A Robotic-Arm-Based Visual System for Plant Monitoring

Nicha Piemkaroonwong
Pannatorn Suparatnodom
Mathara Rojanamontien

Bachelor of Engineering in Software Engineering,
International College,
King Mongkut's Institute of Technology Ladkrabang,
Academic Year 2015

Thesis - Academic Year 2015

Bachelor of Engineering in Software Engineering

Internation College

King Mongkut's Institute of Technology Ladkrabang

Title: A Robotic Arm-Based Visual System for Plant Monitoring

Authors:

1. Nicha Piemkaroonwong Student ID: 55090021
2. Pannatorn Suparatnodom Student ID: 55090029
3. Mathara Rojanamontien Student ID: 55090040

Approved for submission

.....

(Dr. Ukrit Watchareeruetai)

Advisor

Date 30/5/2016

Abstract

Nicha Piemkaroonwong 55090021
Pannatorn Suparatnodom 55090029
Mathara Rojanamontien 55090040
Dr. Ukrit Watchareeruetai Advisor
Academic Year 2015

Keywords: robotic, robotic arm, image processing, image segmentation, black gram, leaf segmentation, plant monitoring, agriculture.

In agriculture, harvesting, monitoring, and taking care of plants can be considered as simple routine tasks that require a large amount of works and abundantly consume time. Thus, in order to reduce human efforts, robots are recently used as tools for doing the tasks.

This thesis proposed an automatic system for monitoring plants from plants' leaves using a robotic arm. In the system, a camera is attached at the robotic arm's tip to obtain images of its environment. However, as a great processing power is needed, an image obtained from the camera cannot be interpreted by the robot itself. Hence, the system includes a computer system, which is used to process the image.

The system consists of three main units: a robot arm unit, a plant rotation unit, and a processing unit. The processing unit is the central unit which coordinates all three units. It processes all information, performs necessary calculation, and commands the robot arm unit and the plant rotation unit. The commands are sent to the other two units over wireless communication channels, which were established in advanced at the beginning of operations. According to the commands sent from the processing unit, the robotic arm moves along a robot rail, adjusts its joints' positions, and captures plants' images, while the plant rotation unit rotates plants and allows each plant to be captured from

various directions. Later, the plants images taken by the robot arm unit are analyzed and processed by the processing unit to segment the leaves out as outputs of the system.

In order to control the robot arm unit, the processing unit has to solve kinematic problems and to generate the segmented leaves images, it has to perform image processing including image preprocessing, mid-vein detection, leaf detection, and leaf segmentation. Firstly, the image preprocessing prepares different formats of input images for the other processes. Next, the mid-vein detection detects the mid-vein of a leaf in an input image. Then, the leaf detection uses the detected mid-vein and a polygon model of black gram to estimate the leaf area. Finally, from the estimated leaf area, the leaf segmentation uses snake algorithm to find an actual area and segments the leaf to generate an output.

Despite the fact that the proposed system only works with black gram plants, it should be able to work with any kinds of plants by changing the model accordingly. Overall from where the system stands, it is useful as a plant monitoring tool in the agricultural field.

Acknowledgments

In order to complete the proposed system, Robotic Arm-Based Visual System for Plant Monitoring, we had done a lot of work and research. On the way, we also ran across various problems but were able to find our ways out. However, without supports and care from many people, it would be impossible to do so. For this reason, we would like to give our sincere gratitude to all of them.

The first person we would like to thank is our advisor, Dr. Ukrit Watchareeruetai, who gave us his provision of expertise and technical support in the implementation, as well as his books and other materials. Without his knowledge and experience, this project will not become reality.

We are also thankful to our lecturers, Dr. Natthapong Jungteerapanich, Dr. Chaiwat Nuthong, and Dr. Isara Anantavrasilp for their logistical support and for providing necessary equipments.

Additionally, we acknowledge the kindness of International College staffs for providing us a suitable working space without difficulty and for lending us essential items we requested for.

Nevertheless, we express our gratitude toward our families and colleagues for their kind co-operation, motivations, suggestions, and encouragement, which help us in completion of this project.

May 23, 2016

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides an introduction to the project. It presents the motivation, problem description, and an overview of a proposed system. In addition, it describes the objectives, scope, and expected benefits of the project.

1.1 Motivation and problem description

Robota is a Czech word which means forced labor [30]. It is where a word robot comes from. From the meaning, a robot can be described as a tool for heavy, repetitive, manufacturing works, or as a tool for dangerous and/or boring routine tasks that are difficult for human to work on. In agriculture, robotics can be applied to manage, organize, and take care of plants. For example, in China, fruit cultivation areas have reached 8–9 million hectares since 1993, and harvesting tasks take around 50–70% of total working hours and mainly depends on manual labour [39]. Thus, there is a strong need of automatic systems for harvesting, which is a problem that robots come in handy. Therefore, a lot of researches on harvesting robots have been carried.

In 1984, Kawamura, Namikawa, Fujiura, and Ura [16] first developed a fruit harvesting robot for orchards, and in 1998, Monta et al. developed a robot for tomato

harvesting [25]. In 2003, a cucumber harvesting robot was developed by Van Lenten, Hemming, Kornet, Bontsema, Van Tuijl, and Van Os [37]. Recent developments of harvesting robots include a strawberry harvesting robot (Hayashi et al. in 2010) [11], a cherry harvesting robot (Tanigaki, Fujiura, Akse, and Imagawa in 2008) [16], and an apple harvesting robot (Zhao et al. in 2011) [42].

There is not only harvesting problem, but also a problem of weeds in cultivation areas that robotics are considered. Because using chemical in farming harms the world, automatic weed control systems are considered. One of the systems was developed by Lee, Slaughter, and Giles [20], which is a robotic weed control system for tomato fields.

Furthermore, there is another problem about increasing crops production. In this problem, there is a need of plants measurement and monitoring to allow a costs-versus-benefits analysis. This problem can also be solved using robots. Recently, Ayan et al. developed the first robotic system for 3D plant growth measurement using a laser scanner and a robot arm [6]. All the robotic systems mentioned before use computer vision techniques to analyze images of agricultural products.

As described earlier, robots have been introduced to reduce the difficulty of human tasks. To achieve the tasks, robots must also process various sorts of visual information [4]. Normally, a camera is attached to a robot in order to obtain images of its environment. An image obtained from the camera cannot immediately be interpreted by the robot itself. However, there is one approach to solve the problem, which is to use a computer system to do an image processing to interpret the image.

Image processing is the science of modifying images by means of a digital computer [22]. It is referred to as digital image processing in case that the image is a digital image. Purposes of image processing are either to improve an images quality for better human interpretation or to makes it more suitable for autonomous machine perception [23], for example, reducing the amount of noise in an image, de-blurring, or sharpening an image to observe more image details, and extracting edges in an image for further analysis.

There are several applications of image processing which aim to solve agricultural problems. In 2015, Khirade and Patile use image processing techniques to detect disease in plant [18], and Singh, Varsha, and Misra use this technique to detect unhealthy region of plant [33]. Another application in image progressing is to use image processing to classify an object in an image, for instance, classification of bamboo plant in 2011 [31], and classification of medicinal plants leaf in 2012 [10]. Moreover, image processing technique is also used to extract plant leaf feature [43], leaf recognition and characterization [36], and disease identification [27].

In this project, an automatic system with a robotic arm is developed to monitor plants instead of humans. The system is designed to reduce time, money, and energy wasted by humans from doing unnecessary simple routine tasks. It consists of three units including a robot arm unit, a plant rotation unit, and a processing unit, as shown in Fig. 1.1. The robot arm unit has a camera attached on its tip as an end effector. The robot arm moves along a planting bed and uses the camera to take photos of each plant in the bed, focusing on its leaves. While the robot arm is taking photos of each plant, the plant rotation unit rotates the plant for the arm and lets the robot arm takes photos from different directions. After each photo is taken, the processing unit gets the photo from the robot arm unit then segments the area of leaves from the background in the photo, and gives segmented leaf images as the system output.

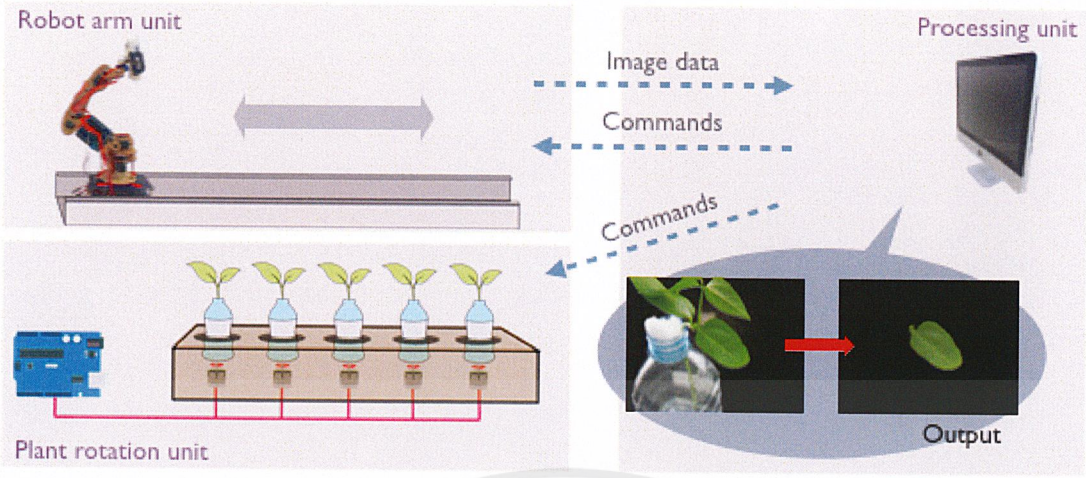


Figure 1.1: System overview

It is to be hoped that this project will contribute to the development of agricultural research, and improve the standard of agricultural activities and related business in Thailand.

1.2 Objectives and scope of work

The objectives include the following:

1. To study and develop an automatic visual-based system that is beneficial in the following:
    - Monitor plants' leaves for further works.
    - Reduce simple routine tasks of people such as taking photos of plants every day.
2. To study and develop image processing and segmentation algorithm for segmenting leaves from the image.
3. To promote the use of electronics and computer technology for agriculture.

The scope of this thesis consists of the following:

1. The type of those plants is black gram (Scientific name: *Vigna Mungo*). Black gram is one of ingredients which has high demand in markets. Furthermore, the project's period is around 240 days and black gram has a very short life span (around 90–120 days) [32], so we can take photos of leaves in various stages of plants' growth to reduce bias of leaf segmentation experiment.

2. The plants' height must not exceed the robot arm's height, so the robot will be able to take a photo of leaves on the top of the plant.

3. The plants' photos must be taken under a controlled environment. In this project, the light intensity is fixed, and the light color is white.

4. When taking photos of plants, everything behind the plants must be solid black. A contrast of leaves and background is high, then a processing can segment a leaf from background easier than non-solid background.

Expected benefits of this system are for people to use it to segment leaves' images from the background. After the images have been segmented by the system, it can be used in various ways, for example, nutrient deficiency detection in plant using its leaf images and an application that records the growth in plant.

1.3 Thesis structure

The rest of the thesis is organized as follows:

Chapter 2 provides all necessary background knowledge requires to implement the system. Is also includes literature reviews and works related to the topic of this thesis. Chapter 3 describes the system in more detail, as well as, provides information of methods and algorithms used to implement the system. Chapter 4 explains six experiments that were done to find suitable values of the system's parameters and to verity the

performance of the chosen algorithms, as well as the performance of the whole system. Lastly, Chapter 5 concludes the thesis's content and suggests ways of improvement.

Chapter 2

Background knowledge and literature reviews

This chapter provides background knowledge for this project. It is separated into two main parts. The first part is focusing on robotics, divided into two subsections, it describes the actuators used to build the system and the general knowledge about robotic arms. Afterward, the second part is focusing on image analysis, describes computer vision and image processing techniques and tools used in the system.

2.1 Robot control

2.1.1 Actuators

So far, there is a common consent on robots definition. In approving, robots are electromechanical devices which have abilities to perceive its environment, and makes decisions or actions in order to achieve a specific task. This means that a mouse or a fan are not classified as robots since they are not sense their environment. In contrast, a vacuum cleaner that can navigate around a room, a car that can perceive obstacles

around itself, or a robotic arms with sensors are considered as a robotic system.

In order to build a robot, there are needs to determine what the robot task is and what components are required to construct and to make the robot moves. *Actuators* is a general term of the devices that drive robots. By definition, an actuator is a device that converts electrical energy into physical motions. The greater number of actuators make either rotational or linear motion. Choosing the right actuators requires some background knowledge of what actuators are available and what are their functions [13]. For this reason, this subsection marks out the devices used to create the proposed system.

2.1.1.1 DC motors

DC motors convert direct current into mechanical power. Most of them depend on the forces produced by magnetic fields and produce rotary motion. DC motors can operate in both clockwise and counter clockwise rotation. The rotation speed can be controlled by changing the strength of current applying to the motors. DC motors are used in many applications including toys, machines in factories, robotic systems, and etc.

However, a micro-controller cannot drive the motors directly if there is a need to change the rotational directions; therefore, drivers are needed to control the speed and direction of motors. L293D is one of the common chips used for controlling DC motors. This chip is designed to control two motors. A L293D chip has two sets of arrangements with each of them has a set of two input pins, two output pins, and an enable pin as shown in Fig. 2.1b. The output pins are connected to DC motors, the enable pins enable the rotation, and the input pins control the rotational direction of the motors [7].

2.1.1.2 R/C servo motors

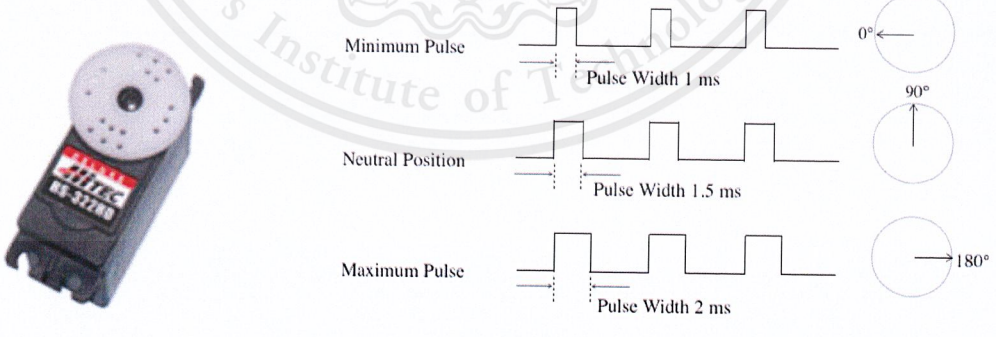
R/C servo motors are actuators that can rotate to a specific angle in both clockwise and counterclockwise directions. Most of them can be rotated only in the range of 0

to 180 degree angles. In a servo motor, it includes a DC motor, a set of gears, and a potentiometer which is used to measure the angle.

Generally, servo motors have three wires: ground, input voltage, and signal. The signal wire is to be supplied with *pulse width modulation* or *PWM* signal. PWM sent to a servo determines the position of the servo shaft and how far its motor turns. The servo motor expects to see a pulse every 20 milliseconds; therefore, a specific PWM signal can be determined to rotate the servo to a desired angle as shown in Fig. 2.2b. By repeatedly sent the specific PWM signal to the servo, the servo rotates to the specific angle. When the servo turns to the angle, it holds that angle, and if an external force is applied against the servo while it is holding the angle, it will resist from moving out of that angle [12].

(a) A DC motor (b) A motor driver, L293D

Figure 2.1: A DC motor and a motor driver [7]



(a) A servo motor [12] (b) PWM signals

Figure 2.2: Servo motor and PWM signals

2.1.1.3 Stepper motors

A stepper motor divides a full rotation into a number of *steps*. All steps turn the motor by an equal angle. Hence, by specifying a number of steps, the stepper motor can be accurately rotated to a specific position. As for how it works, the stepper motor has an *internal rotor* in the middle and has *external magnets* around the rotor. The internal rotor contains permanent magnets while the external magnets are not permanent and only active when enough voltage is supplied. These external magnets will attract the internal rotor and make it rotates by applying voltage to one external magnet at a time. When the internal rotor rotates one time, the stepper motor moves one step. Also, just like DC motors and servo motors, it is able to rotate in both clockwise and counterclockwise by swapping the order of voltage supplies [13].

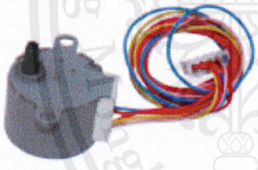
By cause of the step by step rotation, stepper motors offer a very precise angular motion and they are, sometimes, more preferable than servo motors since servos are usually limited to a 0 to 180 degree range, while they can rotate continuously to any angle. Although some of the servo motors that use optical decoder offers higher precision and some of them provide a full 360 degree of rotation, these type of servo motors are very expensive.

There are three step modes for stepper motor including *full-step*, *half-step*, and *micro-step*. The type of step depends on how the motor's driver controlling the stepper motor. For the full-step mode, one pulse from the driver is equal to one full step on the step motor. In order to achieve this, the driver must activate two external magnets at a time and swapping the activated pair alternately. For the half-step mode, one external magnet is activated and then two of them are alternately activated. This cause the internal rotor to move at half the distance of one step. In this mode, the rotation is smoother than it is in full-step mode; however, it provides less torque in comparison. Lastly, micro-step mode rotates the stepper motor in a manner that one normal step is divided into a number of sub-steps. It is usually applied to applications that require high

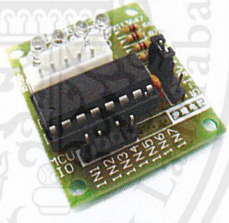
accurate positioning and overly smooth motion over speeds and torque [35].

In addition to the three step modes, there are two types of stepper motors: *bipolar* and *unipolar*. In general, the two types work the same way and one of the differences is their voltage levels. A unipolar stepper motor only operates with positive voltage, hence the high and low voltages used to energize the external magnet are 5V and 0V. On the contrary, a bipolar stepper motor has two polarities, positive and negative, thus the high and low voltages are 2.5V and -2.5V. Other than the voltage levels, there is also a difference in their physical connections as shown in Fig. 2.3c and Fig. 2.3d. Unipolar has an extra wire in the middle of each coil to support the double polarities [14].

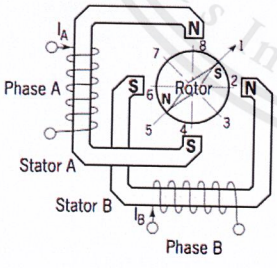
Similar to DC motors, a driver is needed to control a stepper motor. From a microcontroller, step and direction signals are sent to the driver. The driver receives the two signals and converts them into electronic signals for the stepper motor. ULN2003 chip (see Fig. 2.3b) is one of the stepper motor drivers that provides simple control over a stepper motor.



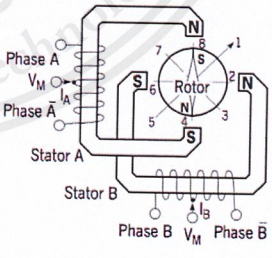
(a) A stepper motor [35]



(b) A stepper motor driver, ULN2003 [1]



(c) A unipolar wound stepper motor [14]



(d) A bipolar wound stepper motor [14]

Figure 2.3: Stepper motor, a stepper motor driver, and the two types of winding

2.1.2 Robotic arm

A robotic arm is a known manipulator consisting of a *base*, *links*, *joints*, and *end effector*, as shown in Fig. 2.4. A base is the arms basic part, which may be fixed on a spot, or be able to move around. An end effector is a part of robot arm attached at the end of a robotic arm to interact with the environment, for example, if the end effector is a gripper, it can be used to perform moving and/or holding action. Links are fixed and supports the arms gripper, and are connected by joints [41].

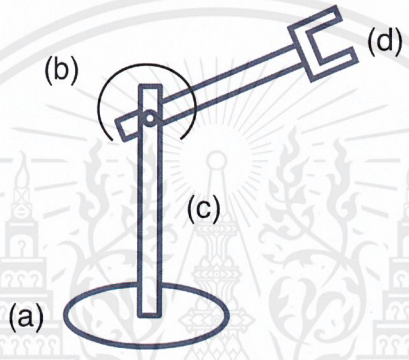


Figure 2.4: Basic robotic arm: (a) a base, (b) a joint, (c) a link, and (d) a gripper

Motions occur at joints by some actuators, which may be motors. With different motors, joints motions are not the same. Basically, there are two kinds of robotic arms joints, revolute joint and prismatic joint (Fig. 2.5). A revolute joint allows a rotary motion, and a prismatic joint allows an extension motion.

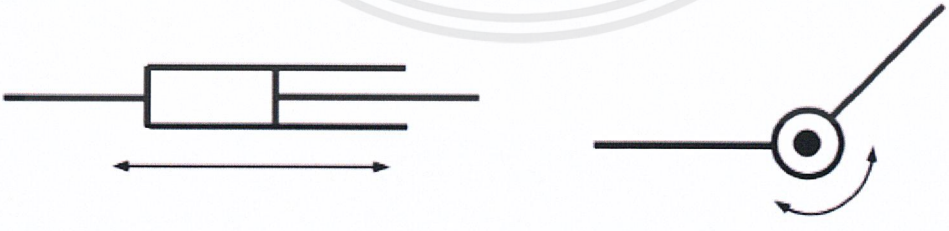


Figure 2.5: Types of joint: (left) a prismatic joint and (right) a revolute joint (figure credited to [3])

2.1.3 Basic manipulator geometry

Links connected by joints can be represented as a kinematic chain that can be opened or closed, depending on a robotic arms configuration.

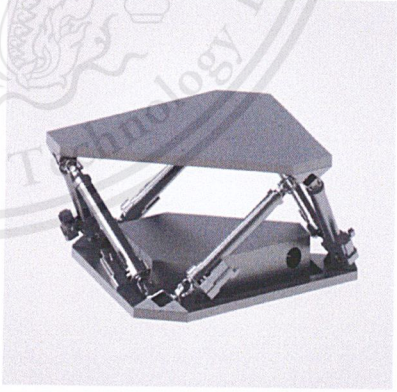
1. Open chain manipulator kinematic

   For open chain manipulator, one end of the chain is the arms base, and another end is the arms gripper, as shown in Fig. 2.6a. For this type, joints can be controlled individually without constraints from other links. A robot manipulator with n joints will have $n+1$ links. For joint i that connects link $i-1$ to link i, the location of joint i is fixed with respect to link $i-1$, and when joint i is actuated, link i moves [34]. Therefore, the final motion can be obtained from the previous motions.

2. Close chain manipulator kinematic

   In a close chain manipulator, motions depend on forces and constraints from other links [41]. Joints cannot be controlled individually because they move at the same time in order to achieve the desired position. An example of this type is a parallel robot, as shown in Fig. 2.6b.

(a) An open chain manipulator



(b) A close chain manipulator (parallel robot, AMEGA steward platform)

Figure 2.6: Kinematic chains

2.1.4 Homogeneous transformation

Homogeneous transformation **H**, is a 4×4 square matrix separated into 4 blocks Eq. (2.1), used to calculate the coordinate value of a robot part. It represents a transformation using a *rotation matrix* **R**, and *translation matrix* **d**, and specifies the orientation and position of a robots part in a space, with respect to the base of the robot. The rotation matrix has a size of 3×3. Its value represents the orientation around each axis. The translation matrix has a size of 3×1. Its value represents the new position of the coordinate system. The last row vector of **H** has a perspective matrix **f** and a scale factor **s**. This row is always fixed to (0, 0, 0, 1) [34].

$$\mathbf{H} = \left[\begin{array}{c|c} \mathbf{R}_{3\times3} & \mathbf{d}_{3\times1} \\ \hline \mathbf{f}_{1\times3} & \mathbf{s}_{1\times1} \end{array}\right] = \left[\begin{array}{c|c} rotation & translation \\ \hline perspective & scale\ factor \end{array}\right] \tag{2.1}$$

A set of basic homogeneous transformations for translation and rotation about x, y and z-axes is shown in Eq. (2.2). $Trans(x, a)$, $Trans(y, b)$, $Trans(z, c)$ are transformation matrices that translate the coordinate system along x-axis by a units, y-axis by b units, and z-axis by b units, respectively. Similarly, $Rot(x, \alpha)$, $Rot(y, \beta)$, $Rot(z, \gamma)$ are transformation matrices that rotate the coordinate system round x-axis by α degrees, y-axis by β degrees, and z-axis by γ degrees, respectively [34].

$$Trans_{x,a} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad ; \quad Rot_{x,\alpha} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_\alpha & -s_\alpha & 0 \\ 0 & s_\alpha & c_\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Trans_{y,b} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad ; \quad Rot_{y,\beta} = \begin{bmatrix} c_\beta & 0 & s_\beta & a \\ 0 & 1 & 0 & 0 \\ -s_\beta & 0 & c_\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$Trans_{z,c} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad ; \quad Rot_{z,\gamma} = \begin{bmatrix} c_\gamma & -s_\gamma & 0 & 0 \\ s_\gamma & c_\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where c_i is $\cos(i)$ for $i = \alpha$, β or γ, and s_i is $\sin(i)$ for $i = \alpha$, β or γ.

2.1.5 Robot kinematics

Kinematic is an analytical study about the motion of a mechanism without consideration of the forces and moments that cause the motion. Robot kinematic is a kinematic of a robot manipulator [19]. It can be separated into two main problems, *forward* or *direct kinematic*, and *inverse kinematic*.

To perform the kinematic analysis, a coordinate frame is attached to each link. Suppose $\mathbf{A}_i$ is a homogeneous transformation matrix of joint i with respect to joint $i-1$. $\mathbf{A}_i$ is not a constant, but can be varied with the change of the robot configuration. Denote $\mathbf{T}_{ij}$ as a transformation matrix of joint i with respect to joint j, and $\mathbf{I}$ as an identity matrix [34].

$$\mathbf{T}_{ij} = \begin{cases} \mathbf{A}_{i+1}\mathbf{A}_{i+2}\ldots\mathbf{A}_{j-1}\mathbf{A}_j & \text{if } i < j, \\ \mathbf{I} & \text{if } i = j, \\ \dfrac{1}{\mathbf{T}_{ij}} & \text{otherwise.} \end{cases} \tag{2.3}$$

2.1.5.1 Forward kinematic

Forward kinematic is a method for determining the orientation and position of an end effector, given the joint angles and length of each link [41]. The objective of forward kinematic analysis is to determine the cumulative effect of the entire set of joint variables [34] or to determine the functions $\mathbf{A}_i$ and multiply them together to get $\mathbf{T}_{ij}$. In an open chain manipulator, there is only one solution; however, in a close chain manipulator, the solution is not unique because one set of coordinates has more different poses of the end effector.

2.1.5.2 Inverse kinematic

In contrast to forward kinematic, inverse kinematic is used when a desired end effector position is given to determine the joint angles required to achieve it. This problem is much difficult compared to the forward kinematic. Moreover, it is more computational expensive and generally takes a very long time. The sources of difficulty are a possible non-existence of a solution when the location required includes non-linear equations of sine and cosine in rotation matrices, an existence of multiple solutions that leads to a complex algorithm to choose an optimal solution, the workspace cannot be reached because of the joints angle restrictions, and a problem of *singularities*. Singularity is a place for infinite acceleration, which can confuse equations or give error to a motor [8]. Since there are cosine and sine functions in the problem, some trigonometric equations are used in the inverse kinematics problem. Some commonly used functions are given in Table 2.1.

Table 2.1: Trigonometric equations used in inverse kinematic problems

	Rules	Equations
1	Symmetry	$\cos(\pi - \theta) = -\cos\theta$
2	Right triangle ratios	$\sin\theta = \frac{\text{opposite}}{\text{hypoteneus}}$, $\cos\theta = \frac{\text{adjacent}}{\text{hypoteneus}}$, $\tan\theta = \frac{\text{opposite}}{\text{adjacent}}$
3	Pythagorean trigonometric identity	$\sin^2\theta + \cos^2 = 1$
4	Pythagorean theorem	$a^2 + b^2 = c^2$ (given a right triangle with side length a,b, and hypoteneus c)
5	Law of cosines	$c^2 = a^2 + b^2 - 2ab\cos\theta$ (given a triange with side lengths a, b, c, and angle across from side c)
6	Angle sum and difference identities	$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$, $\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$, $\tan\alpha \pm \tan\beta = \tan\alpha \pm \tan\beta(1 \mp \tan\alpha\tan\beta)$

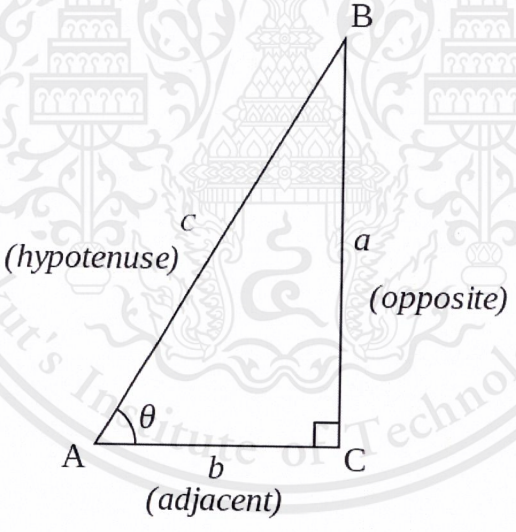


Figure 2.7: A right triangle

Two common approaches are used to solve inverse kinematics problems. The first one is analytical approach which requires a lot of trigonometry and matrix algebra, and the second is iterative approach which is more suitable to use when there are a lot of

links and degree-of-freedoms in a kinematic chain.

2.1.5.3 Analytical approach

An inverse kinematic problem that contains only two or three joints can be analytically solved by drawing a robotic arm with the angles shown on it, then calculating the appropriate angles using geometry and trigonometry. However, when the problem contains more joints, to find the solutions for a joint as a function of the known elements of the end effector, a link of transformations is formed and inverted. $\mathbf{A}_i$ can be found by inverting the rest and multiplying them together with $\mathbf{T}_{ij}$ [34]. Additional constraints of robot joints, such as the joints ranges or limitations, can be added to reduce the number of solutions when many solutions are existing.

$$\mathbf{T}_{ij}\frac{1}{\mathbf{A}_{i+1}\mathbf{A}_{i+2}\ldots\mathbf{A}_{j-1}} = \mathbf{A}_j \tag{2.4}$$

2.1.5.4 Iterative approach

Iterative approach is more popular for programming inverse kinematic problems than analytic approach when there are longer, more complicated chains. The main idea of this approach is to solve the problem in iterations. In each iteration, a robot arm's joint is rotated to move the end effector toward the desired position. The angle of rotation for the joint is calculated as follows.

1. Let $\overrightarrow{v_1}$ be a vector from the joint to the current end effector position.
2. Let $\overrightarrow{v_2}$ be a vector from the joint to the desired end effector position.
3. Take dot product of $\overrightarrow{v_1}$ and $\overrightarrow{v_2}$.
4. Take arccos of the dot product to get the angle to rotate.
5. Take cross product of $\overrightarrow{v_1}$ and $\overrightarrow{v_2}$.

6. Check the sign of the Z element of the cross product for the turning direction.

One of the iterative approach is called *Cyclic Coordinate Descent* (CCD). CCD solves the inverse kinematic problem by trying to minimize the distance between the current end effector position and the desired end effector position. In order to achieve that, the end effector has to be moved to an optimal position in each iteration. For a rotational joint, if there is no angle constraint, the end effector can be rotated around in a circle at the joint, as shown in Fig. 2.8. The optimal position of the end effector is the intersection point between the circle and a straight line from the joint to the desired end effector position.

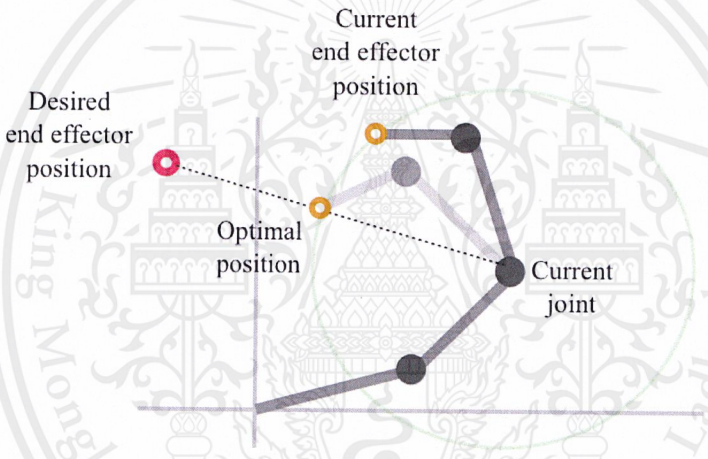


Figure 2.8: Optimal position of the end effector in an iteration

For instance, if joint n is the joint next to the end-effector, joint 0 is the based, and joint $n-1$, joint $n-2$, ..., joint 1 are the joints between them. The first iteration starts with joint n. In the iteration, the joint is rotated to move the end effector to a point on a straight line from joint n to the desired end effector position. The next iteration starts with joint $n-1$, same as the first iteration, the joint is rotated to move the end effector to another straight line from joint $n-1$ to the desired end effector position. The same things are done until the base is rotated then go back to rotating joint n again. This

process is repeating until the end effector is close enough to the desired end effector position, the number of iterations reaches a limit, or the end effector position of the current iteration is the same as the previous iteration's. Moreover, it is possible to start with the bigger joints in the arm when the desired position is far away to quickly reduce the distance. In contrast, start with the smaller ones when there are needed for fine adjustments.

2.2 Image analysis

2.2.1 Computer vision and image processing

Computer vision is the analysis of digital images to extract some information from the images automatically [28]. The goal of computer vision is to use computers to be like a human such as learning and being able to make inference and taking actions based on visual inputs [17]. A basic computer vision system contains a camera, a camera interface, and a computer [26].

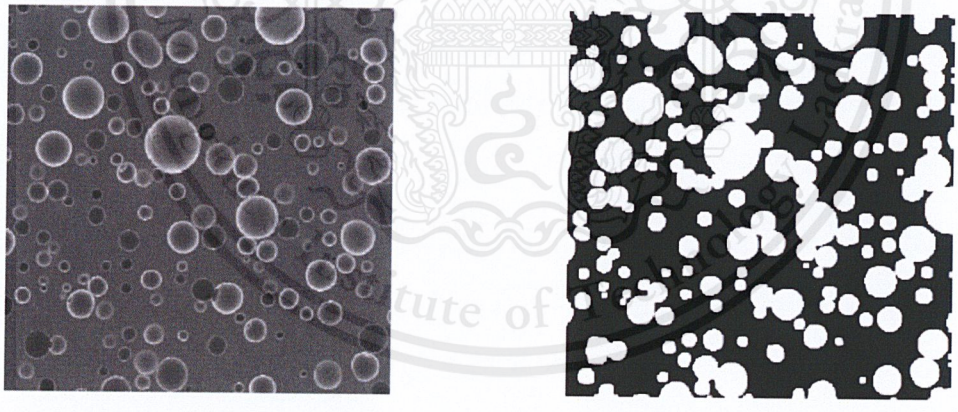
Image processing, or in this case *digital image processing*, is defined as the science of modifying digital images by means of a digital computer [22], a discipline in which both the input and output of a process are images, or process of extracting attributes from images [40] which does not affect other features of that image [17]. The purpose of image processing is either to improve its quality for better human interpretation or to makes it more suitable for autonomous machine perception [23].

However, there is no clear boundary between computer vision and image processing [9]. To make it simpler, both processes can be described as the following three levels [22]. First, a low-level processing which performs primitive operations, for example, noise reduction and contrast enhancement. Second is a middle-level processing, which extracts attributes from images such as edges, contour, and regions. Finally, a high-level processing analyzes and interprets the contents of a scene.

An example of well-known applications in computer vision and image processing are the Facebook facial recognition, fingerprint recognition in a security system, smile detection camera, gesture recognition in Smart TV, and Photoshop.

2.2.2 Image segmentation

In computer vision, an *image segmentation* is a process that subdivides a digital image into multiple components (sets of pixels) which are meaningful with respect to a particular application [21]. In other word, it is used to divide an image into regions. As for the level of division, it depends on the problem being solved. The segmentation should stop when the objects of interest in the application have been solved, as shown in Fig. 2.9. Examples of image segmentation tasks are face detection [38], object detection [2], or fingerprint segmentation [24]. Image segmentation can be conducted using various techniques, such as edge-based segmentation, fuzzy theory based segmentation, neutral network based segmentation, threshold-based segmentation, and region-based segmentation.

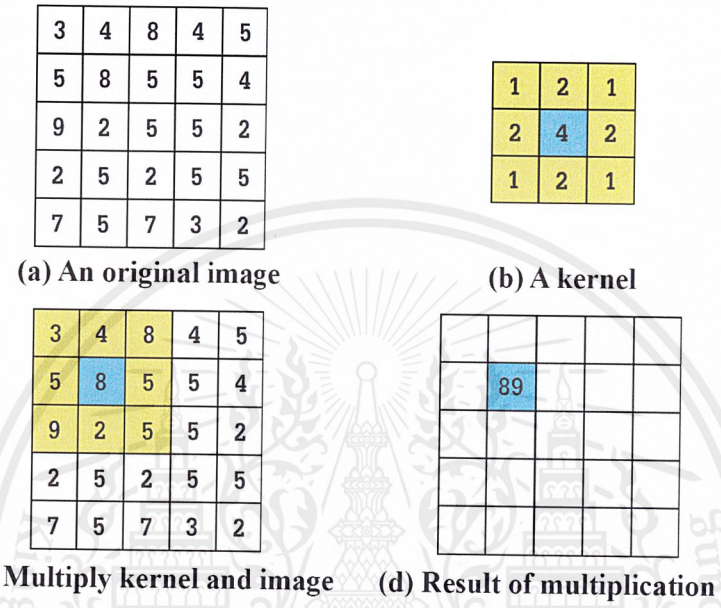


(a) Before applying segmentation (b) After applying segmentation

Figure 2.9: An example of segmentation [29]

2.2.3 Image convolution

Image convolution is a mathematical operation. The operation is done by multiplying a small matrix, called kernel or mask, over an image in spatial domain. The size and the value of the kernel depend on applications.



(a) An original image **(b) A kernel**

(c) Multiply kernel and image **(d) Result of multiplication**

Figure 2.10: An example of image convolution process

If a kernel is symmetry, then it will be placed in a way that its center is at the same position with a target pixel in an image, then pixels in the kernel will be multiplied with the image's pixels at the same position. The result of the convolution is the summation of these multiplications. To make it more understanding, the computation of image convolution is performed using the following formula.

$$p_{ij} = \sum_{j=0}^{N} \sum_{k=0}^{M} (k_{ij} \times m_{ij}), \tag{2.5}$$

where $N \times M$ is the kernel's dimension, p_{ij}, k_{ij}, and m_{ij} are the target pixel's intensity, a kernel pixel's value, and the image pixel's intensity, respectively.

2.2.4 Gaussian filter

Smoothing is a technique which makes an image smoother by applying a kernel, aka a filter, to the image to remove details and noises. Gaussian filter is one of the popular smoothing filters. Relevant to its name, its kernel is created using Gaussian function. In this case, since an image is a 2D representation, 2D Gaussian distribution function is used. The following equation denotes the 2D Gaussian distribution and Fig. 2.11 shows the distribution with mean $(0,0)$ and $\sigma = 1$.

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} exp\left(-\frac{x^2+y^2}{2\sigma^2}\right), \tag{2.6}$$

where (x,y) be a point on 2D plane and σ is the standard deviation of the distribution.

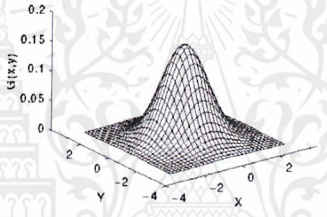


Figure 2.11: Gaussian distribution with mean $(0,0)$ and $\sigma = 1$

Gaussian smoothing uses the above 2D distribution as a *point-spread* function for convolution. Even if in theory, the Gaussian function does not generate zeros, in practice, the results are made zero at approximately more than three standard deviations from the mean; otherwise, an infinitely large convolution kernel is required. Also, the blurring degree can be controlled by adjusting the kernel size and σ. Additionally, a Gaussian filter can be constructed as follows,

$$GC[I]_p = \sum_{q \in S} G_\sigma(\| p - q \|) I_q, \tag{2.7}$$

where S is a set of possible pixels in the image, p is a pixel at the center of a Gaussian

filter, q is a position of neighbor pixels of p, I_q is an intensity of q, and $GC[I]_p$ is a value at position p in the Gaussian filter.

Clarifying, given a pixel p and a neighbor pixel q, if the difference of intensity between p and q is large, then the result from the Gaussian function will be small. Hence, the effect of color intensity of q to p is reduced. On the other hand, if the difference is small, then the result from Gaussian function will be large and the color intensity of q will greatly affect the intensity of p. These happen regardless the identity of the pixels; thus, some of the important information may be eliminated unwarrantably.

2.2.5 Bilateral filter

Bilateral filter is a smoothing filter which was designed based on Gaussian filter. Moreover, unlike other smoothing filters, the bilateral filter smooths an image while preserves edges in the image. The difference between the Gaussian filter and the bilateral filter is that the bilateral filter uses the difference of color intensity to reduce an influence of the neighbor pixel, whereas Gaussian filter does not consider about edge preservation. The formula to create a bilateral filter is denoted by:

$$\begin{aligned} W_p &= \sum_{q \in S} G_{\sigma_s}(\| p-q \|) G_{\sigma_r}(| I_p - I_q |) \\ BF[I]_p &= \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\| p-q \|) G_{\sigma_r}(| I_p - I_q |) I_q, \end{aligned} \tag{2.8}$$

where W_p is a normalization factor that normalizes the result in to a range of 0 to 1, I_p denotes an intensity of p, and $BG[I]_p$ is the value at position p in the Bilateral filter.

2.2.6 Orientation field

An orientation field can be constructed from an image to represent the flow of information in the image. Hence, given an edge image, an orientation field constructed from the image contains the information of the edges' direction. In fact, an element of the orientation field is an orientation computed as follows,

$$\theta = \frac{1}{2} tan^{-1} \left(\frac{\sum_{i=1..N} \sum_{j=1..M} 2g_x(i,j)g_y(i,j)}{\sum_{i=1..N} \sum_{j=1..M} (g_x^2(i,j)g_y^2(i,j))} \right), \tag{2.9}$$

where $N \times M$ is the size of an input image, θ is an orientation of a pixel, and g_x and g_y are gradients of the input image in x direction and y direction, respectively.

2.2.7 Dilation and erosion

In image processing, mathematical morphology refers to a tool which extracts useful components in the representation and description of region shapes, such as boundaries, skeletons, and the convex hull [9]. The morphological techniques are normally used in pre or post image processing. The techniques include morphological filtering, thinning, pruning, and etc. In addition, most of the morphological operations include the two primitive morphological operations, which are dilation and erosion.

Dilation is a morphological operation which is normally applied to a binary or a gray-scale image to bridging gaps and enlarge boundaries, regions, or lines. The dilation of an image A by a structuring element B is defined as,

$$A \oplus B = \{z | (\hat{B})_z \cap A \neq \varnothing\}, \tag{2.10}$$

where z is a set of pixels which are the result of enlarging A by B such that at least one elements in A and B are overlapped.

On the other hand, the boundaries, regions, or lines in the image can be eroded by applying erosion operation. An erosion of A by B is defined as,

$$A \ominus B = \{z|(B)_z \subseteq A\} \tag{2.11}$$

The result of erosion and dilation operations are shown below.

Figure 2.12: A result of two primitive morphological operations. (a) a binary edge image results from Canny edge detector. (b) a binary image after applying dilation operation to the edge image. (c) a binary image after applying erosion operation to the dilated image

Chapter 3

Methodology

In this chapter, a method used to develop the proposed system is described in detail. The chapter is separated into three sections. Each describes the details of robot arm unit, plant rotation unit and processing unit, respectively.

3.1 Robot arm unit

Robot arm unit consists of a robotic arm and a rail. In this section, the detail about the robotic arm is described first, followed by the detail of the rail.

3.1.1 Robotic arm

For the robotic arm, its role is to move the camera to specific locations decided by the processing unit. It consists of three systems including 1) a communication system, 2) a decision making and commanding system, and 3) an interaction system, as shown in Fig. 3.1.

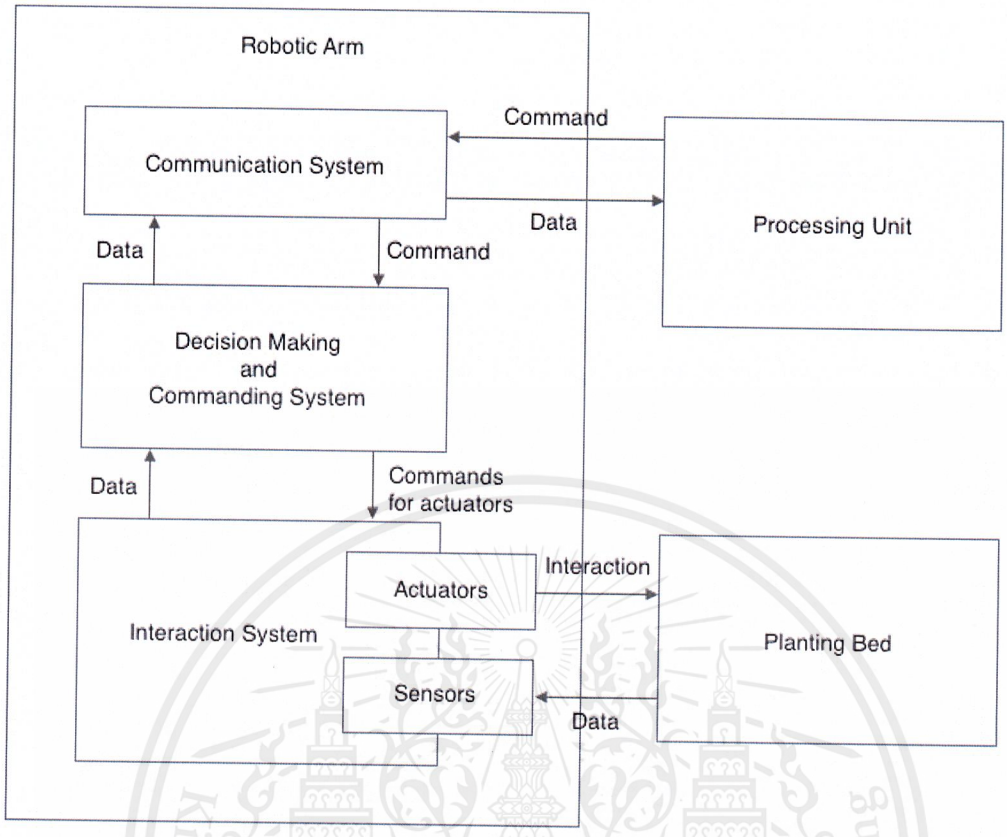


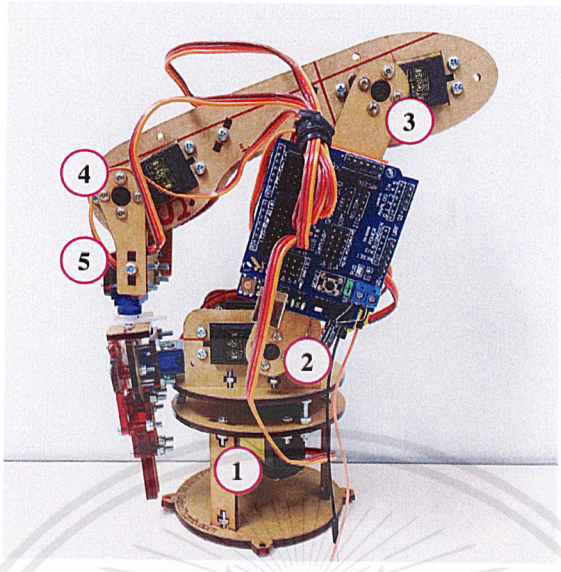
Figure 3.1: Robotic arm structure

The robot arm unit communicates with the processing unit through the communication system. In particular, a command is sent from the processing unit to the arm, and a necessary data is sent from the arm to the processing unit. When the command from the processing unit reaches the communication system, the decision making and commanding system fetches the command and makes a decision whether 1) to move the robot arm to the next position, 2) to move the camera to a specific location, or 3) to stop moving. After the decision is made, a command is sent by the decision making and commanding system to the interaction system. The interaction system consists of sensors and actuators. It controls the actuators according to the command. After the actuators finished moving, the sensors receive information from the environment. Then the information is sent back along the way to the processing unit.

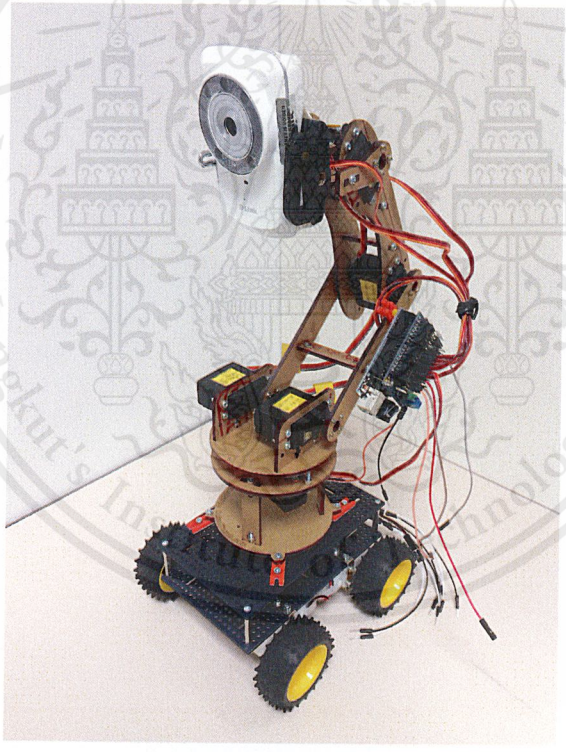
Table 3.1: Robot arm components

Communication system	Adafruit CC3000 WiFi module
Decision making and commanding system	Arduino UNO micro controller
Actuators	MG995 servo motors (joints)
	DC motors (wheels)
Sensors	Sharp 2Y0A21YKF distance measuring sensor
	DLink DCS932L wireless IP camera
Interacted environment	Plants
	A rail

In this project, a five-joint robotic arm with six servo motors (there is a joint that needs two motors to actuate) is used, as shown in Fig. 3.2a. The specification of the robotic arm is shown in Section 3.1.1.1 The fifth joint is replaced by a camera along with a distance measuring sensor which is used to find the distance between the camera and a leaf for the processing unit. An Arduino UNO micro-controller board is used as a controller of the arm. The controller is used with two shields (peripheral boards for Arduino) including a WiFi shield and a sensor shield. The WiFi shield is used to allow the communication between the robot arm unit and the processing unit over WiFi, while the sensor shield is used to increase the available slots for actuators and sensors. As shown in Fig. 3.2b, the robotic arm is then attached on a platform with four wheels which are operated by four DC motors and a L293D driver. The servo motors and the DC motors are controlled by the Arduino using pulse width modulation (PWM) signal. The choice of components to develop each system is shown in Table 3.1. The schematic diagram in Fig. 3.3 shows the connection of the motors in detail.



(a) A five-joint robotic arm



(b) With four-wheel platform

Figure 3.2: The robotic arm used in this project

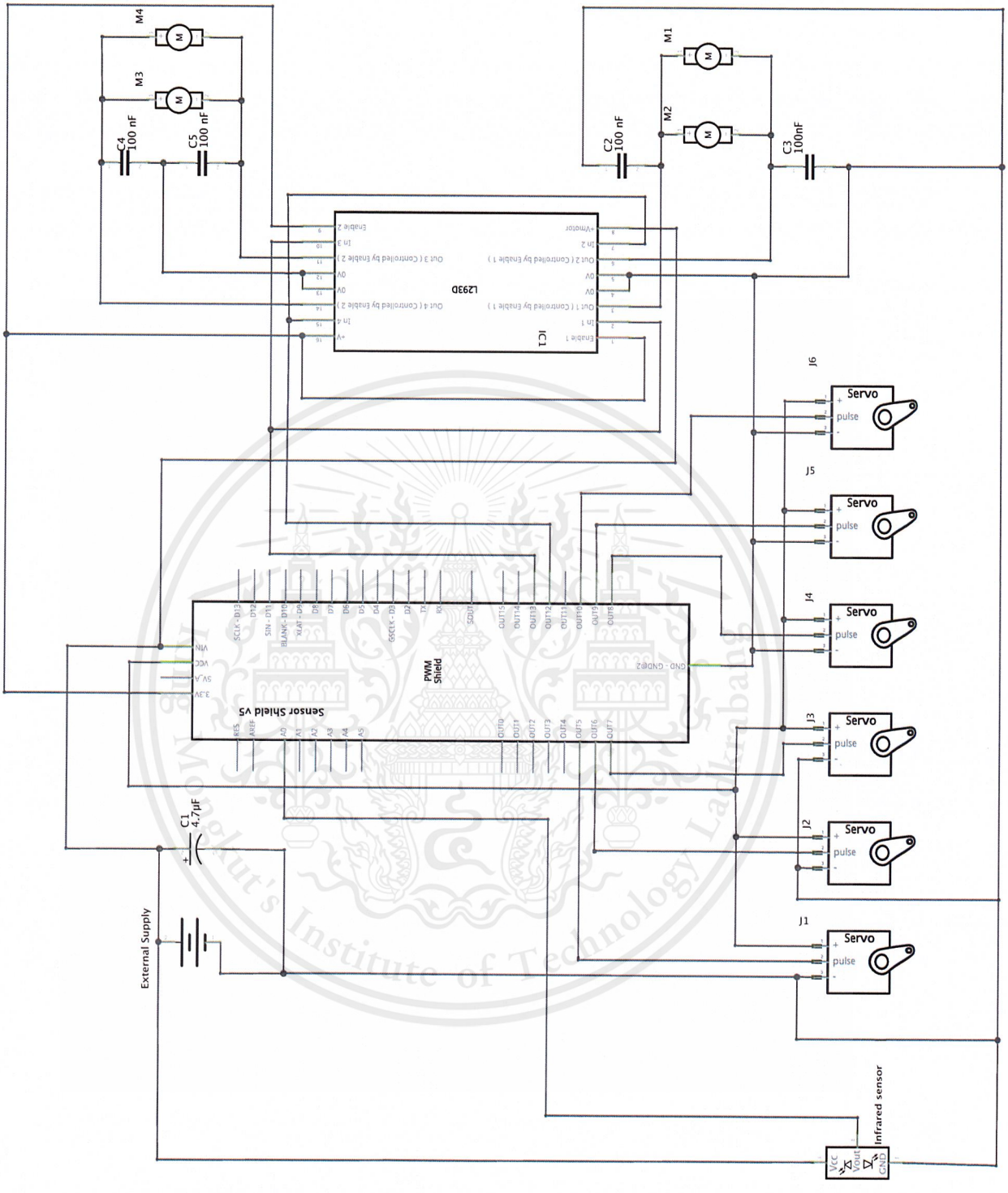


Figure 3.3: Schematic diagram of the robotic arm (only important connections are shown)

3.1.1.1 Robotic arm control

Since the fifth joint is replaced by the camera, the number of joints reduce to four. The joints' rotate direction are shown in Fig. 3.4, the joints angle restriction are shown in Table 3.2, and the links' length are shown in Table 3.3.

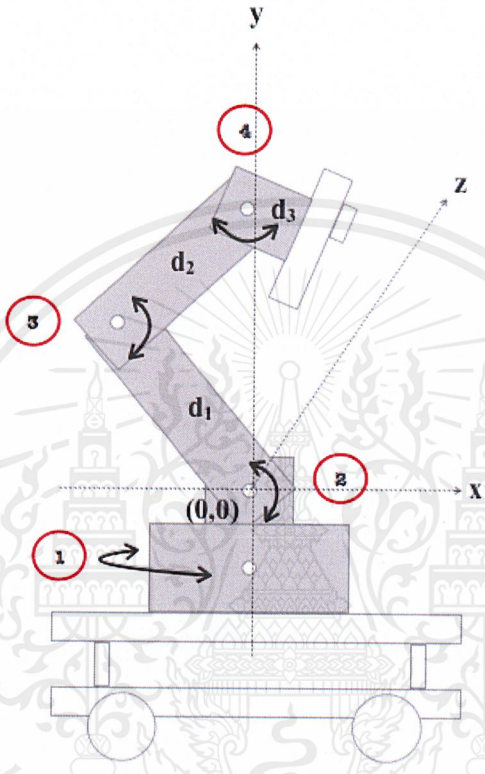


Figure 3.4: A model of the robotic arm used in the system

Table 3.2: Robotic arm angle restrictions in degree.

Joints	Angles
1	[0 , 90]°
2	[55,140]°
3	[50,130]°
4	[0 ,180]°

Table 3.3: Robotic arm links' length in cm. (measured at rotation point)

Links	Lengths
$d1$ (joint 1 to joint 2)	$\sim$ 3.2 cm.
$d2$ (joint 2 to joint 3)	$\sim$ 12.3 cm.
$d3$ (joint 3 to joint 4)	$\sim$ 12.0 cm.
$d4$ (joint 4 to joint 5(camera))	$\sim$ 7.0 cm.

Control type	A	B

Figure 3.5: Command frame

3.1.1.2 How does the processing unit controls the robotic arm?

In order to control the robotic arm unit, the processing unit has to sent a command using a specific format as shown in Fig. 3.5 The control type can be 1 to control the whole unit, 2 to control the platform, or 3 to control the joints' angle. When the control type is 1, A represents the function and B is ignored. The function can be *ready* (1), *sleep* (2), *home* (3), or *disconnect* for the robotic arm to enter the *ready state* in which every joints' angle are set to 90 °, for the robotic arm to enter the *sleep state* in which every joints are set to their minimum angles (see Table 3.2), for the robotic arm to move back to the home position at the end of the rail and enter sleep state, and for the robotic arm unit to stop the communication between the processing unit and itself. When the control type is 2, A represents the joint number and B represents the angle to be rotated. The joint number can be any number in the range of [1,4] and the possible angle of each joint depends on the angle restriction. When the control type is 3, A represents the platform moving direction while B represents the number of steps for the platform to move. The direction can be 0, 1 or 2 which stand for *stop moving*, *moving forward* and *moving backward*, respectively.

Furthermore, when the communication between the robot arm unit and the processing unit takes place, a communication status is sent. The communication status can be

either 0, 1, 2, 3, or 4, for *STATE_NORMAL*, *SENT_COMMAND*, *ACK_AND_RECEIVED*, *ACK_AND_WAIT*, and *FINISH_EXEC*, respectively. When STATE_NORMAL presents, it means there are no interaction between the processing unit and the robot arm unit, when SENT_COMMAND is there, it means there is a command wait for the robot arm unit to fetch, when ACK_AND_RECEIVED is sent, its means the robot arm is already received the command, when ACK_AND_WAIT is sent, its means the processing unit acknowledges the robot arm unit acceptance and is waiting for it to complete its task, and if the status is FINISH_EXEC, it means the robot arm unit is already finish the task, thus the processing unit can get an image from the camera attached at the robotic arm's tip and process the image to generate the system output.

Basically, along with a command, the processing unit has to send the SENT_COMMAND status to the robotic arm. Then when the robotic arm unit notices the status, it fetches the command and sends ACK_AND_RECEIVED status back to notify the processing unit that the command is received properly. After the processing unit noticed this, it sends ACK_AND_WAIT to tell the robot arm unit to start working. Follow after these, when the robot arm unit observes the communication status and knows that the processing unit is waiting, it interprets the command it fetched before, executes the command, and sends back FINISH_EXEC to the processing unit. Now, after the processing unit received the FINISH_EXEC, it sets the communication status to STATE_NORMAL, gets an images, and does its main task, which is explained later in Section 3.3.

On account of these statements, the robot arm unit control program can be expressed as in Fig. 3.6 and the interaction between the processing unit and the robot arm unit can be represented by Fig. 3.7.

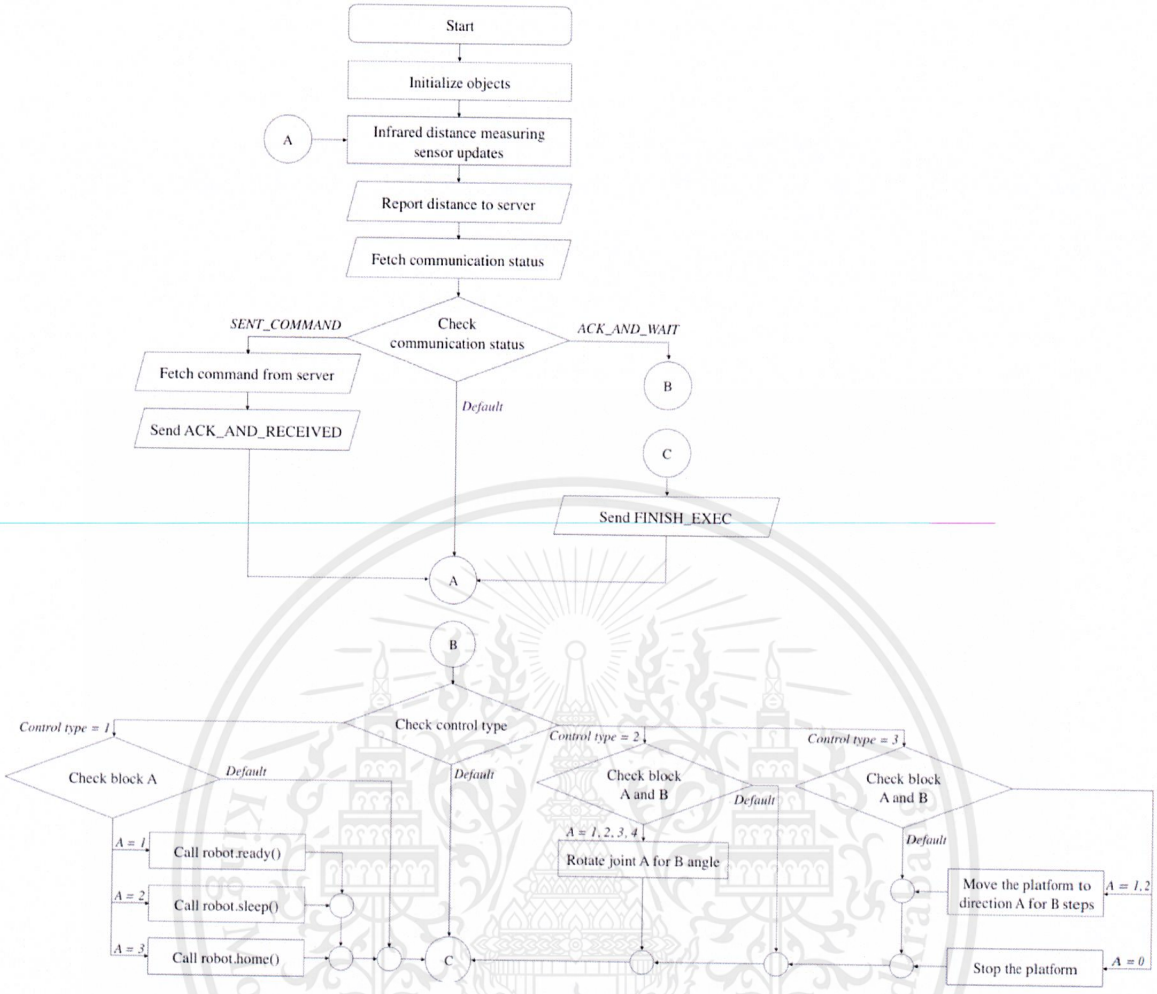


Figure 3.6: The robotic arm unit's program flow

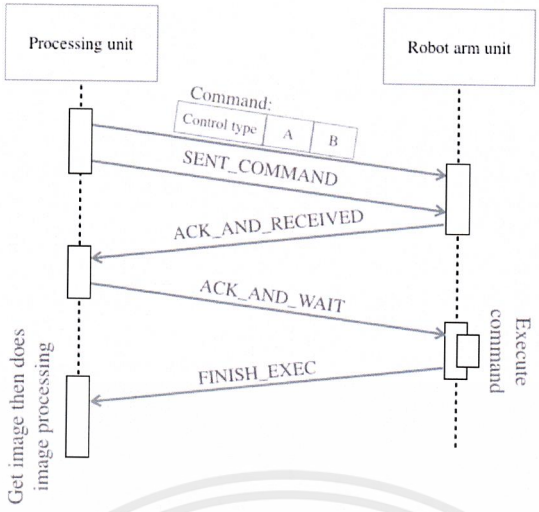


Figure 3.7: The interaction between units

For the robotic arm to reach a specific plant, the processing unit must send a command with control type equal to 3 for the four-wheel platform to move along the *z-axis* and reach the plant. Before the platform is moving, the robotic arm enters sleep state by default, and become ready after it reached the destination. Moreover, to take the plant photos at a desired position, the processing unit has to solve kinematic problems. Propitiously, the problem can be simplified into two-dimension problems with three degree-of-freedoms since the platform is not going to move and the first joint is not going to rotated at the moment; therefore, the problems can be solved analytically for both forward and inverse cases.

For the forward kinematic case, the question is to find the end effector position (x, y) while α, β, d_1 and d_2 are given. The problem is solved using the following homogeneous transformation matrices and trigonometry.

$$\mathbf{R}_{0,1} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \quad \mathbf{R}_{1,2} = \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix} \tag{3.1}$$

$$\mathbf{T}_{0,1} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \mathbf{T}_{1,2} = \begin{bmatrix} d_1 \\ 0 \end{bmatrix}$$

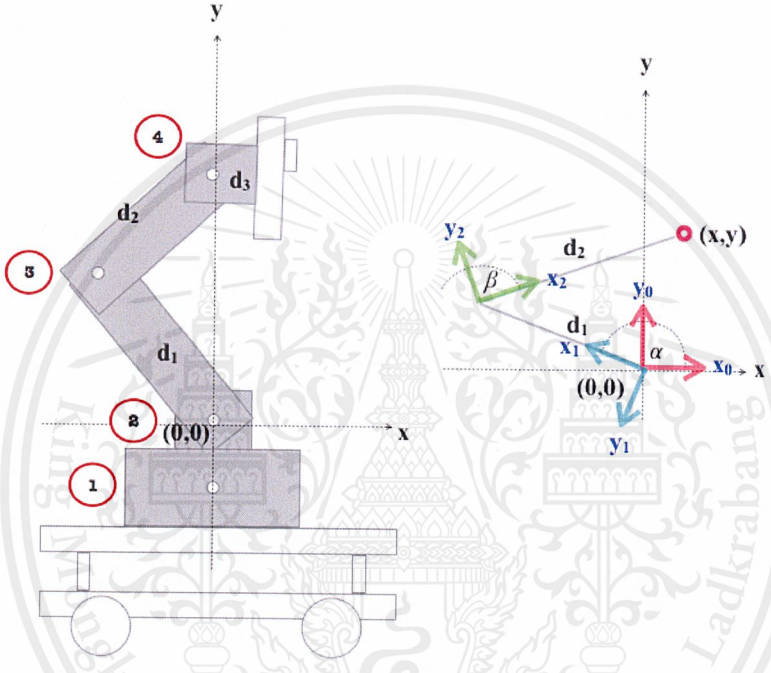


Figure 3.8: An example for the forward kinematic case

The forward kinematic problem is solved using trigonometric equations as follows.

Algorithm 1 Finding the end effector position using forward kinematic

Compute the end effector position in frame 2,

$$\mathbf{p}_2 = \begin{bmatrix} d_2 \\ 0 \end{bmatrix}$$

Compute the end effector postiton in frame 1,

$$\mathbf{p}_1 = \mathbf{R}_{1,2}\mathbf{p}_2 + \mathbf{T}_{1,2}$$

Compute the end effector position in frame 0,

$$\begin{aligned}
\mathbf{p}_0 &= \mathbf{R}_{0,1}\mathbf{p}_1 + \mathbf{T}_{0,1} \\
&= \mathbf{R}_{0,1}(\mathbf{R}_{1,2}\mathbf{p}_2 + \mathbf{T}_{1,2}) \\
&= (\mathbf{R}_{0,1}\mathbf{R}_{1,2})\mathbf{p}_2 + \mathbf{R}_{0,1}\mathbf{T}_{1,2} \\
&= \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} d_2 \\ 0 \end{bmatrix} + \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} d_1 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \cos\alpha\cos\beta - \sin\alpha\sin\beta & -\cos\alpha\sin\beta - \sin\alpha\cos\beta \\ \sin\alpha\cos\beta + \cos\alpha\sin\beta & -\sin\alpha\sin\beta + \cos\alpha\cos\beta \end{bmatrix} \begin{bmatrix} d_2 \\ 0 \end{bmatrix} + \begin{bmatrix} d_1\cos\alpha \\ d_1\sin\alpha \end{bmatrix} \\
&= \begin{bmatrix} d_1\cos\alpha + d_2\cos(\alpha+\beta) \\ d_1\sin\alpha + d_2\sin(\alpha+\beta) \end{bmatrix}
\end{aligned}$$

Compute the actual end effector position (x, y),

$$\begin{aligned}
x &= d_1\cos\alpha + d_2\cos(\alpha+\beta) \text{ and} \\
y &= d_1\sin\alpha + d_2\sin(\alpha+\beta)
\end{aligned}$$

For the inverse kinematic case, the question is to find α and β while (x, y), d_1 and d_2 are given. The problem is solved using trigonometric equations as follows.

Algorithm 2 Finding the required angles using inverse kinematic

Calculate $h = \sqrt{x^2 + y^2}$

Calculate $\alpha_1 = \tan^{-1}\left(\frac{y}{x}\right)$

Calculate $\alpha_2 = \cos^{-1}\left(\frac{d_2^2 - d_1^2 - h^2}{2d_1h}\right)$

Calculate $\alpha = \alpha_1 + \alpha_2$

Calculate $\beta = 180 - \cos^{-1}\left(\frac{h^2 - d_1^2 - d_2^2}{2d_1d_2}\right)$

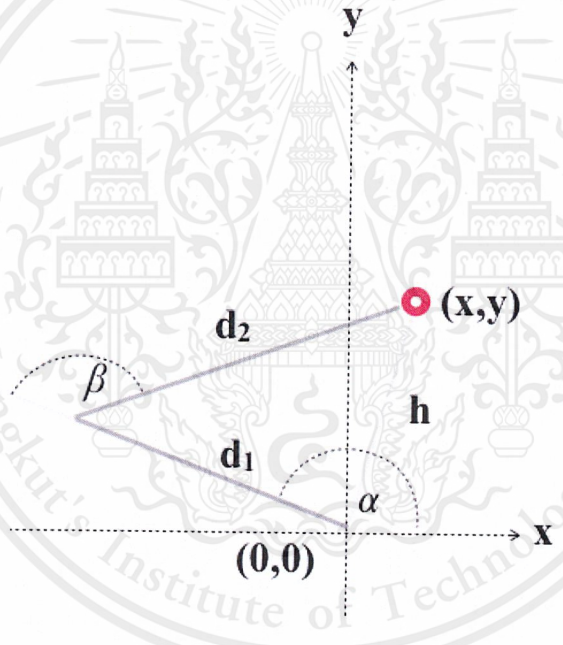


Figure 3.9: An example for the inverse kinematic case

Therefore, when the processing wants the robotic arm unit to move its camera up by a distance, e.g. 10 centimeters, it solves for the target position of the robotic arm's tip by getting the current end-effector position (x,y) of the robotic arm and adding the distance into y component of the position, gets $(x, y+10)$ as a target position, solves

for the essential joints' angles, then commands the robot arm unit to rotate each joint accordingly. Another example, when it wants the robotic arm unit to move the camera toward an object (zoom in), it finds the current object position, which should be located in front of the camera, by getting the distance information reported by the robot arm unit, which is measured using the infrared distance measuring sensor attached at the robotic arm's tip, and solves for the target position as follows.

Algorithm 3 Finding a target position: zoom-in

(x,y) = The current end-effector position
l = The distance reported from the robot arm unit
if $l > 20$ **then**
    There is no object in front of the camera, let d = the distance between the end-effector position and the camera's lens, d_3.
else
    $d = l + d_3$
end if
Calculate a moving distance $d_m = 0.2d$
if $d_m < 1$ **then**
    return (x,y) as the target position
end if
Get the current angle of the robotic arm's camera θ
Calculate $dx = d_m \cos\theta$
Calculate $dy = d_m \sin\theta$
Calculate a target position (x_t,y_t) as $x_t = x + dx$ and $y_t = y + dy$
return (x_t,y_t)

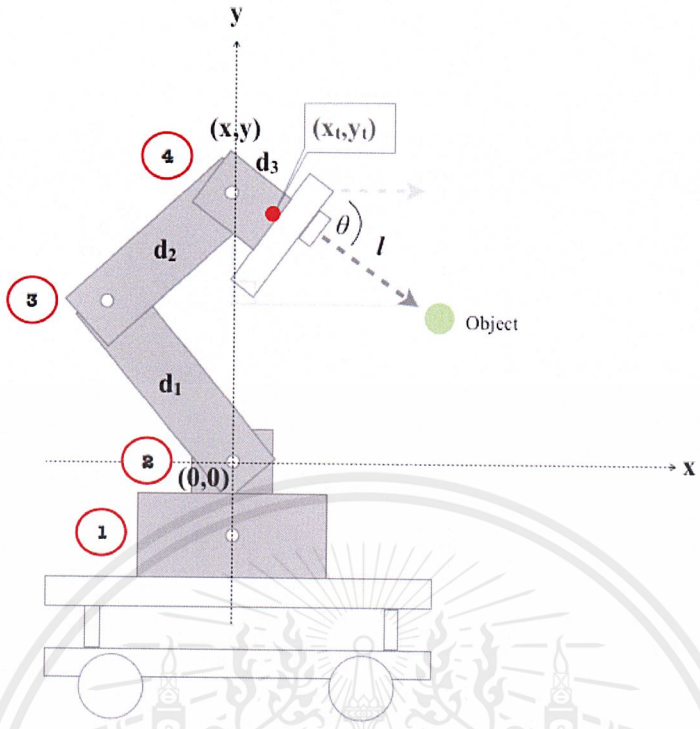


Figure 3.10: An example for finding a target camera position: zoom in case

After getting (x_t, y_t), the processing unit solves for the required joints' angles for the robotic arm, then sends rotating joint commands to the robot arm unit to move the camera toward the object. In contrast, if the processing unit wants the robot arm unit to zoom out its camera, the moving distance d_m is multiplied by -1, indicating the opposite direction, before the target position calculation.

However, after moving the robotic arm's camera to the target position, the camera's lens might not pointing to the correct direction, toward the object. Hence, the camera's angle must be adjusted according to the changed end-effector position and the object's position. It can be calculated as follows.

Algorithm 4 Adjusting the camera's angle

(x_o,y_o) = The object's position obtained before moving to the target position, calculated as $x_o = x + (l + d_3)\cos\theta$ and $y_o = y + (l + d_3)\sin\theta$
(x_t,y_t) = The current end-effector position
(x_2,y_2) = The current robotic arm's joint 2's position
$x_o = |x_o - x_t|$
$y_o = |y_o - y_t|$
$x_2 = |x_2 - x_t|$
$y_2 = |y_2 - y_t|$
$len_1 = \sqrt{x_o^2 + y_o^2}$
$len_2 = \sqrt{x_2^2 + y_2^2}$
$ang\_diff = \arccos(((x_o x_2) + (y_o y_2))/(len_1 len_2))$
Adjust the camera angle to $\gamma = 180 - ang\_diff$

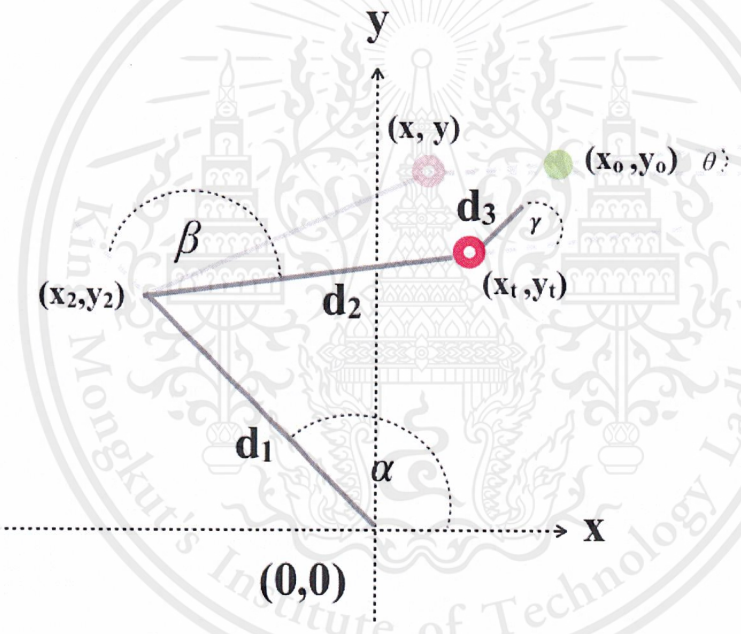


Figure 3.11: An example for adjusting robotic arm camera's angle

3.1.2 The rail

A rail is included in the robotic arm unit to hold the robotic arm discussed in the previous section. With the rail, the robotic arm is kept in a correct track, and moves

along to take photos of different plants. In this project, the rail is constructed using stainless; therefore, it is heavy enough to support the robotic arm. Moreover, as shown in Fig. 3.12, the rail has a bending part at the top to prevent the robot arm from falling. Fig. 3.13 shows the completed version of the rail.

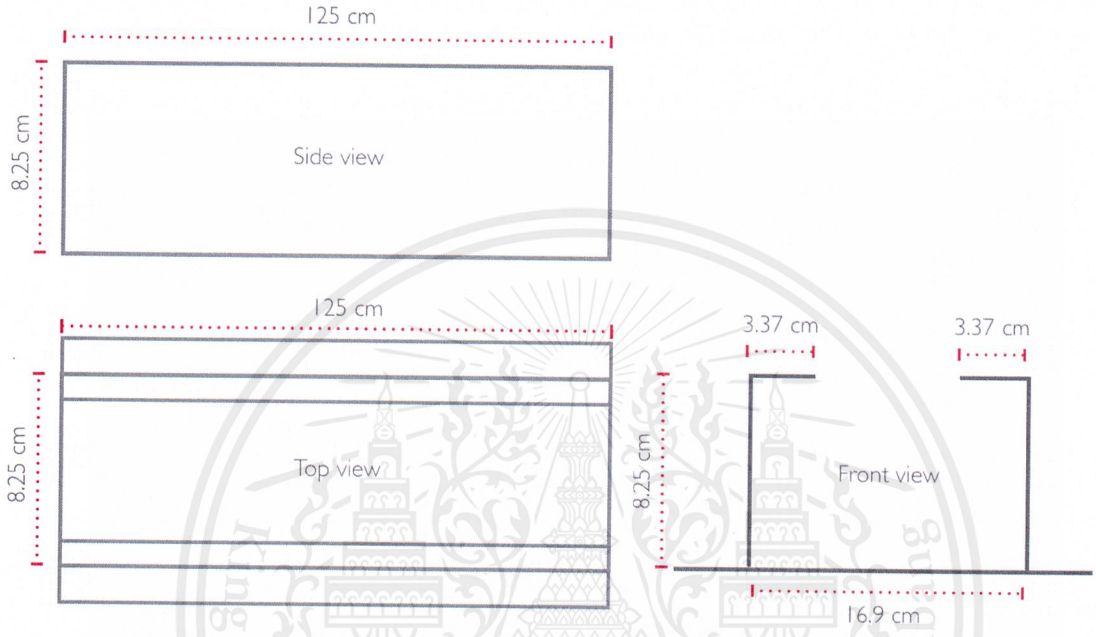


Figure 3.12: The robot rail dimension

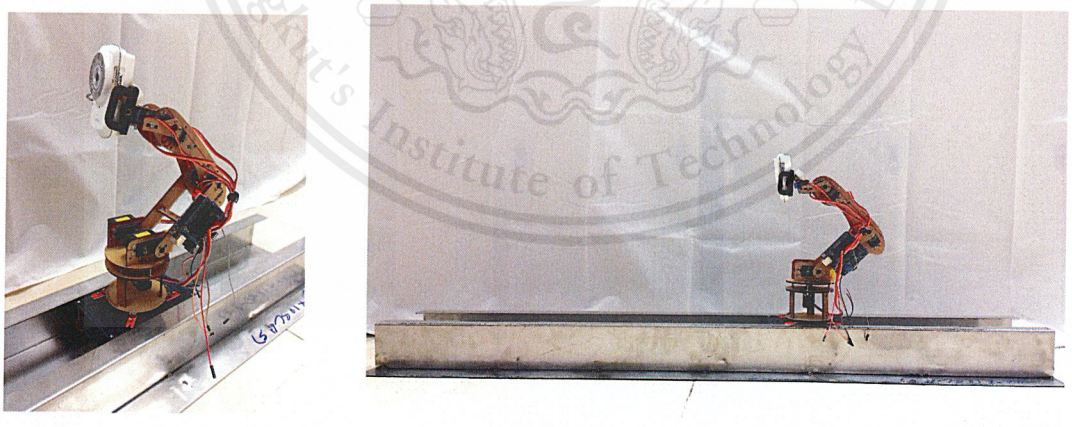


Figure 3.13: A completed robotic arm rail

3.2 Plant rotation unit

The duty of plant rotation unit is to rotate a plant when the robotic arm is taking photo. The unit consists of a set of rotators and a controller. Similar to the robotic arm (Section 3.1.1), this unit has three systems for communication, commanding and decision making, and interaction. Unipolar stepper motors, 28BYJ-48, which are parts of the interaction system, are chosen as the rotators to rotate plants. These motors are parts of the interaction system. One stepper motor is paired with one plant. A plant is placed on the stepper motor as shown in Fig. 3.14. Several pairs of a stepper motor and a plant can be added to a planting bed, while their stepper motors are controlled by the same commanding and decision making system, which is also an Arduino UNO board, as shown in Fig. 3.15. It is stacked with a WiFi shield that allows a communication between itself and the processing unit.

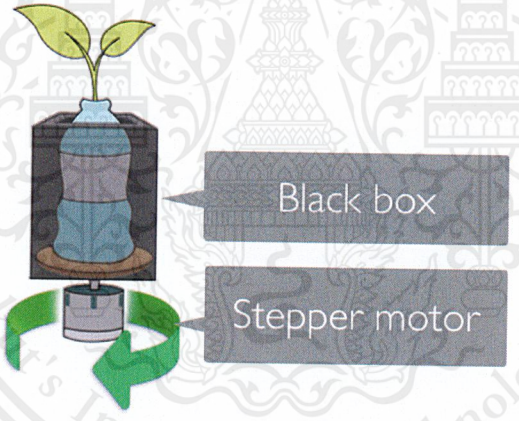


Figure 3.14: A pair of stepper motor and plant.

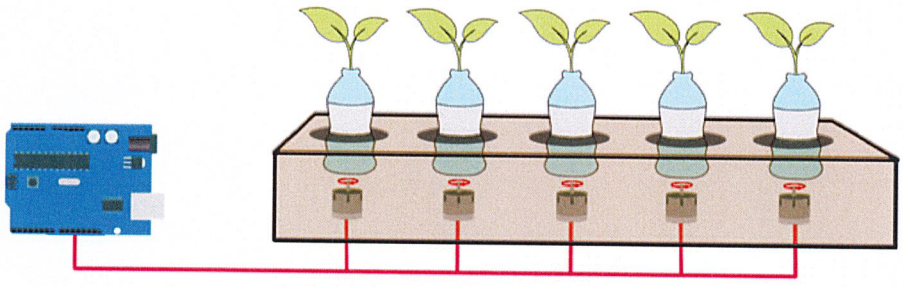


Figure 3.15: Planting bed and a controller.

For the decision making and commanding system, a 3-to-10 line decoder is used to select a stepper to be actuated. One output line corresponds to one stepper. Each output line is connected to a three-state buffer as an enable input signal. The power supply for steppers is connected to the buffers' input pin, and the output of each buffer provides power to the corresponding stepper (Fig. 3.16). The plant rotation unit's components are shown in Table 3.4; also, the connection between the Arduino, one stepper motor, and the decoder is shown in Fig. 3.17. The figure also shows that each stepper motor is driven using a stepper driver chip, ULN2003.

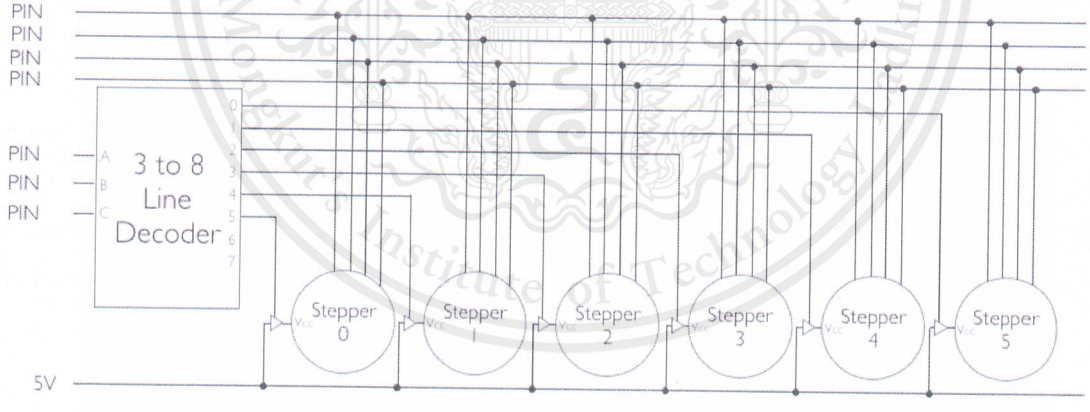


Figure 3.16: Using a 3-to-8 lines decoder to select a stepper motor

Table 3.4: Plant rotation components

Communication system	Adafruit CC3000 WiFi module
Decision making and commanding system	Arduino UNO micro controller
	3-to-8 decoder
Actuators	28BYJ-48 stepper motor
Interacted environment	Plants
	Planting bed

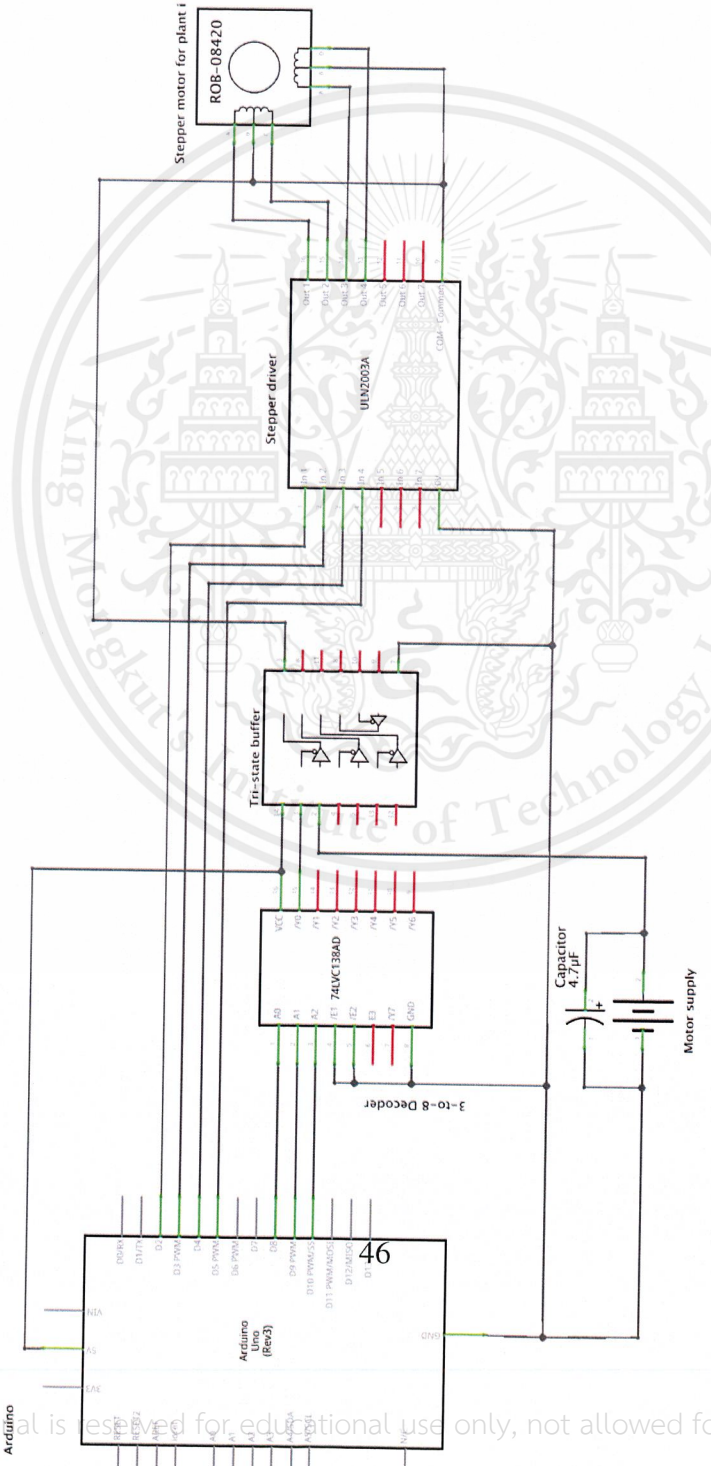


Figure 3.17: Connection of one stepper motor

3.2.1 Planting bed control

3.2.1.1 How does the processing unit control the plant rotation unit?

Match with the robot arm unit, for the processing unit to control the plant rotation unit, a command with the same format as Fig. 3.5 is needed just like how it controls the robotic arm. However, the control type have to be 4 in this case. As for A and B, the first corresponds to an index of the plant in the planting bed, while the latter corresponds to a rotating angle in degree. For instance, if the processing unit wants the plant rotation unit to rotate the third plant in the planting bed by 30 degree, the command sent to the server must be [4 2 30]. In essence, the plant rotation unit control program can be explained using Fig. 3.18

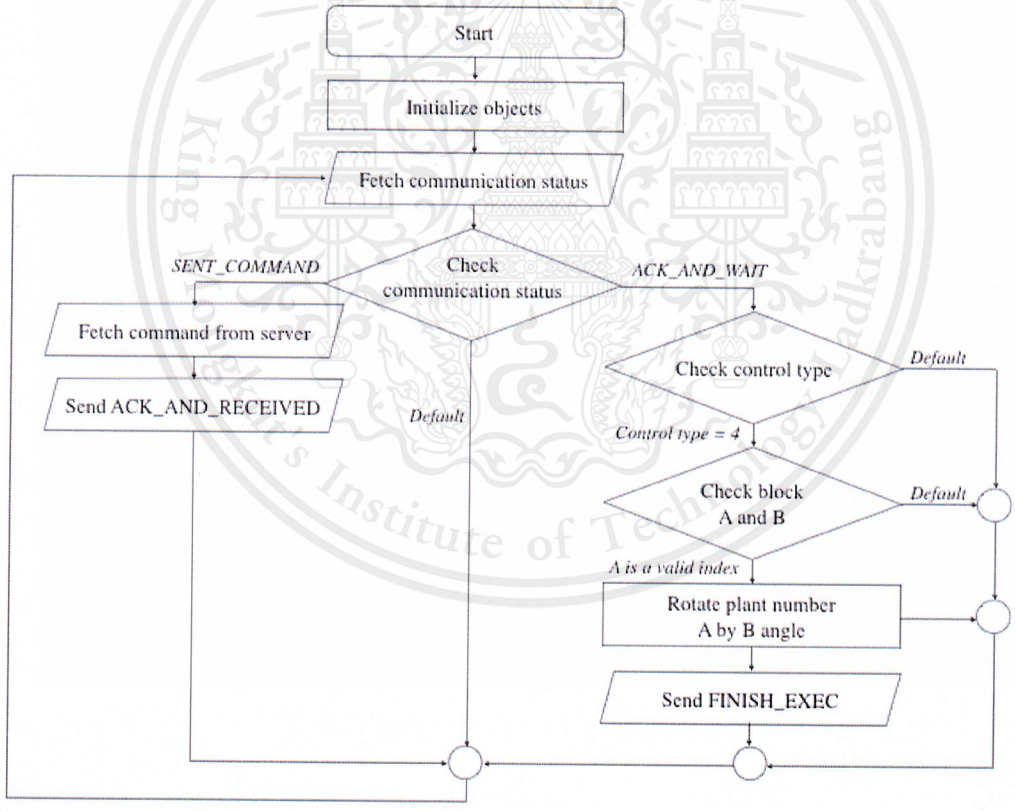


Figure 3.18: The plant rotation unit's program flow

3.3 Processing unit

The processing unit acts as a the system's coordination center. It communicates with the two units discussed in Section 3.1 and Section 3.2. Its main task is to generate the system output which is the segmented leaf images. In order to complete the task, the processing unit applies the following major steps:

Algorithm 5 Major steps in the processing unit

Get an image from robotic arm's camera

Preprocess the image

Find a mid-vein of a leaf in the image

Create a polygon model using the two end points of the mid-vein as apex and base positions and use it to detect a leaf in the image

Segment the leaf using information from the polygon model

The processing unit may command the robot arm unit to move the camera or command the plant rotation unit to rotate the plant to adjust the leaf position in the image. The preprocessing unit can be represented in general as in Fig. 3.19.

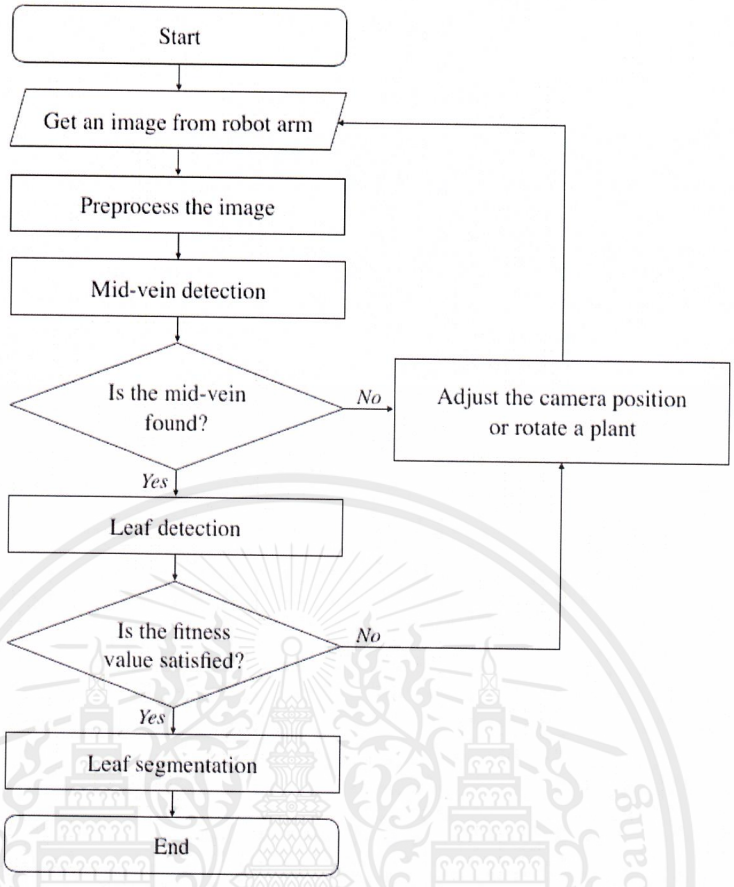


Figure 3.19: Process of the processing unit

3.3.1 Preprocessing

Preprocessing process is applied to convert an input leaf image obtained from the robotic arm's camera to the format required by the latter processes. In this process, a *bilateral filter* (Section 2.2.5) is used to eliminate noises in the image while preserve the edges, which are significant for mid-vein detection. After that, the process keeps objects with green color, which should be leaves, and eliminates objects with other colors using *green hue filtering* as follows,

Algorithm 6 Green hue filtering

Convert the input *image*'s color space from *RGB* to *HSV*

for each pixel in *image* **do**

$[H, S, V]$ = the pixel's intensity level

if $H \notin [30, 70]$ OR $S < 50$ OR $V < 50$ **then**

Set the pixel's color to black

end if

end for

Next, the blurred image is converted into a grayscale image and the edges are enhanced by applying the morphological operations explained in Section 2.2.7 on a canny edge image of the grayscale image, then merging the result with the same grayscale image. This merged image is then returned as a result of the preprocessing process. All in all, the preprocessing process can be described using the following algorithm (see Fig. 3.20).

Algorithm 7 Preprocessing

Perform green hue filtering on *image*

Select the largest contour in *image* as *largest_contour_mask*

Multiply *image* with *largest_contour_mask*

Blur *image* using bilateral filter

Perform green hue filtering on *image*

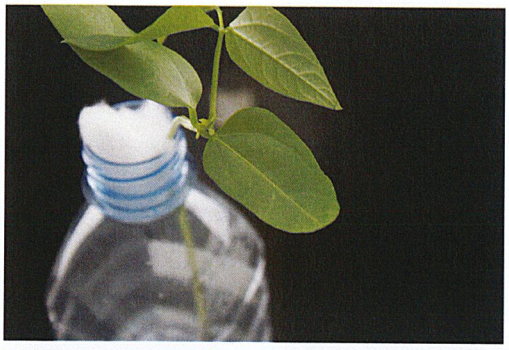
Convert the color of *image* to grayscale

Create *edge_image* of *image* using *canny edge detection*

Enhance *edge_image* using *dilation*, *erosion*, and *thinning*

Merge *image* with *edge_image*

return The merged *image* as output



(a) An input image

(b) Applied green hue filtering

(c) Applied contour extraction and hole filling

(d) Applied bilateral filtering

(e) Applied green hue filtering again

(f) Converted to grayscale image

(g) Performed edge enhancement

Figure 3.20: The steps of the preprocessing

3.3.2 Mid-vein detection

In order to apply a polygon model at a correct place in the leaf image, the position and the orientation of the leaf should be determined beforehand. On this account, the *mid-vein detection process* is introduced. A *mid-vein* or a *primary vein* is the longest vein on a leaf's blade which usually lies on the center of the leaf and connects the leaf's apex and base together, as shown in Fig. 3.21. The mid-vein can be detected easily by human eye; though, it is harder for machines and the mid-vein detection is a challenging task in computer vision. From a mid-vein, approximation of the leaf's position and orientation can be extracted. For instance, a middle point of the mid-vein has high chance to be the same point as the leaf center and the orientation can be calculated as the angle between the mid-vein and a horizontal line.

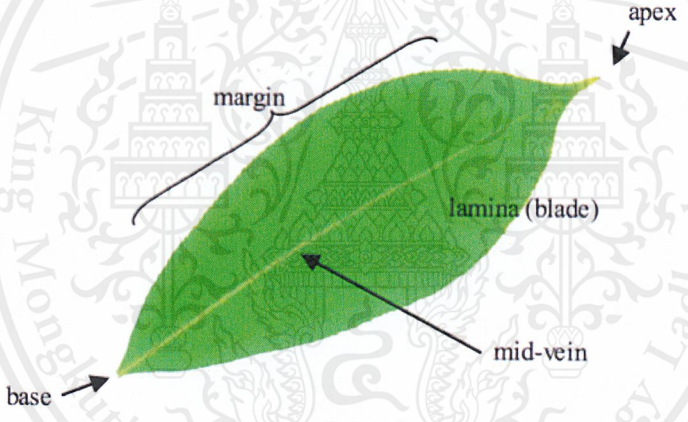


Figure 3.21: Leaf structure

The objective of the mid-vein detection is to automatically detect at least one mid-vein in an input image, which is the result of the preprocessing process. It is required to specify the starting and the ending points of the detected mid-vein as an output. Hereafter, in the next process, a leaf's position and orientation are obtained from the two points and are used to create a proper polygon model for leaf detection. Supplementary, the algorithm of mid-vein detection is described below.

Algorithm 8 Mid-vein detection

image = A preprocessed image obtained from the preprocessing process

Divide *image* into blocks of size $N \times N$

Compute gradient magnitudes g_x and g_y of *image* using *Sobel* operator.

OF_mat = An empty matrix of size $N \times N$

$consecOF\_mat$ = A zero matrix of size $N \times N$

$max\_block = NULL$

$max\_consecOF = 0$

for each block b_{ij} in *image* **do**

Estimate an orientation o_{ij} for b_{ij} using the equation from Section 2.2.6.

Quantize the orientation o_{ij} into 0, 45, 90, or 135 degree.

OF_mat_{ij} = The quantized o_{ij}

for each consecutive block c_{kl} of b_{ij} **do**

if $OF\_mat_{ij} = OF\_mat_{kl}$ AND all pixels in b_{ij} are not edge pixels **then**

$consecOF\_mat_{ij} = consecOF\_mat_{ij} + 1$

if $max\_consecOF < consecOF\_mat_{ij}$ **then**

$max\_consecOF = consecOF\_mat_{ij}$

$max\_block = b_{ij}$

end if

end if

end for

end for

Backtracking to find the starting point s_{ij} of b_{ij} in $consecOF\_mat$

return A pair of s_{ij} and b_{ij}

3.3.3 A model for black gram

A polygon model, which was proposed by Cerutti et al. [5], is a model to find an approximate leaf's shape using polygon. This model considers only simple, non-palmate leaves and assumes that a leaf is symmetry, lying in vertical axis, and is in the center of an image. It requires six input parameters, as follows, to generate the shape model.

1. B: Base position
2. T: Tip position
3. α_B: Angle at the base
4. α_T: Angle at the apex
5. w: Relative maximal width
6. p: Relative position where this width is reach

After the input parameters are obtained, the polygon model will be constructed using the following steps.

Algorithm 9 Polygon model construction

Define B as a base point and T as a tip point of a leaf (Fig. 3.22a).
Suppose that $\mathbf{a}$ is the unit direction vector of the line segment $\overrightarrow{BT}$, $\mathbf{n}$ is a normal vector that perpendicular to $\mathbf{a}$, h is a length from B to T, and $\overrightarrow{BT} = h\mathbf{a}$ (Fig. 3.22b).
C or center point is defined $\overrightarrow{BC} = p\overrightarrow{BT}$ (Fig. 3.22c).
The segment of maximal width $\overrightarrow{C_LC_R}$ is perpendicular to the main axis $\overrightarrow{BT}$ and centered on C (Fig. 3.22d).

$$\begin{aligned} \overrightarrow{CC_L} &= \left(\frac{w}{2}\right) h\mathbf{n} \\ \overrightarrow{CC_R} &= -\left(\frac{w}{2}\right) h\mathbf{n} \end{aligned} \tag{3.2}$$

At the point of C_L and C_R, C_{LT} and C_{LB} can be identified as the point perpendicular C_L. C_{RT} and C_{RB} can be identified as the point perpendicular C_R (Fig. 3.22e).

$$\begin{aligned} \overrightarrow{C_LC_{LT}} &= \frac{h}{5} \times (1 - 2 \times |p - 0.5|) \times \mathbf{a} \\ \overrightarrow{C_LC_{LB}} &= -\frac{h}{5} \times (1 - 2 \times |p - 0.5|) \times \mathbf{a} \\ \overrightarrow{C_RC_{RT}} &= \frac{h}{5} \times (1 - 2 \times |p - 0.5|) \times \mathbf{a} \\ \overrightarrow{C_RC_{RB}} &= -\frac{h}{5} \times (1 - 2 \times |p - 0.5|) \times \mathbf{a} \end{aligned} \tag{3.3}$$

At the tip T, the algorithm creates isosceles triangle and define T_L, T_T, and T_R (Fig. 3.22f).

$$\begin{aligned} \overrightarrow{TT_L} &= \frac{h}{5} \times \sin\left(\frac{\alpha_T}{2}\right) \times \mathbf{n} \\ \overrightarrow{TT_R} &= -\frac{h}{5} \times \sin\left(\frac{\alpha_T}{2}\right) \times \mathbf{n} \\ \overrightarrow{TT_T} &= \frac{h}{5} \times \cos\left(\frac{\alpha_T}{2}\right) \times \mathbf{n} \end{aligned} \tag{3.4}$$

At the tip B, the algorithm creates isosceles triangle and define B_L, B_B, and B_R (Fig. 3.22g).

$$\begin{aligned} \overrightarrow{BB_L} &= \frac{h}{5} \times \sin\left(\frac{\alpha_B}{2}\right) \times \mathbf{n} \\ \overrightarrow{BB_R} &= -\frac{h}{5} \times \sin\left(\frac{\alpha_B}{2}\right) \times \mathbf{n} \\ \overrightarrow{BB_B} &= -\frac{h}{5} \times \cos\left(\frac{\alpha_B}{2}\right) \times \mathbf{a} \end{aligned} \tag{3.5}$$

Finally, the polygon model is obtained by 10 points as an output which are B_B, B_L, C_{LB}, C_{LT}, T_L, T_T, T_R, C_{RT}, C_{RB} and B_R, as shown in Fig. 3.22h.

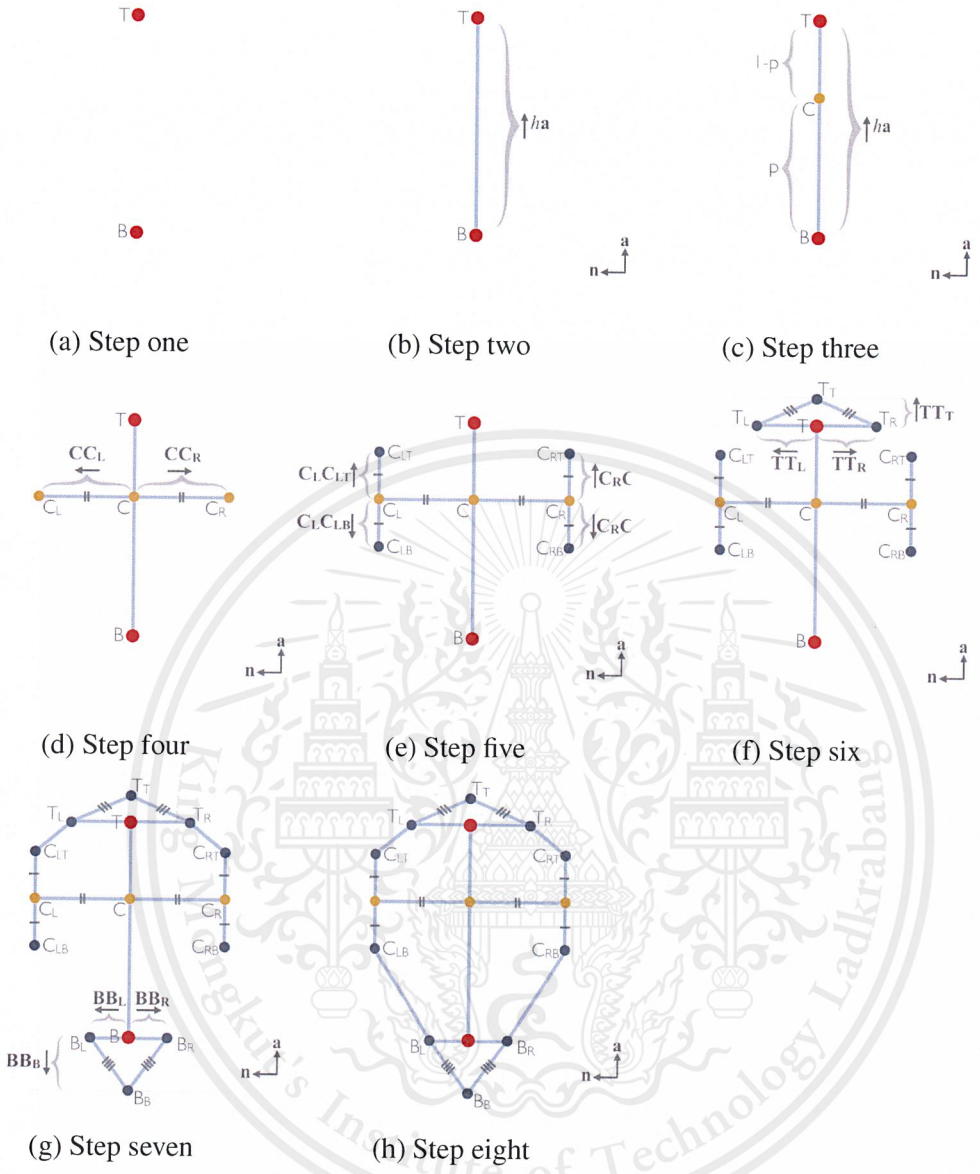
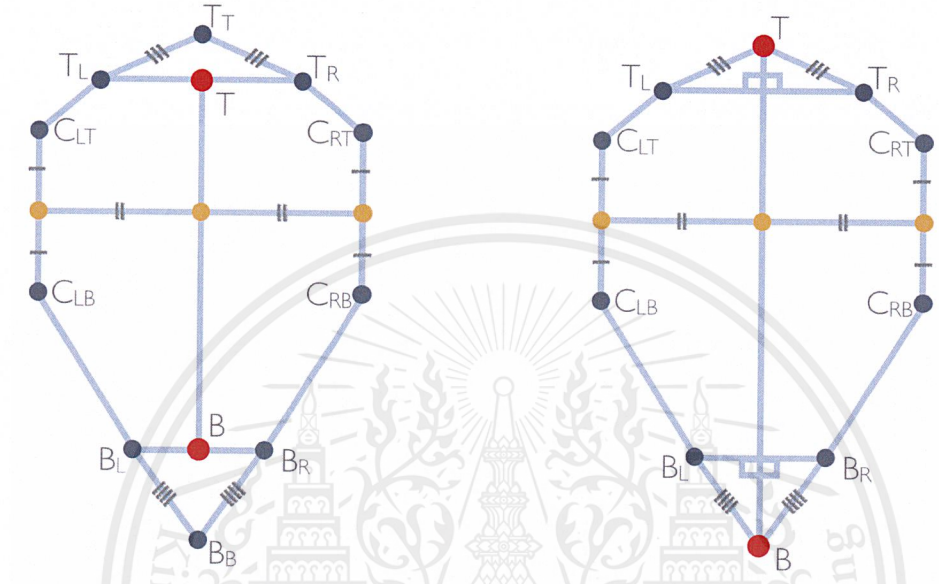


Figure 3.22: The steps of the polygon model

It is important to note that in their work, the tip point T is not the real apex point and base point B is not the real base point of a leaf, as shown in Fig. 3.23a. Since the positions of T and B are difficult to marked, the algorithm is modified by changing them to the real ones. After the algorithm is modified, the tip point T is changed to T_T and

the base point B is changed to B_B which correspond to the actual position of the leaf's apex and base, respectively, as shown in Fig. 3.23b. Moreover, because of the changing positions of the two points, h and two isosceles triangles were calculated using new equations.



(a) Model proposed by G. Cerutti (b) A modified model

Figure 3.23: A polygon model algorithm

$$h' = \frac{h}{1 + \frac{1}{5}\cos\left(\frac{\alpha_B}{2}\right) + \frac{1}{5}\cos\left(\frac{\alpha_T}{2}\right)} \tag{3.6}$$

$$\begin{aligned} \overrightarrow{TT_T}' &= -\frac{h'}{5} \times \cos\left(\frac{\alpha_T}{2}\right) \times \mathbf{a} \\ \overrightarrow{TT_L}' &= \overrightarrow{TT_T}' \times \tan\left(\frac{\alpha_T}{2}\right) \times \mathbf{n} \\ \overrightarrow{TT_R}' &= -\overrightarrow{TT_T}' \times \tan\left(\frac{\alpha_T}{2}\right) \times \mathbf{n} \end{aligned} \tag{3.7}$$

$$\begin{aligned}
\overrightarrow{BB_B'} &= \frac{h'}{5} \times \cos\left(\frac{\alpha_B}{2}\right) \times \mathbf{a} \\
\overrightarrow{BB_L'} &= \overrightarrow{BB_B'} \times \tan\left(\frac{\alpha_B}{2}\right) \times \mathbf{n} \\
\overrightarrow{BB_R'} &= -\overrightarrow{BB_B'} \times \tan\left(\frac{\alpha_B}{2}\right) \times \mathbf{n},
\end{aligned} \tag{3.8}$$

where h' is a new length from B and T, $\overrightarrow{TT_T'}$, $\overrightarrow{TT_L'}$ and $\overrightarrow{TT_R'}$ are new isosceles triangle formula at T, and $\overrightarrow{BB_B'}$, $\overrightarrow{BB_L'}$ and $\overrightarrow{BB_R'}$ are new isosceles triangle formula at B.

While not all of the leaves in this world are symmetry, Cerutti et al. assumes that they are. In contrast to their work, an asymmetric polygon model is more favorable for the proposed system. The asymmetric polygon model can be separated into two sides which are the left and the right. Accordingly, the relative maximal width w, is separated into relative maximal width left w_L, and relative maximal with right w_R, and the relative position p, is separated into relative position left p_L, and relative position right p_R. Likewise, α_B is separated into α_{BL} and α_{BR}, and αT is separated into α_{TL} and α_{TR}, as shown in Fig. 3.24. Owning to these changes, the new model is more fit to the leaves than the old one.

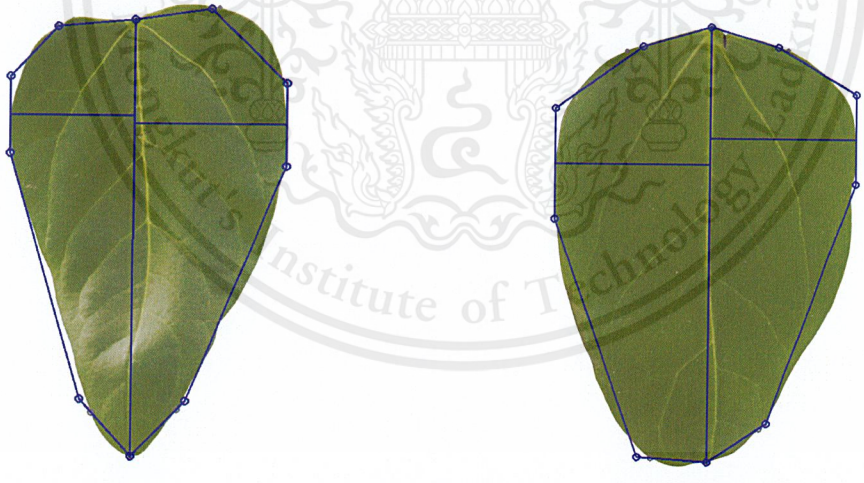


Figure 3.24: Asymmetry polygon model of the black gram leaves

In this project, the algorithm is modified to generate a general model of black gram

leaves. By conducting an experiment Section 4.2, all necessary parameters were fixed beforehand; therefore, by providing apex and base positions, a model can be create immediately.

3.3.4 Leaf detection

Given a mid-vein of a leaf in an image, the next step is to create and apply a polygon model to the image. After the model is applied, it will be evaluated to determine whether the model is fit with the leaf area or not. If the model is fit, it will be sent to the leaf segmentation process; otherwise, it will be adjusted before evaluated again, as shown in Fig. 3.25.

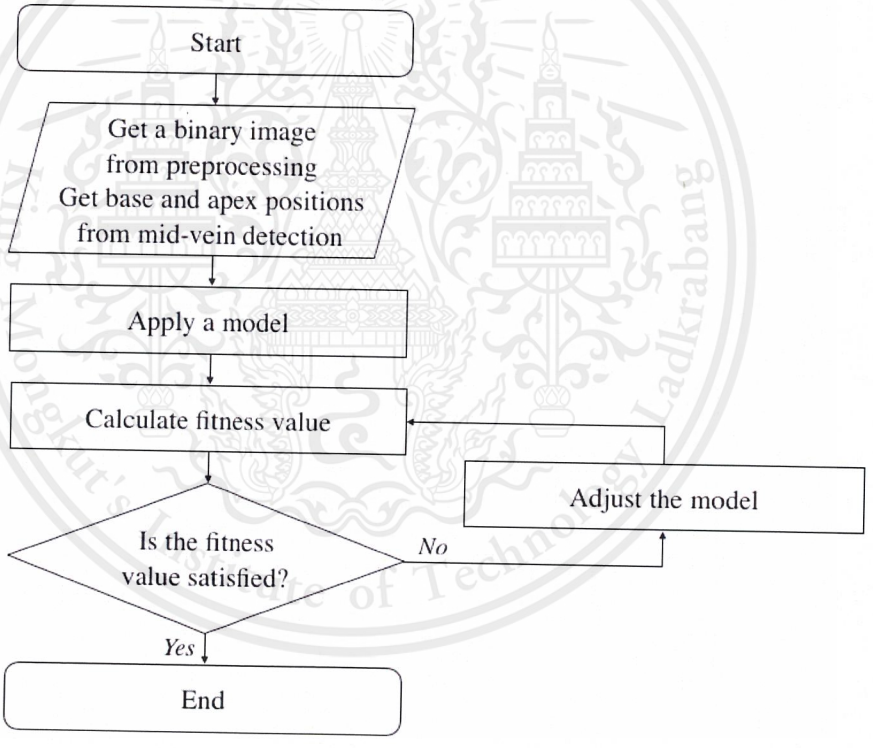


Figure 3.25: Basic leaf detection algorithm

3.3.4.1 How is the model created and applied?

From a mid-vein generated by mid-vein detection process, the mid-vein's end points are assumed to be apex and base positions and are used to create a polygon model. After the model is created, it is placed in the image with its center be the point calculated as follows (Fig. 3.26).

let (x_a, y_a) be the apex position extracted from the mid-vein let (x_b, y_b) be the base position extracted from the mid-vein. The model center $(x_m, y_m) = \left(\frac{x_a + x_b}{2}, \frac{y_a + y_b}{2}\right)$.

Figure 3.26: How is the model applied

3.3.4.2 How is the model evaluated?

The model is evaluated to choose the next action to be performed. The evaluation finds a numerical value of the differences between the model and an actual leaf area in the given image. The differences are represented by error and is calculated as the summation of the Euclidean distances between the model and the leaf contour. Additionally, less error is more preferable.

In the system, an error threshold is defined in advanced and is used as one of the termination criteria. Basically, if the error is less than the error threshold, then the model is supposed to be fit with the leaf. In this case, the leaf detection process will be terminated and the model will be sent to the next process which is the leaf segmentation.

On the other hand, if the error is higher than or equal to the error threshold, the model will be adjusted in the adjustment process and is evaluated again after that.

Algorithm 10 Error calculation

$M = \{\}$
for each point p_i = 1 to 9 **do**
    Find a straight line that connects p_i to p_{i+1}
    Find a point at the middle of the line
    Add the middle point into M
end for
Find a straight line that connects p_{10} to p_1
Find a point at the middle of the line
Add the middle point into M
$sum\_error = 0$
for each point p_i in M **do**
    $error$ = Euclidean distance between p_i and the nearest point that the color changes
    $sum\_error = sum\_error + error$
end for
$sum\_error = sum\_error \div$ polygon model's perimeter

3.3.4.3 How is the model adjusted?

Polygon model adjustment process is done to reduce the error. In this process, there are two actions including resizing and translation. Differently, resizing is to expand or shrink the model while translation is to move the model by some distance toward one direction. Both of them have their own requirements and their own ways to achieve them.

The requirements of resizing action are 1) the model should be expanded if it is smaller than the leaf area and 2) the model should be shrunk if it is larger than the leaf area, as shown in Algorithm 11. Resizing percentage rp is used to control how much the size of the model change. rp value must be initialized in advanced. In the system, 20% rp is set as a starting value in the beginning of the leaf detection process and the percentage will be divided by two when the resizing option changes from expanding to

shrinking or from shrinking to expanding, over iterations. After the model is resized, the distances between the model's points and the model's centers changed by $rp\%$. The points will move away from the center in the expanding case; whereas, they move toward the center when shrinking.

Algorithm 11 Adjust the model with resizing

$M = \{\}$
for each point p_i = 1 to 9 of a polygon model **do**
    Find a straight line that connects p_i to p_{i+1}
    Find a point at the middle of the line
    Add the middle point into M
end for
Find a straight line that connects p_{10} to p_1
Find a point at the middle of the line
Add the middle point into M
$resizing\_indicator = 0$
for each point p_i in M **do**
    $resizing\_value$ = Euclidean distance between p_i and the nearest point that the color changes
    if the nearest point is located inside the model **then**
        $resizing\_value = -1 \times resizing\_value$
    end if
    $resizing\_indicator = resizing\_indicator + resizing\_value$
end for
if $resizing\_indicator < 0$ **then**
    Shrink the polygon model by $rp\%$
else if $resizing\_indicator > 0$ **then**
    Expand the polygon model by $rp\%$
end if

The requirements of translation action are 1) the model should move toward the correct position and 2) the translation should not overshoot the actual position of the leaf area. The calculation of a translation value involves vector summation. Each vector included in the summation starts from a model's middle point and end at the nearest point that the color changes. A result of the summation, the $final\_vector$, is used to determine the translation distance and direction. The distance is cast as the $final\_vector$'s

size divided by the number of vectors included in the summation. Accompanying, the translation direction is an angle between the *final_vector* and a horizontal line. Thereafter, the model's center moves from its current position toward the translation direction by the translation distance, apex and base positions move along with the center, and a new model is created using the two points and is placed at the translated center's position.

Algorithm 12 Adjust the model with translation

$M = \{\}$
for each point $p_i = 1$ to 9 **do**
  Find a straight line that connects p_i to p_{i+1}
  Find a point at the middle of the line
  Add the middle point into M
end for
Find a straight line that connects p_{10} to p_1
Find a point at the middle of the line
Add the middle point into M
final_vector = Vector(0,0)
for each point p_i in M **do**
  translation_vector = A vector from p_i to the nearest point that the color changes
  final_vector = *final_vector* + *translation_vector*
end for
(x_1, y_1) = A point at the beginning of the *final_vector*
(x_2, y_2) = A point at the end of the *final_vector*

$$\theta = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$$

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$(x_c, y_c) = (x_c + d\cos\theta, y_c + d\sin\theta)$
$(x_a, y_a) = (x_a + d\cos\theta, y_a + d\sin\theta)$
$(x_b, y_b) = (x_b + d\cos\theta, y_b + d\sin\theta)$
Create a new polygon model using (x_a, y_a) and (x_b, y_b)
Place the model at the (x_c, y_c) position

Figure 3.27 shows the model in the current iteration, adjusted by resizing and translation, and updated in the next iteration.

Figure 3.27: Resizing and translation of the model

3.3.5 Leaf segmentation

Leaf segmentation is the final process of the processing unit. It eliminates unnecessary information, e.g., background, and extracts a leaf out from an image. The process inputs are 1) a color image, 2) a grayscale image from preprocessing process and 3) a polygon model from the leaf detection. After receiving the inputs, it uses a segmentation algorithm, *active contour*, to segment the leaf, and sends the segmented leaf as the system's output.

3.3.5.1 Active contour algorithm

Active contour algorithm, also called *snake* (see [15]), is a segmentation algorithm using active contour, which uses an approximate leaf contour to guide in the initial step.

For more information, a leaf contour is a leaf edge in which the beginning and the end of the edge are the same point. The goal of this algorithm is to find the exact contour of a leaf in an image.

Snake is an optimization algorithm. It does energy minimizing. It tries to deform the approximated contour and makes the contour fit the actual contour, little by little, over iterations. In the algorithm, there are two kinds of energy: internal energy and external energy. The internal energy $E_{internal}$ affects the contour's shape. This kind of energy controls the curvature, and maintains the shape of the contour. On the contrary, the external energy $E_{external}$ tries to move the contour closer to the real one as much as possible. Specifically, the energy snake tries to minimize is reckoning from the following equation.

$$E_{snake}(\mathbf{v}(s))ds = \int_0^1 (E_{internal}(\mathbf{v}(s)) - E_{external}(\mathbf{v}(s)))\,ds, \tag{3.9}$$

where E_{snake} is the total energy of the contour. $\mathbf{v}(s) = (x(s), y(s))$. $\mathbf{v}$ is a point in the contour. s stands for an arc length parameter, varies in the range of 0 to 1. Integral notation implies that the energy is to be calculated for an open-end snake where the first and the last elements are joined into a closed loop.

The internal energy consists of continuity and curvature energy. The continuity energy $E_{continuity}$ attempts to connect the points in the contour together by spreading the points equally. The curvature $E_{curvature}$ maintains the contour smoothness by withstanding the bending force made on the contour.

$$\begin{aligned} E_{internal} &= E_{continuity} + E_{curvature} \\ &= \frac{1}{2}\left(\left(\alpha|\mathbf{v}_s(s)|^2\right) + \left(\beta(s)|\mathbf{v}_{ss}(s)|^2\right)\right) \end{aligned} \tag{3.10}$$

$$
\begin{aligned}
|\mathbf{v}_s|^2 &= \left|\frac{d\mathbf{v}_i}{ds}\right|^2 \\
&= |\mathbf{v}_i - \mathbf{v}_{i-1}|^2 \\
&= (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2
\end{aligned}
\tag{3.11}
$$

$$
\begin{aligned}
|\mathbf{v}_{ss}|^2 &= \left|\frac{d^2\mathbf{v}_i}{ds^2}\right|^2 \\
&= |\mathbf{v}_{i-1} - 2\mathbf{v}_i + \mathbf{v}_{i+1}|^2 \\
&= (x_{i-1} - 2x_i + x_{i+1})^2 + (y_{i-1} - 2y_i + y_{i+1})^2,
\end{aligned}
\tag{3.12}
$$

where α and $\beta(s)$ are weights defined in advance. i is an iteration. d is the distance between points.

However, the value of $\beta(s)$ is adaptable. When applying the algorithm, if the curvature of a point is larger than the adjacent points' curvature and a threshold, it will be set to 0.0 in order to protect the contour's shape. The threshold value is the max curvature calculated in the previous iteration.

The external energy or *image energy*, attracts the contour toward the actual contour. The higher difference between the adjusting contour and the real one, the higher the energy. Hence, when snake gets closer to its goal, the value becomes small.

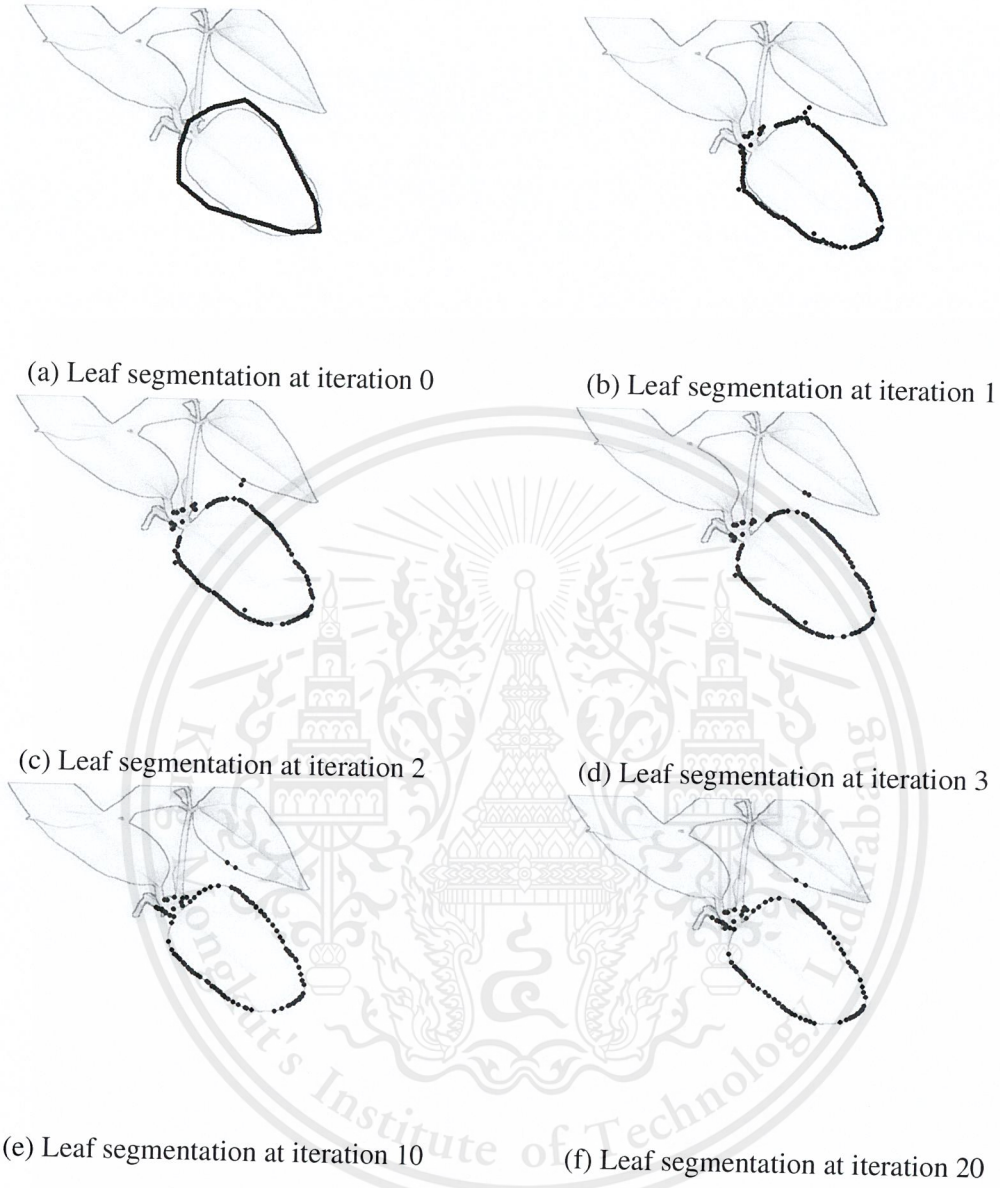
$$
\begin{aligned}
E_{external} &= E_{image} \\
&= \gamma|\nabla I(v(s))|,
\end{aligned}
\tag{3.13}
$$

where γ is the weight of *image energy*. ∇I is an intensity of the gradient image at point $\mathbf{v}$ on the contour.

Algorithm 13 Snake algorithm

MAX_ITERATION = 20
MIN_MOVED_POINTS = 5
current_iteration = 0
count_moved_points = 0
repeat
  for each point $\mathbf{v}_i$ in a contour **do**
    for each point in block size 15×15 which $\mathbf{v}_i$ is the center point **do**
      Find $E_{continuity}$
      Find $E_{curvature}$
      Find E_{image}
    end for
    Normalize $E_{continuity}$
    Normalize $E_{curvature}$
    Normalize E_{image}
    min_energy = INT_MAX
    new_min_energy_point = $\mathbf{v}_i$
    for each point p in block size 15×15 which $\mathbf{v}_i$ is the center point **do**
      Find $E_{snake}(p)$
      if $E_{snake}(p) <$ min_energy OR ($E_{snake}(p) =$ min_energy AND p is the center point) **then**
        new_min_energy_point = p
      end if
    end for
    if $\mathbf{v}_i \neq$ new_min_energy_point **then**
      $\mathbf{v}_i$ = new_min_energy_point
      count_moved_points = count_moved_points + 1
    end if
  end for
  Update $\beta(s)$'s value
  current_iteration = current_iteration + 1
until current_iteration $\leq$ MAX_ITERATION AND count_moved_points $\geq$ MIN_MOVED_POINTS

The figure below shows the leaf segmentation's result at different iterations.



(a) Leaf segmentation at iteration 0

(b) Leaf segmentation at iteration 1

(c) Leaf segmentation at iteration 2

(d) Leaf segmentation at iteration 3

(e) Leaf segmentation at iteration 10

(f) Leaf segmentation at iteration 20

Figure 3.28: Segmentation results of the snake algorithm

Chapter 4

Experiments

Several experiments have been conducted in order to select appropriate values of the factors used in the system and to ensure that the algorithms proposed in the previous chapter are useful. In this chapter, five experiments are explained. The first experiment was conducted to measured the accuracy of the mid-vein detection algorithm. The second experiment is an experiment done to get the necessary parameters of the polygon model. The third experiment was conducted to find an appropriate error threshold for leaf detection. The forth experiment was conducted to measure the correctness of the leaf detection algorithm. Next, the fifth experiment was conducted to observe the performance of the snake algorithm. Finally, the last experiment was done to verify the performance of the entire system after the integration.

4.1 Experiment 1: evaluation of the proposed mid-vein detection

4.1.1 Objective

This section explains an experiment that was conducted to measure the correctness of the mid-vein detection algorithm (see Section 3.3.2).

4.1.2 Experiment setup

The experiment used two kinds of dataset as following

Dataset 1: 101 leaf images were taken using DSLR camera, and each leaf was set in the center of the image, lying in vertical axis. Then, the images were manually segmented the leaf and used black color as a background using Adobe Photoshop CS6. The resolution of the images is 720×480.

Dataset 2: 146 leaves images were taken using DSLR camera, with mostly black background. The resolution of the image is 720×480.

(a) (b)

Figure 4.1: An example input leaf images: (a) a manually segmented leaf image. (b) a leaf image from DSLR camera without segmentation

Mid-vein detection algorithm received an image as an input, and produced as an output a pair of points representing the both ends of the detected mid-vein. To measure the accuracy of the detected mid-vein, it was compared with a ground truth, which was manually marked by human. The angle between the detected mid-vein and the ground truth was used as the error of detection, as shown in Fig. 4.2.

$$\theta = |\phi_1 - \phi_2| \tag{4.1}$$

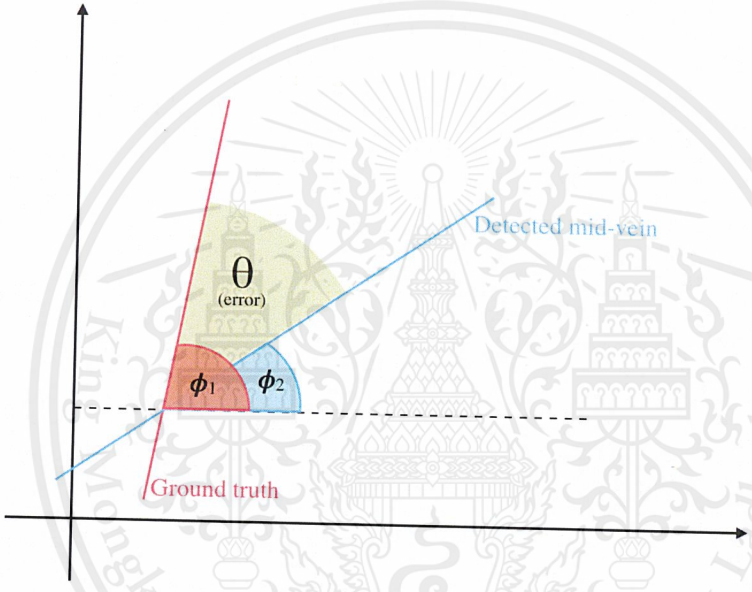
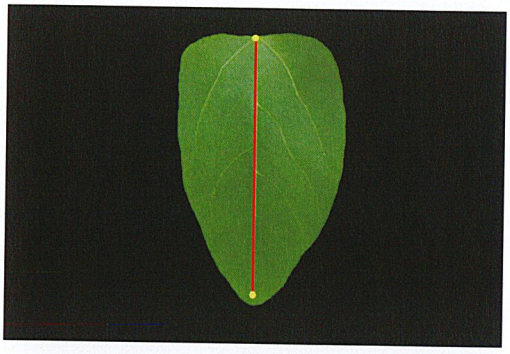


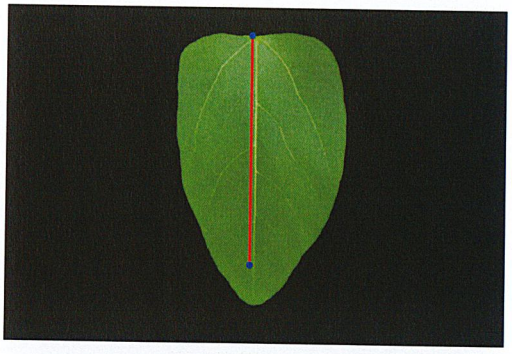
Figure 4.2: Error calculation of mid-vein detection experiment

4.1.3 Result and discussion

Fig. 4.3 shows the result of the experiment. The leaf side is the images with manually marked a pair of points (yellow color). The right one (blue color) is the result images from mid-vein detection algorithm. The error between Fig. 4.3a and Fig. 4.3b is around 0.16° while the error between Fig. 4.3c and Fig. 4.3d is around 2.72°. If the error is less than or equal to 20°, then the image was classified as the correct one.



(a) Ground truth mid-vein



(b) Detected mid-vein

(c) Ground truth mid-vein

(d) Detected mid-vein

Figure 4.3: An examples result of mid-vein detection algorithm. (a) and (b) are leaf image in dataset 1, and (c) and (d) are leaf image in dataset 2.

From Table 4.1, an accuracy (correct detected mid-vein) of the detection algorithm is around 95% and 60% for dataset 1 and dataset 2, respectively. As unnecessary leaves or objects were removed from the images, an accuracy of the detection algorithm was increase.

Table 4.1: Percentages of the correctly detected mid-vein for the two datasets.

	Dataset 1	Dataset 2
Accuracy (%)	95.04	63.01

Unfortunately, there are some cases that the algorithm cannot detect the correct mid-vein. The issue may be caused by:

1. The mid-vein is too blurred to be detected as an image was blurred before applying

the edge enhancement process,

2. A leaf's edge, apart from the leaf's margin, may be detected as mid-vein when the leaf is twisting. Since when computing an orientation field, the edge may have a larger consecOF value than the consecOF of the mid-vein,

3. The mid-vein's intensity is quite similar to the leaf intensity therefore it cannot be detected.

4.2 Experiment 2: finding the parameters for polygon model

4.2.1 Objective

In order to find a proper asymmetric polygon model, this experiment was conducted to obtain the parameters' values which are needed to build the polygon of black gram leaf. In fact, those parameters have to be calculated using the tip angle and the base angle. Hence, the appropriate angle has to be chosen in advance.

4.2.2 Experiment setup

The experiment was done to come by the different angles and select the angle that gives the best result. Also, since the proposed system focuses on black gram, 101 black gram images were used in this experiment. From a black gram leaf image, an angle at T is found by, 1) extracting a contour with the longest distance from the leaf image, 2) separating the contour into two segments. Start from T, the first segment moves along the contour toward one side of the leaf while the second segment moves toward another side, 3) selecting two points from the two segments at two different distances on the contour. The distances were measured as the percentages of the number of pixels on

contour starting from T to B for both sides, and then, 4) measuring the inward angle by combining two angles which are the angle between a vector from T to one point and a vector from T to B, and the angle between a vector from T to the another point and the vector from T to B. Five angles were measured for each of T and B at 5%, 10%, 15%, 20%, and 25% of the number of pixels on each size of the contour (see Table 4.2 and Table 4.3).

After the angles information were collected, four statistically information which are minimum, maximum, average, and standard deviation for each angle were calculated. Lastly, by analyzing the information, the appropriate angles of T and B were chosen and the T position, B position, relative maximum widths w, and relative positions p were obtained.

Finally, the same set of black gram images, which was used to find the T and B angle, was used again to get ws and ps of the polygon model. Recall that the asymmetric polygon model needs eight input parameters: tip point T, base point B, tip angle T α_T, base angle B α_B, relative maximum width for the left side w_L, relative maximum width for the right side w_R and relative position for the left side and the right side p_L and p_R, respectively.

4.2.3 Result and discussion

From Table 4.2, when the distance increases, the standard deviation decreases. Also, with the smaller varieties of the leaf angles, a more general model can be obtained. All things considered, the angle that has the smallest standard deviation should be used. Unfortunately, when the percentage goes higher than 15%, the upper-triangle's and lower-triangle's points of the model are placed inside the leaf area (see Table 4.2), so the angles at 15% distance were chosen, and used to find the model's parameters. For the other parameters, four statistical information were calculated. The information is shown in Table 4.4. These information are used to adjust the shape of the polygon

model and to design the fixed values for model creation in the leaf detection process (see Section 3.3.4).

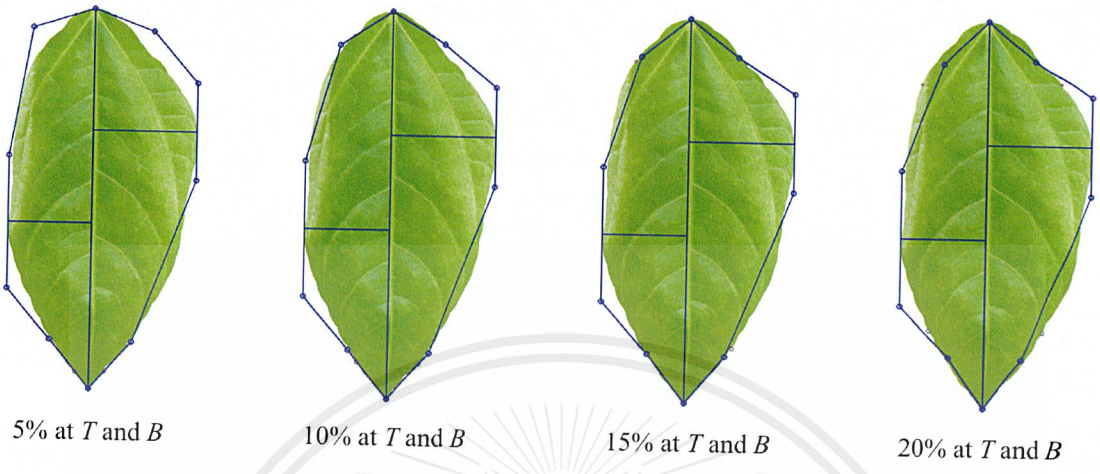


Figure 4.4: The result from calculating angle at T and B

Table 4.2: Statistical values of tip angle (degree) of black gram leaves

	5%		10%		15%		20%	
	α_{TL}	α_{TR}	α_{TL}	α_{TR}	α_{TL}	α_{TR}	α_{TL}	α_{TR}
Min.	5.79	5.36	6.95	4.16	14.17	9.46	19.81	13.44
Max.	99.46	96.52	85.96	80.70	76.69	73.50	71.43	63.50
AVG.	50.43	48.54	45.63	43.79	42.70	41.22	40.30	38.92
SD.	20.18	20.63	15.45	15.13	12.23	12.06	10.06	9.87

Table 4.3: Statistical values of base angle (degree) of black gram leaves

	5%		10%		15%		20%	
	α_{BL}	α_{BR}	α_{BL}	α_{BR}	α_{BL}	α_{BR}	α_{BL}	α_{BR}
Min.	52.54	41.50	43.01	34.64	39.48	26.57	36.41	22.15
Max.	119.75	114.23	112.62	114.55	103.47	108.29	96.24	98.43
AVG.	85.39	82.06	76.61	73.69	71.21	68.48	66.18	63.53
SD.	16.14	16.29	17.67	18.23	15.59	16.79	13.33	14.81

Table 4.4: The parameters of black gram leaf polygon model

	α_{TL} (degree)	α_{TR} (degree)	α_{BL} (degree)	α_{BR} (degree)	w_L	w_R	p_L	p_R
Min.	14.17	9.46	39.48	26.57	0.17	0.10	0.06	0.12
Max.	76.69	73.50	103.47	108.29	0.48	0.51	0.56	0.56
AVG.	42.70	41.22	71.21	68.48	0.31	0.31	0.29	0.30
SD.	12.23	12.06	15.59	16.79	0.06	0.07	0.09	0.08

When the model is used in the leaf detection process, the average of the left and the right of each parameter is used. Then, it is multiplied by two excluding p. In summary, the value of α_T, α_B, w and p, are 41.96, 69.85, 0.31, and 0.30, respectively (see Table 4.5).

Table 4.5: The parameters used in leaf detection

	α_T (degree)	α_B (degree)	w	p
Min.	23.64	66.06	0.28	0.09
Max.	150.20	211.76	1.00	0.56
AVG.	93.92	139.70	0.62	0.30
SD.	24.30	32.38	0.12	0.09

4.3 Experiment 3: finding the error threshold in leaf detection

4.3.1 Objective

An error threshold is one of the important parameter used for polygon model evaluation in the leaf detection process. It represents the differences between the model and the actual leaf area (Section 3.3.4.2). To be specific, the larger threshold, the more errors are accepted, and the harder the segmentation is. On the other hand, if the threshold is too small, the model may never fit with the leaf, and the adjustment will be carried out until reaching the maximum iteration; as a result, a large computational power will be consumed. Therefore, a suitable error threshold is needed and this experiment was designed for this purpose.

4.3.2 Experiment setup

In this experiment, 101 binary black gram leaf images were used as inputs. For each image, there is only one leaf, which vertically lies at the middle of the image, and the apex and base positions were manually specified in advanced. The leaf detection algorithm discussed in Section 3.3.4 was run for 101 times, one time for each image, and in

each run, the maximum number of iterations was 30. In each iteration, the model tried to fit with the leaf as much as possible by adjusting itself in the model adjustment process. After 30 iterations, the algorithm was terminated regardless of the error threshold and sent the final error value as an output.

4.3.3 Result and discussion

Figure 4.5 represents error at the 30th iteration obtained from the experiment. From 101 images, an error value was acquired for each of them. Four statistical values were calculated from the outputs: minimum, maximum, average and standard deviation, as shown in Table 4.6.

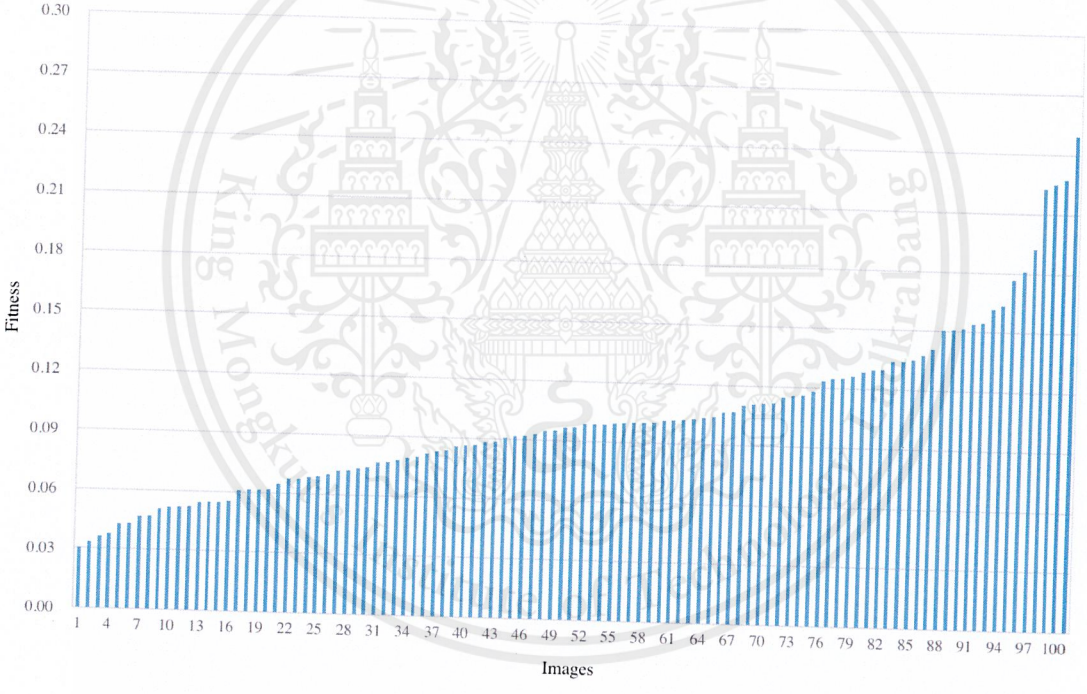


Figure 4.5: Error values at the 30th iteration of 101 images

According to Fig. 4.5, in which the graph was sorted by error, the graph continuously grows until 0.13 and dramatically increases after that. Besides, the threshold should not be higher than the combination of the average and the standard deviation, which is 0.14.

Table 4.6: The statistical values calculated from the outputs

	Min.	Max.	AVG.	SD.
Error	0.03	0.25	0.10	0.04

Hereby, 0.13 was chosen as the error threshold since it is lower than 0.14 and is the place where error increases strikingly. Figure 4.6 shows the results of fit model and unfit model with different error.

(a) Error = 0.03 (fit)

(b) Error = 0.23 (unfit)

Figure 4.6: Examples of leaf detection result

4.4 Experiment 4: evaluation of the proposed leaf detection

4.4.1 Objective

This experiment was done to measure the correctness of the leaf detection algorithm. The correctness represents the number of polygon models that fit with the actual leaf areas in percent. For more information, a model is said to be fit if and only if its error is lower than the predetermined error threshold, as stated in Section 3.3.4.2.

4.4.2 Experiment setup

The same the experiment setup as in Section 4.3, as well as, an error threshold of 0.13, which obtained from Section 4.3, were used in this experiment. After applying the leaf detection algorithm the same datasets of mid-vein detection experiment (see Section 4.1), the results were divided into two groups as follows.

Correctly detected mid-vein as inputs: A group of fit models which found from the correctly detected mid-vein.

Incorrectly detected mid-vein as inputs: A group of fit models which found from the incorrectly detected mid-vein.

4.4.3 Result and discussion

Table 4.7 shows that around 90% of the models were fit given that the mid-veins were detected correctly in dataset 1 while around 30% of them were fit in the dataset 2 given the correctly detected mid-veins. The reason for those unfit model is that some of the leaves' shapes are different from the general black gram, as shown in Fig. 4.7. As such, if the model is able to adjust its shape in an available range of SD, the accuracy should improve. Unfortunately, in this system, the model's parameters and its shape are fixed, and the only available operations are resizing and translation (see Section 3.3.4.3).

Table 4.7: Percentages of the result images of leaf detection

	Accuracy (%)	
	Correctly detected mid-vein as inputs	Incorrectly detected mid-vein as inputs
Dataset 1	92.70	40.00
Dataset 2	48.91	3.70

Figure 4.7: Examples of black gram with irregular shapes

4.5 Experiment 5: evaluation of leaf segmentation using snake algorithm

4.5.1 Objective

In order to perceive the performance of snake algorithm (see Section 3.3.5.1), the results of segmentation were compared with the original images.

4.5.2 Experiment setup

Snake algorithm requires a contour and an edge image as inputs. In this experiment, the second dataset of leaf images used to evaluate the mid-vein detection (Section 4.1.2) was used. For each image in the dataset, preprocessing, mid-vein detection, and leaf detection algorithms were applied to get a leaf contour. Then, if a mid-vein was detected correctly and the model from the leaf detection was classified as a fit model, the blurred gray-scale image from the preprocessing process and the model will be used as inputs of the snake algorithm.

In each round of the segmentation, snake's weight for continuity $\alpha(s)$, weight for curvature $\beta(s)$, and weight for *image energy* $\gamma(s)$ were set to 0.7, 1.0, and 3.0, respec-

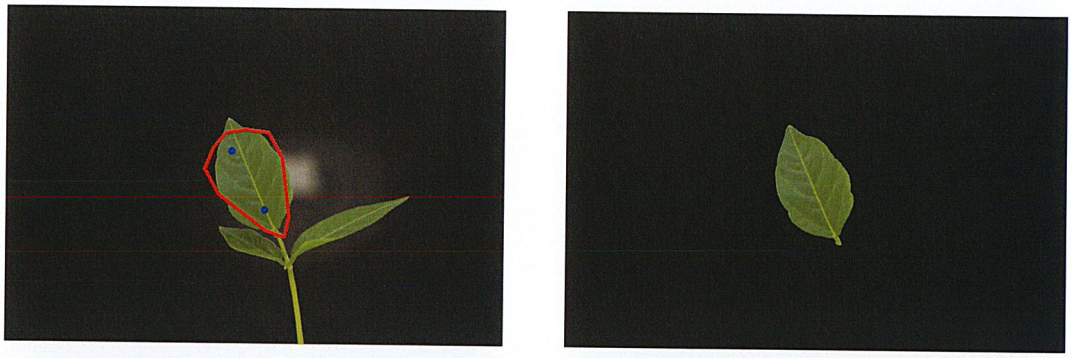
tively (see Section 3.3.5.1).

4.5.3 Result and discussion

Figure 4.8 shows the comparison between the input (left column) and the output images (right column). For each image on the left side, two blue points indicate the output of the mid-vein detection, while the red polygon is the contour generated by the leaf detection sent to snake. The corresponding image on the right is the output. From the result in which most of the leaves are segmented properly, the experiment was summarized as follows,

1. When the leaf shape is regular and there are no objects around the contour (Fig. 4.8a and Fig. 4.8c), the leaf will be segmented successfully (Fig. 4.8b and Fig. 4.8d).
2. When a part of the input contour is inside of the leaf area (Fig. 4.8f), the algorithm detects leaf's veins as the output contour. As a result, some of the leaf area in the output image disappears (Fig. 4.8e).
3. When the input contour is too far from the actual contour (Fig. 4.8g), the output image will contain some backgrounds and stalks (Fig. 4.8h).

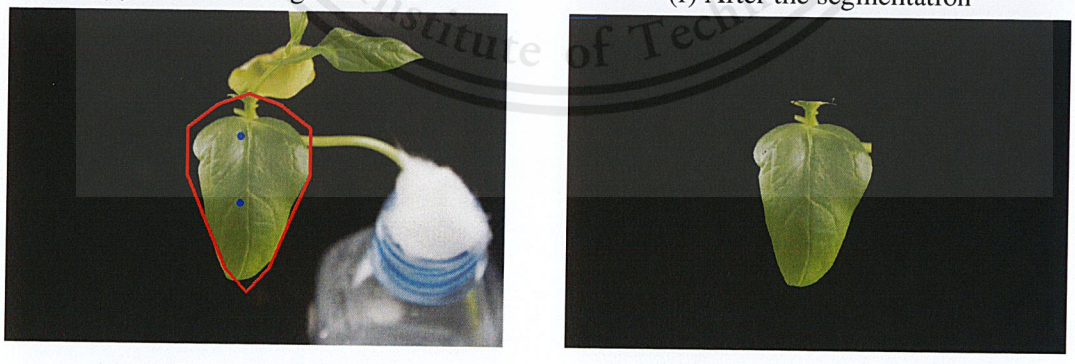
For the second and the third cases, the leaves were not segmented perfectly, though the results are acceptable.



(a) Before the segmentation (b) After the segmentation

(c) Before the segmentation (d) After the segmentation

(e) Before the segmentation (f) After the segmentation



(g) Before the segmentation (h) After the segmentation

Figure 4.8: Examples of image before and after applying snake algorithm

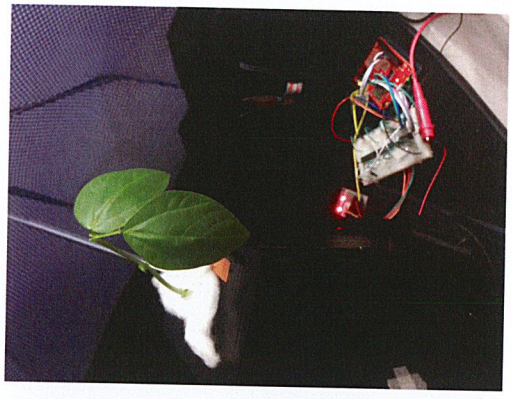
4.6 Experiment 6: verifying the integrated system's performance

4.6.1 Objective

After the integration of the robot arm unit, the plant rotation unit, and the processing unit, the system's performance was measured to verify that the system really works.

4.6.2 Experiment setup

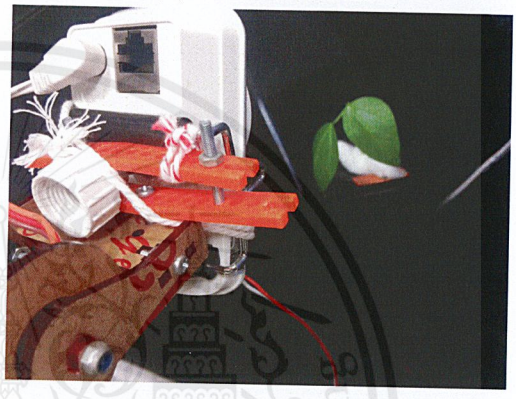
In this experiment, a planting bed with one plant rotator was used in the plant rotation unit and the robot rail were placed beside the planting bed. Further, power of the two Arduino UNO boards were supplied by Apple's MacBook Airs through USB cables, while the robotic arm and the plant rotator were supplied by a generator (see Fig. 4.9). The generator's setting were around 8V, 3A and 5V, 1A for the robotic arm and the plant rotator, respectively. As for the processing unit, it was established on an Apache web-server written in PHP language. Moreover, umbrellas and a photo booth were used to control the lighting condition in the environment.



(a) Plant rotation unit

(b) Robot arm unit

(c) A power supply



(d) Capturing a plant image

Figure 4.9: The integrated system

The system was run for seven times with different black gram plants. In each run, the processing unit processes (see Fig. 3.19) were performed for several iterations to generate outputs.

4.6.3 Result and discussion

During the experiment, the robotic arm's joints were adjusted and the plants were rotated when the processing unit could not detect a mid-vein or could not find a fit model. Henceforth, run times are different and at least one minute were needed when these occurred. The experimental result is shown in Table 4.8 and Table 4.9.

Table 4.8: Experimental result for each run of the integrated system

No. of run	Total iterations	Total segmented images	Total run time (mm:ss)
1	22	7	04:29
2	27	4	03:34
3	16	13	01:12
4	149	46	14:57
5	105	9	09:16
6	39	3	03:48
7	16	11	01:41
Total	374	93	38:56

Table 4.9: Experimental result in more detail

Run	Segmented image number	No. of iterations used for segmentaion	Time used for segmentaion (mm:ss)
1	1	9	02:15
	2	1	00:01
	3	1	00:01
	4	2	01:20
	5	1	00:01
	6	1	00:01
	7	6	00:49
2	1	2	00:33
	2	9	02:17
	3	14	00:43
	4	1	00:01

continued ...

...continued

Run	Segmented image number	No. of iterations used for segmentaion	Time used for segmentaion (mm:ss)
3	1	1	00:44
	2	2	00:07
	3	1	00:01
	4	1	00:01
	5	1	00:01
	6	1	00:01
	7	1	00:01
	8	1	00:02
	9	1	00:02
	10	1	00:01
	11	2	00:05
	12	1	00:02
	13	1	00:02
4	1	7	00:40
	2	25	03:42
	3	1	00:02
	4	7	00:20
	5	1	00:02
	6	1	00:02
	7	1	00:02
	8	1	00:02
	9	1	00:02
	10	1	00:02

continued ...

...continued

Run	Segmented image number	No. of iterations used for segmentaion	Time used for segmentaion (mm:ss)
	11	1	00:01
	12	1	00:02
	13	1	00:02
	14	17	00:48
	15	2	00:07
	16	1	00:02
	17	1	00:02
	18	1	00:02
	19	1	00:02
	20	1	00:02
	21	1	00:01
	22	2	00:04
	23	1	00:02
	24	1	00:01
	25	1	00:02
	26	1	00:02
	27	1	00:02
	28	1	00:02
	29	1	00:02
	30	1	00:01
	31	1	00:02
	32	1	00:02
	33	1	00:02

continued ...

...continued

Run	Segmented image number	No. of iterations used for segmentaion	Time used for segmentaion (mm:ss)
	34	1	00:02
	35	2	00:17
	36	1	00:02
	37	1	00:01
	38	1	00:02
	39	1	00:01
	40	1	00:02
	41	37	04:06
	42	1	00:01
	43	1	00:01
	44	1	00:01
	45	1	00:01
	46	12	03:51
5	1	19	01:25
	2	2	00:03
	3	1	00:02
	4	1	00:01
	5	25	02:54
	6	3	00:07
	7	29	03:47
	8	8	00:17
	9	16	00:40
6	1	36	03:45

continued ...

...continued

Run	Segmented image number	No. of iterations used for segmentaion	Time used for segmentaion (mm:ss)
	2	1	00:01
	3	1	00:01
7	1	1	00:47
	2	1	00:01
	3	1	00:01
	4	1	00:01
	5	4	00:38
	6	1	00:01
	7	2	00:04
	8	1	00:02
	9	1	00:02
	10	1	00:01
	11	1	00:01
Total		367	38:56
MIN		1	00:01
MAX		37	04:06
AVG		3.95	00:25
SD		7.40	00:58

From the tables, the longest run is around fifteen minutes while the shortest run is around one minute. Although, the longest run generated the largest number of segmented leaf images as outputs, it is possible for the number of outputs of shorter runs to be larger than that of the longer runs. From observations, the reasons behind this were:

1. The improper lighting conditions.

   (a) If the intensity of the light is too high, the leaves' areas became white and were filtered out by the green hue filtering of the preprocessing. Hence, zero leaf were found in an image even if there are.

   (b) If the intensity of the light is too low, the mid-vein detection algorithm couldn't detect mid-veins because the leaf veins' color were blended with the leaf blades color.

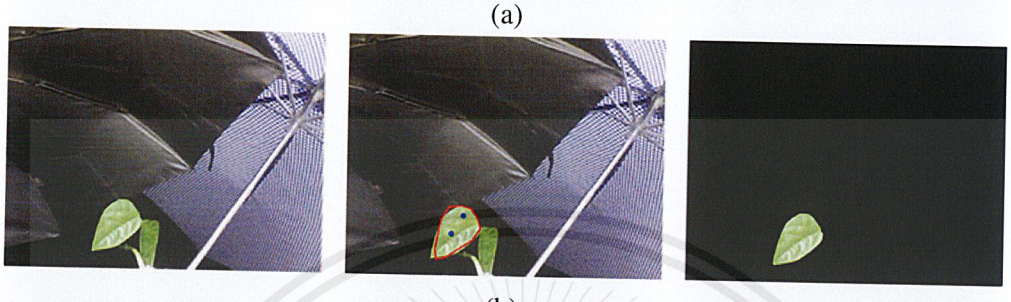
2. The overlapping leaves caused the leaf detection process to fail. The leaf detection process could not generate a fit model when the case happened because the model tried to adjust itself with all overlapped leaves and the error were measure from the the outer contour of the chunk of leaves.

3. The guiding algorithm used to control the robotic arm led the camera to wrong positions when zero green object was existed in the images.

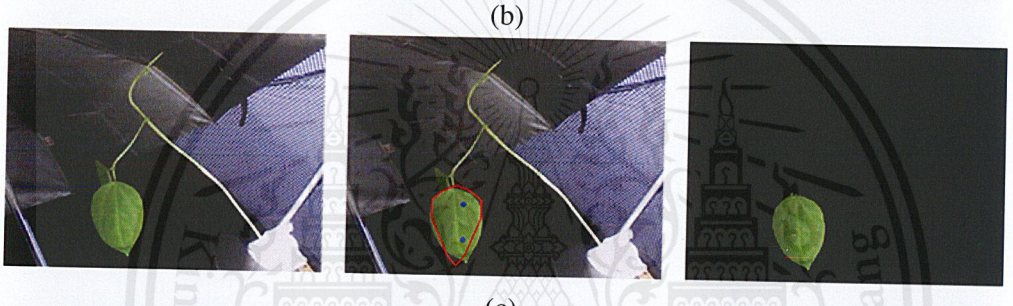
According to the information, it is clear that only a small number of iterations took minutes to complete. For the iteration that did not need to adjust the robotic arm and the plant, it took only a few seconds to generate a segmented image. Therefore, if the algorithm used to guide the robotic arm and the plant rotator is better, the system will use less time to finish the operations. In addition, by controlling the lighting condition and by improving the segmentation algorithm, the unnecessary control commands will not be made.

As for the system outputs, most of the time when the camera was placed at the proper positions, the leaves were found and segmented correctly as shown in Fig. 4.10. All things considered, the system is satisfiable.

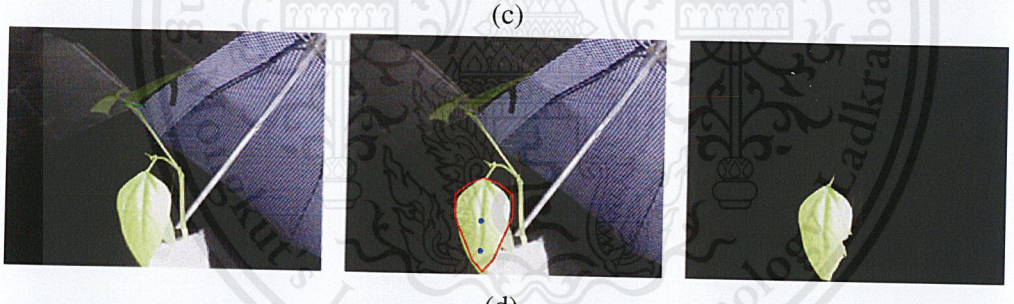
(a)



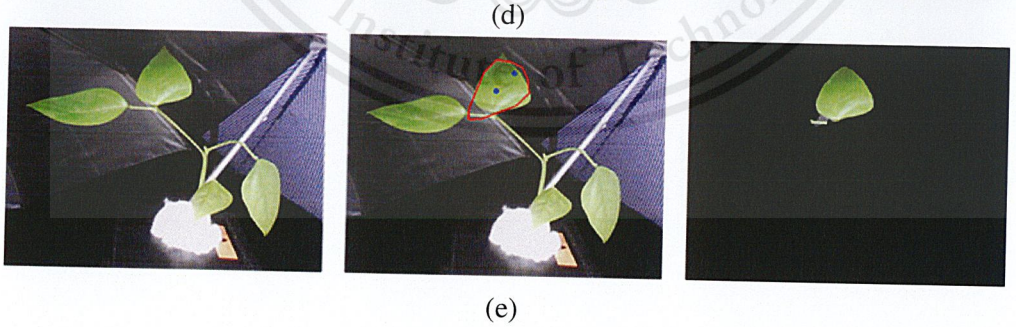
(b)



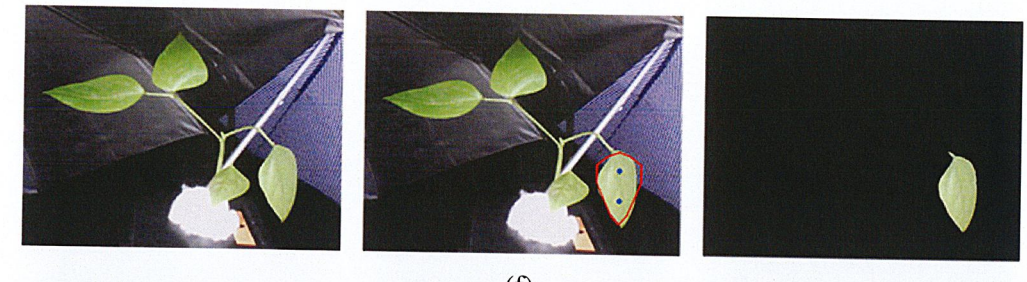
(c)



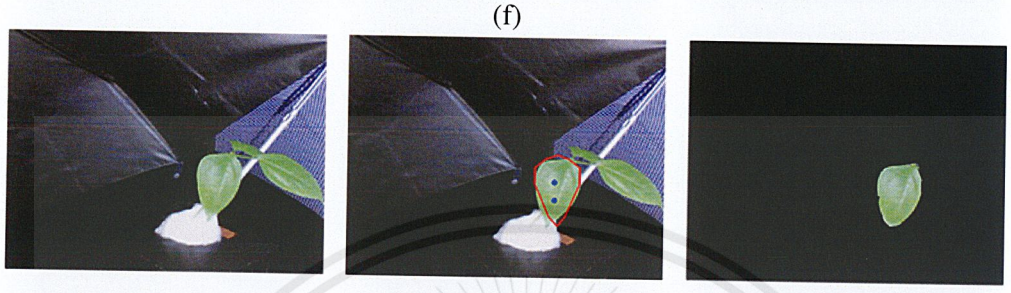
(d)



(e)

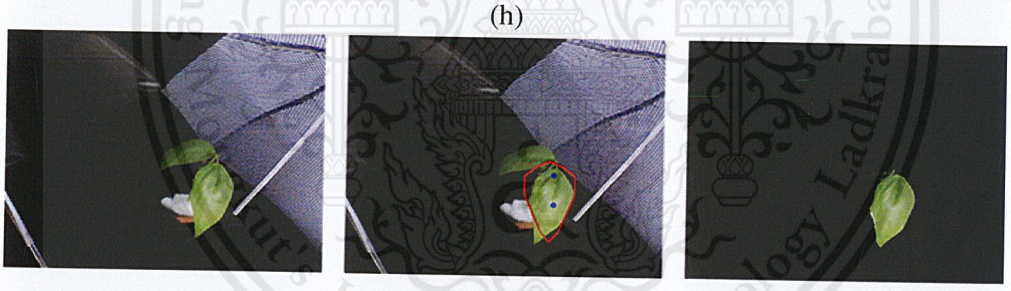


(f)

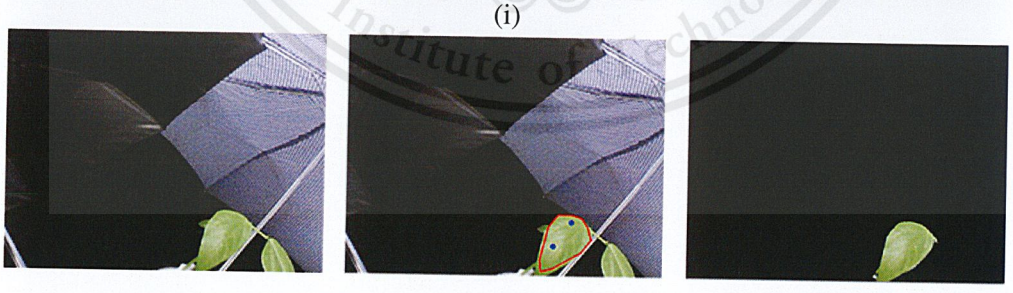


(g)

(h)



(i)



(j)

Figure 4.10: Results of integrated system. The first column consists of input images from the robotic arm's camera. The second column consists of images with mid-vein's points and a fit model. The last one consists of output images of the system.

Chapter 5

Conclusion

To summarize, this thesis proposes a system called Robotic Arm Base Visual System for Plant Monitoring. The system uses a robotic arm to capture plant images, focuses on leaves, and generates images of a leaf with plain background as output. Here, the system is designed to segment black gram leaves with regular shape. Divided into three parts, the system consists of the robot arm unit, the plant rotation unit, and the processing unit.

The robot arm unit consists of a robotic arm and a stainless rail. In the system, the robotic arm has four links, four rotary joints, an infrared distance measuring sensor, and a wireless IP camera attached at its tip. It is also attached on a four-wheel platform that allows it to move along the rail. The robotic arm's actuators and sensors are connected to a micro-controller board, which is an Arduino UNO. The Arduino UNO is also stacked by an Adafruit WiFi shield, which is used to establish a communication with the processing unit. In the system, its task is to move the camera according to commands sent from the processing unit.

The plant rotation unit has stepper motors which connect to another Arduino UNO board. Same as the robot arm unit, the board is also came with a WiFi shield, which is used to establish another connection with the processing unit. In plant rotation unit, the stepper motors are used to rotate plants and allow the plants' images to be taken in

various directions.

The processing unit is employed on a server. It communicates with the two units described before. In the communication, it may ask the robot arm unit to move the robotic arm's joint or the platform wheels to move the robotic arm to a specific plant, or ask the plant rotation unit to rotate a specific plant. However, its main task is to generate the system's outputs, which are segmented leaf images.

The processing unit accesses the IP camera attached at the robotic arm's tip to get input images and does four image processing processes to produce the outputs. The four processes include image preprocessing, mid-vein detection, leaf detection, and leaf segmentation. These processes are applied to an input image in a sequential order.

The first process, which is the preprocessing, converts an input image into the format required by the mid-vein detection using green hue filtering (Algorithm 6) and morphological operations (Section 2.2.7). The mid-vein detection detects a leaf mid-vein in the image and sends the mid-vein's end points to leaf detection. The leaf detection then creates a polygon model from the end points and a set of parameters obtained from an experiment (Section 4.2). After that, it applies the model to the leaf image and evaluates it. Leaf detection either adjusts the model when the model is not fit with a leaf or returns the fit polygon model as output. After the leaf detection, leaf segmentation is performed using snake algorithm. Snake algorithm receives the model and the leaf image as inputs. It leaf model is assumed to be a leaf contour, which initially is not the actual contour of the leaf. Snake tries to wrap the model to the actual leaf contour by adjusting it over iterations. Afterwards, the model adjusted by the algorithm is used to segment the leaf from the input image.

Moreover, in the cases that the mid-vein detection cannot detect a mid-vein or the leaf detection cannot find a fit model, the processing unit specifies a target position of the robotic arm's camera and commands the robotic arm's joints to move, or commands the plant rotation unit to rotate a plant, accordingly.

Additionally, the processes used in the system and the system after the integration were evaluated separately in experiments.

In Section 4.1, the mid-vein detection was evaluated with two different datasets including 101 segmented leaf images and 146 leaf images with plain background. From the experimental result, the accuracies of the proposed mid-vein algorithm is around 95% and 63%, for the first and the second datasets, respectively, which are acceptable. The others incorrect 5% and 37% include the case that the algorithm could not detect a mid-vein, the case that the mid-vein was detected from a leaf that is overlapped by other leaves, and the case that the other edges are detected as mid-veins.

In Section 4.4, the leaf detection was evaluated with the same datasets as in the mid-vein experiment. The experiment's result was divided into two groups including the group of fit models found from the correct mid-vein and the group of fit models found from the incorrect mid-vein. The experimental result shows that for dataset 1, around 90% of the images with the correct mid-vein were detected correctly while around 40% of the images with incorrect mid-vein were detected correctly. While, for dataset 2, only around 49% and 4% were correct for the two groups, respectively. An improvement method was suggest as by allowing an appropriate changes of the model's shape, adding to the two available operations which are resizing and translation, the error should be reduced.

In Section 4.5, the leaf segmentation using the snake algorithm (Section 3.3.5.1) was evaluated. Here, the same images as in the mid-vein experiment ware used again (the mid-vein experiment's second dataset). For each image, all processes explained previously were applies as described before, then the snake algorithm was applied to the polygon model obtained from the leaf detection process and the leaf image was segmented using the model adjusted by the snake algorithm. The segmentation results are favorable as most images were segmented properly. The results were incorrect only when the unfit models were given or when the irregular black gram images were used as

inputs. As such, if the leaves come with the general black gram shape and the leaf detection gives fit models as input, the result will mostly be correct. Besides, the accuracy can be improved by allowing the model to change its shape under some conditions.

The whole system's performance was verified in the last experiment (Section 4.6). The system was tested for seven times with different plants. The experimental result shows that minutes were needed to control the robotic arm and the plant rotator, while only a few seconds were needed for image processing. Furthermore, from the outputs of the system in which leaves ware segmented finely when the camera was placed at proper positions, the system is satisfiable. Specially, if the hardware guiding algorithm is improved, the time needed by one iteration of the system's operation will be shorter. Also, with a perfectly control over illumination and a better segmentation algorithm, the unnecessary commands will be eliminated, which should reduce the iteration's time further.

In conclusion, the system is finished and meeting the team members' expectation. In particular, after the improvements using the suggested methods, the system should be able to be used in practice.

Bibliography

[1] "5V stepper motor and stepper motor driver board," available at http://www.yourduino.com/sunshop2/index.php?l=product_detail&p=126/, last accessed: May 12, 2016.

[2] T. Alexander, T. Ben, and D. Kostas, "Object detection via boundary structure segmentation," Proceedings of 2010 IEEE Conference on Computer Vision and Pattern Recognition, pp.950-957, 2010.

[3] H.H. Assada, Introduction to Robotics, MIT. Boston, 2005.

[4] M. Beetz, Plan-Based Control of Robotic Agents: Improving the Capabilities of Autonomous Robots, Springer-Verlag, 2002.

[5] G. Cerutti, L. Tougne, J. Mille, A. Vacavant, and D. Coquin, "A parametric active polygon for leaf segmentation and shape estimation," Springer Berlin Heidelberg, Advances in Visual Computing, vol.6938, pp.202-203, 2011.

[6] A. Chaudhury, C. Ward, A. Talasaz, A.G. Ivanov, N.P.A. Huner, B. Grodzinski, and R.V. Patel et al., "Computer vision based autonomous robotic system for 3D plant growth measurement," Proceedings of 2015 Conference on Computer and Robot Vision, pp.290-296, 2015.

[7] "DC MOTOR: Basics, Types and Application," available at https://www.elprocus.com/dc-motor-basics-types-application/, last accessed: May 7, 2016.

[8] A. Elouafiq, Design and Engineering of a Chess-Robotic Arm, arXiv:1204.1649, 2012.

[9] R.C. Gonzalez and R.E. Woods, Digital Image Processing (3rd Edition), Prentice Hall, 2007.

[10] A. Gopal, S.P. Reddy, and V. Gayatri, "Classification of selected medicinal plants leaf using image processing," Proceedings of 2012 International Conference on Machine Vision and Image Processing (MVIP), pp.5-8, 2012.

[11] S. Hayashi, K. Shigematsu, S. Yamamoto, K. Kobayashi, Y. Kohno, J. Kamata et al., "Evaluation of a strawberry-harvesting robot in a field test," Biosystems Engineering, vol.105, no.2, pp.160-171, 2010.

[12] "How do servo motors work," available at http://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html/, last accessed: May 7, 2016.

[13] "How to make a robot," available at http://www.robotshop.com/blog/en/robots/gorobotics/tutorials/how-to-make-a-robot/, last accessed: May 7, 2016.

[14] "How to use a stepper motor," available at http://www.instructables .com/id/How-to-use-a-Stepper-Motor/step4/Unipolar-vs-Bipolar-stepper-motors/, last accessed: May 7, 2016.

[15] D.J. Kang, "A fast and stable snake algorithm for medical images," Pattern Recognition Letters, vol.20, no. 5, pp. 507-512, 1999.

[16] N. Kawamura, K. Namikawa, T. Fujiura, and M. Ura, "Study on agricultural robot (part 1)," Journal of the Japanese Society of Agricultural Machinery, vol.46, no.3, pp.353-358, 1984.

[17] M.W. Khan, "Image segmentation techniques: a survey," Journal of Image and Graphics, vol. 1, no.4, pp.166-170, 2013.

[18] S.D. Khirade and A.B. Patil, "Plant disease detection using image processing," Proceedings of 2015 International Conference on Computing Communication Control and Automation (ICCUBEA), pp.768-771, 2015.

[19] S. Kuuk and Z. Bingul, Industrial Robotics: Theory, Modelling and Control, Pro Literatur Verlag, 2006.

[20] W.S. Lee, D.C. Slaughter, and D.K. Giles, "Robotic weed control system for tomatoes," Precision Agriculture, vol.1, pp.95-113, 1999.

[21] H. Leopold and N.Garcia, Multimedia Applications, Services and Techniques - ECMAST'99 (4th Edition), Springer, 2003

[22] O. Marques, Practical Image and Video Processing Using MATLAB, Wiley, 2011.

[23] A. McAndrew, An Introduction to Digital Image Processing with MATLAB, Course Technology, 2004.

[24] B.M. Mehtre, N.N. Murthy, S. Kapoor, and B. Chatterjee, "Segmentation of fingerprint images using the directional image," Pattern Recognition, vol.20, no.4, pp.429-435, 1987.

[25] M. Monta, N. Kondo, K.C. Ting, G.A. Giacomelli, D.R. Mears, Y. Kim et al., "Harvesting end-effector for inverted single truss tomato production systems," Journal of the Japanese Society of Agricultural Machinery, vol.60, no.6, pp.97-104, 1998.

[26] M. Nixon and A. Aguado, Feature Extraction and Image Processing (2nd Edition), Academic Press, 2008.

[27] J.W. Orillo, J.D. Cruz, L. Agapito, P.J. Satimbre, and I. Valenzuela, "Identification of diseases in rice plant (Oryza Sativa) using back propagation artificial

neural network," Proceedings of 2014 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), pp.1-6, 2014.

[28] J.R. Parker, Algorithms for Image Processing and Computer Vision, John Wiley and Sons, 2010.

[29] J.C. Russ, The Image Processing Handbook (6th Edition), CRC Press, 2011.

[30] B. Siciliano and O. Khatib, Springer Handbook of Robotics, Springer-Verlag Berlin Heidelberg, 2008.

[31] K. Singh, I. Gupta, and S. Gupta, "Classification of bamboo plant based on digital image processing by central moment," Proceedings of 2011 International Conference on Image Information Processing (ICIIP), pp.1-5, 2011.

[32] V. Singh, P.C. Pande and D.K.Jain, A Text Book of Botany: Angiosperms (3rd Edition), Rastogi publications, 2008.

[33] V. Singh, Varsha, and A.K. Misra, "Detection of unhealthy region of plant leaves using image processing and genetic algorithm," Proceedings of 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA), pp.1028-1032, 2015.

[34] M.W. Song and M. Vidyasagar, Robot Dynamic and Control (1st Edition), Wiley, 1989.

[35] "Stepper motor basics," available at https://www.circuitspecialists.com/stepp ermotor/, last accessed: May 7, 2016.

[36] N. Valliammal and S.N. Geethalakshmi, "Hybrid image segmentation algorithm for leaf recognition and characterization," Proceedings of 2011 International

Conference on Process Automation, Control and Computing (PACC), pp.1-6, 2011.

[37] E.J. Van Henten, B.A.J. Van Tuijl, J. Hemming, J.G. Kornet, J. Bontsema, and E.A.V. Os, "Field test of an autonomous cucumber picking robot," Biosystems Engineering, vol.86, no.3, pp. 305-313, 2003.

[38] K. Wong, K. Lam, and W. Siu, "An efficient algorithm for human face detection and facial feature extraction under different conditions," Proceedings of 2001 Pattern Recognition, vol.34, no. 10, pp.1993-2004, 2001.

[39] L.M. Xu and T. Z. Zhang, "Present situation of fruit and vegetable harvesting robot and its key problems and measures in application," Transactions of the Chinese Society of Agricultural Engineering, vol.20, no.5, pp.38-42, 2004.

[40] A. Yadav and P. Yadav, Digital Image Processing, Laxmi Publications, 2009.

[41] C. Yetim, Kinematic Analysis for Robot Arm, BSc. Thesis, Yildiz Technical University, Istanbul, 2009.

[42] D. Zhao, J. Lu, J. Wei, Y. Zhang, and Y. Chen, "Design and control of an apple harvesing robot," Proceedings of 2011 Biosystems Engineering, vol.110, pp.112-122, 2011.

[43] Z. Ziao-dong and W. Xiao-jie, "Feature extraction of plant leaf based on visual consistency," Proceedings of 2009 International Symposium on Computer Network and Multimedia Technology, pp.1-4, 2009.