THESIS APPROVAL

GRADUATE SCHOOL, KASETSART UNIVERSITY

Master of Engineering (Information and Communication Technology for Embedded Systems)
DEGREE

Information and Communication Technology for Embedded Systems	Electrical Engineering
FIELD	DEPARTMENT

TITLE: Planar Surface Area Calculation Using Camera and Orientation Sensor

NAME: Ms. Surangrak Sutiworwan

THIS THESIS HAS BEEN ACCEPTED BY

	THESIS ADVISOR
(Mr. Miti Ruchanurucks, Ph.D.)	
	THESIS CO-ADVISOR
(Mr. Supakorn Siddhichai, Ph.D.)	
	THESIS CO-ADVISOR
(Professor Makoto Sato, Ph.D.)	
	THESIS CO-ADVISOR
(Miss Thitiporn Chanwimaluang, Ph.D.)	
	DEPARTMENT HEAD
(Assistant Professor Teerasit Kasetkasem, Ph.D.)	

APPROVED BY THE GRADUATE SCHOOL ON

	DEAN
(Associate Professor Gunjana Theeragool, D.Agr.)	

THESIS

PLANAR SURFACE AREA CALCULATION USING CAMERA AND ORIENTATION SENSOR

SURANGRAK SUTIWORWAN

A Thesis Submitted in Partial Fulfillment of
the Requirements for the Degree of
Master of Engineering (Information and Communication Technology for Embedded Systems)
Graduate School, Kasetsart University
2012

Surangrak Sutiworwan 2012: Planar Surface Area Calculation Using Camera and Orientation Sensor. Master of Engineering (Information and Communication Technology for Embedded Systems), Major Field: Information and Communication Technology for Embedded Systems, Department of Electrical Engineering. Thesis Advisor: Mr. Miti Ruchanurucks, Ph.D. 124 pages.

To calculate planar surface object, this research proposes image warping to top view algorithm. The sensor attached to a camera is used to compensate the camera's orientation in real time. In practice, the alignment of sensor and camera is imperfect. This error makes the calculated area size inaccurate. To address this problem, calibration between camera and sensor is required by using Iterative Least Square method. However, the imperfect alignment also causes the time shifts between the camera and sensor. This is due to the computer system added the unknown latency between sending data. In this contribution, the arrangement processes needs to be identified to reduce the elapsed timing. Then, the extrinsic parameters derived from calibrated sensor and pre-computed intrinsic parameter will be used to generate a homography matrix. In top view image, we can directly count the number of target pixel. Finally, we will also find the relationship between rotation angle and size of each pixel in real-world unit with calculating percentage accuracy of this method. The experimental results show top view target area size generated from the tilted camera using information from the orientation sensor in real time.

______________________ ______________________ ___ / ___ / ___

Student's signature Thesis Advisor's signature

ACKNOWLEDGEMENTS

I would like to grateful thank and deeply indebted to Dr. Miti Ruchanurucks my thesis advisor from Kasetsart University, Dr. Supakorn and Dr. Thitiporn from National Electronics and Computer Technology Center, and Dr. Makoto Sato from Tokyo Institue of Technology for giving me a great suggestion.

This research is supported in part by Center for Promoting Research and Development of Satellite Image Applications in Agriculture and Thailand Advanced Institute of Science and Technology (TAIST), National Science and Technology Development Agency (NSTDA), Tokyo Institute of Technology and Kasetsart University for financial support.

I am thankful all of my friends, especially, Mr. Nueng Jarunroungrok who fulfill other parts of this project and whom I can refer to whenever is needed or necessary for the extempore situation. Here, I am very appreciative to Ms. Pornprawee, our official, as well for all of kindnesses to answer questions and remind me to complete each work in time.

Surangrak Sutiworwan
June 2012

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF ABBREVIATIONS

9DOF	=	9 Degrees of Freedom
AHRS	=	Attitude and Heading Reference System
ASIC	=	Application Specific Integrated Circuit
CCWS	=	China Crop Watch System with Remote Sensing
DCM	=	Direct Cosine Matrix
FPGA	=	Field Programmable Gate Array
GIS	=	Geographic Information System
GPS	=	Global Positioning System
GVG	=	GIS, VIDEO and GPS
IDE	=	Integrated Development Environment
OpenCV	=	Open Computer Vision library
RGB	=	Red, Green, and Blue
SAD	=	Sum of Absolute Difference
VGGVS	=	Vehicle & GIS & GPS & Video System
XML	=	Extensible Markup Language

PLANAR SURFACE AREA CALCULATION USING CAMERA AND ORIENTATION SENSOR

INTRODUCTION

Around 75 percent of the world’s poor people live in rural areas. Most of them are involved in farming. Agricultural development in these areas is often constrained by issued of access to appropriate technologies. Many countries and agricultural systems thus remain mired in underdevelopment and face major barriers to the use of knowledge and innovation for development (Asenso-Okyere *et al*., 2008). To calculate the amount of agricultural product is an arduous work of harvesting field crops. Field survey and satellite image are commonly used for this computation. The satellite image is a favorite media for creating land use and land cover maps which are used to estimate the agricultural area, a period of growths, a harvest time and classify each type of crop. A ground survey is still an essential part for the agricultural product estimation. A field survey is usually performed by human. However, the problem of using field survey is that it uses a lot of man-power, time, and expense. Even in remote sensing of satellite image is sensitive to weather. Also a scheme to perform image classification of satellite data is inadequate (Dengxin.D *et al*., 2011). Therefore, to reduce time and cost, automatic ground survey system has been proposed for crop calculation.

The method of video-based ground survey system may be installed on the vehicle moving around the cropland and uses images captured from that video camera. Recently, In China, Crop Watch System with Remote Sensing uses their algorithm with applying Vehicle & GIS & GPS & Video System (VGGVS) for navigation, data collection, and computation. On the top of a vehicle, this method attaches a camera and an orientation sensor together in order to control the estimation precision. A camera rotation is adjusted by a motor and receives an angel from the sensor. While the camera is receiving the angel information, the motor needs to be perpendicular with ground plane all the time. Therefore, the motor adjusting is necessary for the algorithm. However, there has many disadvantages of adjusting the motor for example it is costly and large, as well as requires high power consumption and high maintenance.

In our work, the motor shall be removed to reduce the system size and cost. The systems have to apply image warping algorithm instead. Warping transforms a perspective image to its top view image from which crop area can be estimated systematically and easily. This transformation requires multiple inputs so called homography. Homography based on the orientation sensor has to be derived to generate a top view image transformation similar to (LinBo *et al*, 2009), (H. Kano *et al*, 2008). One crucial parameter of homography matrix is rotation information. The rotational information is acquired from an orientation sensor ultimately, image warping algorithms using homography matrix based on the sensor are proposed. Then, counting the number of pixel on the top view image followed by calculating the actual area size in the real world has to be performed in the real time.

The camera model in OpenCV (open source computer vision library) is used as it has many good features, such as the precise calibration results, efficient computing, a fast computing speed, etc. as discussed in as discussed in (Wang, Y. M 2010), (Yuan Xin *et al*, 2011),. It is used as a reference throughout this work. For each frame of image, the homography matrix can be derived with camera’s intrinsic and extrinsic parameters. Camera calibration technique is used to identify intrinsic parameters as in (OpenCV 2.0 documentation), whereas extrinsic parameters are derived in real-time using information from an orientation sensor. Furthermore, our contribution also finds a relationship between the top view pixel size and the actual area size.

However, the problem of this calculation causes from an imperfect alignment between the camera and the sensor. First problem is about the position alignment. Coordinates of x, y z, from camera must be parallel with coordinates x, y, z from sensor. Iterative Least Square (ILS) method (Ziraknejad. N ,2007), is used to solve this problem. Second problem is about aligning the camera and the orientation sensor data in time. The computer system adds unknown latencies to the data when acquiring an image or reading sensor data (Frahm, J.-M, 2003). We need to identify how to reduce this different time between camera data and sensor. Finally, height from camera to ground should be able to be varied.

OBJECTIVES

Our proposed scheme of area calculation system focuses on image warping to top view using orientation sensor. For area calculation, the sensor is used to compensate the unknown / varied video camera’s orientation. Our assumptions are among the area to be estimated is planar surface. Then, we can estimate the size of an area by computer vision methods. The objectives consist of 5 steps.

1. The system size and cost consumption need to be reduced. The motor could be eliminated. Image warping to top view is applied instead information from the sensor is used for a warping image algorithm instead of motor adjusting.

2. The top view image is generated using orientation sensor information in real time.

3. The orientation between the camera and the sensor is calibrate for achieving high precision planar surface area calculation.

4. The height of the camera from the planar surface is variable.

5. A number of the target pixels are mapped to planar surface area in real world unit in order to approximate the area size.

LITERATURE REVIEW

1. Field Survey System

There is another way to calculate agricultural area. It is called called on-board field area meter (OBFAM) that equipped with an azimuth sensor and a distance sensor to provide measured area data for agricultural machinery. This system utilizes a simple compass for measuring azimuth angle and a proximity switch for measuring distance.

To measure the azimuth angle, it is performed by detecting the magnet of the earth and the distance sensor is mounted on the wheel shaft of the vehicle. As the wheel shaft is rotating, a square-wave signal is generated. Then, the vehicle moves from a starting point along the contour of the field to calculate the cultivated area by using the counter embedded in the W78E516 microprocessor as shown in Figure 1.

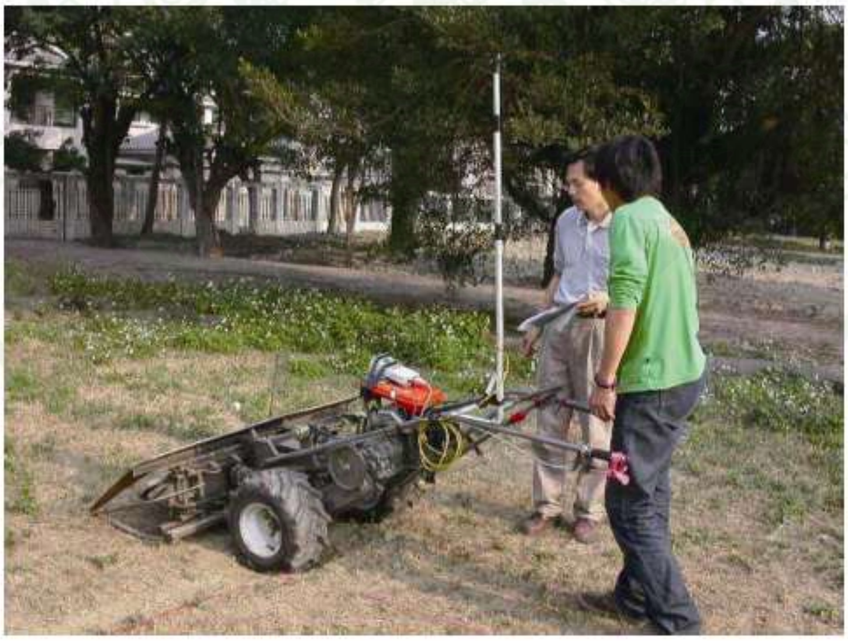


Figure 1 Photograph showing testing of the OBFAM.

Source: Yuang *et al.* (2010)

2. A Video Base Field Survey System on a Vehicle within China Crop Watch System

Wu *et al.* (2006), Wang *et al.* (2006) and Tian *et al.* (2004) proposed China Crop Watch System with Remote Sensing (CCWS). This algorithm applied Vehicle & GIS & GPS & Video System (VGGVS) for navigation implementation, image data collection and crop proportion computation. CCWS consists of seven models: crop growth monitoring, drought monitoring, grain production estimation, crop production prediction, crop planting structure inventory, cropping index monitoring, and grain supply–demand balance and early-warning (Figure 2). The monitoring can be carried out on different scales or levels ranging from a county through a province and the whole country to the main producing countries in the world.

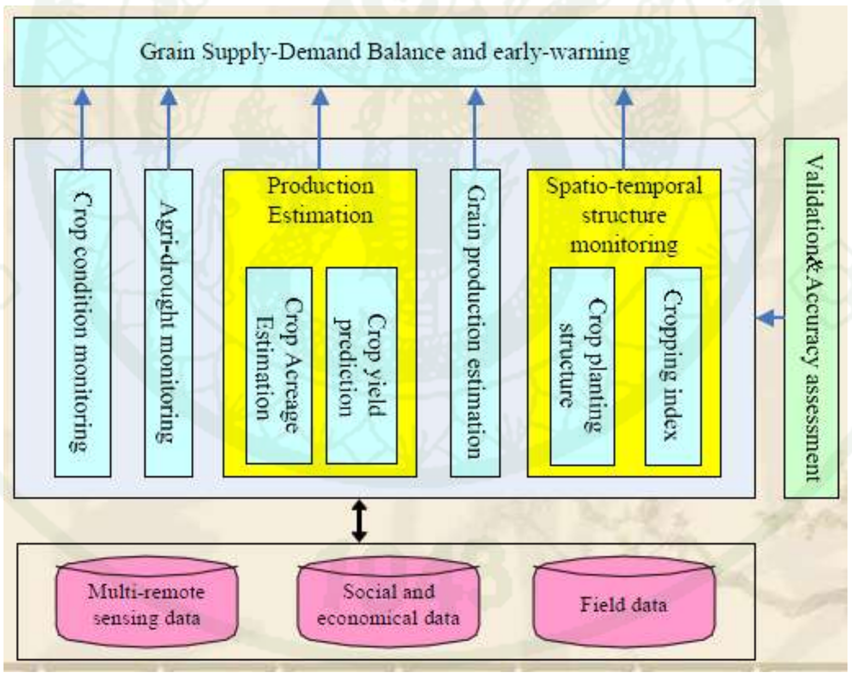

Grain Supply-Demand Balance and early-warning
Crop condition monitoring
Agri-drought monitoring
Production Estimation
Crop Acreage Estimation
Crop yield prediction
Grain production estimation
Spatio-temporal structure monitoring
Crop planting structure
Cropping index
Validation& Accuracy assessment
Multi-remote sensing data
Social and economical data
Field data

Figure 2 Contents of China crop watch system using remote sensing.

Source: Wu *et al.* (2006)

A new crop surveys and analyzes system for VGGVS, which uses video capture camera and GPS receiver to integrate GIS, VIDEO and GPS (GVG) into a notebook computer (Figure 3). When GVG is activated, it saves pictures and at the same time, system notes the position with long/lat format and geographical attribute under the support of GPS and GIS, all data are memorized into database automatically. On account of the GVG catch all data in same time, it's far more effective than those traditional fieldwork methods and so easy to handle, only one person can do all jobs in the car.

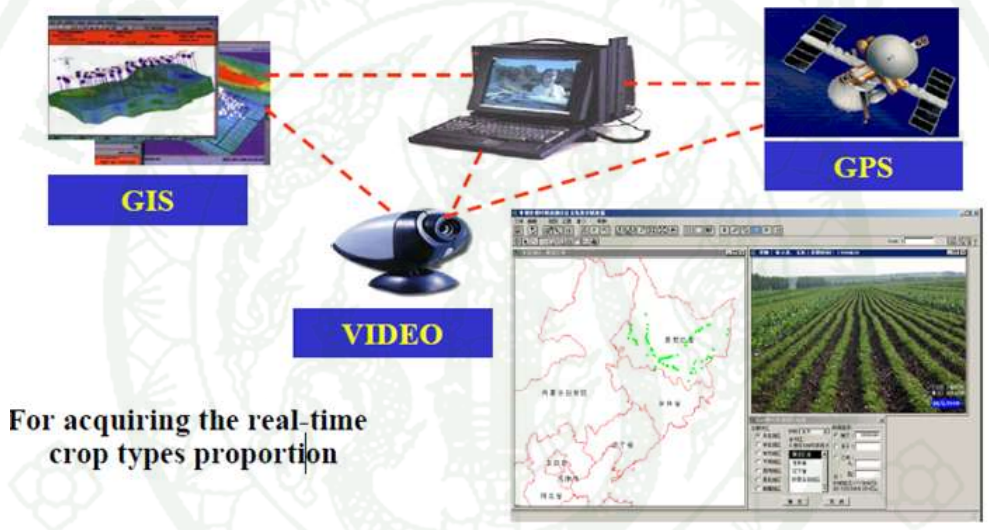


Figure 3 GVG field sampling system.

Source: Tian *et al.* (2004)

A camera with orientation sensor was attached on the top of vehicle that is fixed height as Figure 5, In order to control the estimation precision. The sensor monitors the camera rotation. Then the camera is adjusted by motor which received a camera rotation angle from the sensor. In Crop Watch System with Remote Sensing in China algorithm, the camera that captures input as Figure 5 needs to be perpendicular with ground plane all the time. Hence, the adjusting motor is necessary for the algorithm.

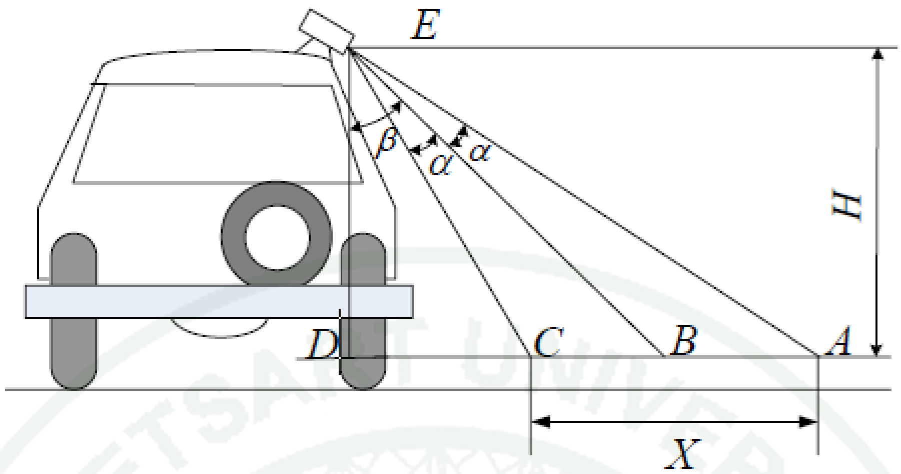


Figure 4 The camera with orientation sensor was attached on the top of vehicle.

Source: Wang *et al.* (2006)

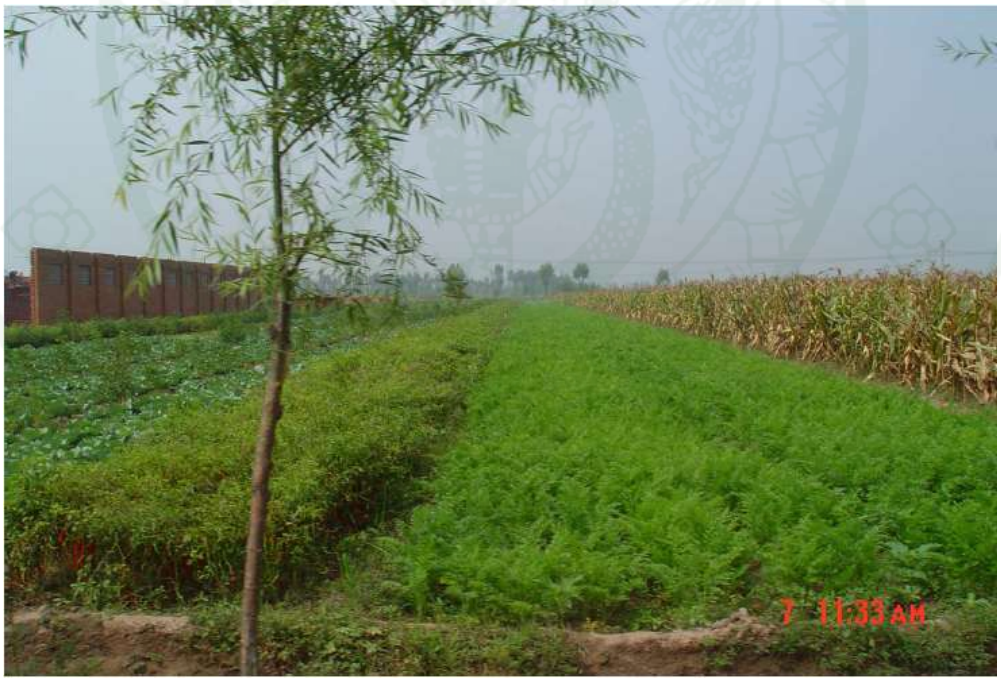


Figure 5 The image is captured from GVG system.

Source: Tian *et al.* (2004)

3. Another Agricultural Field Robot

Doing agricultural task by remote sensing is becoming more and more expensive. This motivates to build a robot capable of navigating in a plantation for performing agricultural tasks. It composed of four parts: computer, micro mobile vehicle, Charge Couple Device (CCD), and navigation sensor. as shown in Figure 6.

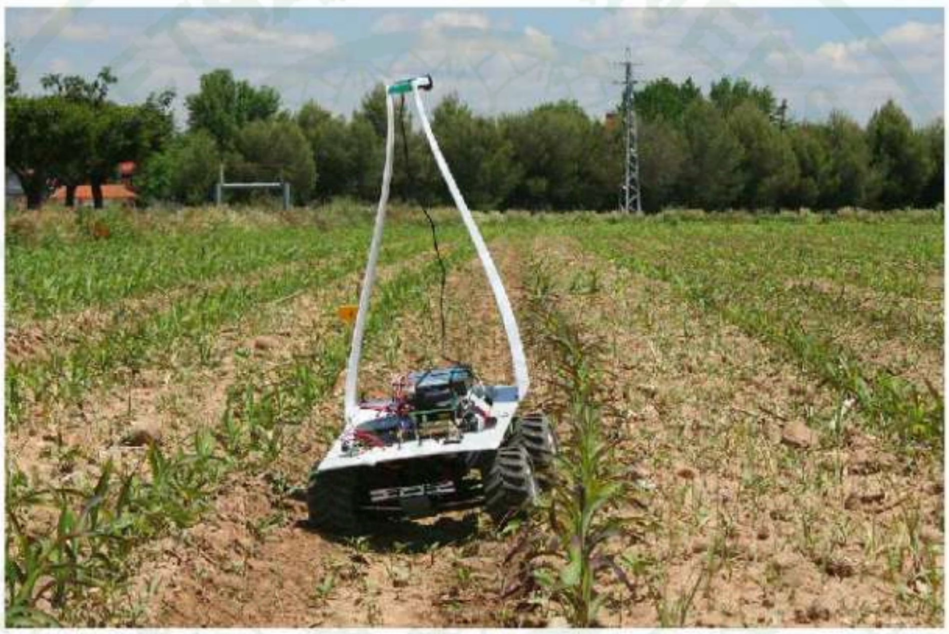


Figure 6 The vehicle in its final state in the testing field.

Source: Richard *et al.* (2008)

The camera is fixed on the top of the robot, and pointing to the ground. Because agricultural robots often operate in farm field with limited elevation changes, it is acceptable to simplify the model from a 3D space to a 2D plane (as shown in Fig. 7) to reduce computational load. We can calibrate the camera using 2D image coordinates and 2D world coordinates.

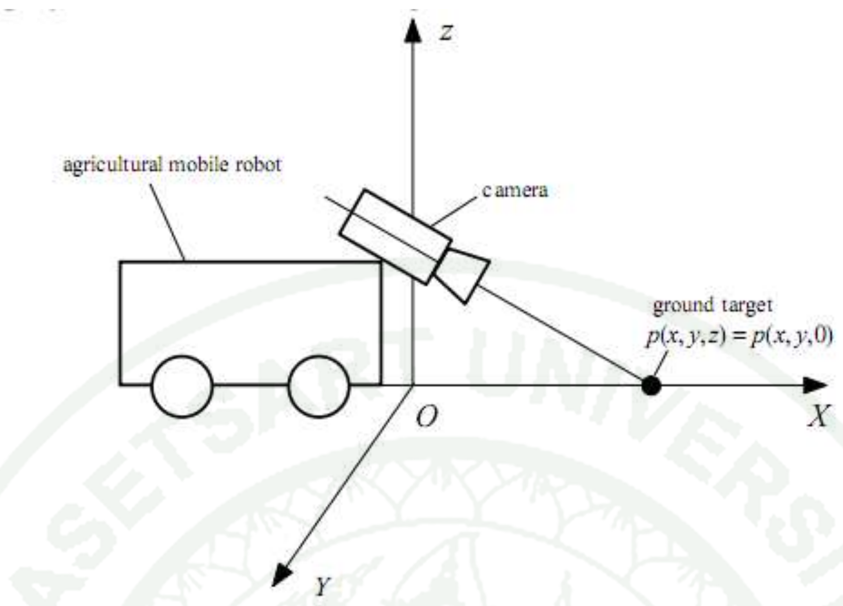


Figure 7 The world coordinate system and a ground target: O-X-Y-Z is the robot coordinate system; *p*(x,y,0) is a ground target in the world coordinate system.

Source: Guoquan *et al.* (2010)

When the micro vehicle is moving, CCD captures the images of crop rows continuously, navigation baselines can be obtained after image processing and Hough transform in a computer. Then according to the guidance information, the computer sends the anticipant oriented-wheel control angle to mobile vehicle. At the same time, system accepts the feedback information in order to adjust the mobile vehicle more effectively, as is shown in Figure 8.

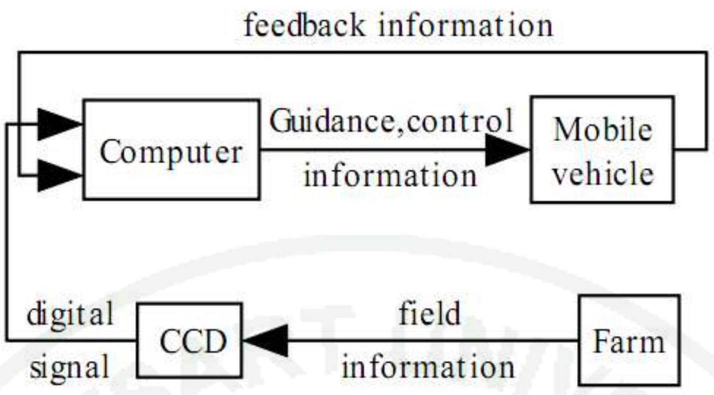


Figure 8 Machine vision navigation system model.

Source: Cui-Jun *et al.* (2010)

The problem is that webcams typically have a viewing angle of less than 50° which would make it necessary to mount the camera at an approximate height of 150cm as shown in Figure 9. That means the CCD is fixed on the top of the robot all the time.

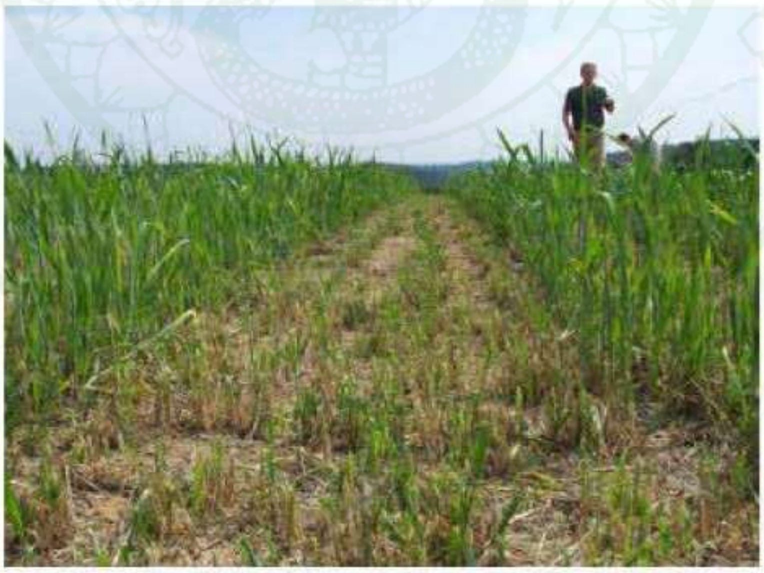


Figure 9 Viewing angle less than 50°.

Source: Jose *et al.* (2006)

There is a solution which is using omnidirectional camera as show in Figure 10. There are several advantages. First more and longer rows can be capture. Second, it sees plants beside the robot which may give better estimate of alignment error. Finally it sees behind the robot which gives the opportunity to achieve better calculation area. Among the omnidirectional cameras there are both cameras with fisheye lens (Figure 10a) ans with catadioptric lens (Figure 10b). The major difference is the range of azimuthal view. A fisheye lens starts from zero which means it sees straight ahead, and end somewhere above 90. The catadioptric lens on the other side cannot see straight ahead due to its construction, but it has a wider range above 90.

The image analysis on omnidirectional images can be categorized in two groups, those who require the image to be unwrapped and those who are applied directly on the omnidirectional image. The unwrapping is a time consuming process and for real-time applications on a mobile robot the latter is to prefer. The algorithms used in this work do not need the images to be unwrapped. However, in an agricultural scene the robot is moving in uneven terrain and the tilt is required to estimate the position of the robot. A drawback of using this is that it does not deal with tilt.

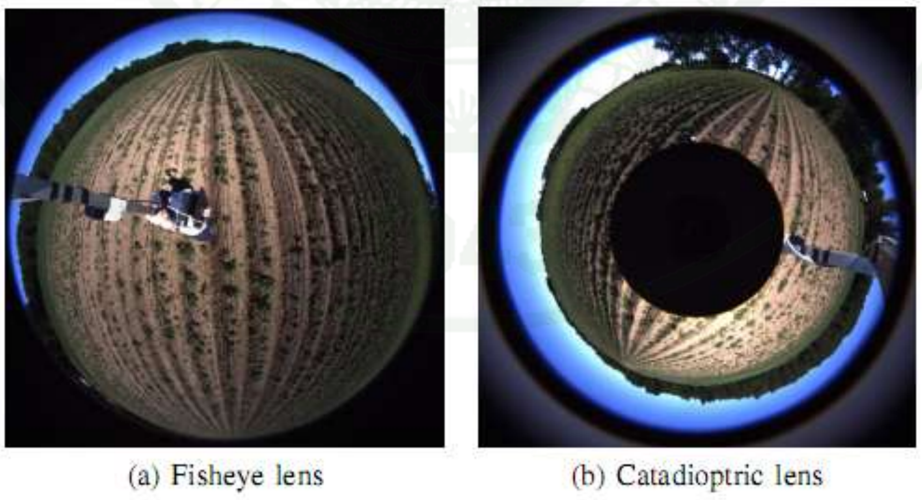


Figure 10 Image on omnidirectional camera.

Source: Stefan *et al.* (2010)

4. Warping to Top view application

Image warping is the process of digitally manipulating an image. It may be used for correcting image distortion as well as for creative purposes. The same techniques are equally applicable to video. In general, the homography matrix or 2D-to-2D projective matrix could be found by four sets of corresponding points between the reference points of the input image as Figure 11a and the calibration pattern points of the output image as Figure 11b. Many researches generated homography matrix from four sets of corresponding points of corner of chessboard.

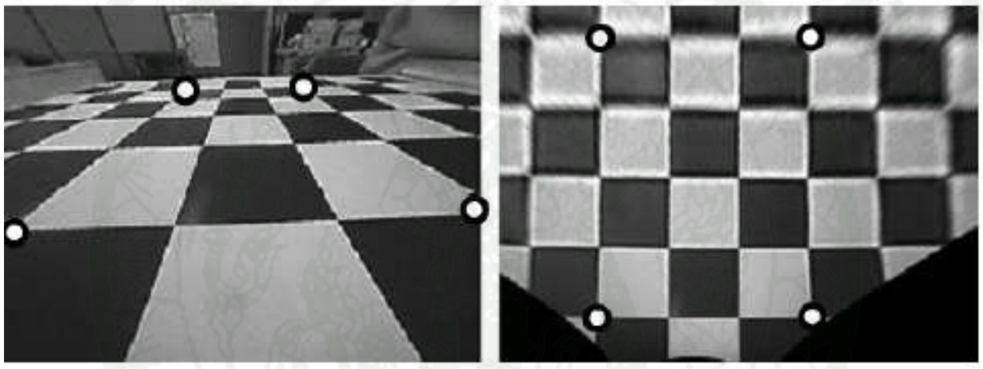
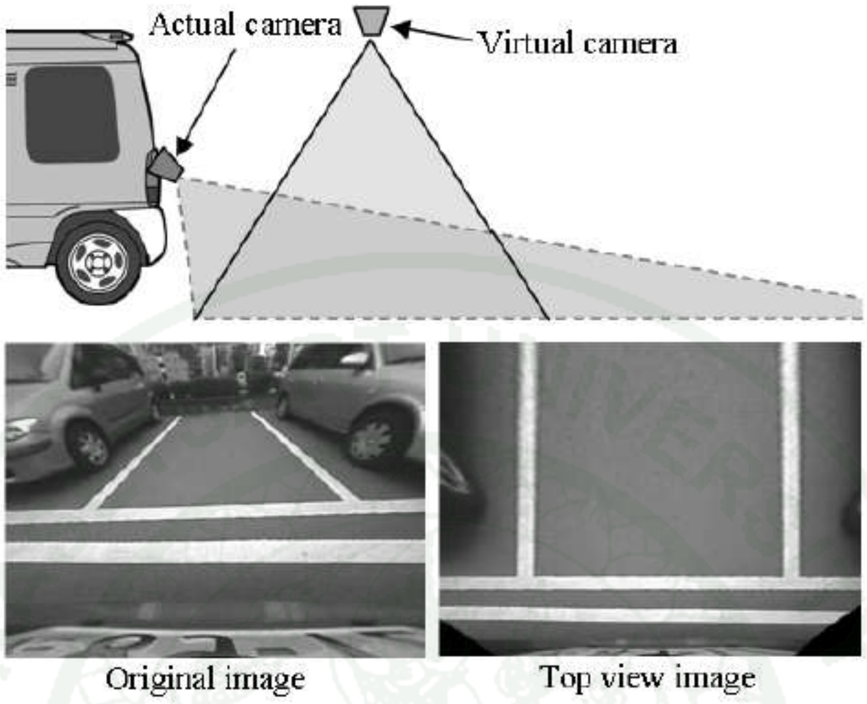


Figure 11 Four sets of corresponding points between (a) the reference points of the input image and (b) the calibration pattern points of the output image.

Source: Kano *et al.* (2008)

Probably one most easily seen application of image warping is automobile's rear camera. The number of rear view backup cameras being installed in automobiles is increasing. The main use of such camera is to show the blind spots of vehicles, however, it is difficult to correctly perceive distances by simply viewing as Figure 6a because of the perspective effect of the camera. Kano *et al.* (2008) applied the top view image transformation for their perceiving distance methods as Figure 12b.

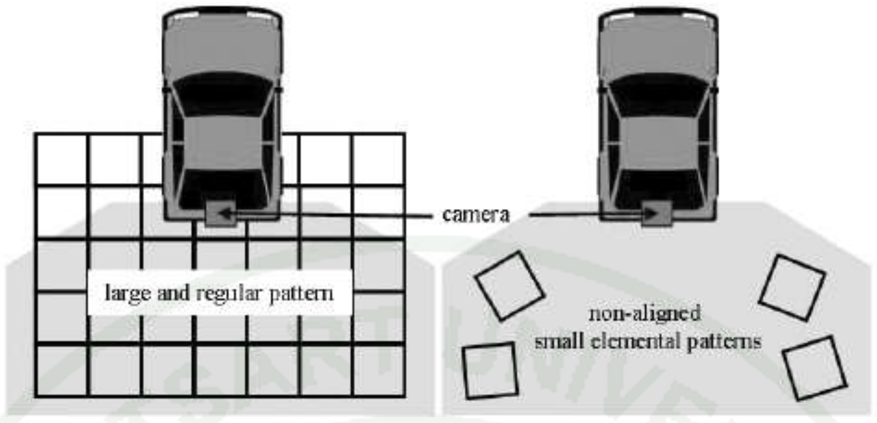


(a) (b)

Figure 12 Rear view cameras in automobiles (a) Original image from Actual camera (b) Top view image from Virtual camera.

Source: Kano *et al.* (2008)

In their algorithms the camera is assumed no rotation. The algorithms were proposed to eliminate the perspective effects for helping driver parking. They compare large and regular pattern method as Figure 13a and non-aligned small elemental pattern method as Figure 13b. These algorithms use smaller elemental patterns and place them at arbitrary positions on the road's surface. All of the element patterns are the same and are square in shape. Starting from an initial homography computed using one of the element patterns, the tentative homography matrix is optimized by iteratively minimizing the error function, taking into account all patterns.



(a) (b)

Figure 13 Rear view cameras in automobiles method (a) large and regular pattern method and (b) non-aligned small elemental pattern method.

Source: Kano *et al.* (2008)

The procedure for optimizing the homography is shown in Figure 14. In the first step, one of the element patterns is selected. In the second step, a homography matrix that transforms the selected pattern into a fixed size square is computed. Finally, this matrix is used as the initial value of an iterative process.

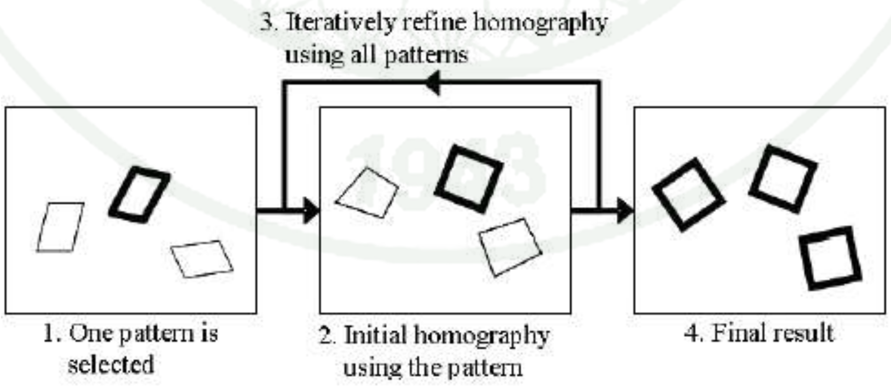

3. Iteratively refine homography using all patterns
1. One pattern is selected
2. Initial homography using the pattern
4. Final result

Figure 14 Homography optimization procedures.

Source: Kano *et al.* (2008)

In the experiment, these algorithms use the rear view camera that its pitch angle is fixed at 45 degrees. The lens distortion is corrected as Figure 15. After that, they use non-aligned small elemental pattern method and show the result as Figure 16.

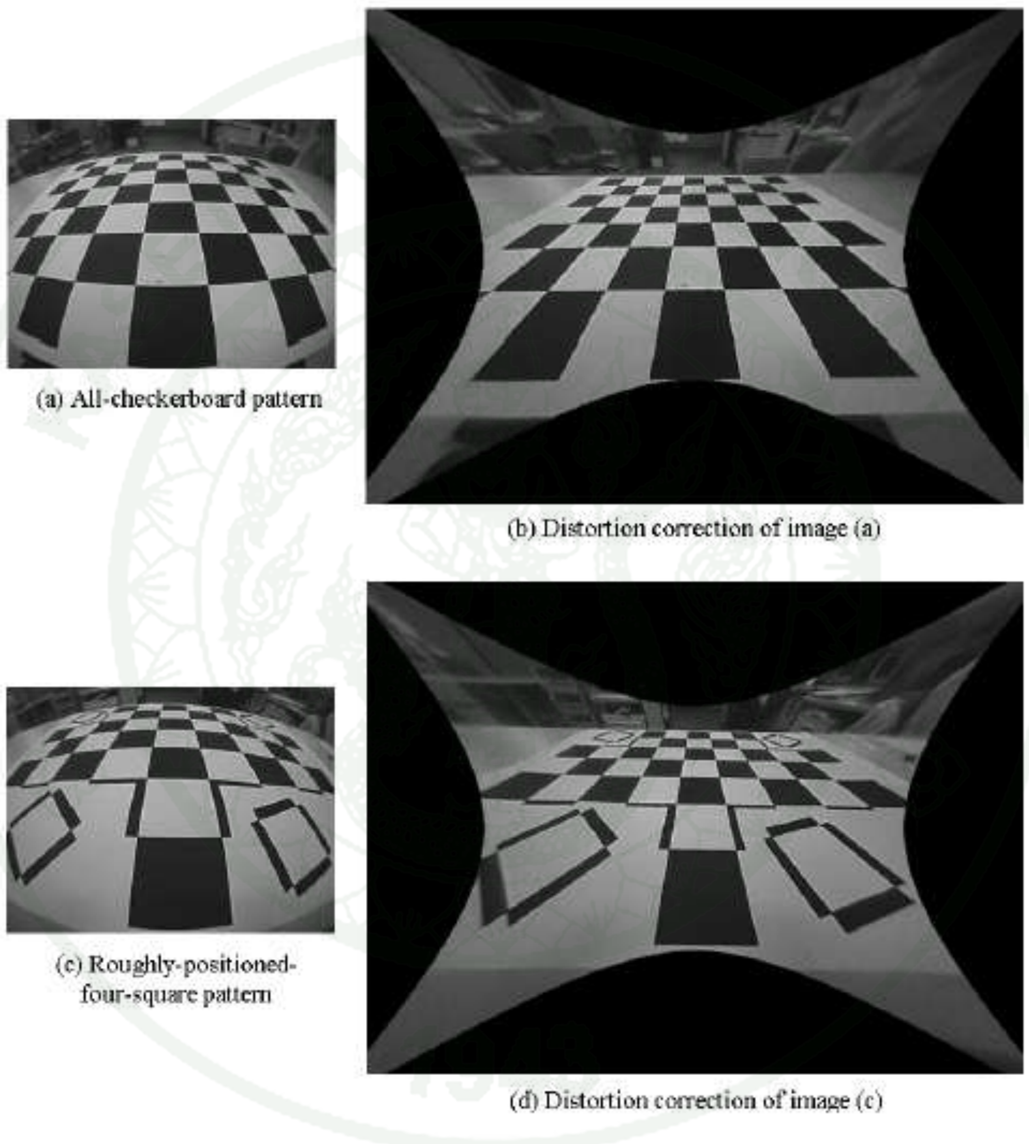

(a) All-checkerboard pattern
(b) Distortion correction of image (a)
(c) Roughly-positioned-four-square pattern
(d) Distortion correction of image (c)

Figure 15 Distortion correction of wide angle camera.

Source: Kano *et al.* (2008)

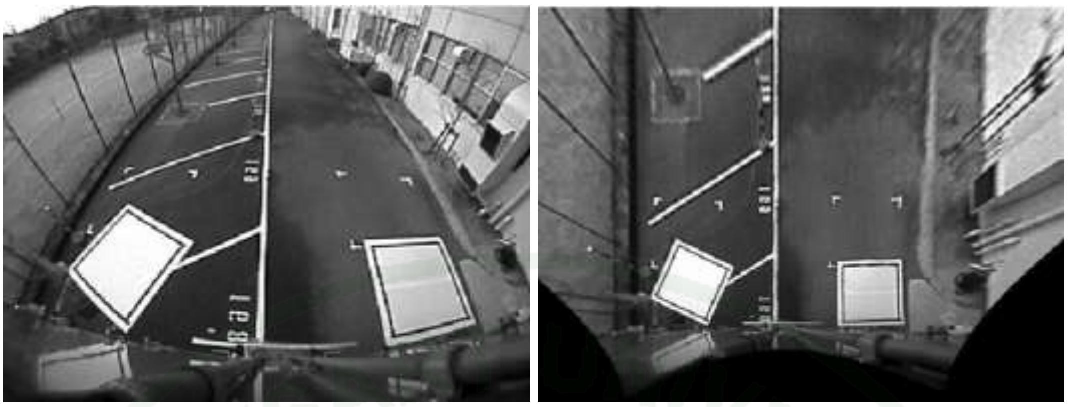

(a) (b)

Figure 16 Top view is generated using camera mounted on a vehicle (a) Input image and (b) top view image.

Source: Kano *et al.* (2008)

Luo *et al.* (2009) implemented homography matrix in hardware. Whole system as Figure 17 is composed of two parts, a soft processor and a FPGA/ASIC hardware. The software part is used to initiate system and control the data flow, while the hardware implementation is used to perform the repetitive tasks. Part I is to calculate transformation matrix as an initialization of whole system. Part II is the transformation module which transfers input original video into top view. Part I is implemented by software purely. Part II is an ASIC or a FPGA-based hardware module. Because they only need to calculate the transformation matrix once while initializing system, Part I doesn’t need to implement by hardware. It can be implemented by an embedded system or a special off-line program. If they get the transformation matrix, Part II can transfer input images into top view images by using the hardware structure. The perspective transformation has to be done repetitively, pixel by pixel, and frame by frame, so that it’s better to implemented by hardware.

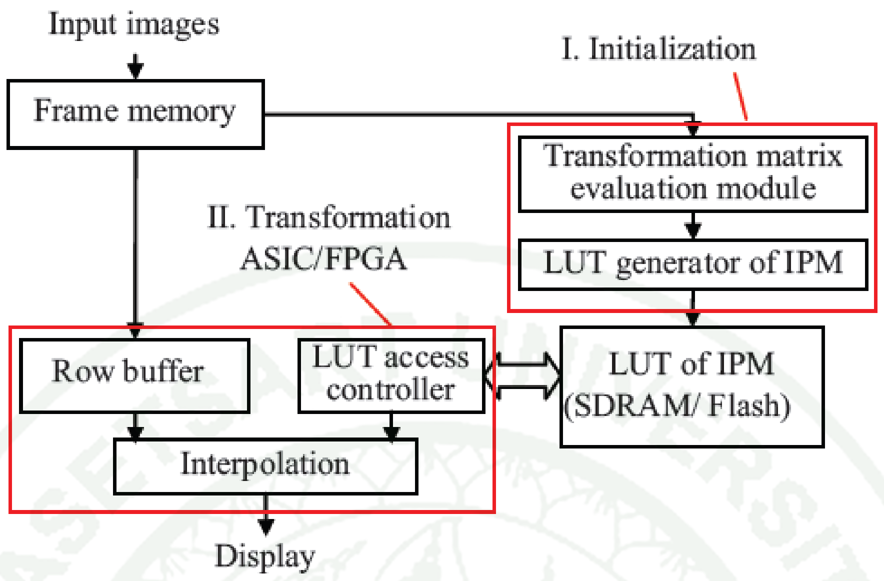


Figure 17 Structure of top view system; Part I: Initialization by software, Part II: Transformation ASIC/FPGA by hardware.

Source: Luo *et al.* (2009)

Teshima *et al.* (2006) proposed Vehicle Lateral Position Estimation Method Based on Matching of Top View Images. The algorithm as Figure 18 obtained homography matrix for calculating the sum of absolute different (SAD) between present and next frame image. The camera attached on the vehicle captures the front view images to find the vehicle lateral position. Computation of a next frame homography matrix depends on the hypothesis vehicle direction angle and the vehicle speed. The input frames are converted to the top view images according to the homography H_N and H_P as shown in Figure 19.

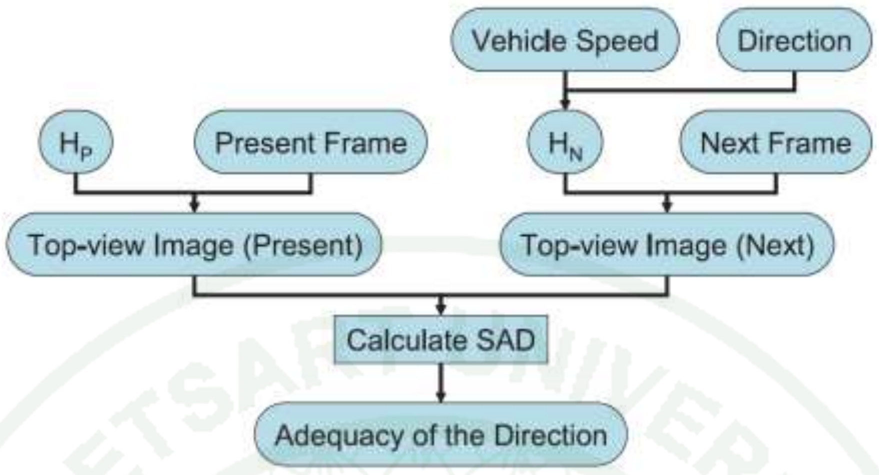
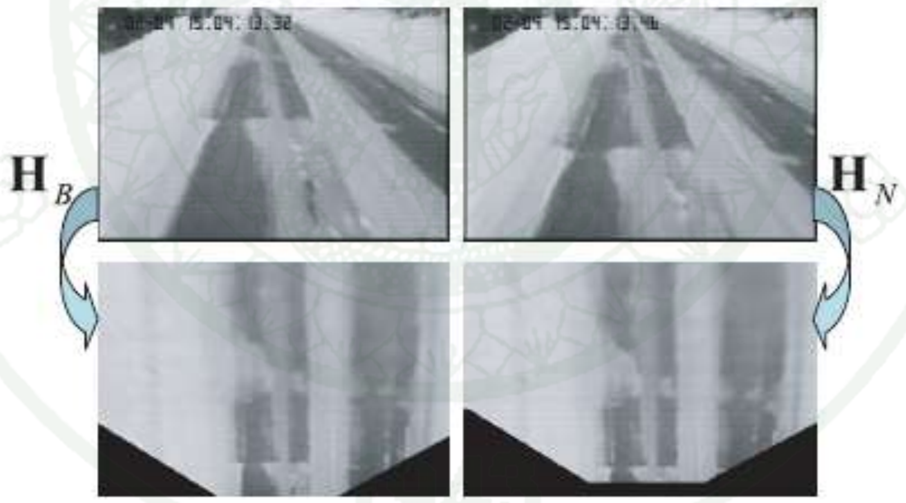


Figure 18 Framework of the Vehicle Lateral Position Estimation Method.

Source: Teshima *et al.* (2006)



(a) (b)

Figure 19 Transforming to the top view for (a) present frame by using H_B and (b) next frame by using H.

Source: Teshima *et al.* (2006)

Zhonlong *et.al* (2011), and Bijo *et.al* (2011) proposed Bird's eye view parking assistance systems that provide a driver an overhead view of the car and its surroundings. This method integrates multiple image from a video captures by the camera stitched around a vehicle as show in Figure 20,

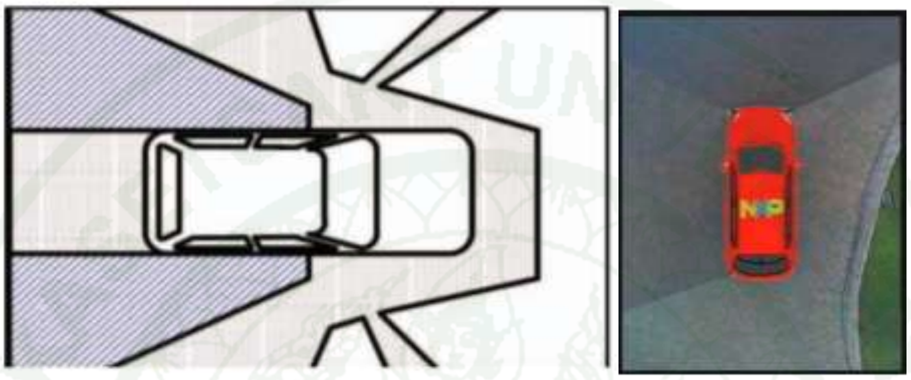


Figure 20 Snapshot of Bird view output.

Source: Zhonlong *et.al* (2011)

The system uses multiple fish eye camera that have to do distortion correction. Moreover, color balancing and picture enchantment are performed to improve bird eye view images as show in Figure 21

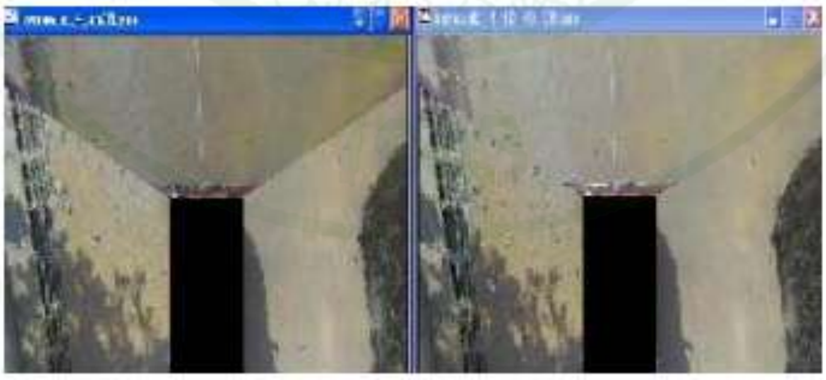


Figure 21 No color balance (left image) and with color balance (right image)

Source: Bijo *et.al*. (2011)

MATERIALS AND METHODS

Materials

Hardware

1. Computer set
2. Orientation sensor (9 Degrees of Freedom)
3. CCD Camera

Software

1. Microsoft Visual C++ 2008 Express Edition software
2. OpenCV (Open Computer Vision) library
3. MATLAB Simulation software
4. Microsoft Excel software
5. Vpython
6. Arduino

etc.

1. Chessboard (a camera calibration pattern)
2. Reference planar area

Methods

1. System Overview

Our hardware system consists of a camera attached to an orientation sensor, both connected to a computer. The camera is used to capture images of planar surface areas in different angles. In this work, we focus on how to generate top view image from perspective view image using the orientation information. This relies on software algorithms.

For calculating the area, images are captured in the perspective view. The captured image's quality is improved by removing the lens's distortion in the undistortion process. Effect of perspective view changing can be reduced by transforming every image to a certain top view image. Therefore, an image warping technique is introduced to visualize a perspective view to a top view. The technique requires calculation of a homography matrix, which consists of three components: intrinsic matrix, translation vector, and rotation matrix.

As the rotation matrix is varied in real time, so we derive it using the orientation sensor that is installed to parallel with the ground. The problem is that the coordinates of orientation sensor and camera are not perfectly aligned. To acquire the corrected rotation matrix, iterative least square method is used for solving this problem. After that the undistorted images are transformed to the top view image by applying the homography matrix image warping technique. Color threshold can be used to detect area of interest. The number of detected pixel is then map to area size in real world unit. This methodology is organized into three steps as shown in Figure 22.

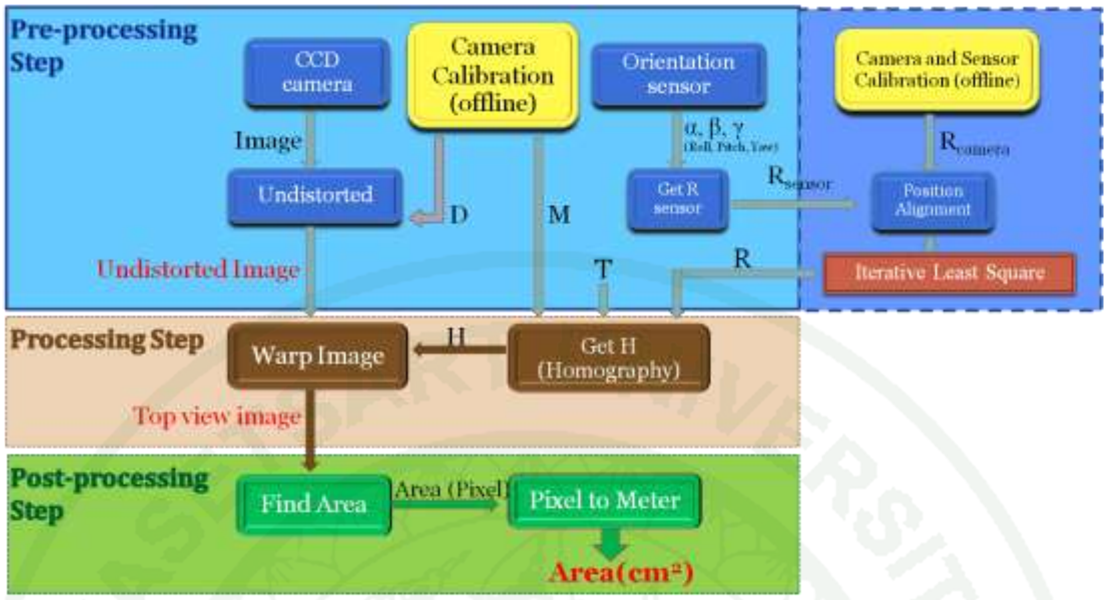


Figure 22 System Overview.

2. Pre-Processing Step

The transformation matrix from perspective to top view is called homography matrix which consists of intrinsic matrix (M), extrinsic matrix (rotation and translation [R|T]). Intrinsic and distortion matrix, mentioned previously are generated by a camera calibration process. Extrinsic matrix is generated by the orientation sensor.

Homography matrix is a transformation matrix from a perspective view to a top view which consists of intrinsic matrix (M) and extrinsic matrix ([R|T]). Intrinsic parameters (M) and distortion matrix (D) are generated by camera calibration process. Extrinsic parameters, in general, translation vector (T) and rotation matrix (R) can be derived by the camera calibration as well. However, we address a problem of translational parameters (T) in homography matrix (H). We found out that two (out of three) translation parameters make the output image unorganized if Roll angle change. So instead of doing translation directly in the homography matrix, a separate translation vector would be applied after the image was warped to top view. We will explain the basic of calibration first, and will show how to further calculate extrinsic matrix for moving camera later.

2.1 Pinhole Camera Model

From Gary and Adrian (2008), simple model as Figure 23, “light is envisioned as entering from the scene, but only a single ray enters from any particular point. In a physical pinhole camera, this point is projected onto an imaging surface. For our idealized pinhole camera, the distance from the pinhole aperture to the screen is precisely the focal length (f).” A similar triangle theory is used for pinhole camera model as Equation 1.

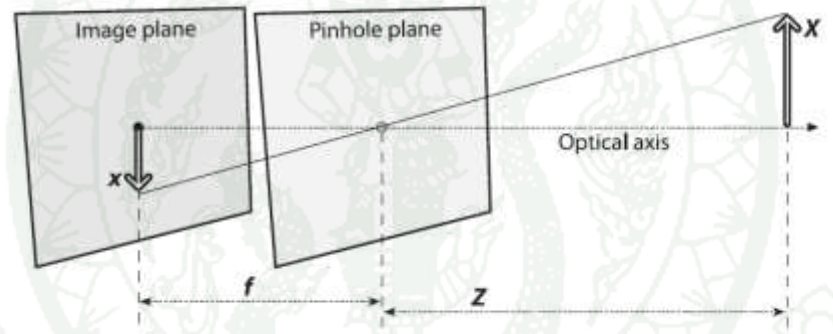


Figure 23 Pinhole Camera Model.

Source: Gary and Adrian (2008)

$$-x = f\frac{X}{Z} \tag{1}$$

Simplification as Figure 24, Gary and Adrian (2008) said “a point Q is projected onto the image plane by the ray passing through the center of projection, and the resulting point on the image is q. The image plane is really just the projection screen pushed in front of the pinhole.”

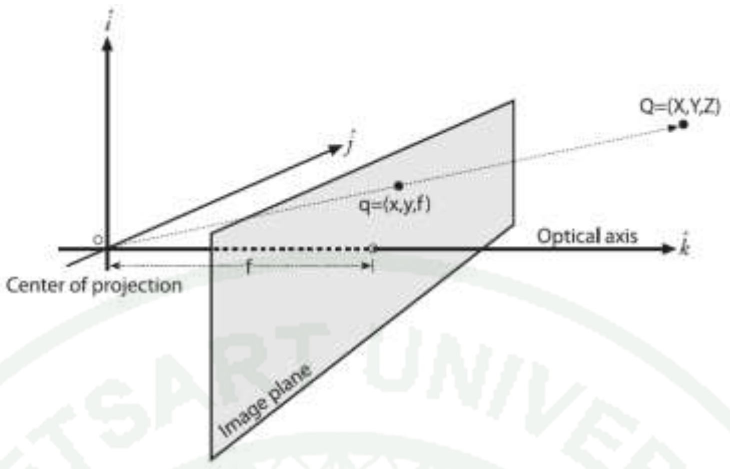


Figure 24 New frontal image plane.

Source: Gary and Adrian (2008)

Normally, principal point is not the center of the imager, so a pixel can be represented as Equation 2 for x axis and Equation 3 for y axis.

$$x = f_x \frac{X}{Z} + c_x \tag{2}$$

$$y = f_y \frac{Y}{Z} + c_y \tag{3}$$

2.2 Camera Calibration

2.2.1 Intrinsic and Extrinsic matrix

Camera Calibration is the process of determining the parameters of the camera setup. The intrinsic parameters are those specific to the camera, such as the focal length, principal point and lens distortion. These parameters can be determined by camera calibration, as proposed by Zhang *et al.* (2006). Extrinsic parameters refer to the 3D position and orientation of the camera. Calibration is often the primary step of many vision applications as it allows systems to determine a relationship between what appears on an image and where it is located in the world.

Knowledge of the camera calibration matrix is required for many basic image processing operations such as the removal of radial distortion.

According to the pinhole model, the camera performs a perspective projection of a 3D point onto an image point located on a retinal plane. Using homogenous coordinates, the projective relation between a 3D point and its image can be expressed as Equation 4.

$$sq = M\left[R \mid T\right]Q \tag{4}$$

The 3x4 matrix $M[R \mid T]$ is the projection matrix. It relates world points to image points according to the camera location with respect to the reference frame, represented by a rotation matrix R and a translation vector T. The rotation parameter $\left[r_{mn}\right]$ and translation parameter $\left[t_m\right]$ is used to describe the rigid motion of an object in front of still camera. It can be written as Equation 5.

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5}$$

Where,

$Q = [X, Y, Z, 1]^t$ is the coordinate of a 3D points in the world coordinate space.

$q = [x, y, 1]^t$ is the coordinate of the projection points in pixels.

M is called a camera matrix, or a matrix of intrinsic parameters. The intrinsic parameters are those specific to the camera, such as the focal length (f_x, f_y) which is expressed in pixel-related units, the principal point (c_x, c_y) which is equivalent to the center of the image.

$[R|T]$ is called a matrix of extrinsic parameters. It is used to describe the camera motion around a static scene, or vice versa, rigid motion of an object in front of still camera. The rotation of camera is first around the z-axis, then around the new position of the y-axis, and finally around the new position of the x-axis. This research focuses on the rotation matrix as Equation 6.

$$R = R_z R_y R_x \tag{6}$$

Or,

$$R = \begin{bmatrix} \cos\gamma\cos\alpha & \cos\beta\sin\alpha + \sin\beta\sin\gamma\cos\alpha & \sin\beta\sin\alpha - \cos\beta\sin\gamma\cos\alpha \\ -\cos\gamma\sin\alpha & \cos\beta\cos\alpha - \sin\beta\sin\gamma\sin\alpha & \sin\beta\cos\alpha + \cos\beta\sin\gamma\sin\alpha \\ \sin\gamma & -\sin\beta\cos\gamma & \cos\beta\cos\gamma \end{bmatrix} \tag{7}$$

Where,

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix}, R_y = \begin{bmatrix} \cos\gamma & 0 & \sin\gamma \\ 0 & 1 & 0 \\ -\sin\gamma & 0 & \cos\gamma \end{bmatrix}, R_z = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

Note that if this rotation matrix based on openCV library, it will be like this.

$$R = \begin{bmatrix} \cos\gamma\cos\alpha & -\cos\beta\sin\alpha + \sin\beta\sin\gamma\cos\alpha & \sin\beta\sin\alpha + \cos\beta\sin\gamma\cos\alpha \\ \cos\gamma\sin\alpha & \cos\beta\cos\alpha + \sin\beta\sin\gamma\sin\alpha & -\sin\beta\cos\alpha + \cos\beta\sin\gamma\sin\alpha \\ -\sin\gamma & \sin\beta\cos\gamma & \cos\beta\cos\gamma \end{bmatrix} \tag{9}$$

Where,

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & \sin\beta \\ 0 & -\sin\beta & \cos\beta \end{bmatrix}, R_y = \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix}, R_z = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

The angular orientation can be specified by three angles: Roll (α), Yaw (γ) and Pitch (β) are the rotation around the Zc-Yc-Xc axis respectively. If we set an object and the camera coordinate as Figure 25, rotating object coordinate (Zp-Yp-Xp axis) to each camera coordinate (Zc-Yc-Xc axis) can be known from the rotation angles which were acquired from the orientation sensor attached on the camera.

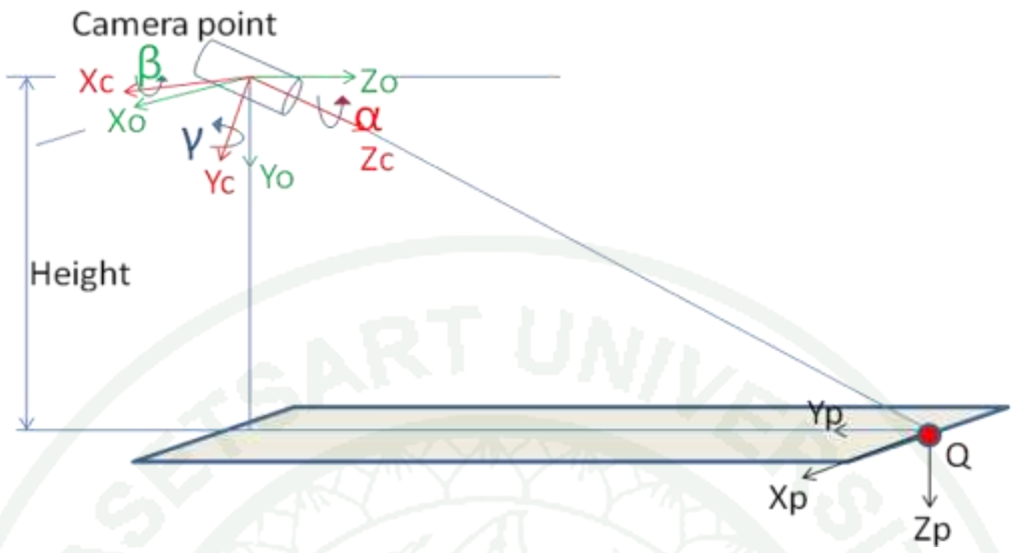


Figure 25 Relation of object and camera coordinate: Roll (α), Yaw (γ) and Pitch (β) are the rotation around the Zc-Yc-Xc axis respectively.

In contrast, we can find each rotational angle (roll, pitch, yaw) by finding from the element of rotation matrix R in eq.(6). Note that this calculation based on equation (7)

$$roll(\alpha) = -\tan^{-1}\left(\frac{r_{21}}{r_{11}}\right) \qquad \mathrm{pitch}(\beta) = -\tan^{-1}\left(\frac{r_{32}}{r_{33}}\right) \qquad \mathrm{yaw}(\gamma) = \sin^{-1}(r_{31}) \qquad (11)$$

Here is finding rotational angel based on openCV as in equation (9)

$$roll(\alpha) = \tan^{-1}\left(\frac{r_{21}}{r_{11}}\right) \qquad \mathrm{pitch}(\beta) = \tan^{-1}\left(\frac{r_{32}}{r_{33}}\right) \qquad \mathrm{yaw}(\gamma) = -\sin^{-1}(r_{31}) \qquad (12)$$

In this research, Yaw angle's rotation does not affect the size of the area after warping to top view as shown in Figure 26, and then the rotation of camera around the y-axis (R_Y) can be neglected.

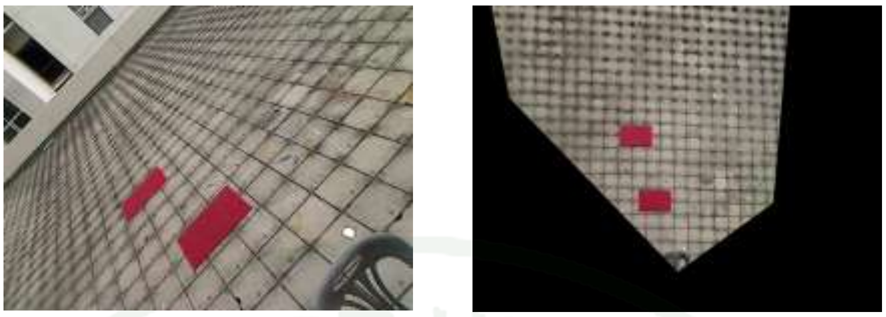

(a) perspective view (b) top view

Figure 26 Pitch and Roll angle are 30^{o}and 48^o.

The rotation of camera is first around the z-axis (roll), then around the new position of the x-axis (pitch). From Equation 7, it can be reduced to Equation 13.

$$R = R_z R_x = \begin{bmatrix} \cos\alpha & -\cos\beta\sin\alpha & \sin\beta\sin\alpha \\ \sin\alpha & \cos\beta\cos\alpha & -\sin\beta\cos\alpha \\ 0 & \sin\beta & \cos\beta \end{bmatrix} \qquad (13)$$

From Equation 9, it can be reduced to Equation 14.

$$R = R_z R_x = \begin{bmatrix} \cos\alpha & \cos\beta\sin\alpha & \sin\beta\sin\alpha \\ -\sin\alpha & \cos\beta\cos\alpha & \sin\beta\cos\alpha \\ 0 & -\sin\beta & \cos\beta \end{bmatrix} \qquad (14)$$

The rotation angles from orientation sensor information, roll and pitch in degree unit, are offset to camera's rotation, α and β, as Equation 15 and Equation 16 that the direction of pitch from sensor is opposite the direction from camera.

$$\alpha = \text{roll} \qquad (15)$$

$$\beta = \text{pitch-90} \qquad (16)$$

A translation vector (T) which each row is a translation axis of X, Y and Z respectively as Equation 17. However, changing the roll angle affects some parameters in translation vector (T), so it makes the output image unorganized. We will discuss the solution further after warping image.

$$T = \begin{bmatrix} t_1 & t_2 & t_3 \end{bmatrix}^t \tag{17}$$

In this research, we will use cvCalibrateCamera2() to find the intrinsic matrix, and cvFindExtrinsicCameraParams2() to find extrinsic matrix.

2.2.2 Undistortion Process

We describe the two main lens distortions and how to model them. Radial distortions arise as a result of the shape of lens, whereas tangential distortions arise from the assembly process of the camera as a whole. For radial distortions, the distortion is zero at the center of the imager and increases as we move toward the periphery which can be expressed as Equation 18 and Equation 19. For tangential distortions, it is due to manufacturing defects resulting from the lens not being exactly parallel to the imaging plane. Tangential distortion is modeled as Equation 20 and Equation 21.

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{18}$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{19}$$

$$x_{corrected} = x + \left[2p_1 xy + p_2 (r^2 + 2x^2) \right] \tag{20}$$

$$y_{corrected} = y + \left[2p_1 xy + p_2 (r^2 + 2y^2) \right] \tag{21}$$

Figure 27a shows the input image and Figure 28b shows the after undistortion process image. For OpenCV, these five distortion matrix: $D = [k_1\, k_2\, p_1\, p_2\, k_3]$ is defined.

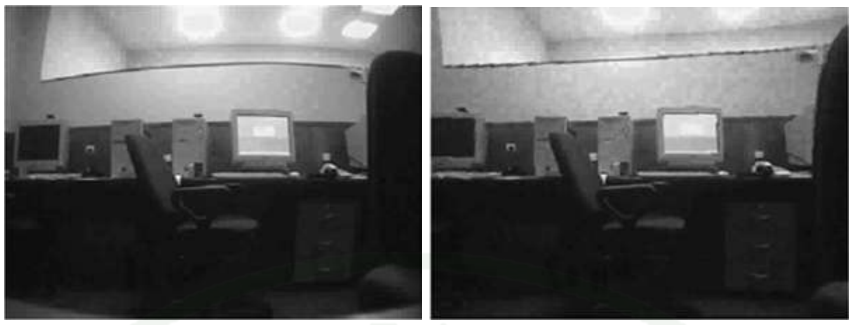


Figure 27 Undistortion Process: (a) Input image (b) Undistort image.

Source: Gary and Adrian (2008)

2.2.3 Chessboard pattern

The calibration object is used in OpenCV which is a flat grid of alternating black and white squares that is usually called a chessboard as Figure 28.

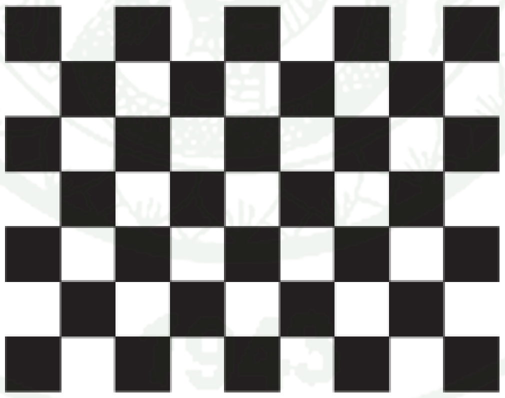


Figure 28 Chessboard pattern.

The corners of black and white squares that width and height are 8 and 6 points is found by using cvFindChessboardCorners() to locate the corners of the chessboard. Result of cvDrawChessboardCorners() can project where these corners were found (small circles on corners) and in what order they belong (as indicated by the lines between circles). Chessboard coordinate are illustrated as Figure 29.

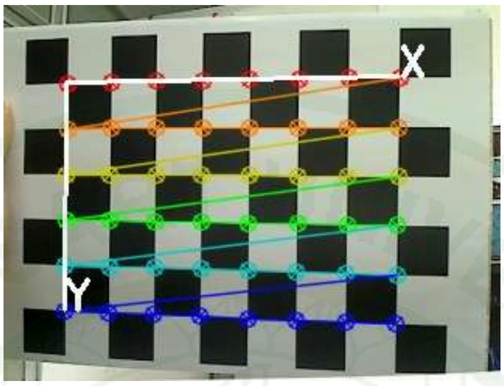


Figure 29 The result of cvFindChessboardCorners(), cvDrawChessboardCorners() and chessboard coordinate.

2.3 Orientation Sensor

The orientation sensor (9DOF Razor IMU as Figure 30) incorporates four sensors which can become an Attitude and Heading Reference System (AHRS) that composes of

• LY530AL (single-axis gyro for Yaw angle) and LPR530AL (dual-axis gyro for Pitch and Roll angle) measure the angular velocity in three dimension.

• ADXL345 (triple-axis accelerometer) measures the acceleration.

• HMC5843 (triple-axis magnetometer) is tilt compensated in Yaw axis.

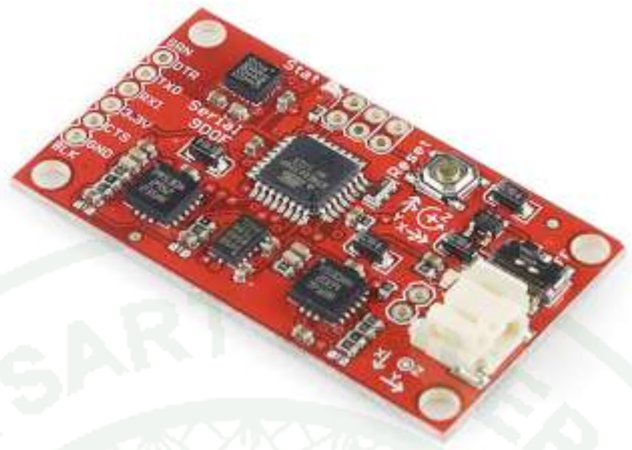


Figure 30 Orientation sensor (9DOF**)** incorporates four sensors which can become an Attitude and Heading Reference System (AHRS)**.**

The 9DOF board is programmed by using Razor 9DOF IMU AHRS V1.1 Lite version for AT328/AT168 as Appendix A for ATmega328 with DCM algorithm on the 8MHz Arduino bootloader by Arduino IDE. The outputs of all sensors are processed by an on-board ATmega328. Outputs of all sensors processed by on-board ATmega328 and sent out via a serial stream which is set COM port and bit rate to 57600 bps.

The pattern of output serial stream show as !ANG:roll,pitch,yaw. For example, !ANG:35.61,10.13,-70.48 mean roll=35.61, pitch 10.13 and yaw -70.48 in degree unit. Interface with Vpython is the python programming language plus a 3D graphics module called "Visual" originated by Scherer, 2000. Vpython makes it easy to create navigable 3D displays and animations, even for those with limited programming experience. Because it is based on python, it also has much to offer for experienced programmers and researchers. In this research, Vpython code as Appendix B is used that output is illustrated as Figure 31.

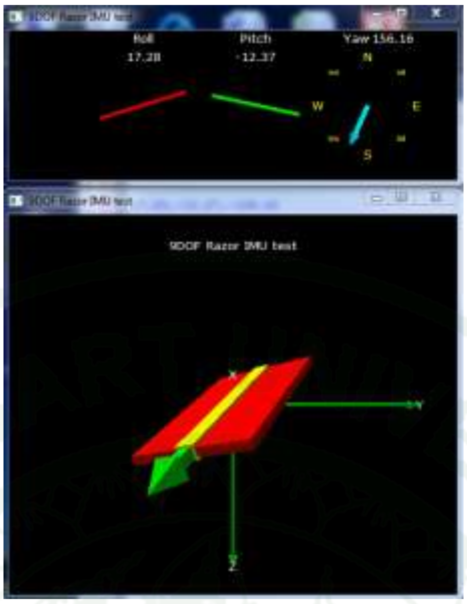


Figure 31 GUI with Vpython interface for calibrate sensor**.**

2.4 Camera and Orientation sensor Calibration

The orientation sensor is attached to a high-resolution web camera or web cam as Figure 32 and Figure 33. A web cam is a video camera which feeds its images in real time to a computer or computer network, often via USB, Ethernet or Wi-Fi. Its common use as a video camera for the World Wide Web gave the webcam its name. Other popular usages include security surveillance and computer vision. The coordinate of camera and orientation sensor must be calibrated together. The three outputs which consist of roll (-180 to 180 degree), pitch (-90 to 90 degree) and yaw (-180 to 180 degree) from orientation sensor will be transformed to camera's rotation for warping to top view algorithm.

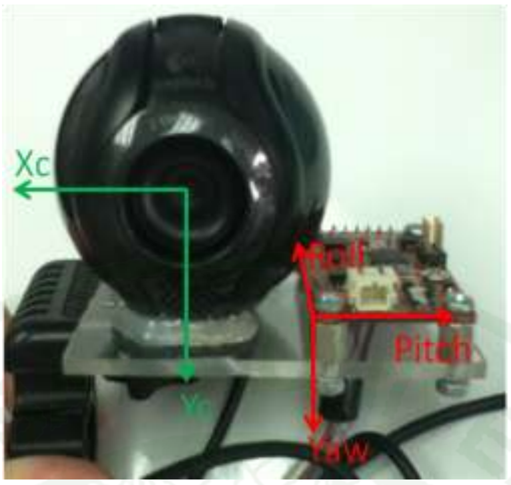


Figure 32 The coordinate of an orientation sensor that is attached to a web camera in front view.

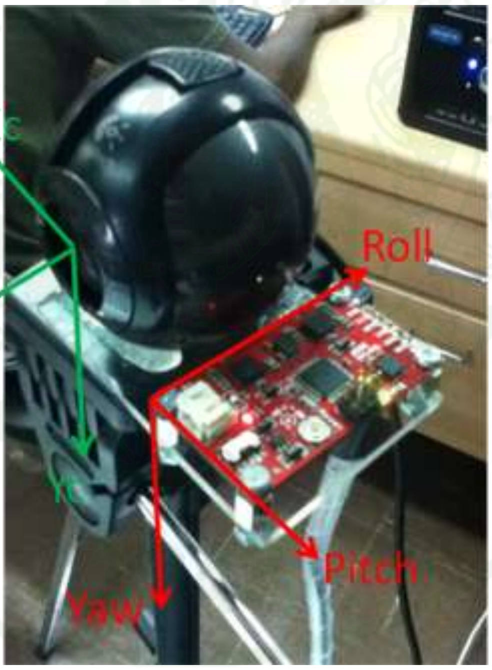


Figure 33 The coordinate of an orientation sensor that is attached to a web camera in side view.

However, the problem of this calculation causes from an imperfect alignment between the camera and the sensor. First problem is about the position alignment. Coordinates of x, y z, from camera must be parallel with coordinates x, y, z from sensor. Iterative Least Square (ILS) method [11] is used to solve this problem. Second problem is about aligning the camera and the orientation sensor data in time. The computer system adds unknown latencies to the data when acquiring an image or reading sensor data [12]. We need to identify how to reduce this different time between camera data and sensor.

2.4.1. Iterative Least Square (ILS)

The least square method is a very popular technique which is used to compute estimations of parameters and to fit a function to a set of data. It exists with several variations: Its simpler version is called ordinary least squares (OLS), a more sophisticated version is called weighted least squares (WLS), which often performs better than OLS because it can modulate the importance of each observation in the final solution. Recent variations of the least square method are alternating least squares (ALS) and partial least squares (PLS).

Our experiment use of OLS as it is the oldest (and still most frequent). It was a linear regression. It often happens that $Ax = b$ has no solution. The usual reason is: *too many equations* (m is greater than n). Unless all measurements are perfect, b is outside that column space. We cannot always get the error $e = b\text{-}Ax$ down to zero When e is zero, x is an exact solution to $Ax = b$. When the length of e is as small as possible, $\hat{x}$ is a least squares solution. Our goal in this section is to compute $\hat{x}$ and use it.

When $Ax = b$ has no solution, multiply by A^T and solve $A^TA\hat{x} = A^Tb$. This seems it is like the pseudo inverse. However, the generalized inverse matrix requires the determination non-zero. Thus, our method will use matrix as it is the pseudo inverse.

To compensate for imperfect alignment, from Equation 1, rotation matrix (R) is acquired from the sensor coordinate must be transferred to that of the camera coordinate using a correction matrix (R_{offset}). In our experiment, we will correct the error of R_x as the pitch affected the area size. In our calibration, the world coordinates

(Q) and the imager coordinate (q) are known. The intrinsic matrix (M) and translation vector (T) are pre-defined. In Equation 22, we need to find Rx_{offset} for correcting R_{sensor} to R_{camera}. Iterative Least Square (ILS) technique [8] is applied to find Rx_{offset} as follow.

$$Rx_{camera} = Rx_{offset} \times Rx_{sensor} \tag{22}$$

Equation 22 can be represented as Equation 23 which consists of 9 unknown parameters. We rearrange Equation 23 as Equation 24.

$$\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \times \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} . \tag{23}$$

$$\begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \\ b_{21} \\ b_{22} \\ b_{23} \\ b_{31} \\ b_{32} \\ b_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} & a_{31} & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{12} & a_{22} & a_{32} & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{13} & a_{23} & a_{33} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{11} & a_{21} & a_{31} & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{12} & a_{22} & a_{32} & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{13} & a_{23} & a_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{11} & a_{21} & a_{31} \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{12} & a_{22} & a_{32} \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{13} & a_{23} & a_{33} \end{bmatrix} \times \begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \\ r_{21} \\ r_{22} \\ r_{23} \\ r_{31} \\ r_{32} \\ r_{33} \end{bmatrix} \tag{24}$$

Alternatively Equation 25 can be written as Equation 26, where i is the ith sample in the data collection process.

$$[M_i]_{9\times1} = [N_i]_{9\times9}[O]_{9\times1} . \tag{25}$$

To estimate 9 parameters in O, it can be determined in a least-squares sense.

$$[O]_{9\times1} = ([N_i^T]_{9\times9}[N_i]_{9\times9})^{-1}([N_i^T]_{9\times9}[M_i]_{9\times1}) . \tag{26}$$

The large number of sample significantly increases the system accuracy. Thus, to update Equation 26 after every control point is collected, Iterative Least Squares method [8] is applied as in Equation 27.

$$[O]_{9\times1} = \sum_i ([N_i^T]_{9\times9}[N_i]_{9\times9})^{-1} \sum_i ([N_i^T]_{9\times9}[M_i]_{9\times1}) \; . \quad (27)$$

After O is calculated, it means the imperfect alignment can be corrected by getting O rearranged in the form of Rx_{offset} before multiply Rx_{offset} back into Equation 22. And then, apply it in homography matrix for warping image more accurate.

2.4.2. Time alignment

As programming codes run in sequential, we have to identify the elapsed time between reading data from sensor and capturing an image from camera as ideally, it should be acquired the data from both sensors at the same time. In this research, every time of 100 ms, Timer Ticker will be called to start warping process as in Figure 25. During Finding Camera Interface process, it takes long interval time to select the camera, so this process is not considerate to align the time shift. Therefore. We should start next process, Reading Angle process when the Capturing image process finishes. This is due to more different duration time causing more errors in querying data, so we need to capturing data from camera before reading data from sensor. It seems to be making these two processes running at the same time.

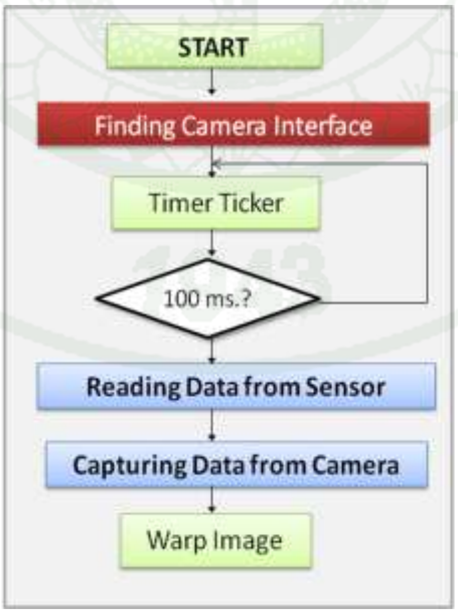


Figure 34 The coordinate of an orientation sensor that is attached to a web camera in side view.

After we adjust the time alignment while acquiring data between sensor and camera, it seems that we reduce this time delay to reading the data that make warping image algorithm more accurate.

`

3. Processing Step

3.1 Homography matrix

When the object is a plane, a simpler formulation becomes available. Since the world coordinate system can be set anywhere, it can be conveniently positioned on the plane, such that latter has zero Z coordinate. From Equation 5, we consider Z is zero which can written as Equation 28.

$$sq = HQ \tag{28}$$

Or,

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Where,

$Q = [X, Y, 1]^t$ is top view output pixels.

$q = [x, y, 1]^t$ is perspective view input pixels.

s is Scale factor.

H is 3x3 Homography matrix which can written as Equation 29.

$$H = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix} \tag{29}$$

3.2 Warping to top view

3.2.1. Using chessboard

cvFindExtrinsicCameraParams2() function returns a rotation matrix (R) derived from Rz, Ry, Rx multiplied together as in Equation 6. This function also return translation vector, but in Equation 29, we select the first two columns as it is a planar surface. Thus, based on Equation 7 from openCV library, the homography matrix (H) using chessboard are shown in Equation 30

$$H = \begin{bmatrix} f_x \cos\alpha \cos\gamma - cx \sin\gamma & f_x \sin\alpha \cos\beta - f_x \cos\beta \sin\alpha + c_x \sin\beta \cos\gamma & f_x t_1 + c_x t_3 \\ f_y \cos\gamma \sin\alpha + c_y \cos\beta \cos\gamma & f_y \cos\alpha \cos\beta + f_y \sin\beta \sin\gamma \sin\alpha + c_y \sin\beta \cos\gamma & f_y t_2 + c_y t_3 \\ -\sin\gamma & \sin\beta \cos\gamma & t_3 \end{bmatrix} \quad (30)$$

Note that this matrix comes from 3 angels (Rz,Ry,Rx) but our experiment uses only 2 angel (pitch, roll). Thus, we will find each angel by using Equation 12 and derive new homography matrix (H) that multiplied 2 rotation matrixes (Rz, Rx).

$$H = \begin{bmatrix} f_x \cos\alpha & -c_x \sin\beta + f_x \sin\alpha \cos\beta & f_x t_1 + c_x t_3 \\ -f_y \sin\alpha & c_y \sin\beta + f_y \cos\alpha \cos\beta & f_y t_2 + c_y t_3 \\ 0 & -\sin\beta & t_3 \end{bmatrix} \quad (31)$$

This parameter H will be putted it into cvWarpPerspective() function to do warping image. In the other hand, openCV provide a library that can calculate this value by using cvFindHomography().

3.2.2. Using orientation sensor in real-time

Rotation matrix (R) in equation (7) will receive Roll (α), Pitch (β), and Yaw (γ) angels from orientation sensor, then the homography will be calculated as following.

$$H = \begin{bmatrix} f_x \cos\alpha \cos\gamma + cx \sin\gamma & f_x \sin\alpha \cos\beta + f_x \cos\beta \sin\alpha - c_x \sin\beta \cos\gamma & f_x t_1 + c_x t_3 \\ -f_y \cos\gamma \sin\alpha + c_y \cos\beta \cos\gamma & f_y \cos\alpha \cos\beta - f_y \sin\beta \sin\gamma \sin\alpha - c_y \sin\beta \cos\gamma & f_y t_2 + c_y t_3 \\ \sin\gamma & -\sin\beta \cos\gamma & t_3 \end{bmatrix} \quad (32)$$

The problem of this homorgraphy matrix is about translation vector. We have to set this value beforehand.

As pitch angel affected the area size, and roll angel affected the current view, we will receive these two angels from sensor and calculate the rotation matrix which uses only Rz, Rx as in Equation 14. From Equation 17, translation parameters (t_1, t_2, t_3) are set to (0, 0, f_x /2). Finally, we will get homography matrix (H) from the sensor as in Equation 33.

$$H = \begin{bmatrix} f_x \cos\alpha & c_x \sin\beta - f_x \sin\alpha \cos\beta & 0.5 c_x f_x \\ f_y \sin\alpha & c_y \sin\beta + f_y \cos\alpha \cos\beta & 0.5 c_y f_x \\ 0 & \sin\beta & 0.5 f_x \end{bmatrix} \quad (33)$$

In warping to top view, the output is top view output pixels (Q) and the input is perspective input pixels. Hence, homography matrix is inverted to multiply with input pixels (q). From Equation 31 can be written as Equation 34 (assume that $A = H^{-1}$).

$$sQ = H^{-1} q = Aq \quad (34)$$

In matrix form,

$$s \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Or,

$$sX = a_{11}x + a_{12}y + a_{13}$$
$$sY = a_{21}x + a_{22}y + a_{23}$$
$$s = a_{31}x + a_{32}y + a_{33}$$

So that,

$$X = \frac{a_{11}x + a_{12}y + a_{13}}{a_{31}x + a_{32}y + a_{33}}$$
$$Y = \frac{a_{21}x + a_{22}y + a_{23}}{a_{31}x + a_{32}y + a_{33}}$$

From Equation 34, we can derive a_{ij} as

$$a_{11} = \frac{\cos(\alpha)}{f_x}$$
$$a_{12} = \frac{\cos(\alpha)}{f_y}$$
$$a_{13} = \frac{(c_y f_x \sin(\alpha) + c_x f_y \cos(\alpha))}{f_x f_y}$$
$$a_{21} = \frac{-\sin(\alpha)}{f_x \cos(\beta)}$$
$$a_{22} = \frac{\cos(\alpha)}{f_y \cos(\beta)}$$
$$a_{23} = \frac{(c_x f_y \sin(\alpha) - c_y f_x \cos(\alpha))}{f_x f_y \cos(\beta)}$$
$$a_{31} = \frac{-2\sin(\alpha)\sin(\beta)}{f_x^{\,2} \cos(\beta)}$$
$$a_{32} = \frac{2\cos(\alpha)\sin(\beta)}{f_x f_y \cos(\beta)}$$
$$a_{33} = \frac{-2}{f_x}\left[\tan(\beta)\left(\frac{c_y \cos(\alpha)}{f_y} - \frac{c_x \sin(\alpha)}{f_x}\right) + 1\right]$$

OpenCV provide a function that calculate top view image pixel (Q) for us automatically, cvWarpPerspective(), by inputting homography matrix (H) and original image. However, cvWarpPerspective() does not require inverse homography matrix as an input. Therefore, we will find a new homography matrix which will be discussed in the next topic.

3.3 Setting translation vector for the warped image to be viewable

cvWarpPerspective() can generate real time top view image using rotation parameters from the orientation sensor information. For translation parameters (t_1, t_2, t_3), are set to (0, 0, f_x /2). Setting t_3, zoom, to f_x /2 makes the top view image covers as many pixels in its original image as possible. This is preferable as we are going to apply the top view image for area calculation. However, the top view image is not a center of image as Figure 35.

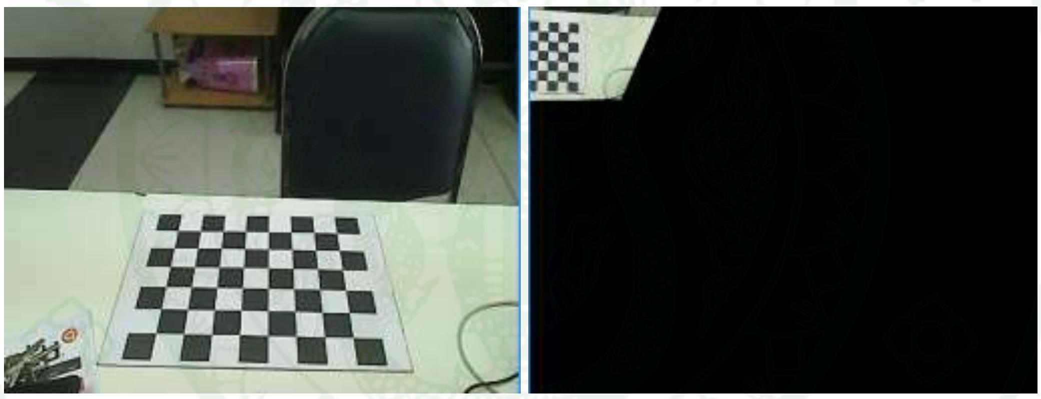

(a) (b)

Figure 35 Warping to top view from (a) input image to (b) the top view output image by translation parameters (t_1, t_2, t_3), are set to (0, 0, f_x /2).

In order to make a top view image to the center, for translation parameters (t_1, t_2, t_3), should be set to (c_x, c_y, f_x /2). For t_1 and t_2, shift, normally it is used to shift the warped image to be viewable. However, doing so introduces distorted perspective when roll angle changes as Figure 36.

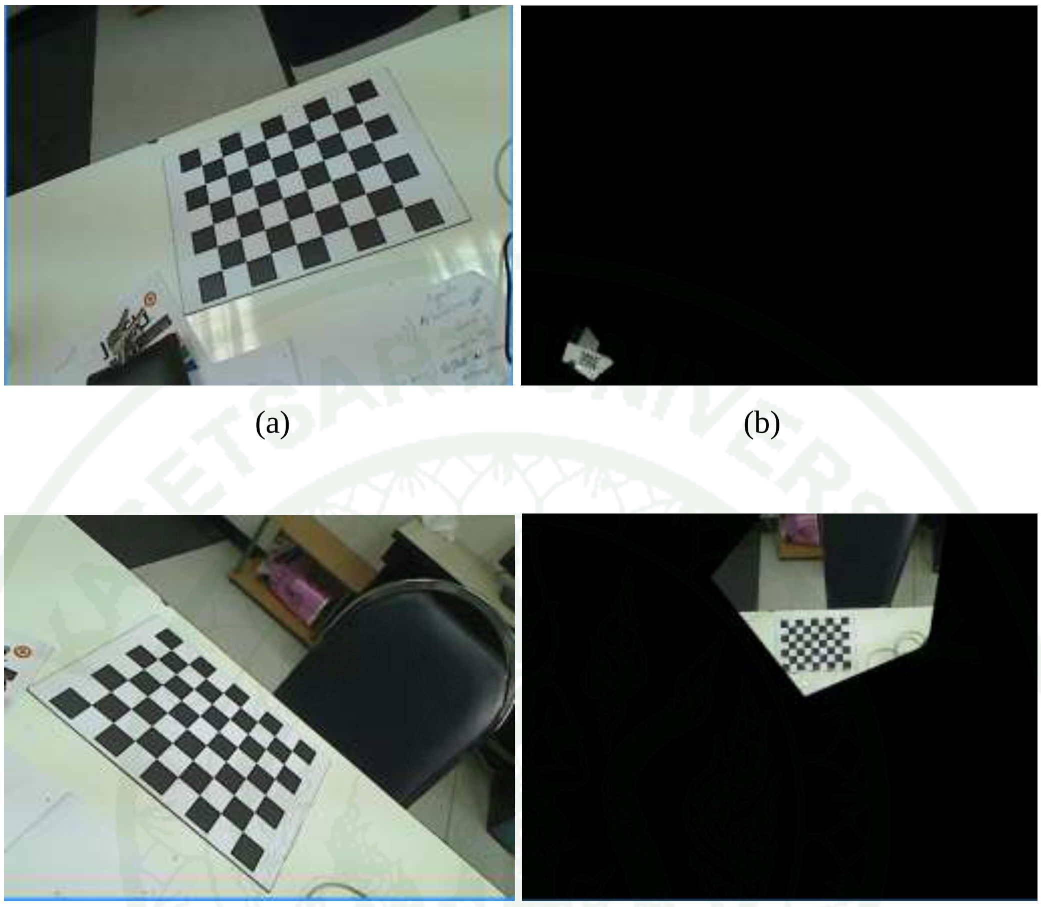


Figure 36 Warping to top view from (a) input image in right tilt to (b) the top view output image and (c) input image in left tilt to (d) the top view output image by translation parameters (t_1, t_2, t_3), are set to (c_x, c_y, f_x /2).

To remove the roll distortion effect, translation parameters (t_1, t_2, t_3), are set to (0, 0, f_x /2). Then, the warped image is shifted to be viewable later. Hence, shift in x and y direction will be performed separately from the homography, for the value of c_x, c_y, respectively as Equation 35.

$$Q = SH^{-1}q \tag{35}$$

Where,

$S = \begin{bmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{bmatrix}$ is a shift matrix and it can be inverted as $S^{-1} = \begin{bmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{bmatrix}$.

But cvWarpperspective() function in OpenCV library need homography matrix as an input. Hence, the shift matrix should be adjusted by using multiplication of invertible matrix properties. Now, the new homography matrix with shift is shown as Equation 36.

$$\left(SH^{-1}\right)^{-1} = HS^{-1} \tag{36}$$

After using Equation 36 as a comprehensive homography matrix, it can remove the roll distortion effect and the warped image is set to be viewable as Figure 37.

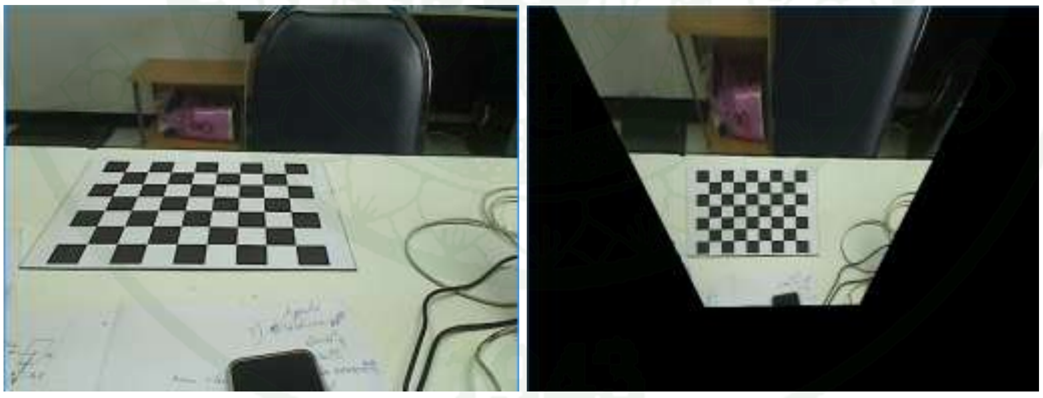


Figure 37 Warping to top view from (a) input image to (b) the top view output image by translation parameters (t_1, t_2, t_3), are set to (0, 0, f_x /2) and shift in x and y direction after warping.

4. Post-Processing Step

4.1 The RGB color model

After image warping to top view, an image processing called thresholding is required to counting feature color pixels. After that, mapping pixels area to real world unit is proposed.

From Wikipedia, “The RGB color model as Figure 28 is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The main purpose of the RGB color model is for the sensing, representation, and display of images in electronic systems, such as televisions and computers, though it has also been used in conventional photography. Before the electronic age, the RGB color model already had a solid theory behind it, based in human perception of colors.”

“Every image is a set of pixels and each pixel represents a color. In the RGB model, the color of each pixel is stored in terms of three components: Red, Green, and Blue. By adjusting the values of the R, G, and B components, a wide variety of colors can be generated.” Hence, if we need only red pixel the color threshold can be used.

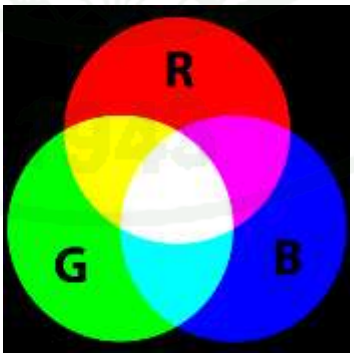


Figure 38 RGB color model.

Source: Wikipedia

4.2 Red color pixels counting scheme

In this research, we verify the system accuracy by estimating an area of red sheets with known size. Figure 29 is shows the red sheet, 70cm x 46.2cm.

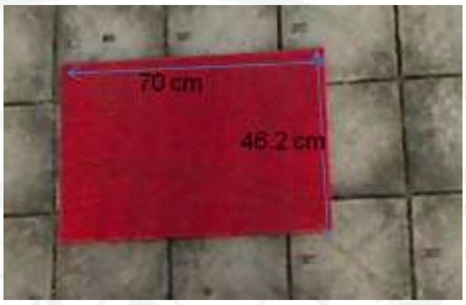
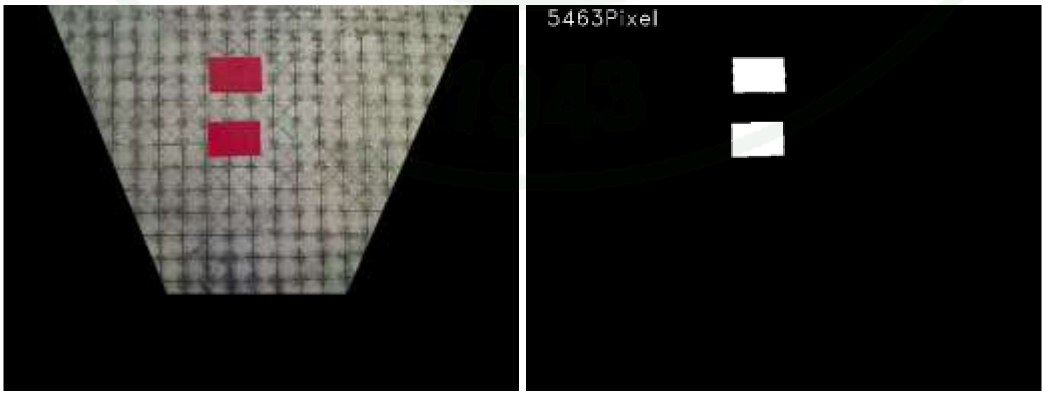


Figure 39 A red sheet, 70cm x 46.2cm, as target surface area.

After warping to top view as Figure 40(a), counting color pixel is performed. The scheme is shown in Figure 41. First, initialize count parameter to zero and read next image pixel. Second, check if all pixels are counted or not. If not, then, color threshold is performed on the present pixel, to verify if it is the target area. In the threshold process, count parameter is increased and the pixel is marked white if it is the target area. Otherwise, mark it as black. This is just to generate an output that is easy to view, as shown in Figure 40(b).



(a) (b)

Figure 40 After warping to (a) top view, (b) counting red color pixel is used.

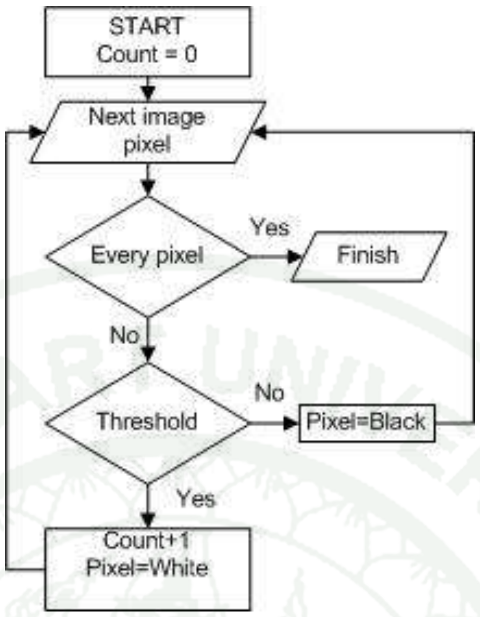


Figure 41 Counting target color pixel schemes.

4.3 Mapping equation

After counting color pixel, we found from experimental result that the area size in pixel unit and the pitch angle has linear relationship approximately. In other words, the target area would get larger in the image when we rotate the camera to higher pitch value.

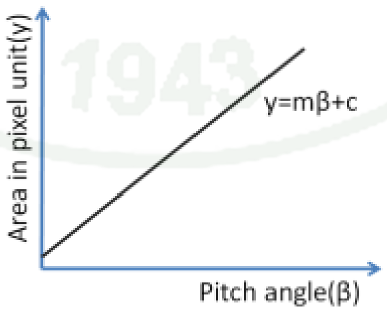


Figure 42 Linear graph which can be written in linear equation shows relation between pitch angle and area in pixel unit.

Finally, the number of target pixels is mapped to area size in real world unit by a pre-calibrated equation. Such warping equation is derived by Equation 37. The calibration requires a known area (T) and its corresponding number of pixels (y). By rule of three, unknown area (Area) can be estimated from its corresponding number of pixels (P).

$$\frac{Area(cm^2)}{T(cm^2)} = \frac{P(pixel)}{y(pixel)} \qquad (37)$$

Remember from Figure 40 that depends on pitch angle (β). Hence Equation 37can be replaced by Equation 38. Where, gradient (m) and intercept (c) of linear equation can be known from a pre-calibrated equation.

$$Area(cm^2) = \frac{P(pixel) \times T(cm^2)}{m\beta + c} \qquad (38)$$

4.4 Percentage Accuracy

We determine our system by using these equations.

$$\%Error = \frac{\left|Area(cm^2) - KnownArea(cm^2)\right|}{KnownArea(cm^2)} \times 100\% \qquad (39)$$

$$\%Accuracy = 100 - \%Error \qquad (40)$$

4.5 Scaling Factor of Height

To perform this algorithm practically, height from camera to the ground should be able to be varied. In our algorithm, height (h) is inversely proportional to the area size in pixel unit (*P*). In other words, as the height increases, area size in pixel unit will decrease. If the system increases *n* times of the height used to calibrate Equation 40, area size should be decreased by n*n. Therefore, we will scale this area by multiplying the result of pixel unit with n*n, resulting in Equation 41.

$$Area(cm^2) = \frac{(n^2) \times P(pixel) \times T(cm^2)}{m\beta + c} \tag{41}$$

RESULTS AND DISCUSSION

Results

Our experiments consist of three phases. Pre-processing step consists of camera calibration, undistortion. Processing step consists of warping to top view using chessboard (to test concept) and using orientation sensor in real-time. Post-processing step consists of comparing the warped image with the ground truth and mapping factor from pixel to actual area size. Our method was implemented in C programming language using OpenCV library. The technique requires calculation of a homography matrix, which consists of three parameters: intrinsic matrix, translation vector, and rotation matrix. Rotation matrix is uncontrolled in real time. The parameters are in Table 1.

Table 1 Experimental Parameter.

Parameters	Values
Pitch angle	20-80 degree
Intrinsic parameters	f_x =1038.71, f_y =1037.18 c_x = 471.87, c_y =361.42
Distortion parameters	k_1=-0.243, k_2=0.14, k_3=0 p_1=-0.0031 p_2= -0.004836
Translation vector	t_1=0, t_2=0, t_3= f_x /2
Image dimensions	640 x 480
Bitmap size	900 KB
Target area	46.2 x 70 cm^2
A known area	9312 cm^2

In this experiment, the camera is calibrated offline. The relationship between camera and orientation sensor is also derived offline whereas the orientation sensor is used to provide rotation angle in real time. Then, homography matrix is formed based on the intrinsic and extrinsic information. The results illustrate the relationship between pitch angle's rotation and size of an area after the top view image was warped. In the experimental setup, the pitch angle was manually varied from 20 degree to 80 degree

1. Intrinsic Parameters

Intrinsic matrix composes of 4 parameters (fx,fy,cx,cy) that are found from calibrating camera with using OpenCV library. For finding these parameters, we will follow these steps.

First, we will find the chessboard corner that this function checks all corners are completely found or not.

```
//Find chessboard corners:
       int found = cvFindChessboardCorners(
                image, board_sz, corners, &corner_count,
                CV_CALIB_CB_ADAPTIVE_THRESH |
CV_CALIB_CB_FILTER_QUADS );
```

After all corners are found, we will convert the image to gray image to find the sub pixel to get more accurate in corner.

```
//Get Subpixel accuracy on those corners
       cvCvtColor(image, gray_image, CV_BGR2GRAY);
       cvFindCornerSubPix(gray_image, corners, corner_count,
                  cvSize(11,11),cvSize(-1,-1), cvTermCriteria(
                  CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1 ));
```

Then, This function draws the chessboard corners.

```
//Draw it
       cvDrawChessboardCorners(image, board_sz, corners,
                  corner_count, found);
```

Next, we will find the image point and object point by using this for loop. If we cannot find all corners from the image, we will set that picture to be gray.

```
// If we got a good board, add it to our data
       if( corner_count == board_n ) {
          cvShowImage( "Calibration", image ); //show in color if
we did collect the image
          step = successes*board_n;
          for( int i=step, j=0; j<board_n; ++i,++j ) {
             CV_MAT_ELEM(*image_points, float,i,0) = corners[j].x;
             CV_MAT_ELEM(*image_points, float,i,1) = corners[j].y;
             CV_MAT_ELEM(*object_points,float,i,0) = j%board_w;
             CV_MAT_ELEM(*object_points,float,i,1) = j/board_w;
             CV_MAT_ELEM(*object_points,float,i,2) = 0.0f;
          }
          CV_MAT_ELEM(*point_counts, int,successes,0) = board_n;
          successes++;
          printf("Collected our %d of %d needed chessboard
images\n",successes,n_boards);

       }
       else{
             cvShowImage( "Calibration", gray_image );} //Show
Gray if we didn't collect the image
    }
```

Finally, finds the camera intrinsic which the input is object_points, image_points, and point_counts.

```
//CALIBRATE THE CAMERA!
  cvCalibrateCamera2(
      object_points, image_points,
      point_counts,  cvGetSize( image ),
      intrinsic_matrix, distortion_coeffs, rotation_vectors,
translation_vectors,0);  //CV_CALIB_FIX_ASPECT_RATIO
```

This program will be identifying the corners on a chessboard pattern by using cvCalibrateCamera2() function. This funxtion estimates the calibration parameters and extract black-white-black-white features 48 points as shown in Figure 33 (width=8, height=6). Then, our program identifies how many numbers of different views that is suitable for using those intrinsic values. The result shows in Table3

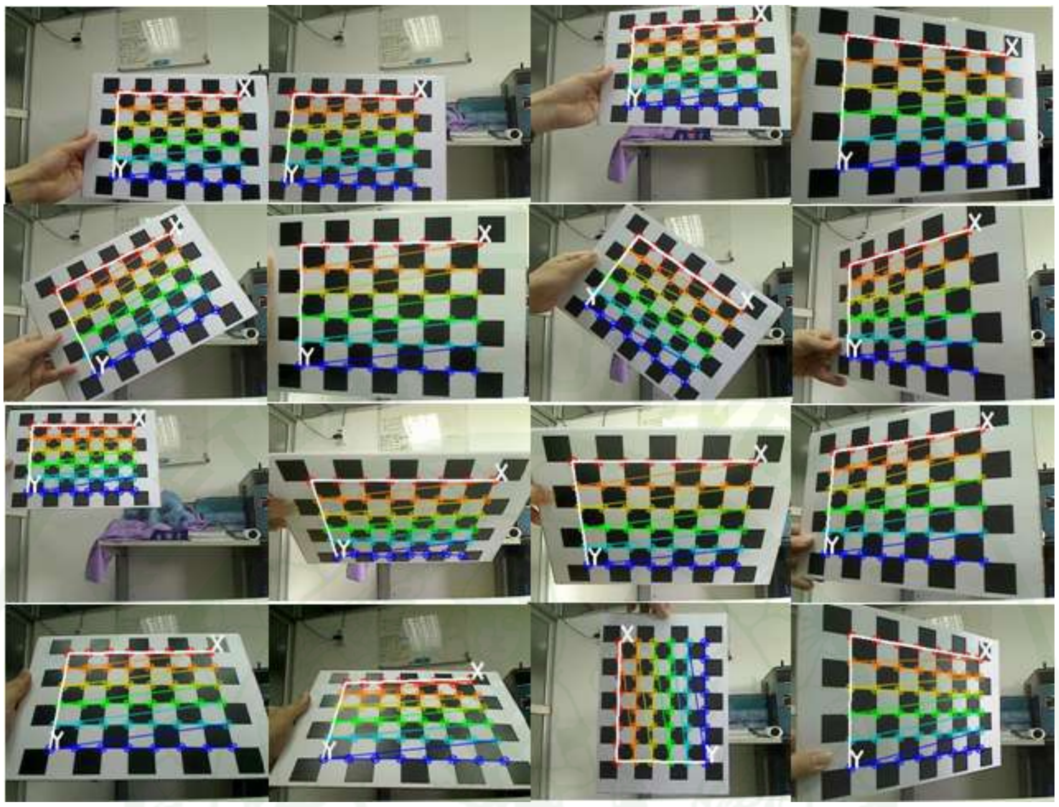


Figure 43 Extract black-white-black-white features 48 points on chessboard using cv calibrateCamera2() function at difference scene.

Outputs of camera calibration are saved in Extensible Markup Language (XML) file, which is a set of rules for encoding documents in machine-readable form. After that, read the saved file to acquire all of parameters. Here, we show the result of cvCalibratecamera2() function.

```
<?xml version="1.0"?>
-<opencv_storage>
        -<Intrinsics type_id="opencv-matrix">
                <rows>3</rows>
                <cols>3</cols>
                <dt>f</dt>
                <data> 5.35105896e+002 0. 3.12345947e+002 0. 5.33465515e+002
        2.38924026e+002 0. 0. 1.</data>
        </Intrinsics2>
</opencv_storage>
```

```
<?xml version="1.0"?>
-<opencv_storage>
        -<Distortion type_id="opencv-matrix">
                <rows>4</rows>
                <cols>1</cols>
                <dt>f</dt>
               <data> 5.15903160e-002 -4.88530621e-002 2.15528603e-003
        9.97108873e-004</data>
        </Distortion2>
</opencv_storage>
```

2. Extrinsic Parameters

As we mentioned in the previous chapter, extrinsic parameters consists of rotational matrix and translation vector. This section shows the result of these values using chessboard and using orientation sensor in the real time.

2.1 Rotation Matrix

2.1.1 Using chessboard

After we got the object point and image point found from previous step, we use those values with intrinsic matrix as it is inputs of the following function, then it return rotation vector, translation vector. Finally, changing the rotation vector into rotation matrix has to be performed.

```
// Calibrate the camera
cvFindExtrinsicCameraParams2(object_points,image_points,intrinsic_
                              matrix,distortion_coeffs,rotation_vec
                              tors,translation_vectors,0);
cvRodrigues2(rotation_vectors,rotation_matrix,0);
```

We reference this code from OpenCV library, but our program uses pitch angle that is affected the area size. Therefore, we have to look into an element of the rotation matrix to extract roll, pitch, yaw angle from this library as in equation (12).

```
xAngle = atan(CV_MAT_ELEM( *rotation_matrix, float, 2, 1 /
              CV_MAT_ELEM( *rotation_matrix, float, 2, 2 ));
yAngle = -asin(CV_MAT_ELEM( *rotation_matrix, float, 2, 0 ));
zAngle = atan(CV_MAT_ELEM( *rotation_matrix, float, 1, 0 )/
              CV_MAT_ELEM( *rotation_matrix, float, 0, 0 ));
```

After we get xAngle, yAngle and zAngle from the sensor, we will put it into Rx, Ry and Rz followed in equation (10). The OpenCV library returned rotation matrix R same as we multiplied each rotation matrix (Rx,Ry,Rz) and then, input these values to compute the homography matrix.

```
CV_MAT_ELEM( *Rx, float, 0, 0 )=1.0;
CV_MAT_ELEM( *Rx, float, 0, 1 )=0.0;
CV_MAT_ELEM( *Rx, float, 0, 2 )=0.0;
CV_MAT_ELEM( *Rx, float, 1, 0 )=0.0;
CV_MAT_ELEM( *Rx, float, 1, 1 )=cos(xAngle);
CV_MAT_ELEM( *Rx, float, 1, 2 )=-sin(xAngle);
CV_MAT_ELEM( *Rx, float, 2, 0 )=0.0;
CV_MAT_ELEM( *Rx, float, 2, 1 )=sin(xAngle);
CV_MAT_ELEM( *Rx, float, 2, 2 )=cos(xAngle);

CV_MAT_ELEM( *Ry, float, 0, 0 )=cos(yAngle);
CV_MAT_ELEM( *Ry, float, 0, 1 )=0.0;
CV_MAT_ELEM( *Ry, float, 0, 2 )=sin(yAngle);
CV_MAT_ELEM( *Ry, float, 1, 0 )=0.0;
CV_MAT_ELEM( *Ry, float, 1, 1 )=1.0;
CV_MAT_ELEM( *Ry, float, 1, 2 )=0.0;
CV_MAT_ELEM( *Ry, float, 2, 0 )=-sin(yAngle);
CV_MAT_ELEM( *Ry, float, 2, 1 )=0.0;
CV_MAT_ELEM( *Ry, float, 2, 2 )=cos(yAngle);

CV_MAT_ELEM( *Rz, float, 0, 0 )=cos(zAngle);
CV_MAT_ELEM( *Rz, float, 0, 1 )=-sin(zAngle);
CV_MAT_ELEM( *Rz, float, 0, 2 )=0.0;
CV_MAT_ELEM( *Rz, float, 1, 0 )=sin(zAngle);
CV_MAT_ELEM( *Rz, float, 1, 1 )=cos(zAngle);
CV_MAT_ELEM( *Rz, float, 1, 2 )=0.0;
CV_MAT_ELEM( *Rz, float, 2, 0 )=0.0;
CV_MAT_ELEM( *Rz, float, 2, 1 )=0.0;
CV_MAT_ELEM( *Rz, float, 2, 2 )=1.0;
```

To be practical, we use orientation sensor to acquire xAngle, yAngle and zAngle

2.1.2 Using orientation sensor in real-time

We receive rotational angle from serial port with baud rate equal to 57600 and set flow control to be X_{on}/X_{off}. The source code for burning this board via Aduino, see on Appendix A.

Next, we can exact the data from serial port by using this following code. It sets the timer reading the data every 100 millisecond.

```
while (this->timer1->Enabled){
      this->m_data = this->serialPort1->ReadLine();
      if(this->m_data->Trim()->StartsWith("#YPR=")){
            this->m_data = this->m_data->Replace("#YPR=",",");
            this->m_getdata = this->m_data->Split(',');
            if (this->m_getdata->Length==4){
                  roll = Convert::ToDouble(this->m_getdata[3]);
                  pitch = Convert::ToDouble(this->m_getdata[2]);
                  yaw = Convert::ToDouble(this->m_getdata[1]);
            }
      }
}
```

Then, we put the reading angle from sensor (roll(z), pitch(x), yaw(y)) in the following code based on Equation 8.

```
CV_MAT_ELEM( *Rx, float, 0, 0 )=1.0;
CV_MAT_ELEM( *Rx, float, 0, 1 )=0.0;
CV_MAT_ELEM( *Rx, float, 0, 2 )=0.0;
CV_MAT_ELEM( *Rx, float, 1, 0 )=0.0;
CV_MAT_ELEM( *Rx, float, 1, 1 )=cos(xAngle);
CV_MAT_ELEM( *Rx, float, 1, 2 )=-sin(xAngle);
CV_MAT_ELEM( *Rx, float, 2, 0 )=0.0;
CV_MAT_ELEM( *Rx, float, 2, 1 )=sin(xAngle);
CV_MAT_ELEM( *Rx, float, 2, 2 )=cos(xAngle);

CV_MAT_ELEM( *Ry, float, 0, 0 )=cos(yAngle);
CV_MAT_ELEM( *Ry, float, 0, 1 )=0.0;
CV_MAT_ELEM( *Ry, float, 0, 2 )=sin(yAngle);
CV_MAT_ELEM( *Ry, float, 1, 0 )=0.0;
CV_MAT_ELEM( *Ry, float, 1, 1 )=1.0;
CV_MAT_ELEM( *Ry, float, 1, 2 )=0.0;
CV_MAT_ELEM( *Ry, float, 2, 0 )=-sin(yAngle);
CV_MAT_ELEM( *Ry, float, 2, 1 )=0.0;
CV_MAT_ELEM( *Ry, float, 2, 2 )=cos(yAngle);

CV_MAT_ELEM( *Rz, float, 0, 0 )=cos(zAngle);
CV_MAT_ELEM( *Rz, float, 0, 1 )=-sin(zAngle);
CV_MAT_ELEM( *Rz, float, 0, 2 )=0.0;
CV_MAT_ELEM( *Rz, float, 1, 0 )=sin(zAngle);
CV_MAT_ELEM( *Rz, float, 1, 1 )=cos(zAngle);
CV_MAT_ELEM( *Rz, float, 1, 2 )=0.0;
CV_MAT_ELEM( *Rz, float, 2, 0 )=0.0;
CV_MAT_ELEM( *Rz, float, 2, 1 )=0.0;
CV_MAT_ELEM( *Rz, float, 2, 2 )=1.0;
```

Finally, multiplying these three matrix together to get the matrix R for performing a warping image further.

2.2 Translation Vector

We will use following inverse matrix that we have mentioned in equation (33). While using orientation sensor, we set translation vector as the initial step (0, 0, 0.5fx). It means that we adjust the size of image view to be viewable beforehand.

```
CV_MAT_ELEM( *T, float, 0, 0 )=1;    // T(inverse)
CV_MAT_ELEM( *T, float, 0, 1 )=0;
CV_MAT_ELEM( *T, float, 0, 2 )=-cx;
CV_MAT_ELEM( *T, float, 1, 0 )=0;
CV_MAT_ELEM( *T, float, 1, 1 )=1;
CV_MAT_ELEM( *T, float, 1, 2 )=-cy;
CV_MAT_ELEM( *T, float, 2, 0 )=0;
CV_MAT_ELEM( *T, float, 2, 1 )=0;
CV_MAT_ELEM( *T, float, 2, 2 )=1;
```

Place the chessboard on the planar surface area as Figure 44(a). Each sub-square of chessboard have width x length as 2.8 cm x 2.8 cm. To generate the top view as Figure 44(b), the algorithm runs as follows.

1. Read the intrinsic and distortion matrix derived earlier in camera calibration process. Then undistort an input image.

2. Find a known object on the ground plane (in this case, a chessboard). Detect corner points at sub pixel accuracy (black-white-black-white feature).

3. Enter the found points, at least four point, into cvFindHomography() function to compute the homography matrix H, that relates such input view and its top view.

4. Use cvWarpPerspective() function with flags CV_INTER_LINEAR + CV_WARP_INVERSE_MAP + CV_WARP_FILL_OUTLIERS to obtain a frontal parallel (top view) view of the ground plane.

The output in XML file of homography matrix by using cvFindHomography() function is as below

H: !!opencv-matrix rows: 3 cols: 3 dt: f

data: [31.82570457, -2.84891891, 234.61930847, 0.34437224, 28.99610901, 162.04414368, 6.99729368e-004, -8.76668375e-003, 18.]

This mean homography matrix is

$$H = \begin{bmatrix} 31.23 & -2.85 & 234.62 \\ 0.34 & 28.99 & 162.04 \\ 0.0007 & -0.0088 & 18 \end{bmatrix}$$

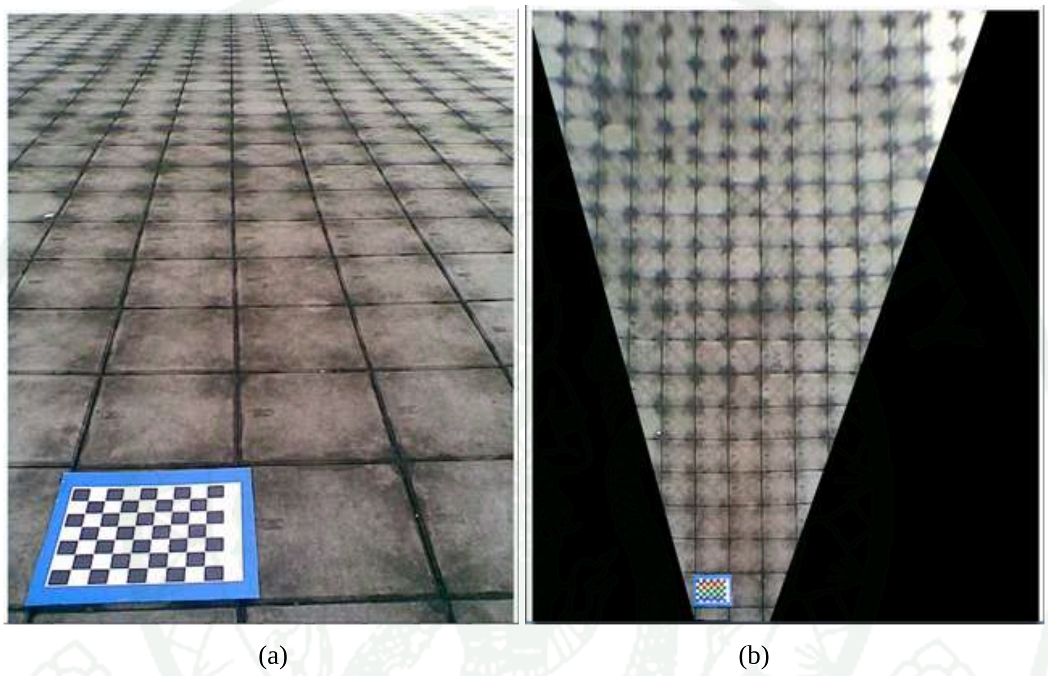

(a) (b)

Figure 44 Warping from (a) perspective view to (b) top view by using feature on chessboard.

4. Warping to top view using orientation sensor in real-time

Practically, we cannot always have chessboard placed on target areas. So the orientation sensor is used to provide rotation angle. Then, homography is formed based on the rotation information. Roll and Pitch angles are acquired through serial port, COM port, from the orientation sensor. After that, multiply rotation matrix, concatenated by translation vector, with intrinsic matrix. Resulting in homography matrix is shown in Equation 33. Because of roll and yaw angle's rotation does not affect the size of the area. Hence, we will focus only relationship between Pitch angle and area size in pixel unit. In the experimental setup, the pitch angle was manually varied from 10 to 80 degree as Figure 45.

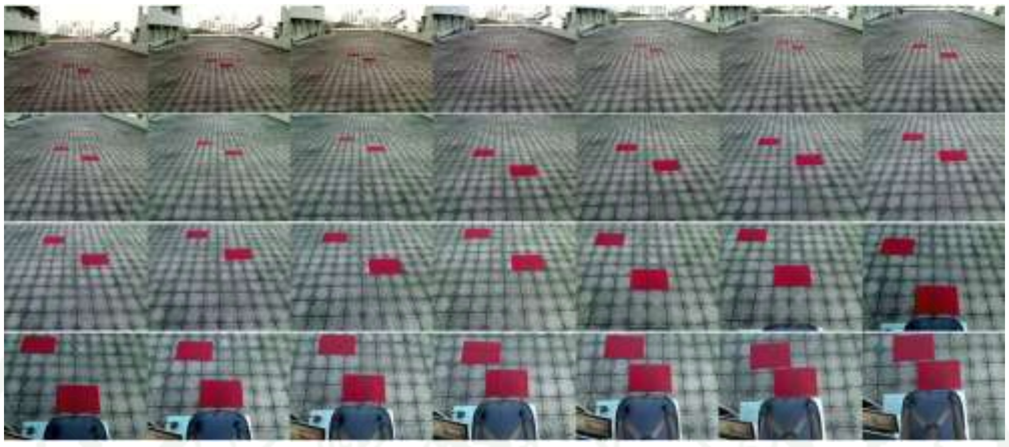


Figure 45 The pitch angle of camera is manually varied from 10 degree to 80 degree that are captured from perspective view.

After that, orientation sensor information is used for warping every image to the top view as Figure 46.

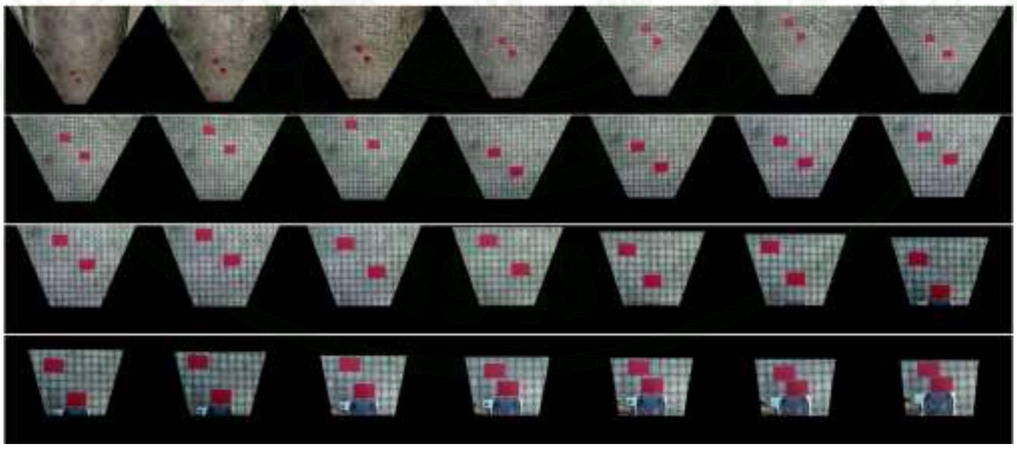
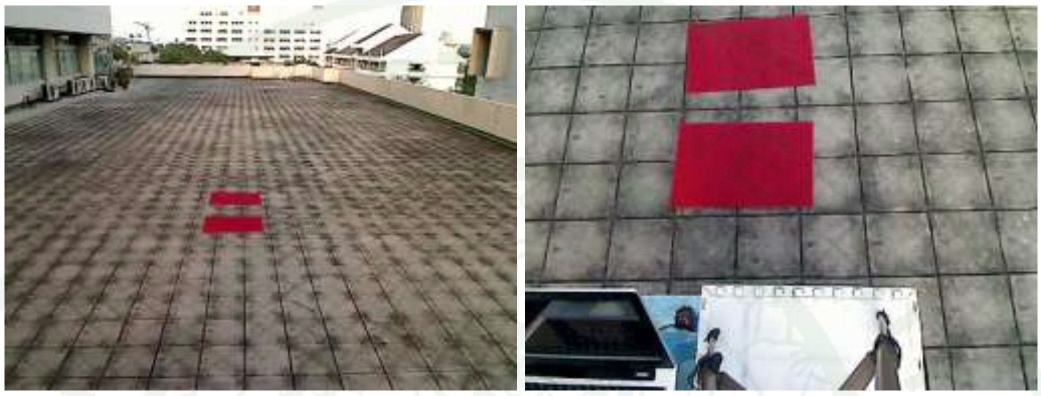


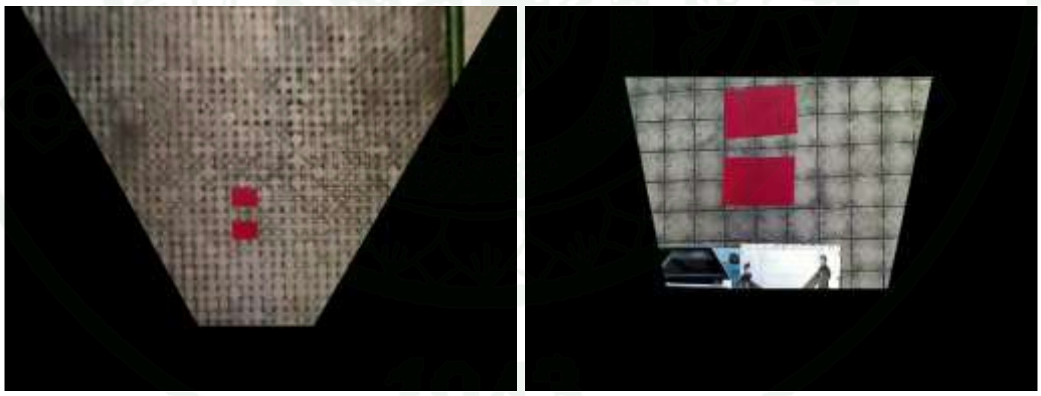
Figure 46 The top view images are generated by orientation sensor information that are varied from 10 degree to 80 degree.

Focusing on Figure 47(a) and 47(b), it shows the perspective view at 20 and 70 degree pitch angle respectively. Figure 48(a) and 48(b) shows the top view warped image at 20 and 70 degree respectively. The area size after warping to top view increases along with the pitch angle.



(a) (b)

Figure 47 Perspective view while pitch angle is (a) 20^o (b) 70^o.



(a) (b)

Figure 48 Warping to top view is generated by orientation sensor information while pitch angle is (a) 20^o (b) 70^o.

```
R=MulMat3x3(Rz,Rx); //roll*pitch
r11=CV_MAT_ELEM(*R,float,0,0);
r12=CV_MAT_ELEM(*R,float,0,1);
r13=CV_MAT_ELEM(*R,float,0,2);

r21=CV_MAT_ELEM(*R,float,1,0);
r22=CV_MAT_ELEM(*R,float,1,1);
r23=CV_MAT_ELEM(*R,float,1,2);

r31=CV_MAT_ELEM(*R,float,2,0);
r32=CV_MAT_ELEM(*R,float,2,1);
r33=CV_MAT_ELEM(*R,float,2,2);

t1=0;
t2=0;
t3=0.5*fx;  // fx=fy from intrinsic parameter
float normalize=(float) 1.0;


CV_MAT_ELEM( *H, float, 0, 0 )=normalize*(fx*r11 + cx*r31);
CV_MAT_ELEM( *H, float, 0, 1 )=normalize*(fx*r12 + cx*r32);
CV_MAT_ELEM( *H, float, 0, 2 )=normalize*(fx*t1 + cx*t3);
CV_MAT_ELEM( *H, float, 1, 0 )=normalize*(fy*r21 + cy*r31);
CV_MAT_ELEM( *H, float, 1, 1 )=normalize*(fy*r22 + cy*r32);
CV_MAT_ELEM( *H, float, 1, 2 )=normalize*(fy*t2 + cy*t3);
CV_MAT_ELEM( *H, float, 2, 0 )=normalize*r31;
CV_MAT_ELEM( *H, float, 2, 1 )=normalize*r32;
CV_MAT_ELEM( *H, float, 2, 2 )=t3;
```

```
CV_MAT_ELEM( *T, float, 0, 0 )=1;   // T(inverse)
CV_MAT_ELEM( *T, float, 0, 1 )=0;
CV_MAT_ELEM( *T, float, 0, 2 )=-cx;
CV_MAT_ELEM( *T, float, 1, 0 )=0;
CV_MAT_ELEM( *T, float, 1, 1 )=1;
CV_MAT_ELEM( *T, float, 1, 2 )=-cy;
CV_MAT_ELEM( *T, float, 2, 0 )=0;
CV_MAT_ELEM( *T, float, 2, 1 )=0;
CV_MAT_ELEM( *T, float, 2, 2 )=1;

H_Sensor=MulMat3x3(H,T);   // H*T(inverse)

Bitmap^ outImage2;

if(warpI == true && cntRecord == 0 ){
      cvWarpPerspective(Img,warpingImg,H_Sensor,CV_INTER_LINEAR |
                        CV_WARP_INVERSE_MAP |
                        CV_WARP_FILL_OUTLIERS);

outImage2 = gcnew Bitmap(warpingImg->width,
                         warpingImg->height,warpingImg-
                         >widthStep,System::Drawing::Imaging::Pixe
                         lFormat::Format24bppRgb,System::IntPtr(wa
                         rpingImg->imageData));
this->outputImage->Image = gcnew Bitmap(outImage2);
```

5. Comparing the warped image with the ground truth

The warped image will be compared with the ground truth by using a scheme as shown Figure 41. The number of red pixels, at 20 and 70 degree, respectively, is shown in Figure 49.

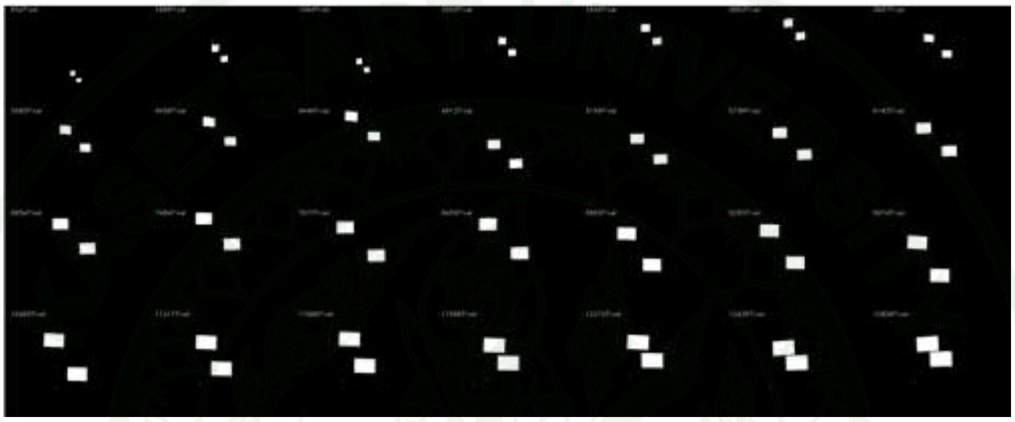
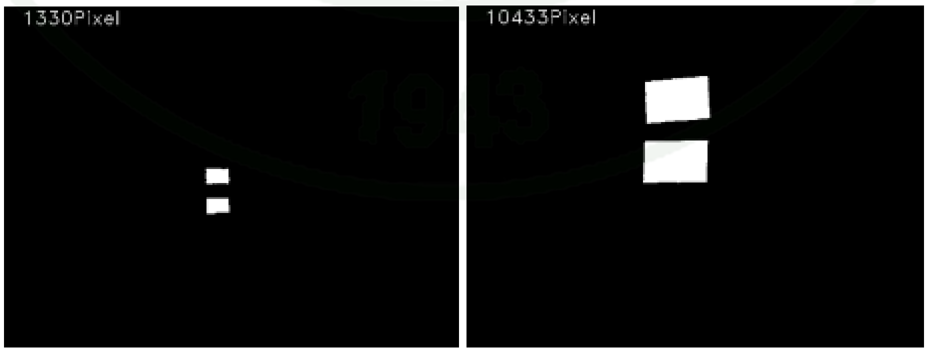


Figure 49 The top view images are generated by orientation sensor information that are varied from 10 degree to 80 degree.

Figure 50(a) and 51(b) shows the number of red pixels at 20 and 70 degree respectively. The area size in pixel unit after warping to top view will increase by depends on the pitch angle.



(a) (b)

Figure 50 The top view images are generated by orientation sensor information at 20, and 70 degree.

```
for(i=0;i < (height);i++) for(j=0;j <(width);j++)
            {
/*As I told you previously you need to select pixels which are
more red than any other color
Hence i select a difference of 29(which again depends on the
scene). (you need to select randomly and test*/

if(((data[i*step+j*channels+2]) > (50+data[i*step+j*channels+1]))
&& ((data[i*step+j*channels+2]) >
(50+data[i*step+j*channels+0]))){

      datar[i*stepr+j*channelsr]=255;
      countpixel=(countpixel+1);}

else{
      datar[i*stepr+j*channelsr]=0;
      noncountpixel=(noncountpixel+1);}


      PercentG=countpixel;//*100.0/(countpixel+noncountpixel));
}
```

This codes count the number o red pixel compared with some number of theshold. There are 3 chennels in a pixel (blue, green, red). If chennel3 (red) has values more than chennel1 (green) plus 80, and if chennel3 (red) also has values more than chennel0 (blue) plus 80, the count oixel can be performed.

In this paper's experiment, we compare accuracy before and after using iteratve least square method (ILS) as in the section 5.1 and 5.2. During experiment data collecttion, height between ground plane and camera is fixed. After that we vary the height and find the result as in the last section.

5.1 Experiment result before using ILS method

In our previous work, we varied angle between 10 to 80 degree and collect data using 1, 2, 3 , and 4 reference objects as shown in Table 2.

Table 2 The number of pixel from target area, while varying pitch angle and number of sheets.

Pitch angle	**Area1(pixel)**	**Area2(pixel)**	**Area3(pixel)**	**Area4(pixel)**
20	699	1330	2085	2778
22.5	829	1672	2315	3354
25	969	1942	3174	4233
27.5	1194	2294	3474	4827
30	1399	2672	4310	5828
32.5	1496	3025	4835	6589
35	1770	3329	5142	7585
37.5	1979	4043	6260	8282
40	2229	4482	6601	9058
42.5	2553	4936	7552	9975
45	2829	5463	8305	10926
47.5	3074	5847	9149	11694
50	3336	6810	9717	13620
52.5	3533	7072	11041	14144
55	3819	7595	11912	15190
57.5	4044	8031	12372	16062
60	4284	8490	13070	16980
62.5	4729	9162	14002	18324
65	4959	9716	14872	19432
67.5	5129	10125	15342	20250
70	5333	10433	16065	20866
72.5	5562	10996	17581	21992
75	5747	11357	18198	22714
77.5	5958	11807	18692	23614
80	6334	12946	18559	25892

After we get the experiment data from the Table 2, we polt it on to the graph. The result shows the relation between pitch angle and the area size in pixel unit.

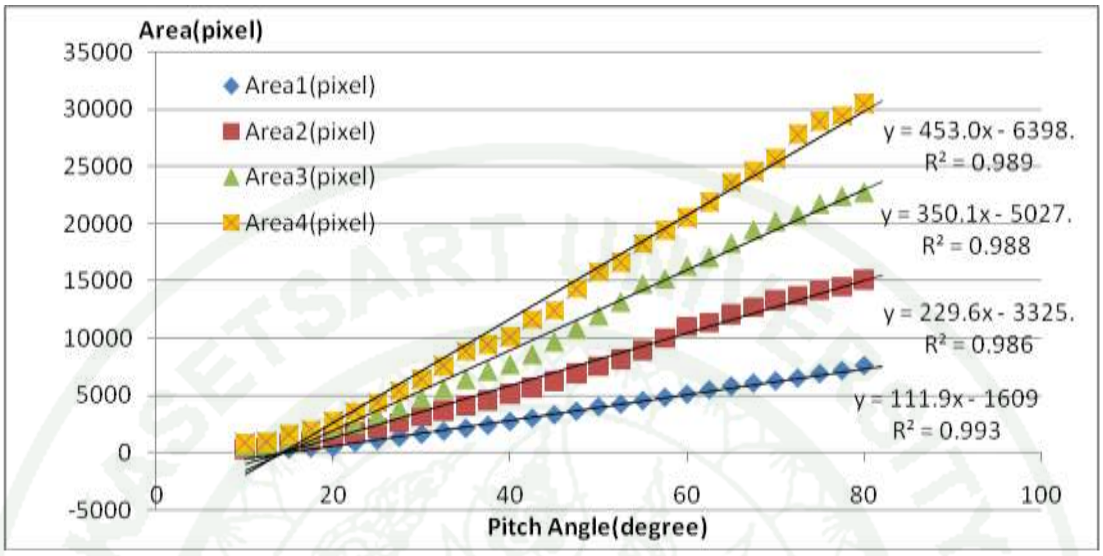


Figure 51 Relation between pitch angle and pixel numbers of the same object varied from 10-80 degree.

The result is shown in Figure xx. At 10 to 20 degree, there are a lot of errors. Therefore, to find the average accurate of both previous work and the present one are considered angles only 20 to 80 degree. Accuracy of our previous system is equal to 94.82% and it will be shown here that the accuracy can be improved.

Table 3 The area in real world unit that is pre-calibrated using images with 2 sheets, while varying pitch angle and number of sheets.

Pitch angle	**Area1(cm^2)**	**Area2(cm^2)**	**Area3(cm^2)**	**Area4(cm^2)**
20	3590.251572	7025.841228	10717.17616	13663.95741
22.5	3116.990652	6429.2591	8668.016132	12264.28018
25	2873.414634	5870.244655	9346.287483	12317.92145
27.5	2922.908529	5712.42365	8429.453413	11664.65637
30	2916.010982	5656.932997	8892.966972	12041.75313
32.5	2714.902022	5569.869975	8677.64325	11890.25654
35	2844.284777	5423.055825	8165.725603	12149.22364
37.5	2853.359786	5905.462018	8914.439201	11925.32775
40	2914.349823	5933.434852	8520.082217	11845.83768
42.5	3053.448276	5974.772596	8913.185448	11948.60778
45	3117.796241	6090.979202	9029.035186	12073.01068
47.5	3141.09551	6042.411475	9219.653602	11991.84951
50	3177.423573	6558.040646	9125.118233	13029.36757
52.5	3151.158339	6375.882384	9707.319199	12679.52294
55	3202.669357	6436.769591	9845.315796	12811.34452
57.5	3200.099833	6421.206189	9647.221684	12789.88224
60	3209.040794	6424.714526	9645.983919	12805.38209
62.5	3362.853687	6580.859368	9808.79552	13124.44363
65	3356.313569	6641.242371	9914.530413	13252.0245
67.5	3311.635215	6601.513575	9756.147829	13179.21344
70	3291.874776	6502.340936	9765.52301	12987.03695
72.5	3288.561023	6563.761314	10235.79733	13115.08605
75	3260.55191	6504.511633	10165.7663	13001.59756
77.5	3248.897597	6498.865068	10035.15401	12994.82726
80	3324.723916	6858.667497	9590.330704	13718.68342

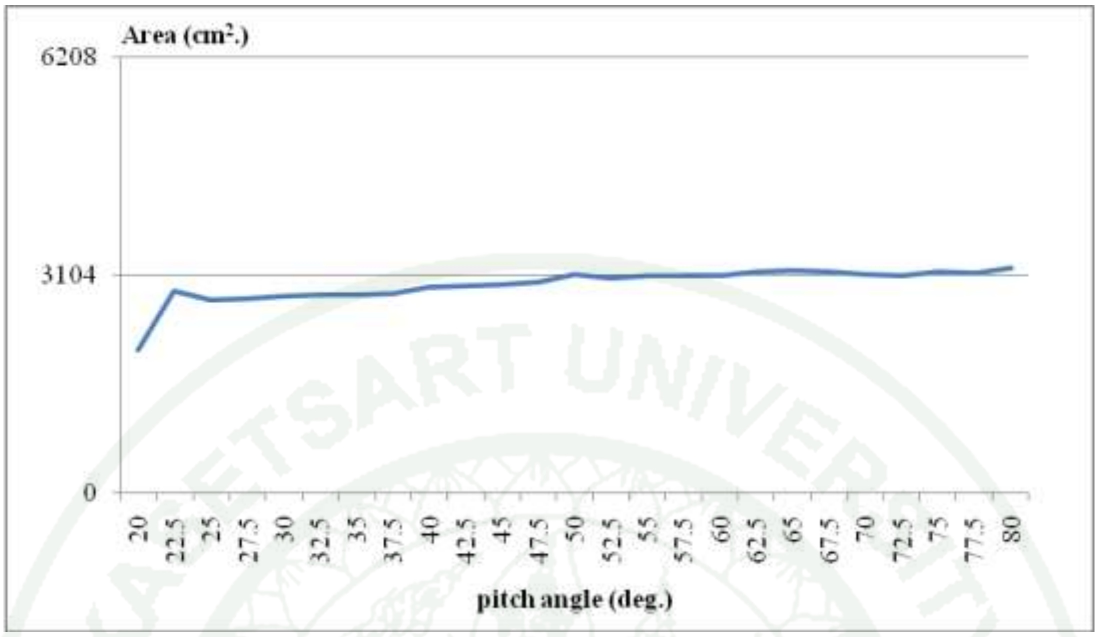


Figure 52 Relation between pitch angle and area size in real world using 1 reference object.

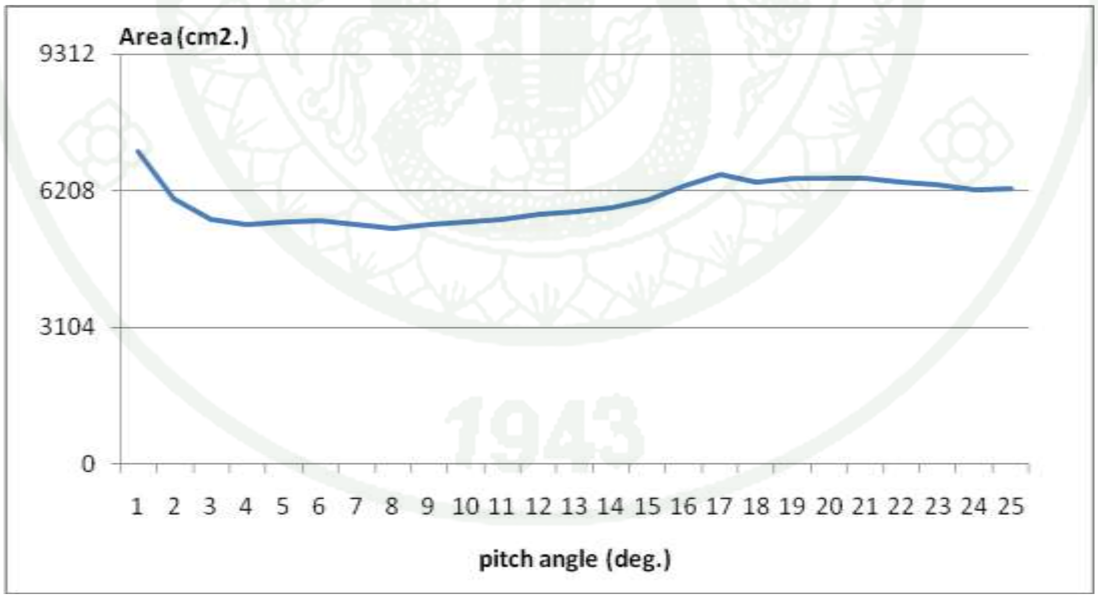


Figure 53 Relation between pitch angle and area size in real world using 2 reference objects.

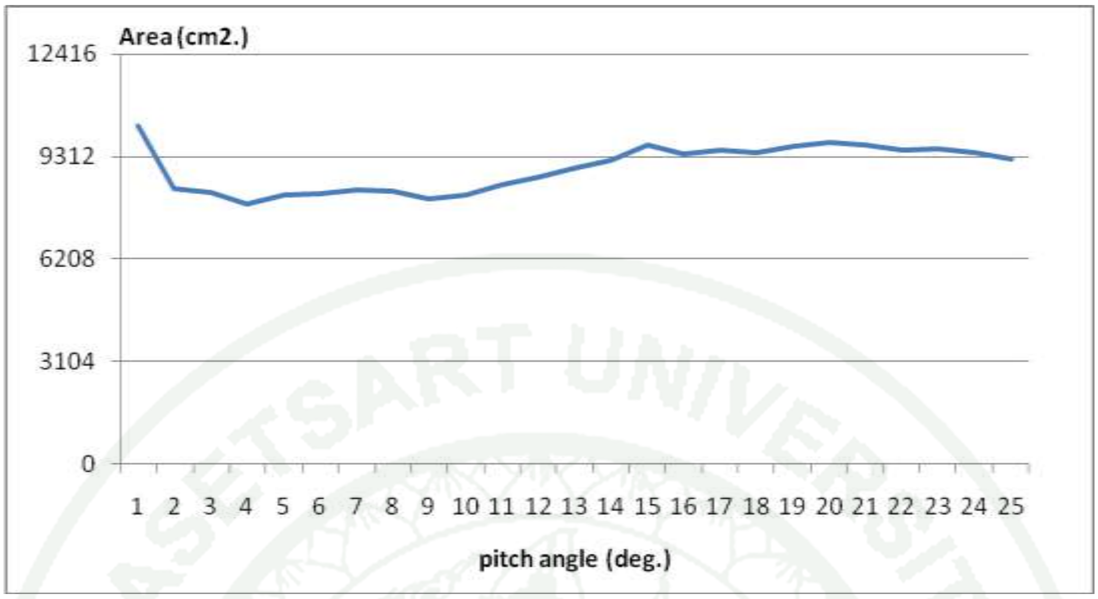


Figure 54 Relation between pitch angle and area size in real world using 3 reference objects.

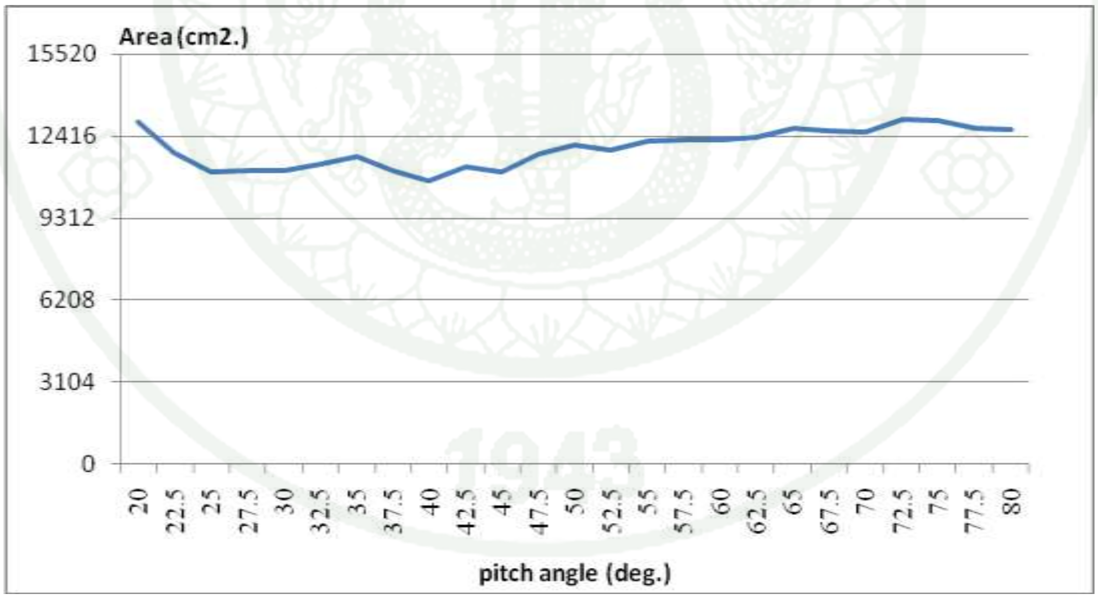


Figure 55 Relation between pitch angle and area size in real world using 4 reference objects.

Table 4 Percent accuracy.

Pitch angle	Area1(%)	Area2(%)	Area3(%)	Area4(%)	Avg.(%)
20	88.9841	91.3753	89.5364	86.3174	89.05336
22.5	96.3819	99.4010	89.3425	69.8132	88.7347
25	88.8501	90.7582	96.33361	75.2761	87.80455
27.5	90.3805	88.3182	86.8836	67.8918	83.36859
30	90.1673	87.4603	91.6611	71.62505	85.22846
32.5	83.9487	86.1142	89.4417	69.8908	82.3489
35	87.9494	83.8444	84.1653	65.7677	80.43175
37.5	88.2300	91.3027	91.8824	71.7979	85.80332
40	90.11595	91.7352	87.8177	68.6217	84.57269
42.5	94.4170	92.3743	91.8695	71.7878	87.61222
45	96.4068	94.1709	93.0636	72.7209	89.0906
47.5	97.12726	93.4200	95.0283	74.2562	89.95799
50	98.2505	98.6079	94.0539	73.4948	91.10183
52.5	97.43845	98.5757	99.9451	78.1839	93.53583
55	99.0312	99.5171	98.5228	79.2953	94.09164
57.5	98.9517	99.2765	99.4353	77.6999	93.8409
60	99.2282	99.3307	99.4226	77.6899	93.9179
62.5	96.0156	98.2551	98.8992	79.0012	93.04282
65	96.2178	97.3215	97.8094	79.8528	92.80043
67.5	97.5994	97.9357	99.4418	78.5772	93.38858
70	98.2104	99.4690	99.3452	78.6527	93.91937
72.5	98.3128	98.5194	94.4980	82.4403	93.4427
75	99.1789	99.4355	95.2198	81.8763	93.92768
77.5	99.5393	99.5228	96.5661	80.8243	94.11316
80	97.1946	93.95999	98.8490	77.2417	91.81135
	94.7251	**94.8001**	**94.3614**	**75.6239**	**89.87765**

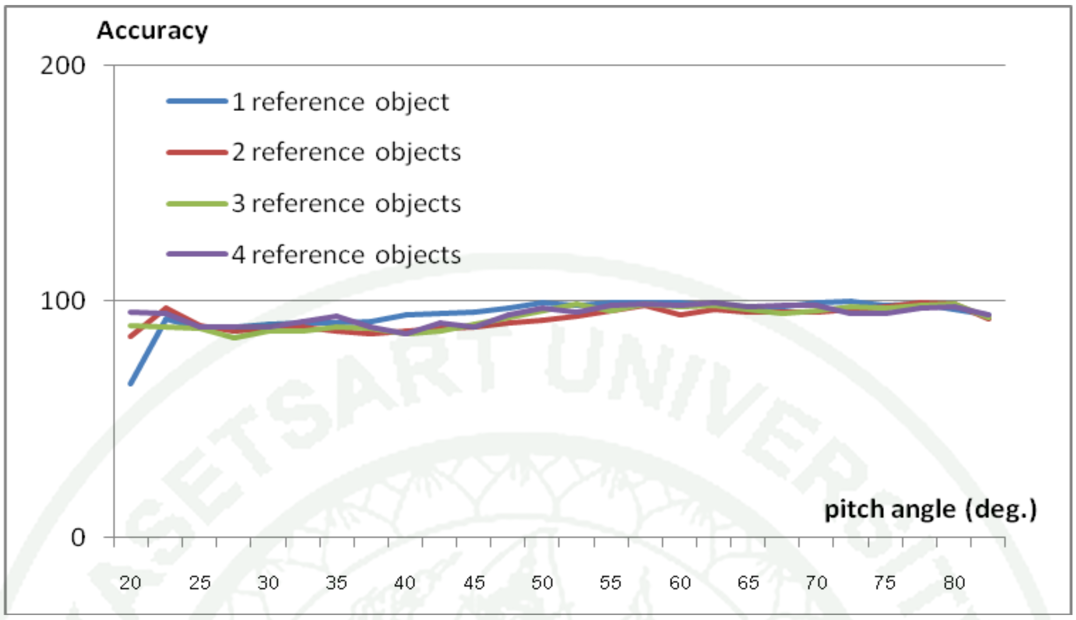


Figure 56 Relation between pitch angle and system accuracy with reference object 1, 2, 3 ,4.

5.2 Experiment result after using ILS method

Table 5 The number of pixel from target area, while varying pitch angle and number of sheets after using ILS method.

Pitch angle	**Area1(pixel)**	**Area2(pixel)**	**Area3(pixel)**	**Area4(pixel)**
20	580	1384	2274	2911
22.5	752	1718	2918	3641
25	931	2052	3675	4664
27.5	1099	2515	4296	5455
30	1382	2954	5450	6751
32.5	1663	3532	6181	7832
35	1890	4121	7157	9110
37.5	2190	4619	8171	10358
40	2514	5246	9149	11612
42.5	2811	5992	10022	13105
45	3090	6345	11244	14381
47.5	3374	7063	12031	15979
50	3950	7854	13394	17295
52.5	4215	8483	14447	18546
55	4545	9297	15760	20239
57.5	4915	10021	16537	22012
60	5279	10968	17798	23636
62.5	5559	11865	18927	25005
65	6008	12939	20004	26746
67.5	6126	13908	20958	28299
70	6482	14543	21784	29694
72.5	6766	15011	22815	30951
75	7191	15764	23304	31760
77.5	7357	16076	24320	32836
80	7786	17084	25225	33771

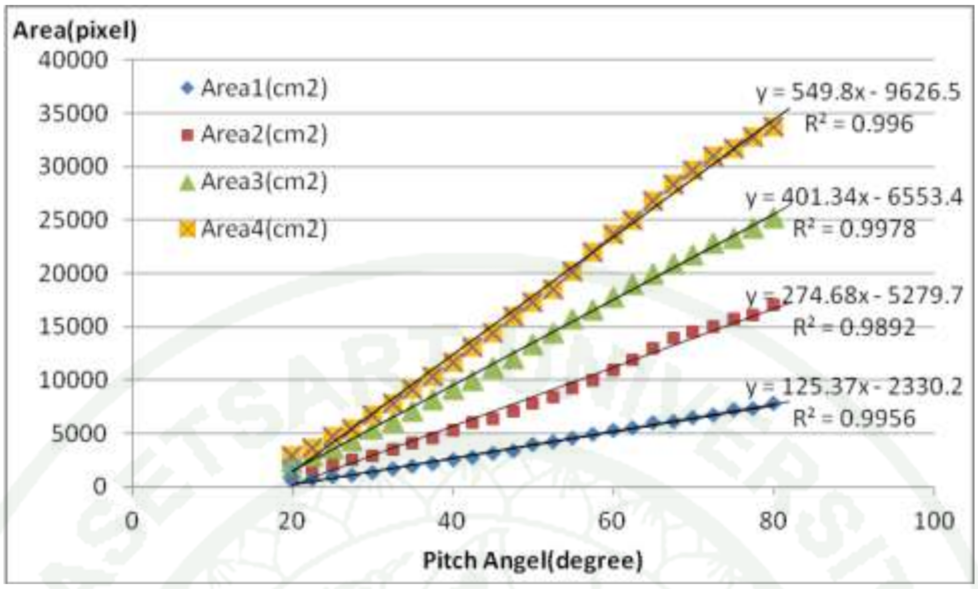


Figure 57 Relation between pitch angle and pixel numbers of the same object varied from 20-80 degree after using ILS.

Table 6 The area in real world unit that is pre-calibrated, while varying pitch angle and number of sheets after using ILS method.

Pitch angle	**Area1(cm^2)**	**Area2(cm^2)**	**Area3(cm^2)**	**Area4(cm^2)**
20	15361.09	40167.7	14457.22	26391.37
22.5	5395.453	11842.49	11008	16474.73
25	3863.143	8025.462	9856.026	14060.51
27.5	3207.688	6865.928	8937.772	12330.11
30	3110.978	6193.951	9261.698	12205.38
32.5	3046.613	6011.585	8877.771	11798.36
35	2919.05	5902.764	8901.501	11762.05
37.5	2923.548	5711.192	8961.296	11700.93
40	2955.183	5706.03	8973.141	11659.42
42.5	2951.715	5817.512	8889.585	11842.19
45	2931.838	5562.818	9103.163	11813.46
47.5	2919.783	5644.872	8958.547	12031.98
50	3141.942	5767.199	9232.455	12020.86
52.5	3101.995	5761.127	9269.572	11969.39
55	3112.114	5872.765	9457.92	12191.02
57.5	3146.524	5916.683	9321.258	12430.12
60	3173.125	6078.809	9457.438	12561.89
62.5	3149.078	6196.093	9512.521	12551.02
65	3218.173	6387.953	9537.151	12718.19
67.5	3111.988	6510.788	9503.598	12783.71
70	3131.196	6472.87	9417.834	12775.02
72.5	3115.454	6367.669	9424.388	12710.45
75	3163.141	6387.377	9216.039	12475.51
77.5	3097.695	6234.371	9224.618	12360.66
80	3143.812	6352.763	9192.075	12204.05

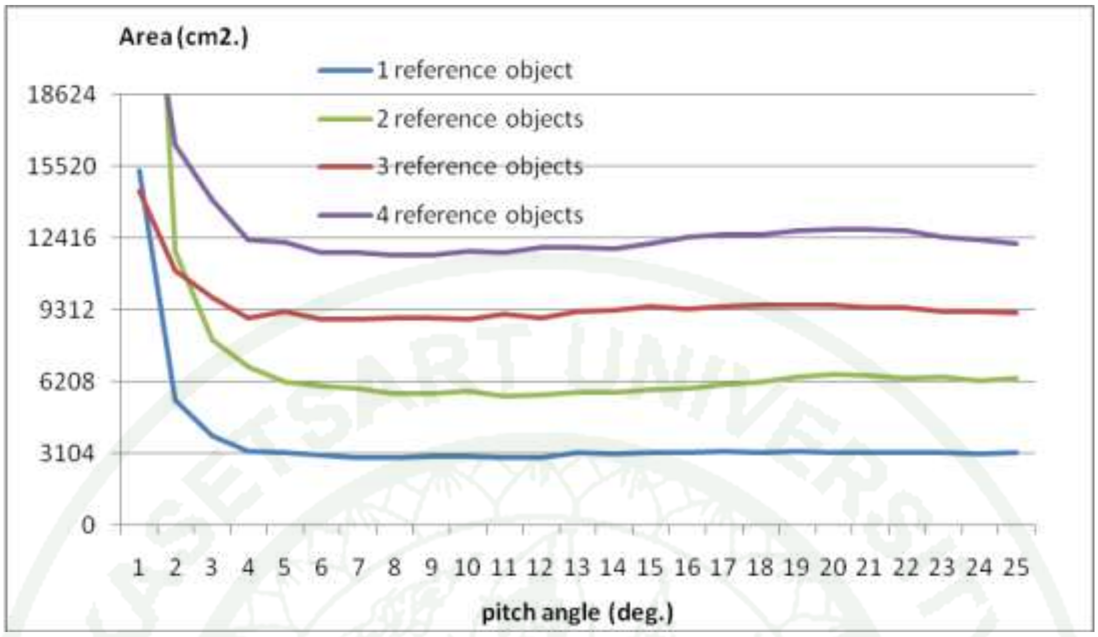


Figure 58 Relation between pitch angle and area size in real world using 1,2,3, 4 reference objects.

We found that there are a lot of error between 20-30 degree, so to fine accuracy, we use 30-80 degree.

Table 7 Percent accuracy.

Pitch angle	Area1(%)	Area2(%)	Area3(%)	Area4(%)	Avg.(%)
20	-294.881	-447.031	44.74636	-12.5593	-177.431
22.5	26.17741	9.238286	81.78697	67.3105	46.12829
25	75.54308	70.72387	94.1578	86.75489	81.79491
27.5	96.65954	89.40193	95.98123	99.30821	95.33773
30	99.77518	99.7737	99.45981	98.3036	99.32808
32.5	98.15118	96.8361	95.33688	95.02548	96.33741
35	94.04155	95.08318	95.59172	94.73301	94.86236
37.5	94.18646	91.99729	96.23385	94.24074	94.16459
40	95.20564	91.91415	96.36105	93.90643	94.34682
42.5	95.09392	93.70992	95.46376	95.37846	94.91151
45	94.45353	89.60725	97.75734	95.14704	94.24129
47.5	94.06517	90.92899	96.20432	96.90703	94.52638
50	98.77764	92.89947	99.14578	96.81753	96.91011
52.5	99.93539	92.80166	99.54438	96.40295	97.1711
55	99.73859	94.59996	98.43299	98.18799	97.73988
57.5	98.63001	95.30739	99.90058	99.8863	98.43107
60	97.77304	97.91895	98.43817	98.82499	98.23879
62.5	98.54775	99.80821	97.84664	98.91252	98.77878
65	96.32173	97.10128	97.58214	97.56611	97.14282
67.5	99.74265	95.12261	97.94246	97.03838	97.46153
70	99.12383	95.73341	98.86347	97.1084	97.70728
72.5	99.631	97.42801	98.79308	97.6285	98.37015
75	98.09468	97.11056	98.96949	99.5207	98.42386
77.5	99.79687	99.57521	99.06162	99.55432	99.497
80	98.71738	97.66812	98.71214	98.29295	98.34765
	97.60968	**95.3774**	**97.8877**	**97.1135**	**96.99707**

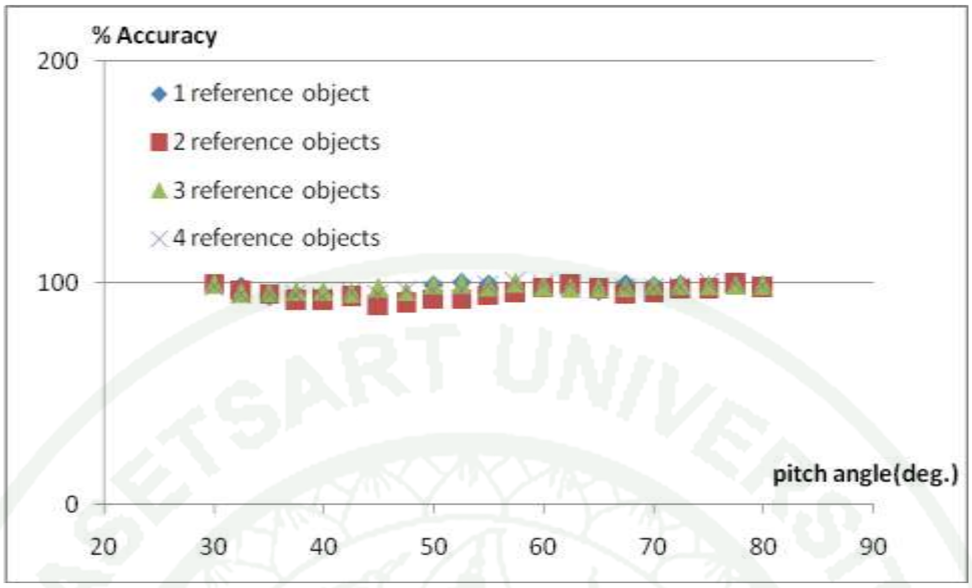


Figure 59 Relation between pitch angle and system accuracy with reference object 1, 2, 3 ,4. After using ILS.

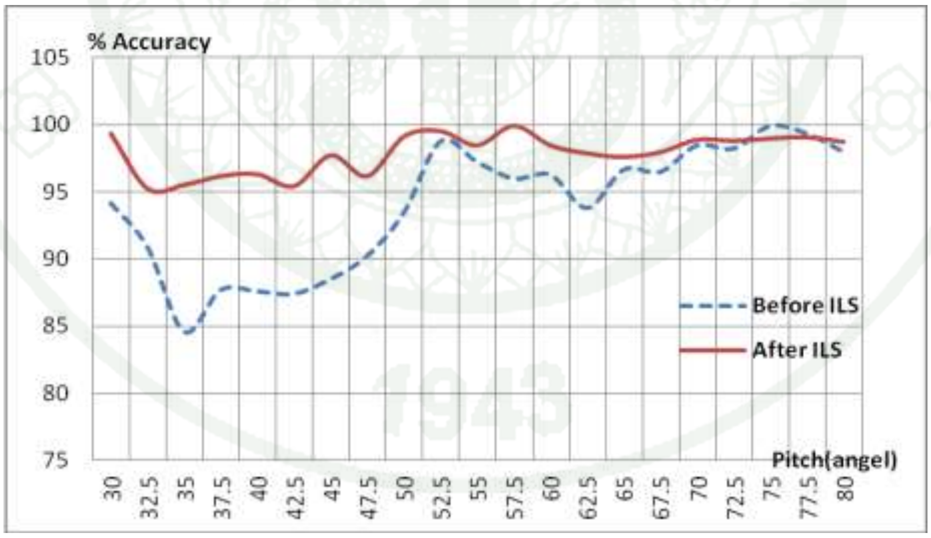


Figure 60 Relation between pitch angle and the average of percent accuracy before and after when using ILS method.

5.3 Varying Height

The result of (15), varying height, is illustrated in Figure 6 where h is varied from 70-490 cm. One can see that n*n*P is almost a constant with accuracy of 95.28%. Such accuracy implies that reliability to use (15) would be reduced a little when height other than one used during calibration is required.

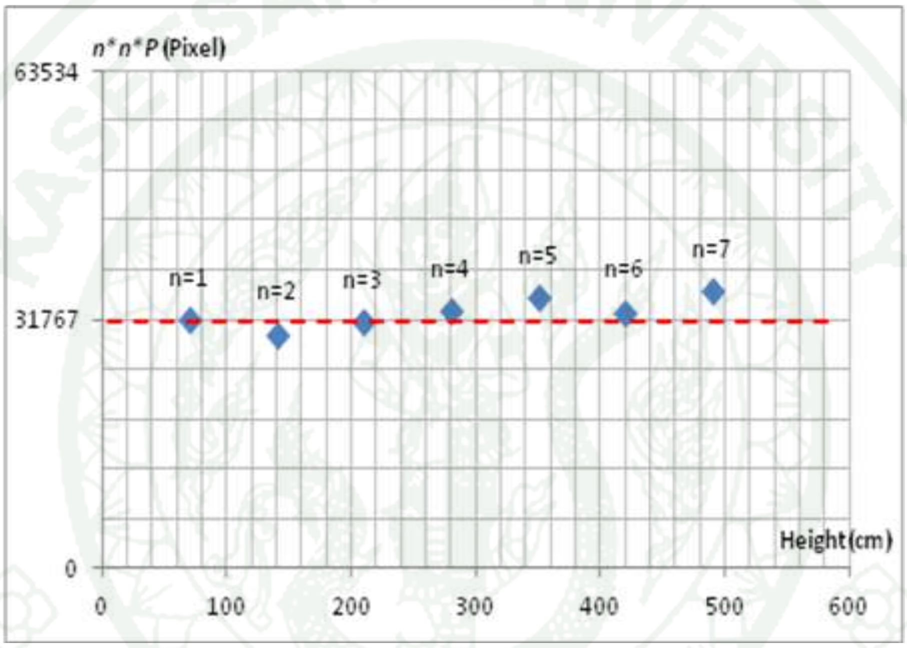


Figure 61 Relation between height and pixel area multipled by n*n factor.

Discussion

From experimental result, we cannot always have chessboard placed on target areas. So the orientation sensor is used to provide rotation angle. Only pitch angle affects output area size, in top view. Roll and yaw angle's rotation does not affect the size of the area. Hence, focus only relationship between pitch angle and area size in pixel unit.

In this thesis, the use of pitch angle range is from 20 to 80 degrees because from 0 to 20 degrees, there is some warping error, and from 80 to 90 degrees, there some the sensor problems. Results indicate that pitch angle range from 30 to 80 degrees produce the highest accuracy. The averaged accuracy of the range is 97.86%.

CONCLUSION AND RECOMMENDATION

Conclusion

To achieve the precision crop area calculation using camera and orientation sensor, we apply a top view image warping algorithm instead of directly adjusting the moving of motor. In order to do so, the homography matrix is computed to transform the input image into its top view image. The intrinsic parameters can be calibrated offline, relationship between camera and sensor is also derived offline, whereas the rotation angel received from sensor is provided in real time.

For the problem caused by imperfect alignment between camera and sensor, we employ Iterative Least Square (ILS) method for solving it. Finally, a number of the target pixels will be map to planar surface area in real world unit in order to approximate the area. Both the warping and the mapping depend largely on pitch angle derived from the orientation sensor. The average of percent accuracy which is varied pitch angle from 30 to 80 degree is 97.86%

Recommendation

The present experiment is based on images of known object from video camera and rotational information from orientation sensor. In the future we aim to achieve area calculation of agricultural crop field. In doing so, classification is required to extract crop area instead of just counting target color pixels. Furthermore the system should be linked to a GPS system to cross validate with Satellite data.

LITERATURE CITED

Asenso Okyere, H. Kwadwo, K. Davis, S. Kristin, T. Aredo, P. Dejene, 2008. **Advancing Agricalture in Developing Countries Through Knowledge and Innovation**, Washington, D.C: International Food Policy Research Institute.

Chuan, Z., T. D. Long, Z. Feng, and D. Z. Li. 2003. A planar homography estimation method for camera calibration, pp.424- 429. **Computational Intelligence in Robotics and Automation, Proceedings, IEEE International Symposium.**

Dengxin. D, Wen. Y, Jan. 2011. Satellite Image Classification via Two-Layer Sparse Coding With Biased Image Representation, **Geoscience and Remote Sensing Letters, IEEE** , vol.8, no.1, pp.173-176.

Frahm, J.-M.; Koch, R.; , "Camera calibration with known rotation," Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on , vol., no., pp.1418-1425 vol.2, 13-16 Oct. 2003

Gary, B., and K. Adrian. 2008. **Learning OpenCV.** 1st ed. O'Reilly.

Gonzalez, R.C. and R.E. Woods. 2008. **Digital Image Processing**. 3rd ed. Prentice-Hall.

H. Kano, K. Asari, Y. Isihii, H. Hongo, "Precise Top view Image Generation without Global Metric Information", IEICE TRANS. On Information and Systems, Japan, 2008

KANO, H., K. ASARI, Y. ISIHII and H. HONGO. 2008. Precise Top view Image Generation without Global Metric Information. pp.1893-1898. In **IEICE TRANS. INF. & SYST., Vol.E91-D, NO.7**

LinBo Luo; InSung Koh; SangYoon Park; ReSen Ahn; JongWha Chong; , "A software-hardware cooperative implementation of bird's-eye view system for camera-on-vehicle," Network Infrastructure and Digital Content, 2009. IC-NIDC 2009. IEEE International Conference on , vol., no., pp.963-967, 6-8 Nov. 2009

Luo, L., I. Koh, S. Park, R. Ahn and J. Chong.2009. A Software-Hardware Cooperative Implementation of Bird’s Eye View System for Camera-On-Vehicle. In **IEEE International Conference on Digital Object Identifier.**

OpenCV 2.1 documentation. **Machine Learning Library**. Available Source: http://opencv.willowgarage.com/documentation/cpp/ml._machine_learning.html

Pharsook, S., P. Larmsrichan, S. Siddhichai, T. Kasetkasem, T. Chanwimaluang and T. Isshiki. 2011. The Comparison of Three Feature Extraction for Classification of Image Texture. *In* **Proceedings of ICICTES2011**, Thailand, Jan 27-29.

Scherer, D., P. Dubois and B. Sherwood. 2000. VPython: 3D Interactive Scientific Graphics for Students, pp82-88. **Computing in Science and Engineering, Sept./Oct.**

Teshima, T., H Saito, K Yamamoto and T Ihara. 2006. Vehicle Lateral Position Estimation Method Based on Matching of Top-View Images, pp.626-629. **The 18th International Conference on Pattern Recognition (ICPR'06).**

Tian, Y., B. Wu, W. Xu, J. Huang, W. Xu. 2004. An Effective Field Method of Crop Proportion Survey in China Based on GVG Integrated System, pp.4028-4030. **Proceedings of IEEE International Conference on Geoscience and Remote Sensing Symposium**.

Wang, S., M Enrong, Y Bangjie, S Zhenghe and L Antao. 2006. A Statistic Method of Crop Acreage Based on Image Recognition, pp.418-422. **Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06).**

Wang, S., Z. Song, Z. Zhu, B. Yang, E. Mao and R. Zhang. 2007. Study on crop Image Feature Extraction of Vehicle-Based Estimation System on Large Scale Crop Acreage. **Proceedings of the 6th IEEE International conference on Machine Learning and Cybernetic**, Hong Kong.

Wang Suxia; , "A Statistic Method of Crop Acreage Based on Image Recognition," Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on , vol.2, no., pp.418-422, 16-18 Oct. 2006

W. Bingfang, "Introduction of China Cropwatch System with Remote Sensing," Proceedings of International Society for Photogrammetry and Remote Sensing, China, 2006

Yichen Tian; Bingfang Wu; Wenbo Xu; Jianxi Huang; Wenting Xu; , "An effective field method of crop proportion survey in China based on GVG integrated system," Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004 IEEE International , vol.6, no., pp.4028-4030 vol.6, 20-24 Sept. 2004

Wang, Y. M.; Li, Y.; Zheng, J. B.; , "A camera calibration technique based on OpenCV," Information Sciences and Interaction Sciences (ICIS), 2010 3rd International Conference on , vol., no., pp.403-406, 23-25 June 2010

Wikipedia the free encyclopedia. **RGB color model**. Available Source: http://en.wikipedia.org/wiki/RGB_color_model

Yuan Xin; Zhu Ruishuang; Su Li; , "A Calibration Method Based on OpenCV," Intelligent Systems and Applications (ISA), 2011 3rd International Workshop on , vol., no., pp.1-4, 28-29 May 2011

Zhang, Z., 2006. A flexible new technique for camera calibration, pp.1330-1334. **IEEE transaction pattern analysis and machine intelligence.**

Ziraknejad, N.; Tafazoliziraknejad, N.; Tafazoli, S.; Lawrence, P.D.; , "Autonomous Stereo Camera Parameter Estimation for Outdoor Visual Servoing," Machine Learning for Signal Processing, 2007 IEEE Workshop on, Vol., No., pp. 157-162, 27-29 Aug. 2007

APPENDICES

Appendix A

Razor 9DOF IMU AHRS V1.1 Lite version for AT328/AT168

Source: http://code.google.com/p/sf9domahrs/downloads/list

SD9DOF_AHRS.pde

```
// Sparkfveun 9DOF Razor IMU AHRS
// 9 Degree of Measurement Attitude and Heading Reference System
// Firmware v1.0 Released under Creative Commons License
// Code by Doug Weibel and Jose Julio
// Based on ArduIMU v1.5 by Jordi Munoz and William Premerlani, Jose Julio and
Doug Weibel
        // Axis definition:
        // X axis pointing forward (to the FTDI connector)
        // Y axis pointing to the right
        // and Z axis pointing down.
        // Positive pitch : nose up
        // Positive roll : right wing down
        // Positive yaw : clockwise
/* Hardware version - v13 ATMega328@3.3V w/ external 8MHz resonator High
Fuse DA Low Fuse FF ADXL345:
        Accelerometer HMC5843: Magnetometer
        LY530:          Yaw Gyro
        LPR530:         Pitch and Roll Gyro
        Programmer : 3.3v FTDI
        Arduino IDE : Select board  "Arduino Duemilanove w/ATmega328" */
// This code works also on ATMega168 Hardware
#include <Wire.h>
// ADXL345 Sensitivity(from datasheet) => 4mg/LSB   1G => 1000mg/4mg = 256
steps
// Tested value : 248
#define GRAVITY 248  //this equivalent to 1G in the raw data coming from the
accelerometer
```

```
#define Accel_Scale(x) x*(GRAVITY/9.81)//Scaling the raw data of the accel to
actual acceleration in meters for seconds square
#define ToRad(x) (x*0.01745329252)  // *pi/180
#define ToDeg(x) (x*57.2957795131)  // *180/pi

// LPR530 & LY530 Sensitivity (from datasheet) => (3.3mv at 3v)at 3.3v: 3mV/º/s,
3.22mV/ADC step => 0.93
// Tested values : 0.92
#define Gyro_Gain_X 0.92 //X axis Gyro gain
#define Gyro_Gain_Y 0.92 //Y axis Gyro gain
#define Gyro_Gain_Z 0.92 //Z axis Gyro gain
#define Gyro_Scaled_X(x) x*ToRad(Gyro_Gain_X) //Return the scaled ADC raw
data of the gyro in radians for second
#define Gyro_Scaled_Y(x) x*ToRad(Gyro_Gain_Y) //Return the scaled ADC raw
data of the gyro in radians for second
#define Gyro_Scaled_Z(x) x*ToRad(Gyro_Gain_Z) //Return the scaled ADC raw
data of the gyro in radians for second

#define Kp_ROLLPITCH 0.02
#define Ki_ROLLPITCH 0.00002
#define Kp_YAW 1.2
#define Ki_YAW 0.00002

/*For debugging purposes*/
//OUTPUTMODE=1 will print the corrected data,
//OUTPUTMODE=0 will print uncorrected data of the gyros (with drift)
#define OUTPUTMODE 1

//#define PRINT_DCM 0     //Will print the whole direction cosine matrix
#define PRINT_ANALOGS 0 //Will print the analog raw data
#define PRINT_EULER 1   //Will print the Euler angles Roll, Pitch and Yaw
```

```
#define ADC_WARM_CYCLES 50
#define STATUS_LED 13
int8_t sensors[3] = {1,2,0};  // Map the ADC channels gyro_x, gyro_y, gyro_z
int SENSOR_SIGN[9] = {-1,1,-1,1,1,1,-1,-1,-1};  //Correct directions x,y,z - gyros,
accels, magnetormeter
float G_Dt=0.02;    // Integration time (DCM algorithm)  We will run the integration
loop at 50Hz if possible
long timer=0;   //general purpuse timer
long timer_old;
long timer24=0; //Second timer used to print values
int AN[6]; //array that store the 3 ADC filtered data (gyros)
int AN_OFFSET[6]={0,0,0,0,0,0}; //Array that stores the Offset of the sensors
int ACC[3];          //array that store the accelerometers data
int accel_x; int accel_y; int accel_z;
int magnetom_x; int magnetom_y; int magnetom_z;
float MAG_Heading;
float Accel_Vector[3]= {0,0,0}; //Store the acceleration in a vector
float Gyro_Vector[3]= {0,0,0};//Store the gyros turn rate in a vector
float Omega_Vector[3]= {0,0,0}; //Corrected Gyro_Vector data
float Omega_P[3]= {0,0,0};//Omega Proportional correction
float Omega_I[3]= {0,0,0};//Omega Integrator
float Omega[3]= {0,0,0};
// Euler angles
float roll; float pitch; float yaw;
float errorRollPitch[3]= {0,0,0};
float errorYaw[3]= {0,0,0};
unsigned int counter=0;
byte gyro_sat=0;
float DCM_Matrix[3][3]= {  {1,0,0  } ,{ 0,1,0  },{ 0,0,1  }};
float Update_Matrix[3][3]={{0,1,2},{3,4,5},{6,7,8}}; //Gyros here
float Temporary_Matrix[3][3]={{0,0,0  } ,{  0,0,0  },{0,0,0  }};
 //ADC variables
```

```
volatile uint8_t MuxSel=0;
volatile uint8_t analog_reference;
volatile uint16_t analog_buffer[8];
volatile uint8_t analog_count[8];

void setup()
{
  Serial.begin(57600);
  pinMode (STATUS_LED,OUTPUT);  // Status LED

  Analog_Reference(DEFAULT);
  Analog_Init();
  I2C_Init();
  Accel_Init();
  Read_Accel();
  Serial.println("Sparkfun 9DOF Razor AHRS");
  digitalWrite(STATUS_LED,LOW);
  delay(1500);
   // Magnetometer initialization
  Compass_Init();
    // Initialze ADC readings and buffers
  Read_adc_raw();
  delay(20);
    for(int i=0;i<32;i++)    // We take some readings...
    {
          Read_adc_raw();
          Read_Accel();
           for(int y=0; y<6; y++)   // Cumulate values
                    AN_OFFSET[y] += AN[y];
                    delay(20);
    }
```

```
 for(int y=0; y<6; y++)
   AN_OFFSET[y] = AN_OFFSET[y]/32;
  AN_OFFSET[5]-=GRAVITY*SENSOR_SIGN[5];

 //Serial.println("Offset:");
 for(int y=0; y<6; y++)
         Serial.println(AN_OFFSET[y]);
  delay(2000);
 digitalWrite(STATUS_LED,HIGH);
 Read_adc_raw();     // ADC initialization
 timer=millis();
 delay(20);
 counter=0;
}

void loop() //Main Loop
{
 if((millis()-timer)>=20)  // Main loop runs at 50Hz
  {
   counter++;
   timer_old = timer;
   timer=millis();
   if (timer>timer_old)
         G_Dt = (timer-timer_old)/1000.0;
// Real time of loop run. We use this on the DCM algorithm (gyro integration time)
   else
     G_Dt = 0;

   // *** DCM algorithm
   // Data adquisition
   Read_adc_raw();   // This read gyro data
   Read_Accel();     // Read I2C accelerometer
```

```
    if (counter > 5)  // Read compass data at 10Hz... (5 loop runs)
   {
   counter=0;
   Read_Compass();    // Read I2C magnetometer
   Compass_Heading(); // Calculate magnetic heading
   }
  // Calculations...
  Matrix_update();
  Normalize();
  Drift_correction();
  Euler_angles();
  // ***
    printdata();
  //Turn off the LED when you saturate any of the gyros.
if((abs(Gyro_Vector[0])>=ToRad(300))||(abs(Gyro_Vector[1])>=ToRad(300))||(abs(
Gyro_Vector[2])>=ToRad(300)))
   {
   if (gyro_sat<50)
    gyro_sat+=10;
   }
  else
   {
   if (gyro_sat>0)
    gyro_sat--;
   }

  if (gyro_sat>0)
   digitalWrite(STATUS_LED,LOW);
  else
   digitalWrite(STATUS_LED,HIGH);
  }
 }
```

ADC.pde

```
/*We are using an oversampling and averaging method to increase the ADC
resolution. Now we store the ADC readings in float format*/
void Read_adc_raw(void)
{
  int i;  uint16_t temp1;     uint8_t temp2;
   // ADC readings...
  for (i=0;i<3;i++)
    {
      do{
        temp1= analog_buffer[sensors[i]];          // sensors[] maps sensors to correct order
        temp2= analog_count[sensors[i]];
        } while(temp1 != analog_buffer[sensors[i]]);  // Check if there was an ADC
interrupt during readings...
       if (temp2>0) AN[i] = (float)temp1/(float)temp2;     // Check for divide by zero
       }
 // Initialization for the next readings...
  for (int i=0;i<3;i++){
    do{
      analog_buffer[i]=0;
      analog_count[i]=0;
      } while(analog_buffer[i]!=0); // Check if there was an ADC interrupt during
initialization...
  }
}
float read_adc(int select)
{
  if (SENSOR_SIGN[select]<0)
    return(AN_OFFSET[select]-AN[select]);
  else
    return(AN[select]-AN_OFFSET[select]);
}
//Activating the ADC interrupts.
void Analog_Init(void)
{
 ADCSRA|=(1<<ADIE)|(1<<ADEN);
 ADCSRA|= (1<<ADSC);
}
void Analog_Reference(uint8_t mode)
{  analog_reference = mode;}
//ADC interrupt vector, this piece of code
//is executed everytime a convertion is done.
ISR(ADC_vect)
{
  volatile uint8_t low, high;
  low = ADCL;
  high = ADCH;
  if(analog_count[MuxSel]<63) {
        analog_buffer[MuxSel] += (high << 8) | low;   // cumulate analog values
        analog_count[MuxSel]++;
   }
  MuxSel++;
  MuxSel &= 0x03;   //if(MuxSel >=4) MuxSel=0;
  ADMUX = (analog_reference << 6) | MuxSel;
 // start the conversion
  ADCSRA|= (1<<ADSC);
}
```

Compass.pde

```
// Local magnetic declination
// I use this web : http://www.ngdc.noaa.gov/geomagmodels/Declination.jsp
#define MAGNETIC_DECLINATION -6.0   // not used now -> magnetic bearing

void Compass_Heading()
{
  float MAG_X;
  float MAG_Y;
  float cos_roll;
  float sin_roll;
  float cos_pitch;
  float sin_pitch;

  cos_roll = cos(roll);
  sin_roll = sin(roll);
  cos_pitch = cos(pitch);
  sin_pitch = sin(pitch);
 // Tilt compensated Magnetic filed X:
  MAG_X = 
magnetom_x*cos_pitch+magnetom_y*sin_roll*sin_pitch+magnetom_z*cos_roll*sin
_pitch;
 // Tilt compensated Magnetic filed Y:
  MAG_Y = magnetom_y*cos_roll-magnetom_z*sin_roll;
 // Magnetic Heading
  MAG_Heading = atan2(-MAG_Y,MAG_X);
}
```

DCM.pde

```
/**************************************************/
void Normalize(void)
{
  float error=0;
  float temporary[3][3];
  float renorm=0;

  error= -Vector_Dot_Product(&DCM_Matrix[0][0],&DCM_Matrix[1][0])*.5; //Equation
19

  Vector_Scale(&temporary[0][0], &DCM_Matrix[1][0], error); //Equation 19
  Vector_Scale(&temporary[1][0], &DCM_Matrix[0][0], error); //Equation 19

  Vector_Add(&temporary[0][0], &temporary[0][0], &DCM_Matrix[0][0]);//Equation 19
  Vector_Add(&temporary[1][0], &temporary[1][0], &DCM_Matrix[1][0]);//Equation 19

  Vector_Cross_Product(&temporary[2][0],&temporary[0][0],&temporary[1][0]); // c= a x
b //Equation 20

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[0][0],&temporary[0][0])); //Equation 21
  Vector_Scale(&DCM_Matrix[0][0], &temporary[0][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[1][0],&temporary[1][0])); //Equation 21
  Vector_Scale(&DCM_Matrix[1][0], &temporary[1][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[2][0],&temporary[2][0])); //Equation 21
  Vector_Scale(&DCM_Matrix[2][0], &temporary[2][0], renorm);
}
```

```
/**********************************************/
void Drift_correction(void)
{
  float mag_heading_x;
  float mag_heading_y;
  float errorCourse;
 //Compensation the Roll, Pitch and Yaw drift.
  static float Scaled_Omega_P[3];
  static float Scaled_Omega_I[3];
  float Accel_magnitude;
  float Accel_weight;


 //*****Roll and Pitch***************

 // Calculate the magnitude of the accelerometer vector
  Accel_magnitude = sqrt(Accel_Vector[0]*Accel_Vector[0] +
Accel_Vector[1]*Accel_Vector[1] + Accel_Vector[2]*Accel_Vector[2]);
  Accel_magnitude = Accel_magnitude / GRAVITY; // Scale to gravity.
 // Dynamic weighting of accelerometer info (reliability filter)
 // Weight for accelerometer info (<0.5G = 0.0, 1G = 1.0 , >1.5G = 0.0)
  Accel_weight = constrain(1 - 2*abs(1 - Accel_magnitude),0,1);  //

  Vector_Cross_Product(&errorRollPitch[0],&Accel_Vector[0],&DCM_Matrix[2][0]);
//adjust the ground of reference
  Vector_Scale(&Omega_P[0],&errorRollPitch[0],Kp_ROLLPITCH*Accel_weight);
Vector_Scale(&Scaled_Omega_I[0],&errorRollPitch[0],Ki_ROLLPITCH*Accel_weig
ht);
Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);
  //*****YAW***************
 // We make the gyro YAW drift correction based on compass magnetic heading


  mag_heading_x = cos(MAG_Heading);
  mag_heading_y = sin(MAG_Heading);
  errorCourse=(DCM_Matrix[0][0]*mag_heading_y) -
(DCM_Matrix[1][0]*mag_heading_x);  //Calculating YAW error
  Vector_Scale(errorYaw,&DCM_Matrix[2][0],errorCourse); //Applys the yaw
correction to the XYZ rotation of the aircraft, depeding the position.
   Vector_Scale(&Scaled_Omega_P[0],&errorYaw[0],Kp_YAW);//.01proportional of
YAW.
  Vector_Add(Omega_P,Omega_P,Scaled_Omega_P);//Adding  Proportional.
  Vector_Scale(&Scaled_Omega_I[0],&errorYaw[0],Ki_YAW);//.00001Integrator
  Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);//adding integrator to the
Omega_I
}
/**************************************************/
/*
void Accel_adjust(void)
{
 Accel_Vector[1] += Accel_Scale(speed_3d*Omega[2]);  // Centrifugal force on Acc_y =
GPS_speed*GyroZ
 Accel_Vector[2] -= Accel_Scale(speed_3d*Omega[1]);  // Centrifugal force on Acc_z =
GPS_speed*GyroY
}
*/
/**************************************************/


void Matrix_update(void)
```

```
{
  Gyro_Vector[0]=Gyro_Scaled_X(read_adc(0)); //gyro x roll
  Gyro_Vector[1]=Gyro_Scaled_Y(read_adc(1)); //gyro y pitch
  Gyro_Vector[2]=Gyro_Scaled_Z(read_adc(2)); //gyro Z yaw

  Accel_Vector[0]=accel_x;
  Accel_Vector[1]=accel_y;
  Accel_Vector[2]=accel_z;

  Vector_Add(&Omega[0], &Gyro_Vector[0], &Omega_I[0]);  //adding proportional
term
  Vector_Add(&Omega_Vector[0], &Omega[0], &Omega_P[0]); //adding Integrator
term

 //Accel_adjust();    //Remove centrifugal acceleration.   We are not using this function
in this version - we have no speed measurement

 #if OUTPUTMODE==1
  Update_Matrix[0][0]=0;
  Update_Matrix[0][1]=-G_Dt*Omega_Vector[2];//-z
  Update_Matrix[0][2]=G_Dt*Omega_Vector[1];//y
  Update_Matrix[1][0]=G_Dt*Omega_Vector[2];//z
  Update_Matrix[1][1]=0;
  Update_Matrix[1][2]=-G_Dt*Omega_Vector[0];//-x
  Update_Matrix[2][0]=-G_Dt*Omega_Vector[1];//-y
  Update_Matrix[2][1]=G_Dt*Omega_Vector[0];//x
  Update_Matrix[2][2]=0;
 #else                   // Uncorrected data (no drift correction)
```

```
  Update_Matrix[0][0]=0;
  Update_Matrix[0][1]=-G_Dt*Gyro_Vector[2];//-z
  Update_Matrix[0][2]=G_Dt*Gyro_Vector[1];//y
  Update_Matrix[1][0]=G_Dt*Gyro_Vector[2];//z
  Update_Matrix[1][1]=0;
  Update_Matrix[1][2]=-G_Dt*Gyro_Vector[0];
  Update_Matrix[2][0]=-G_Dt*Gyro_Vector[1];
  Update_Matrix[2][1]=G_Dt*Gyro_Vector[0];
  Update_Matrix[2][2]=0;
 #endif

  Matrix_Multiply(DCM_Matrix,Update_Matrix,Temporary_Matrix); //a*b=c

  for(int x=0; x<3; x++) //Matrix Addition (update)
  {
    for(int y=0; y<3; y++)
    {
      DCM_Matrix[x][y]+=Temporary_Matrix[x][y];
    }
  }
}

void Euler_angles(void)
{
  pitch = -asin(DCM_Matrix[2][0]);
  roll = atan2(DCM_Matrix[2][1],DCM_Matrix[2][2]);
  yaw = atan2(DCM_Matrix[1][0],DCM_Matrix[0][0]);
}
```

I2C.pde

```
/* ****************************************************** */
/* I2C code for ADXL345 accelerometer                      */
/* and HMC5843 magnetometer                               */
/* ****************************************************** */

int AccelAddress = 0x53;
int CompassAddress = 0x1E;  //0x3C //0x3D;  //(0x42>>1);

void I2C_Init()
{
  Wire.begin();
}

void Accel_Init()
{
  Wire.beginTransmission(AccelAddress);
  Wire.send(0x2D);  // power register
  Wire.send(0x08);  // measurement mode
  Wire.endTransmission();
  delay(5);
  Wire.beginTransmission(AccelAddress);
  Wire.send(0x31);  // Data format register
  Wire.send(0x08);  // set to full resolution
  Wire.endTransmission();
  delay(5);
 // Because our main loop runs at 50Hz we adjust the output data rate to 50Hz (25Hz
bandwidth)
  Wire.beginTransmission(AccelAddress);
  Wire.send(0x2C);  // Rate
  Wire.send(0x09);  // set to 50Hz, normal operation
  Wire.endTransmission();
  delay(5);
}

// Reads x,y and z accelerometer registers
void Read_Accel()
{
  int i = 0;
  byte buff[6];

  Wire.beginTransmission(AccelAddress);
  Wire.send(0x32);        //sends address to read from
  Wire.endTransmission(); //end transmission

  Wire.beginTransmission(AccelAddress); //start transmission to device
  Wire.requestFrom(AccelAddress, 6);    // request 6 bytes from device

  while(Wire.available())   // ((Wire.available())&&(i<6))
  {
   buff[i] = Wire.receive();  // receive one byte
   i++;
  }
  Wire.endTransmission(); //end transmission

  if (i==6)  // All bytes received?
   {
   ACC[1] = (((int)buff[1]) << 8) | buff[0];    // Y axis (internal sensor x axis)
   ACC[0] = (((int)buff[3]) << 8) | buff[2];    // X axis (internal sensor y axis)
```

```
    ACC[2] = (((int)buff[5]) << 8) | buff[4];    // Z axis
    AN[3] = ACC[0];
    AN[4] = ACC[1];
    AN[5] = ACC[2];
    accel_x = SENSOR_SIGN[3]*(ACC[0]-AN_OFFSET[3]);
    accel_y = SENSOR_SIGN[4]*(ACC[1]-AN_OFFSET[4]);
    accel_z = SENSOR_SIGN[5]*(ACC[2]-AN_OFFSET[5]);
    }
  else
    Serial.println("!ERR: Acc data");
}
void Compass_Init()
{
  Wire.beginTransmission(CompassAddress);
  Wire.send(0x02);
  Wire.send(0x00);   // Set continouos mode (default to 10Hz)
  Wire.endTransmission(); //end transmission
}
void Read_Compass()
{
  int i = 0;
  byte buff[6];
   Wire.beginTransmission(CompassAddress);
  Wire.send(0x03);        //sends address to read from
  Wire.endTransmission(); //end transmission
   //Wire.beginTransmission(CompassAddress);
  Wire.requestFrom(CompassAddress, 6);    // request 6 bytes from device
  while(Wire.available())   // ((Wire.available())&&(i<6))
  {
    buff[i] = Wire.receive();  // receive one byte
    i++;
  }
  Wire.endTransmission(); //end transmission
   if (i==6)  // All bytes received?
    {
   // MSB byte first, then LSB, X,Y,Z
    magnetom_x = SENSOR_SIGN[6]*((((int)buff[2]) << 8) | buff[3]);    // X axis (internal
sensor y axis)
    magnetom_y = SENSOR_SIGN[7]*((((int)buff[0]) << 8) | buff[1]);    // Y axis (internal
sensor x axis)
    magnetom_z = SENSOR_SIGN[8]*((((int)buff[4]) << 8) | buff[5]);    // Z axis
    }
  else
    Serial.println("!ERR: Mag data");
}
```

matrix.pde

```
/**************************************************/
//Multiply two 3x3 matrixs. This function developed by Jordi can be easily adapted to
multiple n*n matrix's. (Pero me da flojera!).
void Matrix_Multiply(float a[3][3], float b[3][3],float mat[3][3])
{
  float op[3];
  for(int x=0; x<3; x++)
  {
    for(int y=0; y<3; y++)
    {
      for(int w=0; w<3; w++)
      {
       op[w]=a[x][w]*b[w][y];
      }
      mat[x][y]=0;
      mat[x][y]=op[0]+op[1]+op[2];

      float test=mat[x][y];
    }
  }
}
```

Output.pde

```
void printdata(void)
{
       Serial.print("!");

       #if PRINT_EULER == 1
       Serial.print("ANG:");
       Serial.print(ToDeg(roll));
       Serial.print(",");
       Serial.print(ToDeg(pitch));
       Serial.print(",");
       Serial.print(ToDeg(yaw));
       #endif
       #if PRINT_ANALOGS==1
       Serial.print(",AN:");
       Serial.print(AN[sensors[0]]);  //(int)read_adc(0)
       Serial.print(",");
       Serial.print(AN[sensors[1]]);
       Serial.print(",");
       Serial.print(AN[sensors[2]]);
       Serial.print(",");
```

```
 Serial.print(ACC[0]);
 Serial.print (",");
 Serial.print(ACC[1]);
 Serial.print (",");
 Serial.print(ACC[2]);
 Serial.print(",");
 Serial.print(magnetom_x);
 Serial.print (",");
 Serial.print(magnetom_y);
 Serial.print (",");
 Serial.print(magnetom_z);
 #endif
/*#if PRINT_DCM == 1
 Serial.print (",DCM:");
 Serial.print(convert_to_dec(DCM_Matrix[0][0]));
 Serial.print (",");
 Serial.print(convert_to_dec(DCM_Matrix[0][1]));
 Serial.print (",");
 Serial.print(convert_to_dec(DCM_Matrix[0][2]));
 Serial.print (",");
 Serial.print(convert_to_dec(DCM_Matrix[1][0]));
 Serial.print (",");
 Serial.print(convert_to_dec(DCM_Matrix[1][1]));
 Serial.print (",");
 Serial.print(convert_to_dec(DCM_Matrix[1][2]));
 Serial.print (",");
 Serial.print(convert_to_dec(DCM_Matrix[2][0]));
 Serial.print (",");
 Serial.print(convert_to_dec(DCM_Matrix[2][1]));
 Serial.print (",");
```

```
      Serial.print(convert_to_dec(DCM_Matrix[2][2]));
      #endif*/
      Serial.println();

}

long convert_to_dec(float x)
{
  return x*10000000;
}
```

Vector.pde

```
//Computes the dot product of two vectors
float Vector_Dot_Product(float vector1[3],float vector2[3])
{
  float op=0;

  for(int c=0; c<3; c++)
  {
  op+=vector1[c]*vector2[c];
  }
   return op;
}
//Computes the cross product of two vectors
void Vector_Cross_Product(float vectorOut[3], float v1[3],float v2[3])
{
  vectorOut[0]= (v1[1]*v2[2]) - (v1[2]*v2[1]);
  vectorOut[1]= (v1[2]*v2[0]) - (v1[0]*v2[2]);
  vectorOut[2]= (v1[0]*v2[1]) - (v1[1]*v2[0]);
}
```

```
//Multiply the vector by a scalar.
void Vector_Scale(float vectorOut[3],float vectorIn[3], float scale2)
{
  for(int c=0; c<3; c++)
  {
   vectorOut[c]=vectorIn[c]*scale2;
  }
}
void Vector_Add(float vectorOut[3],float vectorIn1[3], float vectorIn2[3])
{
  for(int c=0; c<3; c++)
  {
     vectorOut[c]=vectorIn1[c]+vectorIn2[c];
  }
}
```

Appendix B

IMU_Razor9DOF.py interface with Vpython

Source: http://code.google.com/p/sf9domahrs/downloads/list

```
# Test for Razor 9DOF IMU
# Jose Julio @2009
# This script needs VPhyton, pyserial and pywin modules
# First Install Python 2.6.4
# Install pywin from http://sourceforge.net/projects/pywin32/
# Install pyserial from http://sourceforge.net/projects/pyserial/files/
# Install Vphyton from http://vpython.org/contents/download_windows.html

from visual import *
import serial
import string
import math
from time import time
grad2rad = 3.141592/180.0
# Check your COM port and baud rate
ser = serial.Serial(port='COM9',baudrate=57600, timeout=1)

# Main scene
scene=display(title="9DOF Razor IMU test")
scene.range=(1.2,1.2,1.2)
#scene.forward = (0,-1,-0.25)
scene.forward = (1,0,-0.25)
scene.up=(0,0,1)

# Second scene (Roll, Pitch, Yaw)
scene2 = display(title='9DOF Razor IMU test',x=0, y=0, width=500,
height=200,center=(0,0,0), background=(0,0,0))
scene2.range=(1,1,1)
scene.width=500
scene.y=200
```

```
scene2.select()

#Roll, Pitch, Yaw
cil_roll = cylinder(pos=(-0.4,0,0),axis=(0.2,0,0),radius=0.01,color=color.red)
cil_roll2 = cylinder(pos=(-0.4,0,0),axis=(-0.2,0,0),radius=0.01,color=color.red)
cil_pitch = cylinder(pos=(0.1,0,0),axis=(0.2,0,0),radius=0.01,color=color.green)
cil_pitch2 = cylinder(pos=(0.1,0,0),axis=(-0.2,0,0),radius=0.01,color=color.green)
#cil_course = cylinder(pos=(0.6,0,0),axis=(0.2,0,0),radius=0.01,color=color.blue)
#cil_course2 = cylinder(pos=(0.6,0,0),axis=(-0.2,0,0),radius=0.01,color=color.blue)
arrow_course = arrow(pos=(0.6,0,0),color=color.cyan,axis=(-0.2,0,0),
shaftwidth=0.02, fixedwidth=1)

#Roll,Pitch,Yaw labels
label(pos=(-0.4,0.3,0),text="Roll",box=0,opacity=0)
label(pos=(0.1,0.3,0),text="Pitch",box=0,opacity=0)
label(pos=(0.55,0.3,0),text="Yaw",box=0,opacity=0)
label(pos=(0.6,0.22,0),text="N",box=0,opacity=0,color=color.yellow)
label(pos=(0.6,-0.22,0),text="S",box=0,opacity=0,color=color.yellow)
label(pos=(0.38,0,0),text="W",box=0,opacity=0,color=color.yellow)
label(pos=(0.82,0,0),text="E",box=0,opacity=0,color=color.yellow)
label(pos=(0.75,0.15,0),height=7,text="NE",box=0,color=color.yellow)
label(pos=(0.45,0.15,0),height=7,text="NW",box=0,color=color.yellow)
label(pos=(0.75,-0.15,0),height=7,text="SE",box=0,color=color.yellow)
label(pos=(0.45,-0.15,0),height=7,text="SW",box=0,color=color.yellow)
L1 = label(pos=(-0.4,0.22,0),text="-",box=0,opacity=0)
L2 = label(pos=(0.1,0.22,0),text="-",box=0,opacity=0)
L3 = label(pos=(0.7,0.3,0),text="-",box=0,opacity=0)

# Main scene objects
scene.select()

# Reference axis (x,y,z)
```

```
arrow(color=color.green,axis=(1,0,0), shaftwidth=0.02, fixedwidth=1)
arrow(color=color.green,axis=(0,-1,0), shaftwidth=0.02 , fixedwidth=1)
arrow(color=color.green,axis=(0,0,-1), shaftwidth=0.02, fixedwidth=1)

# labels
label(pos=(0,0,0.8),text="9DOF Razor IMU test",box=0,opacity=0)
label(pos=(1,0,0),text="X",box=0,opacity=0)
label(pos=(0,-1,0),text="Y",box=0,opacity=0)
label(pos=(0,0,-1),text="Z",box=0,opacity=0)

# IMU object
platform = box(length=1, height=0.05, width=1, color=color.red)
p_line = box(length=1,height=0.08,width=0.1,color=color.yellow)
plat_arrow = arrow(color=color.green,axis=(1,0,0), shaftwidth=0.06, fixedwidth=1)
f = open("Serial"+str(time())+".txt", 'w')
roll=0
pitch=0
yaw=0

while 1:
    line = ser.readline()
    line = line.replace("!ANG:","")   # Delete "!ANG:"
    print line
    f.write(line)                     # Write to the output log file
    words = string.split(line,",")    # Fields split

  if len(words) > 2:
        try:
            roll = float(words[0])*grad2rad
            pitch = float(words[1])*grad2rad
            yaw = float(words[2])*grad2rad
        except:
            print "Invalid line"
        axis=(cos(pitch)*cos(yaw),-cos(pitch)*sin(yaw),sin(pitch))
        up=(sin(roll)*sin(yaw)+cos(roll)*sin(pitch)*cos(yaw),sin(roll)*cos(yaw)-
cos(roll)*sin(pitch)*sin(yaw),-cos(roll)*cos(pitch))

        platform.axis=axis
        platform.up=up
        platform.length=1.0
        platform.width=0.65
        plat_arrow.axis=axis
        plat_arrow.up=up
        plat_arrow.length=0.8
        p_line.axis=axis
        p_line.up=up

        cil_roll.axis=(0.2*cos(roll),0.2*sin(roll),0)
        cil_roll2.axis=(-0.2*cos(roll),-0.2*sin(roll),0)
        cil_pitch.axis=(0.2*cos(pitch),0.2*sin(pitch),0)
        cil_pitch2.axis=(-0.2*cos(pitch),-0.2*sin(pitch),0)

        arrow_course.axis=(0.2*sin(yaw),0.2*cos(yaw),0)

        L1.text = str(float(words[0]))
        L2.text = str(float(words[1]))
        L3.text = str(float(words[2]))
ser.close
f.close
```

Appendix C

Warping image to top view code with C language

```
void WarpImg(float x, float y, float z, static bool cntRed)
        {
                capture = cvCreateCameraCapture(0);
                assert(capture);

                char ImgnamePer[10];
                float fx=CV_MAT_ELEM( *intrinsic_matrix, float, 0, 0 );
                float fy=CV_MAT_ELEM( *intrinsic_matrix, float, 1, 1 );
                float cx=CV_MAT_ELEM( *intrinsic_matrix, float, 0, 2 );
                float cy=CV_MAT_ELEM( *intrinsic_matrix, float, 1, 2 );
                IplImage *image = cvQueryFrame(capture);
                IplImage *warpingImg = cvCreateImage(cvSize(image->width,image-
>height),IPL_DEPTH_8U,3);
                Bitmap^ outImage1 = gcnew Bitmap(image->width,image-
>height,image-
>widthStep,System::Drawing::Imaging::PixelFormat::Format24bppRgb,System::IntPt
r(image->imageData));

                this->inputImage->Image = gcnew Bitmap(outImage1);
                IplImage* Img = image;
                IplImage* Img = Undistort(image);
                //Roll&Pitch
                XangleD =y-90.0;
                ZangleD = x;

                xAngle          = (float)(PI/180)*XangleD;
                yAngle          = (float)(PI/180)*YangleD;
                zAngle          = (float)(PI/180)*ZangleD;

r11=cos(yAngle)*cos(zAngle);
r12=cos(zAngle)*sin(xAngle)*sin(yAngle)-cos(xAngle)*sin(zAngle);
r13=cos(xAngle)*cos(zAngle)*sin(yAngle)+sin(xAngle)*sin(zAngle);
```

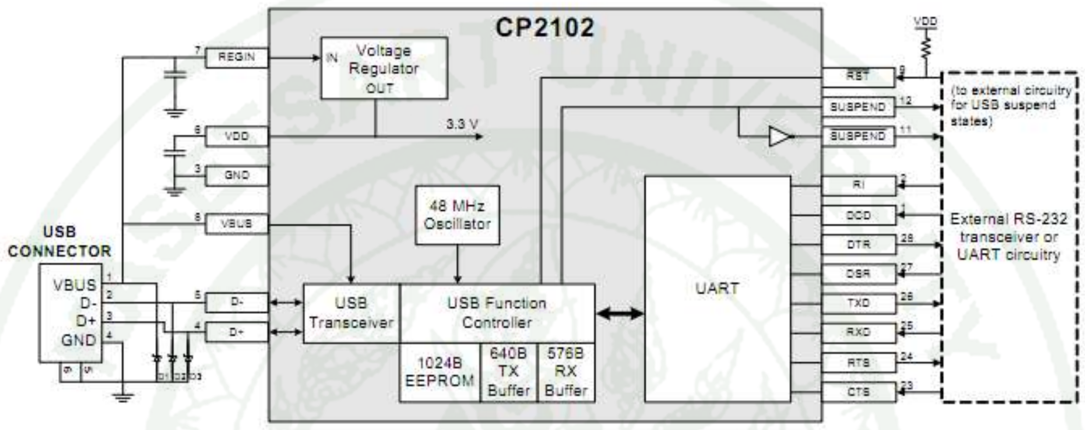
```
r21=cos(yAngle)*sin(zAngle);
r22=sin(zAngle)*sin(xAngle)*sin(yAngle)+cos(xAngle)*cos(zAngle);
r23=cos(xAngle)*sin(yAngle)*sin(zAngle)-cos(zAngle)*sin(xAngle);
r31=-sin(yAngle);
r32=cos(yAngle)*sin(xAngle);
r33=cos(xAngle)*cos(yAngle);
t1=0;
t2=0;
t3=0.5*fx;

            CV_MAT_ELEM( *T, float, 0, 0 )=1;
            CV_MAT_ELEM( *T, float, 0, 1 )=0;
            CV_MAT_ELEM( *T, float, 0, 2 )=-cx;
            CV_MAT_ELEM( *T, float, 1, 0 )=0;
            CV_MAT_ELEM( *T, float, 1, 1 )=1;
            CV_MAT_ELEM( *T, float, 1, 2 )=-cy;
            CV_MAT_ELEM( *T, float, 2, 0 )=0;
            CV_MAT_ELEM( *T, float, 2, 1 )=0;
            CV_MAT_ELEM( *T, float, 2, 2 )=1;


            CV_MAT_ELEM( *H, float, 0, 0 )=(fx*r11 + cx*r31);
            CV_MAT_ELEM( *H, float, 0, 1 )=(fx*r12 + cx*r32);
            CV_MAT_ELEM( *H, float, 0, 2 )=(fx*t1 + cx*t3);
            CV_MAT_ELEM( *H, float, 1, 0 )=(fy*r21 + cy*r31);
            CV_MAT_ELEM( *H, float, 1, 1 )=(fy*r22 + cy*r32);
            CV_MAT_ELEM( *H, float, 1, 2 )=(fy*t2 + cy*t3);
            CV_MAT_ELEM( *H, float, 2, 0 )=r31;
            CV_MAT_ELEM( *H, float, 2, 1 )=r32;
            CV_MAT_ELEM( *H, float, 2, 2 )=t3;

            H2=MulMat3x3(H,T);
```

```
		cvWarpPerspective(Img,warpingImg,H2,CV_INTER_LINEAR |
CV_WARP_INVERSE_MAP | CV_WARP_FILL_OUTLIERS);

if (cntRed==true)
		{
			CountPixel(warpingImg,y);
			itoa((int)y,ImgnamePer,10);
			strcat( ImgnamePer,"Per");
			strcat( ImgnamePer,".bmp");
			cvSaveImage( ImgnamePer,Img);
		}
		Bitmap^ outImage2 = gcnew Bitmap(warpingImg-
>width,warpingImg->height,warpingImg-
>widthStep,System::Drawing::Imaging::PixelFormat::Format24bppRgb,System::IntPt
r(warpingImg->imageData));
		this->outputImage->Image = gcnew Bitmap(outImage2);

		cvReleaseImage(&Img);
		cvReleaseImage(&warpingImg);

	}
```

Appendix D

CP210x USB to UART Bridge VCP Drivers

The CP210x USB to UART Bridge Virtual COM Port (VCP) drivers are required for device operation as a Virtual COM Port to facilitate host communication with CP210x products. These devices can also interface to a host using the USBXpress direct access driver.



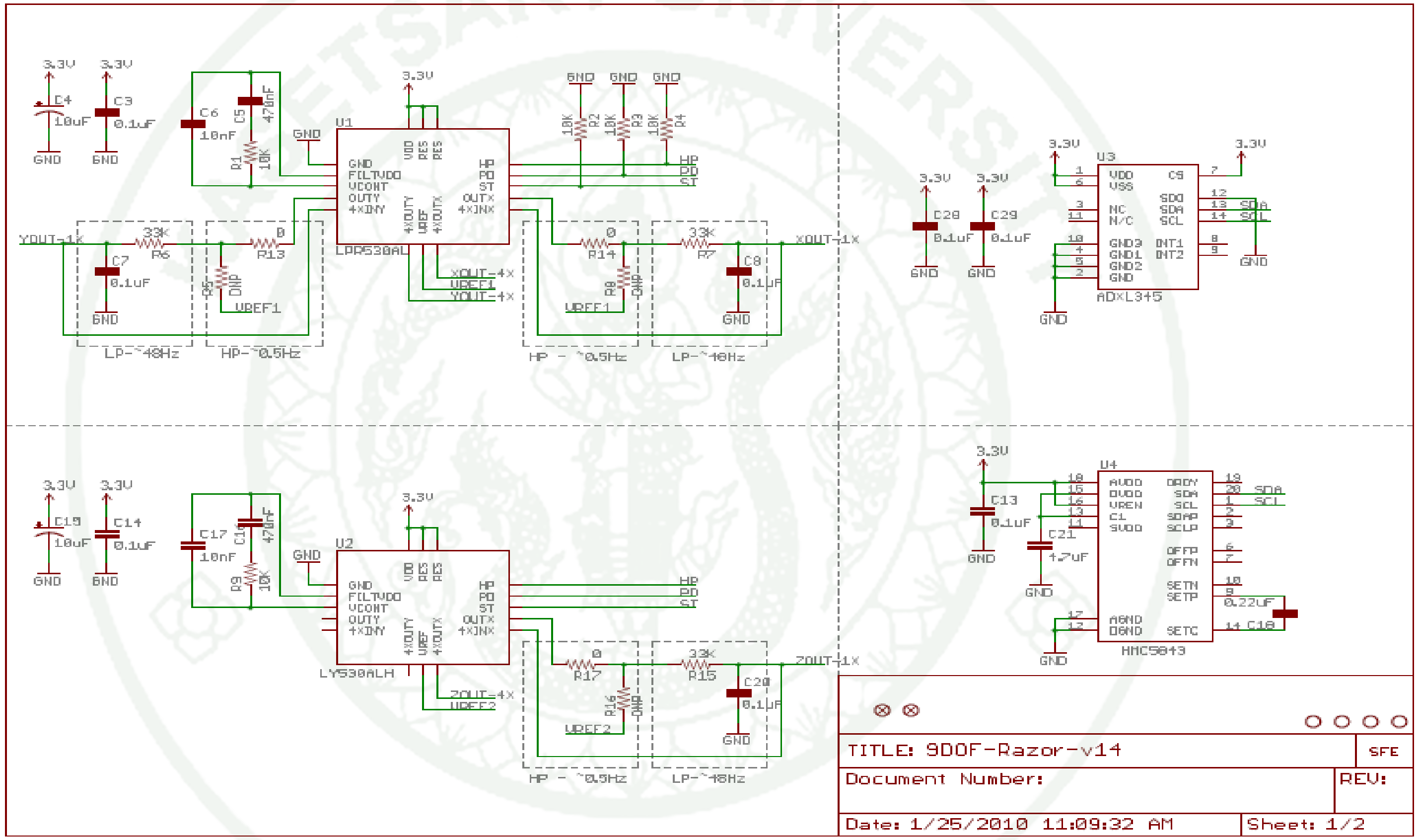
Appendix Figure D1 The CP2102 system diagram

Source: Silicon labs datasheet

The CP2102 is a highly-integrated USB-to-UART Bridge Controller providing a simple solution for updating RS-232 designs to USB using a minimum of components and PCB space. The CP2102 includes a USB 2.0 full-speed function controller, USB transceiver, oscillator, EEPROM, and asynchronous serial data bus (UART) with full modem control signals in a compact 5 x 5 mm QFN-28 package. No other external USB components are required. The on-chip EEPROM may be used to customize the USB Vendor ID, Product ID, Product Description String, Power Descriptor, Device Release Number, and Device Serial Number as desired for OEM applications.

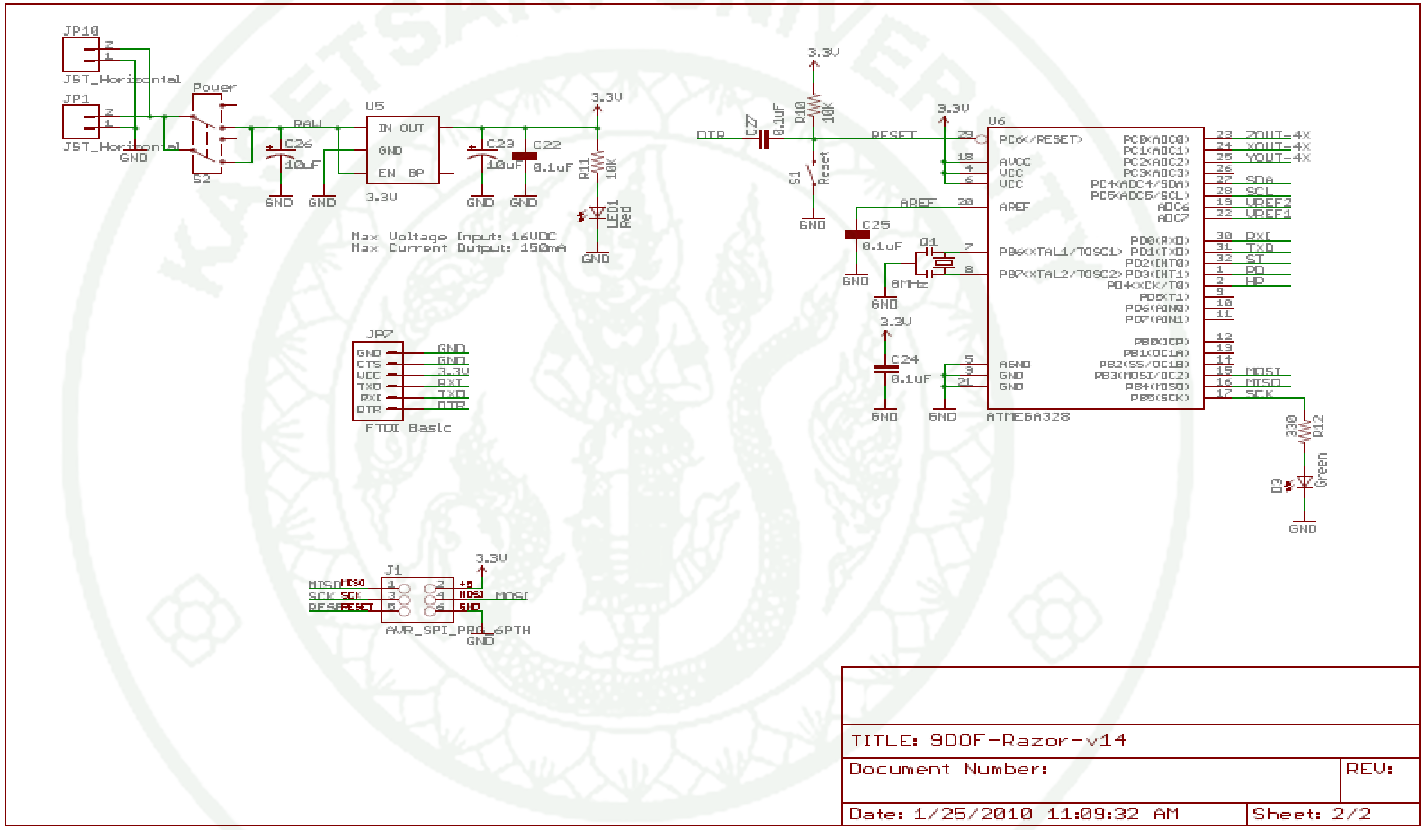
Appendix E

9DOF Schematic



Appendix Figure E1 9DOF Razor v14 schematic**:** LPRS530AL, ADXL345, LY530ALM, HMC5043.

Source: Silicon labs datasheet



Appendix Figure E2 9DOF Razor v14 schematic**:** ATmega328**.**

Source: Silicon labs datasheet

CIRRICULUM VITAE

NAME	:	Ms. Surangrak Sutiworwan
BIRTH DATE	:	January 23, 1988
BIRTH PLACE	:	Bangkok, Thailand

EDUCATION :

YEAR	INSTITUTE	DEGREE/DIPLOMA
2009	Mahidol Univ.	B.Sc. (Information and Communication Technology)
2011	Kasetsart Univ.	M.Eng. (Information and Communication Technology for Embedded Systems)

POSITION/TITLE	:	-
WORK PLACE	:	-
SCHOLARSHIP/AWARDS	:	TAIST ICTES Master Degree Scholarship
PUBLICATIONS		CIT2011, ICICTES2012, IEEE-CYBER 2012