2017

Extracting DBH measurements from RGB photo images

FACULTY OF FORESTRY AND ENVIRONMENTAL MANAGEMENT

WANG HAO-ZHOU

UNIVERSITY OF NEW BRUNSWICK

Extracting DBH measurements from RGB photo images

WANG Hao-Zhou

This thesis is submitted in partical fulfillment

of the requirements for the degree of

Bachelor of Forestry

At the University of New Brunswick

April 4th, 2017

Abstract

Forest inventory requries significant finanicial investment and is labor intensive. Despite efforts of most forest management agencies, field inventory is often quite inefficient with final sample proportions less than 1%. In addition, measurement results are influenced by many factors, and correcting errors often requires revisiting field sites and/or complex rules that may introduce bias. Foresters have always searched for methods to obtian forest parameters, such as diameter at breast height (DBH), more efficiently. While many tools were developed over the years, most are still time consuming, and all still require field checking to correct any potential measurement errors. The availability of essentially free digital images posses a permanent record of trees that can be remeasured without having to return to the field. This report presents one method to extract DBH measurements of trees from digital images using a digital camera and visible laser lines. Both the distance from camera and DBH results are validated using field measurements and show a good accuracy: 41.5% of DBH measurements were withing ± 0.5 cm of field measured DBH and 25.1% were within ± 0.5 cm to 1.0 cm. An open source software package called ImageDBH developed in Python enables users to obtain DBH directly from the digital images.

Contents

List of figures

Acknowledgement

This project would not be accomplished without the help of some individuals. Funding for this project was provided in part by an NSERC Discovery Grant (RGPIN 04280). We also thank Julie Henderson and Elizabeth McGarrigle for their assistance in helping collect field data and images. I would like to thank my advisor John A. Kershaw who is a professor in the Forestry and Environment Management Faculty for all the ideas and instructions. My friend Zhi-Jie HOU, a software engineer in Nanjing University of Posts and Telecommunications, Jiangsu, China, who gave me some guidance for Python programming and GUI making. And finally my girlfriend, Ya-Quan CHANG, without whose support and encouragement I could not have enough passion and courage to overcome difficulities that I met in this project.

1. Introduction

Diameter at breast height (DBH) is an important parameter for estimating forest volume and biomass in the forest inventories (Kershaw et al. 2016), In addition to volume or biomass, DBH is critical to estimating wildlife habitat quality (McTague and Patton 1989, Morrison et al. 1992), estimating leaf area (Larsen and Kershaw 1990, Jonckheere et al. 2004). And developing appropriate silvicultural treatments (Curtis 1970, McCarter and Long 1986). Therefore, it is important that forest inventory crews measurere DBH accurately and efficiently. The current methods of measureing DBH are divided into two categories: direct measurement and indirect measurement. The traditional direct methods include the use of diameter tapes, calipers, and the Biltmore stick (Kershaw et al. 2016). All of these techniques are labor intensive, have low field efficiency, and are subject to much inter-observer errors and variations (Clark et al. 2000b). On very steep terrain or with very large trees or trees that buttress, DBH is difficult to measure and often is visually estimated (Xiaodong and Zhongke 2015).

There are many ways for indirect measurement/estimation of DBH. First, regression models could be used to predict DBH from stumps (Westfall 2010) or from tree height (Brandeis et al. 2009), but regression models require strong field data and can possible be biased due to repeated measurements (Sullivan and Reynolds 1976). Laser technology has resultsed in a number of tools including: laser dendrometers (Skovsgaard et al. 1998); hand-held laser survey instruments ; ground-based laser scanners (Strahler et al. 2008); and airborne LiDAR scanners (Næsset 2002). All of these instruments either have direct diameter measurement/estimation processes or have had processes, such as regression-based estimation, built around the output from these devices. The major shortcoming of these devices is that they are very expensice, and often require a significant amount of office time for post-processing the data.

A relatively inexpensive 2D laser scanner showed promising results in lab conditions (Ringdahl et al. 2013), and was subsequently verified outdoors in the field (Zheng et al. 2012, 2014). Even with optimized geometric algorithms (Wang Yaxiong et al. 2016), this device is still impractical for large-scale applications. However, low-cost digitial cameras

are becoming increasingly popular in field survey applications. Decourt (1956) proposed the use of photography for angle gauge sampling. Stewart et al. (2004) further applied this concept to digital images, and Wang and Feng (2006) developed a field server to automate the measurements. With the development of computer vision, an algorithm-complex stereoscopic imaging method was applied to photos taken from different angles (Hapca et al. 2007, Xiaodong and Zhongke 2015), coupling five cameras for increased accuracy (Forsman et al. 2012). Still the most economic way is a single image measurement which is hard for distance calculations without the aid of benchmarks (Wang and Feng 2006, Xiaodong and Zhongke 2015) or a laser light source (Pengle et al. 2013) on single trees. However, dealing with photos of mutiple trees is still a problem, and there have been no known publications addressing this issue. Additionally, some of the research described above was conducted in the laboratory with no field verification. Even those studies with field verification lack sufficient amounts of data (less than 50 samples) to generalize results and develop wide-range field applications.

In this study we propose a low-cost set up (digital camera and linear laser pen) with common geometric algorithms to extract required measurements. An open source software package called ImageDBH was developed in python to aid users in obtaining the required measurements.

2. Methods and materials

2.1 The Study Area Overview

The study area is a large research plot (50m×50m) located in the University of New Brunswick's Noonan Research Forest (N 45° 59' 58", W 66° 25' 20"). The elevation is approximately 100m above sea level. Summer precipitation is about 200 - 250mm, while the average summer temperatures range between 18 - 20℃ (Roberts and Zhu 2002). The field site was established in 2008. Wooden stakes were carefully surveyed in along a 10m grid across the 50 m by 50 m study area. Tree locations were triangulated from measurements from these wooden stakes (Figure 1). Species of each tree was identified, and diameter at breast height (DBH, 1.3m), diameter at 3m (D3M), diameter at 5m (D5M), and total height

were measured on each tree ⩾ 6.0 cm DBH. DBH, D3M, and D5M were marked on each tree using different colored spray paint. There were a total of 281 trees measured in 2008.

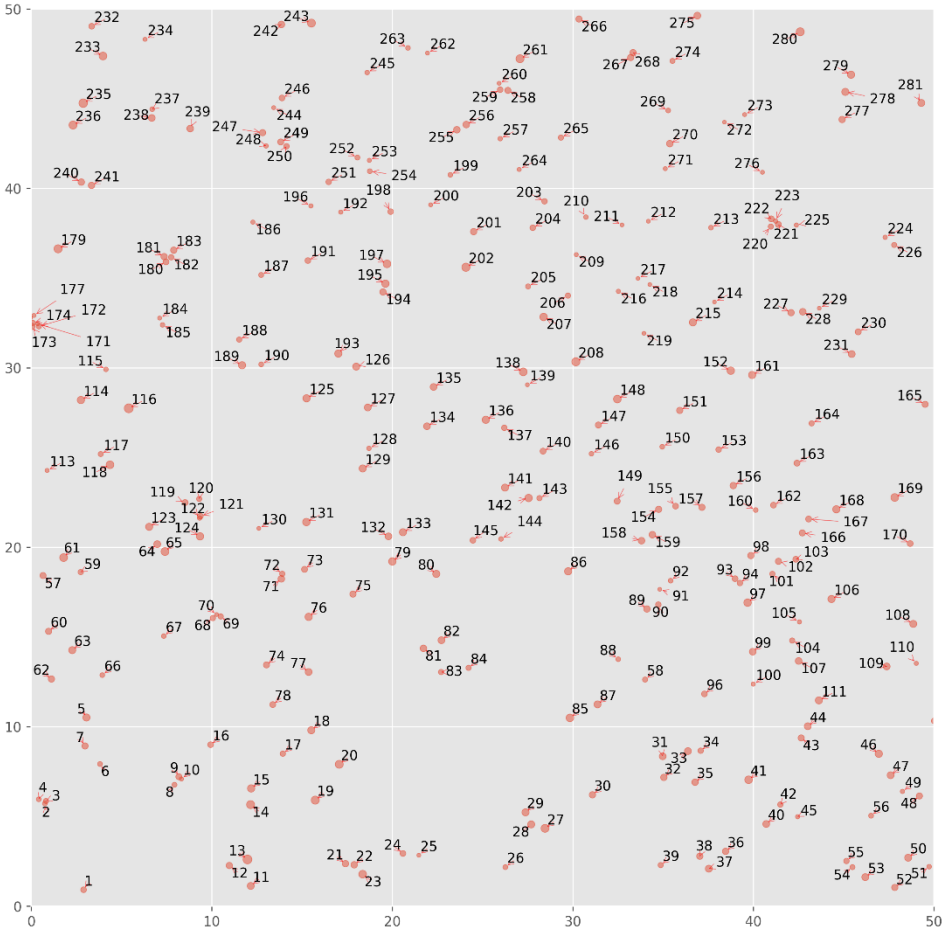


Figure 1:The Location of trees in the sample area.

The tree species include eastern hemlock (EH, *Tsuga canadensis*), red maple (RM, *Acer rubrum)*, eastern white cedar (EC, *Thuja occidentalis*), spruce (SP, *Picea sp.*), yellow birch (YB, *Betula alleghaniensis*), white birch (WB, *Betula papyrifera*), balsam fir (BF, *Abies balsamea*), white ash (WA, *Fraxinus americana*), and mountain ash (MA, *Sorbus americana*).

2.2 Instrument and Equipment

A Kaden spherical panorama tripod head was modified so that two laser generators could be mounted and project parallel horizontal lines (Figure 2). The upper laser generator was located 30 cm above the center of the image plane and the lower laser was located 15 cm below the center of the image plane.

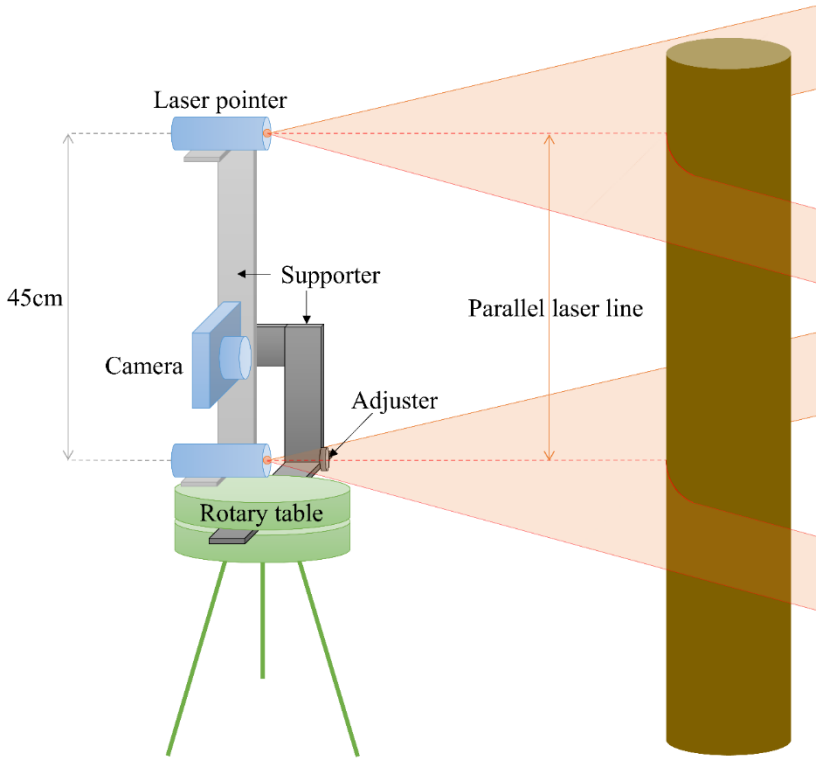


Figure 2: The picture taking equipment.

There are three components to the close-range digital photogrammetric system: the camera/laser system (blue), the spherical support arm (grey), and panoramic rotary base (green). For this study a Canon EOS 5D Mark II was used. The lasers were two 100mW, 665nm red lasers with a linear collimeter lens to project a horizontal line. At each of the 16 interior stakes, 14 pictures were taken such that a 360° panoramic image could be stiched. In this study, only the original individual images were used. For all trees that could be identified by tree number on the image, DBH measurements were extracted. Because some trees appeared in multiple images, those trees have multiple DBH measurements. Images for one stake were lost, and several trrees were not clearly visible from any of the camera locations.

2.3 Algorithm

1) The Geometry

Using the principles of similar triangles and perspective geometry, the DBH is extracted from the images. Figure 3.b shows a 3D model of the geometric system. The geometric relationships between the camera, the laser lines, and the tree within the image frame are depicted. Figure 3.c shows the projection onto the image plane. Because the focal point is located at a point of trisection rather than bisection, the upper laser line is not congruent with the lower laser line. Uc and

Dc are the vertices of these curves while the other points are boundary points. Because of the cylindrical trunk (Figure 3.a), the boundary points (L, R) are farther from the focal point than the center points (C). The projected vertical distance between two boundary points should be smaller than projected distances between the center points according to perspective geometry. Because the distance between the laser lines is known, this relationship is the key geometric identity used in all calculations required to extract DBH measurements.

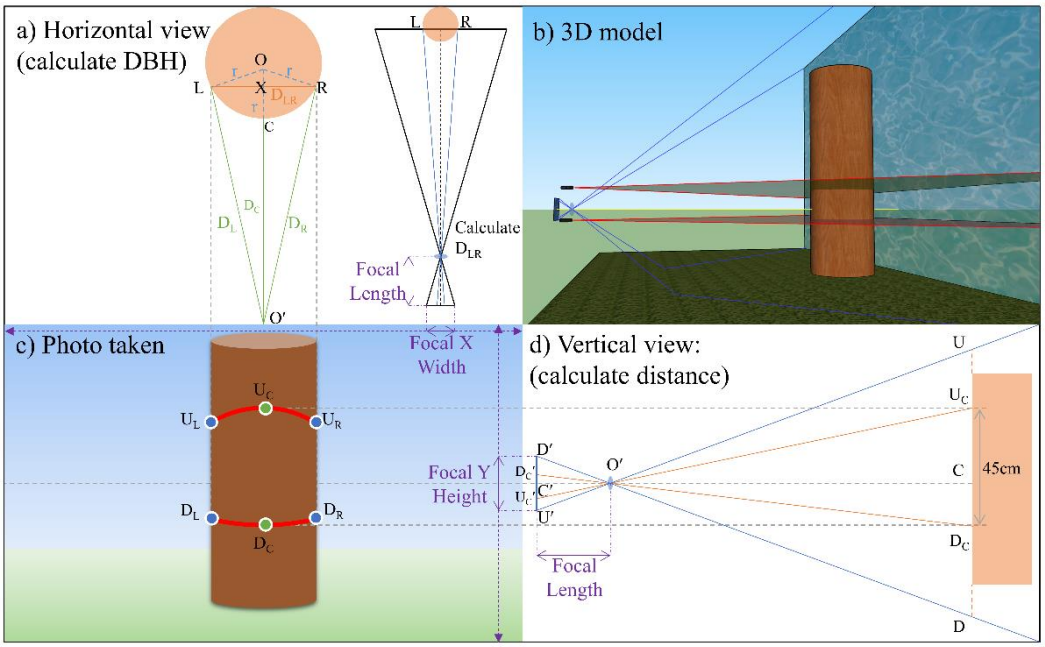


Figure 3: Geometry of DBH measurement extraction: a) horizontal projection; b) 3D model of geometric projection; c) ideal image projection; d) vertical projection.

2) Distance calculation

Figure 3.d is a vertical view of the geometric projection. Here, the distance between camera and tree is calculated using similar triangles. There are two sets of similar triangles, $\triangle O'D'U' \backsim \triangle ODU$ and $\triangle O'Dc'Uc' \backsim \triangle ODcUc$. $O'C'$ is focal length while $D'U'$ is focal Y height. Both of these attributes are available from XML data stored within each photo. The length of $Dc'Uc'$ is determined from the ratio of pixels in the photo.

From $\triangle ODcUc \backsim \triangle O'Dc'Uc'$, $O'C$ is determined using:

$$\frac{O'C}{f} = \frac{DcUc}{Dc'Uc'} \quad \textbf{(1)}$$

where, f is the focal length of the camera, DcUc is the distance between the two laser lines (45cm is our set up). Then Dc'

Uc′ is determined using:

$$\frac{\mathrm{Dc'Uc'}}{Fy} = \frac{n}{N} \tag{2}$$

where, *Fy* represents D′U′, the height of photo (cm). *n* is the number of vertical pixels between the two laser lines. *N* is the total number of vertical pixels along the photo's Y axis. Simultaneously solving equations (1) and (2), yields the following equation for O′C:

$$\mathrm{O'C} = \frac{45 \cdot f \cdot N}{Fy \cdot n} \tag{3}$$

where *f*, *Fy*, and *N* are constants obtained from the photo XML data, and *n* is determined from the photo projections of the laser lines.

3) DBH calculation

Figure 3.a shows the horizontal projection from which stem radius, and ultimately DBH, is calculated.

Distances to the left (D_L) and right (D_R) laser boundaries, as well as the center point of the laser line (D_C) are calculated from equation (3), as described above. The cross-sectional length (LR) is calculated using a relationship similar to what is used in equation (2) except that the goal is to estimate LR (the distance between two points) from known point to camera distances, rather than estimating the point to camera distance from a known distance between two points:

$$\mathrm{LR} = \frac{Dc \cdot m \cdot Fx}{f \cdot M} \tag{4}$$

where, *Dc* is the points-to-camera distance of laser line center (O′C), *m* is the pixel number between left and right laser line boundaries. M is the total number of pixels associated with the photo width (X axis). *f* is focal length and *Fx* is the length of photo width (cm).

From fgiure 3a, it can be seen that O′L is tangent to the stem cross-section, and OL is perpendicular to O′L forming a right triangle △OLO′, The area of △OLO′ is 0.5×OO′×LX which is also equal to 0.5×OL×O′L. So the radius of the stem (*r*) is:

$$\mathrm{r} = \frac{\frac{1}{2} LR \cdot D_C}{D_{L/R} - \frac{1}{2} LR} \tag{5}$$

Finally, DBH is $2 \times r$.

2.4 Software Solution

An open source software package, ImageDBH, was developed in Python (Version 3.5.2, Python Software Foundation, 2016), to facilatate required measurements and automate the calculation of distance and DBH. This software consists of three panels: photo display panel, photo thumbnail panel, and results panel. The graphical user interface (GUI), shown in Figure 4, enables the user to identify the 6 points along the two laser lines needed to implement the algorithm described above. As shown in Figure 4, multiple trees can be selected and DBH extracted for each tree. The boundaries of both laser lines must be clearly visible in order to extract DBH, thus tree 71 in Figure 4 can be measured, while tree 72, which is partially ocluded by tree 71, cannot be measured.

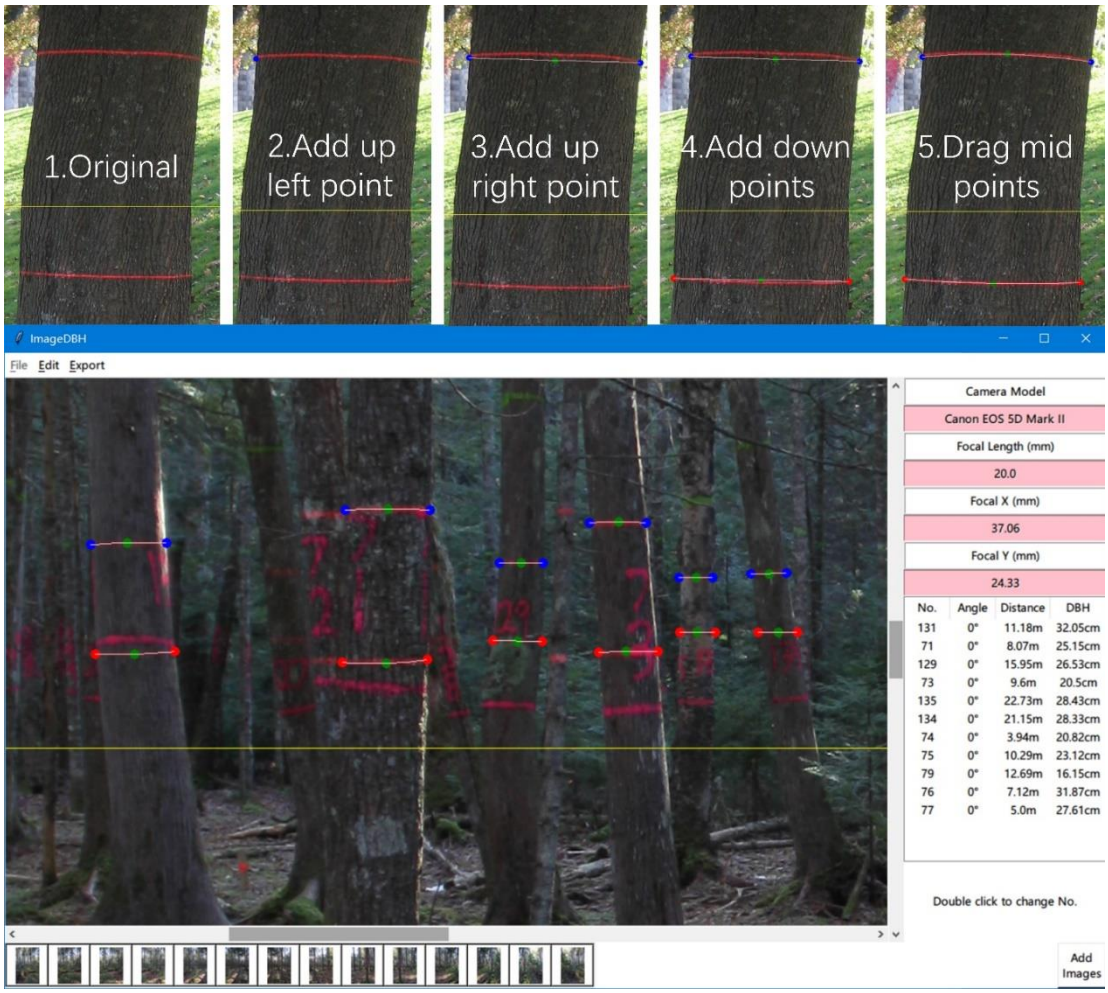


Figure 4: The GUI of ImageDBH.

3. Results

Of the 281 visible trees, we were able to extract 231 DBHs 82%). The number of measurements per identified tree ranged from 1 to 12. Figure 5 shows the distance from identified trees to camera while Figure 6 shows the DBH measurements. The color of the symbols in figure 6 is proportional to the number of photos from which DBH was successfully extracted.

In Figure 5, the x axis is distances from tree to camera calculated by the software, while the y axis is distances measured in the field. The field distance is calcualted from the tree coordinate location and the camera coordinate location and assumes no field errors. Calculated distances are quite close to the field measurements. The red line is the lowess regression line between the two distances and the white line is the 1:1 line. Across much of the range of distances, the photo-based distance appears to be less than the calculated field distance. An equivalence tests showed the the two distances were not similar ($p > 0.05$).

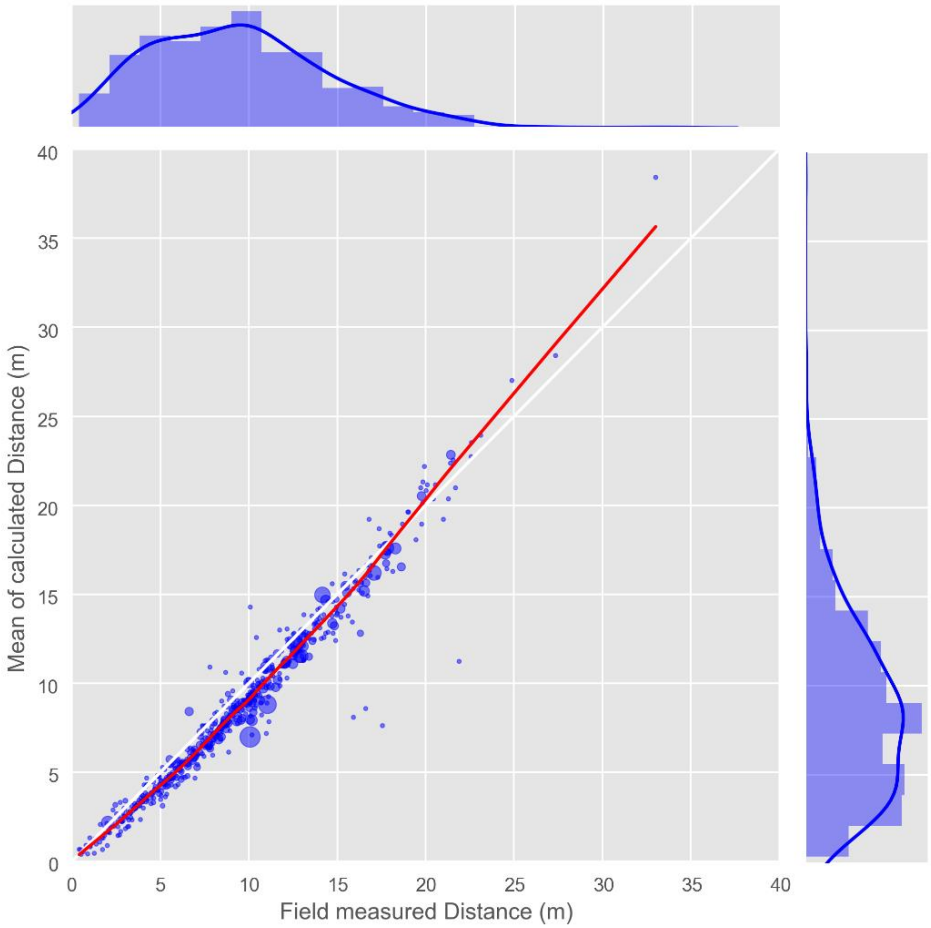


Figure 5: The distance result compaction between photo based calculation and field measured.

Figure 6 shows the relationship between field measured DBH and mean photo extracted DBH. The size of the symbols is proportional to variation in extracted DBH. The colors show how many duplicate measures each tree had. The sub-barplot shows the frequency count of duplicate measurements. The lowess regression (red line) shows that the photo calculated DBHs have a good relationship with field measured DBHs. An equivalence test confirmed that the photo extracted DBH's where similar to the field measured DBHs with a region of similarity of 32% at $\alpha = 0.05$.

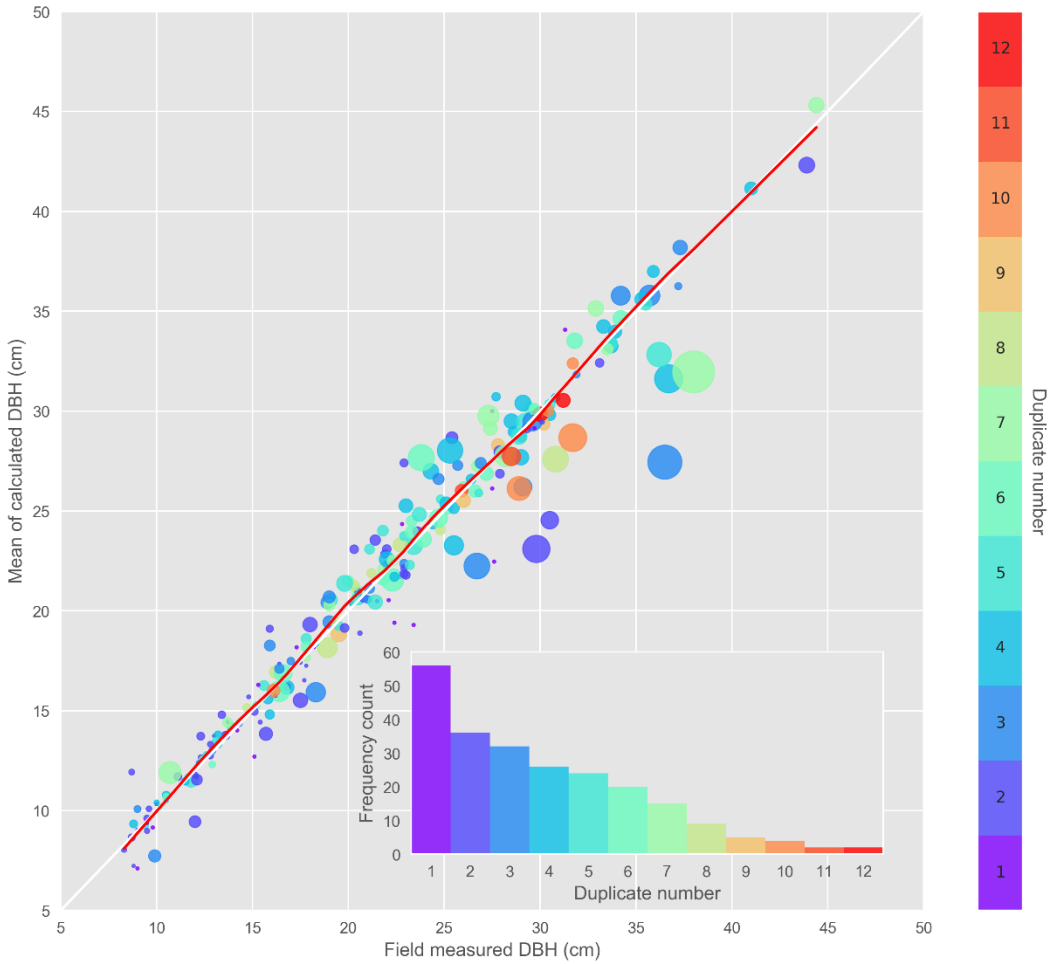
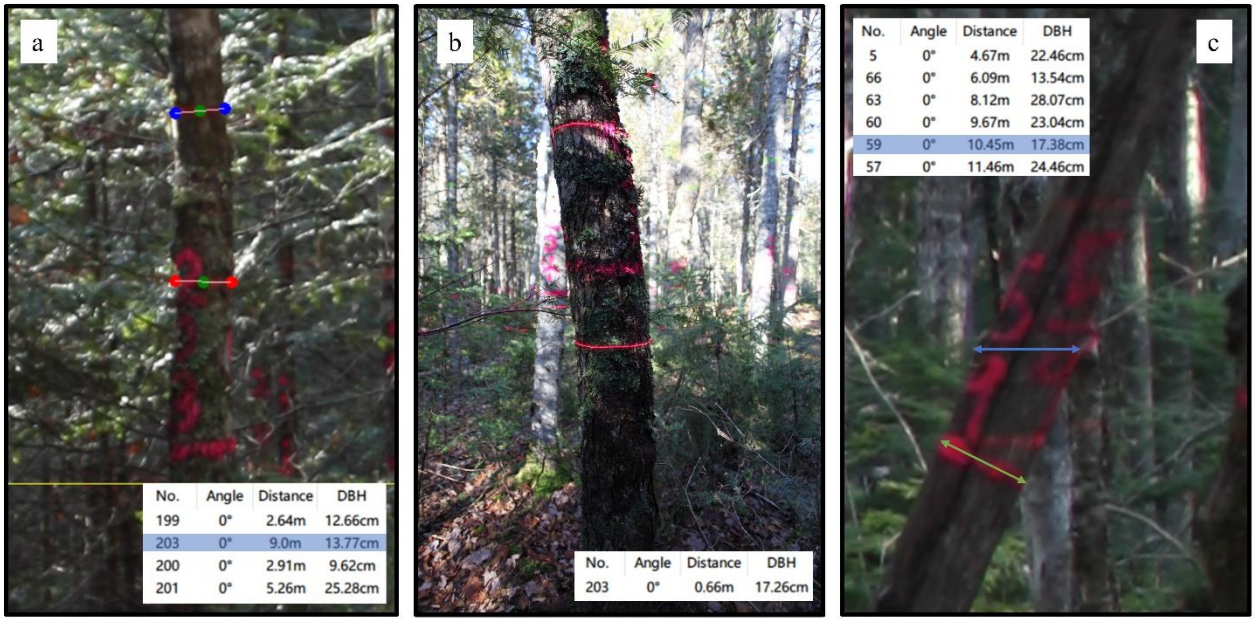


Figure 6: The DBH result compaction between photo based calculation and field measured.

The biggest sources of variation in the photo extracted DBHs was due to eccentricity in tree stem cross sectional area and to tree departure (lean) from the vertical axis (Figure 7). For example, figures 7a and 7b show different views for tree 203. In Figure 7a, the extracted DBH was from a line of sight that captured the short axis of the stem cross-section, while Figure 7b shows a line of sight capturing the long axis. Although tree 203's field measured DBH was 17.5cm, in figure 7a it was estimated to be 13.77cm and in figure 7b, it was estimated to be 17.26cm. Figure 7c shows tree 59 which leans from the perpendicular line of sight. Again, the filed measured DBH (yellow-green arrow in picture) was 16.4cm while the photo estimated DBH (blue arrow in picture) was 17.38cm.



Field measured: No.203 = 17.5cm No.59 = 16.4cm

Figure 7: The influences of treeshape and lean

4. Discussion

4.1 Results analysis

Calculated distances were mostly distributed between 1m to 15m from the camera (Figure 5). There were a few distances that exceeded 25m. Occulsion by other trees was the main factor limiting visible distances. Because the photo points used the 10m plot measurement grid (Figure 1) it makes sense that most sight distances were between 1m and 15m. Those distances exceeding 15m were for trees beyond the adjacent 10m cells. Clark (2000b) pointed out that the best measurement distances should be no more than about 15m. Dick et al. (2010) found that the number of trees visible on panoramic photos decreased rapidly as distance from camera increased, with approximately 68% visible at 10m. By taking photos at multiple points throughout the plot, we were able to increase the sighting probabilities of many trees; however, because we only used interior grid points, trees in the outer cells of the plot were less likely to be observed and at best only have a single observation photo.

Based on the lowess regression, when distances are smaller than 20m, calculated distances are underestimated relative to field measured distances, and when distances greater than 20m, calculated distances are overestimated relative

to field measured distances (Figure 5). When distance are far, the trunks in the photos are small, and may be only a few pixels in size, a single pixel deviation can cause a large change in the distance calculation results. Because it is difficult to guarantee the accuracy of the distances measured by the tape, the distances calculated from the photos may be more accurate. The fact that the DBH estimates were very accurate, and dependant upon the distances calculations, supports the assertion that the photo-based distances are more accurate than the field-based distances.

The DBH measurements were quite good (Figure 6). The lowess regression line corresponded very well with the 1:1 line, and the equivalence test showed a region of similarity of 32% at $\alpha = 0.05$. Standard errors associated with multiple DBH measurements increased with increasing distance from the camera. Different photo angles may capture shorter or longer axes of the stem cross-section (Figure 7a and 7b) which can result in very different DBH measurements. Tree lean is another factor that contributes to large deviations in DBH measurement on photos (Figure 7c). In addition, the photo measured diameter position was consistently higher than that of field survey (Figure 4 and Figure 7c). The original purpose of the photos taken in this study was for panoramic angle count samples, tree positioning was not considered. The results obtained in this study could be much improved by careful placement of the camera to obtain clear images of individual tree boles at reasonable distances from the tree. However, images focused on individual trees would increase field collection time. The ability to sample multiple trees from single images with reasonable accuracy may be more useful than more exact measurements on fewer trees.

There are other factors that may cause deviations. We assume that the camera is located exactly over the grid point. Deviations from the grid point would cause deviations in the distance estimates, but should not necessarily influence DBH measurements. The key points of laser line are marked mannually which may cause a small errors in selecting pixels. These errors become increasingly important as trees get farther away from the camera

4.2 Comparisons with other DBH measurement methods

There are many alternatives for indirect DBH measurement, including statistical models by regression analysis, laser

technology, and photos with computer vision.

There are several different laser-based tools available. Some are special instruments that measure directly, such as laser dendrometers (Skovsgaard et al. 1998) and hand-held laser survey instruments (Peet et al. 1997). These instruments often have an accuracy of around 1cm and can be quite time consuming to use in the field (Skovsgaard et al. 1998, Clark et al. 2000a). These instruments often have a single function, and, while useful for specialized research applications, are not suitable for extended inventory applications. Both ground-based laser scanners and airborne laser scanners can generate 3D point clouds (Næsset 2002, Strahler et al. 2008) This technology produces a spatially rich 3D dataset (Strahler et al. 2008, Li 2009, Parker and Evans 2009). However, not only are these instruments very expensive, but the point cloud data requires extensive processing in office to extract useful parameters and requires a strong field dataset to build required correlations (Hayashi et al. 2015). 2D laser scanner (Zheng et al. 2012, Ringdahl et al. 2013) is a cheaper and less expensive alternative, although even with optimized geometric algorthms (Wang Yaxiong et al. 2016), these devices are still inapplicable for large-scale surveys.

Computer vision using structure from motion (SfM) with different angled photos or single photo analyses presents other close-range photogrammetric techniques that have been applied to tree measurements. SfM is a 3D reconstruction method that builds 3D point clouds based on the theory of parallax (Andrews 1936, Hapca et al. 2007, Forsman et al. 2012). This technology obtains LiDAR-like scanning results with much less costs. However, this technology only obtains surface point clouds without perspective function, and only first “hits” can be obtained (Forsman et al. 2012), rather than the “inner” points generated by LiDAR with multiple “hits” per laser beam. SfM is observer intensive and requires large amounts of computational time to produce rich point clouds..

The vanishing point horizon and the variable photo scale are difficulties associated with single photo analysis. Benchmarks are normally required for most single photo analyses (Wang and Feng 2006, Xiaodong and Zhongke 2015). Benchmark adds inconvenience, espacially for multiple trees in a photo. Every object that requires measurement, must

have a benchmark placed on or beside them. Similar to our work here, some researchers have used the laser point to establish scale and achieved good reuslts on single trees (Pengle et al. 2013). Although a laser point is more convenient benchmarks, multiple trees are still problematic. Our parallel laser lines provide a straight forward method for obtaining DBH measurements from multiple trees using a single photo.

4.3 Improvement

Accoring to other studies, here are some suggestions to make this method better. 1) Make the photo calculated and field measured places on trunk closed to each other; 2) The best distance between trees and camera is around 10-20m (Clark et al. 2000b); 3) Just take photos for single trees rather than multiple trees (Wang and Feng 2006, Hapca et al. 2007); and 4) automate laser line detection through artificial intellegence or other computer-aided vision technique to minimize human-induced errors associated with manual selection of laser points (Kavdır 2004, Xiaodong and Zhongke 2015).

In the future, this method could be applied to measure the upper-stem diameter which is hard and dangerous to measure in the field by adding an elevation angle calculation algorithm (Grosenbaugh 1963, Leary and Beers 1963, Clark et al. 2000b, Hapca et al. 2007). How to reduce the human participation in selecting laser line key points (vertex and terminal points) is another worthwhile question for more research (Wang et al. 2004).

5. Conclusion

In this study, the procedure and algrothms on how to extract the DBH of trees from digital images with laser lines have been illustrated. The calculated results have also been validated with field survey data. The results show that it is possible to obtain DBH from photos with reasonable accuracy and effort in the office. Due to the limitation of picture resolution, which could make trees not recognizable, the best survey distance is smaller than 15m. Under the circumstance of tree shield, the survey ratio is up to 82.2% which has high corelation with field survey data. An open source software developed in Python simplifies the calculation procedure and makes it convenient for other users. Though extracting key

points of laser lines is finished by human-computer interaction, it is a great field for further research on how to optimize image identification algorithms and reduce human judgement. In the future, this new technology could be applied for estimating upper-stem diameters which are difficult to measured with any degree of accuracy and speed.

6. References

Andrews, G. S. 1936. Tree-heights from air photographs by simple parallax measurements. Forestry Chronicle 12:152–197.

Brandeis, T., K. C. Randolph, and M. R. Strub. 2009. Modeling Caribbean tree stem diameters from tree height and crown width measurements. International Journal of Mathematical and Computational Forestry & Natural-Resource Sciences 1:78–85.

Clark, N. A., R. H. Wynne, and D. L. Schmoldt. 2000a. A review of past research on dendrometers. Forest Science 46:570–576.

Clark, N. A., R. H. Wynne, D. L. Schmoldt, and M. Winn. 2000b. An assessment of the utility of a non-metric digital camera for measuring standing trees. Computers and Electronics in Agriculture 28:151–169.

Curtis, R. O. 1970. Stand density measures: An interpretation. Forest Science 16:403–414.

DeCourt, N. 1956. Utilisation de la photographie pour mesurer les surfaces terrières (The use of photography for measuring basal area). Revue Forestière Française 8:505–507.

Dick, A. R., J. A. Kershaw Jr., and D. A. MacLean. 2010. Spatial tree mapping using photography. Northern Journal of Applied Forestry 27:68–74.

Forsman, M., N. Börlin, and J. Holmgren. 2012. Estimation of tree stem attributes using terrestrial photogrammetry. Pages B5–261 XXII ISPRS Congress in Melbourne, Australia, Aug 25-Sep 1, 2012. ISPRS-International Society for Photogrammetry and Remote Sensing.

Grosenbaugh, L. R. 1963. Optical dendrometers for out-of-reach diameters: A conspectus and some new theory. Forest Science Monographs 4:48.

Hapca, A. I., F. Mothe, and J.-M. Leban. 2007. A digital photographic method for 3D reconstruction of standing tree shape. Annals of Forest Science 64:631–637.

Hayashi, R., J. A. Kershaw Jr., and A. R. Weiskittel. 2015. Evaluation of alternative methods for using LiDAR to predict aboveground biomass in mixed species and structurally complex forests in northeastern North America. Mathematical and Computational Forestry and Natural Resources Sciences 7:49–65.

Jonckheere, I., S. Fleck, K. Nackaerts, B. Muys, P. Coppin, M. Weiss, and F. Baret. 2004. Review of methods for in situ leaf area index determination: Part I. Theories, sensors and hemispherical photography. Agricultural and Forest Meteorology 121:19–35.

Kavdır, I. 2004. Application note Discrimination of sunflower, weed and soil by artificial neural networks. Computers and Electronics in Agriculture 44:153–160.

Kershaw, J. A., Jr., M. J. Ducey, T. W. Beers, and B. Husch. 2016. Forest Mensuration. 5th edition. Wiley/Blackwell, Hobokin, NJ.

Larsen, D. R., and J. A. Kershaw Jr. 1990. The measurement of leaf area. Pages 465–475 Techniques in Forest Tree Ecophysiology. Lassoie, J. and T. Hinkley (eds.). CRC Press, Boca Raton, FL.

Leary, R. A., and T. W. Beers. 1963. Measurement of upper-stem diameter with a transit. Journal of Forestry 61:448–450.

Li, Y. 2009. A comparison of forest height prediction from FIA field measurement and LiDAR data via spatial models. Page 14. USDA, Forest Service, Rocky Mountain Research Station.

McCarter, J. B., and J. N. Long. 1986. A lodgepole pine density management diagram. Western Journal of Applied Forestry 1:6–11.

McTague, J.-P., and D. R. Patton. 1989. Stand density index and its application in describing wildlife habitat. Wildlife Society Bulletin 17:58–62.

Morrison, M. L., B. G. Marcot, and R. W. Mannan. 1992. Wildlife–habitat relationships: Concepts and applications. University of Wisconsin Press, Madison, WI.

Næsset, E. 2002. Predicting forest stand characteristics with airborne scanning laser using a practical two-stage procedure and field data. Remote Sensing of Environment 80:88–99.

Parker, R. C., and D. L. Evans. 2009. LiDAR Forest Inventory with Single-Tree, Double-, and Single-Phase Procedures. International Journal of Forestry Research 2009:6.

Peet, F. G., D. J. Morrison, and K. W. Pellow. 1997. Using a hand-held electronic laser-based survey instrument for stem mapping. Canadian Journal of Forest Research 27:2104–2108.

Pengle, C., L. Jinhao, and W. Dian. 2013. Measuring diameters at breast height using combination method of laser and machine vision (in Chinese). Transactions of the Chinese Society for Agricultural Machinery 44:271–275.

Ringdahl, O., P. Hohnloser, T. Hellström, J. Holmgren, and O. Lindroos. 2013. Enhanced Algorithms for Estimating Tree Trunk Diameter Using 2D Laser Scanner. Remote Sensing 5:4839–4856.

Roberts, M. R., and L. Zhu. 2002. Early response of the herbaceous layer to harvesting in a mixed coniferous–deciduous forest in New Brunswick, Canada. Forest Ecology and Management 155:17–31.

Skovsgaard, J. P., V. K. Johannsen, and J. K. Vanclay. 1998. Accuracy and precision of two laser dendrometers. Forestry 71:131–139.

Stewart, B., C. . Cieszewski, and M. Zasada. 2004. Use of a camera as an angle-gauge in angle-count sampling. Pages 375–380 Second International Conference on Forest Measurements and Quantitative Methods and Management. Warnell School of Forestry and Natural Resources, University of Georgia, Athens, GA.

Strahler, A. H., D. L. B. Jupp, C. E. Woodcock, C. B. Schaaf, T. Yao, F. Zhao, X. Yang, J. Lovell, D. S. Culvenor, G. Newnham, W. Ni-Miester, and W. Boykin-Morris. 2008. Retrieval of forest structural parameters using a ground-based lidar instrument (Echidna®). Canadian Journal of Remote Sensing 34:S426–S440.

Sullivan, A. D., and M. R. Reynolds. 1976. Notes: Regression problems from repeated measurements. Forest Science 22:382–385.

Wang, X., and Y. Feng. 2006. Study on Field Server-based Remote Tree Diameter Measurement (in Chinese). FOREST RESEARCH-CHINESE ACADEMY OF FORESTRY 19:675.

Wang, X., C. Zhang, and S. Tang. 2004. A technology of gathering forest diameter based on image understanding (in Chinese). Scientia Silvae Sinicae 41:16–20.

Wang Yaxiong, Kang Feng, Li Wenbin, and Zheng Yongjun. 2016. Optimization of Geometry Algorithm for DBH of Standing Tree on 2D Laser Detection (in Chinese). Transactions of the Chinese Society of Agricultural Machinery 47:290–296.

Westfall, J. A. 2010. New models for predicting diameter at breast height from stump dimensions. Northern Journal of Applied Forestry 27:21–27.

Xiaodong, H., and F. Zhongke. 2015. Obtainment of Sample Tree's DBH Based on Digital Camera (in Chinese). Transactions of the Chinese Society for Agricultural Machinery 46:266–272.

Zheng, Y., J. Liu, D. Wang, and R. Yang. 2012. Laser Scanning Measurements on Trees for Logging Harvesting Operations. Sensors 12:9273–9285.

Zheng, Y., J. Liu, S. Zhang, and T. Ge. 2014. Extraction of Trees Stem Diameters at Breast Height by Terrestrial Laser Sensor for Selective Cutting. Sensors & Transducers 164:65.

7. Appendix (codes)

7.1 ImageDBH.py

```
# -*- coding:UTF-8 -*-
import traceback
import xlwt
from tkinter.messagebox import *
from tkinter.filedialog import *
from tkinter.simpledialog import *
from tkinter import ttk
from PIL import ExifTags
from PIL.ImageTk import Image
from PIL.ImageTk import PhotoImage
import DBHCalculation


class ScrolledCanvas(Frame):
    Imagedir = ''
    NewTree_OnOff = -1
    TreeNum = 0
    PointNum = {'UP1': [],'UP2':[], 'UC': [],'UL':[], 'DP1': [],'DP2':[], 'DC': [], 'DL':[],'Comb':[]}
    PhotoSize = []
    Rotate = 0 # []+(逆时针 90 度), -(顺时针 90 度)
    ISIN = False
    '''DataFrame
    Not need to record the position because can get the point position by :canvas.coords(ID)
    coords(i, new_xy) # change coordinates
    TreeID                  [0, 1, 2, 3, 4, 5...]
    UP1 (Up Point1)     [2,10,...]
    UP2 (Up Point2)     [3,11,...]
    UC    (UP Centre)     [4,12,...]
    UL    (Up Line)         [5,13,...]
    DP1 (Down Point1) [6,14,...]
    DP2 (Down Point2) [7,15,...]
    DC    (Down Cnetre) [8,16,...]
    DL    (Down Line)     [9,17,...]
    Comb[0 [[UP1,UP2,UC] -> UL,[DP1,DP2,UC] -> DL],
         1 [[UL]                    ,[DL]                    ],
         2 [[]                      ,[]                      ],...]
    '''
    def __init__(self,parent=None):
        Frame.__init__(self,parent)
        self.pack(expand=YES,fill=BOTH)
```

```
        canvas = Canvas(self, relief=SUNKEN)
        canvas.config(width=800, height=600, bg='white',bd=1)
        canvas.config(highlightthickness=0)
        canvas.bind('<ButtonPress-1>', self.onPutPoint)
        canvas.bind('<B1-Motion>', self.onMovePoint)
        canvas.bind('<ButtonRelease-1>', self.LooseMouse)

        sbarx = Scrollbar(self,orient='horizontal')
        sbary = Scrollbar(self)
        sbarx.config(command=canvas.xview,bg='white')
        sbary.config(command=canvas.yview,bg='white')
        canvas.config(xscrollcommand=sbarx.set)
        canvas.config(yscrollcommand=sbary.set)

        sbary.pack(side=RIGHT, fill=Y)
        sbarx.pack(side=BOTTOM, fill=X)
        canvas.pack(side=TOP, expand=YES,fill=BOTH)

        self.canvas = canvas

    def Open_Picture(self,event=None):
        self.ClearCanvas()
        image = Image.open(self.Imagedir)
        if self.Rotate == 90 or self.Rotate == -270:
            photo = PhotoImage(image.transpose(Image.ROTATE_270))
        elif self.Rotate == 180 or self.Rotate == -180:
            photo = PhotoImage(image.transpose(Image.ROTATE_180))
        elif self.Rotate == 270 or self.Rotate == -90:
            photo = PhotoImage(image.transpose(Image.ROTATE_90))
        else:
            photo = PhotoImage(image)
        self.photo = photo
        self.canvas.create_image(0, 0, image=photo, anchor=NW)
        self.canvas.config(scrollregion=(0,0,photo.width(),photo.height()))
        self.canvas.create_line((0,photo.height()/2,photo.width(),photo.height()/2),fill='yellow')
        self.PhotoSize=[photo.width(),photo.height()]
        # Draw points from SysTemp['PointPosition'] if it is not empty
        global SysTemp
        Position = SysTemp['PointPosition'][PicSelectMenu.NowPicNum]
        if Position != []:
            self.Position2Num(Position)

    def ClearCanvas(self,event=None):
```

```
        # event.widget.delete('all')    # use tag all
        self.canvas.delete('all')
        self.PointNum = {'UP1': [],'UP2':[], 'UC': [],'UL':[],
                         'DP1': [],'DP2':[], 'DC': [], 'DL':[],
                         'DBH': [],'Comb':[]}
        self.TreeNum = 0

    def onPutPoint(self,event):
        if len(self.canvas.find_all()) > 0:
            x=self.canvas.canvasx(event.x);y=self.canvas.canvasy(event.y)
            i=self.TreeNum
            # if it is the first click -> Up point 1
            if self.NewTree_OnOff == 0:
                # Draw points
                ID_p1 = self.Create_Point(x, y, 'blue')
                # Add information to DataFrame
                self.PointNum['UP1'].append(ID_p1)
                self.PointNum['Comb'].append([])
                # Fresh OnOff index
                self.NewTree_OnOff = 1
            # if it is the second click -> Up point 2
            elif self.NewTree_OnOff == 1:
                # Draw points
                ID_p2 = self.Create_Point(x, y, 'blue')
                # Add information to DataFrame
                self.PointNum['UP2'].append(ID_p2)
                # Calculate centre point
                (Cx, Cy)=self.Calcu_CentrePoints(self.PointNum['UP1'][i],
                                                 self.PointNum['UP2'][i])
                # Draw centre point
                ID_c = self.Create_Point(Cx, Cy, 'green')
                # Add centre point information to DataFrame
                self.PointNum['UC'].append(ID_c)
                # Add curve line comb
                self.PointNum['Comb'][i].append([self.PointNum['UP1'][i],
                                                 self.PointNum['UP2'][i],
                                                 self.PointNum['UC'][i]])
                # Draw curve line
                ID_l = self.Create_Curveline(self.PointNum['Comb'][i][0])
                # Add line information to DataFrame
                self.PointNum['UL'].append(ID_l)
                # Fresh OnOff index
                self.NewTree_OnOff = 2
            # if it is the tird click -> Down point 1
```

```
            elif self.NewTree_OnOff == 2:
                # Draw points
                ID_p1 = self.Create_Point(x, y, 'red')
                # Add information to DataFrame
                self.PointNum['DP1'].append(ID_p1)
                # Fresh OnOff index
                self.NewTree_OnOff = 3
            # if it is the forth click -> Down point2
            elif self.NewTree_OnOff == 3:
                # Draw points
                ID_p2 = self.Create_Point(x, y, 'red')
                # Add information to DataFrame
                self.PointNum['DP2'].append(ID_p2)
                # Calculate centre point
                (Cx, Cy) = self.Calcu_CentrePoints(self.PointNum['DP1'][i],
                                                   self.PointNum['DP2'][i])
                # Draw centre point
                ID_c = self.Create_Point(Cx, Cy, 'green')
                # Add centre point information to DataFrame
                self.PointNum['DC'].append(ID_c)
                # Add curve line comb
                self.PointNum['Comb'][i].append([self.PointNum['DP1'][i],
                                                 self.PointNum['DP2'][i],
                                                 self.PointNum['DC'][i]])
                # Draw curve line
                ID_l = self.Create_Curveline(self.PointNum['Comb'][i][1])
                # Add line information to DataFrame
                self.PointNum['DL'].append(ID_l)
                # Fresh OnOff index
                self.NewTree_OnOff = -1
                # Add tree Number
                self.TreeNum += 1
                global SysTemp,TreeNo
                # save points and tree number information
                SysTemp['PointPosition'][PicSelectMenu.NowPicNum] = self.Num2Position()
                SysTemp['TreeNo.'][PicSelectMenu.NowPicNum].append(TreeNo)
                PicSelectMenu.ShowInTable()

    def onMovePoint(self,event):
        # not in add point mode(make sure this click is move points rather than add points)
        if self.NewTree_OnOff == -1:
            x = self.canvas.canvasx(event.x)
            y = self.canvas.canvasy(event.y)
            # Select a new point, initialise self.ISIN
```

```
            if self.ISIN == False:
                idtouched = event.widget.find_closest(x, y)
                # if the canvas is empty(just open without any photo),
                #       IDtouched == (), function self.isin goes error
                if idtouched:
                    isin = self.isin(idtouched[0])
                    # if selected point belongs to PointNum,
                    #       isin returns(Ture,PointKind[int], pointLine[int],IDtouched)
                    #       e.g. ISIN = (True,1,3,2)
                    # else not in
                    #       ISIN returns (False,-1,-1,2)
                    if isin[0]:
                        self.ISIN = isin
            # Do not release mouse, keep moving
            else:
                # set selected point position == mouse position
                self.canvas.coords(self.ISIN[3], (x - 5, y - 5, x + 5, y + 5))
                Comb = self.PointNum['Comb']
                #print(Comb)
                for i in range(len(Comb)):
                    # move points are up points
                    if self.ISIN[3] in Comb[i][0]:
                        lineID = self.PointNum['UL'][i]
                        # move line
                        self.Create_Curveline(self.PointNum['Comb'][i][0],lineID)
                    # move points are down points
                    if self.ISIN[3] in Comb[i][1]:
                        lineID = self.PointNum['DL'][i]
                        # move line
                        self.Create_Curveline(self.PointNum['Comb'][i][1],lineID)

def LooseMouse(self,event):
    global SysTemp
    if self.ISIN:
        if self.ISIN != -1:
            x = self.canvas.canvasx(event.x)
            y = self.canvas.canvasy(event.y)
            # change moved point position in SysTemp file
            SysTemp['PointPosition'][PicSelectMenu.NowPicNum][self.ISIN[1]][self.ISIN[2]] = [x,y]
        PicSelectMenu.ShowInTable()
        self.ISIN = False


def Create_Curveline(self,ID,lineID=None):
```

```
        # create_line has three points to draw curve line
        # but if these three points is P_baseL, P_baseR, and P_top
        # the curve line would scross P_centre rather than P_top
        # So we need to calculate P_top if we want the curve line across P_centre
        # (P)_baseL--------(P)_baseR
        #     \                    /
        #       \     (P)_c    /
        #         \            /
        #          (P)_top
        ID_1=ID[0];ID_2=ID[1];ID_C=ID[2]
        P1 = self.ID2Position(ID_1)
        P2 = self.ID2Position(ID_2)
        Pc = self.ID2Position(ID_C)
        # coords(i, new_xy)
        Xbl = P1[0]
        Ybl = P1[1]
        Xc = Pc[0]
        Yc = Pc[1]
        Xbr = P2[0]
        Ybr = P2[1]
        Xtop = 2 * Xc - (Xbl + Xbr) / 2
        Ytop = 2 * Yc - (Ybl + Ybr) / 2
        if lineID == None:
            ObjectID = self.canvas.create_line((Xbl, Ybl), (Xtop, Ytop), (Xbr, Ybr), smooth=True, fill="pink")
            return ObjectID
        else:
            self.canvas.coords(lineID,[Xbl, Ybl, Xtop, Ytop, Xbr, Ybr])

    def Create_Point(self,x,y,color):
        ObjectID = self.canvas.create_oval(x - 5, y - 5, x + 5, y + 5, fill=color, outline=color)
        return ObjectID

    def Calcu_CentrePoints(self,ID1,ID2):
        P1 = self.ID2Position(ID1)
        P2 = self.ID2Position(ID2)
        x1=P1[0]
        y1=P1[1]
        x2=P2[0]
        y2=P2[1]
        Cx=(x1+x2)/2
        Cy=(y1+y2)/2
        return Cx,Cy

    def ID2Position(self,ID):
```

```
        Position = self.canvas.coords(ID)
        # curve line returns 6 parametres while points(circle) returns 4 parameters
        if len(Position) == 4:
            X_5 = Position[0]
            Y_5 = Position[1]
            Position = [X_5+5,Y_5+5]
        return Position

    def isin(self,ID):
        isin = False
        PointKind = -1
        PointLine = -1
        for i in ['UP1','UP2', 'UC', 'DP1','DP2', 'DC']:
            Ans = ID in self.PointNum[i]
            if Ans or False:
                isin = True
                # record the point kind in order to change SysTemp by mouse moving function
                PointKind = ['UP1','UP2', 'UC', 'DP1','DP2', 'DC'].index(i)
                PointLine = self.PointNum[i].index(ID)
        return (isin,PointLine,PointKind,ID)

    def Position2Num(self,Position):
        for j in range(len(Position)):
            P4Use = Position[j][:6]
            i = self.TreeNum
            # P4Use = [[UP1.x, UP1.y], [UP2.x, UP2.y], [UC.x, UC.y], [DP1.x, DP1.y], [DP2.x, DP2.y], [DC.x,
DC.y]]
            # Draw points - UP1
            ID_p1 = self.Create_Point(P4Use[0][0], P4Use[0][1], 'blue')
            # Add information to DataFrame
            self.PointNum['UP1'].append(ID_p1)
            self.PointNum['Comb'].append([])
            # Draw points - UP2
            ID_p2 = self.Create_Point(P4Use[1][0], P4Use[1][1], 'blue')
            # Add information to DataFrame
            self.PointNum['UP2'].append(ID_p2)
            # Draw centre point - UC
            ID_c = self.Create_Point(P4Use[2][0], P4Use[2][1], 'green')
            # Add centre point information to DataFrame
            self.PointNum['UC'].append(ID_c)
            # Add curve line comb
            self.PointNum['Comb'][i].append([self.PointNum['UP1'][i],
                                             self.PointNum['UP2'][i],
                                             self.PointNum['UC'][i]])
```

```
            # Draw curve line
            ID_l = self.Create_Curveline(self.PointNum['Comb'][i][0])
            # Add line information to DataFrame
            self.PointNum['UL'].append(ID_l)
            # Draw points - DP1
            ID_p1 = self.Create_Point(P4Use[3][0], P4Use[3][1], 'red')
            # Add information to DataFrame
            self.PointNum['DP1'].append(ID_p1)
            # Draw points - DP2
            ID_p2 = self.Create_Point(P4Use[4][0], P4Use[4][1], 'red')
            # Add information to DataFrame
            self.PointNum['DP2'].append(ID_p2)
            # Draw centre point - DC
            ID_c = self.Create_Point(P4Use[5][0], P4Use[5][1], 'green')
            # Add centre point information to DataFrame
            self.PointNum['DC'].append(ID_c)
            # Add curve line comb
            self.PointNum['Comb'][i].append([self.PointNum['DP1'][i],
                                             self.PointNum['DP2'][i],
                                             self.PointNum['DC'][i]])
            # Draw curve line
            ID_l = self.Create_Curveline(self.PointNum['Comb'][i][1])
            # Add line information to DataFrame
            self.PointNum['DL'].append(ID_l)
            # Add tree Number
            self.TreeNum += 1

    def Num2Position(self,event=None):
        # UP1 | UP2 | UC | DP1 | DP2 | DC | PhotoSize
        PointPosition = []
        for i in range(len(self.PointNum['DC'])):
            PointPosition.append([self.ID2Position(self.PointNum['UP1'][i]),
                                  self.ID2Position(self.PointNum['UP2'][i]),
                                  self.ID2Position(self.PointNum['UC'][i]),
                                  self.ID2Position(self.PointNum['DP1'][i]),
                                  self.ID2Position(self.PointNum['DP2'][i]),
                                  self.ID2Position(self.PointNum['DC'][i]),
                                  self.PhotoSize])
        return PointPosition

    def getCamInfo(self,img,event=None):#img=Image.open(Imagedir)
        exif_human = {ExifTags.TAGS[k]: v for k, v in img._getexif().items() if k in ExifTags.TAGS}
        # XResolution = exif_human['XResolution'][0]
        # YResolution = exif_human['YResolution'][0]
```

```
            if exif_human['FocalLength'][1]==0: # lack data
                FocalLength = 0
            else:
                FocalLength = exif_human['FocalLength'][0]/exif_human['FocalLength'][1] # mm
            if exif_human['FocalPlaneResolutionUnit'] == 2: # inch(default)
                FPX = exif_human['FocalPlaneXResolution'][1] * 2.54/100 # mm
                FPY = exif_human['FocalPlaneYResolution'][1] * 2.54/100 # mm
            else:
                FPX = str(exif_human['FocalPlaneXResolution'][1])+'mm'
                FPY = str(exif_human['FocalPlaneYResolution'][1])+'mm'
            info = {'Size': img.size,
                        # 'YResolution': YResolution,
                        # 'XResolution': XResolution,
                        'FocalLength': FocalLength,
                        'FPX':FPX,
                        'FPY':FPY,
                        'Model': exif_human['Model'],
                        }
            return info

        def my_except_hook(type, value, tb):
            exception_string = "".join(traceback.format_exception(type, value, tb))
            showerror('Error!',exception_string)

        sys.excepthook = my_except_hook

class MenuBar(Frame):

        def __init__(self,parent=None):
            Frame.__init__(self, parent)
            self.pack()
            menubar = Frame(self)
            menubar.config(bg='white')
            menubar.pack(side=TOP, fill=X)

            fbutton = Menubutton(menubar, text='File', underline=0)
            fbutton.pack(side=LEFT)
            file = Menu(fbutton, tearoff=False)
            file.add_command(label='New', command=self.NewProj, underline=0)
            file.add_command(label='Open', command=self.OpenProj, underline=1)
            file.add_command(label='Quit', command=self.Quit, underline=0)
            fbutton.config(menu=file, bg='white')

            ebutton = Menubutton(menubar, text='Edit', underline=0,state=DISABLED)
```

```
        ebutton.pack(side=LEFT)
        edit = Menu(ebutton, tearoff=False)
        edit.add_command(label='Add points', command=self.Add_points_on, underline=0)
        edit.add_command(label='Delete points', command=self.notdone, underline=0)
        edit.add_separator()
        ebutton.config(menu=edit, bg='white')

        cbutton = Menubutton(menubar, text='Export', underline=0, state=DISABLED)
        cbutton.pack(side=LEFT)
        export = Menu(cbutton, tearoff=False)
        export.add_command(label='picture', command=self.notdone, underline=0)
        export.add_command(label='excel', command=self.export_excel, underline=0)
        cbutton.config(menu=export, bg='white')

        submenu = Menu(edit, tearoff=False)
        submenu.add_command(label='Clockwise 90°', command=self.cw90,underline=0)
        submenu.add_command(label='Anti-Clockwise 90°', command=self.acw90, underline=0)
        submenu.add_command(label='Clockwise 180°', command=self.cw180, underline=0)
        edit.add_cascade(label='Rotate image', menu=submenu,underline=0)

        self.cbutton = cbutton
        self.ebutton = ebutton
        self.fbutton = fbutton

    def notdone(self):
        showerror('Not implemented','Not yet available')

    def Add_points_on(self):
        # set tree number as global to save it in the end of add points in ScrolledCanvas.onAddPoint
        global SysTemp,TreeNo
        TreeNo = askstring('Notice', 'Print Tree number')
        if TreeNo != None:
            if TreeNo == '':
                TreeNo = str(ScrolledCanvas.TreeNum+1)
            ScrolledCanvas.NewTree_OnOff=0        # activate left click to add point

    def cw90(self):
        ScrolledCanvas.Rotate += 90
        if ScrolledCanvas.Rotate == 360:
            ScrolledCanvas.Rotate = 0
        SysTemp['Rotate'][PicSelectMenu.NowPicNum] = ScrolledCanvas.Rotate
        ScrolledCanvas.Open_Picture()

    def acw90(self):
```

```
        ScrolledCanvas.Rotate -= 90
        if ScrolledCanvas.Rotate == -360:
            ScrolledCanvas.Rotate = 0
        SysTemp['Rotate'][PicSelectMenu.NowPicNum] = ScrolledCanvas.Rotate
        ScrolledCanvas.Open_Picture()

    def cw180(self):
        ScrolledCanvas.Rotate += 180
        if ScrolledCanvas.Rotate == 360:
            ScrolledCanvas.Rotate = 0
        SysTemp['Rotate'][PicSelectMenu.NowPicNum] = ScrolledCanvas.Rotate
        ScrolledCanvas.Open_Picture()

    def NewProj(self):
        global Projectdir,SysTemp
        Projectdir = asksaveasfilename(title='New project', filetypes=[('DBH project', '.dbh')])
        if Projectdir != '':
            if Projectdir[-4:]=='.dbh':
                Projectdir=Projectdir[:-4]
                #print(Projectdir)
            SysTemp = {'photos': [], 'PointPosition': [], 'CamInfo':[], 'CalcuData': [],
                       'CtrlOnOff': [], 'Rotate': [], 'TreeNo.':[]}
            PicSelectMenu.AddPicbtn.config(state=NORMAL)
            self.cbutton.config(state=NORMAL)
            self.ebutton.config(state=NORMAL)
            self.fbutton.config(state=DISABLED)

    def OpenProj(self):
        global Projectdir,SysTemp
        Projectdir = askopenfilename(title='New project', filetypes=[('DBH project', '.dbh')])
        Projectdir = Projectdir[:-4]
        #print(Projectdir)
        if Projectdir != '':
            SysTemp = {'photos': [], 'PointPosition': [], 'CamInfo':[], 'CalcuData': [],
                       'CtrlOnOff': [], 'Rotate': [], 'TreeNo.':[]}
            f = open(Projectdir + '.dbh', 'r')
            SysTemp = eval(f.read())
            f.close()

            # check if the pictures are exist, if exist load, if not, let user to re-link pictures
            # **The best algorithm for this place is iteration check function and bulk replacement**
            # **(no time for me to achieve it T_T)**
            PicOk = True
            photo_folder_old = []
```

```
            photo_folder_new = []
            for photo in SysTemp['photos']:
                line = SysTemp['photos'].index(photo)
                if not os.path.exists(photo):
                    PicOk =False
                    # the next photo path is same to former successfully replaced photo path
                    if photo_folder_new and photo_folder_old == photo[:photo.rindex('/')]:
                        newphoto = photo_folder_new + '/' + photo[photo.rindex('/'):]
                        SysTemp['photos'][line] = newphoto
                        print(newphoto)
                        PicOk = True
                    # this photo is the first picture (photo_folder_old==[])
                    # or this photo's folder is different from former successfully replaced photo's folder
                    else:
                        photo_folder_old = photo[:photo.rindex('/')]
                        ans = askyesno('Image missing', photo + '\n re-link it?')
                        if ans:        # re-link
                            newphoto = askopenfilename(title='Relink:        ' + photo,
                                                       filetypes=[('Image file', photo[-4:])])
                            photo_folder_new = newphoto[:newphoto.rindex('/')]
                            if newphoto == '':
                                showerror('Image missing', 'Could not link photos, please open other projects!')
                            else:
                                SysTemp['photos'][line] = newphoto
                                PicOk = True
                        else:        # not relink -> delete missing message
                            del SysTemp['photos'][line]
                            del SysTemp['CtrlOnOff'][line]
                            del SysTemp['PointPosition'][line]
                            del SysTemp['Rotate'][line]
                            del SysTemp['CalcuData'][line]
                            del SysTemp['CamInfo'][line]
                            del SysTemp['TreeNo.'][line]
                            PicOk = True
            if PicOk:
                PicSelectMenu.AddPicButton(SysTemp['photos'],new=False)
                PicSelectMenu.AddPicbtn.config(state=NORMAL)
                self.cbutton.config(state=NORMAL)
                self.ebutton.config(state=NORMAL)
                self.fbutton.config(state=DISABLED)

    def Quit(self):
        ans = askokcancel('Verfy exit', "Really quit?")
        if ans:
```

```
            Frame.quit(self)

    def export_excel(self):
        global SysTemp
        Calcu_Data = SysTemp['CalcuData']
        savepath = asksaveasfilename(title='export data', filetypes=[('excelfile', '.xls')])
        print(savepath)
        if savepath != '':
            if savepath[-4:] == '.xls':
                savepath = savepath[:-4]
            wb = xlwt.Workbook(encoding='utf-8')
            i = 1
            for onepic in Calcu_Data:
                worksheet = wb.add_sheet(str(i))
                worksheet.write(0, 0, label='treenum')
                worksheet.write(0, 1, label='angle(°)')
                worksheet.write(0, 2, label='distance(m)')
                worksheet.write(0, 3, label='DBH(cm)')
                t = 1
                for onetree in onepic:
                    worksheet.write(t, 0, label=str(onetree[0]))
                    worksheet.write(t, 1, label=float(onetree[1][:-1]))
                    worksheet.write(t, 2, label=float(onetree[2][:-1]))
                    worksheet.write(t, 3, label=float(onetree[3][:-2]))
                    t += 1
                i += 1

            wb.save(savepath + '.xls')
            print('done')


    def my_except_hook(type, value, tb):
        exception_string = "".join(traceback.format_exception(type, value, tb))
        showerror('Error!',exception_string)
    sys.excepthook = my_except_hook


class PicSelectMenu(Frame):

    def __init__(self,parent=None):
        Frame.__init__(self,parent)

        MainFrame = Frame(parent, bg='white')
        MainFrame.pack(side=BOTTOM, fill=X)
```

```
        toolbar = Frame(MainFrame, relief=SUNKEN, bd=2)
        toolbar.config(height=45, bg='white')
        toolbar.pack(side=LEFT, fill=X)
        AddPicbtn = Button(MainFrame, text='Add\nImages',
                                   command=self.AddPic, bg='white', state=DISABLED)
        AddPicbtn.pack(side=RIGHT, fill=Y)



        self.toolbar=toolbar
        self.AddPicbtn = AddPicbtn
        self.toolPhotoDir = []
        self.AddPicButton(SysTemp['photos'])
        self.NowPicNum = 0

    def AddPicButton(self,Picdir,new=True):
        global SysTemp
        if new:
            k = len(SysTemp['photos'])
        else:
            k = 0
        for i in range(len(Picdir)):
            if new:
                SysTemp['photos'].append(Picdir[i])
                self.AddSysTempInfo()
            imgobj = Image.open(Picdir[i])
            imgobj.thumbnail((40, 40), Image.ANTIALIAS)
            img = PhotoImage(imgobj)
            btn = Button(self.toolbar, image=img, cursor='hand2')
            # print(SysTemp['photos'],k+i)
            handler = lambda savefile=(SysTemp['photos'][k + i],k+i): self.ShowInCanvas(savefile)
            SysTemp['CamInfo'][k+i] = ScrolledCanvas.getCamInfo(img=imgobj)
            btn.config(relief=RAISED, bd=2, bg='white', command=handler)
            btn.config(width=40, height=40)
            btn.pack(side=LEFT)
            self.toolPhotoDir.append((img,imgobj))       # why lack this sentence thumbnail don't show?

    def ShowInCanvas(self,savefile):
        global SysTemp
        imagedir=savefile[0];num=savefile[1]
        self.NowPicNum = num
        ScrolledCanvas.Imagedir = imagedir
        ScrolledCanvas.Rotate = SysTemp['Rotate'][self.NowPicNum]
        ScrolledCanvas.Open_Picture()
```

```
        self.ShowInTable()

    def ShowInTable(self):
        global Systemp
        # show Caminfo
        CamInfo = SysTemp['CamInfo'][self.NowPicNum]
        TreeNo = SysTemp['TreeNo.'][self.NowPicNum]
        TableInfo.Ml.delete('0', END)
        TableInfo.Ml.insert(0, CamInfo['Model'])
        TableInfo.FL.delete('0', END)
        TableInfo.FL.insert(0, str(round(CamInfo['FocalLength'],2)))
        TableInfo.FX.delete('0', END)
        TableInfo.FX.insert(0, str(round(CamInfo['FPX'],2)))
        TableInfo.FY.delete('0', END)
        TableInfo.FY.insert(0, str(round(CamInfo['FPY'],2)))
        # get data
        # if number of trees in SysTem['CalcuData'] not equal to latest trees number
        #       which means add a new tree
        # or move a tree point
        # then refresh information in table panel
        if len(SysTemp['CalcuData'][self.NowPicNum]) != \
                len(SysTemp['PointPosition'][self.NowPicNum]) or ScrolledCanvas.ISIN:
            PointPosition = ScrolledCanvas.Num2Position()
            data = DBHCalculation.output(PointPosition, CamInfo, TreeNo)
            SysTemp['CalcuData'][self.NowPicNum] = data
        else:
            data = SysTemp['CalcuData'][self.NowPicNum]
        # show in table
        TableInfo.clearTree()
        for values in data:
            TableInfo.Tree.insert('', 'end', values=values)

    def AddPic(self):
        NewPicdirlist = list(askopenfilenames(title='Select pictures',
                                              filetypes=[('jpg files', '.jpg'),
                                                         ('png files', '.png'),
                                                         ('tif files','.tif')]))
        NewPicdir = []
        for Dir in NewPicdirlist:
            if Dir not in SysTemp['photos']:
                NewPicdir.append(Dir)
        if len(NewPicdir) != 0:
            self.AddPicButton(NewPicdir)
```

```
    def AddSysTempInfo(self):
        SysTemp['PointPosition'].append([])
        SysTemp['CamInfo'].append([])
        SysTemp['CalcuData'].append([])
        SysTemp['CtrlOnOff'].append([])
        SysTemp['Rotate'].append(0)
        SysTemp['TreeNo.'].append([])

    def my_except_hook(type, value, tb):
        exception_string = "".join(traceback.format_exception(type, value, tb))
        showerror('Error!',exception_string)
    sys.excepthook = my_except_hook

class TableInfo(Frame):
    LabIndex = ['Camera Model', 'Focal Length (mm)', 'Focal X (mm)', 'Focal Y (mm)']
    def __init__(self, parent=None):
        Frame.__init__(self, parent)
        MainFrame = Frame(parent,bg='white')
        MainFrame.pack(fill=BOTH, expand=YES)

        # CamInfoFrame panel
        CamInfoFrame = Frame(MainFrame)
        CamInfoFrame.pack(side=TOP, fill=BOTH, expand=YES)
        # Model
        lab = Label(CamInfoFrame, text=self.LabIndex[0], relief=RIDGE, bg='white')
        self.Ml = Entry(CamInfoFrame, text='', relief=SUNKEN, bg='pink', justify='center')
        lab.pack(side=TOP, expand=YES, fill=BOTH)
        self.Ml.pack(side=TOP, expand=YES, fill=BOTH)
        # FocalLength
        lab = Label(CamInfoFrame, text=self.LabIndex[1], relief=RIDGE, bg='white')
        self.FL = Entry(CamInfoFrame, text='', relief=SUNKEN, bg='pink', justify='center')
        lab.pack(side=TOP, expand=YES, fill=BOTH)
        self.FL.pack(side=TOP, expand=YES, fill=BOTH)
        # FocalX
        lab = Label(CamInfoFrame, text=self.LabIndex[2], relief=RIDGE, bg='white')
        self.FX = Entry(CamInfoFrame, text='', relief=SUNKEN, bg='pink', justify='center')
        lab.pack(side=TOP, expand=YES, fill=BOTH)
        self.FX.pack(side=TOP, expand=YES, fill=BOTH)
        # FocalY
        lab = Label(CamInfoFrame, text=self.LabIndex[3], relief=RIDGE, bg='white')
        self.FY = Entry(CamInfoFrame, text='', relief=SUNKEN, bg='pink', justify='center')
        lab.pack(side=TOP, expand=YES, fill=BOTH)
        self.FY.pack(side=TOP, expand=YES, fill=BOTH)
```

```
        # Result Panel
        Tree = ttk.Treeview(MainFrame, show="headings", columns=('No.', 'Angle', 'Distance', 'DBH'))
        Tree.bind("<Double-Button-1>", self.on_tree_select)
        Tree['columns'] = ('No.', 'Angle', 'Distance', 'DBH')

        Tree.column('No.', width=50, anchor='center')
        Tree.column('Angle', width=50, anchor='center')
        Tree.column('Distance', width=60, anchor='center')
        Tree.column('DBH', width=60, anchor='center')
        Tree.heading('No.', text='No.')
        Tree.heading('Angle', text='Angle')
        Tree.heading('Distance', text='Distance')
        Tree.heading('DBH', text='DBH')
        Refresh = Label(MainFrame, text='Double click to change No.', bg='white')
        Refresh.pack(side=BOTTOM, fill=X, expand=YES)
        Tree.pack(side=BOTTOM, fill=BOTH, expand=YES)

        self.Tree = Tree

    def clearTree(self):
        TableInfo.Tree.delete(*TableInfo.Tree.get_children())

    def on_tree_select(self,event):
        TreeNo = askstring('Notice', 'Change the tree number to')
        if TreeNo != None:
            if TreeNo == '':
                showwarning(title='Warning!',message='Input should not be empty!')
            else:
                global SysTemp
                # get selected line data
                curItem = self.Tree.focus()
                curValue = list(self.Tree.item(curItem,'values'))
                # get all values in the table
                allValues = SysTemp['CalcuData'][PicSelectMenu.NowPicNum]
                # get the line number of selected data
                curNum = allValues.index(curValue)
                # refresh tree number
                SysTemp['CalcuData'][PicSelectMenu.NowPicNum][curNum][0] = TreeNo
                SysTemp['TreeNo.'][PicSelectMenu.NowPicNum][curNum] = TreeNo
                # refresh table
                PicSelectMenu.ShowInTable()

if __name__ == '__main__':
    # show trackback in errormessage
```

```
def my_except_hook(type, value, tb):
    exception_string = "".join(traceback.format_exception(type, value, tb))
    showerror('Error!', exception_string)

def quit_save_confirm():
    # Save project file when exit
    if Projectdir != '':    # if user do not choose any project, exist without writing project file
        ans = askyesno('warning', 'Save changes?')
        if ans:
            f = open(Projectdir + '.dbh', 'w')
            f.write(str(SysTemp))
            f.close()
    root.destroy()

global SysTemp, Projectdir
Projectdir=''
SysTemp = {'photos':[],'PointPosition':[],'CamInfo':[],
           'CalcuData':[],'CtrlOnOff':[],'Rotate':[],'TreeNo.':[]}

# main body starts
root = Tk()
root.title('ImageDBH')
root.config(bg='white')
MenuBar = MenuBar(root)
MenuBar.pack(side=TOP, fill=X)
PicSelectMenu = PicSelectMenu(root)
PicSelectMenu.pack(side=BOTTOM)
ScrolledCanvas = ScrolledCanvas(root)
ScrolledCanvas.pack(side=LEFT)
TableInfo = TableInfo(root)
TableInfo.pack(side=RIGHT)
root.protocol("WM_DELETE_WINDOW", quit_save_confirm)
sys.excepthook = my_except_hook
root.mainloop()
```

7.2 DBHCalculation.py

```
# -*- coding:UTF-8 -*-


def Calcu_TanTheta(RotateWH,Attribution):
    RotateX = RotateWH[0]
    RotateY = RotateWH[1]
    PicSize0 = Attribution['Size'][0]
    FocalLength = Attribution['FocalLength']    # (mm)
    if RotateX == PicSize0:    # no rotate or rotate 180
        FPX = Attribution['FPX']    # (mm)
        FPY = Attribution['FPY']    # (mm)
    else: # rotate 90°
        FPX = Attribution['FPY']    # (mm)
        FPY = Attribution['FPX']    # (mm)
    #   Theta = 1/2 FOV(Field of View)
    TanThetaX = 0.5 * FPX / FocalLength
    TanThetaY = 0.5 * FPY / FocalLength
    a = (FocalLength/FPY)*RotateY*45
    return TanThetaX,TanThetaY,a

def Judge_PointRealPosition(OneTreePoints):
    # ----------coordinate---------
    #               ● → x (bigger)
    #               ↓
    #               y (bigger)
    # ------------------------------
    # Average Y of up edge points bigger than Y of centre point => edge points lower => is real up points series
    if (OneTreePoints[0][1] + OneTreePoints[1][1]) / 2 > OneTreePoints[2][1]:
        UC = OneTreePoints[2]    # Y_up_centre
        DC = OneTreePoints[5]    # Y_down_centre
        # Judge which Up edge point is left
        if OneTreePoints[0][0] < OneTreePoints[1][0]: # Edge point1 X   < Edge point2 X
            UEL = OneTreePoints[0]    # Up edge point1 (Left)
            UER = OneTreePoints[1]    # Up edge point2 (Right)
        else:
            UEL = OneTreePoints[1]    # Up edge point1 (Left)
            UER = OneTreePoints[0]    # Up edge point2 (Right)
        # Judge which Down Point is left
        if OneTreePoints[3][0] < OneTreePoints[4][0]: # Edge point1 X   < Edge point2 X
            DEL = OneTreePoints[3]    # down edge point1 (Left)
            DER = OneTreePoints[4]    # down edge point2 (Right)
        else:
            DEL = OneTreePoints[4]    # down edge point1 (Left)
```

```
            DER = OneTreePoints[3]       # down edge point2 (Right)
    # average Y of up edge points smaller than Y of centre point => edge points higher => is opposite up points series
    else:
        UC = OneTreePoints[5]       # Y_up_centre
        DC = OneTreePoints[2]       # Y_down_centre
        # Judge which Up edge point is left
        if OneTreePoints[3][0] < OneTreePoints[4][0]: # Edge point1 X   < Edge point2 X
            UEL = OneTreePoints[3]       # Up edge point1 (Left)
            UER = OneTreePoints[4]       # Up edge point2 (Right)
        else:
            UEL = OneTreePoints[4]       # Up edge point1 (Left)
            UER = OneTreePoints[3]       # Up edge point2 (Right)
        # Judge which Down Point is left
        if OneTreePoints[0][0] < OneTreePoints[1][0]: # Edge point1 X   < Edge point2 X
            DEL = OneTreePoints[0]       # down edge point1 (Left)
            DER = OneTreePoints[1]       # down edge point2 (Right)
        else:
            DEL = OneTreePoints[1]       # down edge point1 (Left)
            DER = OneTreePoints[0]       # down edge point2 (Right)
    COC = [OneTreePoints[6][0]/2,OneTreePoints[6][1]/2]       # centre of camera
    ReadyData = {'UEL':UEL,'UER':UER,'DEL':DEL,'DER':DER,'UC':UC,'DC':DC,'COC':COC}
    return ReadyData

def Calcu_Angle(ReadyData):
    Angle = 0
    return Angle

def Calcu_DBH(ReadyData,Yscale,Dc,De):
    Xuel = ReadyData['UEL'][0]
    Xuer = ReadyData['UER'][0]
    Xdel = ReadyData['DEL'][0]
    Xder = ReadyData['DER'][0]
    Avg_x = (1/3)*(Xuer-Xuel)+(2/3)*(Xder-Xdel)
    StringX = Avg_x*Yscale['YE']
    d = 0.5*StringX
    '''
              ●
             /|   ↘
      R    / d        ↘ De
         /   |           ↘
        ○--┴------)---★
        |-      R       -|   Dc
    d*(R+Dc)=R*De => R = (Dc*d)/(De-d)
    '''
```

```
    R = (Dc * d) / (De - d)
    DBH = 2*R
    return DBH

def Calcu_Distance(ReadyData,YScale,TanY):
    YLength = YScale*ReadyData['COC'][1]*2
    D = YLength/(2*TanY)
    return D

def _Calcu_Scale(ReadyData):
    # Calculate the centre point Y scale
    # Yuc-Ymc=30cm, Ydc-Ymc=15cm, Yuc-Ydc=45cm
    Yuc = ReadyData['UC'][1]
    Ydc = ReadyData['DC'][1]
    Ymc = ReadyData['COC'][1]        # centre of camera
    YScale_C3 = 45/abs(Yuc-Ydc)       # (cm/pix)
    YScale_C =YScale_C3
    # Calculate the edge point Y scale
    Yuel = ReadyData['UEL'][1]
    Yuer = ReadyData['UER'][1]
    Ydel = ReadyData['DEL'][1]
    Yder = ReadyData['DER'][1]
    YScale_EL3 = 45/abs(Yuel-Ydel)      # (cm/pix)
    YScale_ER3 = 45/abs(Yuer-Yder)      # (cm/pix)
    YScale_E = (YScale_EL3 + YScale_ER3)/2
    YScale = {'YE':YScale_E,'YC':YScale_C}
    return YScale

def output(PointPosition, CamInfo, TreeNo):
    # Calcudata=[[No., Angle, Distance, DBH].
    #                   [No., Angle, Distance, DBH]...]
    CalcuData = []
    for i in range(len(PointPosition)):
        TanX, TanY, a = Calcu_TanTheta(PointPosition[i][6], CamInfo)
        Ready_Data = Judge_PointRealPosition(PointPosition[i])

        YScale = _Calcu_Scale(Ready_Data)
        Ag = Calcu_Angle(Ready_Data)
        Dc = Calcu_Distance(Ready_Data,YScale['YC'],TanY)
        De = Calcu_Distance(Ready_Data,YScale['YE'],TanY)

        DBH = Calcu_DBH(Ready_Data, YScale, Dc, De)

        CalcuData.append([str(TreeNo[i]), str(Ag)+'°', str(round(Dc/100,2))+'m', str(round(DBH,2))+'cm'])
```

```
    return CalcuData

if __name__ == '__main__':
    # test data
    PointPosition = [
        [[940.0, 1295.0], [1070.0, 1296.0], [1005.0, 1291.0], [952.0, 1486.0], [1117.0, 1483.0], [1035.0, 1488.0],
         [1944, 2592]]]
    CamInfo={'FocalLength': 5.8, 'FPY': 4.2672, 'Model': 'Canon PowerShot SD430 WIRELESS', 'FPX': 5.715,
                                'Size': (2592, 1944)}
    output(PointPosition,CamInfo)
```