空间时间分辨率的大尺度的数据[6-8]，但它们需要地面上的验证。陆基测量均代价很高且耗费时间，并常常依赖于破坏性方法。现场数据通常在小面积上采集；扩展这些观测的尺度至更长的时间段或更广的空间范围往往很难。激光扫描系统例如 LIDAR 在近期变得可被利用，提供了一个机载和地面测量的高效的非破坏性方法[9-15]。然而，LIDAR 仍然昂贵，其数据可得到性仍然局限，尤其是个别地点的冠层结构时序数据。廉价、便携的生物量和植物结构的陆基测量仪器将会增加结构测量的空间和时间覆盖，并可能被整合入大型仪器网络例如 FluxNet 或 SpecNet[16-18]。

视频游戏和数码娱乐的消费者市场已在近三十年里急剧地扩张了，使得先进科技的价格变为可承担的水平。微软生产的 Kinect 传感器是这些科技的一个优秀的例子：Kinect 是一个为在单关节水平上跟踪身体位置和运动而设计的红外传感器。Kinect 传感器仅需 150 美元即可获得，这仅相当于一个研究级别陆基 LIDAR 系统耗资的大约 0.1%。在本“概念验证”论文中，我们显示了 Kinect 在包括基圆直径、高度和体积等的植物结构现场测量，以及评价冠层建模和生物量估计的合适实体形状近似方面是有用的。我们观测了两个植物物种的 4-8 个副本来测试 Kinect 传感器的现场表现。我们直接比较了人工测量的和从 Kinect 获取的植物点云中提取的基圆直径和高度的测量结果。我们也比较了用点云提取的凸包计算的植物体积和可供选择的实体形状近似。最后，我们测量了样本植物的干生物量，并用 Kinect 提取的结构性变量来导出异速生长关系。

2. 方法

2.1. 微软 Kinect 传感器

Kinect 以每秒 30 帧的速率（*fps*）捕获同步的颜色和深度图像，并以 57 °×43 °的视场角利用一个 RGB 相机（8 位 640×480 像素的 VGA 分辨率）与一个深度成像器对齐。Kinect 深度传感器由 PrimeSense 设计，由一个 IR 激光投影仪和一个单色 640×480 像素 IR CMOS 传感器组成。深度信息以每像素 11 位精度输出[19]。Kinect 深度测量原理不同于激光扫描仪例如 LIDAR。LIDAR 通过测量利用机动云台单元顺序扫描整个场景的单束激光脉冲的飞行时间生成深度图像。Kinect 激光投影仪一次性照亮整个场景，利用一种施加在光束上的一致斑点模式的衍射网格[20]。深度进而通过由 CMOS 传感器捕获的激光模式和存储在传感器存储器中的参考模式之间的相关和三角测量而计算得到[20,21]。这种获取方法回避了移动部件的需要，从而减小了传感器的质量、体积、获取时间和能量需求。

通过使用外部软件，深度图像可以被转换并存储在名为点云的三维表现形式中。点云中的每一个点由一个坐标的集合 *P={x,y,z}*表达，它定义了该点在空间中的位置。点坐标也可以被扩展为 *P={x,y,z,R,G,B}*的形式，它包括了由对齐的 RGB 相机记录的红色、绿色、蓝色成分，或者 *P={x,y,z,I}*，它包括了强度信息。本研究中仅使用了包含 *x,y,z* 坐标的点云。Kinect 提取的点

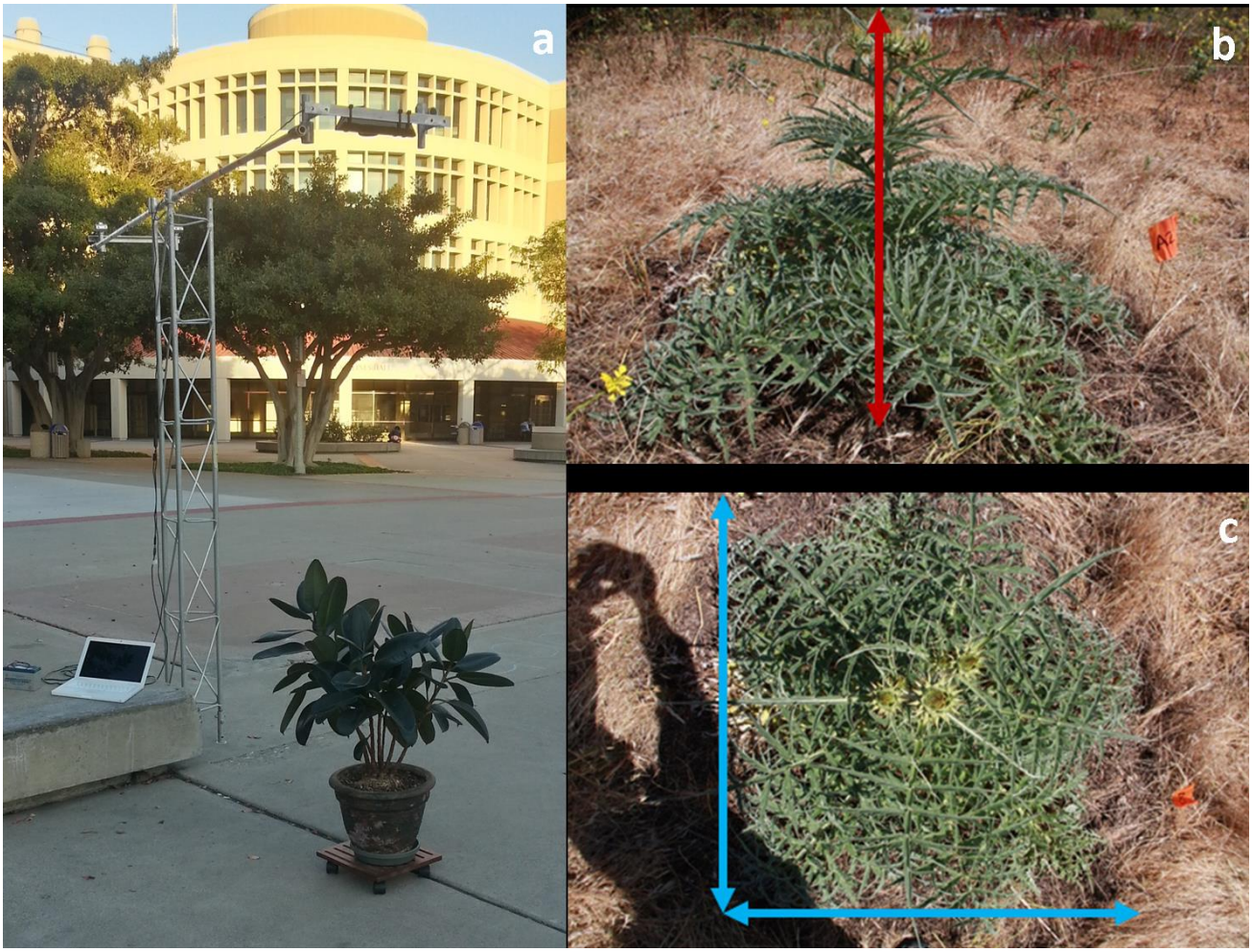
云，它的每一帧包含大约 300,000 个点，在水平轴上有 3 毫米的误差。随着离传感器的距离的增大，误差也在增大[20]；深度误差在 2 米处为±1 厘米，而在 5 米处为±7 厘米[20]。微软推荐的适宜测量范围为 1.2-3.5 米，尽管测试表明 Kinect 能够胜任 0.8-6.0 米范围的测量[19]。其他因素也会影响深度误差，包括光条件和目标反射率。明亮的光和高反射率降低了目标表面的激光模式的对比度，这造成了结果点云中的缝隙和离群点[20]，也如我们的预先测试中展现的那样（见 3.1 节）。

2.2. 植物点云的获取、处理和分析

点云的获取、处理和分析是利用名为点云库（PCL，[22]）的开源 C++库中包含的工具和方法来执行的。这些方法的详细描述不在本文的范围内；完整文档提供在 PCL 网站上（www.pointclouds.org）。

2.2.1. 点云获取

图 1. **(a)**用于 Kinect 俯视测量的“塔模式”试验布局（见 2.3 和 2.4 节）。现场测试中记录的一株野生朝鲜蓟植物的**(b)**侧视图和**(c)**俯视图（见 2.4 节）。蓝色和红色线条表示以人工尺子测量的植物的基圆直径和高度。



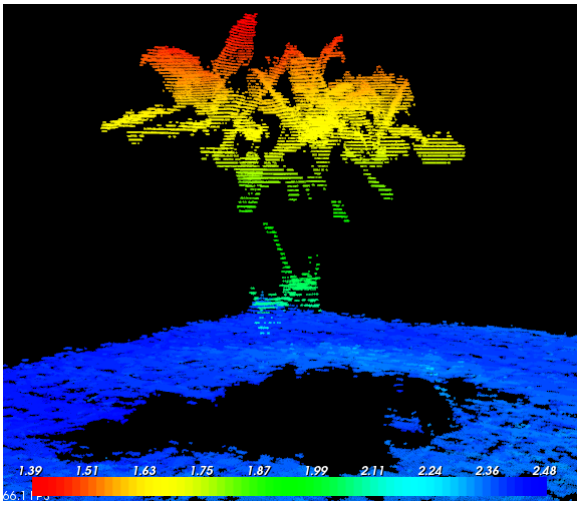
点云是利用 PCL OpenNI 抓取器通过 Kinect 获取的，这是一个兼容许多设备的 I/O 接口。获取软件运行在一个装有 Linux Ubuntu 12.04 LTS 的笔记本电脑上。本文中使用了两种获取策略。(*i*)“塔模式”：Kinect 安装在一个 1.4 米长的桅杆上，它从一个 3 米高的铝塔上水平伸出。定向传感器以提供一个植物的俯视图（见图 1(a)）。从塔布局的获取以单次拍照模式执行：记录下单个点云和对应的原始红外图像。(*ii*)“多角度模式”：将 Kinect 围绕植物人工移动 360 度，同时点云的流被记录下来。一个完整的植物 360 度扫描生成大约 2000 个点云，并占用约 2GB 的磁盘空间。流化多角度点云进而利用 Kinfu PCL 工具（Kinect Fusion，[23]）被合并为单个点云。Kinfu 不依赖于目标或用户交互来配准点云。由于它巨大的计算需求，Kinfu 必须在配备图形处理单元（GPU）的电脑上运行。如果获取及其配备了这种设备，配准可以实时执行，而不是在记录的流中以“离线”模式执行。

2.2.2. 来自点云的冠层结构

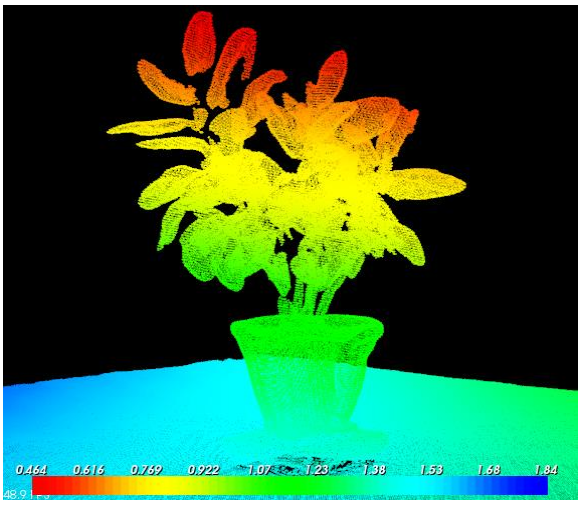
点云接下来通过下述四步进行处理：(*i*)噪声消减和过滤以去除离群点，(*ii*)个体植物描述和点提取，(*iii*)计算植物的 *x*，*y* 和 *z* 大小，以及(iv)计算植物的体积。所有后处理和分析均在实验室中执行，利用一台配备 1.5Gb GPU 并运行 Linux Ubuntu 12.04 LTS 的桌面计算机。

1. 噪声消减和过滤以去除离群点：欺骗性的单个点（离群点）由统计性滤波器去除。点云之后进行向下采样到 1 厘米的网格以减少计算需求。
2. 个体植物描述和点提取：利用直观确定的“箱坐标”，即植物占据的 *x,y,z* 范围，将单个植物初步隔离。也探索了无监督的植物检测替代方法，包括土壤平面的平面分割和欧式聚类。不幸的是，这些方法给出了不一致的结果，特别是应用在现场获取的点云中时（如 2.4 节中）。下层植被的不规则性和包括多个植物的点云显示出了特别的挑战。
3. 计算植物的 *x*，*y* 和 *z* 大小：每个点（*P*）表达为 *x,y,z* 坐标；沿着每个轴的植物大小由 $d_i=max(P_i)-min(P_i)$来计算，此处 *i* 可以是 *x*、*y* 或 *z*，以及 *P* 属于隔离的植物点子集。
4. 计算植物的体积：有许多几何算法可用于计算一个植物点云的体积。原则上，这些方法应该提供一个比实体形状近似可达到的更为准确的体积估计。凸包和凹包已被证实尤其适用于这种问题，例如[13]。一个平面上的点的集合 $Q=\{p_1,p_2,\ldots,p_n\}$的凸（凹）包 *C* 是“独特的凸（凹）多边形，其顶点均为 *Q* 中的点并包含 *Q* 中的所有点”[24]。这个定义可以被扩展到三维，*C* 则是一个曲面网格而非多边形。植物的体积和结构应该利用凹包而非凸包来进行理想估计，因为这可以排除树枝和树叶间的空隙。凹包估计需要每一个植物的完整点云，且它只能应用于配准的多角度点云。从仅俯视图的点云进行包估计可能会遇到问题，因为树叶和树干在 *z* 轴上存在着互相遮挡，导致一个在顶部较大部分和土壤之间的植物的不完整表达。在仅俯视图的点云中凸包在植物基点确定后进行估计。我们使用植物的阴影来定义基。阴影点利用基于深度数据的边缘识别方法来识别。这种方法返回点的三个子集，它们对应于边缘（图 2(c)中以白色显示）、阴影（图 2(c)中以灰色显示）和遮罩点。

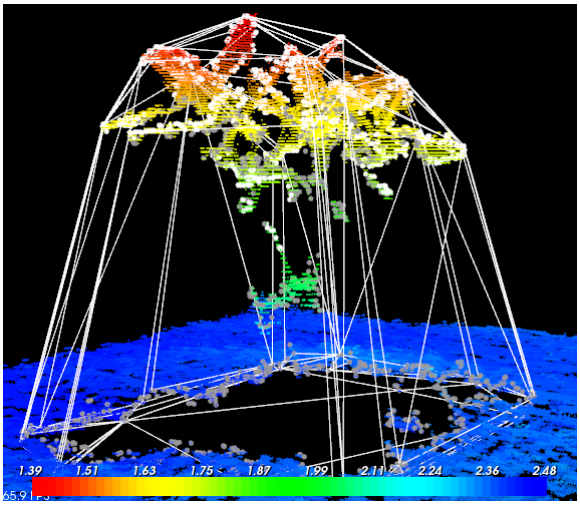
图 2. 仅俯视图（a 和 c；利用“塔模式”采集）和配准的多角度（b 和 d；利用“多角度模式”采集）点云之间的比较。上半嵌板显示从一个任意的水平透视视角观察的点云。下半嵌板显示以白色线条叠映的凸包**(c)**和凹包**(d)**。嵌板**(c)**显示阴影点（灰色）和边缘点（白色），它们用于定义凸包。点云的比色刻度尺是距离传感器最低点的距离（米）。



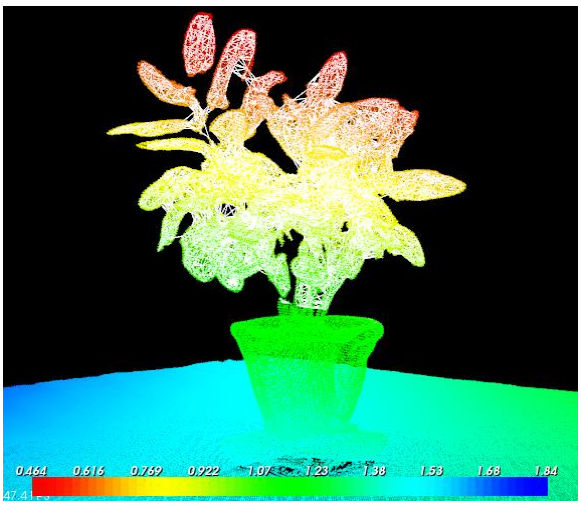
(**a**)仅俯视图点云



(**b**)配准的多角度点云



(**c**)带阴影和边缘点的凸包



(**d**)凹包

2.3. 在 Kinect 上的预先测试

我们在一定范围条件下测试 Kinect 以：(a)评估光条件对测量的影响，及(b)比较固定角度观察的 Kinect（即用“塔模式”的点云）和配准获取（即用“多角度模式”的点云）得到的信息。

我们在一个盆栽橡胶榕树（印度榕）上执行这些测试，它约为 1.1 米高（图 1(a)）。印度榕的大而光滑的叶子特别容易被 Kinect 所检测。植物被放置于一个光滑的混凝土表面上，它的红外反射率与植物叶片存在显著差异。利用“塔模式”布局每 30 分钟获取一次图像，始于日落前 1.5 小时并结束于日落后 1.5 小时。此外，将传感器从塔上取下并在夜间用于“多角度模式”以获取完整的植物点云。

2.4. 在现场测试 Kinect

我们也在更为自然的草地环境下评估了 Kinect 的表现和局限性。一个在现场利用 Kinect 提供冠层结构信息的可靠性的完整评价可能要利用与对应陆基 LIDAR 测量的直接比较。不幸的是，我们无法得到这样的设备，因而仅比较植物大小的 Kinect 测量结果和尺子测量结果。用尺子测量植物维度是一个现场生态学的常见的做法，尽管得到的结果数据可能不足以解释植物复杂的形状。我们的现场比较并非意在量化利用 Kinect 测量植物维度的绝对准确性。事实上，可能 Kinect 观测远比尺子测量更为准确。

现场地址位于加州大学欧文分校（坐标：33.640102°N，117.845314°W）。这一地区具有地中海气候特点；大部分降水在冬季，而夏季确实干燥。研究在 2011 年五月初进行，这是植物生长季节的末期。此地由一年生草地和矮小灌木镶嵌而成，尽管研究期间草已经枯萎了。在研究期间两种大型植物物种非常丰富，并达到了它们生长季节的顶峰：朝鲜蓟（野生洋蓟；图 1(b,c)）和刺缘毛莲菜（刚毛毛连芽）。因为它们的充足性、它们的大小范围以及它们在干燥植被环境中表现出的对比度，这两个物种被选用于研究。

具有不同大小的朝鲜蓟的八个植株和刺缘毛莲菜的四个植株被选择并标记。我们在夜间获取了每个研究植物的几个点云，此时的光照条件使得测量误差达到最小。所有点云的获取均使用“塔模式”完成，塔用手携带至每个选定植株。我们随后用尺子测量了每个植株的基圆直径和高度。两个物种都具有圆锥的形状，它们的基圆直径在地面水平沿着两个正交轴进行测量（两个中的一个是最大直径，见图 1(b,c)）。两个基底测量值随后进行平均以供分析和比较。植物体积利用人工测量的基圆直径（d）和高度（h）进行计算，通常使用实体形状包括矩形箱（$V=d_2\,h$）、圆柱（$V=d_2\cdot\pi\cdot h/4$）、圆锥（$V=d_2\cdot\pi\cdot h/12$）、抛物锥（$V=d_2\cdot\pi\cdot h/8$）和椭圆锥（$V=d_2\cdot\pi\cdot h/6$）。我们随后收割了植物并测量了它们的干生物量。

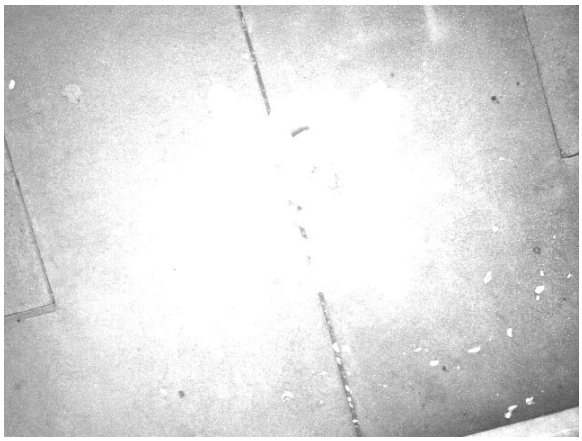
3. 结果

3.1. 使用 Kinect 预先测试

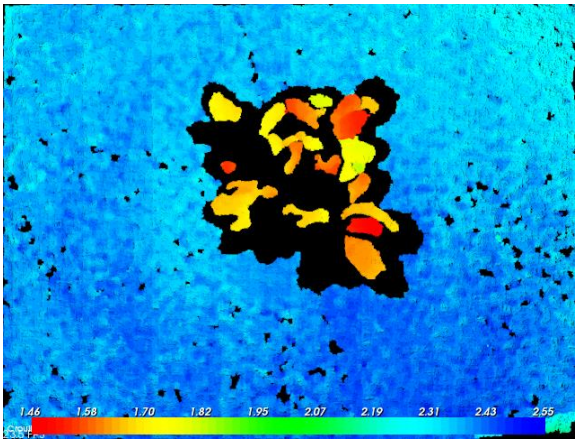
我们的预先测试显示出光条件和目标反射率影响深度测量的质量。图 3 的一系列图像显示出深度范围是如何随着光条件变暗而逐步增大的。在日光下的表现较差（见图 3(a,b)）明显是由于叶子的高 IR 反射率。来自日光的反射的 IR 辐射明显降低了植物表面的 Kinect 激光模式的对比度（图 3(a)），导致点云包含很多缺失点（图 3(b)）。为了获得良好定义的橡胶榕树植物的点云，夜间测量是需要的（图 3(e,f)）。背景混凝土表面是距离传感器最远的目标，它在所有光条件下均可检测到，尽管在较明亮的日光下存在缺失点。混凝土在 IR 下反射率较小（图 3(a,c)），这减少了由日光污染造成的误差和数据间隙。

多角度点云的配准在我们的预先测试中提供了极佳的结果（图 2(a,b)）。配准的点云提供了研究植株的完整表达，使得利用凹包对冠层结构进行详细建模成为可能（比较图 2(d)和图 2(c)）。附加测试（未在本文中体现）显示，多角度获取对于在具有复杂下层植被地区的小叶植物产生的结果稍显不一致。

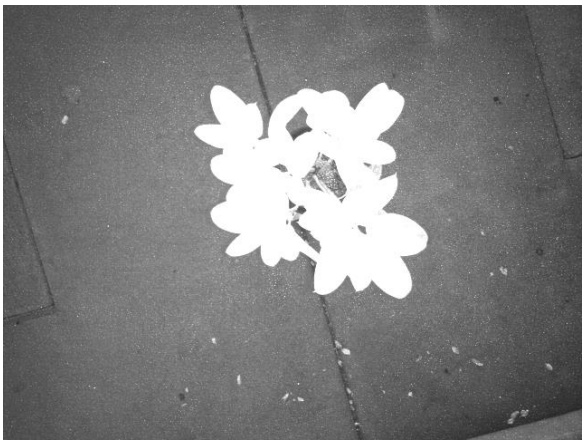
图 3. 在不同光条件下的 Kinect 原始红外输出（**a**，**c**，**e**）和对应的点云（**b**，**d**，**f**）的比较。所有图像均为一个约 1.1 米高的盆栽橡胶树植物（即图 1(a)）。点云的比色刻度尺是距离传感器最低点的距离（米）。



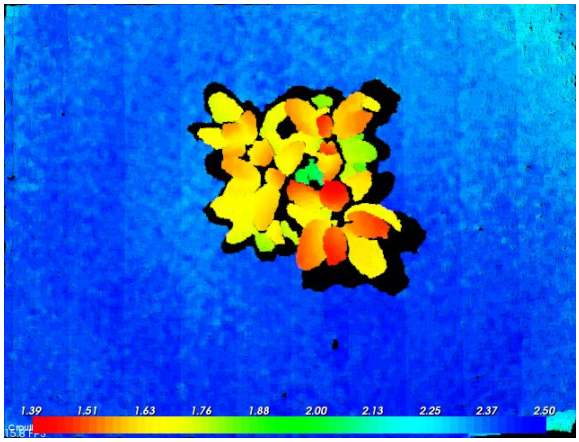
(**a**)午后红外原始图像



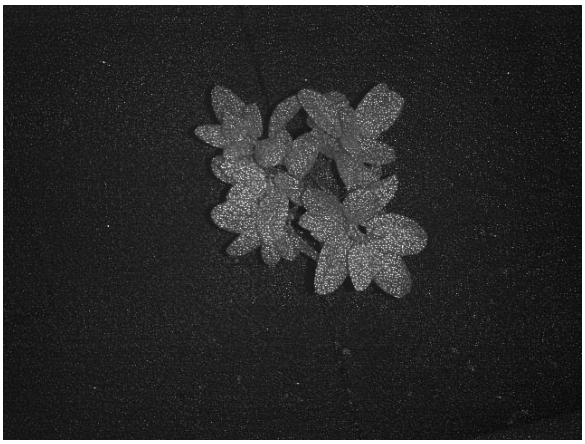
(**b**)午后点云



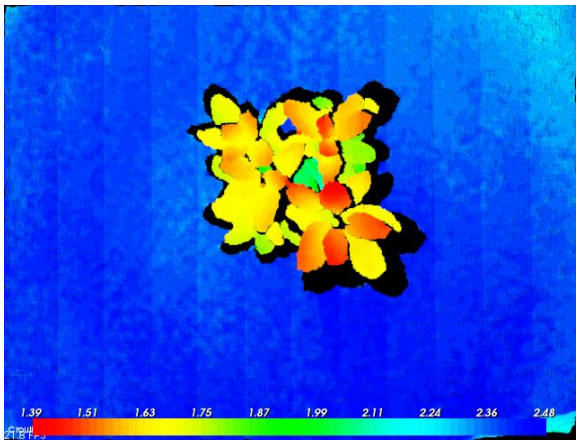
(**c**)傍晚红外原始图像



(**d**)傍晚点云



(**e**)夜间红外原始图像

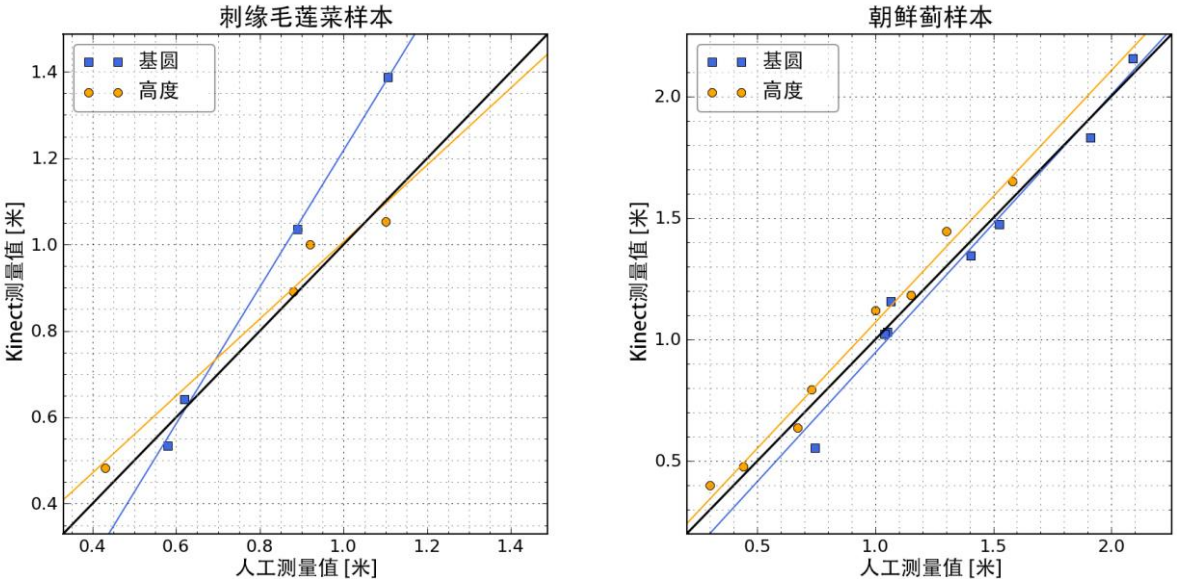


(**f**)夜间点云

3.2. 在现场测试 Kinect

植物垂直和水平维度的人工的和 Kinect 提取的测量结果比较显示在图 4 中。结果显示所有的测量都吻合的较好，尤其是朝鲜蓟。从恒等线计算的标准化均方根误差（$NRMSE_{y=x}$）在 2.7%到 19.1%的范围内，最小的误差对应于朝鲜蓟的基圆测量，而最大的误差对应于刺缘毛莲菜的基圆测量。朝鲜蓟得到的数据吻合的更好可能反映了这些植株具有更为明确的形状，从而简化了手工测量。

图 4. 植物基圆直径（“基圆”；x 和 y 测量值的平均）和植物高度（“高度”）的人工和 Kinect 测量结果的直接比较。实线表示最小二乘线性拟合。



使用箱体、圆柱、圆锥、抛物锥和椭圆锥实体模型并根据人工测量结果对植物体积的估计，与用凸包根据 Kinect 数据计算的结果进行了比较（图 5）。$RMSE_{y=x}$ 值的范围在 1.0%到 106.1%。椭圆锥提供了两个植物物种的最优形状近似，其 $RMSE_{y=x}$ 值对朝鲜蓟为 1.0%，对刺缘毛莲菜则为 2.1%。箱体和圆柱模型常用于冠层建模，却有着最大的 $RMSE_{y=x}$ 值（从 57%到 106%）。

将干生物量与 Kinect 测定的高度、直径和体积进行比较以导出异速生长关系（见图 6）。通过一个 $y=A+B\ \text{log}(x+C)/(\text{log}(D)+1)$形式的对数函数，不同的植物大小的测度都与植物的生物量良好相关。结果的决定系数（r^2）范围在 0.97 至 1.0 之间。

图 5. 凸包得到的体积和使用多种实体模型近似的人工测量结果导出的体积的直接比较。实线表示最小二乘线性拟合。

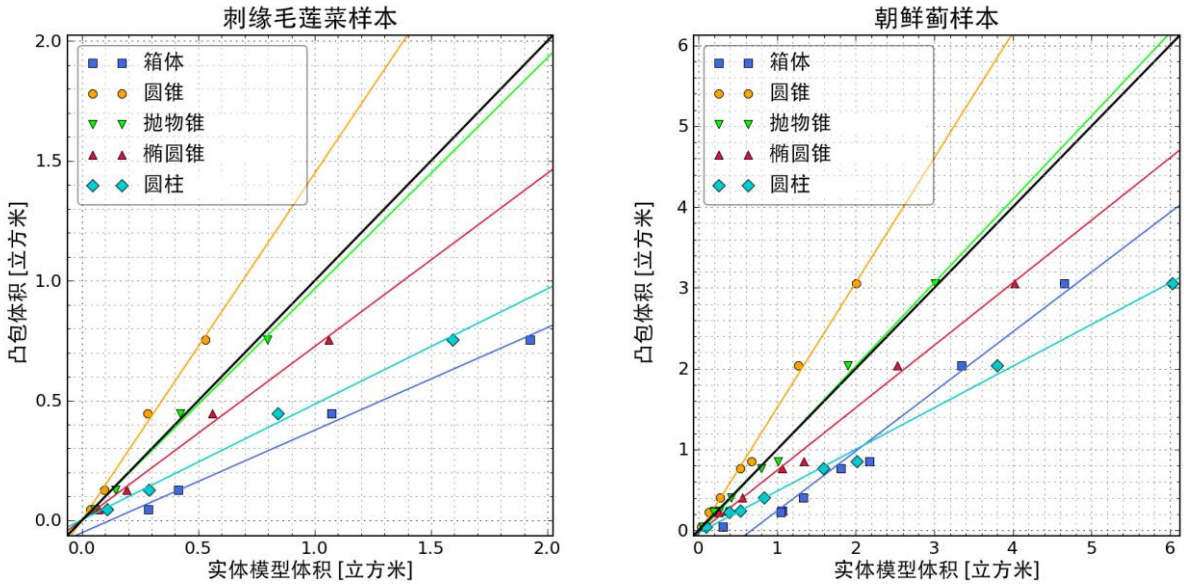
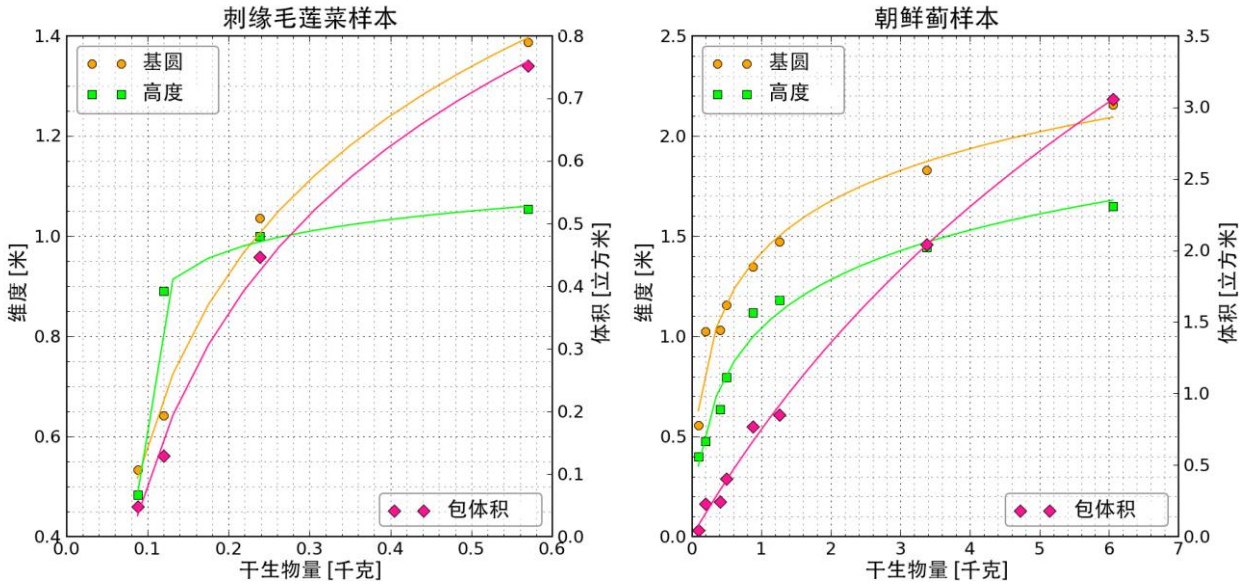


图 6. Kinect 提取的维度（基圆和高度）与干生物量测量结果之间的异速生长关系。实线表示 $y=A+B\,\log(x+C)/(\log(D)+1)$形式的最小二乘对数拟合。



4. 讨论

4.1. Kinect 传感器的优势和劣势

微软 Kinect 传感器结合点云库提供了对植物高度和基圆直径的测量，这种测量能够与人工测量结果吻合的较好。相似的，三维凸包估计提供了有关植物体积和形状的有用信息，这与生物物理冠层建模和生物量估计密切相关。用 Kinect 取得的结构性信息可用于得到两个植物物

种的异速生长关系。Kinect 的较轻质量、较快获取时间、较低能量要求和费用使之成为一个冠层结构现场快速调查，尤其是对较小的植物的，有前景的工具。

承认 Kinect 传感器在植物传感方面存在局限性也是十分重要的。Kinect 无法在日光下测量植物，且测量范围局限在几米内。受可及性限制，在现场地点的夜间测量可能很困难。在一个固定地点测量，例如一个通量塔的顶部，可以避免这个问题，但在空间上将会受到 Kinect 测量范围的限制。

“塔模式”采集的图像结果在挤满浓密的植被、或具有复杂的活的地面覆盖物的情况下稍显不一致。“多角度模式”格外具有前景，它使得创造具有超凡细节的点云成为可能（图 2(d)），但较小叶片和浓密植被可能会使配准过程出现问题。

4.2. 对未来的展望

Kinect 在现场测量植物的局限性并不令人吃惊；Kinect 是为室内视频游戏而设计的，而并非为了现场的生态学观测。但是，当前形式的 Kinect 已经成功用于一些生态学应用，这些测量原则也为设计专用于植物特性描述的新传感器打开了可能性。

基于 Kinect 设计的新的植物传感器可以使用一个更为强力的激光源，以增加测量范围并使较高植物的特性描述成为可能。这也可能会改善白天的观测，尤其如果投影仪的波长可以为了增强日光环境下的植物目标的对比度来进行选择。白天使用的深度传感器的发展将使获取对齐的 RGB 相机的图像成为可能；与每个点关联的 RGB 信息可以接着被用于物候学研究，如[25]。

另外，Kinect 方法可能被用于确定植物三维结构的光谱反射率。激光源的强度是已知的，那么一旦每一帧上反射斑点模式的强度得到后就可以计算出反射率。原理上，可以使用多个照明波长，使得计算光谱植被指数成为可能。在现场的多光谱反射率测量中使用主动照明而非被动照明，已经被最近的研究所测试，如[26,27]。主动照明具有减弱对天气和光条件的依赖的优点。此外，源传感器几何已经固定且已知，避免了考虑双向反射分布函数（BRDF）的需要[26]，尽管离散的激光返回仍然依赖于如植物结构、几何、入射角等因素。

获取程序、软件和数据处理都可以进行改进。获取和配准多角度点云的算法可以进行改进以使在具有较小植物或复杂下层植被等当前易出错的现场的获取成为可能。现场的多角度测量可以采取现场活动时手动控制，或使用云台单元（如在通量塔上），或使用多个传感器，或使传感器在较低高度飞行，如[28-30]。量化结构性参数如叶面积指数和叶角度分布的更为准确的算法，可以从配准的点云开始发展。

5. 结论

微软 Kinect 展现了现场测量小植物结构的潜能，但它的设计在当前限制了生态学研究的可能应用范围。但是，软件和廉价消费电子产品的发展正为生态学应用开启令人激动的可能性。考虑到廉价和易用的传感器的快速发展，对这些技术的进一步研究是无法避免的。这转而增加了地面数据的空间和时间覆盖以及可得到性，提供了遥感测量验证的更好的信息并改进了生物物理冠层建模（如图 2(d)）。

感谢

NASA 和 DOE 提供了研究支持（资助号 NNX10AL14G 和 0011182）。Greg Jenkins 及其家人以研究生奖学金支持本研究。

参考文献

1. Brunner, A. A light model for spatially explicit forest stand models. *Forest Ecol. Manage.* **1998**, *107*, 19–46.
2. Pielke, R.A.I.; Avissar, R. Influence of landscape structure on local and regional climate. *Landscape Ecol.* **1990**, *4*, 133–155.
3. Rotenberg, E.; Yakir, D. Distinct patterns of changes in surface energy budget associated with forestation in the semiarid region. *Glob. Change Biol.* **2011**, *17*, 1536–1548.
4. Lee, X.; Goulden, M.L.; Hollinger, D.Y.; Barr, A.; Black, T.A.; Bohrer, G.; Bracho, R.; Drake, B.; Goldstein, A.; Gu, L.; *et al.* Observed increase in local cooling effect of deforestation at higher latitudes. *Nature* **2011**, *479*, 384–387.
5. Bonan, G.B. Forests and climate change: Forcings, feedbacks, and the climate benefits of forests. *Science* **2008**, *320*, 1444–1449.
6. Lu, D. The potential and challenge of remote sensing based biomass estimation. *Int. J. Remote Sens.* **2006**, *27*, 1297–1328.
7. Powell, S.L.; Cohen, W.B.; Healey, S.P.; Kennedy, R.E.; Moisen, G.G.; Pierce, K.B.; Ohmann, J.L. Quantification of live aboveground forest biomass dynamics with Landsat time-series and field inventory data: A comparison of empirical modeling approaches. *Remote Sens. Environ.* **2010**, *114*, 1053–1068.
8. Bergen, K.M.; Goetz, S.J.; Dubayah, R.O.; Henebry, G.M.; Hunsaker, C.T.; Imhoff, M.L.; Nelson, R.F.; Parker, G.G.; Radeloff, V.C. Remote sensing of vegetation 3-D structure for biodiversity and habitat: Review and implications for lidar and radar spaceborne missions. *J. Geophys. Res.* **2009**, *114*, 1–13.
9. Seidel, D.; Beyer, F.; Hertel, D.; Fleck, S.; Leuschner, C. 3D-laser scanning: A non-destructive method for studying above-ground biomass and growth of juvenile trees. *Agr. Forest Meteorol.* **2011**, *151*, 1305–1311.
10. Henning, J.; Radtke, P. Ground-based laser imaging for assessing three dimensional forest canopy structure. *Photogramm. Eng. Remote Sens.* **2006**, *72*, 1349–1358.
11. Naesset, E.; Gobakken, T.; Nasset, E. Estimation of above- and below-ground biomass across regions of the boreal forest zone using airborne laser. *Remote Sens. Environ.* **2008**, *112*, 3079–3090.
12. Zhao, K.; Popescu, S.; Nelson, R. Lidar remote sensing of forest biomass: A scale-invariant estimation approach using airborne lasers. *Remote Sens. Environ.* **2009**, *113*, 182–196.

13. Morsdorf, F.; Meier, E.; Allgo ̈wer, B.; Nu ̈esch, D. Clustering in airborne laser scanning raw data for segmentation of single trees. *Int. Arch. Photogramm. Remote Sens. Spat. Inform. Sci.* **2003**, *34*, 27–33.
14. Jochem, A.; Hollaus, M.; Rutzinger, M.; Ho ̈fle, B. Estimation of aboveground biomass in alpine forests: A semi-empirical approach considering canopy transparency derived from airborne LiDAR data. *Sensors* **2011**, *11*, 278–295.
15. Wang, Y.; Weinacker, H.; Koch, B. A LiDAR point cloud based procedure for vertical canopy structure analysis and 3D single tree modelling in forest. *Sensors* **2008**, *8*, 3938–3951.
16. Baldocchi, D.; Falge, E.; Gu, L.; Olson, R.; Hollinger, D.; Running, S.; Anthoni, P.; Bernhofer, C.; Davis, K.; Evans, R.; Fuentes, J.; Goldstein, A.; Katul, G.; Law, B.E.; Lee, X.; Malhi, Y.; Meyers, T.; Munger, W.; Oechel, W.; Paw U, K.T.; Pilegaard, K.; Schmid, H.P.; Valentini, R.; Verma, S.; Vesala, T.; Wilson, K.; Wofsy, S. FLUXNET: A new tool to study the temporal and spatial variability of ecosystem-scale carbon dioxide, water vapor, and energy flux densities. *Bull. Amer. Meteorol. Soc.* **2001**, *82*, 2415–2434.
17. Gamon, J.; Rahman, A.; Dungan, J.; Schildhauer, M.; Huemmrich, K. Spectral Network (SpecNet)-What is it and why do we need it? *Remote Sens. Environ.* **2006**, *103*, 227–235.
18. Balzarolo, M.; Anderson, K.; Nichol, C.; Rossini, M.; Vescovo, L.; Arriga, N.; Wohlfahrt, G.; Calvet, J.C.; Carrara, A.; Cerasoli, S.; *et al.* Ground-based optical measurements at european flux sites: A review of methods, instruments and current controversies. *Sensors* **2011**, *11*, 7954–7981.
19. Livingston, M.A.; Sebastian, J.; Ai, Z.; Decker, J.W. Performance Measurements for the Microsoft Kinect Skeleton. In *Proceedings of the IEEE Virtual Reality Short Papers and Posters*, Costa Mesa, CA, USA, 4–8 March 2012; pp. 119–120.
20. Khoshelham, K. Accuracy Analysis of Kinect Depth Data. In *Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Calgary, Canada, 29–31 August 2011.
21. Freedman, B.; Shpunt, A.; Meir, M.; Yoel, A. Depth Mapping Using Projected Patterns. US Patent 8,150,142, 6 September 2007.
22. Rusu, R. 3D is Here: Point Cloud Library (PCL). In *Proceedings of the IEEE International Conference on Robotics and Automation*, Shanghai, China, 9–13 May 2011; pp. 1–4.
23. Izadi, S.; Kim, D.; Hilliges, O. KinectFusion: Real-Time 3D Reconstruction and Interaction Using a Moving Depth Camera. *In Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, Santa Barbara, CA, USA, 16–19 October 2011.
24. De Berg, M.; Cheong, O.; Van Kreveld, M.; Overmars, M. *Computational Geometry. Algorithms and Applications*; 3rd ed.; Springer-Verlag: Heidelberg, Germany, 2008.
25. Richardson, A.; Jenkins, J. Use of digital webcam images to track spring green-up in a deciduous broadleaf forest. *Oecologia* **2007**, *152*, 323–334.

26. Fitzgerald, G.J. Characterizing vegetation indices derived from active and passive sensors. *Int. J. Remote Sens.* **2010**, *31*, 4335–4348.
27. Puttonen, E.; Suomalainen, J.; Hakala, T.; Ra ̈ikko ̈nen, E.; Kaartinen, H.; Kaasalainen, S.; Litkey, P. Tree species classification from fused active hyperspectral reflectance and LIDAR measurements. *Forest Ecol. Manage.* **2010**, *260*, 1843–1852.
28. Dandois, J.P.; Ellis, E.C. Remote sensing of vegetation structure using computer vision. *Remote Sens.* **2010**, *2*, 1157–1176.
29. Wallace, L.; Lucieer, A.; Watson, C.; Turner, D. Development of a UAV-LiDAR system with application to forest inventory. *Remote Sens.* **2012**, *4*, 1519–1543.
30. Rosnell, T.; Honkavaara, E. Point cloud generation from aerial image data acquired by a quadrocopter type micro unmanned aerial vehicle and a digital still camera. *Sensors* **2012**, *12*, 453–480.

Sensors **2013**, *13*, 2384-2398; doi:10.3390/s130202384

OPEN ACCESS

sensors

ISSN 1424-8220

www.mdpi.com/journal/sensors

Article

Rapid Characterization of Vegetation Structure with a Microsoft Kinect Sensor

George Azzari [1,*], Michael L. Goulden [1] and Radu B. Rusu [2]

[1] Department of Earth System Science, University of California Irvine, Irvine, CA 92697, USA; E-Mail: mgoulden@uci.edu

[2] Open Perception, Inc., 68 Willow Road, Menlo Park, CA 94025, USA; E-Mail: rusu@openperception.org

* Author to whom correspondence should be addressed; E-Mail: gazzari@uci.edu; Tel.: +1-949-824-9273.

Received: 14 November 2012; in revised form: 26 December 2012 / Accepted: 31 January 2013 / Published: 11 February 2013

Abstract: The importance of vegetation structure and biomass in controlling land-atmosphere exchange is widely recognized, but measurements of canopy structure are challenging, time consuming, and often rely on destructive methods. The Microsoft Kinect is an infrared sensor designed for video gaming that outputs synchronized color and depth images and that has the potential to allow rapid characterization of vegetation structure. We compared depth images from a Kinect sensor with manual measurements of plant structure and size for two species growing in a California grassland. The depth images agreed well with the horizontal and vertical measurements of plant size made manually. Similarly, the plant volumes calculated with a three-dimensional convex hulls approach was well related to plant biomass. The Kinect showed some limitations for ecological observation associated with a short measurement range and daytime light contamination. Nonetheless, the Kinect's light weight, fast acquisition time, low power requirement, and cost make it a promising tool for rapid field surveys of canopy structure, especially in small-statured vegetation.

Keywords: terrestrial ecology; field measurements; canopy structure; biomass; LIDAR; Microsoft Kinect; point clouds; depth images; convex hulls; concave hulls

1. Introduction

The importance of vegetation structure and biomass in controlling terrestrial ecosystem function and land-atmosphere exchange is widely recognized. Canopy architecture affects the interception of light by leaves, which is a dominant factor controlling primary production, evapotranspiration, and plant competition [1]. Vegetation structure influences ecosystem-atmosphere mass and energy exchange through variation in albedo, emissivity, latent heat flux, and sensible heat flux; these properties ultimately affect climate on local and regional scales [2–5]. The accumulation of biomass plays a major role in the local and global carbon cycle, while also determining fuel accumulation and contributing to soil nutrient balance [6].

Canopy light transfer models, which are basic components of land surface models, often rely on simplified canopies that represent plants as solid shapes such as boxes, spheres, cylinders, or cones ([1], and references therein). Biomass is often estimated using allometric relations based on similar simplified assumptions about plant shape and volume. Realistic information on stand structure, especially if extended in space and time, could improve biophysical canopy models and biomass estimations, while contributing to our understanding of ecosystem function.

Measurements of canopy structure are challenging [6]. Remotely sensed measurements from satellites can provide large scale data with good spatial and temporal resolution [6–8], but they require validation on the ground. Ground-based measurements are costly and time consuming, and often rely on destructive methods. Field data are usually collected over small areas; it is difficult to scale up these observations to longer time periods or larger spatial scales. Laser scanning systems such as LIDAR have recently become more available, offering an effective non-destructive method for airborne and ground measurements [9–15]. However, LIDAR remains expensive and data availability is still limited, especially time-series of canopy structure at individual locations. Inexpensive and portable instrumentation for ground-based measurements of biomass and vegetation structure would increase the spatial and temporal coverage of structural measurements, and might be integrated into large instrument networks such as FluxNet or SpecNet [16–18].

The consumer market for video-games and digital entertainment has expanded dramatically in the last three decades, bringing the cost of advanced technology to affordable levels. The Kinect sensor produced by Microsoft is a good example of this technology: Kinect is an infrared sensor designed to track body position and movement at a single-articulation level. Kinect sensors are available for US $150, which corresponds to roughly 0.1% of the cost of a research-grade, ground-based LIDAR system. In this "proof of concept" paper, we show that the Kinect sensor is useful for field measurements of vegetation structure, including base diameter, height, and volume, and for assessing the optimal solid shape approximation for canopy modelling and biomass estimation. We made observations on 4 to 8 replicates of two plant species to test the performance of Kinect sensors in the field. We directly compared measurements of basal diameter and height taken manually with those derived from vegetation point clouds acquired with a Kinect. We also compared the plant volumes calculated with point-cloud-derived convex hulls against alternative solid shape approximations. Finally, we measured the dry biomass of the sample plants, and used the Kinect-derived structural variables to derive allometric relations.

2. Methods

2.1. Microsoft Kinect Sensor

Kinect captures synchronized color and depth images at a rate of 30 frames per second (fps) and with a field of view of $57° \times 43°$, using a RGB camera ($8\ bit$ VGA resolution with $640 \times 480\ pixel$) aligned with a depth imager. The Kinect depth sensor was designed by PrimeSense, and is composed of an IR laser projector and a monochrome $640 \times 480\ pixel$ IR CMOS sensor. Depth information is output with $11\ bit$ precision for each pixel [19]. Kinect's depth measurement principle differs from laser scanners such as LIDAR. LIDAR produces depth images by measuring the time of flight of individual laser pulses that sequentially scan the entire scene using motorized pan-tilt units. Kinect's laser projector illuminates the entire scene at once, using a diffraction grid that imposes a consistent pattern of speckles on the beam [20]. Depth is then calculated by correlation and triangulation between the laser pattern captured by the CMOS sensor and a reference pattern stored in the sensor's memory [20,21]. This acquisition method bypasses the need for moving parts, which reduces the sensor's weight, size, acquisition time, and power requirement.

Using external software, depth images can be converted and stored in three-dimensional representations called point clouds. Each point in the cloud is represented by a set of coordinates $P = \{x, y, z\}$, which defines its position in space. Point coordinates can also be extended to the form $P = \{x, y, z, R, G, B\}$, which includes the red, green, and blue components recorded by the aligned RGB camera, or $P = \{x, y, z, I\}$, which includes intensity information. Only point clouds with x, y, z coordinates were used in this study. Kinect-derived point clouds, which contain about 300,000 points for each frame, have a $3\ mm$ error along the horizontal axes. Error increases with distance from the sensor [20]; the depth error is $\pm 1\ cm$ at $2\ m$ and $\pm 7\ cm$ at $5\ m$ [20]. The optimal measurement range recommended by Microsoft is 1.2–3.5 m, though tests show Kinect is capable of measurements at 0.8–6.0 m [19]. Additional factors influence depth errors, including light conditions and target reflectivity. Bright light and high reflectivity reduce the laser pattern contrast on targeted surfaces, which creates gaps and outliers in the resulting point cloud [20], as also shown in our preliminary tests (see Section 3.1).

2.2. Acquisition, Processing and Analysis of Vegetation Point Clouds

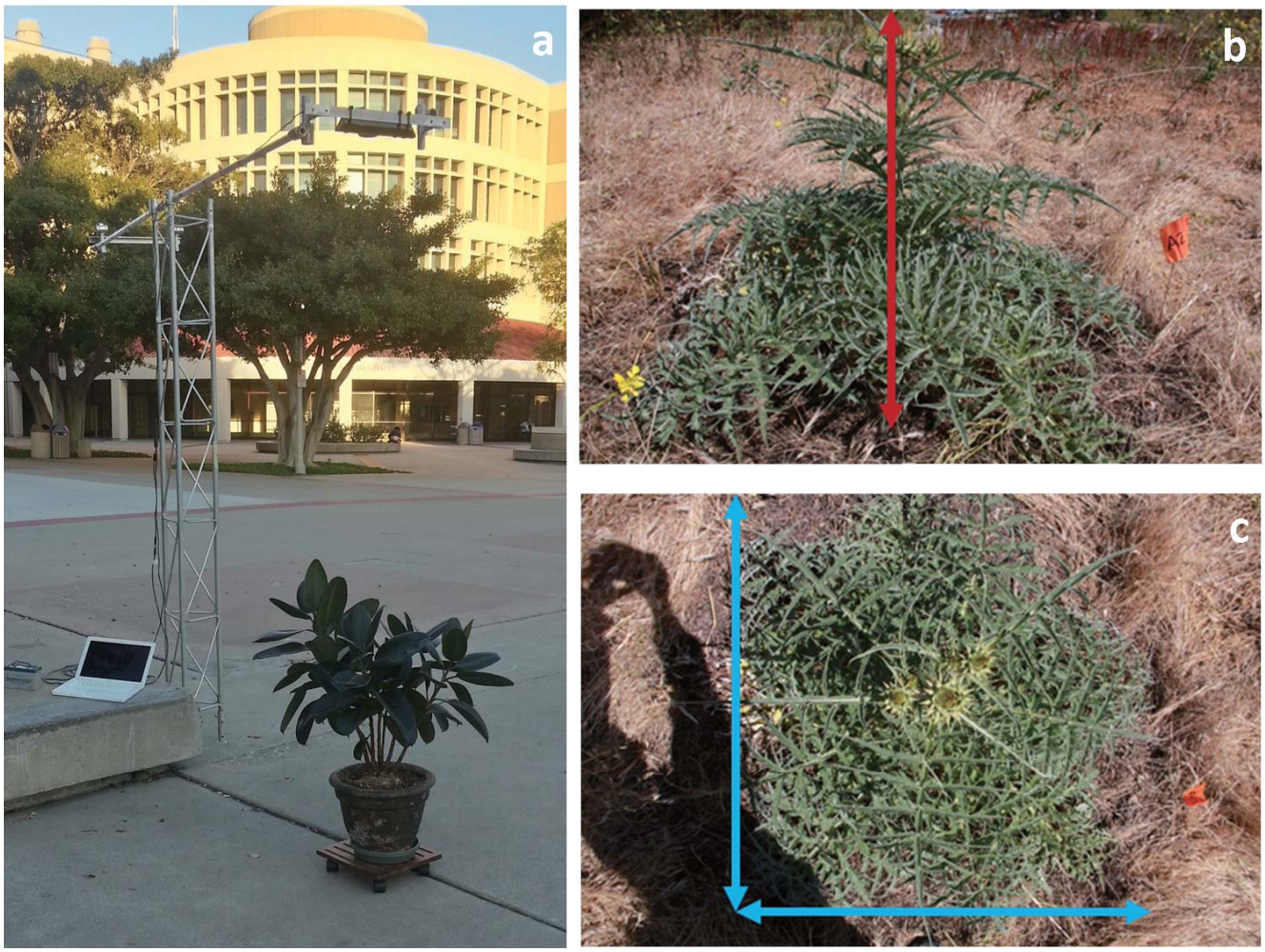
Acquisition, processing and analysis of point clouds were performed using tools and methods included in an open-source C++ library called Point Cloud Library (PCL, [22]). A detailed description of these methods is beyond the scope of this paper; full documentation is provided on the PCL website (www.pointclouds.org).

2.2.1. Point Clouds Acquisition

Point clouds were acquired from Kinect using the PCL OpenNI grabber, an I/O interface that is compatible with many devices. The acquisition software ran on a laptop computer with Linux Ubuntu 12.04 LTS. Two acquisition strategies were used in this paper: (*i*) "Tower Mode": The

Kinect was mounted on a 1.4 m long mast extending horizontally from a 3 m tall aluminum tower. The sensor was oriented to provide a nadir view of a plant (see Figure 1(a)). Acquisition from the tower setup was operated in single-shot mode: a single point cloud was recorded along with the corresponding raw infrared image. (*ii*) "Multi-angular Mode": The Kinect was manually moved 360 degrees around a plant, while a stream of point clouds was recorded. A complete 360 degrees scan of a plant yielded about 2000 point clouds and took up about 2 GB of disk space. Streamed multi-angular point clouds were then merged into a single cloud using the Kinfu PCL utility (Kinect Fusion, [23]). Kinfu does not rely on targets or user interaction to co-register point clouds. Because of its high computational needs, Kinfu must be run on computers equipped with Graphic Processors Units (GPU). If the acquisition machine is equipped with such a device, the co-registration can be performed in real time, rather than in "offline" mode on recorded streams.

Figure 1. (**a**) The "tower mode" experimental setup used for nadir measurements with Kinect (see Sections 2.3 and 2.4). (**b**) Side view and (**c**) nadir view of a wild artichoke plant recorded during the field test (see Section 2.4). Blue and red lines represent plant's basal diameter and height as measured manually with a ruler.

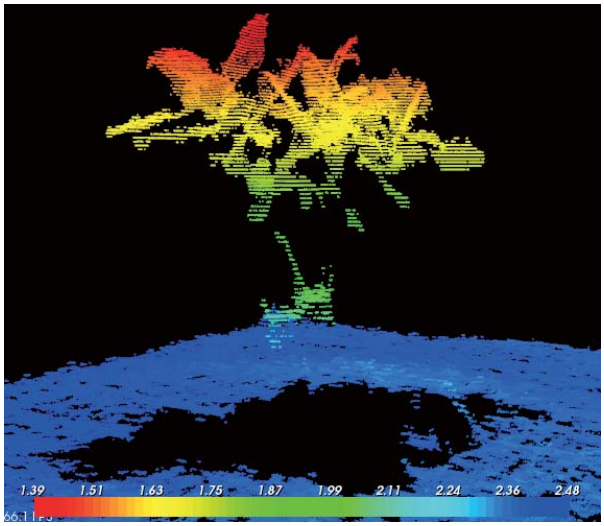


2.2.2. Canopy Structure from Point Clouds

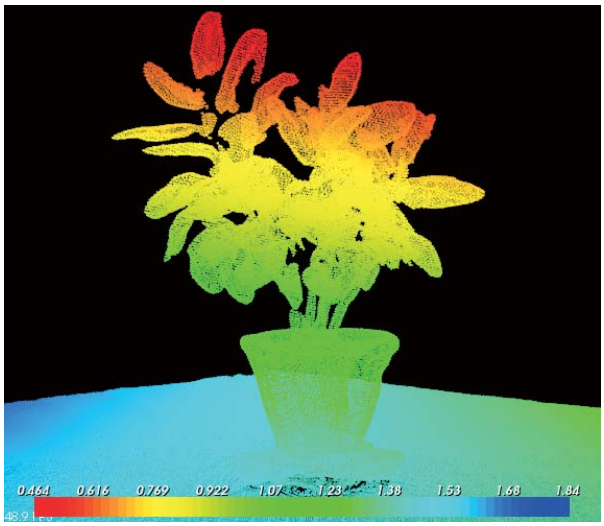
The point clouds were then processed following four steps: (i) Noise reduction and filtering to remove outliers, (ii) Delineation of individual plants and point extraction, (iii) Calculation of plant x, y, and z size, and (iv) Calculation of plant volume. All the post-processing and analysis were performed in the lab using a desktop computer equipped with a 1.5 Gb GPU running Linux Ubuntu 12.04 LTS.

1. Noise reduction and filtering to remove outliers: spurious individual points (outliers) were removed using statistical filters. The point clouds were then down-sampled to a 1 cm grid to reduce computational demand.
2. Delineation of individual plants and point extraction: single plants were initially isolated by visually determining the "box coordinates", *i.e.*, the x, y, z ranges occupied by a plant. Alternative methods for unsupervised plant detection were explored, including planar segmentation of the soil plane and Euclidean clustering. Unfortunately, these methods gave inconsistent results, especially when applied to point clouds acquired in the field (as in Section 2.4). Irregularity of the understory and point clouds that included multiple plants presented particular challenges.
3. Calculation of plant x, y, and z size: each point (P) is represented by x, y, z coordinates; plant size along each axes was calculated as $d_i = max(P_i) - min(P_i)$, where i can be x, y, or z, and P belongs to the isolated plant points subset.
4. Calculation of plant volume: there are several geometrical algorithms that can be used to calculate the volume of a vegetation point cloud. In principle, these methods should provide a much more accurate estimate of volume than would be possible with a solid shape approximation. Convex and concave hulls have proven especially well suited for this problem e.g., [13]. The convex (concave) hulls C of a set of points $Q = \{p_1, p_2, ..., p_n\}$ on a plane is *"the unique convex (concave) polygon whose vertices are points from Q and that contains all points of Q"* [24]. This definition can be extended to 3-dimensions, with C being a surface mesh instead of a polygon. The volume and structure of a plant should ideally be estimated using concave rather than convex hulls, since this will exclude the empty spaces between branches and leaves. Concave hulls estimation requires a complete cloud of each plant, and it can be applied only to co-registered, multi-angular point clouds. Hulls estimation from nadir-only point clouds can be problematic, because leaves and stems shade each other along the z axis, resulting in an incomplete representation of the plant between the larger parts of the crown and the soil. Convex hulls in nadir-only point clouds were estimated after the base points of the plant were determined. We used the plant's shade to define its base. Shade points were identified using a border recognition method based on depth values. This method returns three subsets of points corresponding to borders (shown in white in Figure 2(c)), shades (shown in grey in Figure 2(c)), and veil points.

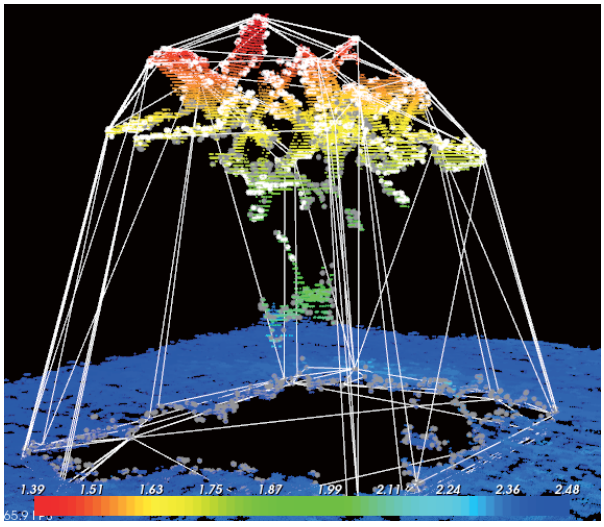
Figure 2. Comparison between nadir-only (a and c; collected using the "Tower Mode") and co-registered multi-angular (b and d; collected using the "Multi-angular Mode") point clouds. The upper panels show the point clouds viewed from an arbitrary horizontal perspective. The lower panels show convex (**c**) and concave (**d**) hulls superimposed as white lines. Panel (**c**) shows the shade (grey) and border (white) points used to define the convex hulls. Point cloud color scale is distance (meters) from the sensor at nadir.



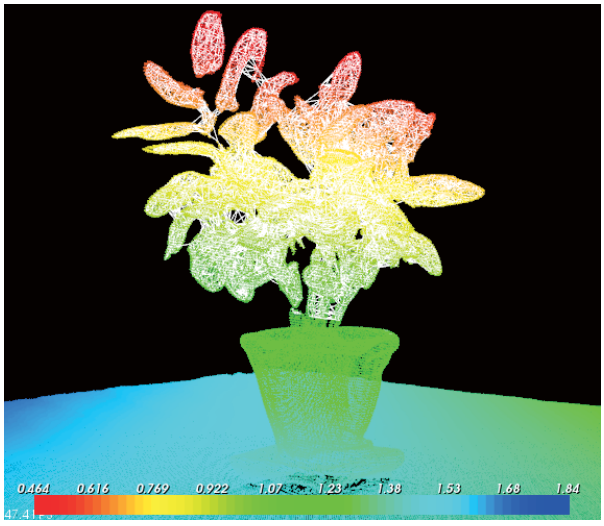
(a) Nadir-only point cloud



(b) Co-registered multi-angular point clouds



(c) Convex hulls with shade and border points



(d) Concave hulls

2.3. Preliminary Tests on Kinect

We tested Kinect under a range of conditions to: (a) evaluate the effects of light conditions on the measurements, and (b) compare the information obtained by a fixed-angle looking Kinect (*i.e.*, clouds using the "Tower Mode") with that obtained by co-registered acquisition (*i.e.*, clouds using the "Multi-angular Mode").

We carried out these tests on a potted rubber fig tree (Ficus elastica), which was about 1.1 m tall (Figure 1(a)). Ficus elastica's large, smooth leaves are particularly easy to detect using Kinect. The plant was placed on a smooth concrete surface that had an infrared reflectance that differed markedly from that of the plant's leaves. Images were acquired using the "Tower Mode" configuration at 30 minutes intervals beginning 1.5 hours before sunset and ending 1.5 hours after sunset. Additionally, the sensor was detached from the tower and used in "Multi-angular Mode" at night to obtain complete point clouds of the plant.

2.4. Testing Kinect in the Field

We also evaluated Kinect's performance and limitations in a more natural grassland setting. A complete assessment of Kinect's reliability in providing canopy structure information in the field might make use of a direct comparison with corresponding ground-based LIDAR measurements. Unfortunately, we lacked access to such a device and consequently compared Kinect's results with ruler-based measurements of plant size. Measuring plant dimensions using rulers is a common practice in field ecology, though the resulting data may inadequately account for a plant's complex shape. Our field comparison was not intended to quantify the absolute accuracy of the plant dimensions measured with the Kinect. In fact, it is likely the Kinect observations are far more accurate than those made with a ruler.

The field site was located on the University of California, Irvine campus (coordinates: $33.640102°N$, $117.845314°W$). The area is characterized by a Mediterranean climate; most of the precipitation falls in winter and the summer is reliably dry. The study was conducted in early May, 2011, which was at the end of the growing season. The site was a mosaic of annual grassland and small shrubs, though the grasses had fully senesced at the time of the study. Two large plant species were abundant and reaching the peak of their growing season during our study: Cynara cardunculus (wild artichoke; Figure 1(b,c)) and Picris echioides (bristly ox-tongue). Because of their abundance, their ranges of size, and the contrast the plants presented with the surrounding dry vegetation, these two species were selected for investigation.

Eight plants of Cynara cardunculus and four plants of Picris echioides that ranged in size were selected and marked. We acquired several point clouds for each study plant at night, when the illumination conditions minimized the measurement error. All point clouds acquisitions were done using the "Tower Mode", with the tower carried by hand to each selected plant. We subsequently measured the basal diameter and height of each plant using a ruler. Both species had a conic shape, and their basal diameters were measured along two orthogonal axes at ground level (one of the two being the maximum diameter, see Figure 1(b,c)). The two basal measurements were subsequently averaged for analysis and comparison. Plant volume was calculated using the manual measurements of basal diameter (d) and height (h) and commonly used solid shapes including a rectangular box ($V = d^2 \cdot h$), cylinder ($V = d^2 \cdot \pi \cdot h/4$), cone ($V = d^2 \cdot \pi \cdot h/12$), parabolic cone ($V = d^2 \cdot \pi \cdot h/8$), and elliptical cone ($V = d^2 \cdot \pi \cdot h/6$). We subsequently harvested the plants and measured their dry biomass.

3. Results

3.1. Preliminary Tests Using Kinect Our preliminary tests showed that light condition and target reflectance influence the depth measurement quality. The series of images in Figure 3 show how depth range increases progressively with darker light conditions. The poor performance obtained under sunlight (see Figure 3(a,b)) was apparently due to the high IR reflectivity of foliage. The reflected IR radiation from sunlight apparently reduced the contrast of the Kinect's laser pattern over vegetated surfaces (Figure 3(a)), resulting in point clouds with many missing points (Figure 3(b)). Nocturnal measurements were needed to obtain well-defined point clouds of the rubber fig plant (Figure 3 (e,f)). The background concrete surface, which is the furthest target from the sensor, was detected under all light conditions, though with some missing points in brighter sunlight. The concrete is less reflective in the IR (Figure 3(a,c)), which reduces errors and data gaps caused by sunlight contamination.

Co-registration of multi-angular point clouds gave excellent results in our preliminary test (Figure 2(a,b)). Co-registered point clouds provided a complete representation of the study plant, allowing detailed modeling of the canopy structure using concave hulls (compare Figure 2(d) and Figure 2(c)). Additional tests (not shown in this paper) showed that multi-angular acquisition on plants with smaller-leaves plants and in locations with a complex understory produced less consistent results.

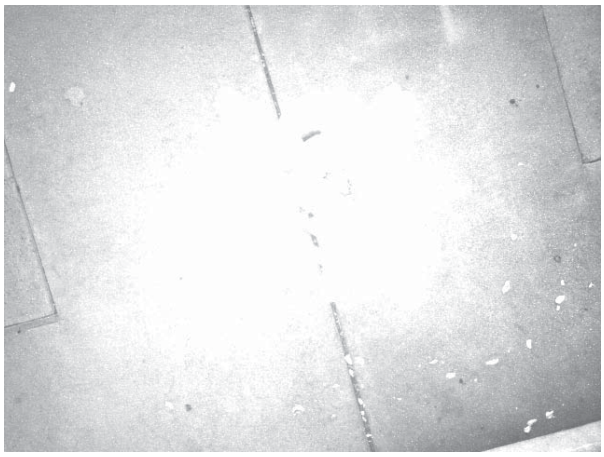
3.2. Testing Kinect in the Field

Comparisons of manual and Kinect-derived measurements of vertical and horizontal plant dimensions are shown in Figure 4. Results showed good agreement for all measurements, especially those on Cynara cardunculus. The Normalized Root-Mean-Squared Error calculated from the identity line ($NRMSE_{y=x}$) ranged from 2.7% to 19.1%, with the lowest error corresponding to Cynara cardunculus base measurements and the highest corresponding to Picris echioides base measurements. The stronger agreement obtained for Cynara cardunculus probably reflects the better defined shape of these plants, which simplified the manual measurements.

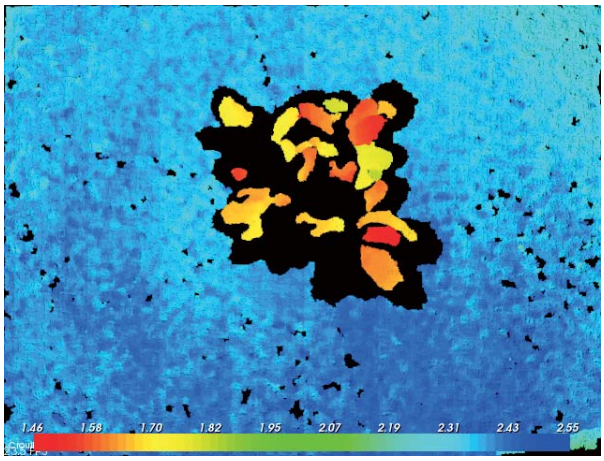
Plant volume estimated from the manual field measurements using box, cylinder, cone, parabolic cone, and elliptical cone solid models was compared with that obtained from the Kinect using convex hulls (Figure 5). $RMSE_{y=x}$ values ranged from 1.0% to 106.1%. The elliptical cone provided the best shape approximation for both plant species, with a $RMSE_{y=x}$ of 1.0% for Cynara and 2.1% for Pychris. Box and cylinder models, which are frequently used for canopy modeling, had the largest $RMSE_{y=x}$ values (from 57% to 106%).

Dry biomass was compared with the Kinect-determined heights, diameters, and volumes to derive allometric relationships (see Figure 6). The various measures of plant size were well correlated with plant biomass through a logarithmic function of the form: $y = A + B \cdot \log(x + C)/(log(D) + 1)$. The resulting coefficients of determination (r^2) ranged from 0.97 to 1.0.

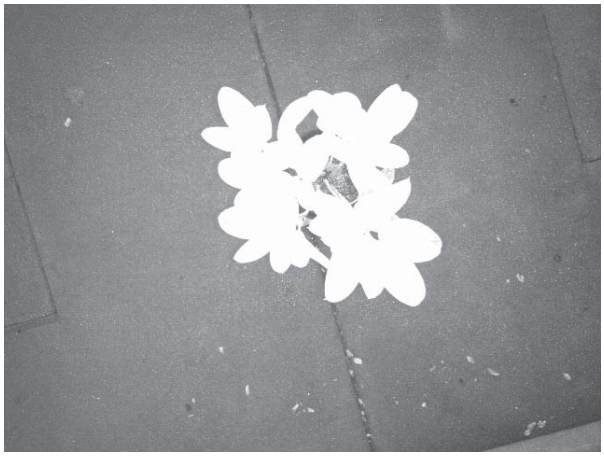
Figure 3. Comparison between raw infrared output from Kinect (**a**, **c**, **e**) and corresponding point clouds (**b**, **d**, **f**) in different light conditions. All images are for a potted rubber tree plant that was about 1.10 m tall (*i.e.*, Figure 1(a)). Point cloud color scale is distance (meters) from the sensor at nadir.



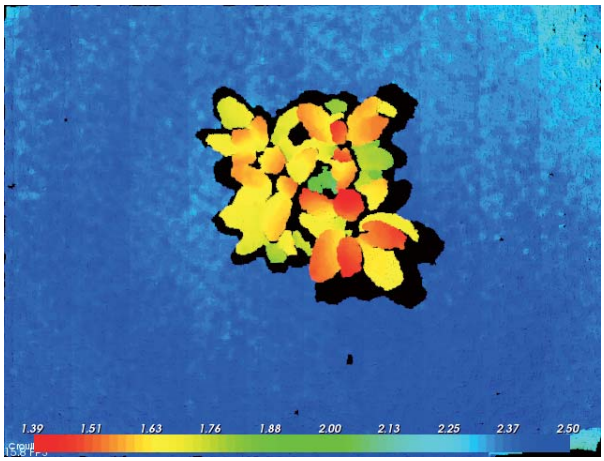
(a) Early afternoon Infrared Raw Image



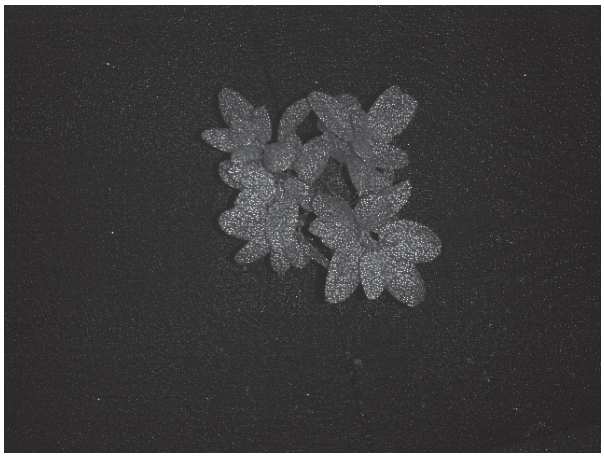
(b) Early afternoon Point Cloud



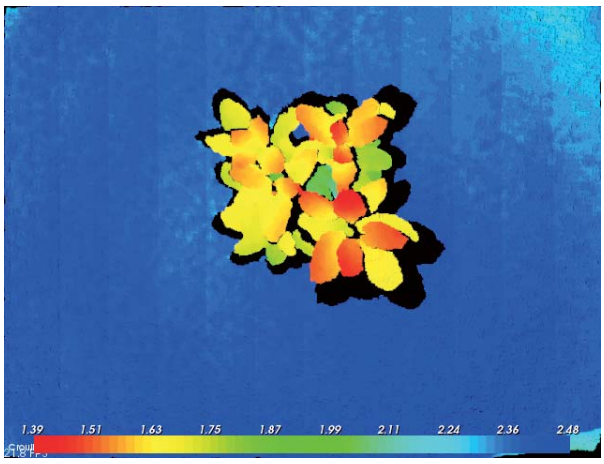
(c) Late afternoon Infrared Raw Image



(d) Late afternoon Point Cloud



(e) Night Infrared Raw Image



(f) Night Point Cloud

Figure 4. Direct comparison of manual and Kinect measurements of plant basal diameter ("Base"; the average of x and y measurements) and plant height ("Height"). Solid lines represent least squares linear fits.

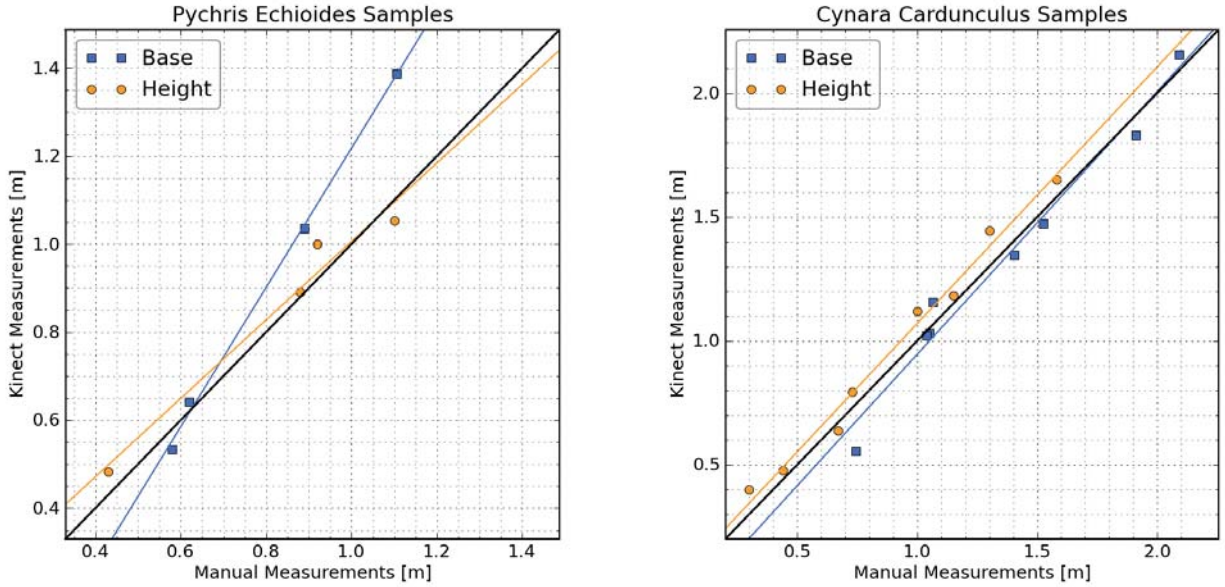


Figure 5. Direct comparison of volumes obtained from convex hulls and those derived from manual measurements with various solid shape approximations. Solid lines represent least squares linear fits.

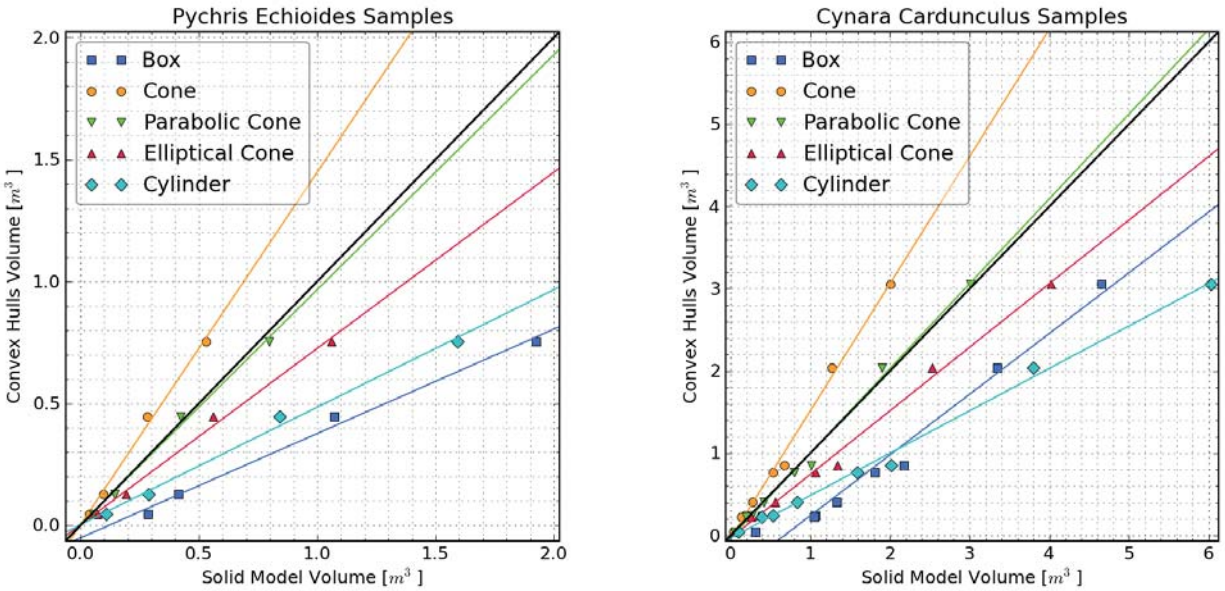
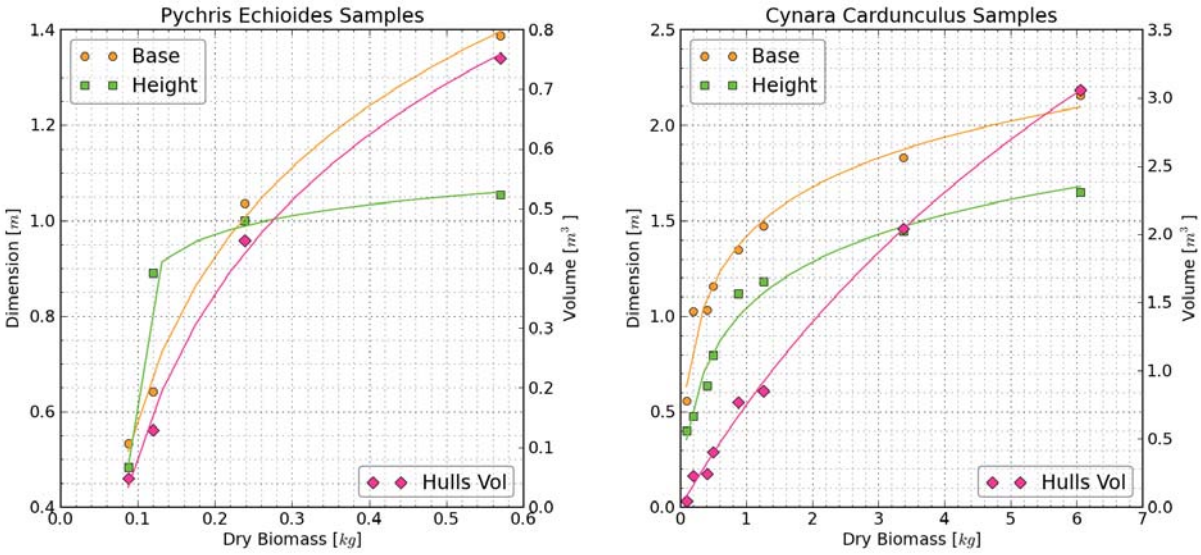


Figure 6. Allometric relations between Kinect-derived dimensions (base and height) and dry biomass measurements. Solid lines represent least squares logarithmic fits of the form $y = A + B \cdot \log(x + C)/(log(D) + 1)$.



4. Discussion

4.1. Strengths and Limitations of Kinect Sensors

The Microsoft Kinect sensor combined with the Point Cloud Library provided measurements of plant height and base diameter that agreed well with the corresponding manual measurements. Similarly, the three-dimensional convex hulls estimations provided useful information on plant volume and shape, which are relevant to biophysical canopy modeling and biomass estimation. The structural information retrieved with Kinect was used to obtain allometric relations for two species of plants. The light weight, small size, fast acquisition time, low power requirement, and cost make the Kinect a promising tool for rapid field surveys of canopy structure, especially in small-statured vegetation.

It is important to acknowledge the Kinect sensor's limitations for vegetation sensing. The Kinect was unable to measure vegetation in daylight, and the measurement range was limited to a few meters. Nighttime measurements may be difficult in field sites with limited accessibility. Measurements from a fixed location, such as the top of a flux tower, circumvent this problem, but would be limited spatially by the Kinect's measurement range.

The results for images collected in "Tower Mode" were less consistent in situations with densely packed vegetation or with complex, live ground cover. The "Multi-angular Mode" is particularly promising and allows the creation of point clouds with remarkable detail (Figure 2(d)), but smaller leaves and dense vegetation can create problems during the co-registration process.

4.2. Perspectives for the Future

The Kinect's limitations for measuring vegetation in the field are not surprising; the Kinect was designed for video-gaming indoors rather than ecologic observations in the field. Nonetheless, the Kinect

is already useful for some ecological applications in its present forms, and the measurement principle opens possibilities for designing new sensors dedicated to vegetation characterization.

New vegetation sensors based on the Kinect's design could use a more powerful laser source to increase the measurement range and allow characterization of taller vegetation. This might also improve the daylight observations, especially if the projector's wavelength could be selected to enhance the contrast over vegetated targets within sunlit environments. The development of a depth sensor for daylight use would allow retrieval of images from the aligned RGB camera; the RGB information associated with each point could then be used for phenological studies e.g., [25].

Alternatively, the Kinect approach might be used to determine plant spectral reflectance along with the 3-dimensional structure. The intensity of the laser source is known, and reflectance can be calculated once the intensity of the reflected speckled pattern is retrieved at each frame. In principle, multiple illumination wavelengths could be utilized, allowing the calculation of spectral vegetation indices. The use of active rather than passive illumination for multi-spectral reflectance measurements in the field has been tested in recent studies e.g., [26,27]. Active illumination has the advantage of eliminating dependence on weather or light conditions. Moreover, the source-sensor geometry is fixed and known, avoiding the need to consider the Bidirectional Reflectance Distribution Function (BRDF) [26], though discrete laser returns still depend on factors such as vegetation structure, geometry, and angles of incidence.

Improvements could be made to the acquisition procedure, the software and the data processing. Algorithms for acquisition and co-registration of multi-angular point clouds could be improved to allow acquisition in the field, where small plants or complex understory currently create errors. Multi-angular measurements in the field may be readily acquired by hand during field campaigns, or using pan-tilt units (e.g., on flux towers), or with multiple sensors, or with sensors flown at low altitude e.g., [28–30]. More accurate algorithms for quantifying structural parameters such as Leaf Area Index and Leaf Angle Distribution could be developed starting from co-registered point clouds.

5. Conclusions

The Microsoft Kinect shows potential for measuring the structure of small plants in the field, but its design currently limits the range of possible applications for ecological studies. Nevertheless, the development of software and inexpensive consumer electronics is opening exciting possibilities for ecological applications. Further research on these technologies is inevitable given the rapid development of inexpensive and easy-to-use sensors. This, in turn, can increase the spatial and temporal coverage and availability of ground data, providing better information for validation of remote sensing measurements and improving biophysical canopy modeling (e.g., Figure 2(d)).

Acknowledgments

NASA and DOE provided research support (grant numbers NNX10AL14G and 0011182). Greg Jenkins and family supported this study with a graduate student fellowship.

References

1. Brunner, A. A light model for spatially explicit forest stand models. *Forest Ecol. Manage.* **1998**, *107*, 19–46.
2. Pielke, R.A.I.; Avissar, R. Influence of landscape structure on local and regional climate. *Landscape Ecol.* **1990**, *4*, 133–155.
3. Rotenberg, E.; Yakir, D. Distinct patterns of changes in surface energy budget associated with forestation in the semiarid region. *Glob. Change Biol.* **2011**, *17*, 1536–1548.
4. Lee, X.; Goulden, M.L.; Hollinger, D.Y.; Barr, A.; Black, T.A.; Bohrer, G.; Bracho, R.; Drake, B.; Goldstein, A.; Gu, L.; *et al.* Observed increase in local cooling effect of deforestation at higher latitudes. *Nature* **2011**, *479*, 384–387.
5. Bonan, G.B. Forests and climate change: Forcings, feedbacks, and the climate benefits of forests. *Science* **2008**, *320*, 1444–1449.
6. Lu, D. The potential and challenge of remote sensing based biomass estimation. *Int. J. Remote Sens.* **2006**, *27*, 1297–1328.
7. Powell, S.L.; Cohen, W.B.; Healey, S.P.; Kennedy, R.E.; Moisen, G.G.; Pierce, K.B.; Ohmann, J.L. Quantification of live aboveground forest biomass dynamics with Landsat time-series and field inventory data: A comparison of empirical modeling approaches. *Remote Sens. Environ.* **2010**, *114*, 1053–1068.
8. Bergen, K.M.; Goetz, S.J.; Dubayah, R.O.; Henebry, G.M.; Hunsaker, C.T.; Imhoff, M.L.; Nelson, R.F.; Parker, G.G.; Radeloff, V.C. Remote sensing of vegetation 3-D structure for biodiversity and habitat: Review and implications for lidar and radar spaceborne missions. *J. Geophys. Res.* **2009**, *114*, 1–13.
9. Seidel, D.; Beyer, F.; Hertel, D.; Fleck, S.; Leuschner, C. 3D-laser scanning: A non-destructive method for studying above-ground biomass and growth of juvenile trees. *Agr. Forest Meteorol.* **2011**, *151*, 1305–1311.
10. Henning, J.; Radtke, P. Ground-based laser imaging for assessing three dimensional forest canopy structure. *Photogramm. Eng. Remote Sens.* **2006**, *72*, 1349–1358.
11. Naesset, E.; Gobakken, T.; Nasset, E. Estimation of above- and below-ground biomass across regions of the boreal forest zone using airborne laser. *Remote Sens. Environ.* **2008**, *112*, 3079–3090.
12. Zhao, K.; Popescu, S.; Nelson, R. Lidar remote sensing of forest biomass: A scale-invariant estimation approach using airborne lasers. *Remote Sens. Environ.* **2009**, *113*, 182–196.
13. Morsdorf, F.; Meier, E.; Allgöwer, B.; Nüesch, D. Clustering in airborne laser scanning raw data for segmentation of single trees. *Int. Arch. Photogramm. Remote Sens. Spat. Inform. Sci.* **2003**, *34*, 27–33.
14. Jochem, A.; Hollaus, M.; Rutzinger, M.; Höfle, B. Estimation of aboveground biomass in alpine forests: A semi-empirical approach considering canopy transparency derived from airborne LiDAR data. *Sensors* **2011**, *11*, 278–295.
15. Wang, Y.; Weinacker, H.; Koch, B. A LiDAR point cloud based procedure for vertical canopy structure analysis and 3D single tree modelling in forest. *Sensors* **2008**, *8*, 3938–3951.

16. Baldocchi, D.; Falge, E.; Gu, L.; Olson, R.; Hollinger, D.; Running, S.; Anthoni, P.; Bernhofer, C.; Davis, K.; Evans, R.; Fuentes, J.; Goldstein, A.; Katul, G.; Law, B.E.; Lee, X.; Malhi, Y.; Meyers, T.; Munger, W.; Oechel, W.; Paw U, K.T.; Pilegaard, K.; Schmid, H.P.; Valentini, R.; Verma, S.; Vesala, T.; Wilson, K.; Wofsy, S. FLUXNET: A new tool to study the temporal and spatial variability of ecosystem-scale carbon dioxide, water vapor, and energy flux densities. *Bull. Amer. Meteorol. Soc.* **2001**, *82*, 2415–2434.
17. Gamon, J.; Rahman, A.; Dungan, J.; Schildhauer, M.; Huemmrich, K. Spectral Network (SpecNet)-What is it and why do we need it? *Remote Sens. Environ.* **2006**, *103*, 227–235.
18. Balzarolo, M.; Anderson, K.; Nichol, C.; Rossini, M.; Vescovo, L.; Arriga, N.; Wohlfahrt, G.; Calvet, J.C.; Carrara, A.; Cerasoli, S.; *et al.* Ground-based optical measurements at european flux sites: A review of methods, instruments and current controversies. *Sensors* **2011**, *11*, 7954–7981.
19. Livingston, M.A.; Sebastian, J.; Ai, Z.; Decker, J.W. Performance Measurements for the Microsoft Kinect Skeleton. In *Proceedings of the IEEE Virtual Reality Short Papers and Posters*, Costa Mesa, CA, USA, 4–8 March 2012; pp. 119–120.
20. Khoshelham, K. Accuracy Analysis of Kinect Depth Data. In *Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Calgary, Canada, 29–31 August 2011.
21. Freedman, B.; Shpunt, A.; Meir, M.; Yoel, A. Depth Mapping Using Projected Patterns. US Patent 8,150,142, 6 September 2007.
22. Rusu, R. 3D is Here: Point Cloud Library (PCL). In *Proceedings of the IEEE International Conference on Robotics and Automation*, Shanghai, China, 9–13 May 2011; pp. 1–4.
23. Izadi, S.; Kim, D.; Hilliges, O. KinectFusion: Real-Time 3D Reconstruction and Interaction Using a Moving Depth Camera. *In Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, Santa Barbara, CA, USA, 16–19 October 2011.
24. De Berg, M.; Cheong, O.; Van Kreveld, M.; Overmars, M. *Computational Geometry. Algorithms and Applications*; 3rd ed.; Springer-Verlag: Heidelberg, Germany, 2008.
25. Richardson, A.; Jenkins, J. Use of digital webcam images to track spring green-up in a deciduous broadleaf forest. *Oecologia* **2007**, *152*, 323–334.
26. Fitzgerald, G.J. Characterizing vegetation indices derived from active and passive sensors. *Int. J. Remote Sens.* **2010**, *31*, 4335–4348.
27. Puttonen, E.; Suomalainen, J.; Hakala, T.; Räikkönen, E.; Kaartinen, H.; Kaasalainen, S.; Litkey, P. Tree species classification from fused active hyperspectral reflectance and LIDAR measurements. *Forest Ecol. Manage.* **2010**, *260*, 1843–1852.
28. Dandois, J.P.; Ellis, E.C. Remote sensing of vegetation structure using computer vision. *Remote Sens.* **2010**, *2*, 1157–1176.
29. Wallace, L.; Lucieer, A.; Watson, C.; Turner, D. Development of a UAV-LiDAR system with application to forest inventory. *Remote Sens.* **2012**, *4*, 1519–1543.

30. Rosnell, T.; Honkavaara, E. Point cloud generation from aerial image data acquired by a quadrocopter type micro unmanned aerial vehicle and a digital still camera. *Sensors* **2012**, *12*, 453–480.

毕业论文（设计）文献综述和开题报告考核

一、对文献综述、外文翻译和开题报告评语及成绩评定：

成绩比例	文献综述 占（10%）	开题报告 占（20%）	外文翻译 占（10%）
分 值			

开题报告答辩小组负责人（签名）

年　　月　　日