A SMART COST-EFFECTIVE REAL-TIME OUTDOOR CAR PARKING SPACE DETECTION USING VIDEO PROCESSING

Somyod Saewong

A Thesis Submitted in Fulfillment of Requirements
for the Degree of Master of Engineering Program in Engineering Technology
Graduate School
Thai-Nichi Institute of Technology
Academic Year 2013

Thesis Title	A Smart Cost-Effective Real-Time Outdoor Car Parking Space Detection Using Video Processing
By	Somyod Saewong
Field of Study	Engineering Technology
Advisor	Asst. Prof. Dr. Wimol San-Um

The Graduate School of Thai-Nichi Institute of Technology has been approved and accepted as partial fulfillment of the requirement for the Master's Degree.

..... Dean of Graduate School
(Assoc. Prof. Dr. Pichit Sukchareonpong)
Month ……..... Date ....., Year.....

Thesis Committees

..... Chairperson
(Assoc. Prof. Dr. Worapong Tangsrirat)

..... Committee
(Asst. Prof. Dr. Warakorn Srichavengsup)

..... Committee
(Dr. Kanticha Kittipeerachon)

..... Advisor
(Asst. Prof. Dr. Wimol San-Um)

SOMYOD SAEWONG : A SMART COST-EFFECTIVE REAL-TIME OUTDOOR CAR PARKING SPACE DETECTION USING VIDEO PROCESSING. ADVISOR : ASST. PROF. DR. WIMOL SAN-UM, 140 PP.

The existing outdoor car parks do not have the systematic system. All of them are managed by human and ineffective therefore problems were occurred. The critical problem of the outdoor parking is wasting time in finding available parking lot. The drivers have to drive around the car park until they found the available parking lot. The problem always occurs in the city areas. Where the number of cars are higher than the number of parking lots. This problem is remaining because the technologies were overlooking. Various intelligent systems have been done to facilitate the traffic in the car parks. The old manual systems were change to be computer automatic system. An operator and the entrance in the old systems are replaced by the barrier gates and automatics tickets for the assessment. With the innovative of the technology, these systems have been applied in differences purpose.

Therefore, the research focuses on applying technology on the outdoor parking system, especially using the image processing to detecting the status of the parking lots in the car park is a cost effective solution. The outdoor car park consists of variety factors need to be considered. The proper object detection techniques are required for the detection accuracy in difference car park conditions. Hence, a smart cost effective real time outdoor car parking space detection using video processing is valuable.

The research was carried out by using MATLAB programming software together with display hardware. Numbers of image processing techniques were applied to improve the system efficiency. The system was included input module, three detection modules and output module. Experiments were done on prototype and actual car parks in differences conditions.

Graduate School Student's Signature .....
Field of Engineering Technology Advisor's Signature .....
Academic Year 2013

Acknowledgements

The author wishes to express his profound gratitude and to respectfully dedicate this work to his parent and family members for their endless encouragements, love and sacrifices. The author is most grateful to his advisor, Asst. Prof. Dr. Wimol San-Um, for his valuable supervision, supports and encouragements throughout the study. Grateful acknowledgements are also made to Asst. Prof. Dr. Wipawadee Wongsuwan, the director of master of engineering technology program for her supports. In addition, grateful acknowledgements are made to Asst. Prof. Dr. Worapong Tangsrirat, Dr. Kanticha Kittipeerachon, Asst. Prof. Dr. Warakorn Srichavengsup, members of thesis committee, for their valuable suggestions and comments. The author also wishes to acknowledge Thai-Nichi Institute of Technology for a great opportunity of master study. The author sincerely appreciates all of his friends in Intelligent Electronic Systems Research laboratory for their friendships and good will.

Somyod Saewong

Table of Contents

Table Contents (Continued.)

List of Table

List of Figures

List of Figures (Continued)

Chapter 1
Introduction

1.1 Background

Digital Image processing is the technique of using computer algorithms to process the digital image. One of the particularly interesting application is an analysis of visual objects in images is a very important component in computer vision systems which perform object recognition, image retrieval, and image registration. Areas where such systems are deployed are diverse and include such applications as surveillance, video forensics, and medical image analysis for computer-aided diagnosis. The object recognition problem has attracted much attention recently due to the increasing demand for developing real-world systems. Generally, recognition is mainly divided into two parts: recognition and detection. The goal of object category recognition is to classify a given object into one of the several prespecified categories, while object detection is to separate objects of interest from the background.

Specifically, extracting the points from an image that can give best defines from an object in image namely keypoints is very important and valuable. These points have many applications in image processing like object detection, object and shape recognition, image registration and object tracking. By extracting the keypoints, we can use them for finding objects in the other images. Detect and recognize the object by using the keypoints and a segmentation algorithm are very accurate because if the keypoints are correctly identified, they achieve the best information.

Of particular interest in object detection in digital images, current approaches to object detection can be categorized by top-down, bottom-up or combination of the two. Top-down approaches often include a training stage to obtain class-specific model features or to define object configurations. Hypotheses are found by matching models to the image features. Bottom-up approaches start from low-level or mid-level image features, i.e. edges or segments. These methods build up hypotheses from such features, extend them by construction rules and then evaluate by certain cost functions. The third category of approaches combining top-down and bottom-up methods have become prevalent because they take advantage of both aspects.

Although top-down approaches can quickly drive attention to promising hypotheses, they are prone to produce many false positives when features are locally extracted and matched. Features within the same hypothesis may not be consistent with respect to low-level image segmentation. On the other hand, bottom-up approaches try to keep consistency in low level image segmentation, but usually need much more efforts in searching and grouping.

Based on the above information, this thesis focuses on the applications of image processing algorithm for object detection in digital images. The challenges are to detect the objects in static images. New algorithms will be developed for both real time object detection base on reference image and without reference image. The application under interest is the real-time monitoring of vacancy in car parking lots. The applicated program will also be developed. The expected outcomes involve smart and cost effective system with high efficiency.

1.2 Research Objective

1.2.1 To develop an algorithm for real-time object detections of digital images.

1.2.2 To implement the vacancy detection prototype system for outdoor car parking lots using the developed real-time object detections of digital image.

1.2.3 To implement smart and cost effective system for outdoor car parking lots vacancy detection.

1.3 Research Scope

The experiment is firstly conducted under laboratory scale. Second the experiment was applied in real environmental factors for function flexibility test. Figure 1.1 shows the scope of object detection algorithm using reference base. The camera will capture two images with different time, i.e. a current and a future one. The two pictures will be compared through image differencing algorithms and proceeded into a simple binary image. The resulting image will be converted into as inputs of object detection algorithm for space classifications. The output of object

detection will indicate the approximate key point of interest that shows the existence of object in the image.

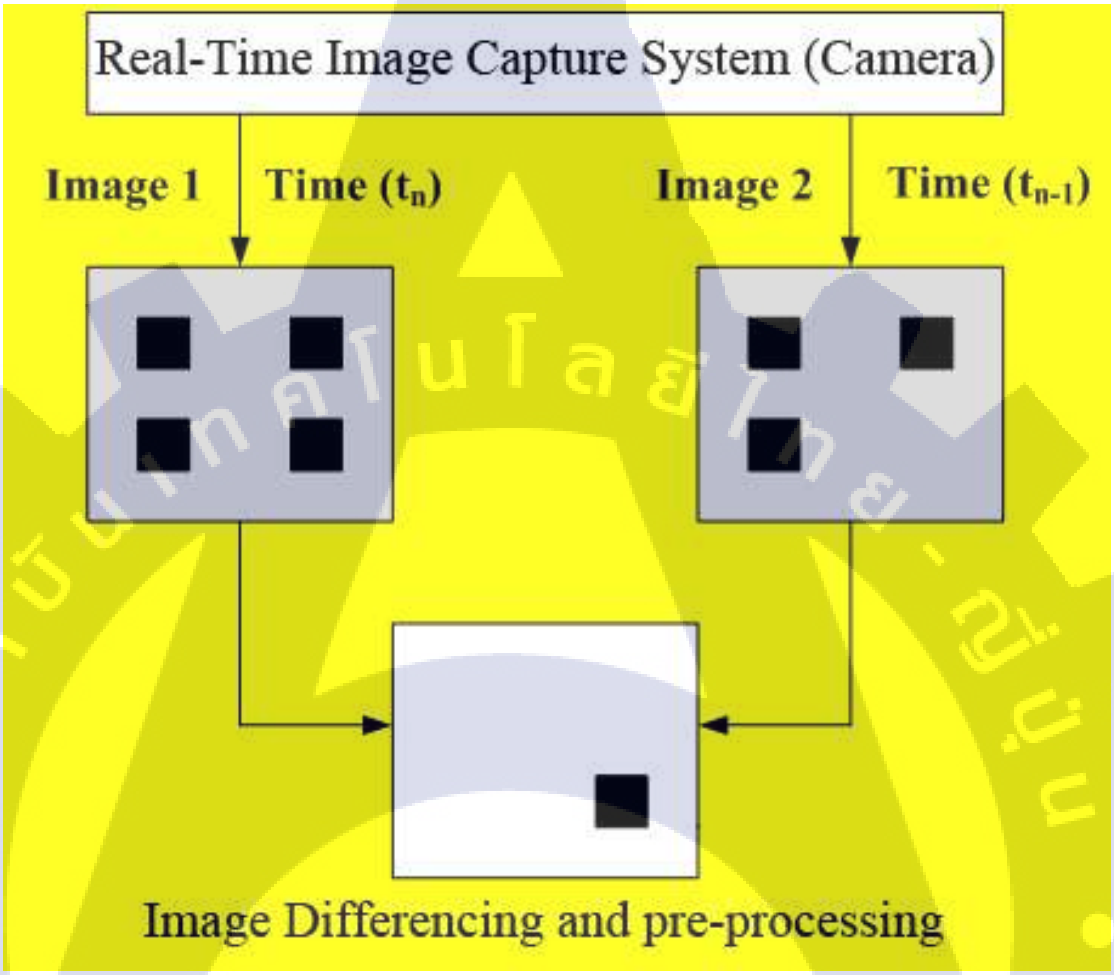


Figure 1.1 The scope of object detection algorithm using reference base.

Figure 1.2 shows the scope of the application of the proposed algorithm. The digital images from the civilian camera will be capture with different time, i.e. 1 minute for real-time monitoring system. The markets are cars that parked in the outdoor parking lots. The image will be transferred into a binary image as previously described in Figure 1.1. The vacancy of the parking space detected will be displayed as a program that shows lots available.

Figure 1.2 The scope of alienation in vacancy detection in car parking lots.

1.4 Expected Outcomes

1.4.1 Achieve an algorithm for real-time object detections of digital images.

1.4.2 Achieve the system for outdoor car parking lots vacancy detection.

1.4.3 Achieve the implementation of a smart and cost effective system for outdoor car parking lots vacancy detection.

1.5 Technical Term Declarations

1. Digital Image is electronic snapshot taken of a scene or scanned from document, such as photograph, manuscript, printed text, and artwork. The digital image is sampled and mapped as a grid of dots or picture elements (pixels). Each pixel is assigned a tonal value (black, white, shades of gray or color), which is represented in binary code (zeros and ones). The binary digits ("bits") for each pixel are stored in a sequence by a computer and often reduced to a mathematical representation (compressed). The bits are then interpreted and read by the computer to produce an analog version for display or printing.

2. Object Detection Algorithm is the process of finding instances of real-world objects such as faces, bicycles, and buildings in images or videos. Object Detection Algorithms typically use extracted features and learning algorithms to recognize instances of an object category. Object detection is commonly used in applications such as image retrieval, security, surveillance, and automated vehicle parking systems.

3. Region of Interest (ROI) is the user defined area on the input video snapshot. The area is known as the parking lot location on the snapshot. There are ten parking lots for the experiment car park model.

4. Light Emitting Diode Display (LED Display) is the set of LED to show the parking lots status. The LED set is consisting of ten pair of green and red LED to show the ten parking lots status.

5. Function is the pre-programmed code in MATLAB. There are numbers of functions were used in this research.

6. Module is the concept of operation function. Input module and output module are consisting of both hardware and software. Texture Extraction module, Histogram Differentiation module and Blob Analysis module are only software implementation.

1.6 Research Action Plan

Table 1.1 The research plan

Action Plan	Year 2556									Year 2557			
	Apr.	May.	Jun.	Jul.	Aug.	Sep.	Oct.	Nov.	Dec.	Jan.	Feb.	Mar.	Apr.
Topic proposal													
Review related research													
Present Thesis													
Experiment and collect data													
Conclusion of the experiment													
Preparing a thesis document													
Apply for thesis approval test													

Chapter 2
Literature Reviews

2.1 Digital Images

DIGITAL IMAGES are electronic snapshots taken of a scene or scanned from documents, such as photographs, manuscripts, printed texts, and artwork. The digital image is sampled and mapped as a grid of dots or picture elements (pixels). Each pixel is assigned a tonal value (black, white, shades of gray or color), which is represented in binary code (zeros and ones). The binary digits ("bits") for each pixel are stored in a sequence by a computer and often reduced to a mathematical representation (compressed). The bits are then interpreted and read by the computer to produce an analog version for display or printing. Pixel Values: Figure 2.1 shows an example of bitonal image in digital image; each pixel is assigned a tonal value, in this example 0 for black and 1 for white.

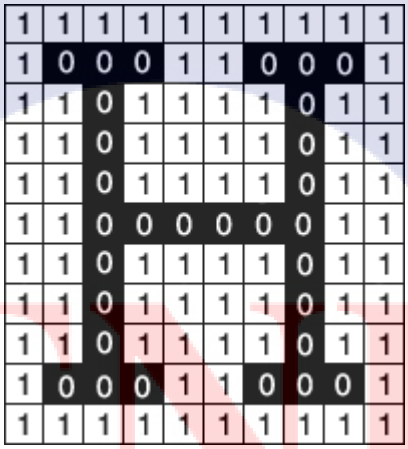


Figure 2.1 Example of Bitonal Image

2.2 Object Detections in Digital Images

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. Well-

researched domains of object detection include face detection and pedestrian detection. Object detection has applications in many areas of computer vision,

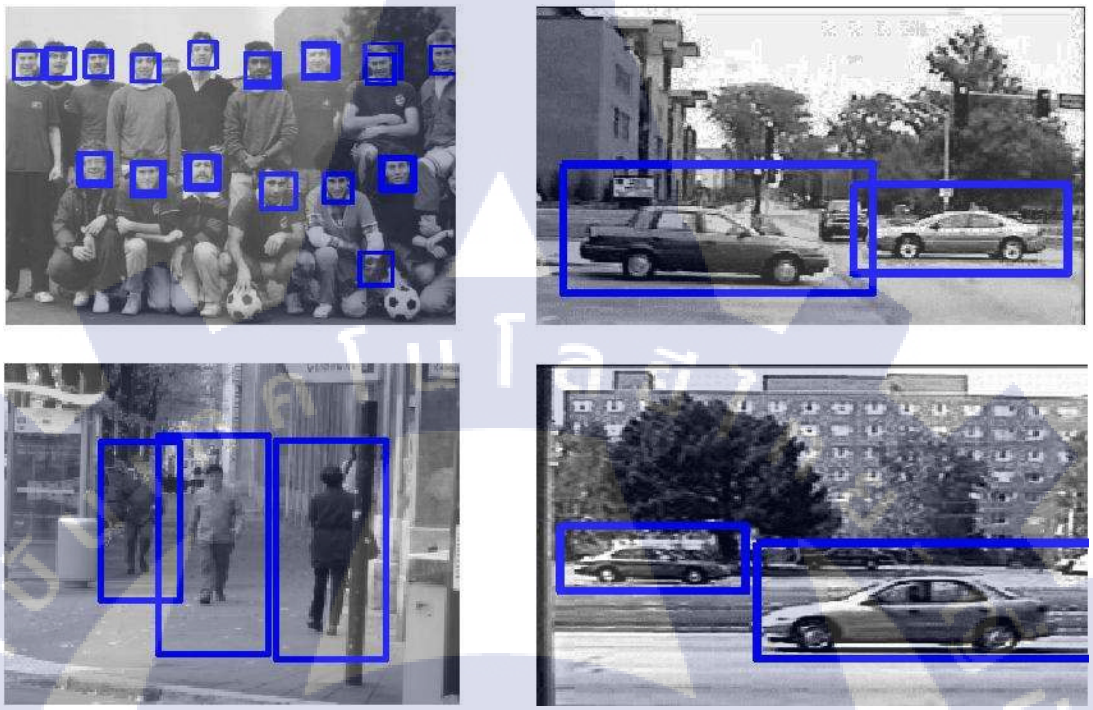


Figure 2.2 Example of object detection in digital images

including image retrieval and video surveillance. Figure 2.2 shows some examples of object detection in digital images. In general, Saliency detection on images has been studied for a long time. In recent years, many saliency detection methods have been designed because of its broad applications.

2.3 Related Research

Table 2.1 show the list of related research between year 2010 to 2014.

Table 2.1 List of related research

No.	Years	Authors	Proposed Techniques
1	2010	Ankit Sharma; and Nirbhowjap Singh	Object Detection In Image Using Particle Swarm Optimization
2	2010	Nibin Ghosha; and J.K.Mandal	Image authentication technique in frequency domain based on discrete Fourier transformation.
3	2010	Ching Chun Huang; and Sheng Jyh Wang	A Hierarchical Bayesian Generation Framework for Vacant Parking Space Detection
4	2010	H.Ichihashi; and Colleague	Improvement in the Performance of Camera Based Vehicle Detector for Parking Lot
5	2010	Li-Chih Chen; and Colleague	Vision-Based Vehicle Surveillance and Parking Lot Management Using Multiple Cameras
6	2011	C.NagaRaju; and Colleague	Morphological Edge Detection Algorithm Based on Multi-Structure Elements of Different Directions
7	2012	Reza Oji	An Automatic Algorithm for Object Recognition and Detection Based On ASIFT keypoints
8	2012	Ching-Chun Huang; and Colleague	A Surface-based Vacant Space Detection for an Intelligent Parking Lot

Table 2.1 List of related research (Continued)

No.	Years	Authors	Proposed Techniques
9	2012	Lih Lin; and Hong Siang	Vision-Based Activities Recognition by Trajectory Analysis for Parking Lot Surveillance
10	2012	Jermsak; and Colleague	Car Parking Vacancy Detection and Its Application in 24-Hour Statistical Analysis
11	2012	Wang Lixia; and Jiang Dalin	A method of Parking space detection based on image segmentation and LBP
12	2013	Diana Delibaltov; and Colleague	Parking Lot Occupancy Determination from Lamp-post Camera Images
13	2013	Guanglink Sun; and Colleague	Blurred image classification based on adaptive dictionary
14	2013	Kairoek Choeychuen	Automatic parking lot mapping for available parking space detection

Ankit Sharma; and Nirbhowjap Singh [1] had presented the research entitled "Object Detection in Image Using Particle Swarm Optimization". The authors had proposed the PSO based methods to solve the object detection problem. The PSO based algorithm has superior features, including high-quality solution, stable convergence characteristic and good computation efficiency. The results show that the proposed method is capable of obtaining higher quality solution efficiency. It is clear from the results that the proposed PSO based method can avoid the shortcoming of old template matching algorithm and can provide higher quality solution with better computation efficiency.

Nibin Ghosha; and J.K.Mandal [2] had presented the research entitled “Image Authentication Technique Technology in Frequency Domain based on Discrete Fourier Transfromation”. The authors had proposed IATFDDFT technique for image authentication process in frequency domain to enhance the security compared to the existing algorithms. In compare to DCT and QFT based watermarking technique, the proposed algorithm is applicable for any type of color images authentication and strength is high. First frequency component in each mask is used for re-adjusting. The control technique is applied to optimized the noise addition as a result PSNR is increased with low MSE and IF is nearer to 1. In the proposed IATFDDFT authentication is done in frequency domain without changing visual property of the authenticated image. In IATFDDFT distortion of image and change of fidelity is negligible.

Ching Chun Huang; and Sheng Jyh Wang [3] had presented the research entitled “A Hierarchical Bayesian Generation Framework for Vacant Parking Space Detection”. The authors had proposed a three layer Bayesian hierarchical framework (BHF) for robust vacant parking space detection. In practice, the challenges of vacant parking space inference come from dramatic luminance variations, shadow effect, perspective distortion, and the inter-occlusion among vehicles. By using a hidden labeling layer between an observation layer and a scene layer, the BHF provides a systematic generative structure to model these variations. In the proposed BHF, the problem of luminance variations is treated as a color classification problem and is tackled via a classification process from the observation layer to the labeling layer, while the occlusion pattern, perspective distortion, and shadow effect are well modeled by the relationships between the scene layer and the labeling layer. With the BHF scheme, the detection of vacant parking spaces and the labeling of scene status are regarded as a unified Bayesian optimization problem subject to a shadow generation model, an occlusion generation model, and an object classification model. The system accuracy was evaluated by using outdoor parking lot videos captured from morning to evening. Experimental results showed that the proposed framework can systematically determine the vacant space number, efficiently label ground and car regions, precisely locate the shadowed regions, and effectively tackle the problem of luminance variations.

H.Ichihashi; and Colleague [4] had presented the research entitled “Improvement in the Performance of Camera Based Vehicle Detectino for Parking Lot”. The research had reports on the performance of the detector based on the fuzzy c-means (FCM) clustering and the hyperparameter tuning by particle swarm optimization (PSO). The new system was introduced to an underground parking lot in Tokyo in early October 2009 and achieved the detection rate (sensitivity/specificity) of 99.9%. The system was also tested at an outdoor (rooftop) parking lot for a period of two months and achieved 99.6%. The performance clearly surpassed the initial goal of the project. In terms of classification accuracy, the FCM classifier is better than the support vector machine (SVM) and the computation time for training is an order of magnitude smaller than that of SVM.

Li-Chih Chen; and Colleague [5] had presented the research entitled “Vision-Based Vehicle Surveillance and Parking Lot Management Using Multiple Cameras”. The authors had proposed a vision-based vehicle surveillance system for parking lot management in outdoor environments. Due to the limited field of view of camera, this system uses multiple cameras for monitoring a wide parking area. Then, an affine transformation is used for merging the scenes obtained from these multiple cameras. Two major components are included, i.e., vehicle counting and parking lot management. For the first one, this paper integrates three features, i.e., color, position, and motion together for well tracking vehicles across different cameras. Thus, even though vehicles are occluded together, they still can be well tracked and identified across different cameras and under different lighting changes. For the second one, authors proposed a modelbased approach to model the color changes of parking ground for determining whether a parking space is vacant. Due to the perspective effects, the visibility of a parking space is often affected by the vehicle parking on its neighborhood. To tackle this problem, two geometrical models (ellipses and grids) are proposed for well representing a parking space. Then, with different weights, a hybrid scheme is then constructed for well determining whether a parking space is vacant. The experimental results reveal that the system works well and accurately under different lighting and occlusion conditions.

C. NagaRaju; and Colleague [6] had presented the research entitled “Morphological Edge Detection Algorithm Based on Multi-Structure Elements of

Different Directions". The authors had proposed a novel multi-structure elements morphological edge detection algorithm to detect image edge. The technique developed is very useful for Image segmentation and classification. The selection of structure element is a key factor in morphological image processing. The size and shape of SE decide the final result of detected edges. The basic theory of multi-structure elements morphology is to construct different structure elements in the same square window. And these structures elements comprise almost all the line extending directions in the square window. The given experimental results show that the algorithm is more efficient than the usually used single and symmetrical SE morphological edge detection operator and differential edge detection operators such as watershed method, Sobel operator and canny operator. The detected edge is more pinpointed, integral and continual, and the edge information is more abundant. Moreover, the novel proposed algorithm can filer the noise more successfully than other operators by high lighting brighter edges. Even though this method produces better results, it fails to shadow elimination of Images. The eight different edge detection results are obtained by using morphological gradient algorithm are better edges over traditional methods.

Jermsak Jermsurawong; and Colleague [7] had presented the research entitled "Car Parking Vacancy Detection and Its Application in 24-hour Statistical Analysis". The proposed solution uses trained neural networks to determine occupancy states based on features extracted from parking spots. This method addresses three technical problems. First, it responds to changing light intensity and non-uniformity by having adaptive reference pavement pixel value to calculate the color distance between the parking spots in question and the pavement. Second, it approximates images with limited lighting to have similar feature values to images with sufficient illumination, merging the two patterns. Third, the solution separately considers nighttime vacancy detection, choosing appropriate regions to get reference color value pixels. The presented method relies only few pixels compared with other methods, being able to cover large number of parking spots with a single camera. Moreover, the system is robust to changing light conditions and light non-uniformity due to shadow from the surrounding. The accuracy for this 24-hour period is 97.9% for empty spots. Besides giving accurate depiction of the car park's utilization rate,

this study also revealed the patterns of parking events at different time of the day and insights to the activities that car drivers engaged with.

Reza Oji [8] has presented the research entitled “An automatic algorithm for object recognition and detection based on ASIFT keypoints”. The paper presented an object recognition and detection algorithm. These targets are achieved by combining ASIFT and a region merging segmentation algorithm based on a similarity measure. The author trains different objects separately in several images with multiple aspects and camera viewpoints to find the best keypoints for recognizing them in the other images. These keypoints were applied to the region merging algorithm. The merging process is started by using keypoints and presented similarity measure. The regions were merged based on the merging role, and finally the object was detected well, with its boundary. A final conclusion is that the more keypoints are obtained, and the more accurate they are, the results will be better and more acceptable.

Ching-Chun Huang; and Colleague [9] had presented the research entitled “A Surface-based Vacant Space Detection for an Intelligent Parking Lot”. The authors had proposed a vacant parking space detection system working day and night. The obstacles come from occlusion effect, shadow effect, perspective distortion, and lighting change. To overcome the problems, the authors proposed the surface-based modeling that regards the parking lot as a structure consisting of plentiful surfaces. With the structure, the authors are able to import the texture information and 3-D scene information simultaneously for space detection. Experiments have shown that the approach performs well in the complicated parking lot environment in both day time and night time.

Lih Lin; and Colleague [10] had presented the research entitled “Vision-Based Activities Recognition by Trajectory Analysis for Parking Lot Surveillance”. The authors had proposed a novel event recognition framework in video surveillance system, particularly for parking lot environment. The proposed video surveillance system employs the adaptive Gaussian Mixture Model (GMM) and connected component analysis for background modeling and objects tracking. Spatial-temporal information of motion trajectories are extracted from video samples of known events to form representative feature vectors for event recognition purposes. An event is represented by feature vector that contains dynamic information of the motion

trajectory and the contextual information of the tracked object. The event classification is accomplished by measuring the similarity of the extracted feature vector to the labeled definition of known events and analyzing the contextual information of the detected event. Experiments have been carried out on the live video stream captured by the outdoor camera, and the results have demonstrated great accuracy of the proposed event recognition algorithm.

Wang Lixia; and Jiang Dalin [11] had presented the research entitled “A method of Parking space detection based on image segmentation and LBP”. The authors had proposed a parking spaces detection algorithm which is based on image segmentation and local binary pattern. The vehicles are usually contains a lot of compositions, while the vacant parking spaces’ composition is relatively small. According to this characteristic, authors segment the parking image. To judge whether each parking area has a large number of small split or not, can achieve the detection of the parking stalls. In the research, the authors improve the Mean Shift algorithm and achieve the accurate segmentation result. The proposed method was tested on indoor and outdoor parking lots. The result confirmed the efficiency of the proposed method, with the detection rate being over 97%. But this method fails to detect non-vehicle objects and when the vehicle color and ground color is very similar. Therefore the authors introduce the texture features, use LBP (local binary pattern to extract the parking texture feature. Using the complementary between features and ultimately to achieve accurate detection.

Guangling Sun; and Colleague [12] had presented the research entitled “Blurred Image Classification based on Adaptive Dictionary”. The authors had proposed two types of framework for blurred image classification and space-invariant blur kernel is assumed. The two frameworks are based on adaptive dictionary and neither demands image deblurring. The essential idea is that a new dictionary being capable of adaptive to inferred PSF from input blurred image is relearned for every input image. Therefore, for each blurred image patch, the sparse coefficient obtained by adaptive dictionary is insensitive to arbitrary blur. Meanwhile, for the two frameworks, the performance of the latter is higher than that of the former, since the former infers the PSF as a separate step, and the latter updates the PSF and sparse coefficient of gradient feature alternatively so as to better combine PSF estimation

and sparse coefficient calculation. The proposed framework can tackle any blur resulting from camera defocus, simple relative motion between camera and object, to camera shake.

Diana Delibaltov; and Colleague [13] had presented the research entitled “Parking Lot Occupancy Determination from Lamp-post Camera Images”. The authors had proposed a framework for the automatic detection of vacant parking spaces from a lamp-post camera view of a parking lot. Proposed method model the 3-D volume parking space based on the parking lot geometry. The occupancy of the parking lot is determined based on a vehicle detector and the inferred volume of each space. The authors evaluate their method on three different datasets and show that its accuracy is close to 80% on a wide variety of test images.

Kairoek Choeychuen [14] had presented the research entitled “Automatic parking lot mapping for available parking space detection”. The author had proposed a method to estimate map of parking lot for automatic system of available parking space detection. The parking space detection is important module for the parking guidance system (PGS) that can help drivers to find the parking space efficiently. Instead of masked parking slots by human, the author try to mask them by using histograms of spatial features. However, under poor image quality, the author faces dynamic change of background. Although the author have applied adaptive background model for background subtraction, the problem still remains. To overcome the problem, an automatic thresholding was proposed for tuning object (car) detection in adaptive background model method. The fine-tuning technique can improve accuracy of the histogram for the parking lot mapping. The experimental results were shown both simulation on the parking model and real parking lot correctly.

Chapter 3
Research Methodology

3.1 Process and Methodology

The researcher has defined the procedures and processes for the thesis research as the following detail.

(1) Use a camera to take pictures of the car park model.

(2) Use image processing to detect and identify the available parking slots.

(3) Record the result and display available parking slots on the LED display.

(4) Repeat the entire processes in differences situation.

(5) Identify the error and limitation then find the solution.

(6) Repeat the experiments with the new solution.

(7) Summarize the experiments then accomplish the thesis report.

3.2 Designing Experiment and Identifying the Variable

To achieving the expected outcome or be able to identifying the error that might be occur, equipment have to be set up same as in the hypothesis. The following are the example of error avoidance.

(1) The resolution of camera that use for the experiment have to match with the experimental scale.

(2) The camera has to be installing at the proper position and away from disturbance.

(3) The parking lot model is in the good condition for the experiment.

(4) Matlab is configure correctly for the experiment.

3.2.1 Experiment Procedure

(1) Take the first picture of the parking lot and then take the second picture in few second later.

(2) Put the two pictures in the Matlab and used the algorithm to detect the objects in the picture.

(3) Identify the available parking lots and show on the display.

3.2.2 Data Gathering.

Experimental data have to be gathering systematically for all the related variables. The time of taking each picture is very importance to the result. Brightness is very sensitive to the experimental result as same as the clarity of the pictures.

3.2.3 Date Analysis.

(1) The different object detection algorithm is suitable for different parking case.

(2) The available parking lot and actual parking lot need to be recorded correctly for verifying the result.

(3) Finally, analyze all the data and summarize the experiment.

Chapter 4
Results and Discussion

4.1 Results

The project consists of five modules and included three inherit techniques. The first module is input acquisition module; it is the initial step of the system. The input video and captured image of parking scene was in this module. The second module was Histogram Differentiation module. It used the differences of histogram between the reference image and the captured image from the camera as the methodology to identify the status of the parking lots. The Blob Analysis technique was used for the third module. The pixels' information of the image were analyzed and made the decision for the validity of the parking lots. The last technique that applied in the fourth module was Texture Extraction. This technique was based on Entropy Analysis. The differences in groups of pixels in the capture image were used to indicate the status of the parking lots. The fifth module was the output module. This module was finalized the outputs from the previous three module and show on the LED display. The diagram of the system module is illustrated in Figure 4.1

4.1.1 Input Module

The camera is assumed to be in a fixed position and facing a fixed direction all the time. The Figure 4.2 is shown the hardware setup of the system. The initialization process began with the video acquisition. Function "videoinput" was used to configure the camera and get the input video. Function "getsnapshot" was used for taking the snapshot of the input video signal to use as the reference image. The ROI (Region of Interest : ROI) or the pixels' location of ten parking lots in the reference image was analyzed and set into the system. The programmed software had defined ten variables which use for store the pixel coordination of the ten parking lots. The ten variables not only used by the Input module but also used by the three parking lots detection modules. The real time scene of the car park was then taken and makes a snapshot. The reference image and the snap shot of the input video were created a copy. The parking lots of the two copied image were cropped into pieces by using the

function “imcrop”. The size and dimension of cropped images was the same as the pre-defined pixel coordination of the parking lots or ROI in the previous step. The cropped images were then passed to Histogram Differentiation Module and Blob Analysis Module for the further image processing. Figure 4.3 shows the reference image and the cropped images of the ten parking lots in the reference image. Figure 4.4 shows the snap shot of real time parking and follow by the cropped image of each parking lot. The capture of real time parking scene was passed to the Texture Extraction module without modification to the image.

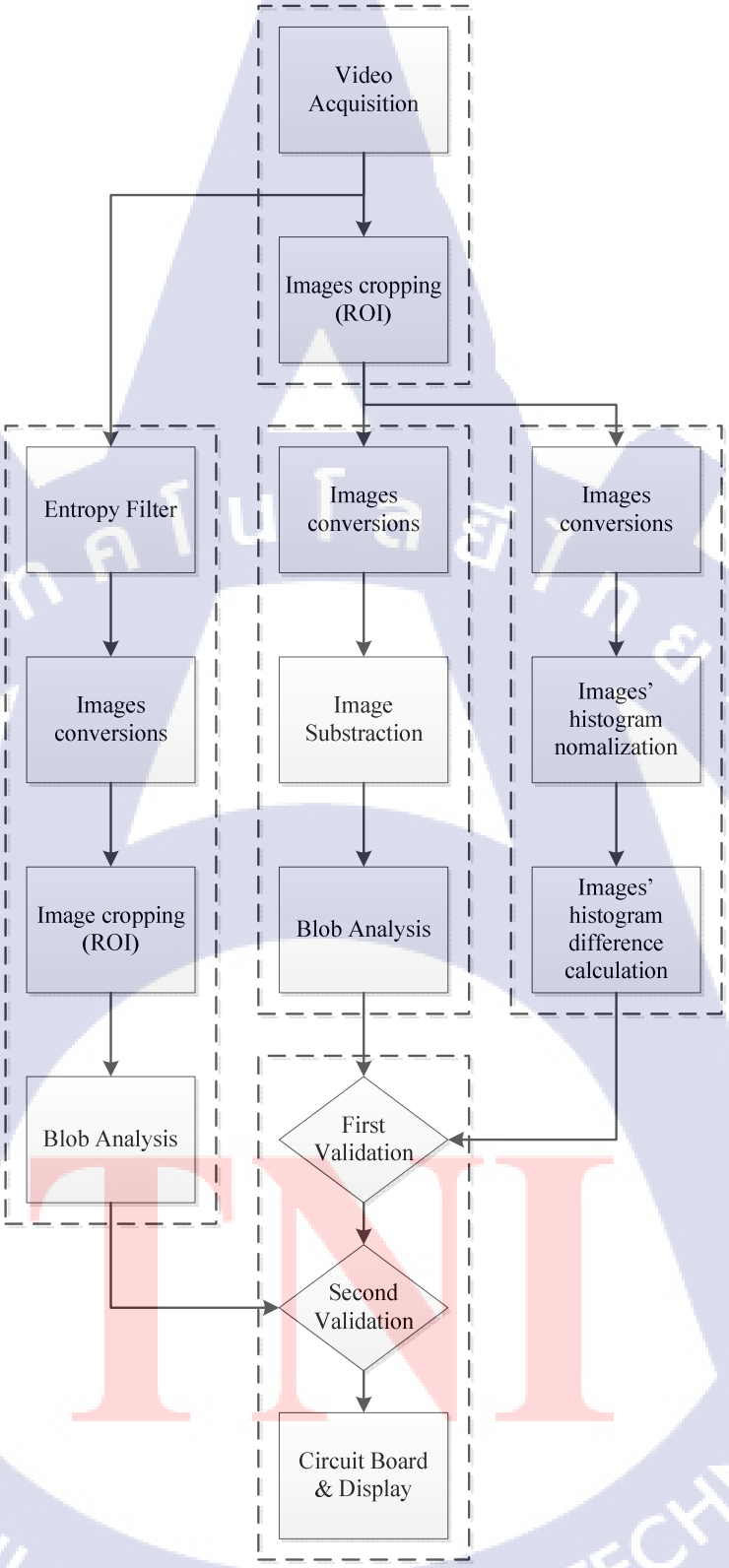


Figure 4.1 Diagram of the system module. Input (top), Texture Extraction (left), Histogram Differentiation (center), Blob Analysis (right), Output (bottom)

4.1.2 Histogram Differentiation Module

The output of the input module became the input of this module. The input cropped images then converted from RGB to be grey scale images by used the function “rgb2gray”. The histograms of all parking lots were generated for the normalizing process. The function for generate the histogram is call “imhist”. The gray scale histograms of each of the parking lots in the reference image were generated as shown in the Figure 4.5. The histograms of the parking lots in capture image from the input video were also generated. The histogram of each pair of the parking lot was normalized. The histogram of the each parking lot of the reference image was subtracted by the histogram of the same parking lot as in the capture image of input video. The subtraction result was compare to the standard value for making the decision of each parking lot status. The parking lots status was then passed to first validation process of the output module.

4.1.3 Blob Analysis Module

The Blob Analysis is suitable for detecting the object with reference base as the same as the propose system. Function “vision.BlobAnalysis” was applied for the Blob Analysis module is this research. The result of the subtraction between the real time image and the reference image are the group of the difference pixels. Therefore applied Blob Analysis to analyzed or detected the group of pixel in the result image is an efficient application of the algorithm. The process began with subtract of each parking lot of real time parking scene by the parking lot of its pair in the reference image. The different between two images was labelled as object pixel or background pixel. The objective pixels are labelled as 1 and background as 0. Figure 4.7 shows the image labeled by 1 and 0, the applied function was “im2bw”. Blob analysis was applied to filtering the group of connecting pixel. The noise was removing and the remaining group of pixel was used for analyze the status of the parking lots. The parking lots status was then passed to the first validation process in the output module.

4.1.4 Texture Extraction Module

The texture of image's created the different pixel value. In this module entropy filtering was applied as the object detection technique. Function "entropyfilt" was applied as the main function in this module. Input image was applied entropy filter and rescale to change the texture into the useful format by using the function "mat2gray". The input image and entropy filtered image was shown in Figure 4.8. The filtered image was converts into binary mode as shown in Figure 4.9. The pixels with fewer neighborhoods were deleted as shown in Figure 4.10. The hole in the group of pixel was fulfilling as shown in Figure 4.11. The fulfill image was cropped by the ROI which was set in the Input module. Figure 4.12 shown the cropped image by ROI. Function "bwareaopen", "imfill" and "imclose" were used for the filtering and noise removal in the image. Each parking lot was applied Blob Analysis to identify the status of each parking lot. The result of this module was passed to the second validation process of the output module for further process.

4.1.5 Output Module

The output module is consisting of three parts which are First Validation, Second Validation and Circuit Board & Display.

First Validation: The results from the Histogram Differentiation Module and Blob Analysis Module were the input of this operation. Since both the two modules were using reference image base of parking detection, therefor the results of these two modules were add up by using union concept. The result of union was passed to the Second Validation process for finalize the result.

Second Validation: The output from the First Validation process and the result from the Texture Extraction Module were used to generate the final result. The Second Validation process applied AND operation to the two input results and generate the final parking lots status. The status was generated the result signals then passed to the Circuit Board & Display.

Circuit Board & Display: There were two input signals which are number of available parking lots and status of each parking lot. The number of valid and invalid parking lot was shown on the LED display, while the status of each parking lot was shown by the Green/Red LEDs. The ten pair of green and red LEDs was settled in the

same layout of the car park. The green LED was indicated that the parking lot is available and the red LED was indicated that the parking lot is not available.

In addition, there have some various function have been employed through the operation including “imread”, “imopen”, imwrite”, “imshow”, “imadjust” and “figure” which mainly focused on image utilization on software base. Where the function “size”, “numel”, “step” and “sum” is part of the three object detection module.

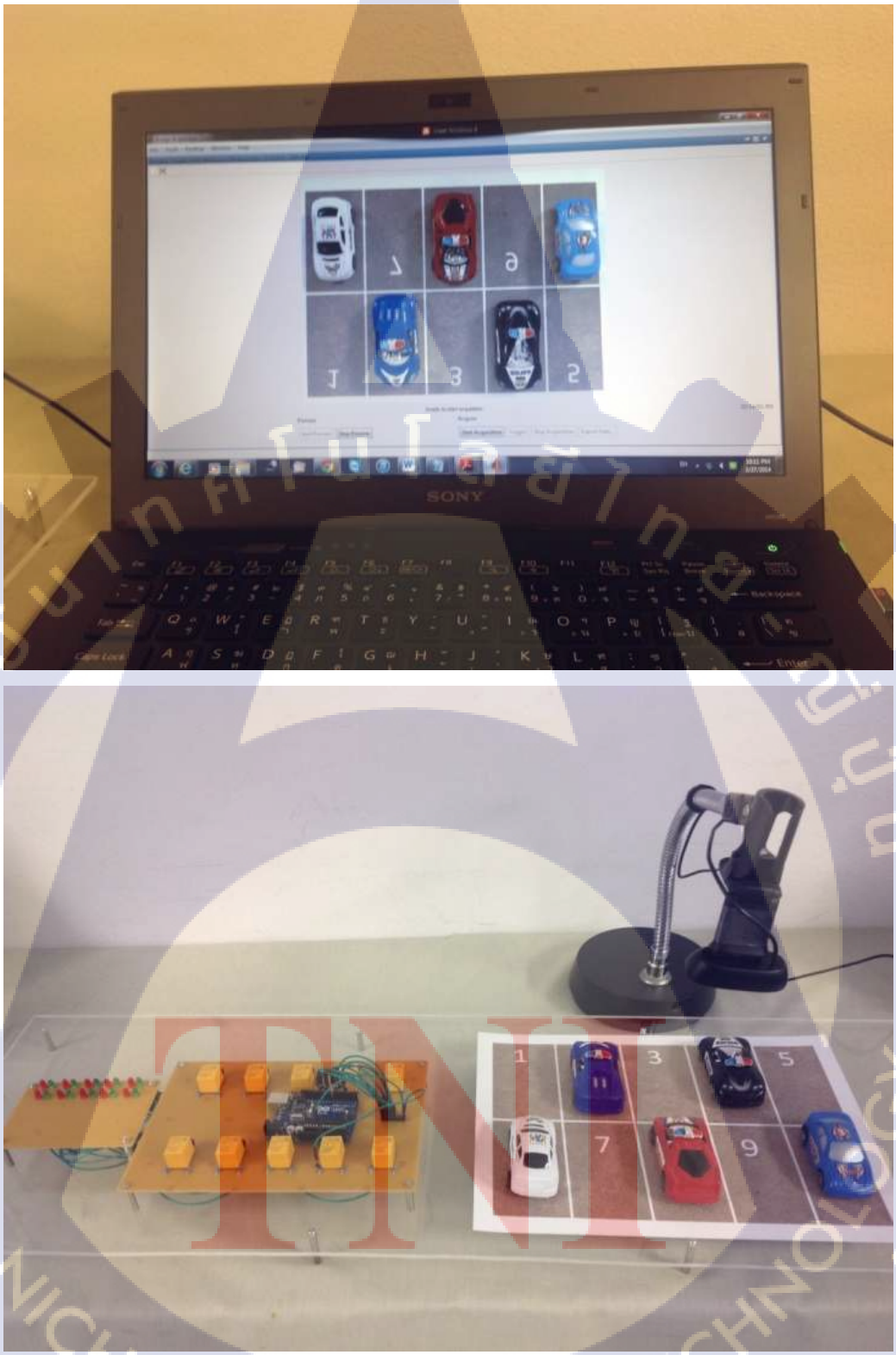


Figure 4.2 Hardware setup of the system. Video signal from the camera (top), output hardware setup (bottom)

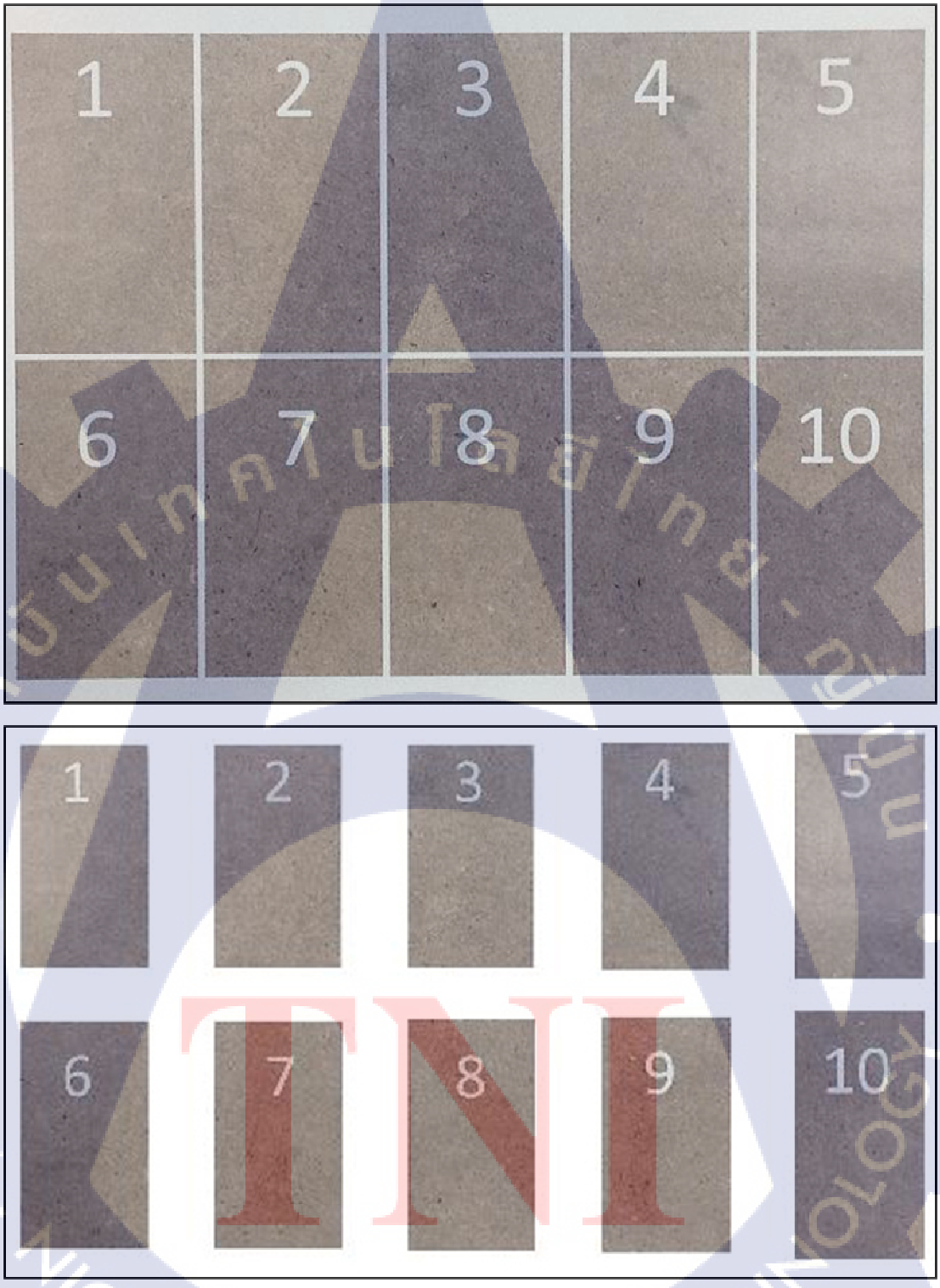


Figure 4.3 Reference image (top) and cropped reference image (bottom)

Figure 4.4 Captured image (top) and cropped captured image (bottom)

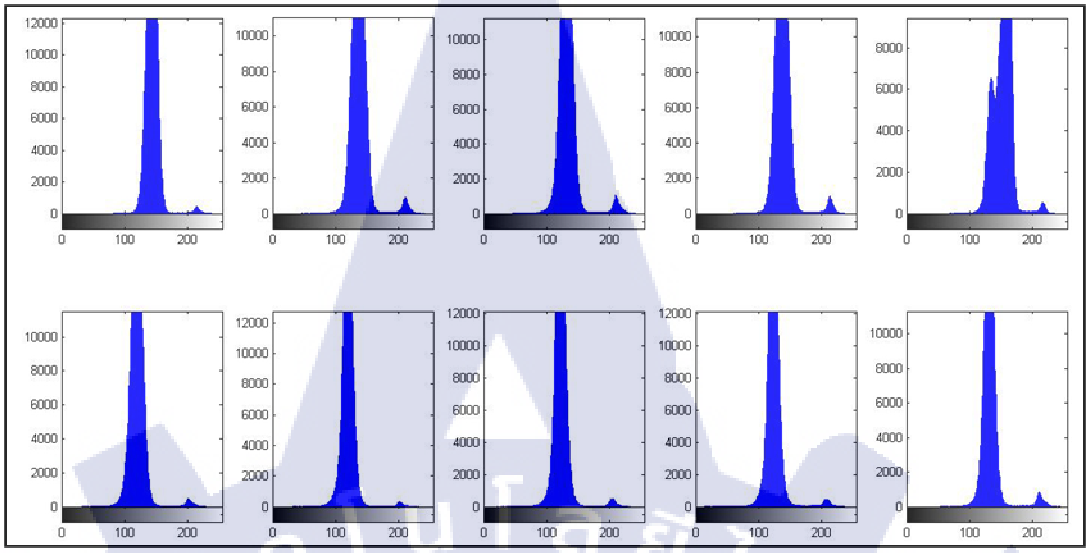


Figure 4.5 Gray scale histograms of reference image parking lots.

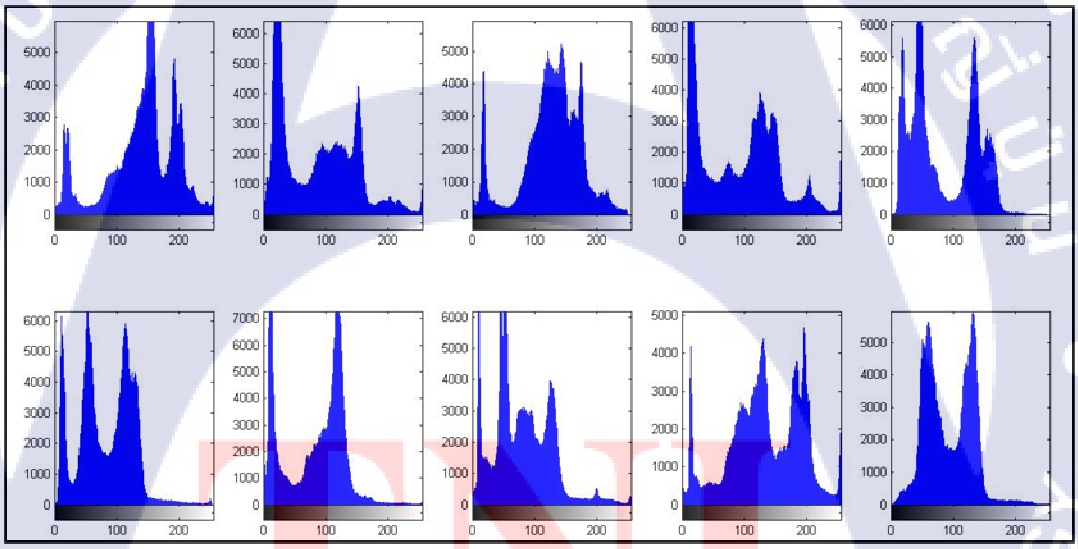


Figure 4.6 Gray scale histogram of real capture image parking lots.

Figure 4.7 Real time input video cropped image (left), Labeled image with 1 and 0 (right)

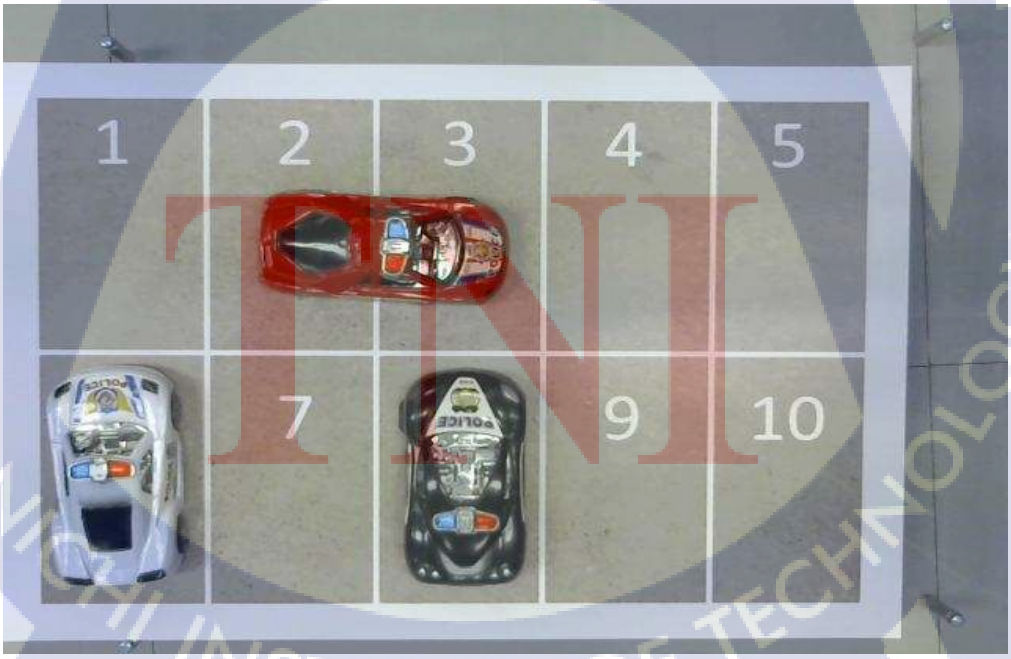


Figure 4.8 Captured video snapshot for test the Texture Extraction module.

Figure 4.9 The entropy filtered image

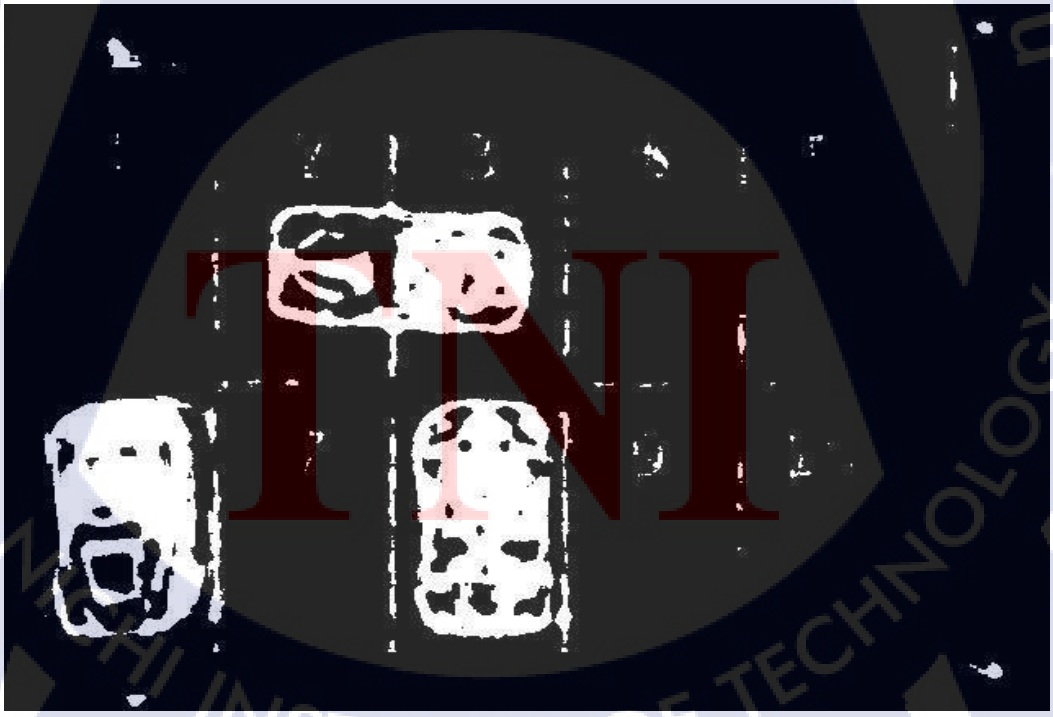


Figure 4.10 Image converted to binary mode

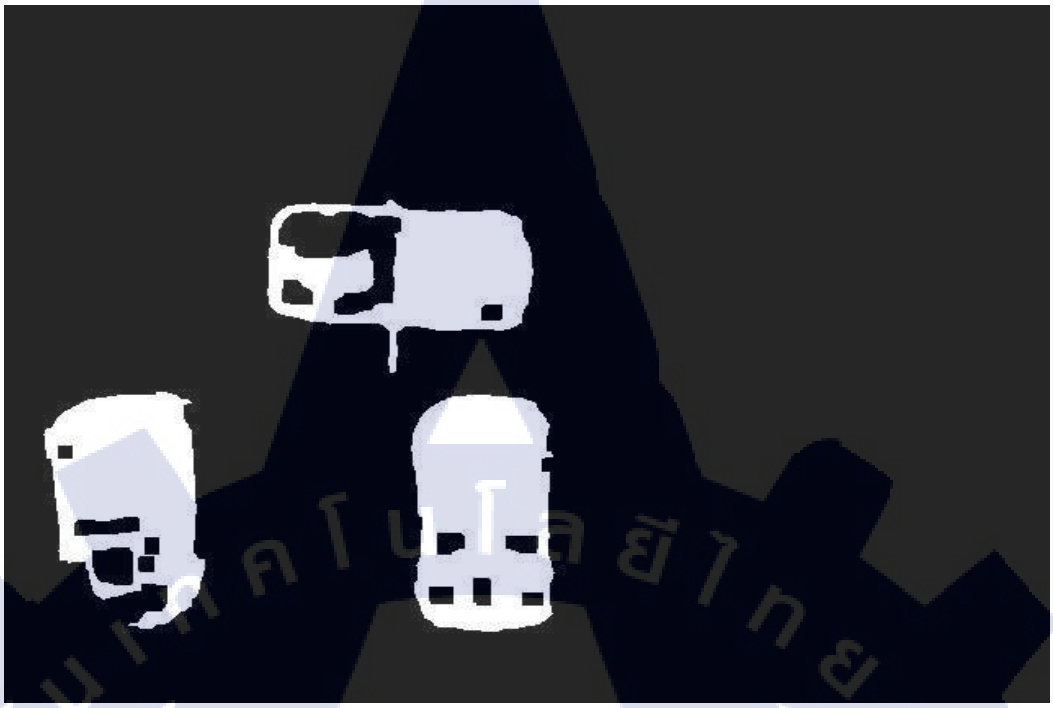


Figure 4.11 The small object removal

Figure 4.12 Image holes filling

Figure 4.13 The cropped image by ROI

4.2 Discussion

The proposed system consists of five modules, which are input module, three parking lot detection modules and output module. The Histogram Differentiation module and Blob Analysis module used as the reference base techniques, while the Texture Extraction module was based on entropy analysis technique. Since the reference base technique has to take the reference image of the car park when the system is start running therefore detecting the extra object in the car park in not a heavy work. The Texture Extraction module was no reference for the detection process therefore identifying the objects in the image would be more complicated compared to the other two modules. The proposed technique of the Texture Extraction module worked properly with the prototype of car park. The result from the Texture Extraction module was shown in Figure 4.13.

Testing the Texture Extraction module at the actual car park has been done to assure the proposed technique is practical. List of error factors was made for the experiment. The first factor was shadow test as shown in the Figure 4.14. The test was made on the shadow electricity post which was dropped and covered half of the parking lot. The shadow test has also been made to the human shadow. The human

shadow was covered 20% of the parking lots as shown in Figure 4.15. The proposed technique was concerned on the texture. The shadow dropped on the ground didn't change the texture of the image therefore it was not affects the detection result. Since the shadows only making dropped area darker therefore the size also not affects the result.

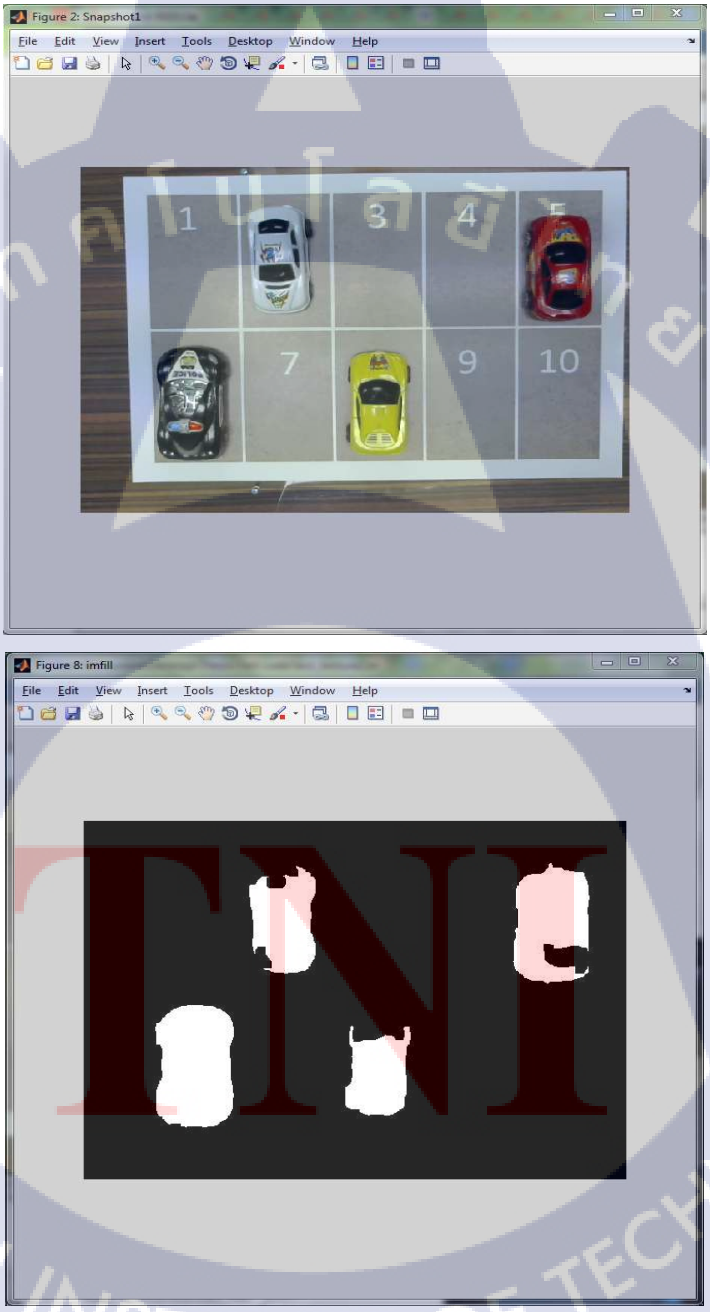


Figure 4.14 Car detection results by Texture Extraction module

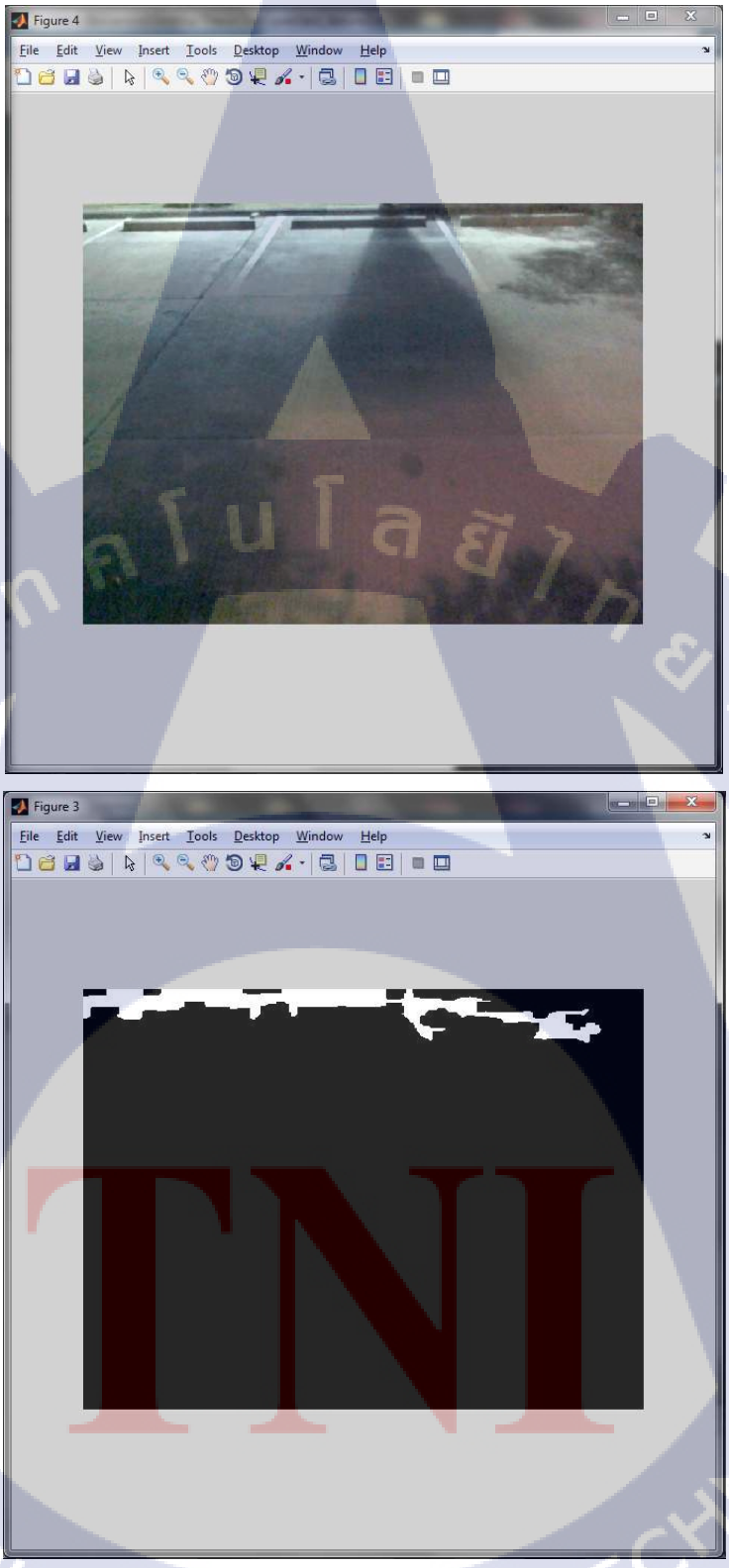


Figure 4.15 Electricity post shadow test by Texture Extraction module

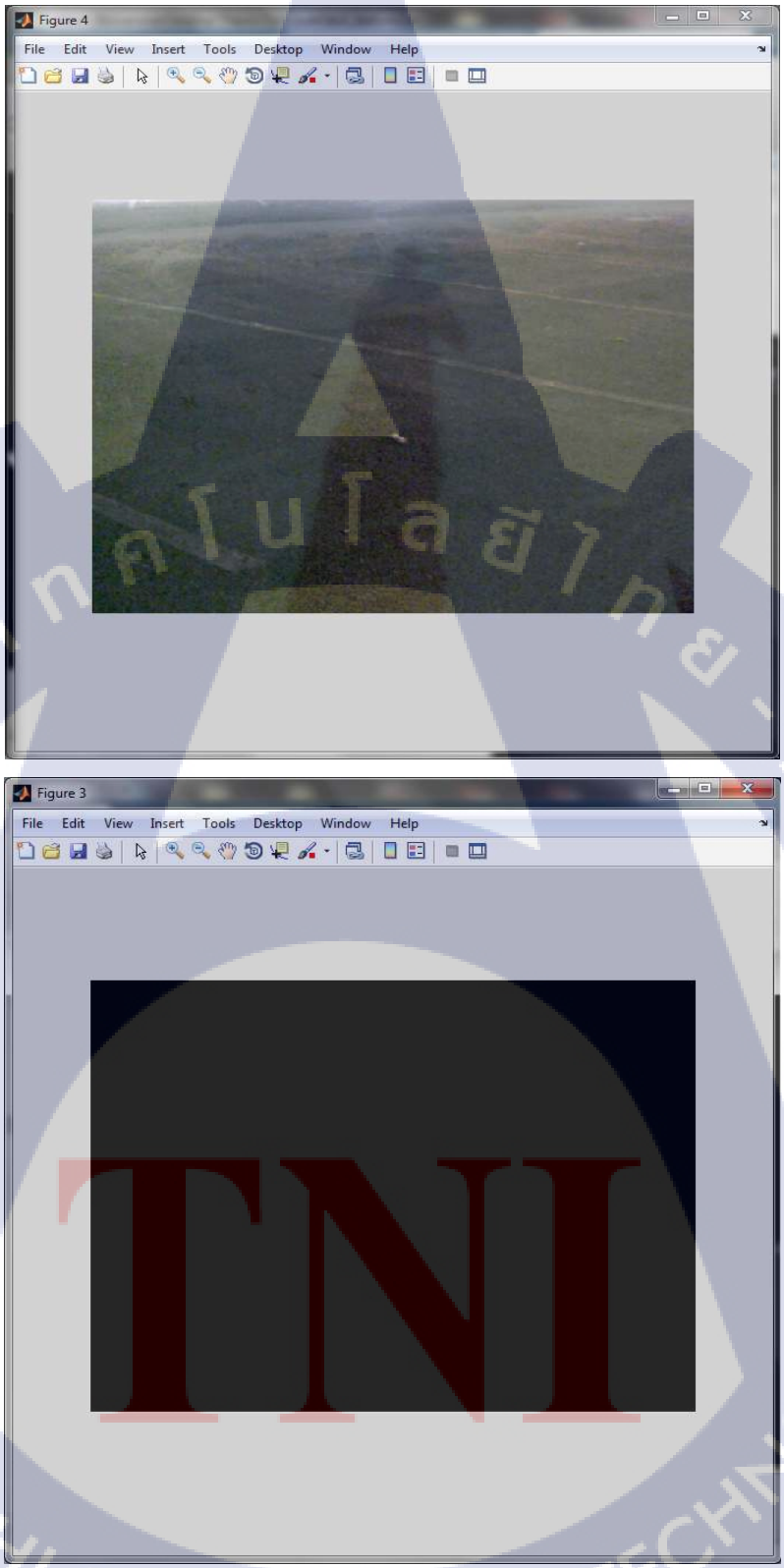


Figure 4.16 Human's shadow test by Texture Extraction module

Noise test has been done as shown in Figure 4.16. The leaves dropped on the ground created the random texture to the parking lots. Since the proposed module included filter for deleted the small object there for leaves on the ground have been handle the filter. Chair test has been done for testing the detection of object smaller than the car's size. Chair in the video captured image was detected by the proposed technique as shown in Figure 4.17. The chair detection was the same concept as detecting the leaves dropped on the parking lots. Since the size of the chair was a lot bigger than leaves therefore it was not deleted by the small object filter of the proposed module.

Car parks at different places are different ground surface therefore different texture could be created. Figure 4.18 and Figure 4.19 were showing the functional of the proposed technique applied to the differences car parks ground surface. The texture on the ground was in pattern while the proposed module was focused on detecting the object in the input image with random texture. Therefore the texture on the ground was dispose after passed the entropy filtering and the binary conversion process.

The last test was made at the car park in the evening. There were three testing condition at the same time. The first was the random texture on the rough ground surface. The second was the medium size object; there were two boxes near the white truck which was on the left of the image. The last was the light; low light at the evening could make the camera unable to capture the texture in detail. Figure 4.20 show the test at the actual car park with low light condition. The result in this experiment proved that the proposed module was flexible and applicable to the actual car park.

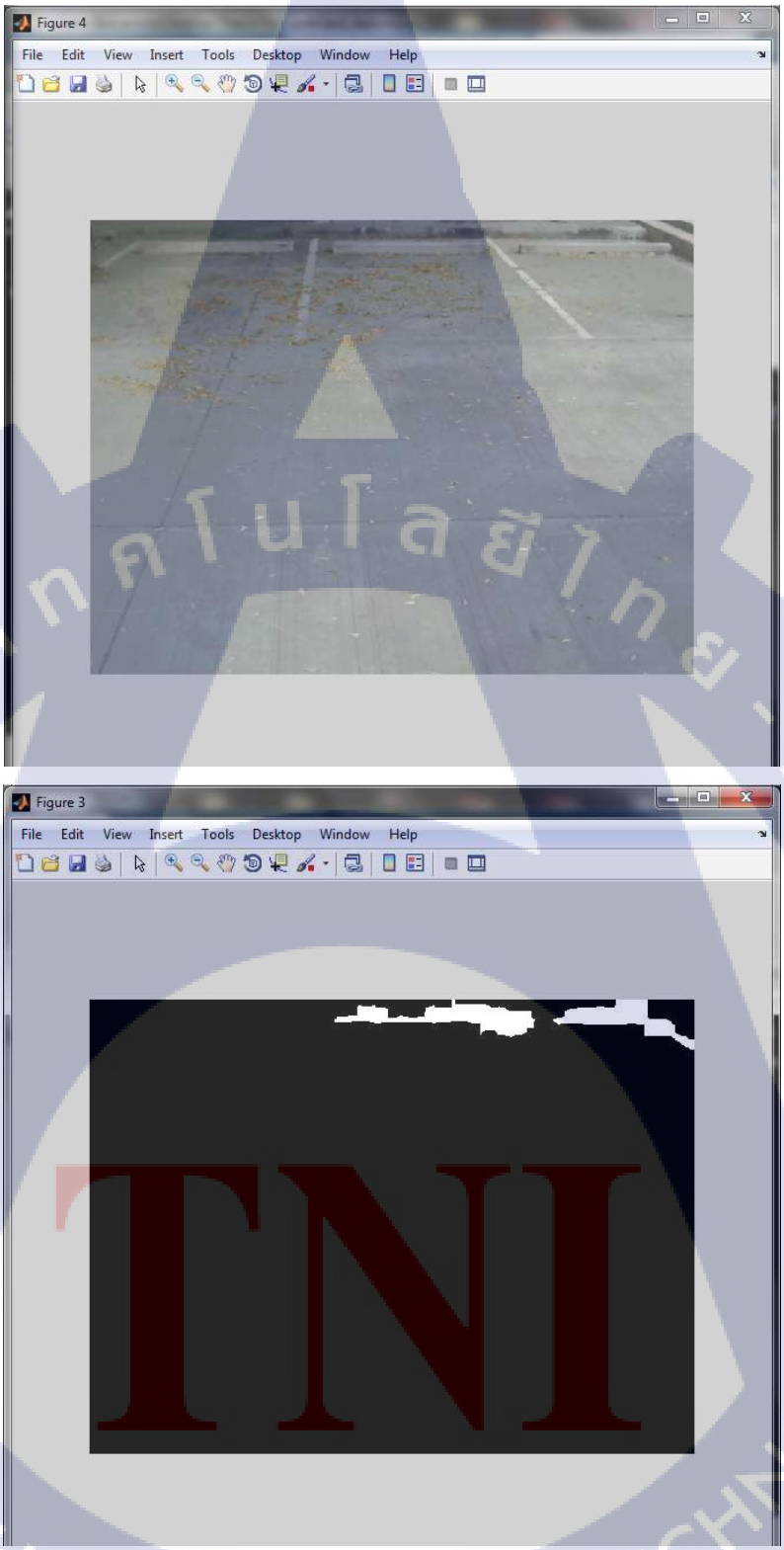


Figure 4.17 Leaf test by Texture Extraction module

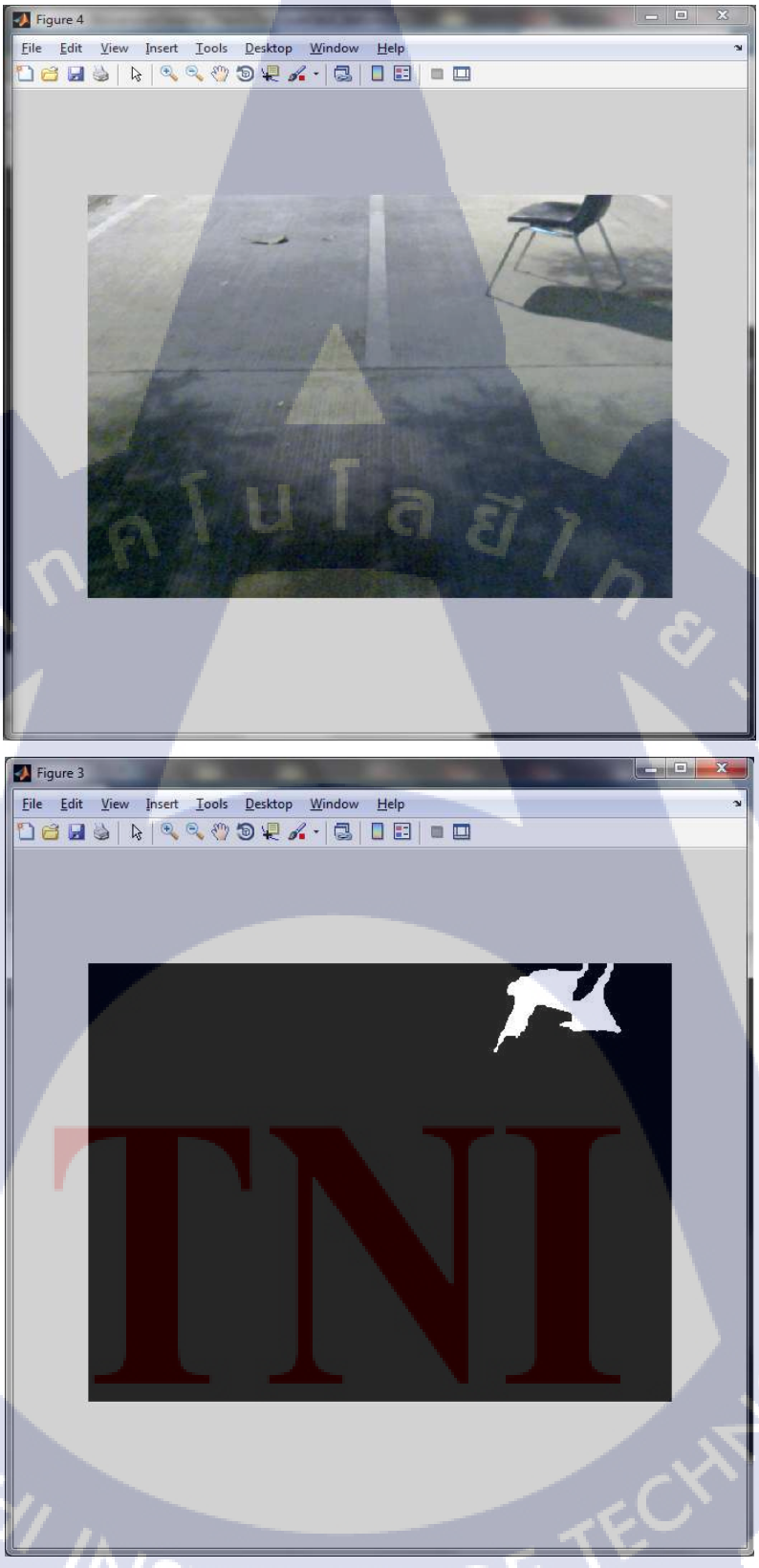


Figure 4.18 Chair test by Texture Extraction module

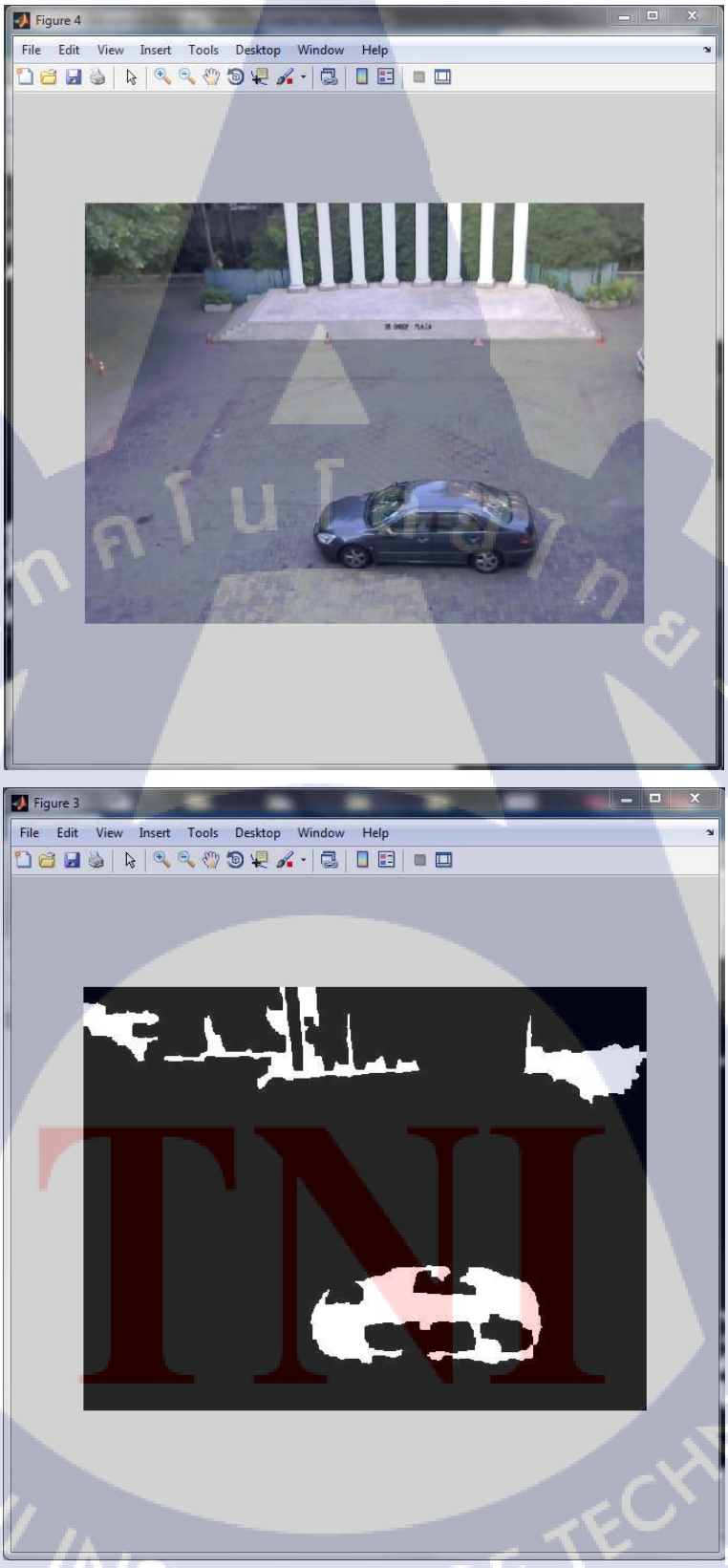


Figure 4.19 Ground with brick surface test by Texture Extraction module

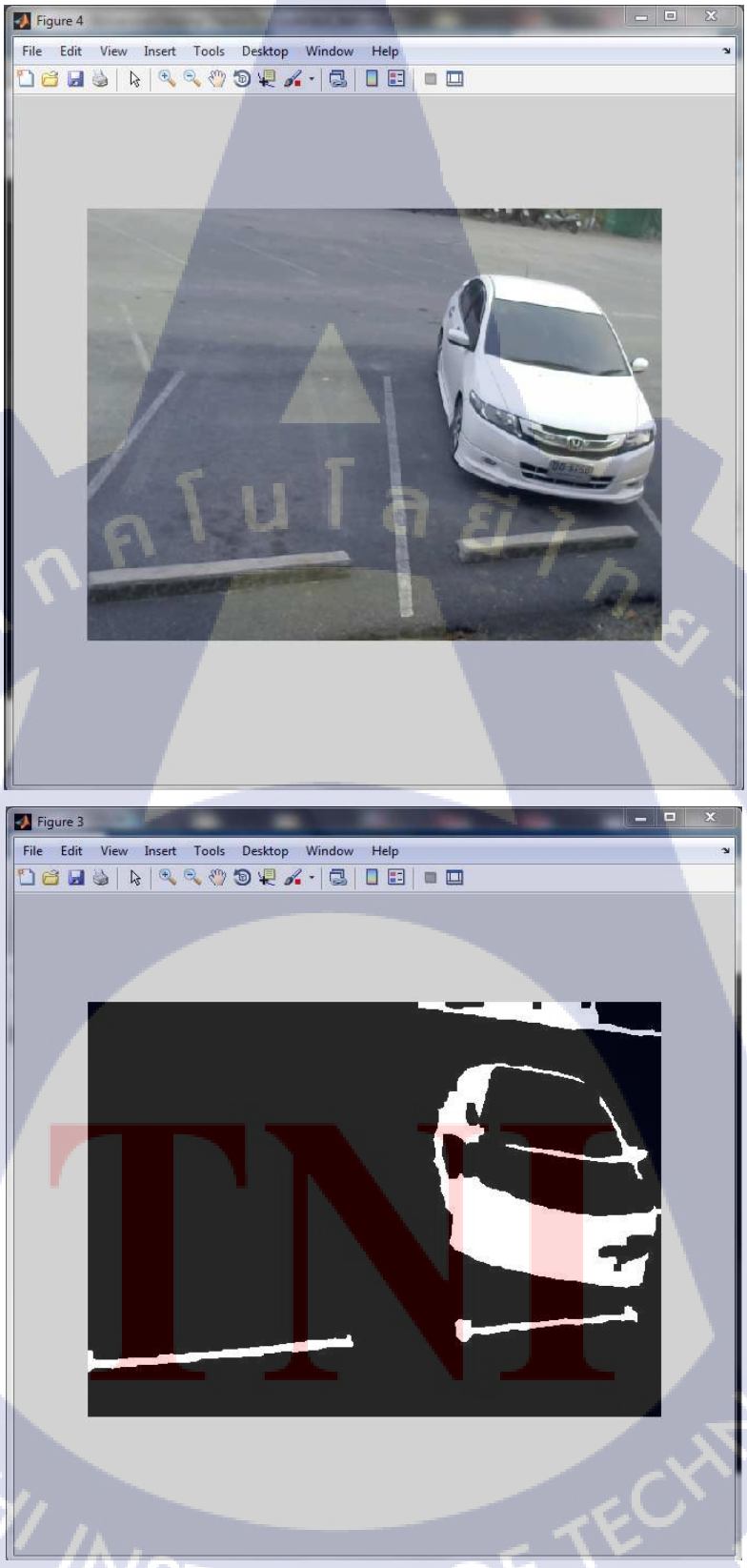


Figure 4.20 Ground with concrete surface test by Texture Extraction module

Figure 4.21 Actual car parking lots experimental result. Captured image of car park (top), output image from Texture Extraction module (bottom)

Chapter 5
Conclusion and Suggestion

5.1 Conclusion

A Smart Cost-Effective Real-Time Outdoor Car Parking Space Detection using Video Processing was designed and tested. The system was defined in five modules consist of Input Module, Histogram Differentiation Module, Blob Analysis Module, Texture Extraction Module and Output Module. The Histogram Differentiation module and Blob Analysis module are reference base technique while the technique applied for Texture Extraction Module was based on Entropy Filtering. These tree modules were used as parking lot detection modules. The system was tested on the prototype. The input video was acquired by the Input module and the output of this module was passed to the three parking lot detection modules. The output from the Histogram Differentiation module and Blob Analysis module were passed to the First Validation process of the Output module. The first validation result was then passed to the Second Validation process. The output of Texture Analysis module and the first validation result were processed in the Second Validation process to generate the parking lots status. The parking lots status was display by LED in the hardware part of the Output module.

Four parking cases were setup for testing the proposed system as shown in Figure 4.21. The detection results were shown in Table 4.1. The result of experiment for case 1 with not car parking was 100% detected by all three detection modules. The result of the case 2 that has full parking car was 100% detected by Histogram Differentiation module and Texture Extraction module while the Blob analysis module can only detected 70%. The case 3 with five cars parking in order was 100% detected by Histogram Differentiation module and Texture Extraction Module and 90% detected by Blob analysis module. The last case that has two cars with inappropriate parking was 80%, 90% and 100% detected by Histogram Differentiation module, Blob Analysis module and Texture Extraction module respectively.

The result generated by the Output module has shown the system detected the status of the parking lots correctly. The error or undetected car in module was solved by the output of the other detection module that passed to the double validation process.

5.2 Suggestion

The detection time has to be considered for the real time system. The software of the proposed system was developed based on MATLAB, therefore some programming code took several second to process by the computer. OpenCV is alternative choice for the computer vision programming. The varieties image processing functions of the OpenCV offer the developer created better image processing software.

Figure 5.1 Four parking cases for the experiment. (a) Case 1: No car, (b) Case 2: Full parking car, (c) Case 3: Five cars parking in order, (d) Case 4: Two cars with inappropriate parking

Table 5.1 System test results on four parking cases. The parking lot number with car parking and detection accuracy.

Parking Situation	**Case 1**	**Case 2**	**Case 3**	**Case 4**
Histogram Differentiation (% efficiency)	No-Parking (100%)	Full-parking (100%)	2,4,6,8,10 (100%)	7 (80%)
Blob Analysis (% efficiency)	No-Parking (100%)	1,2,3,4,5,8,10 (70%)	2,4,8,10 (90%)	3,4 (90%)
First Validation (% efficiency)	No-Parking (100%)	Full-parking (100%)	2,4,6,8,10 (100%)	3,4,7 (100%)
Texture Analysis (% efficiency)	No-Parking (100%)	Full-parking (100%)	2,4,6,8,10 (100%)	3,4,7 (100%)
Second Validation (% efficiency)	No-Parking (100%)	Full-parking (100%)	2,4,6,8,10 (100%)	3,4,7 (100%)
System Outcome (% efficiency)	No-Parking (100%)	Full-parking (100%)	2,4,6,8,10 (100%)	3,4,7 (100%)

References

References

[1] Ankit Sharma; and Nirbhowjap Singh. (2010). Object Detection In Image Using Particle Swarm Optimization. **International Journal of Engineering and Technology**. 2 (6) : 419-426.

[2] Nabin Ghoshal; and J. K. Mandal. (2010). Image Authentication Technique in Frequency Domain based on Discrete Fourier Transformation (IATFDDFT). **Proceeding of ICCS.** November 19-20, 2012. P. 144-150.

[3] Ching-Chun Huang; and Sheng-Jyh Wang. (2010). A Hierarchical Bayesian Generation Framework for Vacant Parking Space Detection. **IEEE Transactions on Circuits and Systems for Video** Technology**.** 20 (12) : 1,770-1,785.

[4] H.Ichihashi; and Colleague. (2010). Improvement in the Performance of Camera Based Vehicle Detector for Parking Lot. **IEEE**. 978-1-4244-8126-2/10. P.1-7.

[5] Li-Chih Chen; and Colleague. (2010). Vision-Based Vehicle Surveillance and Parking Lot Management Using Multiple Cameras. Sixth International Conference on Intelligent Information Hiding and Multimedia Signal Processing. **IEEE**. 978-0-7695-4222-5/10. P. 631-634.

[6] C. NagaRaju; and Colleague. (2011). Morphological Edge Detection Algorithm Based on Multi-Structure Elements of Different Directions. **International Journal of Information and Communication Technology Research**. 1 (1) : 37-43.

[7] Jermsak Jermsurawong; and Colleague. (2012). Car Parking Vacancy Detection and Its Application in 24-Hour Statistical Analysis. **10th International Conference on Frontiers of Information Technology.** 978-0-7695-4927-9/12. P. 84-90.

[8] Reja Oji. (2012). An Automatic Algorithm for Object Recognition and Detection based on Asift Keypoints. **Signal & Image Processing : An International Journal (SIPIJ).** 3 (5) : 29-39.

[9] Ching-Chun Huang; and Colleague. (2012). A Surface-based Vacant Space Detection for an Intelligent Parking Lot. **12th International Conference on ITS Telecommunications.** 978-1-4673-3070-1/2. P. 254-288.

[10] Lih Lin, Ng; and Colleague. (2012). Vision-Based Activities Recognition by Trajectory Analysis for Parking Lot Surveillance. **IEEE.** 978-1-4673-3119-7/12. P. 137-142.

[11] Wang Lixia; and Jiang Dalin. (2012). A Method of Parking Space Detection based on Image Segmentation and LBP**. Fourth International Conference on Multimedia Information Networking and Security. IEEE**. 978-0-7695-4852-4/12. P. 229-232.

[12] Guangling Sun; and Colleague. (2013). Blurred Image Classification based on Adaptive Dictionary. **The International Journal of Multimedia & Its Applications (IJMA)**. 5 (1) : 1-9.

[13] Diana Delibaltov; and Colleague. (2013). Parking Lot Occupancy Determination from Lamp-Post Camera Images. **Proceedings of the 16th International IEEE Annual Conference on Intelligent Transportation System. IEEE.** 978-1-4799-2914-613. P. 2,387-2,392.

[14] Kairoek Choeychuen. (2013). Automatic Parking Lot Mapping for Available Parking Space Detection. **5th International Conference on Knowledge and Smart Technology (KST). IEEE.** 978-1-4673-4853-9/13. P. 117-121.

[15] Huan Weia**.** (2013). Vehicle Detection from Parking Lot Aerial Images. **IEEE.** 978-1-4799-1114-1/13 P. 4,002-4,005.

[16] Prabhakar Telagarapu; M.V.Nageswara Rao; and Gulivindala Suresh. (2008). A Novel Traffic-tTacking System using Morphological and Blob Analysis. **Computing, Communication and Applications (ICCCA) International Conference**. 978-1-4673-0270-8. P.1-4.

[17] K. Ueda; and Colleague. (1991). An Algorithm for Detection Parking Cars by the use of Picture Processing. **IEICE Trans. Information and Systems.** 74 (1) : 1,379-1,389.

[18] E. Maeda; and K. Ishii. (1992). Evaluation of Normalized Principal Component Features in Object Detection. **IEICE Trans. Information and Systems**. 75 (3) : 520-529.

[19] M. Yachida; M. Asada; and S. Tsuji. (1981). Automatic Analysis of Moving Image. **IEEE Trans. Pattern Anal and Mach.Intell.** 3 (1) : 12-19.

[20] S. Arora; and Colleague. (2007). Multilevel Thresholding for Image Segmentation through a Fast Statistical Recursive Algorithm. Elsevier. **Pattern Recognition Letters**. 29 (2007) : 119-125.

[21] Dung Duc Nguyen; and Colleague. (2008). Finger Extraction from Scene with Grayscale Morphology and BLOB Analysis. Robotics and Biomimetics, ROBIO 2008. **IEEE International Conference**. 2009 : 324 – 329.

[22] Tao Jia; Nong-Liang Sun; and Mao-Yong. (2008). Moving Object Detection based on Blob Analysis. **IEEE International Conference**. 978-1-4244-2503-7. P. 322 – 325.

[23] Z. Bin, J. Dalin; W. Fang; and W. Tingting. (2009). A Design of Parking Space Deteector based on Video Image. **The Ninth International Conference on Electronic Measurement & Instruments (ICEMI).** 978-1-4244-3864-8. P. 253-256.

[24] T. Hasegawa; and S. Ozawa. (1994). Counting Cars by Tracking of Moving Object in the Outdoor Parking Lot. **Vehicle Navigation and Information Systems Conference.** 0-7803-2105-7. P. 63-68.

[25] Sayanti Banerjee; Pallavi Choudekar; and M. K. Muju. (2011). Real Time Car Parking System using Image Processing. **IEEE.** 978-1-4244-8679-3. P. 99-103.

[26] L.Wang; and J. Bai. (1999). Threshold Selection by Clustering Grey Levels of Boundary. Elsevier Science. **Pattern Recognition Letters 24**. P. 1,983-1,999.

[27] Thiang; Andre Teguh Guntoro; and Resmana Lim. (2001). Type of Vehicle Recognition using Template Matching Method. **International Conference on Electrical, Electronics, Communication, Information.** March 7-8, 2001. P. 1-5.

[28] S. Saleh Al-Amri; N. V. Kalyankar; and Khamitkar S. (2010). Image Segmentation by using Thershold Techniques. **Journal of Computing**. 2 (5) : 83-86.

[29] Per Christian Henden. (2004). **Exercise in Computer: a Comparison of Thresholding Methods**. Retrieved May 1, 2014, from http://www.pvv.org/~perchrh/papers/datasyn/paper2/report.pdf

Appendix

Appendix - A

MATLAB Functions

videoinput

Create video input object

Syntax

```
obj = videoinput(adaptorname)
obj = videoinput(adaptorname,deviceID)
obj = videoinput(adaptorname,deviceID,format)
obj = videoinput(adaptorname,deviceID,format,P1,V1,...)
```

Description

`obj = videoinput(adaptorname)` constructs the video input object `obj`. A video input object represents the connection between MATLAB® and a particular image acquisition device. *`adaptorname`* is a text string that specifies the name of the adaptor used to communicate with the device. Use the `imaqhwinfo` function to determine the adaptors available on your system.

`obj = videoinput(adaptorname,deviceID)` constructs a video input object `obj`, where `deviceID` is a numeric scalar value that identifies a particular device available through the specified adaptor, *`adaptorname`*. Use the `imaqhwinfo(adaptorname)` syntax to determine the devices available through the specified adaptor. If `deviceID` is not specified, the first available device ID is used. As a convenience, a device's name can be used in place of the `deviceID`. If multiple devices have the same name, the first available device is used.

`obj = videoinput(adaptorname,deviceID,format)` constructs a video input object, where *`format`* is a text string that specifies a particular video format supported by the device or the full path of a device configuration file (also known as a camera file).

To get a list of the formats supported by a particular device, view the `DeviceInfo` structure for the device that is returned by the `imaqhwinfo` function. Each `DeviceInfo` structure contains a `SupportedFormats` field. If `format` is not specified, the device's default format is used.

When the video input object is created, its `VideoFormat` field contains the format name or device configuration file that you specify.

`obj = videoinput(adaptorname,deviceID,format,P1,V1,...)` creates a video input object `obj` with the specified property values. If an invalid property name or property value is specified, the object is not created.

The property name and property value pairs can be in any format supported by the `set` function, i.e., parameter/value string pairs, structures, or parameter/value cell array pairs.

To view a complete listing of video input object functions and properties, use the `imaqhelp` function.

```
imaqhelp videoinput
```

In the documentation, see **Image Acquisition Toolbox Properties** for links to the property reference pages.

Examples

Construct a video input object.

```
obj = videoinput('matrox', 1);
```

Select the source to use for acquisition.

```
set(obj, 'SelectedSourceName', 'input1')
```

View the properties for the selected video source object.

```
src_obj = getselectedsource(obj);
get(src_obj)
```

Preview a stream of image frames.

```
preview(obj);
```

Acquire and display a single image frame.

```
frame = getsnapshot(obj);
image(frame);
```

Remove video input object from memory.

```
delete(obj);
```

More About

Collapse all

Tips

The toolbox chooses the first available video source object as the selected source and specifies this video source object's name in the object's `SelectedSourceName` property. Use `getselectedsource(obj)` to access the video source object that is used for acquisition.

See Also

`delete` | `imaqfind` | `isvalid` | `preview`

getsnapshot

Immediately return single image frame

Syntax

```
frame = getsnapshot(obj)
[frame, metadata] = getsnapshot(obj)
```

Description

`frame = getsnapshot(obj)` immediately returns one single image frame, `frame`, from the video input object `obj`. The frame of data returned is independent of the video input object `FramesPerTrigger` property and has no effect on the value of the `FramesAvailable` or `FramesAcquired` property.

The object `obj` must be a 1-by-1 video input object.

`frame` is returned as an H-by-W-by-B matrix where

H	Image height, as specified in the `ROIPosition` property
W	Image width, as specified in the `ROIPosition` property
B	Number of bands associated with `obj`, as specified in the `NumberOfBands` property

`frame` is returned to the MATLAB® workspace in its native data type using the color space specified by the `ReturnedColorSpace` property.

You can use the MATLAB `image` or `imagesc` function to view the returned data.

`[frame, metadata] = getsnapshot(obj)` returns metadata, a 1-by-1 array of structures. This structure contains information about the corresponding frame. The metadata structure contains the field `AbsTime`, which is the absolute time the frame was acquired, expressed as a time vector. In addition to that field, some adaptors may choose to add other adaptor-specific metadata as well.

> **Note** If `obj` is running but not logging, and has been configured with a hardware trigger, a timeout error will occur.

To interrupt the `getsnapshot` function and return control to the MATLAB command line, issue the `^C` (**Ctrl+C**) command.

Examples

Create a video input object.

```
obj = videoinput('matrox', 1);
```

Acquire and display a single frame of data.

```
frame = getsnapshot(obj);
image(frame);
```

Remove the video input object from memory.

```
delete(obj);
```

For an example of using `getsnapshot`, see the Image Acquisition Toolbox™ example **Acquiring a Single Image in a Loop** in the **Examples** list at the top of the Image Acquisition Toolbox main Documentation Center page, or open the file demoimaq_GetSnapshot.m in the MATLAB Editor.

See Also

getdata | **imaqhelp** | **peekdata**

imcrop

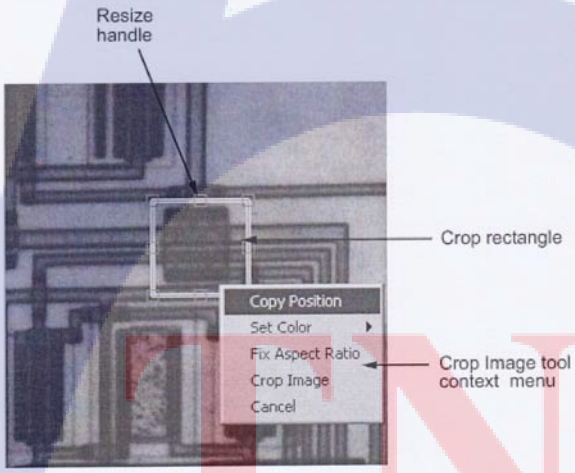
Crop image

Syntax

```
I = imcrop
I2 = imcrop(I)
X2 = imcrop(X, map)
I = imcrop(h)
I2 = imcrop(I, rect)
X2 = imcrop(X, map, rect)
[...] = imcrop(x, y,...)
[I2 rect] = imcrop(…)
[X,Y,I2,rect] = imcrop(…)
```

Description

`I = imcrop` creates an interactive Crop Image tool associated with the image displayed in the current figure, called the target image. The Crop Image tool is a moveable, resizable rectangle that you can position interactively using the mouse. When the Crop Image tool is active, the pointer changes to cross hairs + when you move it over the target image. Using the mouse, you specify the crop rectangle by clicking and dragging the mouse. You can move or resize the crop rectangle using the mouse. When you are finished sizing and positioning the crop rectangle, create the cropped image by double-clicking the left mouse button or by choosing **Crop Image** from the context menu. `imcrop` returns the cropped image, `I`. The following figure illustrates the Crop Image tool with the context menu displayed. For more information about the interactive capabilities of the tool, see the table that follows.



Interactive Behavior	Description
Deleting the Crop Image tool.	Press **Backspace**, **Escape** or **Delete**, or right-click inside the crop rectangle and select **Cancel** from the context menu.

Interactive Behavior	Description
	Note: If you delete the ROI, the function returns empty values.
Resizing the Crop Image tool.	Select any of the resize handles on the crop rectangle. The pointer changes to a double-headed arrow ↔. Click and drag the mouse to resize the crop rectangle.
Moving the Crop Image tool.	Move the pointer inside the boundary of the crop rectangle. The pointer changes to a fleur shape ✥. Click and drag the mouse to move the rectangle over the image.
Changing the color used to display the crop rectangle.	Right-click inside the boundary of the crop rectangle and select **Set Color** from the context menu.
Cropping the image.	Double-click the left mouse button or right-click inside the boundary of the crop rectangle and select **Crop Image** from the context menu.
Retrieving the coordinates of the crop rectangle.	Right-click inside the boundary of the crop rectangle and select **Copy Position** from the context menu. `imcrop` copies a four-element position vector (`[xmin ymin width height]`) to the clipboard.

`I2 = imcrop(I)` displays the image `I` in a figure window and creates a cropping tool associated with that image. `I` can be a grayscale image, a truecolor image, or a logical array. The cropped image returned, `I2`, is of the same type as `I`.

`X2 = imcrop(X, map)` displays the indexed image `X` in a figure using the colormap `map`, and creates a cropping tool associated with that image.

`I = imcrop(h)` creates a cropping tool associated with the image specified by handle `h`. `h` may be an image, axes, uipanel, or figure handle. If `h` is an axes, uipanel, or figure handle, the cropping tool acts on the first image found in the container object.

> **Note:** With these interactive syntaxes, the cropping tool blocks the MATLAB® command line until you complete the operation.

`I2 = imcrop(I, rect)` crops the image `I`. `rect` is a four-element position vector `[xmin ymin width height]` that specifies the size and position of the crop rectangle.

`X2 = imcrop(X, map, rect)` crops the indexed image `X`. `map` specifies the colormap used with `X`. `rect` is a four-element position vector `[xmin ymin width height]` that specifies the size and position of the cropping rectangle.

`[...] = imcrop(x, y,...)` specifies a non-default spatial coordinate system for the target image. `x` and `y` are two-element vectors specifying `XData` and `YData`.

`[I2 rect] = imcrop(...)` returns the cropping rectangle in `rect`, a four-element position vector.

`[X,Y,I2,rect] = imcrop(...)` returns `x` and `y`, two-element vectors that specify the `XData` and `YData` of the target image.

Class Support

If you specify `rect` as an input argument, the input image can be logical or numeric, and must be real and nonsparse. `rect` is of class double.

If you do not specify `rect` as an input argument, `imcrop` calls `imshow`. `imshow` expects `I` to be `logical`, `uint8`, `uint16`, `int16`, `single`, or `double`. A truecolor image can be `uint8`, `int16`, `uint16`, `single`, or `double`. `X` can be `logical`, `uint8`, `uint16`, `single`, or `double`. The input image must be real and nonsparse.

If you specify an image as an input argument, the output image has the same class as the input image.

If you don't specify an image as an input argument, i.e., you call `imcrop` with no input arguments or a handle, the output image has the same class as the input image except for `int16` or `single`. If the input image is `int16` or `single`, the output image is `double`.

Examples

```
I = imread('circuit.tif');
I2 = imcrop(I,[75 68 130 112]);
imshow(I), figure, imshow(I2)
```

More About

collapse all

Tips

Because `rect` is specified in terms of spatial coordinates, the `width` and `height` elements of `rect` do not always correspond exactly with the size of the output image. For example, suppose `rect` is `[20 20 40 30]`, using the default spatial coordinate system. The upper-left corner of the specified rectangle is the center of the pixel (20,20) and the lower-right corner is the center of the pixel (50,60). The resulting output image is 31 -by-41, not 30-by-40, because the output image includes all pixels in the input image that are completely *or partially* enclosed by the rectangle.

See Also

`imrect` | `zoom`

rgb2gray

Convert RGB image or colormap to grayscale

Syntax

```
I = rgb2gray(RGB)
newmap = rgb2gray(map)
gpuarrayB = rgb2gray(gpuarrayA)
```

Description

`I = rgb2gray(RGB)` converts the truecolor image `RGB` to the grayscale intensity image I. `rgb2gray` converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.

`newmap = rgb2gray(map)` returns a grayscale colormap equivalent to map.

`gpuarrayB = rgb2gray(gpuarrayA)` performs the operation on a GPU. `gpuarrayA` can be either an RGB image or a colormap. `gpuarrayB` is a gpuArray of the same type as `gpuarrayA`. This syntax requires the Parallel Computing Toolbox™

Note: A grayscale image is also called a gray-scale, gray scale, or gray-level image.

Class Support

If `I` is an RGB image, it can be of class `uint8`, `uint16`, `single`, or `double`. The output image `I` is of the same class as the input image. If `I` is a colormap, the input and output colormaps are both of class `double`.

If `gpuarrayA` is an RGB gpuArray image, it can be `uint8`, `uint16`, `single`, or `double`. The output gpuArray image `gpuarrayB` has the same class as the input image. If `gpuarrayA` is a colormap, the input and output colormaps are gpuArrays of class `double`.

Examples

Convert an RGB image to a grayscale image.

```
I = imread('board.tif');
J = rgb2gray(I);
figure, imshow(I), figure, imshow(J);
```

Convert an RGB image to a grayscale image on a GPU.

```
I = gpuArray(imread('board.tif'));
J = rgb2gray(I);
figure, imshow(I), figure, imshow(J);
```

Convert the colormap to a grayscale colormap.

```
[X,map] = imread('trees.tif');
gmap = rgb2gray(map);
figure, imshow(X,map), figure, imshow(X,gmap);
```

Convert the colormap to a grayscale colormap on a GPU. Note how the example pass the colormap to the gpuArray function.

```
[X,map] = imread('trees.tif');

% Pass colormap to gpuArray
gmap = rgb2gray(gpuArray(map));
figure, imshow(X,map), figure, imshow(X,gmap);
```

More About

Collapse all

Algorithms

`rgb2gray` converts RGB values to grayscale values by forming a weighted sum of the *R*, *G*, and *B* components:

```
0.2989 * R + 0.5870 * G + 0.1140 * B
```

Note that these are the same weights used by the `rgb2ntsc` function to compute the *Y* component.

See Also

gpuArray | **ind2gray** | **mat2gray** | **ntsc2rgb** | **rgb2ind** | **rgb2ntsc**

imhist

Histogram of image data

Syntax

```
imhist(I)
imhist(I,n)
imhist(X,map)
[counts,x] = imhist(__)
[__] = imhist(gpuarrayA, __)
```

Description

`imhist(I)` calculates the histogram for the intensity image `I` and displays a plot of the histogram. The number of bins in the histogram is determined by the image type.

- If `I` is a grayscale image, `imhist` uses a default value of 256 bins.
- If `I` is a binary image, `imhist` uses two bins.

`imhist(I,n)` displays a histogram for the intensity image `I`, where `n` specifies the number of bins used in the histogram. `n` also specifies the length of the colorbar displayed at the bottom of the histogram plot. If `I` is a binary image, `n` can only have the value 2.

`imhist(X,map)` displays a histogram for the indexed image `X`. This histogram shows the distribution of pixel values above a colorbar of the colormap `map`. The colormap must be at least as long as the largest index in `X`. The histogram has one bin for each entry in the colormap.

`[counts,x] = imhist(__)` returns the histogram counts in `counts` and the bin locations in `x` so that `stem(x,counts)` shows the histogram. For indexed images, `imhist` returns the histogram counts for each colormap entry. The length of `counts` is the same as the length of the colormap.

`[__] = imhist(gpuarrayA, __)` performs the histogram calculation on a GPU. The input image and the return values are gpuArrays. This syntax requires the Parallel Computing Toolbox™. When the input image is a gpuArray, `imhist` does not automatically display the histogram. To display the histogram, use `stem(x,counts)`.

> **Note:** The maximum value on the *y*-axis may be automatically reduced, so outlier spikes do not dominate. To show the full range of *y*-axis values, call `imhist` with the following syntax:
>
> ```
> [counts,x] = imhist(...)
> ```
>
> Then call `stem`:
>
> ```
> stem(x,counts)
> ```

Code Generation

`imhist` supports the generation of efficient, production-quality C/C++ code from MATLAB. When generating code, the optional second input argument, `n`, must be a compile-time constant. In addition, nonprogrammatic syntaxes are not supported. For example, the syntax `imhist(I)`, where `imhist` displays the histogram, is not

supported. Generated code for this function uses a precompiled **platform-specific (http://www.mathworks.com/support/sysreq/current_release/)** shared library. To see a complete list of toolbox functions that support code generation, see **List of Supported Functions with Usage Notes**.

Class Support

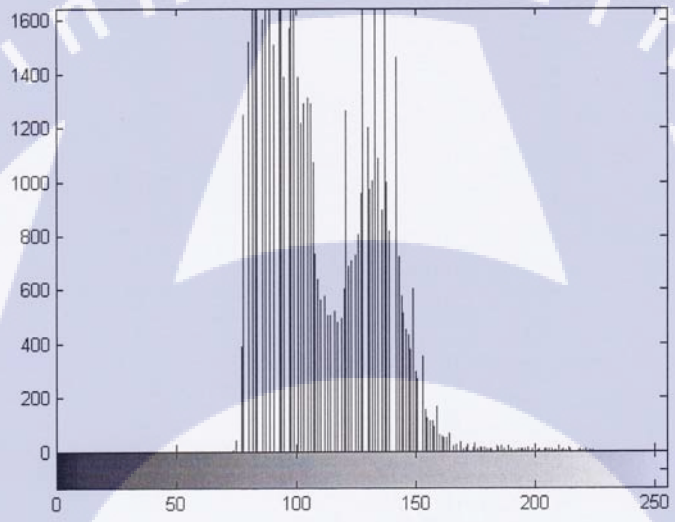
An input intensity image can be of class `uint8`, `int8`, `uint16`, `int16`, `uint32`, `int32`, `single`, `double`, or `logical`. An input indexed image can be of class `uint8`, `uint16`, `single`, `double`, or `logical`.

An input gpuArray intensity image can be of class `uint8`, `int8`, `uint16`, `int16`, `uint32`, `int32`, `single`, `double`, or `logical`. An input indexed image can be of class `uint8`, `uint16`, `single`, `double`, or `logical`.

Examples

Calculate Histogram

```
I = imread('pout.tif');
imhist(I)
```



Calculate Histogram on a GPU

Calculate histogram on a GPU. Because `imhist` does not automatically display the plot of the histogram when run on a GPU, this example uses `stem` to plot the histogram.

```
I = gpuArray(imread('pout.tif'));
[counts,x] = imhist(I);
stem(x,counts);
```

More About

Collapse all

Tips

For intensity images, the `n` bins of the histogram are each half-open intervals of width $A/(n-1)$. In particular, for intensity images that are not `int16`, the pth bin is the half-open interval

$$\frac{A(p-1.5)}{(n-1)} \le x < \frac{A(p-0.5)}{(n-1)},$$

where x is the intensity value. For `int16` intensity images, the pth bin is the half-open interval

$$\frac{A(p-1.5)}{(n-1)} - 32768 \le x < \frac{A(p-0.5)}{(n-1)} - 32768,$$

where x is the intensity value. The scale factor A depends on the image class. A is 1 if the intensity image is `double` or `single`, A is 255 if the intensity image is `uint8`, and A is 65535 if the intensity image is `uint16` or `int16`.

See Also

`gpuArray` | `hist` | `histeq`

im2double

Convert image to double precision

Syntax

```
I2 = im2double(I)
RGB2 = im2double(RGB)
I = im2double(BW)
X2 = im2double(X,'indexed')
gpuarrayB = im2double(gpuarrayA, ___ )
```

Description

`I2 = im2double(I)` converts the intensity image `I` to double precision, rescaling the data if necessary.

If the input image is of class `double`, the output image is identical.

`RGB2 = im2double(RGB)` converts the truecolor image `RGB` to double precision, rescaling the data if necessary.

`I = im2double(BW)` converts the binary image `BW` to a double-precision intensity image.

`X2 = im2double(X,'indexed')` converts the indexed image `X` to double precision, offsetting the data if necessary.

`gpuarrayB = im2double(gpuarrayA, ___ )` performs the conversion on a GPU. The input image and the output image are gpuArrays. This syntax requires the Parallel Computing Toolbox™.

Code Generation

`im2double` supports the generation of efficient, production-quality C/C++ code from MATLAB. To see a complete list of toolbox functions that support code generation, see **List of Supported Functions with Usage Notes**.

Class Support

Intensity and truecolor images can be `uint8`, `uint16`, `double`, `logical`, `single`, or `int16`. Indexed images can be `uint8`, `uint16`, `double` or `logical`. Binary input images must be `logical`. The output image is `double`.

If the input gpuArray `gpuarrayA` is an intensity or truecolor image, it can be `uint8`, `uint16`, `double`, `logical`, `single`, or `int16`. If `gpuarrayA` is an indexed image, it can be `uint8`, `uint16`, `double` or `logical`. If `gpuarrayA` is a binary image, it must be `logical`. The output gpuArray image is `double`.

Examples

Convert array to class `double`.

```
I1 = reshape(uint8(linspace(1,255,25)),[5 5])
I2 = im2double(I1)
```

Convert array to class `double` on the GPU.

```
I1 = gpuArray(reshape(uint8(linspace(1,255,25)),[5 5]))
I2 = im2double(I1)
```

See Also

double | gpuArray | im2int16 | im2single | im2uint16 | im2uint8

vision.BlobAnalysis System object

Package: vision

Properties of connected regions

Description

The `BlobAnalysis` object computes statistics for connected regions in a binary image.

Use the `step` syntax below with input binary image, `BW`, blob analysis object, `H`, and any optional properties. The `step` method computes and returns statistics of the input binary image depending on the property values specified. The order of the returned values when there are multiple outputs are in the order they are described below:

`[AREA,CENTROID,BBOX] = step(H,BW)` returns the area, centroid and the bounding box of the blobs when the `AreaOutputPort`, `CentroidOutputPort` and `BoundingBoxOutputPort` properties are set to `true`. These are the only properties that are set to true by default. If you set any additional properties to `true`, the corresponding outputs follow the `AREA`,`CENTROID`, and `BBOX` outputs.

`[__,MAJORAXIS] = step(H,BW)` computes the major axis length `MAJORAXIS` of the blobs found in input binary image `BW` when the `MajorAxisLengthOutputPort` property is set to `true`.

`[__,MINORAXIS] = step(H,BW)` computes the minor axis length `MINORAXIS` of the blobs found in input binary image `BW` when the `MinorAxisLengthOutputPort` property is set to `true`.

`[__,ORIENTATION] = step(H,BW)` computes the `ORIENTATION` of the blobs found in input binary image `BW` when the `OrientationOutputPort` property is set to `true`.

`[__,ECCENTRICITY] = step(H,BW)` computes the `ECCENTRICITY` of the blobs found in input binary image `BW` when the `EccentricityOutputPort` property is set to `true`.

`[__,EQDIASQ] = step(H,BW)` computes the equivalent diameter squared EQDIASQ of the blobs found in input binary image `BW` when the `EquivalentDiameterSquaredOutputPort` property is set to `true`.

`[__,EXTENT] = step(H,BW)` computes the EXTENT of the blobs found in input binary image `BW` when the `ExtentOutputPort` property is set to `true`.

`[__,PERIMETER] = step(H,BW)` computes the `PERIMETER` of the blobs found in input binary image `BW` when the `PerimeterOutputPort` property is set to `true`.

`[__,LABEL] = step(H,BW)` returns a label matrix `LABEL` of the blobs found in input binary image `BW` when the `LabelMatrixOutputPort` property is set to true.

Code Generation Support
Supports MATLAB® Function block: Yes
System Objects in MATLAB Code Generation.
Code Generation Support, Usage Notes, and Limitations.

Input/Output	Format	Supported Data Types
BW	Vector or matrix that represents a binary image	Boolean
AREA	Vector that represents the number of pixels in labeled regions	32-bit signed integer
CENTROID	*M*-by-2 matrix of centroid coordinates, where *M* represents the number of blobs	▪ Double-precision floating point ▪ Single-precision floating point ▪ Fixed point
BBOX	*M*-by-4 matrix of [x y width height] bounding box coordinates, where *M* represents the number of blobs and [x y] represents the upper left corner of the bounding box.	32-bit signed integer
MAJORAXIS	Vector that represents the lengths of major axes of ellipses	▪ Double-precision floating point ▪ Single-precision floating point
MINORAXIS	Vector that represents the lengths of minor axes of ellipses	Same as MajorAxis port
ORIENTATION	Vector that represents the angles between the major axes of the ellipses and the *x*-axis.	Same as MajorAxis port
ECCENTRICITY	Vector that represents the eccentricities of the ellipses	Same as MajorAxis port
EQDIASQ	Vector that represents the equivalent diameters squared	Same as Centroid port
EXTENT	Vector that represents the results of dividing the areas of the blobs by the area of their bounding boxes	Same as Centroid port

Input/Output	Format	Supported Data Types
PERIMETER	Vector containing an estimate of the perimeter length, in pixels, for each blob	Same as Centroid port
LABEL	Label matrix	8-, 16-, or 32-bit unsigned integer

Construction

`H = vision.BlobAnalysis` returns a blob analysis System object, `H`, used to compute statistics for connected regions in a binary image.

`H = vision.BlobAnalysis(Name,Value)` returns a blob analysis object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*`Name1`*, *`Value1`*,...,*`NameN`*,*`ValueN`*).

Properties

`AreaOutputPort`

Return blob area

Setting this property to `true` outputs the area of the blobs. The default is `true`.

`CentroidOutputPort`

Return coordinates of blob centroids

Set this property to `true` to output the coordinates of the centroid of the blobs. The default is `true`.

`BoundingBoxOutputPort`

Return coordinates of bounding boxes

Set this property to `true` to output the coordinates of the bounding boxes. The default is `true`.

`MajorAxisLengthOutputPort`

Return vector whose values represent lengths of ellipses' major axes

Set this property to `true` to output a vector whose values represent the lengths of the major axes of the ellipses that have the same normalized second central moments as the labeled regions. This property applies when you set the `OutputDataType` property to `double` or `single`. The default is `false`.

`MinorAxisLengthOutputPort`

Return vector whose values represent lengths of ellipses' minor axes

Set this property to `true` to output a vector whose values represent the lengths of the minor axes of the ellipses that have the same normalized second central moments as the labeled regions. This property is available when the `OutputDataType` property is `double` or `single`. The default is `false`.

`OrientationOutputPort`	Return vector whose values represent angles between ellipses' major axes and x-axis Set this property to true to output a vector whose values represent the angles between the major axes of the ellipses and the x-axis. This property applies when you set the `OutputDataType` property to `double` or `single`. The default is `false`.
`EccentricityOutputPort`	Return vector whose values represent ellipses' eccentricities Set this property to `true` to output a vector whose values represent the eccentricities of the ellipses that have the same second moments as the region. This property applies when you set the `OutputDataType` property to `double` or `single`. The default is `false`.
`EquivalentDiameterSquaredOutputPort`	Return vector whose values represent equivalent diameters squared Set this property to `true` to output a vector whose values represent the equivalent diameters squared. The default is `false`.
`ExtentOutputPort`	Return vector whose values represent results of dividing blob areas by bounding box areas Set this property to `true` to output a vector whose values represent the results of dividing the areas of the blobs by the area of their bounding boxes. The default is `false`.
`PerimeterOutputPort`	Return vector whose values represent estimates of blob perimeter lengths Set this property to `true` to output a vector whose values represent estimates of the perimeter lengths, in pixels, of each blob. The default is `false`.
`OutputDataType`	Output data type of statistics Specify the data type of the output statistics as `double`, `single`, or `Fixed point`. Area and bounding box outputs are always an `int32` data type. `Major axis length`, `Minor axis length`, `Orientation` and `Eccentricity` do not apply when you set this property to `Fixed point`. The default is `double`.
`Connectivity`	Which pixels are connected to each other Specify connectivity of pixels as `4` or `8`. The default is `8`.
`LabelMatrixOutputPort`	Return label matrix Set this property to `true` to output the label matrix. The default is `false`.
`MaximumCount`	Maximum number of labeled regions in each input image

Specify the maximum number of blobs in the input image as a positive scalar integer. The maximum number of blobs the object outputs depends on both the value of this property, and on the size of the input image. The number of blobs the object outputs may be limited by the input image size. The default is `50`.

`MinimumBlobArea` — Minimum blob area in pixels

Specify the minimum blob area in pixels. The default is `0`. This property is tunable.

`MaximumBlobArea` — Maximum blob area in pixels

Specify the maximum blob area in pixels. The default is `intmax ('uint32')`. This property is tunable.

`ExcludeBorderBlobs` — Exclude blobs that contain at least one border pixel

Set this property to `true` if you do not want to label blobs that contain at least one border pixel. The default is `false`.

▸ **Fixed-Point Properties**

Methods

clone	Create blob analysis object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Compute and returns statistics of input binary image

Examples

Find the centroid of a blob.

```
hblob = vision.BlobAnalysis;
hblob.AreaOutputPort = false;
hblob.BoundingBoxOutputPort = false;
img = logical([0 0 0 0 0 0; ...

       0 1 1 1 1 0; ...

       0 1 1 1 1 0; ...

       0 1 1 1 1 0; ...

       0 0 0 0 0 0]);
centroid = step(hblob, img); % [x y] coordinates of the centroid
```

Algorithms

This object implements the algorithm, inputs, and outputs described on the **Blob Analysis** block reference page. The object properties correspond to the block parameters, except:

The **Warn if maximum number of blobs is exceeded** block parameter does not have a corresponding object property. The object does not issue a warning.

- The **Output blob statistics as a variable-size signal** block parameter does not have a corresponding object property.

See Also

`vision.Autothresholder` | `vision.ConnectedComponentLabeler`

im2bw

Convert image to binary image, based on threshold

Syntax

```
BW = im2bw(I, level)
BW = im2bw(X, map, level)
BW = im2bw(RGB, level)
```

Description

`BW = im2bw(I, level)` converts the grayscale image `I` to a binary image. The output image `BW` replaces all pixels in the input image with luminance greater than `level` with the value 1 (white) and replaces all other pixels with the value 0 (black). Specify `level` in the range [0,1]. This range is relative to the signal levels possible for the image's class. Therefore, a `level` value of `0.5` is midway between black and white, regardless of class. To compute the `level` argument, you can use the function `graythresh`. If you do not specify `level`, `im2bw` uses the value `0.5`.

`BW = im2bw(X, map, level)` converts the indexed image `X` with colormap `map` to a binary image.

`BW = im2bw(RGB, level)` converts the truecolor image `RGB` to a binary image.

If the input image is not a grayscale image, `im2bw` converts the input image to grayscale, and then converts this grayscale image to binary by thresholding.

Class Support

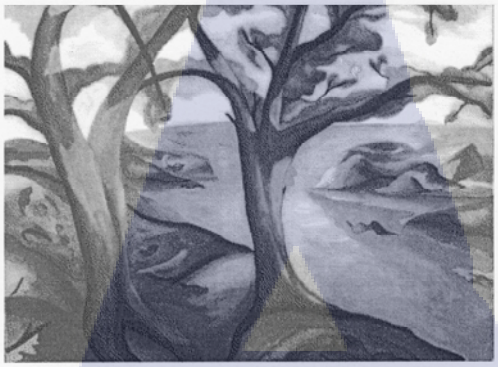
The input image can be of class `uint8`, `uint16`, `single`, `int16`, or `double`, and must be nonsparse. The output image `BW` is of class `logical`. `I` and `X` must be 2-D. RGB images are `M`-by-`N`-by-3.

Examples

collapse all

Convert an Indexed Image To a Binary Image

```
load trees
BW = im2bw(X,map,0.4);
imshow(X,map), figure, imshow(BW)
```



See Also

`graythresh` | `ind2gray` | `rgb2gray`

bwareaopen

Remove small objects from binary image

Syntax

```
BW2 = bwareaopen(BW, P)
BW2 = bwareaopen(BW, P, conn)
```

Description

`BW2 = bwareaopen(BW, P)` removes from a binary image all connected components (objects) that have fewer than `P` pixels, producing another binary image, `BW2`. This operation is known as an area opening. The default connectivity is 8 for two dimensions, 26 for three dimensions, and `conndef(ndims(BW), 'maximal')` for higher dimensions.

`BW2 = bwareaopen(BW, P, conn)` specifies the desired connectivity. `conn` can have any of the following scalar values.

Value	Meaning
Two-dimensional connectivities	
4	4-connected neighborhood
8	8-connected neighborhood
Three-dimensional connectivities	
6	6-connected neighborhood
18	18-connected neighborhood
26	26-connected neighborhood

Connectivity can be defined in a more general way for any dimension by using for `conn` a 3-by-3-by-...-by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the central element of `conn`. Note that `conn` must be symmetric about its central element.

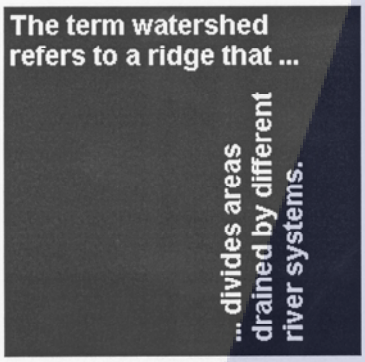
Class Support

`BW` can be a logical or numeric array of any dimension, and it must be nonsparse. The return value `BW2` is of class `logical`.

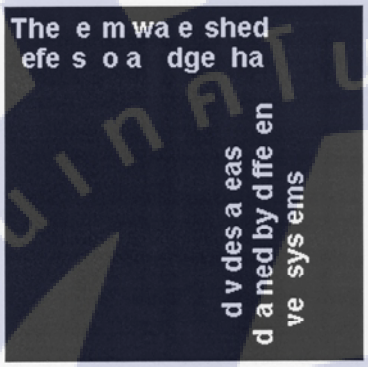
Examples

Remove all objects in the image `text.png` containing fewer than 50 pixels:

```
BW = imread('text.png');
BW2 = bwareaopen(BW, 50);
imshow(BW);
```




```
figure, imshow(BW2)
```



More About

Collapse all

Algorithms

The basic steps are

1. Determine the connected components:

```
CC = bwconncomp(BW, conn);
```

2. Compute the area of each component:

```
S = regionprops(CC, 'Area');
```

3. Remove small objects:

```
L = labelmatrix(CC);
BW2 = ismember(L, find([S.Area] >= P));
```

See Also

bwconncomp | conndef

imfill

Fill image regions and holes

Syntax

```
BW2 = imfill(BW)
[BW2,locations] = imfill(BW)
BW2 = imfill(BW,locations)
BW2 = imfill(BW,'holes')
I2 = imfill(I)
BW2 = imfill(BW,locations,conn)
[gpuarrayI2]= imfill(gpuarrayI, ___)
```

Description

`BW2 = imfill(BW)` displays the binary image `BW` on the screen and lets you define the region to fill by selecting points interactively by using the mouse. To use this interactive syntax, `BW` must be a 2-D image. Press **Backspace** or **Delete** to remove the previously selected point. A shift-click, right-click, or double-click selects a final point and starts the fill operation. Pressing **Return** finishes the selection without adding a point.

`[BW2,locations] = imfill(BW)` returns the locations of points selected interactively in `locations`. `locations` is a vector of linear indices into the input image. To use this interactive syntax, `BW` must be a 2-D image.

`BW2 = imfill(BW,locations)` performs a flood-fill operation on background pixels of the binary image `BW`, starting from the points specified in `locations`. If `locations` is a P-by-1 vector, it contains the linear indices of the starting locations. If `locations` is a `P`-by-`ndims(BW)` matrix, each row contains the array indices of one of the starting locations.

`BW2 = imfill(BW,'holes')` fills holes in the binary image `BW`. A hole is a set of background pixels that cannot be reached by filling in the background from the edge of the image.

`I2 = imfill(I)` fills holes in the grayscale image `I`. In this syntax, a hole is defined as an area of dark pixels surrounded by lighter pixels.

`BW2 = imfill(BW,locations,conn)` fills the area defined by `locations`, where `conn` specifies the connectivity. `conn` can have any of the following scalar values.

Value	Meaning
Two-dimensional connectivities	
4	4-connected neighborhood
8	8-connected neighborhood
Three-dimensional connectivities	
6	6-connected neighborhood
18	18-connected neighborhood

Value	Meaning
26	26-connected neighborhood

Connectivity can be defined in a more general way for any dimension by using for `conn` a 3-by-3-by- ... -by-3 matrix of `0`'s and `1`'s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element.

`[gpuarrayI2] = imfill(gpuarrayI, ___)` performs the fill operation on a GPU. The input image and the return image are 2-D gpuArrays. This syntax requires the Parallel Computing Toolbox™.

> **Note:** The GPU implementation of this function does not support the interactive syntaxes where you select locations.

Specifying Connectivity

By default, `imfill` uses 4-connected background neighbors for 2-D inputs and 6-connected background neighbors for 3-D inputs. For higher dimensions the default background connectivity is determined by using `conndef(NUM_DIMS,'minimal')`. You can override the default connectivity with these syntaxes:

```
BW2 = imfill(BW,locations,conn)
BW2 = imfill(BW,conn,'holes')
I2  = imfill(I,conn)
```

To override the default connectivity and interactively specify the starting locations, use this syntax:

```
BW2 = imfill(BW,0,conn)
```

Code Generation

`imfill` supports the generation of efficient, production-quality C/C++ code from MATLAB.

When generating code, note the following:

- The optional input arguments, `conn` and `'holes'` must be a compile-time constants.
- Supports only up to 3-D inputs. (No N-D support.)
- The interactive mode to select points, `imfill(BW,0,CONN)` is not supported.
- With the `locations` input argument, once you select a format at compile-time, you cannot change it at run-time. However, the number of points in locations can be varied at run-time.

Generated code for this function uses a precompiled **platform-specific (http://www.mathworks.com/support/sysreq/current_release/)** shared library. To see a complete list of toolbox functions that support code generation, see **List of Supported Functions with Usage Notes**.

Class Support

The input image can be numeric or logical, and it must be real and nonsparse. It can have any dimension. The output image has the same class as the input image.

The input gpuArray image can have its underlying class be `logical` or numeric (excluding `uint64` or `int64`), and it must be real and 2-D. The output image has the same class as the input image.

Examples

Collapse all

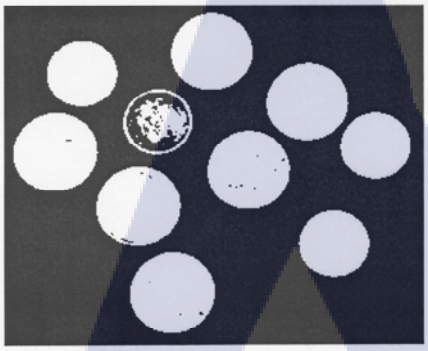
Fill Image from Specified Starting Point

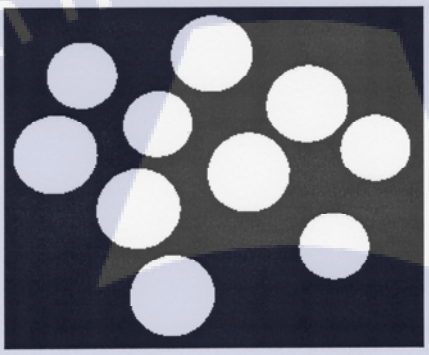
```
BW1 = logical([1 0 0 0 0 0 0 0
               1 1 1 1 1 0 0 0
               1 0 0 0 1 0 1 0
               1 0 0 0 1 1 1 0
               1 1 1 1 0 1 1 1
               1 0 0 1 1 0 1 0
               1 0 0 0 1 0 1 0
               1 0 0 0 1 1 1 0]);

BW2 = imfill(BW1,[3 3],8)
```

```
BW2 =

     1     0     0     0     0     0     0     0
     1     1     1     1     1     0     0     0
     1     1     1     1     1     0     1     0
     1     1     1     1     1     1     1     0
     1     1     1     1     1     1     1     1
     1     0     0     1     1     1     1     0
     1     0     0     0     1     1     1     0
     1     0     0     0     1     1     1     0
```

Fill Holes in a Binary Image

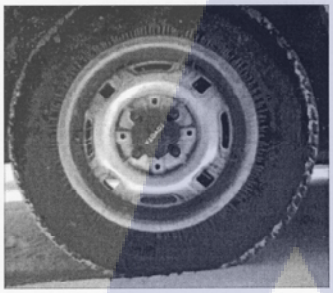
```
BW4 = im2bw(imread('coins.png'));
BW5 = imfill(BW4,'holes');
imshow(BW4), figure, imshow(BW5)
```

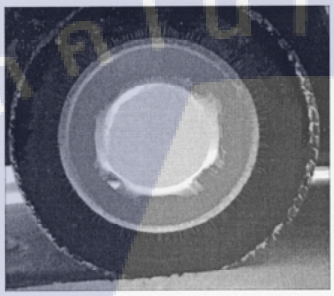




Fill Holes in a Grayscale Image

```
I = imread('tire.tif');
I2 = imfill(I,'holes');
figure, imshow(I), figure, imshow(I2)
```





Fill Operation on a GPU

Fill in the background of a binary gpuArray image from a specified starting location.

```
BW1 = logical([1 0 0 0 0 0 0 0
               1 1 1 1 1 0 0 0
               1 0 0 0 1 0 1 0
               1 0 0 0 1 1 1 0
               1 1 1 1 0 1 1 1
               1 0 0 1 1 0 1 0
               1 0 0 0 1 0 1 0
               1 0 0 0 1 1 1 0]);
BW1 = gpuArray(BW1);
BW2 = imfill(BW1,[3 3],8)
```

More About

Collapse all

Algorithms

`imfill` uses an algorithm based on morphological reconstruction [1].

References

[1] Soille, P., *Morphological Image Analysis: Principles and Applications*, Springer-Verlag, 1999, pp. 173-174.

See Also

`bwselect` | `imreconstruct` | `roifill`

imclose

Morphologically close image

Syntax

```
IM2 = imclose(IM,SE)
IM2 = imclose(IM,NHOOD)
gpuarrayIM2 = imclose(gpuarraryIM, __ )
```

Description

`IM2 = imclose(IM,SE)` performs morphological closing on the grayscale or binary image `IM`, returning the closed image, `IM2`. The structuring element, `SE`, must be a single structuring element object, as opposed to an array of objects. The morphological close operation is a dilation followed by an erosion, using the same structuring element for both operations.

`IM2 = imclose(IM,NHOOD)` performs closing with the structuring element `strel(NHOOD)`, where `NHOOD` is an array of 0's and 1's that specifies the structuring element neighborhood.

`gpuarrayIM2 = imclose(gpuarraryIM, __ )` performs the operation on a graphics processing unit (GPU), where `gpuarrayIM` is a `gpuArray` containing the grayscale or binary image. `gpuarrayIM2` is a `gpuArray` of the same class as the input image. This syntax requires the Parallel Computing Toolbox™.

Code Generation

`imclose` supports the generation of efficient, production-quality C/C++ code from MATLAB. When generating code, the image input argument, `IM`, must be 2-D or 3-D and the structuring element input argument, `SE`, must be a compile-time constant. Generated code for this function uses a precompiled **platform-specific (http://www.mathworks.com/support/sysreq/current_release/)** shared library. To see a complete list of toolbox functions that support code generation, see **List of Supported Functions with Usage Notes**.

Class Support

`IM` can be any numeric or logical class and any dimension, and must be nonsparse. If `IM` is logical, then `SE` must be flat.

`gpuarrayIM` must be a `gpuArray` of type `uint8` or `logical`. When used with a `gpuarray`, the structuring element must be flat and two-dimensional.

The output has the same class as the input.

Examples

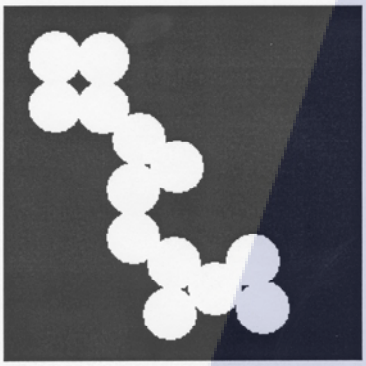
Collapse all

Use Morphological Closing to Fill Gaps in an Image

Use morphological closing to join the circles in an image together by filling in the gaps between them and by smoothing their outer edges.

Read the image into the MATLAB® workspace and view it.

```
originalBW = imread('circles.png');
imshow(originalBW);
```



Create a disk-shaped structuring element. Use a disk structuring element to preserve the circular nature of the object. Specify a radius of 10 pixels so that the largest gap gets filled.

```
se = strel('disk',10);
```

Perform a morphological close operation on the image.

```
closeBW = imclose(originalBW,se);
figure, imshow(closeBW)
```

Use Morphological Closing to Fill Gaps in an Image on a GPU

Use morphological closing to join the circles in an image together by filling in the gaps between them and by smoothing their outer edges.

Read the image into the MATLAB workspace and view it.

```
originalBW = imread('circles.png');
imshow(originalBW);
```

Create a disk-shaped structuring element. Use a disk structuring element to preserve the circular nature of the object. Specify a radius of 10 pixels so that the largest gap gets filled.

```
se = strel('disk',10);
```

Perform a morphological close operation on the image on a GPU.

```
closeBW = imclose(gpuArray(originalBW),se);
figure, imshow(closeBW)
```

See Also

gpuArray | **imdilate** | **imerode** | **imopen** | **strel**

mat2gray

Convert matrix to grayscale image

Syntax

```
I = mat2gray(A, [amin amax])
I = mat2gray(A)
gpuarrayI = mat2gray(gpuarrayA, __)
```

Description

`I = mat2gray(A, [amin amax])` converts the matrix `A` to the intensity image `I`. The returned matrix `I` contains values in the range 0.0 (black) to 1.0 (full intensity or white). `amin` and `amax` are the values in `A` that correspond to 0.0 and 1.0 in `I`. Values less than `amin` become 0.0, and values greater than `amax` become 1.0.

`I = mat2gray(A)` sets the values of `amin` and `amax` to the minimum and maximum values in `A`.

`gpuarrayI = mat2gray(gpuarrayA, __)` performs the operation on a GPU. This syntax requires the Parallel Computing Toolbox™.

Class Support

The input array `A` can be `logical` or numeric. The output image `I` is `double`.

The input gpuArray `gpuarrayA` can be `logical` or numeric. The output gpuArray image `gpuarrayI` is `double`.

Examples

Collapse all

Convert a Matrix into an Image

Read an image and, for this example, turn it into a numeric matrix.

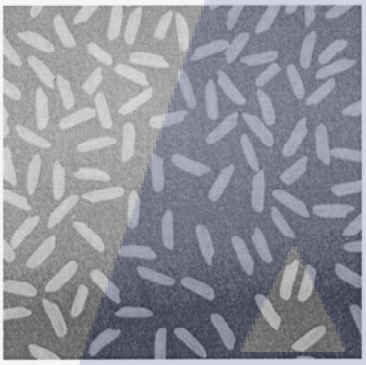
```
I = imread('rice.png');
J = filter2(fspecial('sobel'),I);
```

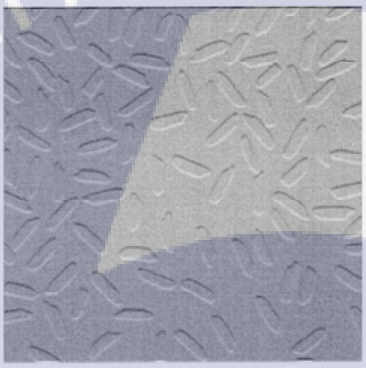
Convert the matrix into an image.

```
K = mat2gray(J);
```

Display the original image and the result of the conversion.

```
imshow(I), figure, imshow(K)
```





See Also

`gpuArray` | `gray2ind` | `ind2gray` | `rgb2gray`

entropyfilt

Local entropy of grayscale image

Syntax

```
J = entropyfilt(I)
J = entropyfilt(I,NHOOD)
```

Description

`J = entropyfilt(I)` returns the array `J`, where each output pixel contains the entropy value of the 9-by-9 neighborhood around the corresponding pixel in the input image `I`. `I` can have any dimension. If `I` has more than two dimensions, `entropyfilt` treats it as a multidimensional grayscale image and not as a truecolor (RGB) image. The output image `J` is the same size as the input image `I`.

For pixels on the borders of `I`, `entropyfilt` uses symmetric padding. In symmetric padding, the values of padding pixels are a mirror reflection of the border pixels in `I`.

`J = entropyfilt(I,NHOOD)` performs entropy filtering of the input image `I` where you specify the neighborhood in `NHOOD`. `NHOOD` is a multidimensional array of zeros and ones where the nonzero elements specify the neighbors. `NHOOD`'s size must be odd in each dimension.

By default, `entropyfilt` uses the neighborhood `true(9)`. `entropyfilt` determines the center element of the neighborhood by `floor((size(NHOOD) + 1)/2)`. To specify neighborhoods of various shapes, such as a disk, use the `strel` function to create a structuring element object and then use the `getnhood` function to extract the neighborhood from the structuring element object.

Class Support

`I` can be `logical`, `uint8`, `uint16`, or `double`, and must be real and nonsparse. `NHOOD` can be logical or numeric and must contain zeros or ones. The output array `J` is of class `double`.

`entropyfilt` converts any class other than logical to `uint8` for the histogram count calculation so that the pixel values are discrete and directly correspond to a bin value.

Examples

```
I = imread('circuit.tif');
J = entropyfilt(I);
imshow(I), figure, imshow(J,[]);
```

References

[1] Gonzalez, R.C., R.E. Woods, S.L. Eddins, *Digital Image Processing Using MATLAB*, New Jersey, Prentice Hall, 2003, Chapter 11.

See Also

`entropy` | `imhist` | `rangefilt` | `stdfilt`

imread

Read image from graphics file

Syntax

```
A = imread(filename, fmt)
[X, map] = imread(...)
[...] = imread(filename)
[...] = imread(URL,...)
[...] = imread(...,Param1,Val1,Param2,Val2...)
```

Description

`A = imread(filename, fmt)` reads a grayscale or color image from the file specified by the string `filename`. If the file is not in the current folder, or in a folder on the MATLAB® path, specify the full pathname.

The text string `fmt` specifies the format of the file by its standard file extension. For example, specify `'gif'` for Graphics Interchange Format files. To see a list of supported formats, with their file extensions, use the **`imformats`** function. If `imread` cannot find a file named `filename`, it looks for a file named `filename.fmt`.

The return value `A` is an array containing the image data. If the file contains a grayscale image, `A` is an M-by-N array. If the file contains a truecolor image, `A` is an M-by-N-by-3 array. For TIFF files containing color images that use the CMYK color space, `A` is an M-by-N-by-4 array. See **TIFF** in the Format-Specific Information section for more information.

The class of `A` depends on the bits-per-sample of the image data, rounded to the next byte boundary. For example, `imread` returns 24-bit color data as an array of `uint8` data because the sample size for each color component is 8 bits. See **Tips** for a discussion of bitdepths, and see **Format-Specific Information** for more detail about supported bit depths and sample sizes for a particular format.

`[X, map] = imread(...)` reads the indexed image in `filename` into `X` and its associated colormap into `map`. Colormap values in the image file are automatically rescaled into the range `[0,1]`.

`[...] = imread(filename)` attempts to infer the format of the file from its content.

`[...] = imread(URL,...)` reads the image from an Internet URL. The `URL` must include the protocol type (e.g., `http://`).

`[...] = imread(...,Param1,Val1,Param2,Val2...)` specifies parameters that control various characteristics of the operations for specific formats. For more information, see **Format-Specific Information**.

Format-Specific Information

The following sections provide information about the support for specific formats, listed in alphabetical order by format name. These sections include information about format-specific syntaxes, if they exist.

BMP — Windows Bitmap	**JPEG — Joint Photographic Experts Group**	**PNG — Portable Network Graphics**
CUR — Cursor File	**JPEG 2000 — Joint Photographic Experts Group 2000**	**PPM — Portable Pixmap**

GIF — Graphics Interchange Format	PBM — Portable Bitmap	RAS — Sun Raster
HDF4 — Hierarchical Data Format	PCX — Windows Paintbrush	TIFF — Tagged Image File Format
ICO — Icon File	PGM — Portable Graymap	XWD — X Window Dump

BMP — Windows Bitmap

Supported Bitdepths	No Compression	RLE Compression	Output Class	Notes
1-bit	y	–	logical	
4-bit	y	y	`uint8`	
8-bit	y	y	`uint8`	
16-bit	y	–	`uint8`	1 sample/pixel
24-bit	y	–	`uint8`	3 samples/pixel
32-bit	y	–	`uint8`	3 samples/pixel (1 byte padding)

CUR — Cursor File

Supported Bitdepths	No Compression	Compression	Output Class
1-bit	y	–	logical
4-bit	y	–	`uint8`
8-bit	y	–	`uint8`

Format-specific syntaxes:

`[...] = imread(..., idx)` reads in one image from a multi-image icon or cursor file. `idx` is an integer value that specifies the order that the image appears in the file. For example, if `idx` is `3`, `imread` reads the third image in the file. If you omit this argument, `imread` reads the first image in the file.

`[A, map, alpha] = imread(...)` returns the AND mask for the resource, which can be used to determine the transparency information. For cursor files, this mask may contain the only useful data.

> **Note** By default, Microsoft® Windows® cursors are 32-by-32 pixels. MATLAB pointers must be 16-by-16. You will probably need to scale your image. If you have Image Processing Toolbox™, you can use its `imresize` function.

GIF — Graphics Interchange Format

Supported Bitdepths	Output Class
1-bit	`logical`

Supported Bitdepths	Output Class
2-bit to 8-bit	uint8

Format-specific syntaxes:

`[...] = imread(..., idx)` reads in one or more frames from a multiframe (i.e., animated) GIF file. `idx` must be an integer scalar or vector of integer values. For example, if `idx` is `3`, `imread` reads the third image in the file. If `idx` is `1:5`, `imread` returns only the first five frames.

`[...] = imread(..., 'frames', idx)` is the same as the syntax above except that `idx` can be `'all'`. In this case, all the frames are read and returned in the order that they appear in the file.

> **Note** Because of the way that GIF files are structured, all the frames must be read when a particular frame is requested. Consequently, it is much faster to specify a vector of frames or `'all'` for `idx` than to call `imread` in a loop when reading multiple frames from the same GIF file.

HDF4 — Hierarchical Data Format

Supported Bitdepths	Raster Image with colormap	Raster image without colormap	Output Class	Notes
8-bit	y	y	uint8	
24-bit	–	y	uint8	3 samples/pixel

Format-specific syntaxes:

`[...] = imread(..., ref)` reads in one image from a multi-image HDF4 file. `ref` is an integer value that specifies the reference number used to identify the image. For example, if `ref` is `12`, `imread` reads the image whose reference number is 12. (Note that in an HDF4 file the reference numbers do not necessarily correspond to the order of the images in the file. You can use `imfinfo` to match image order with reference number.) If you omit this argument, `imread` reads the first image in the file.

ICO — Icon File

See **CUR — Cursor File**

JPEG — Joint Photographic Experts Group

`imread` can read any baseline JPEG image as well as JPEG images with some commonly used extensions. For information about support for JPEG 2000 files, see **JPEG 2000**.

Supported Bits-per-sample	Lossy Compression	Lossless Compression	Output Class	Notes
8-bit	y	y	uint8	Grayscale or RGB
12-bit	y	y	uint16	Grayscale or RGB
16-bit	–	y	uint16	Grayscale

JPEG 2000 — Joint Photographic Experts Group 2000

For information about JPEG files, see **JPEG**.

Note: Indexed JPEG 2000 images are not supported. Only JP2 compatible color spaces are supported for JP2/JPX files. By default, all image channels are returned in the order they are stored in the file.

Supported Bits-per-sample	Lossy Compression	Lossless Compression	Output Class	Notes
1-bit	y	y	`logical`	Grayscale only
2- to 8-bit	y	y	`uint8` or `int8`	Grayscale or RGB
9- to 16-bit	y	y	`uint16` or `int16`	Grayscale or RGB

Format-specific syntaxes:

`[...] = imread(..., 'Param1', value1, 'Param2', value2, ...)` uses parameter-value pairs to control the read operation, described in the following table.

Parameter	Value
`'ReductionLevel'`	A non-negative integer specifying the reduction in the resolution of the image. For a reduction level *L*, the image resolution is reduced by a factor of 2^*L*. Its default value is `0` implying no reduction. The reduction level is limited by the total number of decomposition levels as specified by the `'WaveletDecompositionLevels'` field in the structure returned by the **`imfinfo`** function.
`'PixelRegion'`	`{ROWS, COLS}` — The `imread` function returns the sub-image specified by the boundaries in `ROWS` and `COLS`. `ROWS` and `COLS` must both be two-element vectors that denote the 1-based indices `[START STOP]`. If `'ReductionLevel'` is greater than 0, then `ROWS` and `COLS` are coordinates in the reduced-sized image.
`'V79Compatible'`	A logical value. If `true`, the image returned is transformed to grayscale or RGB, consistent with previous versions of `imread` (MATLAB 7.9 [R2009b] and earlier). Use this option to transform YCC images into RGB. The default is `false`.

PBM — Portable Bitmap

Supported Bitdepths	Raw Binary	ASCII (Plain) Encoded	Output Class
1-bit	y	y	logical

PCX — Windows Paintbrush

Supported Bitdepths	Output Class	Notes
1-bit	`logical`	Grayscale only

Supported Bitdepths	Output Class	Notes
8-bit	`uint8`	Grayscale or indexed
24-bit	`uint8`	RGB Three 8-bit samples/pixel

PGM — Portable Graymap

Supported Bitdepths	Raw Binary	ASCII (Plain) Encoded	Output Class	Notes
8-bit	y	–	`uint8`	
16-bit	y	–	`uint16`	
Arbitrary	–	y	1-bit to 8-bit: `uint8` 9-bit to 16-bit: `uint16`	Values are scaled

PNG — Portable Network Graphics

Supported Bitdepths	Output Class	Notes
1-bit	`logical`	Grayscale
2-bit	`uint8`	Grayscale
4-bit	`uint8`	Grayscale
8-bit	`uint8`	Grayscale or Indexed
16-bit	`uint16`	Grayscale or Indexed
24-bit	`uint8`	RGB Three 8-bit samples/pixel.
48-bit	`uint16`	RGB Three 16-bit samples/pixel.

Format-specific syntaxes:

`[...] = imread(...,'BackgroundColor',BG)` composites any transparent pixels in the input image against the color specified in `BG`. If `BG` is `'none'`, then no compositing is performed. If the input image is indexed, `BG` must be an integer in the range `[1,P]` where `P` is the colormap length. If the input image is grayscale, `BG` should be an integer in the range `[0,1]`. If the input image is RGB, `BG` should be a three-element vector whose values are in the range `[0,1]`. The string `'BackgroundColor'` can be abbreviated.

`[A, map, alpha] = imread(...)` returns the alpha channel if one is present. If no alpha channel is present, or if you specify `'BackgroundColor'`, then `alpha` is `[]`. The `map` output can be empty if the file contains a grayscale or truecolor image.

If you specify the `alpha` output argument, `BG` defaults to `'none'`, if not specified. Otherwise, if the PNG file contains a background color chunk, that color is used as the default value for `BG`. If `alpha` is not used and the

file does not contain a background color chunk, then the default value for `BG` is `1` for indexed images; `0` for grayscale images; and `[0 0 0]` for truecolor (RGB) images.

PPM — Portable Pixmap

Supported Bitdepths	Raw Binary	ASCII (Plain) Encoded	Output Class
Up to 16-bit	y	–	`uint8`
Arbitrary	–	y	

RAS — Sun Raster

The following table lists the supported bitdepths, compression, and output classes for RAS files.

Supported Bitdepths	Output Class	Notes
1-bit	`logical`	Bitmap
8-bit	`uint8`	Indexed
24-bit	`uint8`	RGB Three 8-bit samples/pixel
32-bit	`uint8`	RGB with Alpha Four 8-bit samples/pixel

TIFF — Tagged Image File Format

Most images supported by the TIFF specification or LibTIFF can be read by `imread`.

`imread` supports the following TIFF capabilities:

- Any number of samples-per-pixel
- CCITT group 3 and 4 FAX, Packbits, JPEG, LZW, Deflate, ThunderScan compression, and uncompressed images
- Logical, grayscale, indexed color, truecolor and hyperspectral images
- RGB, CMYK, CIELAB, ICCLAB color spaces. If the color image uses the CMYK color space, A is an M-by-N -by-4 array. To determine which color space is used, use `imfinfo` to get information about the graphics file and look at the value of the `PhotometricInterpretation` field. If a file contains CIELAB color data, `imread` converts it to ICCLAB before bringing it into the MATLAB workspace because 8- or 16-bit TIFF CIELAB-encoded values use a mixture of signed and unsigned data types that cannot be represented as a single MATLAB array.
- Data organized into tiles or scanlines

Note:

- YCbCr images are converted into the RGB colorspace.
- All grayscale images are read as if black=0, white=largest value.
- 1-bit images are returned as class `logical`.
- CIELab images are converted into ICCLab colorspace.

The following are format-specific syntaxes for TIFF files.

`A = imread(...)` returns color data that uses the RGB, CIELAB, ICCLAB, or CMYK color spaces. If the color image uses the CMYK color space, A is an M-by-N-by-4 array.

`[...] = imread(..., 'Param1', value1, 'Param2', value2, ...)` uses parameter/value pairs to control the read operation. The following table lists the parameters you can use.

Parameter	Value
`'Index'`	Positive integer specifying which image to read. For example, if you specify the value `3`, `imread` reads the third image in the file. If you omit this argument, `imread` reads the first image in the file.
`'Info'`	Structure array returned by `imfinfo`. Note: When reading images from a multi-image TIFF file, passing the output of `imfinfo` as the value of the `'Info'` argument helps `imread` locate the images in the file more quickly.
`'PixelRegion'`	Cell array, `{Rows, Cols}`, specifying the boundaries of the region. `Rows` and `Cols` must be either two- or three-element vectors. If you specify two elements, the values denote the 1-based indices `[start stop]`. If you specify three elements, the values denote the 1-based indices `[start increment stop]`, to allow image downsampling.

For copyright information, see the `libtiffcopyright.txt` file.

XWD — X Window Dump

The following table lists the supported bitdepths, compression, and output classes for XWD files.

Supported Bitdepths	ZPixmaps	XYBitmaps	XYPixmaps	Output Class
1-bit	y	–	y	`logical`
8-bit	y	–	–	`uint8`

Class Support

For most image file formats, `imread` uses 8 or fewer bits per color plane to store image pixels. The following table lists the class of the returned array for the data types used by the file formats.

Data Type Used in File	Class of Array Returned by imread
1-bit per pixel	`logical`
2- to 8-bits per color plane	`uint8`
9- to 16-bit per pixel	`uint16` (`BMP`, `JPEG`, `PNG`, and `TIFF`) For the 16-bit BMP packed format (5-6-5), MATLAB returns `uint8`

Note For indexed images, `imread` always reads the colormap into an array of class `double`, even though the image array itself may be of class `uint8` or `uint16`.

Examples

expand all

Read and Display Image

Convert Indexed Image to RGB

Read Specific Image in Multi-Page TIFF File

Return Alpha Channel of PNG Image

Read Specified Region of TIFF Image

More About

expand all

Tips

See Also

`double` | `fread` | `image` | `imfinfo` | `imformats` | `imwrite` | `ind2rgb` | `uint16` | `uint8`

imopen

Morphologically open image

Syntax

```
IM2 = imopen(IM,SE)
IM2 = imopen(IM,NHOOD)
gpuarrayIM2 = imopen(gpuarrayIM, ___)
```

Description

`IM2 = imopen(IM,SE)` performs morphological opening on the grayscale or binary image `IM` with the structuring element `SE`. The argument `SE` must be a single structuring element object, as opposed to an array of objects. The morphological open operation is an erosion followed by a dilation, using the same structuring element for both operations.

`IM2 = imopen(IM,NHOOD)` performs opening with the structuring element `strel(NHOOD)`, where `NHOOD` is an array of 0's and 1's that specifies the structuring element neighborhood.

`gpuarrayIM2 = imopen(gpuarrayIM, ___)` performs the operation on a graphics processing unit (GPU) with the structuring element `strel(NHOOD)`, if `NHOOD` is an array of `0`s and `1`s that specifies the structuring element neighborhood, or `strel(gather(NHOOD))` if `NHOOD` is a `gpuArray` object that specifies the structuring element neighborhood. This syntax requires the Parallel Computing Toolbox™.

Code Generation

`imopen` supports the generation of efficient, production-quality C/C++ code from MATLAB. When generating code, the image input argument, `IM`, must be 2-D or 3-D and the structuring element input argument, `SE`, must be a compile-time constant. Generated code for this function uses a precompiled **platform-specific (http://www.mathworks.com/support/sysreq/current_release/)** shared library. To see a complete list of toolbox functions that support code generation, see **List of Supported Functions with Usage Notes**.

Class Support

`IM` can be any numeric or logical class and any dimension, and must be nonsparse. If `IM` is logical, then `SE` must be flat.

`gpuarrayIM` must be a `gpuArray` of type `uint8` or `logical`. When used with a `gpuarray`, the structuring element must be flat and two-dimensional.

The output has the same class as the input.

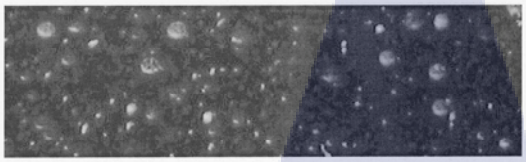
Examples

collapse all

Morphologically Open Image with a Disk-shaped Structuring Element

Read the image into the MATLAB® workspace and display it.

```
original = imread('snowflakes.png');
figure, imshow(original);
```

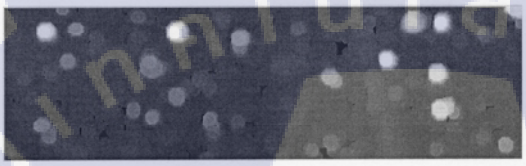


Create a disk-shaped structuring element with a radius of 5 pixels.

```
se = strel('disk',5);
```

Remove snowflakes having a radius less than 5 pixels by opening it with the disk-shaped structuring element.

```
afterOpening = imopen(original,se);
figure, imshow(afterOpening,[]);
```



Morphologically Open Image with Disk-shaped Structuring Element on a GPU

Read an image.

```
original = imread('snowflakes.png');
```

Create a disk-shaped structuring element.

```
se = strel('disk',5);
```

Morphologically open the image on a GPU, using a `gpuArray` object, and display the images.

```
afterOpening = imopen(gpuArray(original),se);
figure, imshow(original), figure, imshow(afterOpening,[])
```

See Also

gpuArray | **imclose** | **imdilate** | **imerode** | **strel**

imwrite

Write image to graphics file

Syntax

```
imwrite(A,filename)
```
example

```
imwrite(A,map,filename)
```
example

```
imwrite(___,fmt)
```
example

```
imwrite(___,Name,Value)
```
example

Description

example

`imwrite(A,filename)` writes image data `A` to the file specified by `filename`, inferring the file format from the extension. `imwrite` creates the new file in your current folder. The bit depth of the output image depends on the data type of `A` and the file format. For most formats:

- If `A` is of data type `uint8`, then `imwrite` outputs 8-bit values.
- If `A` is of data type `uint16` and the output file format supports 16-bit data (JPEG, PNG, and TIFF), then `imwrite` outputs 16-bit values. If the output file format does not support 16-bit data, then `imwrite` errors.
- If `A` is a grayscale or RGB color image of data type `double` or `single`, then `imwrite` assumes the dynamic range is [0,1] and automatically scales the data by 255 before writing it to the file as 8-bit values. If the data in `A` is `single`, convert `A` to `double` before writing to a GIF or TIFF file.
- If `A` is of data type `logical`, then `imwrite` assumes the data is a binary image and writes it to the file with a bit depth of 1, if the format allows it. BMP, PNG, or TIFF formats accept binary images as input arrays.

If `A` contains indexed image data, you should additionally specify the `map` input argument.

example

`imwrite(A,map,filename)` writes the indexed image in `A` and its associated colormap, `map`, to the file specified by `filename`.

- If `A` is an indexed image of data type `double` or `single`, then `imwrite` converts the indices to zero-based indices by subtracting 1 from each element, and then writes the data as `uint8`. If the data in `A` is `single`, convert `A` to `double` before writing to a GIF or TIFF file.

example

`imwrite(__,fmt)` writes the image in the format specified by `fmt`, regardless of the file extension in `filename`. You can specify `fmt` after the input arguments in any of the previous syntaxes.

example

`imwrite(__,Name,Value)` specifies additional parameters for output GIF, HDF, JPEG, PBM, PGM, PNG, PPM, and TIFF files, using one or more name-value pair arguments, in additional to any of the arguments in the previous syntaxes.

Examples

expand all

Resize and Save Image

Write Indexed Image Data to PNG

Write Indexed Image with MATLAB Colormap

Write Grayscale Image to PNG

Write Truecolor Image to JPEG

Write Multiple Images to TIFF File

Write Animated GIF

Input Arguments

expand all

`A` — Image data to write
matrix

`filename` — Name of output file
string

`map` — Colormap of indexed image
`m`-by-3 array

`fmt` — Format of output file
string

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `imwrite(A,'myFile.png','BitDepth',8)` writes the data in `A` using 8 bits to represent each pixel.

GIF — Graphics Interchange Format

`'BackgroundColor'` — Color to use as background color
scalar integer

`'Comment'` — Comment to add to image
string | cell array of strings

`'DelayTime'` — Delay before displaying next image
0.5 (default) | scalar value in the range [0,655]

'DisposalMethod' — Disposal method of animated GIF
'doNotSpecify' (default) | 'leaveInPlace' | 'restoreBG' | 'restorePrevious'

'Location' — Offset of screen relative to image
[0,0] (default) | two-element vector

'LoopCount' — Number of times to repeat animation
Inf (default) | integer in the range [0,65535]

'ScreenSize' — Height and width of frame
height and width of input image (default) | two-element vector

'TransparentColor' — Color to use as transparent color
scalar integer

'WriteMode' — Writing mode
'overwrite' (default) | 'append'

HDF4 — Hierarchical Data Format

'Compression' — Compression scheme
'none' (default) | 'jpeg' | 'rle'

'Quality' — Quality of JPEG-compressed file
75 (default) | scalar in the range [0,100]

'WriteMode' — Writing mode
'overwrite' (default) | 'append'

JPEG — Joint Photographic Experts Group

'BitDepth' — Number of bits per pixel
8 (default) | scalar

'Comment' — Comment to add to image
string | character array | n-by-1 cell array of strings

'Mode' — Type of compression
'lossy' (default) | 'lossless'

'Quality' — Quality of output file
75 (default) | scalar in the range [0,100]

JPEG 2000— Joint Photographic Experts Group 2000

'Comment' — Comment to add to image
string | character array | cell array of strings

'CompressionRatio' — Target compression ratio
1 (default) | scalar

'Mode' — Type of compression
'lossy' (default) | 'lossless'

'ProgressionOrder' — Order of packets in code stream
'LRCP' (default) | 'RLCP' | 'RPCL' | 'PCRL' | 'CPRL'

'QualityLayers' — Number of quality layers

1 (default) | integer in the range [1,20]

'ReductionLevels' — Number of reduction levels
4 (default) | integer in the range [1,8]

'TileSize' — Tile height and width
image size (default) | two-element vector

PBM-, PGM-, and PPM — Portable Bitmap, Graymap, Pixmap

'Encoding' — Encoding
'rawbits' (default) | 'ASCII'

'MaxValue' — Maximum gray or color value
scalar

PNG — Portable Network Graphics

In addition to the following name-value pair arguments, you can use any parameter name that satisfies the PNG specification for keywords. That is, the name uses only printable characters, contains 80 or fewer characters, and does not contain leading or trailing spaces. The value corresponding to these user-specified names must be a string that contains no control characters other than linefeed.

'Alpha' — Transparency of each pixel
matrix of values in the range [0,1]

'Author' — Author information
string

'Background' — Background color when compositing transparent pixels
scalar in the range [0,1] | integer in the range [1,P] | 3-element vector in the range [0,1]

'BitDepth' — Number of bits per pixel
scalar

'Chromaticities' — Reference white point and primary chromaticities
8-element vector

'Comment' — Comment to add to image
string

'Copyright' — Copyright notice
string

'CreationTime' — Time of original image creation
string

'Description' — Description of image
string

'Disclaimer' — Legal disclaimer
string

'Gamma' — File gamma
scalar

'ImageModTime' — Time of last image modification
serial date number | date string

'InterlaceType' — Interlacing scheme

`'none'` (default) | `'adam7'`

`'ResolutionUnit'` — Unit for image resolution
`'unknown'` (default) | `'meter'`

`'SignificantBits'` — Number of bits to regard as significant
`[]` (default) | scalar | vector

`'Software'` — Software used to create the image
string

`'Source'` — Device used to create the image
string

`'Transparency'` — Pixels to consider transparent
`[]` (default) | scalar in the range [0,1] | vector

`'Warning'` — Warning of nature of content
string

`'XResolution'` — Image resolution in horizontal direction
scalar

`'YResolution'` — Image resolution in vertical direction
scalar

RAS — Sun Raster Graphic

`'Alpha'` — Transparency of each pixel
`[]` (default) | matrix

`'Type'` — Image type
`'standard'` (default) | `'rgb'` | `'rle'`

TIFF — Tagged Image File Format

`'ColorSpace'` — Color space representing color data
`'rgb'` (default) | `'cielab'` | `'icclab'`

`'Compression'` — Compression scheme
`'packbits'` | `'none'` | `'lzw'` | `'deflate'` | `'jpeg'` | `'ccitt'` | `'fax3'` | `'fax4'`

`'Description'` — Image description
string

`'Resolution'` — X- and Y-resolution
72 (default) | scalar | two-element vector

`'RowsPerStrip'` — Number of rows to include in each strip
scalar

`'WriteMode'` — Writing mode
`'overwrite'` (default) | `'append'`

More About

Expand all

Tips

See Also

`fwrite` | `getframe` | `imfinfo` | `imformats` | `imread` | `Tiff`

imadjust

Adjust image intensity values or colormap

Syntax

```
J = imadjust(I)
J = imadjust(I,[low_in; high_in],[low_out; high_out])
J = imadjust(I,[low_in; high_in],[low_out; high_out],gamma)
newmap = imadjust(map,[low_in; high_in],[low_out; high_out],gamma)
RGB2 = imadjust(RGB1,___)
gpuarrayB = imadjust(gpuarrayA,___)
```

Description

`J = imadjust(I)` maps the intensity values in grayscale image `I` to new values in `J` such that 1% of data is saturated at low and high intensities of `I`. This increases the contrast of the output image `J`. This syntax is equivalent to `imadjust(I,stretchlim(I))`.

`J = imadjust(I,[low_in; high_in],[low_out; high_out])` maps the values in `I` to new values in `J` such that values between `low_in` and `high_in` map to values between `low_out` and `high_out`. Values for `low_in`, `high_in`, `low_out`, and `high_out` must be between 0 and 1. Values below `low_in` and above `high_in` are clipped; that is, values below `low_in` map to `low_out`, and those above `high_in` map to `high_out`. You can use an empty matrix (`[]`) for [`low_in high_in`] or for [`low_out high_out`] to specify the default of `[0 1]`.

`J = imadjust(I,[low_in; high_in],[low_out; high_out],gamma)` maps the values in `I` to new values in `J`, where `gamma` specifies the shape of the curve describing the relationship between the values in `I` and `J`. If `gamma` is less than 1, the mapping is weighted toward higher (brighter) output values. If `gamma` is greater than 1, the mapping is weighted toward lower (darker) output values. If you omit the argument, `gamma` defaults to 1 (linear mapping).

`newmap = imadjust(map,[low_in; high_in],[low_out; high_out],gamma)` transforms the colormap associated with an indexed image. If `low_in`, `high_in`, `low_out`, `high_out`, and `gamma` are scalars, then the same mapping applies to red, green, and blue components. Unique mappings for each color component are possible when

`low_in` and `high_in` are both 1-by-3 vectors.

`low_out` and `high_out` are both 1-by-3 vectors, or `gamma` is a 1-by-3 vector.

The rescaled colormap `newmap` is the same size as `map`.

`RGB2 = imadjust(RGB1,___)` performs the adjustment on each image plane (red, green, and blue) of the RGB image `RGB1`. As with the colormap adjustment, you can apply unique mappings to each plane.

`gpuarrayB = imadjust(gpuarrayA,___)` performs the adjustment on a GPU. The input gpuArray `gpuarrayA` is an intensity image, RGB image, or a colormap. The output gpuArray `gpuarrayB` is the same as the input image. This syntax requires the Parallel Computing Toolbox™.

Note If `high_out < low_out`, the output image is reversed, as in a photographic negative.

Class Support

The input image `I` can be of class `uint8`, `uint16`, `int16`, `single`, or `double`. The output image `J` has the same class as the input image. `map` and `newmap` are of class `double`.

The input gpuArray intensity or RGB image `gpuarrayA` can be of class `uint8`, `uint16`, `int16`, `single`, or `double`. The output gpuArray image `gpuarrayJ` has the same class as the input image. The `map` and `newmap` gpuArrays are of class `double`..

Examples

Adjust a low-contrast grayscale image.

```
I = imread('pout.tif');
J = imadjust(I);
imshow(I), figure, imshow(J)
```

Now adjust the same image, this time specifying contrast limits.

```
K = imadjust(I,[0.3 0.7],[]);
figure, imshow(K)
```

Adjust an RGB image.

```
RGB1 = imread('football.jpg');
RGB2 = imadjust(RGB1,[.2 .3 0; .6 .7 1],[]);
imshow(RGB1), figure, imshow(RGB2)
```

Adjust a low-contrast grayscale image on a GPU.

```
I = gpuArray(imread('pout.tif'));
J = imadjust(I);
figure, imshow(I), figure, imshow(J)
```

Now adjust the same image on a GPU, this time specifying contrast limits.

```
K = imadjust(I,[0.3 0.7],[]);
figure, imshow(K)
```

Adjust an RGB image on a GPU.

```
RGB1 = gpuArray(imread('football.jpg'));
RGB2 = imadjust(RGB1,[.2 .3 0; .6 .7 1],[]);
figure, imshow(RGB1), figure, imshow(RGB2)
```

See Also

brighten | **gpuArray** | **histeq** | **stretchlim**

figure

Create figure graphics object

Syntax

```
figure
figure
figure('PropertyName',propertyvalue,...)
figure(h)
h = figure(...)
```

Properties

For a list of properties, see **`Figure Properties`**.

Description

`figure` creates figure graphics objects. Figure objects are the individual windows on the screen in which the MATLAB® software displays graphical output.

`figure` creates a new figure object using default property values. This automatically becomes the current figure and raises it above all other figures on the screen until a new figure is created or called.

`figure` creates a new figure object using default property values. This automatically becomes the current figure and raises it above all other figures on the screen until a new figure is created or called.

Number property has been assigned the smallest positive integer not already assigned to another Figure. This number also appears in the new Figure's title bar.

`figure('PropertyName',propertyvalue,...)` creates a new figure object using the values of the properties specified. For a description of the properties, see **`Figure Properties`**. MATLAB uses default values for any properties that you do not explicitly define as arguments.

`figure(h)` does one of the following (assuming `IntegerHandle` is its default value, `on`):

- If `h` is the handle to an existing figure, `figure(h)` makes the figure identified by `h` the current figure, makes it visible, and attempts to raise it above all other figures on the screen. The current figure is the target for graphics output.
- If `h` is not the handle to an existing figure, but is an integer, `figure(h)` creates a figure and assigns it the handle `h`.
- If `h` is not the handle to a figure, and is not an integer, MATLAB returns an error.

`h = figure(...)` returns the handle to the figure object.

Examples

Specifying Figure Size and Screen Location

To create a figure window that is one quarter the size of your screen and is positioned in the upper left corner, use the root object's `ScreenSize` property to determine the size. `ScreenSize` is a four-element vector: `[left, bottom, width, height]`:

```
scrsz = get(0,'ScreenSize');
figure('Position',[1 scrsz(4)/2 scrsz(3)/2 scrsz(4)/2])
```

To position the full figure window including the menu bar, title bar, tool bars, and outer edges, use the `OuterPosition` property in the same manner.

Specifying the Figure Window Title

You can add your own title to a figure by setting the `Name` property and you can turn off the figure number with the `NumberTitle` property:

```
figure('Name','Simulation Plot Window','NumberTitle','off')
```

See **`Figure Properties`** for a description of all properties.

Setting Default Properties

You can set default figure properties only on the **`rootobject`** level.

```
set(0,'DefaultFigureProperty',PropertyValue...)
```

where *`Property`* is the name of the figure property and `PropertyValue` is the value you are specifying. Use **`set`** and **`get`** to access figure properties.

See **Setting Default Property Values** for more information.

More About

Collapse all

Tips

To create a figure object, MATLAB creates a new window whose characteristics are controlled by default figure properties (both factory installed and user defined) and properties specified as arguments. See **`Figure Properties`** for a description of these properties.

You can specify properties as property name/property value pairs, structure arrays, and cell arrays (see the **`set`** and **`get`** reference pages for examples of how to specify these data types).

Use `set` to modify the properties of an existing figure or `get` to query the current values of figure properties.

The **`gcf`** command returns the handle to the current figure and is useful as an argument to the `set` and `get` commands.

Figures can be docked in the desktop. The `DockControls` property determines whether you can dock the figure.

Making a Figure Current

The current figure is the target for graphics output. There are two ways to make a figure `h` the current figure.

- Make the figure `h` current, visible, and displayed on top of other figures:

```
figure(h);
```

- Make the figure `h` current, but do not change its visibility or stacking with respect to other figures:

```
set(0,'CurrentFigure',h);
```

See Also

axes | clf | close | Figure Properties | gcf | ishghandle | rootobject | uicontrol | uimenu

if, elseif, else

Execute statements if condition is true

Syntax

```
if expression
   statements
elseif expression
   statements
else
   statements
end
```

Description

`if` *`expression`*, *`statements`*, `end` evaluates an expression, and executes a group of statements when the expression is true.

`elseif` and `else` are optional, and execute statements only when previous expressions in the `if` block are false. An `if` block can include multiple `elseif` statements.

An evaluated expression is true when the result is nonempty and contains all nonzero elements (logical or real numeric). Otherwise, the expression is false.

Expressions can include relational operators (such as `<` or `==`) and logical operators (such as `&&`, `||`, or `~`). MATLAB® evaluates compound expressions from left to right, adhering to **operator precedence** rules.

> **Note:** Within the condition *`expression`* of an `if` or `while` statement, logical operators `&` and `|` behave as short-circuit operators. This behavior is the same as `&&` and `||`, respectively. Since `&&` and `||` consistently short-circuit in `if` and `while` condition expressions and statements, it is good practice to use `&&` and `||` instead of `&` and `|` within *`expression`*.

Examples

Assign to a matrix values that depend on their indices.

```
% Preallocate a matrix
nrows = 10;
ncols = 10;
myData = ones(nrows, ncols);

% Loop through the matrix
for r = 1:nrows
   for c = 1:ncols

      if r == c
         myData(r,c) = 2;
      elseif abs(r - c) == 1
         myData(r,c) = -1;
      else
```

```
            myData(r,c) = 0;
        end

    end
end
```

Respond to command-line input. Because the input string could be more than one character, use `strcmp` rather than `==` to test for equality.

```
reply = input('Would you like to see an echo? (y/n): ', 's');
if strcmp(reply,'y')
  disp(reply)
end
```

Find the indices of values in a vector that are greater than a specified limit.

```
A = rand(1,10);
limit = .75;

B = (A > limit);   % B is a vector of logical values
if any(B)
  fprintf('Indices of values > %4.2f: \n', limit);
  disp(find(B))
else
  disp('All values are below the limit.')
end
```

Concatenate two variables when they are the same size. To avoid an error when the variables have different dimensions, compare the sizes using `isequal` rather than the `==` operator.

```
A = ones(2,3);              % Two-dimensional array
B = rand(3,4,5);            % Three-dimensional array

if isequal(size(A), size(B))
   C = [A; B];
else
   warning('A and B are not the same size.');
   C = [];
end
```

Take advantage of short-circuiting to avoid error or warning messages.

```
x = 42;
if exist('myfunction.m') && (myfunction(x) >= pi)
   disp('Condition is true')
end
```

More About

Collapse all

Tips

- You can nest any number of `if` statements. Each `if` statement requires an `end` keyword.
- Avoid adding a space within the `elseif` keyword (`else if`). The space creates a nested `if` statement that requires its own `end` keyword.

- **Relational Operators**

See Also

`for` | `Logical Operators: Short Circuit` | `return` | `switch` | `while`

numel

Number of array elements

Syntax

```
n = numel(A)
```

example

Description

example

`n = numel(A)` returns the number of elements, `n`, in array `A`, equivalent to `prod(size(A))`.

Examples

collapse all

Number of Elements in 3-D Matrix

Create a 4-by-4-by-2 matrix.

```
A = magic(4);
A(:,:,2) = A'
```

```
A(:,:,1) =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

A(:,:,2) =
    16     5     9     4
     2    11     7    14
     3    10     6    15
    13     8    12     1
```

`numel` counts 32 elements in the matrix.

```
n = numel(A)
```

```
n =

    32
```

Number of Elements in Cell Array of Strings

Create a cell array of strings.

```
A = {'dog','cat','fish','horse'};
```

`numel` counts 4 string elements in the array.

```
n = numel(A)
```

```
n =

     4
```

Number of Elements in Table

Create a table with four variables listing patient information for five people.

```
LastName = {'Smith';'Johnson';'Williams';'Jones';'Brown'};
Age = [38;43;38;40;49];
Height = [71;69;64;67;64];
Weight = [176;163;131;133;119];
BloodPressure = [124 93; 109 77; 125 83; 117 75; 122 80];

A = table(Age,Height,Weight,BloodPressure,'RowNames',LastName)
```

```
A =

                Age    Height    Weight    BloodPressure
                ---    ------    ------    -------------

    Smith       38     71        176       124          93
    Johnson     43     69        163       109          77
    Williams    38     64        131       125          83
    Jones       40     67        133       117          75
    Brown       49     64        119       122          80
```

Find the number of elements in the table.

```
n = numel(A)
```

```
n =

    20
```

`numel` returns a value equivalent to `prod(size(A))` corresponding to the 5 rows and 4 variables.

Input Arguments

collapse all

A — Input array

scalar | vector | matrix | multidimensional array

Input array, specified as a scalar, vector, matrix, or multidimensional array. This includes numeric arrays, logical arrays, character arrays, categorical arrays, tables, structure arrays, cell arrays, and object arrays.

Limitations

- If `A` is a table, `numel` returns the number of elements in the table, `A`, equivalent to `prod(size(A))`. Variables in a table can have multiple columns, but `numel(A)` only accounts for the number of rows and number of variables.

More About

- **Caution When Customizing Classes**

See Also

`prod` | `size` | `subsref`

imshow

Display image

Syntax

```
imshow(I)
```
example

```
imshow(I,RI)
```
example

```
imshow(X,map)
```
example

```
imshow(X,RX,map)
```

```
imshow(filename)
```
example

```
imshow(___,Name,Value...)
```

```
imshow(gpuarrayIM, ___)
```
example

```
imshow(I,[low high])
```
example

```
himage = imshow(___)
```

Description

example

`imshow(I)` displays the image `I` in a Handle Graphics® figure, where `I` is a grayscale, RGB (truecolor), or binary image. For binary images, `imshow` displays pixels with the value `0` (zero) as black and `1` as white.

example

`imshow(I,RI)` displays the image `I` with associated 2-D spatial referencing object `RI`.

example

`imshow(X,map)` displays the indexed image `X` with the colormap `map`. A color map matrix may have any number of rows, but it must have exactly 3 columns. Each row is interpreted as a color, with the first element specifying the intensity of red light, the second green, and the third blue. Color intensity can be specified on the interval 0.0 to 1.0.

`imshow(X,RX,map)` displays the indexed image `X` with associated 2-D spatial referencing object `RX` and colormap `MAP`.

example

`imshow(filename)` displays the image stored in the graphics file `filename`. The file must be in the current directory or on the MATLAB® path and must contain an image that can be read

by `imread` or `dicomread`. `imshow` calls `imread` or `dicomread` to read the image from the file, but does not store the image data in the MATLAB workspace. If the file contains multiple images, `imshow` displays only the first one.

`imshow(__,Name,Value...)` displays the image, specifying additional options with one or more `Name,Value` pair arguments, using any of the previous syntaxes.

example

`imshow(gpuarrayIM, __)` displays the image contained in a `gpuArray`. This syntax requires the Parallel Computing Toolbox™.

example

`imshow(I,[low high])` displays the grayscale image `I`, specifying the display range as a two-element vector, `[low high]`. For more information, see the`DisplayRange` parameter.

`himage = imshow(__)` returns the handle to the image object created by `imshow`.

Examples

Collapse all

Display image from file

Specify image file.

```
imshow('board.tif')
```

Display indexed image

Read indexed image and associated color map from file and display it.

```
[X,map] = imread('trees.tif');
imshow(X,map)
```

Display grayscale image

Read grayscale image from file and display it.

```
I = imread('cameraman.tif');
imshow(I)
```

Display grayscale image, adjusting display range

Read grayscale image and specify display range.

```
I = imread('cameraman.tif');
h = imshow(I,[0 80]);
```

Display grayscale image using associated spatial referencing object

Read image into workspace.

```
I = imread('pout.tif');
```

Create a spatial referencing object associated with the image. Then specify *X* and *Y* limits in a world coordinate system.

```
RI = imref2d(size(I));
RI.XWorldLimits = [0 3];
RI.YWorldLimits = [2 5];
```

Display the image, specifying the spatial referencing object.

```
imshow(I,RI);
```

Display Image on a GPU

Read image into a `gpuArray`.

```
X = gpuArray(imread('pout.tif'));
```

Display it.

```
figure; imshow(X)
```

Input Arguments

collapse all

`I` — Input image

grayscale image | RGB image | binary image

Input image, specified as a grayscale, RGB, or binary image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`X` — Indexed image

2-D array of real numeric values.

Indexed image, specified as a 2-D array of real numeric values. The values in `X` are an index into `map`, an *n*-by-3 array of RGB values.

Data Types: `single` | `double` | `uint8` | `logical`

`map` — Colormap

n-by-3 array

Colormap, specified as an *n*-by-3 array. Each row specified an RGB color value.

Data Types: `single` | `double` | `uint8` | `logical`

`filename` — Name of file containing an image
text string

Name of file containing an image, specified as a text string. The image must be readable by `imread` or by `dicomread`. `imshow` calls `imread` or `dicomread` to read the image from the file, but does not store the image data in the MATLAB workspace. If the file contains multiple images, `imshow` displays the first image in the file.

Data Types: `char`

`RI` — 2-D spatial referencing object associated with the input image
`imref2d` object

2-D spatial referencing object associated with input image, specified as an `imref2d` object .

`RX` — 2-D spatial referencing object associated with an indexed image
`imref2d` object

2-D spatial referencing object associated with an indexed image, specified as a `imref2d` object.

`gpuarrayIM` — Image to be processed on a graphics processing unit (GPU)
`gpuArray` object

Image to be processed on a graphics processing unit (GPU), specified as a `gpuArray`.

`[low high]` — Display range of the image
two-element vector

Display range of the image, specified as a two-element vector.

Example: `[50 250]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Border','tight'`

`'Border'` — Control whether figure window includes a border
value returned by `iptgetpref('ImshowBorder')` (default) | `'tight'` or `'loose'`

Controls whether `imshow` includes a border around the image displayed in the figure window. `'tight'` or `'loose'`There can still be a border if the image is very small, or if there are other objects besides the image and its axes in the figure.

Example:

Data Types: `char`

`'Colormap'` — Value of figure's colormap property
(default) | 2–D, real, *m*-by-3 matrix

`imshow` uses this to set the figure's `colormap` property. Use this parameter to view grayscale images in false color. If you specify an empty colormap (`[]`), `imshow` ignores this parameter.

Example:

Data Types: `double`

`'DisplayRange'` — Display range of grayscale image
`[min(I(:)) max(I(:))]` (default) | two-element vector

Display range of a grayscale image, specified as a two-element vector `[LOW HIGH]`. `imshow` displays the value `low` (and any value less than `low`) as black, and the value `high` (and any value greater than `high`) as white. Values in between are displayed as intermediate shades of gray, using the default number of gray levels. If you specify an empty matrix (`[]`), `imshow` uses `[min(I(:)) max(I(:))]`; that is, use the minimum value in `I` as black, and the maximum value as white.

> **Note:** Including the parameter name is optional, except when the image is specified by a filename. The syntax `imshow(I,[LOW HIGH])` is equivalent to `imshow(I,'DisplayRange',[LOW HIGH])`. When calling `imshow` with a filename, you must specify the `'DisplayRange'` parameter.

Example: `h = imshow(I,'DisplayRange',[0 80]);`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`'InitialMagnification'` — Initial magnification used to display image
value returned by `iptgetpref('ImshowInitialMagnification')` (default) | numeric scalar value | text string `'fit'`

Initial magnification used to display an image, specified as the numeric value or the text string `'fit'`.

When set to 100, `imshow` displays the image at 100% magnification (one screen pixel for each image pixel). When set to `'fit'`, `imshow` scales the entire image to fit in the window.

On initial display, `imshow` always displays the entire image. If the magnification value is large enough that the image would be too big to display on the screen, `imshow` warns and displays the image at the largest magnification that fits on the screen.

If the image is displayed in a figure with its `'WindowStyle'` property set to `'docked'`, `imshow` warns and displays the image at the largest magnification that fits in the figure.

Note: If you specify the axes position (using `subplot` or `axes`), `imshow` ignores any initial magnification you might have specified and defaults to the `'fit'` behavior.

When used with the `'Reduce'` parameter, only `'fit'` is allowed as an initial magnification.

Example:

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char`

`'Parent'` — Axes that is the parent of the image object
handle

Axes that is the parent of the image object, specified as a handle. Created by `imshow`.

> **Note:** If you are building a GUI where you want to control the figure and axes properties, be sure to use the `imshow(..., 'Parent', ax)` syntax.

Data Types: `function_handle`

`'Reduce'` — Subsample image in `filename`
logical value

Subsample image in `filename`, specified as a logical value. Only valid for TIFF images. Use this parameter to display overviews of very large images.

Data Types: `logical`

`'Xdata'` — Limits along X axis of a nondefault spatial coordinate system
two-element vector

Limits along X axis of a nondefault spatial coordinate system, specified as a two-element vector. Establishes a nondefault spatial coordinate system by specifying the image `XData`. The value can have more than two elements, but only the first and last elements are actually used.

Example: `[100 200]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`'YData'` — Limits along Y axis of a nondefault spatial coordinate system
two-element vector

Limits along Y axis of a nondefault spatial coordinate system, specified as a two-element vector. The value can have more than two elements, but only the first and last elements are actually used.

Example: `[100 200]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

collapse all

`himage` — Image object created by `imshow`
handle

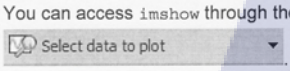
Image object created by `imshow`, specified as a handle.

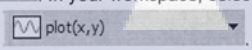
Alternative Functionality

App

`imshow` is the toolbox's fundamental image display function, optimizing figure, axes, and image object property settings for image display. `imtool` provides all the image display capabilities of `imshow` but also provides access to several other tools for navigating and exploring images, such as the Pixel Region tool, Image Information tool, and the Adjust Contrast tool. `imtool` presents an integrated environment for displaying images and performing some common image processing tasks.

The `imshow` function is not supported when MATLAB is started with the `-nojvm` option.

You can access `imshow` through the Plot Selector workspace tool, which is represented by this icon: 

. In your workspace, select the data you want to display. The Plot Selector icon changes to look like this: 

. Scroll down to **Image Processing Toolbox Plots**. Select **imshow(I)**.

More About

Collapse all

Tips

- You can use the `iptsetpref` function to set several toolbox preferences that modify the behavior of `imshow`.
  - `'ImshowBorder'` controls whether `imshow` displays the image with a border around it.
  - `'ImshowAxesVisible'` controls whether `imshow` displays the image with the axes box and tick labels.
  - `'ImshowInitialMagnification'` controls the initial magnification for image display, unless you override it in a particular call by specifying `imshow(...,'InitialMagnification',initial_mag)`.

  For more information about these preferences, see `iptprefs`.

See Also

`gpuArray` | `image` | `imagesc` | `imread` | `imtool` | `iptprefs` | `subimage` | `truesize` | `warp`

step

Step response plot of dynamic system

Syntax

```
step(sys)
step(sys,Tfinal)
step(sys,t)
step(sys1,sys2,...,sysN)
step(sys1,sys2,...,sysN,Tfinal)
step(sys1,sys2,...,sysN,t)
y = step(sys,t)
[y,t] = step(sys)
[y,t] = step(sys,Tfinal)
[y,t,x] = step(sys)
[y,t,x,ysd] = step(sys)
[y,...] = step(sys,...,options)
```

Description

`step` calculates the step response of a dynamic system. For the state space case, zero initial state is assumed. When it is invoked with no output arguments, this function plots the step response on the screen.

`step(sys)` plots the step response of an arbitrary **dynamic system model** `sys`. This model can be continuous or discrete, and SISO or MIMO. The step response of multi-input systems is the collection of step responses for each input channel. The duration of simulation is determined automatically, based on the system poles and zeros.

`step(sys,Tfinal)` simulates the step response from `t = 0` to the final time `t = Tfinal`. Express `Tfinal` in the system time units, specified in the `TimeUnit` property of `sys`. For discrete-time systems with unspecified sampling time (`Ts = -1`), `step` interprets `Tfinal` as the number of sampling periods to simulate.

`step(sys,t)` uses the user-supplied time vector `t` for simulation. Express `t` in the system time units, specified in the `TimeUnit` property of `sys`. For discrete-time models, `t` should be of the form `Ti:Ts:Tf`, where `Ts` is the sample time. For continuous-time models, `t` should be of the form `Ti:dt:Tf`, where `dt` becomes the sample time of a discrete approximation to the continuous system (see **Algorithms**). The `step` command always applies the step input at `t=0`, regardless of `Ti`.

To plot the step response of several models `sys1`,..., `sysN` on a single figure, use

```
step(sys1,sys2,...,sysN)
```

```
step(sys1,sys2,...,sysN,Tfinal)
```

```
step(sys1,sys2,...,sysN,t)
```

All of the systems plotted on a single plot must have the same number of inputs and outputs. You can, however, plot a mix of continuous- and discrete-time systems on a single plot. This syntax is useful to compare the step responses of multiple systems.

You can also specify a distinctive color, linestyle, marker, or all three for each system. For example,

```
step(sys1,'y:',sys2,'g--')
```

plots the step response of `sys1` with a dotted yellow line and the step response of `sys2` with a green dashed line.

When invoked with output arguments:

```
y = step(sys,t)

[y,t] = step(sys)

[y,t] = step(sys,Tfinal)

[y,t,x] = step(sys)
```

`step` returns the output response `y`, the time vector `t` used for simulation (if not supplied as an input argument), and the state trajectories `x` (for state-space models only). No plot generates on the screen. For single-input systems, `y` has as many rows as time samples (length of `t`), and as many columns as outputs. In the multi-input case, the step responses of each input channel are stacked up along the third dimension of `y`. The dimensions of `y` are then

$$(length\ of\ t) \times (number\ of\ outputs) \times (number\ of\ inputs)$$

and `y(:,:,j)` gives the response to a unit step command injected in the `j`th input channel. Similarly, the dimensions of `x` are

$$(length\ of\ t) \times (number\ of\ states) \times (number\ of\ inputs)$$

For identified models (see `idlti` and `idnlmodlel`) `[y,t,x,ysd] = step(sys)` also computes the standard deviation `ysd` of the response `y` (`ysd` is empty if `sys` does not contain parameter covariance information).

`[y,...] = step(sys,...,options)` specifies additional options for computing the step response, such as the step amplitude or input offset. Use **`stepDataOptions`** to create the option set `options`.

Examples

Example 1

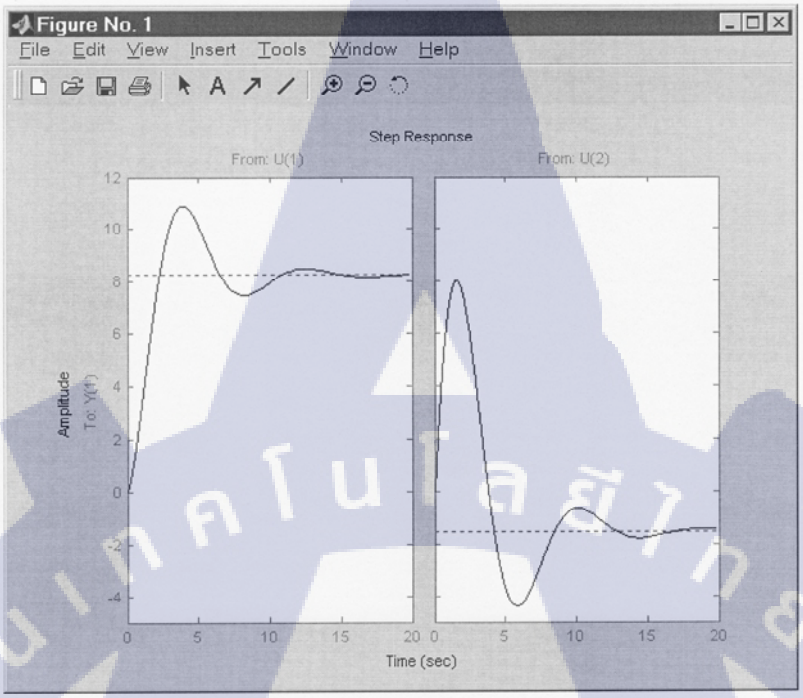
Step Response Plot of Dynamic System

Plot the step response of the following second-order state-space model.

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}-0.5572 & -0.7814\\ 0.7814 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}1 & -1\\ 0 & 2\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

$$y = \begin{bmatrix}1.9691 & 6.4493\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

```
a = [-0.5572   -0.7814;0.7814   0];
b = [1 -1;0 2];
c = [1.9691   6.4493];
sys = ss(a,b,c,0);
step(sys)
```

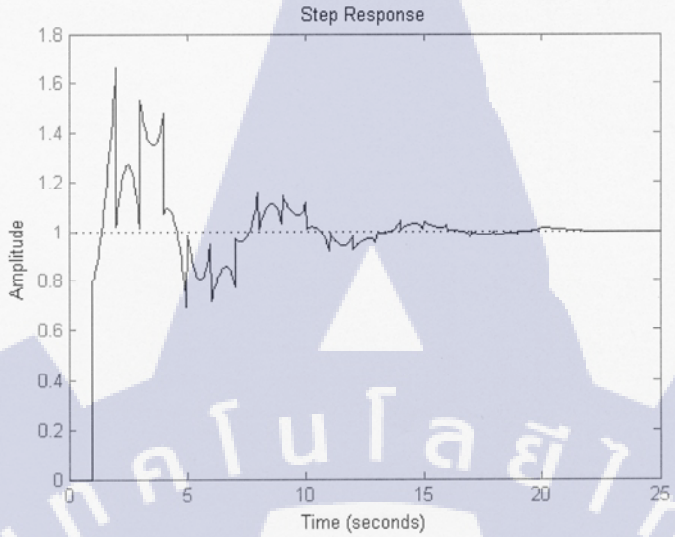


The left plot shows the step response of the first input channel, and the right plot shows the step response of the second input channel.

Step Response Plot of Feedback Loop with Delay

Create a feedback loop with delay and plot its step response.

```
s = tf('s');
G = exp(-s) * (0.8*s^2+s+2)/(s^2+s);
T = feedback(ss(G),1);
step(T)
```



The system step response displayed is chaotic. The step response of systems with internal delays may exhibit odd behavior, such as recurring jumps. Such behavior is a feature of the system and not software anomalies.

Example 3

Compare the step response of a parametric identified model to a non-parametric (empirical) model/ Also view their *3-σ* confidence regions.

```
load iddata1 z1
sys1 = ssest(z1,4);
```

parametric model

```
sys2 = impulseest(z1);
```

non-parametric model

```
[y1, ~, ~, ysd1] = step(sys1,t);
[y2, ~, ~, ysd2] = step(sys2,t);

plot(t, y1, 'b', t, y1+3*ysd1, 'b:', t, y1-3*ysd1, 'b:')
hold on
plot(t, y2, 'g', t, y2+3*ysd2, 'g:', t, y2-3*ysd2, 'g:')
```

Example 4

Validation the linearization of a nonlinear ARX model by comparing their small amplitude step responses.

```
nlsys = nlarx(z2,[4 3 10],'tree','custom',...
  {'sin(y1(t-2)*u1(t))+y1(t-2)*u1(t)+u1(t).*u1(t-13)',...
   'y1(t-5)*y1(t-5)*y1(t-1)'},'nlr',[1:5, 7 9]);
```

Determine an equilibrium operating point for `nlsys` corresponding to a steady-state input value of 1:

```
u0 = 1;
[X,~,r] = findop(nlsys, 'steady', 1);
y0 = r.SignalLevels.Output;
```

Obtain a linear approximation of `nlsys` at this operating point.

```
sys = linearize(nlsys,u0,X)
```

Now validate the usefulness of `sys` by comparing its small-amplitude step response to that of `nlsys`. The nonlinear system `nlsys` is operating an equilibrium level dictated by (`u0, y0`). About this steady-state, we introduce a step perturbation of size `0.1`. The corresponding response is computed as follows:

```
opt = stepDataOptions;
opt.InputOffset = u0;
opt.StepAmplitude = 0.1;
t = (0:0.1:10)';

ynl = step(nlsys, t, opt);
```

The linear system `sys` expresses the relationship between the perturbations in input to the corresponding perturbation in output. It is unaware of nonlinear system's equilibrium values. The step response of the linear system is:

```
opt = stepDataOptions;
opt.StepAmplitude = 0.1;
yl = step(sys, t, opt);
```

To compare, add the steady-state offset, `y0`, to the response of the linear system:

```
plot(t, ynl, t, yl+y0)
legend('Nonlinear', 'Linear with offset')
```

Example 5

Compute the step response of an identified time series model.

A time series model, also called a signal model, is one without measured input signals. The step plot of this model uses its (unmeasured) noise channel as the input channel to which the step signal is applied.

```
load iddata9
sys = ar(z9, 4);
```

`ys` is a model of the form `A y(t) = e(t)`, where `e(t)` represents the noise channel. For computation of step response, `e(t)` is treated as an input channel, and is named `"e@y1"`.

```
step(sys)
```

More About

Collapse all

Tips

You can change the properties of your plot, for example the units. For information on the ways to change properties of your plots, see **Ways to Customize Plots**.

Algorithms

Continuous-time models without internal delays are converted to state space and discretized using zero-order hold on the inputs. The sampling period, `dt`, is chosen automatically based on the system dynamics, except when a time vector `t = 0:dt:Tf` is supplied (`dt` is then used as sampling period). The resulting simulation time steps `t` are equisampled with spacing `dt`.

For systems with internal delays, Control System Toolbox™ software uses variable step solvers. As a result, the time steps `t` are not equisampled.

References

[1] L.F. Shampine and P. Gahinet, "Delay-differential-algebraic equations in control theory," *Applied Numerical Mathematics*, Vol. 56, Issues 3–4, pp. 574–588.

See Also

`impulse` | `initial` | `lsim` | `ltiview` | `stepDataOptions`

sum

Sum of array elements

Syntax

```
S = sum(A)
```
example

```
S = sum(A,dim)
```
example

```
S = sum(__,type)
```
example

Description

example

`S = sum(A)` returns the sum of the elements of A along the first array dimension whose size does not equal 1:

- If `A` is a vector, then `sum(A)` returns the sum of the elements.
- If `A` is a nonempty, nonvector matrix, then `sum(A)` treats the columns of `A` as vectors and returns a row vector whose elements are the sums of each column.
- If `A` is an empty 0-by-0 matrix, then `sum(A)` returns `0`, a 1-by-1 matrix.
- If `A` is a multidimensional array, then `sum(A)` treats the values along the first array dimension whose size does not equal 1 as vectors and returns an array of row vectors. The size of this dimension becomes `1` while the sizes of all other dimensions remain the same.

example

`S = sum(A,dim)` sums the elements of `A` along dimension `dim`. The `dim` input is a positive integer scalar.

example

`S = sum(__,type)` accumulates in and returns an array in the class specified by `type`, using any of the input arguments in the previous syntaxes. `type` can be `'double'` or `'native'`.

Examples

Collapse all

Sum of Vector Elements

Create a vector and compute the sum of the elements.

```
A = 1:10;
S = sum(A)
```

```
S =

    55
```

Sum of Elements in Each Column

Create a 3-by-3 matrix.

```
A = [1 3 2; 4 2 5; 6 1 4]
```

```
A =

     1     3     2
     4     2     5
     6     1     4
```

Compute the sum of the elements in each column.

```
S = sum(A)
```

```
S =

    11     6    11
```

Sum of Elements in Each Row

Create a 3-by-3 matrix.

```
A = [1 3 2; 4 2 5; 6 1 4]
```

```
A =

     1     3     2
     4     2     5
     6     1     4
```

Compute the sum of the elements in each row.

```
S = sum(A,2)
```

```
S =

     6
    11
    11
```

Sum of Elements in Each Plane

Create a 4-by-2-by-3 multidimensional array of ones.

```
A = ones(4,2,3)
```

```
A(:,:,1) =

     1     1
     1     1
     1     1
     1     1


A(:,:,2) =

     1     1
     1     1
     1     1
     1     1


A(:,:,3) =

     1     1
     1     1
     1     1
     1     1
```

Compute the sum along the third `(dim = 3)` dimension.

```
S = sum(A,3)
```

```
S =

     3     3
     3     3
     3     3
     3     3
```

The length of the first dimension matches `size(A,1)`, the length of the second dimension matches `size(A,2)`, and the length of the third dimension is `1`.

Integer Overflow Due to Output Data Type

Create a vector of signed 8-bit integers and compute the sum of the elements.

```
A = int8(1:20);
S = sum(A)
```

```
S =

   210
```

The output has data type `double` because of default behavior.

```
class(S)
```

```
ans =

double
```

To keep the same data type in the output as in the input, specify `type` as `'native'`.

```
S = sum(A,'native')
```

```
S =

   127
```

This disagrees with the double-precision value of 210 because the output has data type `int8` and is saturated.

```
class(S)
```

```
ans =

int8
```

Input Arguments

Collapse all

`A` — Input array

scalar | vector | matrix | multidimensional array

Input array, specified as a scalar, vector, matrix, or multidimensional array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char`

Complex Number Support: Yes

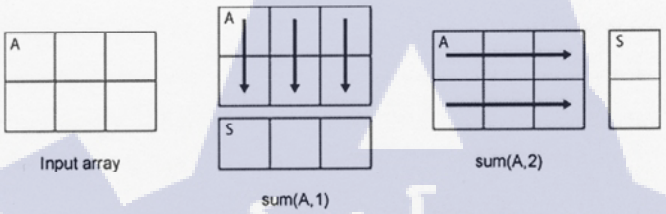
`dim` — Dimension to operate along

positive integer scalar

Dimension to operate along, specified as a positive integer scalar. If no value is specified, the default is the first array dimension whose size does not equal 1.

Consider a two-dimensional input array, `A`:

- `sum(A,1)` works on successive elements in the columns of `A` and returns a row vector of the sums of each column.
- `sum(A,2)` works on successive elements in the rows of `A` and returns a column vector of the sums of each row.



`sum` returns `A` if `dim` is greater than `ndims(A)`.

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`type` — Output class

`'double'` | `'native'`

Output class, specified as `'double'` or `'native'`, defines the data type that the operation is performed in and returned in.

- If `type` is `'double'`, then `sum` computes and returns a double-precision array, regardless of the input data type. For example, if `A` is `single`, then `sum` accumulates in and returns in `double`. This is the default behavior for integer data types when `type` is not specified.
- If `type` is `'native'`, `sum` accumulates natively and returns an array with the same data type as the input array `A`. For example, if `A` has data type `int8`, then `sum(A,'native')` accumulates in and returns in `int8`. This is the default behavior for `single` and `double` data types when `type` is not specified.

Data Types: `char`

Output Arguments

collapse all

`S` — Sum array

scalar | vector | matrix | multidimensional array

Sum array, returned as a scalar, vector, matrix, or multidimensional array. The dimension of `A` acted on by `sum` has size `1` in `S`.

The class of `S` is as follows:

- If the `type` argument is not used and the input is not `single`, then the output is `double`.

- If the `type` argument is not used and the input is `single`, then the output is `single`.
- If the `type` argument specifies `'double'`, then the output is `double` regardless of the input data type.
- If the `type` argument specifies `'native'`, then the output is the same data type as the input.

See Also

cumsum | **diff** | **isfloat** | **prod**

size

Array dimensions

Syntax

```
d = size(X)
[m,n] = size(X)
m = size(X,dim)
[d1,d2,d3,...,dn] = size(X),
```

Description

`d = size(X)` returns the sizes of each dimension of array `X` in a vector, `d`, with `ndims(X)` elements.

- If `X` is a scalar, then `size(X)` returns the vector `[1 1]`. Scalars are regarded as a 1-by-1 arrays in MATLAB®.
- If `X` is a table, `size(X)` returns a two-element row vector consisting of the number of rows and the number of variables in the table. Variables in the table can have multiple columns, but `size` only counts the variables and rows.

`[m,n] = size(X)` returns the size of matrix `X` in separate variables `m` and `n`.

`m = size(X,dim)` returns the size of the dimension of `X` specified by scalar `dim`.

`[d1,d2,d3,...,dn] = size(X)`, for `n > 1`, returns the sizes of the dimensions of the array `X` in the variables d1,d2,d3,...,dn, provided the number of output arguments `n` equals `ndims(X)`. If `n` does not equal `ndims(X)`, the following exceptions hold:

`n < ndims(X)`	`di` equals the size of the `i`th dimension of `X` for `0<i<n`, but `dn` equals the product of the sizes of the remaining dimensions of `X`, that is, dimensions `n` through `ndims(X)`.
`n > ndims(X)`	`size` returns ones in the "extra" variables, that is, those corresponding to `ndims(X)+1` through `n`.

Note For a Java array, `size` returns the length of the Java array as the number of rows. The number of columns is always 1. For a Java array of arrays, the result describes only the top level array.

Examples

expand all

Size of Dimensions in 3-D Array

Size of Table

Specify Different Number of Outputs than `ndims(X)`

See Also

`exist` | `length` | `ndims` | `numel` | `whos`

Appendix - B

Circuit Diagram