กล้องตรวจจับคุณภาพวัตถุในโรงงาน โดย Ni-MyRio

Camera Detector By Ni-MyRio

โดย
นายสิรวิชญ์ บุญลอย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมโทรคมนาคม
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2558

กล้องตรวจจับคุณภาพวัตถุในโรงงาน โดย Ni-MyRio

Camera Detector By Ni-MyRio

โดย

นาย สิรวิชญ์ บุญลอย 55011312

อาจารย์ที่ปรึกษา

ผศ.ดร. สิรภพ ตู้ประกาย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2558

ผ่านการตรวจรูปเล่มแล้ว

(.....)
อาจารย์ที่ปรึกษา
13/06/59

KMITL วิศวกรรมโทรคมนาคม Telecommunications Engineering

ผ่านการตรวจชิ้นงานแล้ว

(.....)
กรรมการผู้ตรวจชิ้นงาน
13/06/59

KMITL วิศวกรรมโทรคมนาคม Telecommunications Engineering

ปริญญานิพนธ์ปีการศึกษา 2558

สาขาวิชาโทรคมนาคม

คณะ วิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **กล้องตรวจจับคุณภาพวัตถุในโรงงาน โดย Ni-MyRio**

Camera Detector By Ni-MyRio

ผู้จัดทำ

1. นาย สิรวิชญ์ บุญลอย 55011312

..... อาจารย์ที่ปรึกษา

(ผศ.ดร. สิรภพ ตู้ประกาย)

กิตติกรรมประกาศ

ขอกราบขอบพระคุณบิดา มารดา ที่ให้การสนับสนุนและให้กำลังใจตลอดการศึกษา ขอขอบพระคุณ ผศ.ดร. สิรภพ ตู้ประกาย และ รศ.ดร.กอบชัย เดชหาญ ที่ให้คำแนะนำและช่วยเหลือสนับสนุนที่เป็นประโยชน์ต่อการทำโครงงานนี้ให้สำเร็จลุล่วง ขอขอบพระคุณ คุณเอกลักษณ์ เล็กเลิศศิริวงศ์ ที่ให้คำแนะนำในการใช้โปรแกรมต่าง และไมโครคอนโทรเลอร์ ที่เป็นประโยชน์ในการทำโครงงาน และขอขอบพระคุณพี่ๆ น้องๆ และเพื่อนๆ ที่ให้การช่วยเหลือ ให้คำปรึกษาต่างๆ เกี่ยวกับการทำโครงงานนี้ ผู้จัดทำโครงงานรู้สึกซาบซึ้งในความอนุเคราะห์จากท่านและขอกราบขอบพระคุณเป็นอย่างสูง

นายสิรวิชญ์ บุญลอย รหัสนักศึกษา 55011312

ระบบตรวจสอบคุณภาพวัสดุภัณฑ์ในโรงงาน

Quality monitoring system in plant materials.

โดย

นาย สิรวิชญ์ บุญลอย 5501131

อาจารย์ที่ปรึกษา ผศ.ดร. สิรภพ ตู้ประกาย

บทคัดย่อ

บทคัดย่อ โครงงานนี้จัดทำขึ้นเพื่อศึกษาการตรวจสอบคุณภาพชิ้นงาน และคัดแยกชิ้นงานที่ไม่มีคุณภาพออกจากกันได้ โดยชิ้นงานจะเป็นกล่องบรรจุภัณฑ์ มีความยาว 8.2เซนติเมตร มีความกว้าง 5.8 เซนติเมตร และสูง 2.8 เซนติเมตร โดยจะตรวจสอบจากกล้องคอมพิวเตอร์ จะแบ่งเป็น 2 ส่วนคือ 1. โปรแกรม Labview จะใช้ในการประมวลภาพอัตโนมัติ และส่วนที่ 2 คือ ไมโครคอนโทลเลอร์ Ni-MyRio ใช้ในการเชื่อต่อกับโปรแกรม Labview

Abstract

Abstract— This project is designed to determine the quality of the workpiece. And separate pieces that no quality apart. The Product is packaging box material is 8.2 cm in length with a width of 5.8 cm and 2.8 cm high. by a check from the webcam camera into two parts: 1. Labview software is used to process the image automatically, and the second. the microcontroller Ni-MyRio in connection with Labview.

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญรูป

หน้า

รูปที่

สารบัญรูป(ต่อ)

หน้า

รูปที่

สารบัญรูป(ต่อ)

หน้า

รูปที่

สารบัญตาราง

หน้า

ตารางที่

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันการพัฒนาเทคโนโลยีทางด้านการตรวจสอบคุณภาพวัสดุภัณฑ์ในโรงงานนั้น มีการพัฒนาไปอย่างรวดเร็ว และมีความต้องการทางด้านประสิทธิภาพการทำงานที่สูงขึ้น มีความรวดเร็วในการทำงานมากขึ้น มีความแม่นยำสูงขึ้น ประกอบกับต้นทุนที่ต่ำลง จึงทำให้ต้องมีการพัฒนาทางด้านเทคโนโลยีในการตรวจสอบคุณภาพ ซึ่งในการตรวจสอบนั้น มีทั้งการตรวจสอบในการวัดขนาด การวัดจำนวน และการอ่านตัวอักษร ซึ่งจะส่งผลให้เพิ่มประสิทธิภาพในการผลิต ผลิตภัณฑ์ต่างๆ การตรวจสอบคุณภาพด้วยกล้อง เป็นที่นิยมอย่างมาก และมีความสำคัญในโลกปัจจุบันมาก โดยในปัจจุบันมีความต้องการในการซื้อของผู้บริโภคเป็นจำนวนเพิ่มขึ้นเรื่อยๆ โดยมีความต้องการสินค้าที่มีคุณภาพ และราคาเหมาะสม แต่เกิดข้อจำกัดของการผลิตทำให้ไม่สามรถตอบสนองความต้องการของผู้บริโภคได้ เช่น มีความล่าช้าการผลิตสินค้า สินค้าที่ผลิตได้ไม่ตรงตามมาตรฐาน ราคาต้นทุนในการผลิตสูงจึงเป็นผลทำให้ราคาสินค้าสูง เป็นต้น จึงได้มีการพัฒนาเทคโนโลยีในการผลิตสินค้า เพื่อปรับปรุง และแก้ไขปัญหาดังกล่าว ได้อย่างมีประสิทธิภาพมากขึ้น

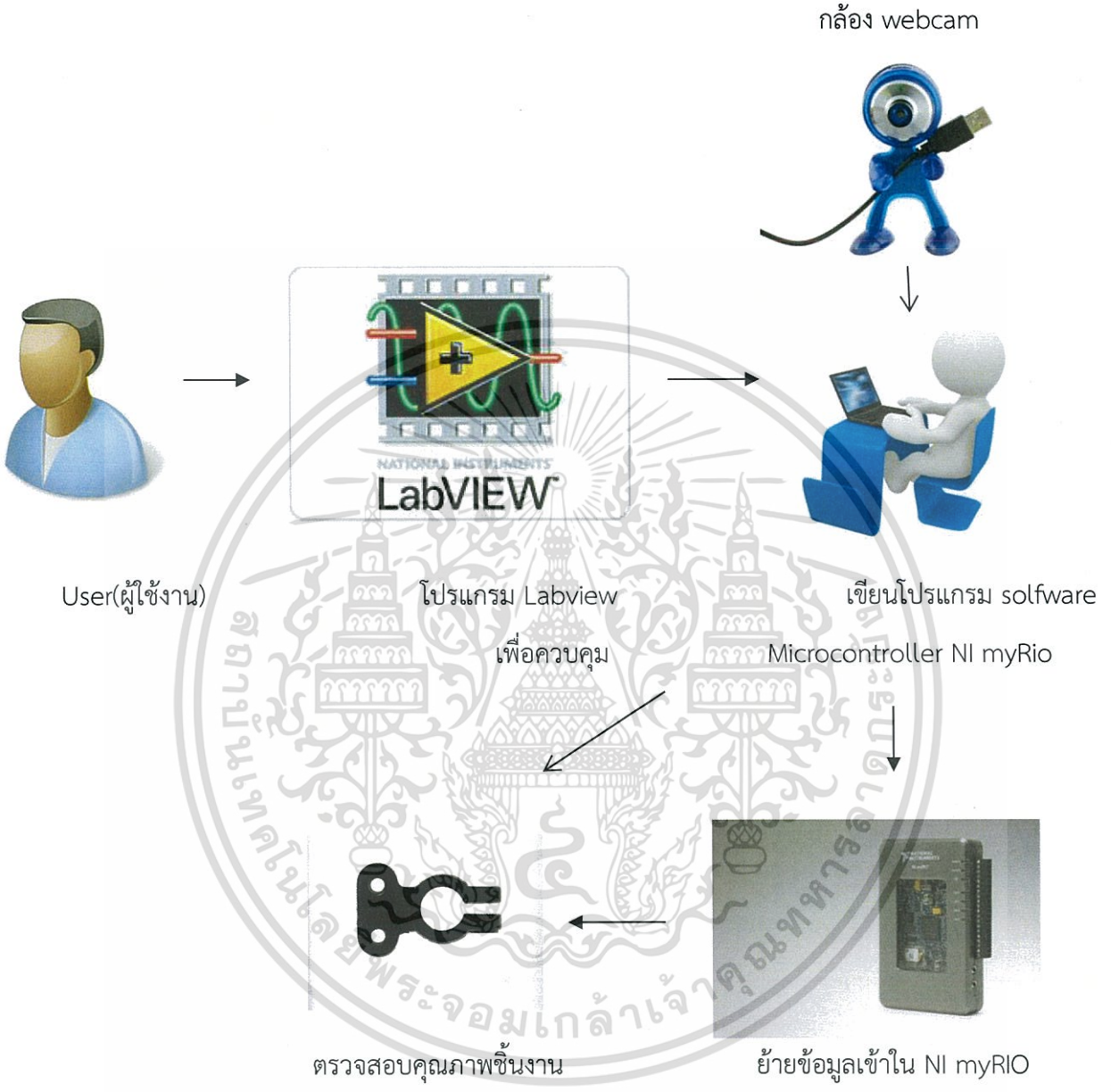
1.2 วัตถุประสงค์ของโครงงาน

1.เพื่อให้นักศึกษาในการใช้ NI myRIO และ โปรแกม Labview

2.เพื่อศึกษา เป็นแนวทางในการตรวจสอบคุณภาพของวัตถุเมื่อเทียบกับต้นแบบ

3.เพื่อศึกษา เป็นแนวทางในการ สร้างอุปกรณ์ตรวจสอบคุณภาพของ ผลิตภัณฑ์ ที่ผลิตจากโรงงานอุตสาหกรรมให้มีมาตรฐานตรงตามที่ต้องการ

1.3 ขอบเขตโครงงาน



รูปที่ 1.1 Block Diagram ของโครงงาน

โครงงานนี้เป็นสร้างอุปกรณ์ เพื่อตรวจสอบคุณภาพชิ้นงานให้ตรงตามมาตรฐาน โดยเขียน solfware โดยใช้โปรแกรม Labview แล้วใช้รูปที่ถ่ายจากจากกล้อง webcam นำมาประมวลผลในโปรแกรม Labview เพื่อตรวจสอบขนาดของชิ้นงานให้ตรงตามมาตรฐานที่ต้องการ โดยสามารถใช้ microcontroller NI myRIO เป็นตัวเชื่อมในการตรวจสอบชิ้นงานได้ เพื่อเป็นพื้นฐานในการออกแบบ และพัฒนา โปรแกรมในการสร้างระบบเพื่อตรวจสอบคุณภาพชิ้นงานหรือผลิตภัณฑ์ ต่อไปในอนาคต

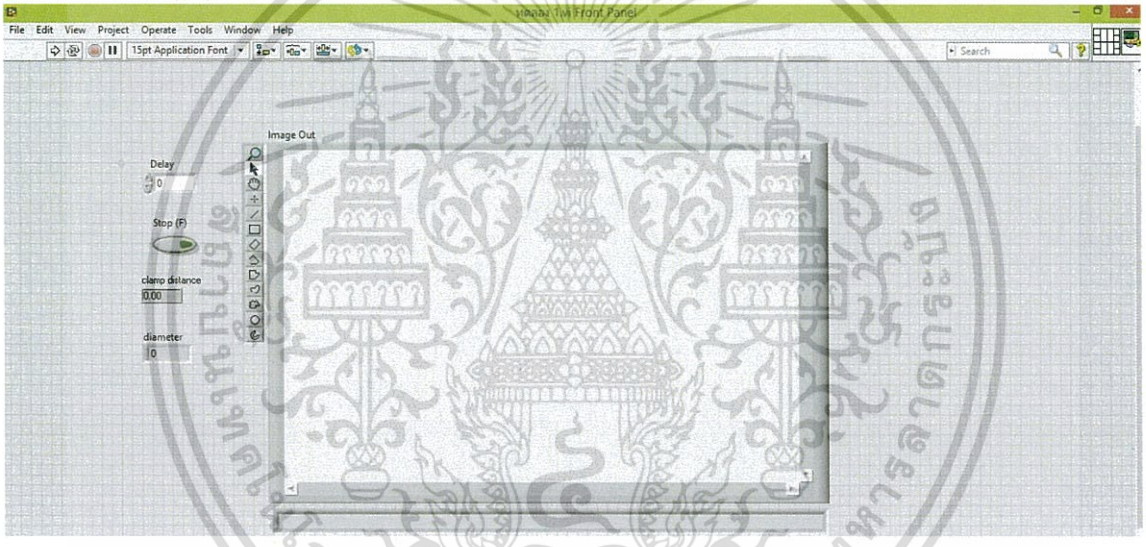
บทที่ 2

ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1.LabVIEW

2.1.1 โปรแกรม LabVIEW

LabVIEW ย่อ มาจาก Laboratory Virtual Instrument Engineering Work-bench LABVIEW จะเรียกว่า Virtual Instrument หรือจะเรียกย่อ ๆ ว่า VI จากรูปนี้เป็น Oscilloscope ที่ได้ทำการสร้างขึ้นบนหน้าจอ คอมพิวเตอร์

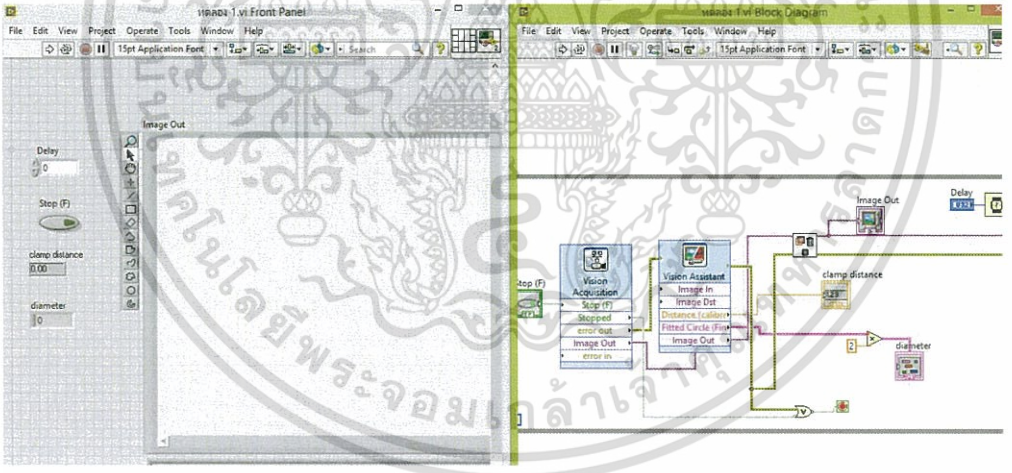


รูปที่ 2.1 แสดงตัวอย่างเครื่องมือวัดเสมือนที่สร้างจาก LabVIEW

LabVIEW มีจุดกำเนิดในปี1983 โดยทางบริษัท National Instrument และในปี 1986 บริษัทได้ ปล่อย LabVIEW Version 1 สู่ตลาดเพื่อใช้กับคอมพิวเตอร์ Macintosh เท่านั้น เพราะแม้ว่า เครื่อง Macintosh จะไม่เป็นที่ใช้อย่างกว้างขวางในงานด้านวิศวกรรม แต่ด้วยลักษณะการแสดงผลแบบ กราฟฟิกของเครื่อง Macintosh ทำให้เหมาะสมกับการประยุกต์ใช้กับ LabVIEW และในปี 1990 ทาง NI ได้ประสบผลสำเร็จในการนำ LabVIEW version 2 ออกสู่ตลาด โดยได้ ปรับแก้และเขียนระบบควบคุมใหม่ทั้งหมด ตามคำแนะนำ ของผู้ใช้งาน โดยเฉพาะการเขียน Compiler ที่ทำให้เวลาการทำงานของโปรแกรมรวดเร็วขึ้น หลังจากนั้น บริษัทก็ได้พัฒนาโปรแกรมให้เหมาะสมกับเทคโนโลยียิ่งขึ้น ตามรูปแบบ ปฏิบัติการที่เปลี่ยนแปลงไปเช่น LABVIEW สำหรับ Windows NT, Windows 95 รวมถึงการสร้าง Version ใหม่ เพื่อจัดระบบและการเขียนโปรแกรมให้สะดวกมากขึ้น ตลอดจนสามารถเชื่อมต่อกับ อุปกรณ์ต่างๆ มากขึ้น พร้อมทั้งสร้าง

ฟังก์ชั่น ต่างๆ เพื่อให้เหมาะสมกับการใช้งานมากขึ้น

จุดประสงค์หลักของการทำงานของโปรแกรมนี้ก็คือการจัดการในด้านการวัดและเครื่องมือวัดอย่างมีประสิทธิภาพ และในตัวของโปรแกรมจะประกอบไปด้วยฟังก์ชั่น ที่ใช้ช่วยใน การวัดมากมายและแน่นอนที่สุด โปรแกรมนี้จะมีประโยชน์อย่างสูงเมื่อใช้ร่วมกับเครื่องมือวัดทางวิศวกรรมต่างๆ สิ่งที่ LabVIEW แตกต่างจากโปรแกรมอื่นอย่างเห็นได้ชัดที่สุดก็คือ LabVIEW นี้ เป็นโปรแกรมประเภท GUI (Graphic User Interface) นั่นคือไม่จำเป็นต้องเขียน code หรือคำสั่งใดๆ ทั้งสิ้น ข้อได้เปรียบสูงสุดของ LabVIEW คือการพยายามทำให้เครื่องคอมพิวเตอร์ของเราเมื่อรวมกับ LabVIEW และ อุปกรณ์เชื่อมต่อเพื่อการเก็บ ข้อมูล (Data Acquisition Card) แล้วสามารถเปลี่ยนเครื่องคอมพิวเตอร์ส่วนบุคคลของเราให้ กลายเป็นเครื่องมือวัดในหลายรูปแบบ ไม่ว่า จะเป็น Oscilloscope, Multi-meter, Function Generator, Strain meter Thermometer หรือเครื่องมือวัดอื่นๆ ตามที่เราต้องการทำให้สามารถใช้ คอมพิวเตอร์ในการทำการวัดและเครื่องมือวัดได้อย่างกว้างขวาง ซึ่งจุดนั้นเองที่เป็นที่มาของชื่อ เครื่องมือวัดเสมือนจริง (Virtual Instrument) และข้อได้เปรียบเหนือการใช้อุปกรณ์จริงเหล่านั้นคือ Virtual Instrument สามารถปรับเปลี่ยนให้เหมาะสมกับการใช้งานของผู้ใช้แต่ละกลุ่มได้โดยการ เปลี่ยน VI ให้เป็นไปตามต้องการเป็นเรื่องที่ไม่ยุ่งยากนัก



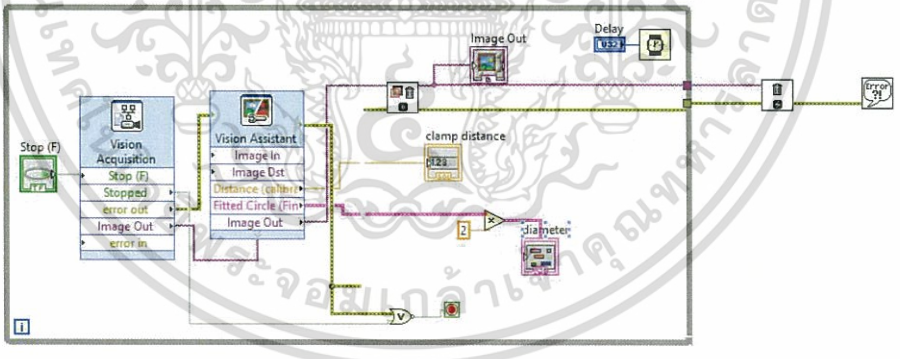
รูปที่ 2.2 แสดงหน้าจอการเขียนโปรแกรมและหน้าจอแสดงผล

2.1.2 DATA FLOW AND PROGRAMING

เนื่องจาก LABVIEW เป็นโปรแกรมที่ใช้รูปภาพ หรือสัญลักษณ์แทนการเขียนด้วย ตัวอักษรเหมือนโปรแกรมปกติทั่วไป ข้อแตกต่างอีกประการหนึ่งที่สำคัญของการเขียนโปรแกรมแบบ G กับการ เขียนด้วยตัวหนังสือก็คือ การเขียนด้วยภาษา G นี้เป็นการเขียนโดยใช้หลักการของ Data Flow ซึ่ง เมื่อเริ่มส่งข้อมูลเข้าสู่โปรแกรม เราจะต้องกำหนดทิศทางไหลของข้อมูลว่าจะไปที่ส่วนใด ผ่านการ ประเมินผลและคำนวณในส่วนใดบ้างและจะให้แสดงผลอย่างไรซึ่งลักษณะการเขียนภาษา G หรือ Data Flow นี้จะมีลักษณะ

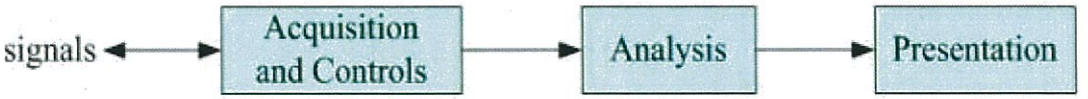
เหมือนกับการเขียน Block Diagram ซึ่งทำให้ผู้เขียนโปรแกรมสามารถให้ความสนใจกับการเคลื่อนที่และเปลี่ยนแปลงข้อมูลได้โดยไม่ต้องจดจำรูปแบบคำสั่งที่ยุ่งยาก และถ้าหากเราจำได้ถึงขั้นตอนการเขียนโปรแกรมว่าก่อนที่จะเขียนโปรแกรม จะต้องเขียน Flow Chart ให้เสร็จสิ้นก่อนหลังจากตรวจสอบ Flow Chart เรียบร้อยแล้วเราจึงนำไปเขียนโปรแกรม ซึ่งจะมีความสะดวกมากขึ้น ถ้าหากการเขียน Flow Chart ของ LabVIEW ก็คือการเขียนโปรแกรมนั่นเองซึ่งเป็นการลดขั้นตอนการทำงานลงไปได้เป็นอย่างมากแม้ว่าการเขียนโปรแกรมใน LabVIEW ไม่จำเป็นต้องมีความรู้ด้านการเขียนโปรแกรมใดๆ มาก่อนเลย แต่การมีความรู้ด้านการเขียนโปรแกรมหรือใช้ โปรแกรมสำเร็จรูปอื่นๆ จะสามารถนำมาใช้ประโยชน์ได้เป็นอย่างดี

LabVIEW จะมี Front Panel ซึ่งเปรียบเสมือนได้กับสิ่งที่ผู้ใช้จะเห็นและควบคุมการทำงาน ผู้ใช้สามารถสร้างรูปแบบขึ้นเองได้อย่างรวดเร็วเพราะ LabVIEW มีส่วนประกอบต่างๆ ที่ใช้สำหรับออกแบบหน้าจอมากมาย เช่น จอแสดงผลแบบออพิโลลโครบ, ปุ่มหมุน (Dial) และ สวิทช์ เป็นต้น โดย LabVIEW จะแสดงผลและควบคุมการทำงานผ่านทางคอมพิวเตอร์พื้นที่ส่วนเขียนโปรแกรมจะเรียกว่า Block Diagram เปรียบเสมือนกับ Hardware ภายในเครื่องมือวัด โดย LabVIEW จะเขียนโปรแกรมโดยอาศัยรูปภาพ



รูปที่ 2.3 แสดง Block Diagramของ LabVIEW

LabVIEW อาศัยหลักการทำงานของเครื่องมือวัดหรือการวัดคุมทำให้ผู้ใช้สามารถออกแบบตามที่ ผู้ใช้ต้องการหลักการดังกล่าวแบ่งออกเป็น 3 ส่วนใหญ่ๆ คือ



รูปที่ 2.4 Block Diagram เครื่องมือวัดที่สร้างจาก LabVIEW

2.1.2.1 Acquisition ซึ่งเป็นส่วนที่รับข้อมูล (Input) จากสิ่งแวดล้อมภายนอกเข้าสู่ระบบในที่นี้คือคอมพิวเตอร์โดยข้อมูลที่เข้าสู่ระบบนี้อาจมาจากการ์ด DAQ (สำหรับสัญญาณทางไฟฟ้า)

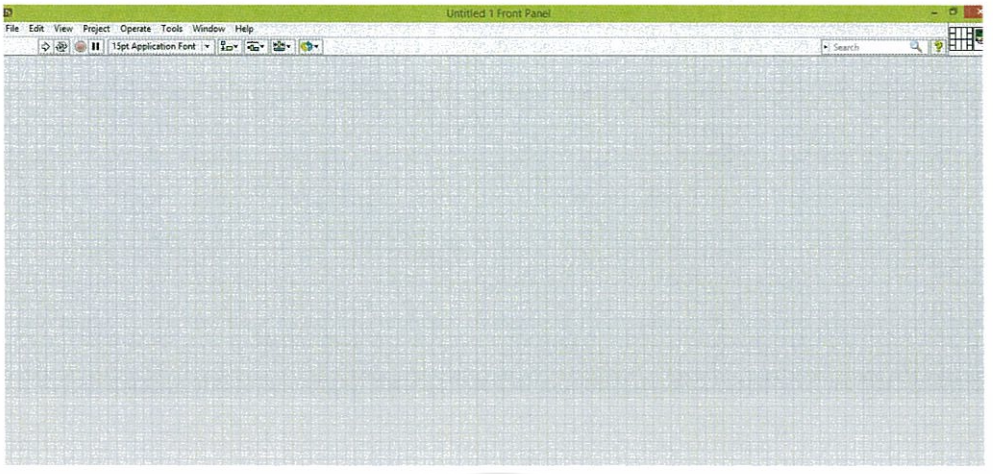
2.1.2.2 Analysis หลังจากที่ได้รับข้อมูลแล้วอาจจะผ่านฟังก์ชั่น ในการวิเคราะห์ข้อมูล ซึ่งจะแสดงผลในรูปที่สื่อความหมายในสิ่งที่ผู้ใช้งานสามารถนำไปแสดงแทนสื่อที่วัดได้ และใช้งานได้

2.1.2.3 Presentation คือ การแสดงผลในรูปแบบที่เป็นประโยชน์ต่อผู้ใช้งานโดยอาจแสดงบนหน้าจอคอมพิวเตอร์ เช่น DMM (Digital Multimeter) แสดงผลเฉพาะที่วัดได้โดยไม่จำเป็นต้องรู้ความสำคัญกับเวลา หรือ Spectrum Analysis จะแสดงสัญญาณในรูปความถี่หรือการพิมพ์ออกมาเป็นรายงานหรือเก็บข้อมูลในฮาร์ดดิสก์

2.1.3 ส่วนประกอบต่างๆ ใน LabVIEW

โปรแกรมที่เขียนขึ้นมาโดย LabVIEW เราจะเรียกว่า Virtual Instrument (VI) เพราะลักษณะที่ปรากฏทางจอภาพเมื่อผู้ใช้ใช้งานจะเหมือนกับเครื่องมือหรืออุปกรณ์ทางวิศวกรรม ในขณะเดียวกันหลังฉากของอุปกรณ์เสมือนจริงเหล่านั้นจะเป็นการทำงานของ ฟังก์ชั่น , Subroutines และโปรแกรมหลักเหมือนกับภาษาทั่ว ไปสำหรับ VI หนึ่งๆ จะประกอบด้วยส่วนประกอบ 3ส่วน คือ

2.1.3.1 Front Panel หรือหน้าปัทม์ จะเป็นส่วนที่ใช้สื่อความกันระหว่างผู้ใช้กับโปรแกรม (หรือที่นิยมเรียก User Interface) โดยทั่วไปจะมีลักษณะเหมือนกับหน้าปัทม์ของเครื่องมือหรืออุปกรณ์ที่ใช้งานด้านการวัดทั่วๆ ไป โดยทั่วไปจะประกอบด้วยสวิตซ์ปิดเปิด, ปุ่มปิด, ปุ่มกด จอแสดงผล หรือแม้แต่ค่าที่ผู้ใช้สามารถกำหนดสำหรับผู้ที่คุ้นเคยกับการเขียนโปรแกรมประเภท Visual ทั้งหลายคงจะเข้าใจกันดีว่า Front Panel นี้จะเปรียบเสมือนเป็น GUI ของโปรแกรมหรือ VI นั่นเอง



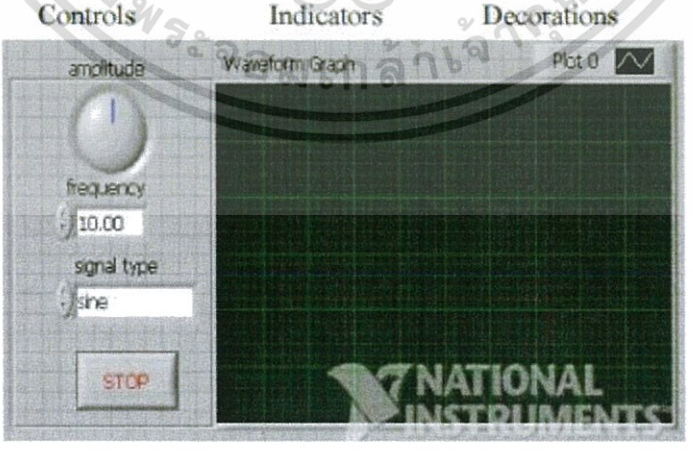
รูปที่ 2.5 Front Panelของ LabVIEW

Object ที่อยู่บน Front Panel จะมีอยู่สามประเภท คือ

2.1.3.1.1 Control คือประเภทที่รับค่าจากผู้ใช้(Input) ซึ่งผู้ใช้สามารถพิมพ์ค่าลงไป หรือใช้เมาส์คลิกเพื่อเปลี่ยนแปลงค่าได้เช่น ปุ่มหมุน ปุ่มเลื่อน สวิตช์เป็นต้น

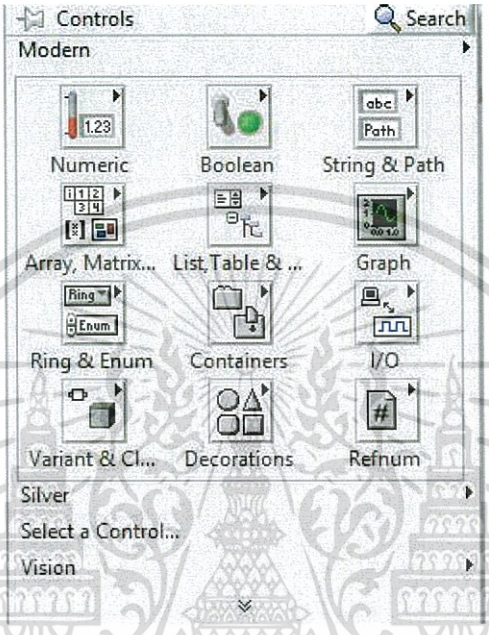
2.1.3.1.2 Indicators คือประเภทที่ใช้แสดงค่าต่างๆเท่านั้น (Output) ผู้ใช้ไม่สามารถแก้ไขได้เช่น กราฟ มิเตอร์ LED

2.1.3.1.3 Decorations เป็น Object ที่ไม่เกี่ยวข้องกับโปรแกรมและcode บน Block Diagram เลย แต่มีไว้เพื่อความสวยงามเป็นระเบียบของ Front panel เท่านั้นนั่นเอง ลักษณะของ Front Panel แสดงดังรูปต่อไปนี้



รูปที่ 2.6 Object ที่อยู่บน Front Panel ของ LabVIEW

เครื่องมือที่ใช้ออกแบบ Front Panel เครื่องมือที่ใช้ในการออกแบบ Front Panel จะประกอบไปด้วย Control Palette และ Tools Palette ซึ่ง LabVIEW มี Controls Palette ที่ใช้ในการออกแบบ Front Panel แสดงดังรูป 1.7 ซึ่งเป็น ส่วนที่ติดต่อกับผู้ใช้งาน (User Interface) โดยจะจัดเป็นกลุ่มต่าง ๆ เช่น กลุ่มของตัวเลข (Numeric) ซึ่งภายในกลุ่มจะมีControl และ Indicator ต่างๆ ที่เกี่ยวกับตัวเลข



รูปที่ 2.7 Controls Palette ที่ใช้ในการออกแบบ Front Panel

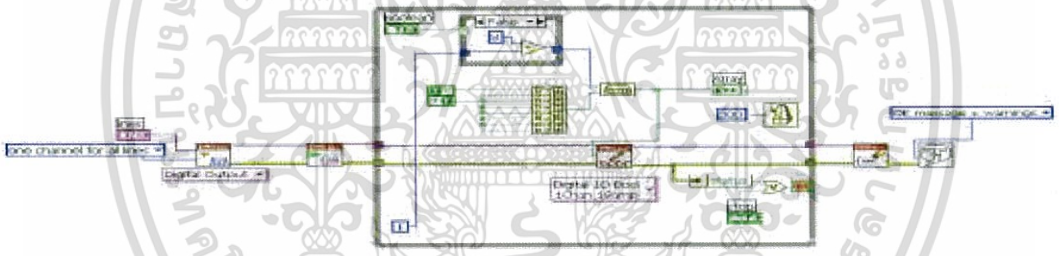
Tools Palette คือ เครื่องมือที่ใช้ในการพัฒนาโปรแกรม ซึ่งจะใช้ทั้งการออกแบบ Front Panel และ Block Diagram ในส่วนนี้จะกล่าวถึง Tools Palette สำหรับออกแบบ Front Panel

รูปที่ 2.8 Tools Palette ที่ใช้ในการออกแบบ Front Panel

2.1.4 Block Diagram

เพื่อให้เกิดความเข้าใจง่ายขึ้น เราอาจมอง Block Diagram นี้เป็นเสมือนกับ Source Code หรือโปรแกรมของ LabVIEW ซึ่งปรากฏว่าอยู่ในรูปของภาษา G ซึ่ง Block Diagram นี้ถือว่าเป็น Executable Program คือสามารถที่จะทำงานได้ทันทีและข้อดีอีกประการหนึ่งก็คือ LabVIEW จะมีการตรวจสอบความผิดพลาดของโปรแกรมตลอดเวลาทำให้โปรแกรมจะทำงานได้ก็ต่อเมื่อไม่มี ข้อผิดพลาดในโปรแกรมเท่านั้นโดยผู้ใช้สามารถที่จะดูรายละเอียดของความผิดพลาดแสดงให้เห็น ได้ตลอดเวลาทำให้การเขียนโปรแกรมนั้นง่ายขึ้นมาก

ส่วนประกอบภายใน Block Diagram นี้จะประกอบด้วยฟังก์ชั่นค่าคงที่โปรแกรมควบคุมการทำงานหรือโครงสร้าง จากนั้นในแต่ละส่วนเหล่านี้ซึ่งจะปรากฏในรูปของ Block เราจะได้รับ การต่อสาย (Wire) สำหรับ Block ที่เหมาะสมเข้าด้วยกัน เพื่อกำหนดลักษณะการไหลของข้อมูล ระหว่าง Block เหล่านั้นทำให้ข้อมูลได้รับการประมวลผลตามที่ต้องการและแสดงผลออกมาให้แก่ ผู้ใช้ต่อไป



รูปที่2.9 ตัวอย่าง Block Diagram

2.1.4.1 Block Diagram Node

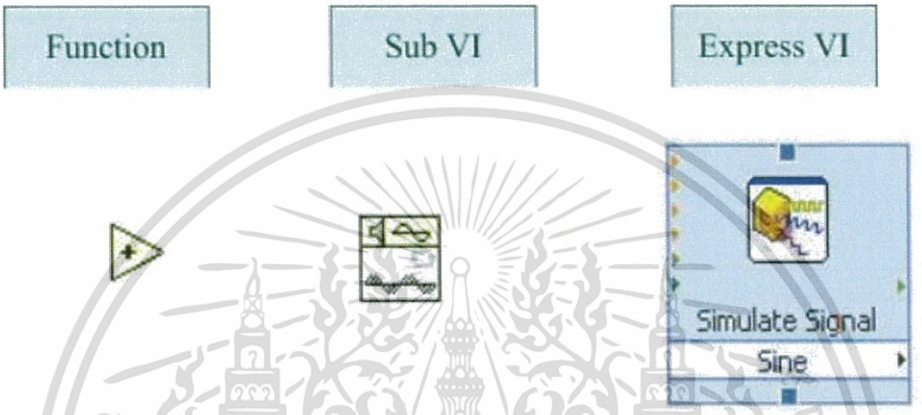
Node คือรูป Icon ที่อยู่บน Block Diagram ซึ่งมีInput และ/หรือ Output และจะทำงานตาม หน้าที่เมื่อมีการรันโปรแกรม โดนแบ่งเป็นสามชนิดหลัก

2.1.4.1.1 Function คือ Node ที่มีหน้าที่พื้นฐานของคอมพิวเตอร์ซึ่งเราไม่สามารถที่จะเจาะเข้าไป ดูรายละเอียดภายในได้อีกเช่น การบวกการคูณ

2.1.4.1.2 SubVIs หรือในภาษาทางซอฟต์แวร์อาจจะเรียกว่า Subroutine หรือ Subprogram คือ โปรแกรมย่อยที่ถูกเขียนขึ้นมาเพื่อถูกนา มาเรียกใช้ในอีกโปรแกรมหนึ่ง เราสามารถเปิดเข้าไปดู front panel และ block diagram ได้เมื่อ double click ที่ Icon ของมัน

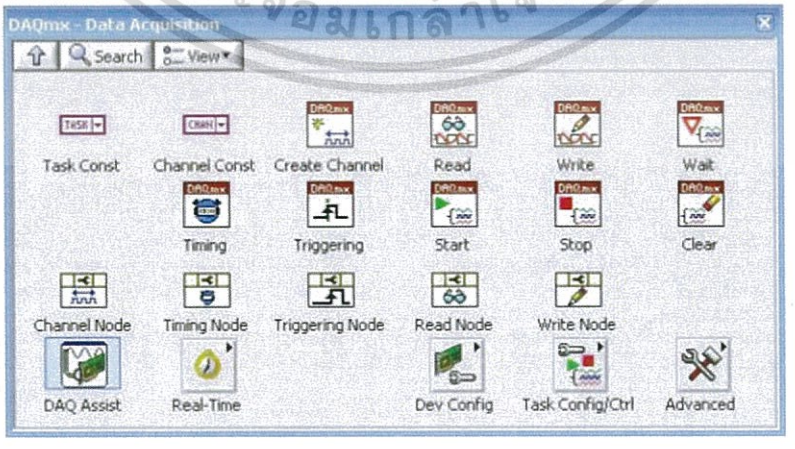
2.1.4.1.3 Express VIs เป็น subVIs ประเภทพิเศษคือเมื่อเราเลือก Express VI มา

วางบน Block Diagram มันจะปรากฏหน้าต่าง Configuration ขึ้นมาเพื่อนให้เราเข้าไปป้อนค่า Parameters ต่าง ตามต้องการและเมื่อเราป้อนค่าเสร็จ มันก็จะสร้างโค้ดไว้ภายในอัตโนมัติตามที่เราได้ตั้งค่าไว้ ซึ่งความสามารถของ Express VI นี้ทำให้เราแทบไม่จำ เป็นต้องต่อสาย Input เลยเพราะ Parameter ทั้งหมดได้ถูกสร้างขึ้นมาแล้วถูกเก็บไว้ภายในเรียบร้อยแล้ว จึงทา ให้การเขียน LabVIEW ง่ายและ เร็วขึ้นมาก สังเกตง่ายๆ Express VIจะมีIcon ขนานใหญ่ที่มีพื้นหลงเป็นสีฟ้า



รูปที่ 2.10 ตัวอย่าง Block Diagram Node

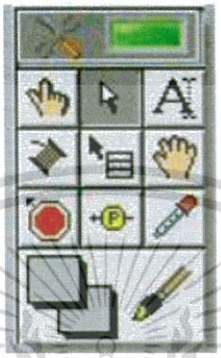
2.1.4.2 เครื่องมือที่ใช้ในการเขียนโปรแกรมบน Block Diagram LabVIEW ใช้ Functions Palette ซึ่งจะมี Function และ SubVI ต่าง ๆ ที่มีอยู่แล้วให้ผู้ใช้ เลือกใช้ โดย Function และ SubVI จัดเป็นกลุ่ม ๆ เช่น Numeric Function จะมี Function ต่าง ๆ เกี่ยวกับตัวเลขเช่น บวกลบ คูณ หารแสดงดังรูป 2.10



รูปที่2.11 เครื่องมือสำหรับ Dam – Data Acquisition

Tools Palette สำหรับ Block diagram เปิด Tools Palette โดยการ Click ที่ Window >> Show Tools Palette หรือกด Shift Right Click แล้ว Click เลือก Tool ที่ต้องการใช้

(ถ้า Automatic Tools Selection เปิดอยู่แล้ว(LED) เป็นสีเขียว ให้Click เพื่อ ปิด Automatic Tool Selection ซึ่งมีเฉพาะใน Version 6.1 ขึ้นไป) Tool ที่ใช้มีดังต่อไปนี้



รูปที่ 2.12 เครื่องมือ Tools Palette

2.1.4.2.1 Operation Tool ใช้ในการเปลี่ยนแปลงค่าหรือเลือกค่าคงที่ใน Block Diagram

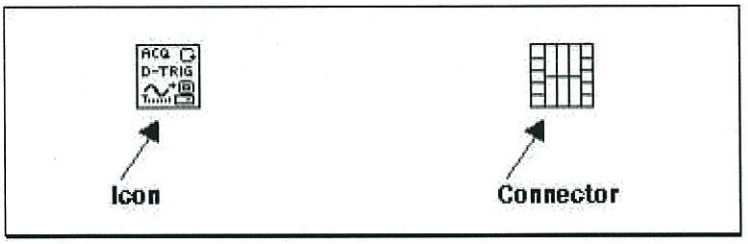
2.1.4.2.2 Position/Size/Select ใช้ในการเลือก/เคลื่อนย้าย/จัดขนาดของสิ่งที่สร้างขั้นบน Block Diagram

2.1.4.2 3 Edit Text Tool ใช้ในการแก้ไขข้อความที่เป็นตัวอักษร หรือเพิ่มข้อความลงบน Front Panel

2.1.4.2 4 Wiring Tool ใช้ในการโยงสาย (Wiring) ระหว่า ง Terminal หรือ Node ซึ่งสายที่โยงนี้ จะเป็นเส้นทางเดินของข้อมูล

2.1.4.3 Icon และ Connector เปรียบเสมือนโปรแกรมย่อ ย Subroutine ในโปรแกรมปกติทั่ว ๆ ไป โดย Icon จะหมายถึง Block Diagram ตัวหนึ่งที่มีการส่งข้อมูลเข้าและออกผ่านทาง Connector ซึ่ง ใน LabVIEW เราจะเรียก Subroutine นี้ว่า SubVIข้อดีของการเขียนโปรแกรมด้วยภาษา G นี้ก็คือเรา สามารถสร้าง VI ทีละส่วนขึ้นมาให้ทำงานด้วยตัวเองได้อย่างอิสระ จากนั้น ในภายหลักหากเรา ต้องการเราก็สามารถเขียน โปรแกรมอื่นขึ้นมาเพื่อเรียกใช้งาน VI ที่เราเคยสร้างขึ้นก่อนหน้านี้ทีละ ตัว ซึ่งทำให้ VI ที่เราเขียนขึ้นก่อนกลายเป็น SubVI ไป การเขียนในลักษณะนี้เราเรียกว่า เขียนเป็น Module

สำหรับลักษณะทั่ว ไปของ Icon และ Connector จะแสดงในรูปต่อไปนี้เราจะเห็นว่า เมื่อเรา แสดงในรูปของ Connector เราจะพบว่า มีช่องต่อข้อ มูลหรือที่เรียกว่า Terminal ปรากฏให้เห็น



รูปที่ 2.13 แสดงลักษณะทั่ว ไปของ Icon และ Connector

คำศัพท์ต่างๆที่ใช้กัน ใน LabVIEW นี้ออกจะแตกต่างจากที่เราใช้กัน ในภาษาการเขียนโปรแกรม ตัวหนังสือทั่ว ๆไปในหลายๆด้าน ดังนั้น เพื่อให้ผู้ ทำเริ่มใช้ LabVIEW เข้าใจถึงศัพท์ต่างๆ ที่ใช้ใน โปรแกรม เราจึงขอเปรียบเทียบศัพท์ใช้ใน LabVIEW กับโปรแกรมพื้นฐานทั่ว ๆ ไปตามตารางที่ได้ แสดงต่อไปนี้

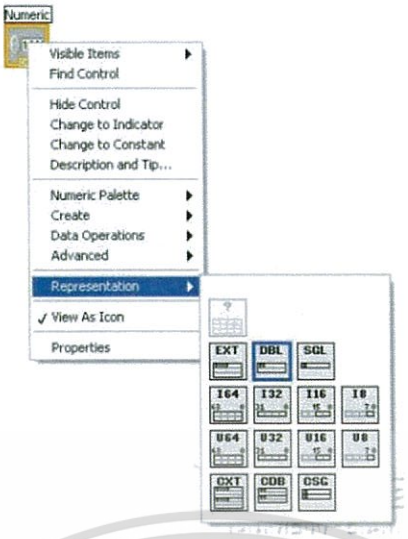
LabVIEW	โปรแกรมพื้นฐาน	หน้าที่
VI	Program	ตัวโปรแกรมหลัก
Function	function	ฟังก์ชันสำเร็จรูปที่สร้างขึ้นมากับโปรแกรมนั้นเช่น sin, log เป็นต้น
SubVI	Subroutine	โปรแกรมย่อยที่ถูกเรียกใช้โดยโปรแกรมหลัก
Front Panel	user interface	ส่วนที่ติดต่อกับผู้ใช้
Block Diagram	Program code	การเขียนตามขั้นตอนของที่แต่ละ โปรแกรมกำหนดขึ้น

ตารางที่ 2.1 เปรียบเทียบคำศัพท์ที่ใช้ในการเขียนโปรแกรม

2.1.5 ประเภทของข้อมูล

ในการเขียนโปรแกรมทั่วๆไปจะต้องมีการประกาศตัวแปรก่อนที่จะใช้ตัวแปรนั้น แต่ สำหรับโปรแกรม LabVIEW มันจะจัดการให้เองหมดโดยผู้ใช้ไม่จำ เป็นต้องทำ เอง เพียงแค่เลือก ประเภทของข้อมูลที่มาวางบนโค้ดให้ถูกต้องเท่านั้น ประเภทของของข้อมูลภายใน LabVIEW ก็มี หลายอย่างที่เหมือนกับโปรแกรมในภาษาอื่นๆ และยังมีอีกบางประเภทที่ใช้ใน LabVIEW เท่านั้น โปรแกรม LabVIEW แบ่งข้อ มูลเป็น 6 ชนิดดังนี้คือ

2.1.5.1 Numeric คือข้อมูลประเภทตัวเลข มีทั้งจำนวนเต็ม ซึ่งใน Block Diagram จะเห็นเป็น สีน้ำ เงิน และจำนวนทศนิยมจะเห็นเป็นสีส้ม และสามารถเปลี่ยนไปมาได้โดยการคลิกขวาที่ตัวเลขนั้น แล้วเลือกrepresentation และเลือกประเภทตัวเลขได้เลย แสดงดังรูป 2.14



รูปที่ 2.14 แสดงข้อมูลประเภท Numeric

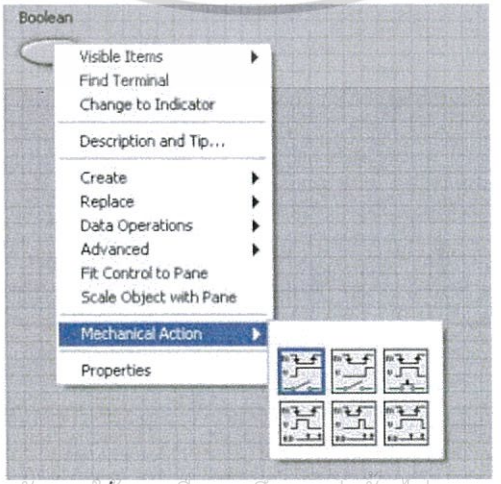
2.1.5.2 Boolean คือข้อ มูลประเภทที่มีสองค่า คือ True และ false บน Block Diagram จะ แสดงข้อมูลเป็นสีเขียว และสำหรับ Front Panel ตัว Boolean จะมีลักษณะเป็น ตัว Control หรือ สวิตช์ ถ้าเป็น Outputก็จะเป็น LED หรือหลอดไฟประเภทต่างๆ แสดงดังรูป 2.15

2.1.5.3 String คือข้อมูลประเภทที่เป็นตัวอักษร Icon จะแสดงเป็นสีชมพู สำหรับการ แสดงผลจะมีอยู่4แบบ คือ Normal Display คือการแสดงปกติ

Code Display คือการแสดงแบบ โค้ด มีประโยชน์สา หรับแสดงตัวอักษรที่ตาเปล่า มองไม่เห็น การเว้นวรรคแทบ หรือการขึ้นบรรทัดใหม่

Password Display คือการแทนตัวอักษรด้วย *

Hex Displayแสดงผลเป็นรหัสเลขฐานสิบหก



รูปที่ 2.15 แสดงข้อมูลประเภท Boolean

2.1.5.4 Enum คือข้อมูลประเภทแสดงให้ผู้ใช้เห็นเป็นตัวหนังสือ แต่ค่าจริงของมันคือตัวเลข ดังนั้น บน Block Diagram เราจึงมองเห็นข้อมูลประเภทนี้เป็นสีน้ำ เงิน ซึ่งเหมือนกับจำนวนเต็ม

2.1.5.5 Dynamic เป็นข้อมูลที่อยู่ในรูปของ Waveform บน Block Diagram ถูกแสดงด้วยสีน้ำ เงินเข้มซึ่งภายในจะประกอบด้วย Array ของเวฟฟอร์ม Time Stamp ชื่อของสัญญาณ ข้อมูล ประเภท Dynamic นี้ส่วนใหญ่ใช้ใน Express VIจา พวกการอ่าน กำเนิดและวิเคราะห์สัญญาณ

2.1.5.6 Time Stamp เป็นข้อมูลที่ประกอบด้วยวันที่ และเวลาที่มีความละเอียดถึงมิลลิวินาที Time Stamp บน Block diagram จะมีหน้าตาที่เป็นสีน้ำ ตาลเส้นหน้า สามารถนา มาแปลงให้เป็น วันที่ เวลา แบบ String ได้

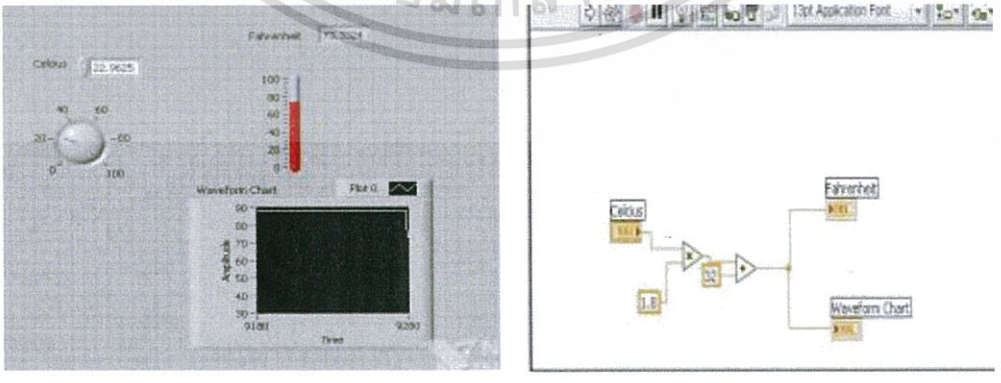
2.1.6 หลักการทำงานของ Dataflow Programming

ก่อนจะเริ่มพัฒนาโปรแกรมเรามาดูหลักการทา งานของโปรแกรมในรูปแบบ Data-flow และรูปแบบข้อมูล (Data Type) ซึ่งเป็นสิ่งสำคัญ ในการพัฒนาโปรแกรม โปรแกรมที่เขียนขึ้นด้วย LabVIEW จะทำงานโดยอาศัยหลักการของ Dataflow ซึ่งมีหลักการ ดังต่อไปนี้คือ

2.1.6.1 ฟังก์ชัน หรือ SubVI จะทำงานเมื่อมีข้อมูล (Input)

2.1.6.2 เมื่อฟังก์ชัน หรือ SubVI ทำงานเสร็จจะให้ข้อมูล (Output) ไปยังฟังก์ชัน หรือ SubVI อื่น ๆ ที่ต้องการข้อมูล

2.1.6.3 ข้อมูลจะถูกส่งผ่านโดยสาย(Wire) ในรูปที่ 2.16 เป็นตัวอย่างการเขียนโปรแกรม LabVIEW โดยให้หน้าจอแสดงผลค่า อุณหภูมิเป็นองศาฟาเรนไฮท์ โดยแสดงผลเป็นแบบตัวเลขและกราฟ โดยมีค่าอินพุตเป็นองศา เซลเซียส



(a) (B)

รูปที่ 2.16 (a) แสดงโปรแกรมที่เขียนขึ้น (B) จอแสดงผลที่ต้องการ

จากรูป 2.16 ขั้น ตอนการทำงานของโปรแกรมจะเป็นดังนี้

2.1.6.4 ฟังก์ชันคูณ (Multiply) จะทา งานก่อนฟังก์ชัน บวก เพราะฟังก์ชันคูณมีข้อมูล (Input) พร้อม แต่ฟังก์ชันบวกต้องรอ Outputจากฟังก์ชันคูณ (หลักการ Dataflow ข้อ 1)

2.1.6.5 หลังจากฟังก์ชันคูณทา งานเสร็จจะส่งผ่านข้อมูลไปยังฟังก์ชัน บวก (หลักการ Dataflow ข้อที่ 2 และ 3)

2.1.6.6 ฟังก์ชัน บวกทา งาน (หลักการข้อ 1) เพราะมีข้อมูลพร้อม

2.1.6.7 หลังจากฟังก์ชัน บวกทา งานจะส่งผลลัพธ์ไปให้Terminal ทั้ง 2 คือ Fahrenheit และ Waveform Chart พร้อมกัน (หลักการข้อ 2และ3)

2.2Webcam

2.2.1 เว็บแคม คืออะไร ?

เว็บแคม (Webcam) หรือ ชื่อเรียกเต็มๆว่า Web Camera แต่ในบางครั้งก็มีคนเรียกว่า Video Camera หรือ Video Conference ก็แล้วแค่ความเข้าใจแต่ละคน เว็บแคมเป็นอุปกรณ์อินพุตที่ สามารถจับภาพเคลื่อนไหวของเราไปปรากฏในหน้าจอมอนิเตอร์ และสามารถส่งภาพเคลื่อนไหวนี้ผ่านระบบเครือข่ายเพื่อให้คนอีกฟากหนึ่งสามารถเห็นตัวเราเคลื่อนไหว ได้เหมือนอยู่ต่อหน้า ถือว่าเป็นอุปกรณ์ที่มีประโยชน์อีกตัวหนึ่ง และเริ่มมีความจำเป็นมากขึ้นเรื่อยๆ

2.2.2 ประเภทของเว็บแคม

อุปกรณ์อย่างกล้องเว็บแคมไม่ใช่ว่าจะเหมือนกันหมดทุกตัว แต่ละรุ่น แต่ละยี่ห้อจะมีลักษณะและคุณสมบัติที่แตกต่างกันไปตามแต่ผู้ผลิตจะคิดค้นและออกแบบมาให้เหมาะสมกับการใช้งานอย่างไร ซึ่งสามารถแบบประเภทของเว็บแคมได้ดังนี้

2.2.2.1 แบ่งตามรูปทรงของกล้อง

โดยปกติกล้องเว็บแคมส่วนใหญ่จะเป็นทรงกลม เนื่องจากเป็นรูปทรงต้นแบบที่ทำกันมานาน และก็ทำให้รู้ได้ทันทีว่านี้คืออุปกรณ์ เว็บแคม แต่ไม่จำเป็นที่กล้องเว็บแคมต้องเป็นทรงกลมเสมอไป เพราะบางครั้ง กล้องเว็บแคม ก็จำเป็นต้องมีรูปทรงอื่นๆ เพื่อให้เข้ากับการใช้งานในบางลักษณะ ดังนั้น การเลือกรูปทรงให้เหมาะสมนั้น ก็จะขึ้นอยู่กับลักษณะการใช้งานของเรามากกว่า

2.2.2.2 แบ่งตามประเภทของขาตั้งกล้อง โดยส่วนใหญ่ลักษณะของฐานตั้งกล้องจะเป็นแบบตั้งพื้นเสียส่วนใหญ่ โดยแบบแรก คือแบบมีขาสำหรับวางบนพื้น อาจจะมีขา 3 ขา หรือ 4 ขา ก็แล้วแต่การออกแบบ แต่ฐานแบบ 3 ขา จะมี ปัญหาตรงที่ วางแล้วยังไม่มั่นคงดีนัก และไม่สามารถหมุนตัวกล้องได้สะดวกนัด ดังนั้นถ้าต้องการเว็บแคมที่มีฐานมั่นคงและสามารถหมุนได้ง่ายๆ ก็ต้องเลือกแบบฐานทรงกลมขนาดใหญ่ ซึ่งแบบนี้จะมีข้อดีตรงที่ วางได้มั่นคงและยังสามารถหมุนแกน ของตัวกล้องได้ไม่จำเป็นต้องยกตัวกล้องหมุนไปมาให้เสียเวลา

2.2.2.3 แบ่งตามชนิดของเซ็นเซอร์ สำหรับเซ็นเซอร์ที่กล้องเว็บแคมใช้นั้นจะมีหลักๆอยู่ 2 ชนิด คือ CCD และ CMOS แต่ที่นิยมใช้กันมากที่สุดในตอนนี้ก็คือ CMOS เนื่องจากเหตุผลหลายๆประการและตัวเซ็นเซอร์ แบบ CMOS เองก็สามารถแบบออกได้ถึง 2 ชนิดด้วยกันคือ CLF Color CMOS Censor ที่มีความละเอียดของพิกเซลแค่ 110,000 พิกเซล (367 x 291) เท่านั้น ในขณะที่ VGA Color CMOS Censor ให้ความละเอียดที่สูงกว่าที่ 350,000 พิกเซล (655 x 493) ดังนั้น เวลาเลือกซื้อกล้องเว็บแคมก็ดูได้ทั้งความละเอียดที่ระบุไว้ หรือชนิดของ CMOS สำหรับเซ็นเซอร์แบบ CCD จะเป็นเซ็นเซอร์ที่นิยมใช้ในกล้องดิจิตอล เพราะให้ความละเอียดที่สูงกว่าและก็มี noise ไม่มากเหมือนกับเซ็นเซอร์แบบ CMOS

2.2.2.4 แบ่งตามรูปแบบการเชื่อมต่อ สำหรับการเชื่อมต่อของกล้องเว็บแคมในปัจจุบันส่วนใหญ่ จะเป็นอินเทอร์เฟซแบบ USB แทบทั้งสิ้นโดย USB ที่ใช้ก็จะเป็นเวอร์ชัน 1.1 เสียส่วนมาก แต่ก็จะมีเวอร์ชัน 2.0 ในบางรุ่นกล้องเว็บแคมแบบไร้สายจะใช้การเชื่อมต่อในแบบ WiFi หรือ Wireless lan นั่นเองทำให้สามารถเคลื่อนย้ายไปได้ทุกที่โดยไม่ต้องคำนึงถึงสายให้วุ่นวาย แต่เว็บแคมที่เป็น Wireless ตอนนี้ก็ยังมีราคาค่อนข้างแพงอยู่

2.2.3 การเลือกซื้อกล้องเว็บแคม

ขั้นตอนแรกเราต้องรู้ว่าจะนำกล้องเว็บแคม มาใช้งานกับเครื่องคอมพิวเตอร์ประเภทใด ถ้าเป็นโน้ตบุ๊กก็ต้องเป็นกล้องเว็บแคม ขนาดเล็กกะทัดรัด และสามารถติดตั้งบนจอแอลซีดีของโน้ตบุ๊กได้ แต่ถ้าใช้กับเครื่องคอมพิวเตอร์เดสก์ทอปก็ แนะนำรุ่นที่มีขาตั้งที่มั่นคงสามารถวางบนจอมอนิเตอร์

เมื่อเลือกรูปแบบของกล้องได้แล้ว ก็มาเลือกตามคุณสมบัติภายในของกล้องเว็บแคมโดยเลือกจากชนิดของเซ็นเซอร์ที่ใช้กับภาพ โดยจะมีให้เลือกเป็น CMOS ในแบบ CIF และ VGA ซึ่งแนะนำว่าเป็นแบบ VGA จะให้ความละเอียดที่สูงกว่า หรือถ้าต้องการความละเอียดที่มากกว่านี้ ก็เลือกเซ็นเซอร์แบบ CCD จะดีกว่าแต่ทั้งนี้ราคาก็จะเพิ่มสูงขึ้น ตามชนิดของเซ็นเซอร์ และ ความละเอียดของตัวกล้องเว็บแคม

2.3.NI myRIO

2.3.1 อะไรคือ NI Myrio

การประมวลผลแบบ real-time และ Xilinx FPGA ซึ่งการประมวลผลนี้สามารถปรับแต่ง Input / Output ให้นักเรียนสามารถใช้งานกับอุปกรณ์ onboard ซึ่งซอฟแวร์ก็มีประสิทธิภาพพอใช้งานได้ และมีบทเรียนและแบบฝึกหัดตามในหนังสือ, NI myRIO เป็นเครื่องมือที่เหมาะสมที่นักเรียนสามารถใช้ทำโปรเจคในวิชาวิศวกรรมในหนึ่งภาคการศึกษา

2.3.1.1 สอนให้ใช้อุปกรณ์หลายๆอย่าง

อุปกรณ์หนึ่งสำหรับการควบคุมการเรียนการสอน, หุ่นยนต์, เมคคาทรอนิคส์และแนวความคิดที่ฝังตัว

2.3.1.2 สามารถจินตนาการและสร้างมันได้

โปรแกรม onboard อุปกรณ์และเชื่อมต่อเซ็นเซอร์ของ 3 สิ่งเข้าด้วยกัน

2.3.1.3 การออกแบบให้ทันเวลา

นักเรียนได้ออกแบบระบบวิศวกรรมจริงเร็วขึ้นกว่าเดิม

2.3.1.4 สอนหลายแนวความคิดด้วยเครื่องมือเดียว

2.3.1.5 เรียนรู้และการออกแบบในเครื่องมือหนึ่ง

นักเรียนสามารถเรียนรู้บนเครื่องมือเดียวกันว่าเราจะมาใช้ในการสร้างโครงการ การใช้เทคโนโลยีที่เป็นมาตรฐานอุตสาหกรรมในโลกแห่งความเป็นจริง นักเรียนสามารถสำรวจความหลากหลายของแนวคิดวิศวกรรมโดยสังเกตจากผลงานโปรเจคในบนโลกนี้

2.3.1.6 บูรณาการกับการเรียนรู้ที่มีอยู่

เสริมหลักสูตรของคุณด้วยบทเรียนที่สามารถดาวน์โหลดและตัวอย่างโปรเจค โดยผสมผสานบนแนวทางการเรียนการสอน, คุณสามารถสร้างศักยภาพให้ตัวเองได้

2.3.1.7 สามารถจินตนาการและสร้างมันได้

2.3.1.8 รูปแบบฟังก์ชั่นหลายๆอย่าง

NI myRIO ใช้เทคโนโลยี Zynq ล่าสุดจาก Xilinx FPGA ที่มีบูรณาการกับหน่วยประมวลผลที่ทำงานแบบ real-time OS ซึ่งเป็นเทคโนโลยีที่มีประสิทธิภาพควบคู่ไปกับเซนเซอร์เครื่องวัดความเร่ง ,ไฟ LED, โปรแกรมเสียง I / O แบบอะนาล็อกและดิจิตอล I / O และพอร์ต USB ช่วยให้โครงการสำเร็จไปหลายพันกว่าโครงการในชีวิต

2.3.1.9 ราคาไม่แพงและพกพาสะดวก

NI myRIO กรงนักเรียนพร้อมและการกำหนดราคาส่วนลดทางด้านวิชาการ หมายถึงนักเรียนสามารถทำงานบนอุปกรณ์ฮาร์ดแวร์ของตัวเองภายในหรือนอกห้องเรียน ด้วยความสามารถในตัวอินเตอร์เน็ตไร้สาย, นักเรียนสามารถถ่ายโอนหรือส่งข้อมูลแบบไร้สายและการปรับใช้รหัสด้วยตัวเอง

2.3.1.10 สามารถใช้โปรแกรมได้ทุกระดับ

นักเรียนสามารถดำเนินการโครงการในหนึ่งภาคการศึกษาไม่ว่าจะมีประสบการณ์เขียนโปรแกรมมากหรือน้อยก็ตาม NI myRIO ใช้เขียนโปรแกรมแบบ FPGA กับซอฟต์แวร์ LabVIEW และมีตัวเลือกในการเขียนโปรแกรมการประมวลผลใน LabVIEW หรือ C / C ++ นักเรียนสามารถเขียนโปรแกรมได้อย่างมีประสบการณ์จนจบการศึกษาในปัจจุบันและต่อให้มีความซับซ้อนมากขึ้นนักเรียนก็สามารถทำให้เป็นเรื่องง่ายได้เลย

2.3.1.11 สามารถทำงานร่วมกันและขยายการเรียนรู้ได้

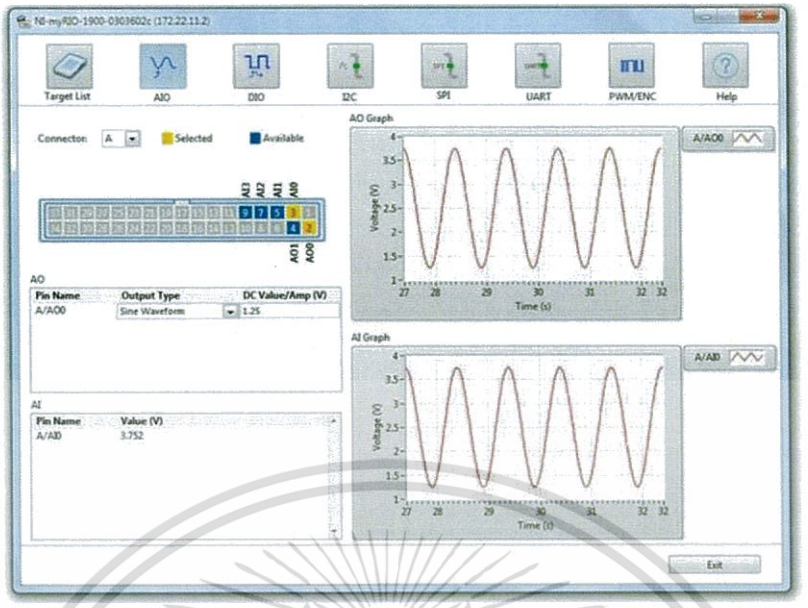
นักเรียนได้รับประสบการณ์ในด้านหนึ่ง สามารถขยายรูปแบบการเรียนรู้ได้ด้วยความสามารถในระดับการเรียนรู้ของพวกเขาและสร้างโครงการที่ซับซ้อนมากขึ้นในเวลาที่สั้นกว่า การใช้ทั้งเซ็นเซอร์ใหม่และที่มีอยู่และกระตุ้น, NI myRIO ให้ความรู้กับนักศึกษาสามารถที่จะนำมาใช้อุปกรณ์จากแต่ละภาคการศึกษาได้

2.3.2 มีอะไรใหม่ใน NI myrio?

2.3.2.1 คุณสมบัติของซอฟต์แวร์

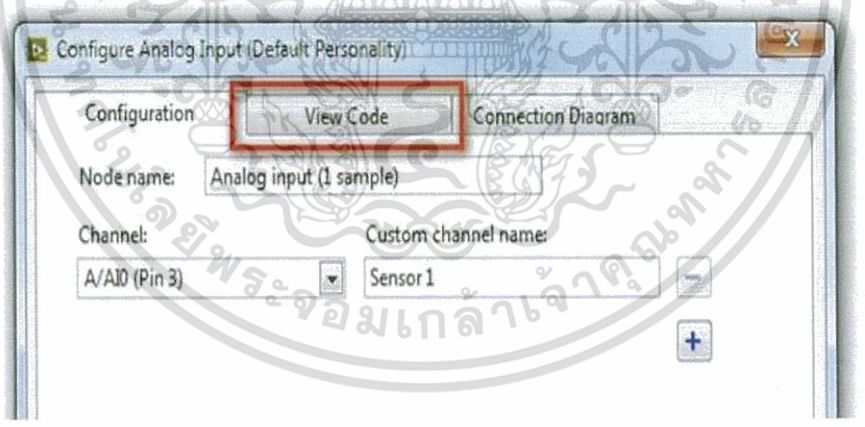
LabVIEW 2014 myRIO เครื่องมือแนะนำคุณสมบัติใหม่ ๆ สำหรับผู้ใช้ myRIO

ด้วย myRIO I / O Monitor, นักเรียนและนักศึกษาสามารถทดสอบการเชื่อมต่อกับขาใด ๆ ที่มีอยู่ใน FPGA ซึ่งเป็นจุดเริ่มต้นของ myRIO



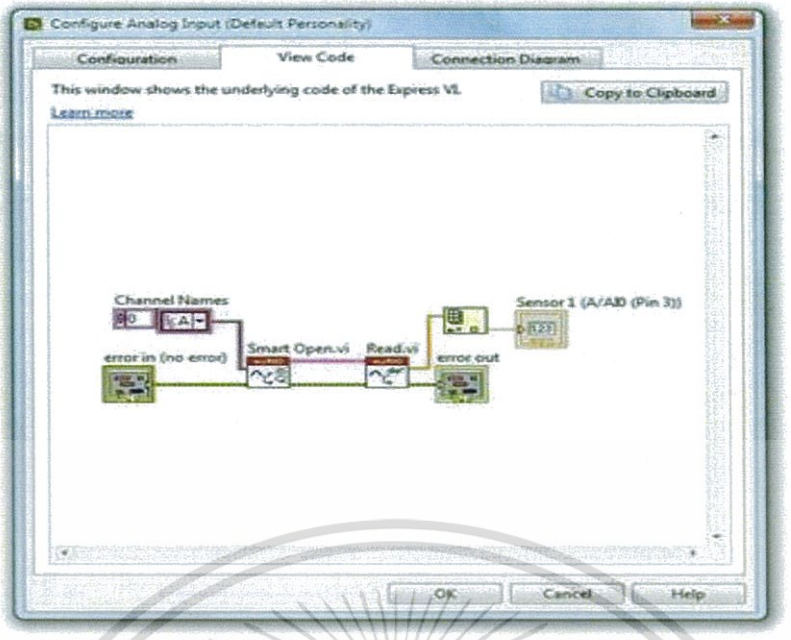
รูปที่ 2.17 myRIO I/O Monitor [5]

โดยใช้รูปแบบใหม่ของ VI นักเรียนไม่เพียงแต่เห็นภาพรหัส code แต่ยังเชื่อมต่อกับทางกายภาพที่ต้องการที่จะทำให้การใช้ประโยชน์จาก Input / Output ตรงตามประเภทที่เราเลือก



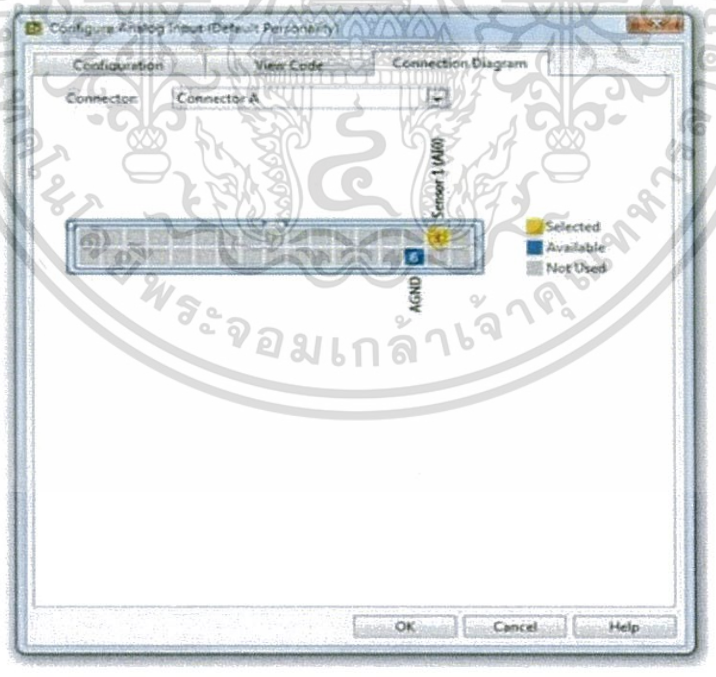
รูปที่ 2.18 myRIO Express VI [5]

คลิกดูรหัสแท็บใด ๆ ในmyRIO express VI จะเผยให้เห็นรหัสระดับที่ต่ำกว่าซึ่งจะช่วยให้นักเรียนได้เรียนรู้แนวคิดการเขียนโปรแกรมขั้นสูง



รูปที่ 2.19 myRIO Express VI View Code Feature [5]

เลือกแท็บแผนภาพการเชื่อมต่อ จะแสดงให้เห็นแผนภาพการอ้างอิงเพื่อช่วยในการเซ็นเซอร์การเดินสายไฟและกระตุ้นการ myRIO



รูปที่ 2.20 myRIO Express VI Connection Diagram [5]

myRIO Express VI แผนภาพการเชื่อมต่อ

LabVIEW 2014 myRIO เครื่องมือนี้การตรวจสอบสำหรับการขัดจังหวะเพื่อให้นักเรียนสามารถเรียกเหตุการณ์ที่เกิดขึ้นเมื่อมีเงื่อนไขขึ้นอยู่กับ FPGA เช่นเดียวกับสำหรับการรวมกับจอยสติ๊ก myRIO

เริ่มต้น myRIO FPGA ที่เหมาะสำหรับการควบคุมและดำเนินงานในตัวอย่างใหม่ล่าสุดของข้อมูลที่ได้รับ Advanced users สามารถดาวน์โหลด FPGA อื่นซึ่งมีองค์ประกอบอ่านและเขียนในช่องทางเลือกหากข้อมูลรูปแบบของคลื่นเป็นสิ่งจำเป็นสำหรับการประยุกต์ใช้

2.3.2.2 myRIO Apps สำหรับเครื่องมือเครือข่าย LabVIEW

myRIO แอปพลิเคชั่น LabVIEW , เครื่องมือเครือข่ายซอฟต์แวร์ LabVIEW , ดาวน์โหลด Add-on ที่ขยายการทำงานของ myRIO , ตัวอย่างของแอพพลิเคชันที่มี Oscilloscope / Function Generator ,Lock-in เครื่องขยายเสียง, Equalizer เสียง

LabVIEW เครื่องมือเร่งพัฒนาเครือข่ายการผลิตโดยการให้การเข้าถึงและรับรองของการเพิ่มบุคคลที่สามเข้าไปที่จะช่วยให้ผู้ใช้ขยายอำนาจของซอฟต์แวร์การออกแบบระบบ LabVIEW เครือข่ายเครื่องมือให้สำหรับนักพัฒนาโปรแกรมแนวหน้า LabVIEW ในการเพิ่มการแบ่งปันและการขายของพวกเขา นักเรียนและนักการศึกษาก็จะได้เข้าศึกษา Labview กับชุมชนของเรา ที่นักพัฒนาได้สร้างขึ้น

2.3.3ฮาร์ดแวร์ Add-on

NI ภาคภูมิใจที่ได้ร่วมมือกับหลาย บริษัท ทั่วโลกที่จะขยายการทำงานของ myRIO ผ่านทางฮาร์ดแวร์ add-on

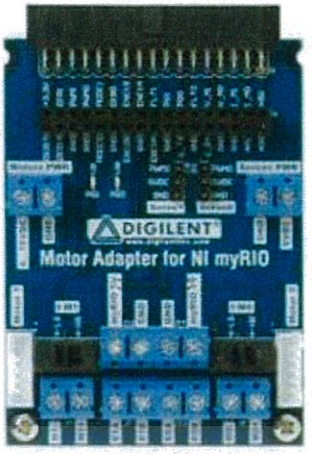
Add-ons รวมถึง:Quanser QUBE สำหรับการควบคุมการเรียนการสอน

รวมฮาร์ดแวร์ myRIO ซอฟต์แวร์ LabVIEW และ Quanser QUBE-Servo ในการสร้างแบบครบวงจร, การแก้ปัญหาห้องปฏิบัติการพร้อมสำหรับนักเรียนที่จะ "ทำวิศวกรรม" จากเบื้องต้นจากการเรียนการควบคุมขั้นสูง

รูปที่2.21 NI myRIO [5]

รูปที่ 2.22 Stratom X-CAN Adapter [5]

เชื่อมต่อเซ็นเซอร์ถึง myRIO ส่งและรับข้อความที่สามารถใช้ฮาร์ดแวร์ add-on ที่ใช้งานง่าย และ LabVIEW API



รูปที่ 2.23 DIGILENT Motor Adapter [5]

เชื่อมต่อและควบคุมทั้งมอเตอร์เดี่ยว สองมอเตอร์กระแสตรงหรือ 2 servos หรือผ่านการเชื่อมต่อแบบอิสระ MXP บนอุปกรณ์ myRIO

รูปที่ 2.24 Digilent High Current Adapter [5]

ใช้อุปกรณ์ myRIO ในการโต้ตอบกับแอพพลิเคชั่นที่มีกระแสไฟฟ้าแรงสูงในปัจจุบันเช่น แถบไฟ LED, รีเลย์ และServos

รูปที่ 2.25 Digilent NXT Sensor Adapter [5]

เพิ่ม LEGO® MINDSTORMS® NXT เซ็นเซอร์ กับระบบนิเวศ myRIO

รูปที่ 2.26 Digilent Shield Adapter [5]

เชื่อมต่ออุปกรณ์ myRIO เพื่อที่จะป้องกันระบบ Arduino .

บทที่ 3

การออกแบบและการจัดทำโครงงาน

เป็นการออกแบบซอฟแวร์ โดยใช้ โปรแกรม LabVIEW ในการออกแบบ เพื่อสร้างโปรแกรมเพื่อตรวจสอบคุณภาพชิ้นงานโดยให้ตรงตามมาตรฐาน โดยใช้ NI myRIO

3.1 การออกแบบซอฟต์แวร์ตรวจสอบคุณภาพชิ้นงาน

3.1.1 การลงโปรแกรมต่างๆที่สำคัญ

3.1.1.1 ลงโปรแกรม NI LABVIEW 2014 MYRIO SOFTWARE BUNDLE DVD 1 และ DVD 2 เพื่อให้สามารถใช้โปรแกรม LabVIEW 2014 ได้

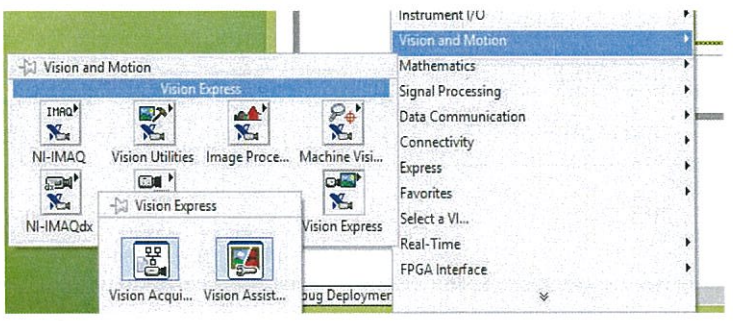
3.1.1.2 ลงโปรแกรมเสริมที่ใช้ใน LabVIEW 2014 คือ Vision development module (VDM) และ Vision acquisition software (VAS) เพื่อให้สามารถใช้ฟังก์ชั่น Vision and motion ได้

3.1.1.3 ลงโปรแกรม driver ของกล้อง webcam;207:16M.Pixel Web Camera with Infrared and Microphone เพื่อให้สามารถใช้กล้องเวปแคมกับคอมพิวเตอร์โน้ตบุ๊กได้

3.1.2 ออกแบบซอฟต์แวร์ตรวจสอบคุณภาพชิ้นงาน

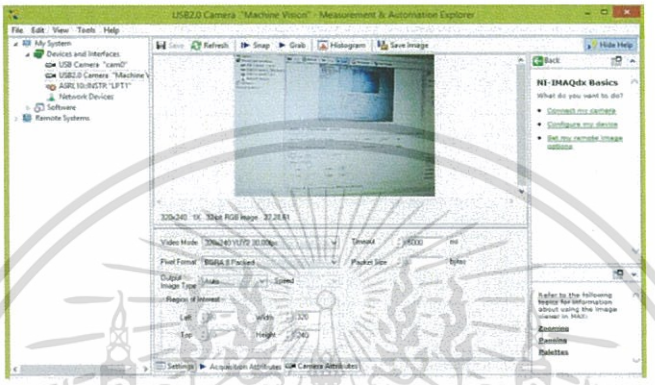
3.1.2.1 การเตรียมซอฟต์แวร์และกล้อง

ตรวจสอบว่าได้ติดตั้งซอฟต์แวร์เสริม คือ VDM และ VAS โดยตรวจสอบได้จาก Functions Palette : >>Vision and Motion>>Vision Express จะต้องมีบล็อก Vision Acquisition และ Vision Assistant อยู่ครบทั้ง 2 อัน ดังรูปที่ 3.1



รูปที่ 3.1 VDM และ VAS

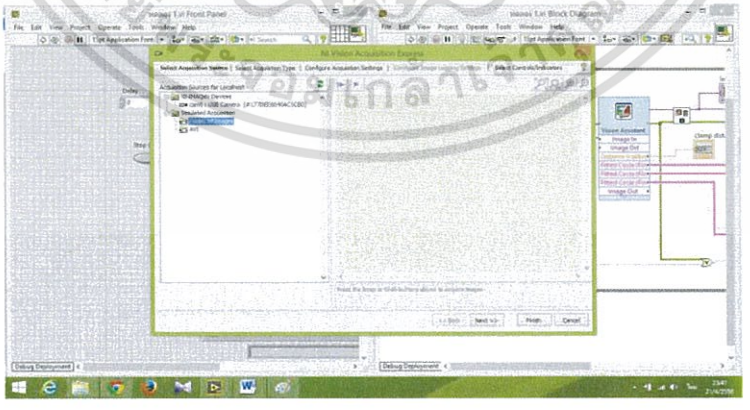
จากนั้น เข้าไปใน MAX แล้วเลือกหัวข้อ Device and Interface >> NI-IMAQdx Devices ดังรูป 3.2 เพื่อถ่ายภาพชิ้นงานที่ต้องการนำมาตรวจสอบคุณภาพโดยเลือกกล้องเวปแคมที่จะนำมาใช้ถ่ายรูปคือ USB 2.0 Camera ในการถ่ายรูปต้องให้ระยะที่ถ่ายรูปนั้นเท่ากันเสมอ รวมถึงถ่ายรูปชิ้นงานที่เป็นต้นแบบด้วย เมื่อได้ภาพชิ้นงานที่จะนำมาตรวจสอบ กด Save image เพื่อบันทึกภาพ ถ้ามีชิ้นงานที่นำมาตรวจสอบหลายชิ้นก็สามารถถ่ายแล้วนำมาตรวจสอบทีเดียวได้



รูปที่ 3.2 USB2.0 Camera “Machine Vision” – Measurement & Automation Explorer

3.1.2.2 การดึงภาพและแสดงภาพใน LabVIEW

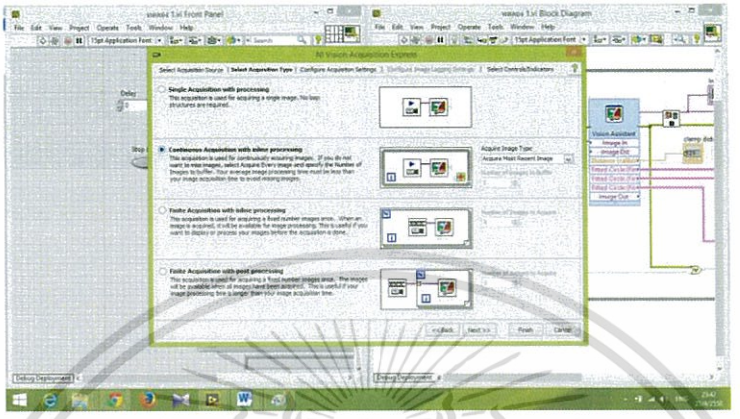
เราจะใช้ Vision Acquisition Express VI สำหรับดึงภาพ โดยเมื่อเราวางตัวนี้ลงไป เราจะพบหน้าต่างสำหรับตั้งค่า เริ่มจากหัวข้อ Select Acquisition Source เลือก Folder of images เพื่อนำภาพชิ้นงานที่จะนำมาตรวจสอบที่ถ่ายไว้ก่อนหน้านี้มาใช้ ดังรูปที่ 3.3



รูปที่ 3.3 NI Vision Acquisition Express : Select Acquisition source

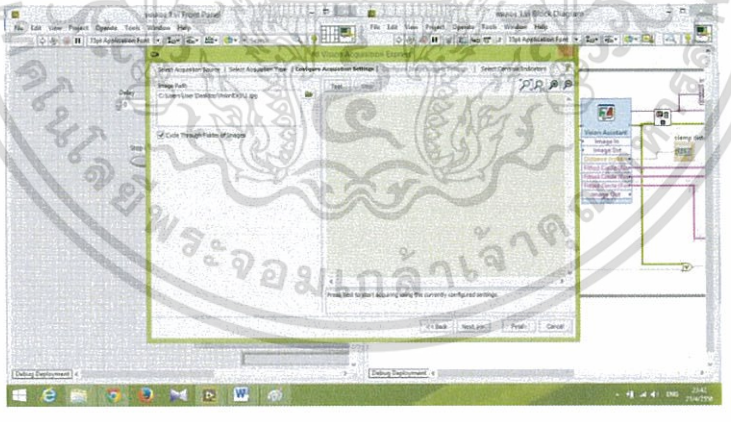
ถัดไปกดปุ่ม Next ไปที่หัวข้อ Select Acquisition Type โดยเราจะเลือกโหมด “Continuous Acquisition with inline Processing” เพื่อให้ดึงภาพต่อเนื่องแล้วประมวลผลต่อไปในลูปทันที และเลือก “Ac-

quire Most Recent Image" สำหรับกรณีที่ดึงภาพด้วยกล้องแล้วใช้เวลาประมวลผลช้ากว่าความเร็วของกล้อง ในครั้งต่อไปซอฟต์แวร์จะทำการข้ามภาพในช่วงที่กำลังประมวลผลออกไปแล้วดึงภาพที่เป็นปัจจุบันที่สุดแทน ดังรูปที่ 3.4



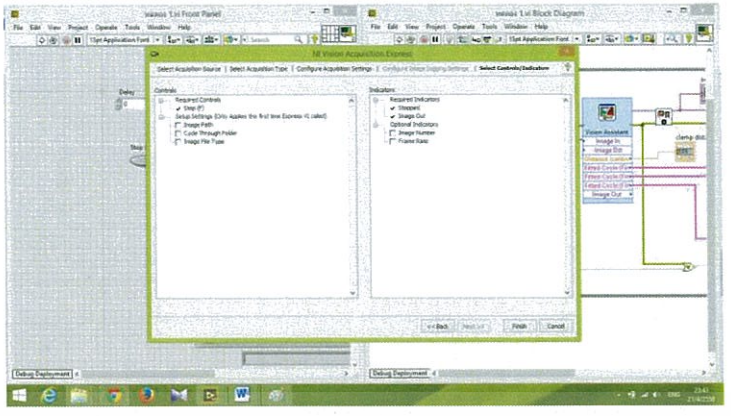
รูปที่ 3.4 NI Vision Acquisition Express : Select Acquisition Type

ถัดไปกดปุ่ม next ไปที่หัวข้อ Configure Acquisition Settings ก่อนหน้าที่จะเลือกภาพที่จะนำมาตรวจสอบให้รวมภาพทั้งหมดไว้ในโฟลเดอร์เดียว จากนั้นเลือกไฟล์ภาพที่ต้องการจาก โฟลเดอร์นั้นมาหนึ่งภาพ แล้วเลือกที่ "Cycle Through Folder od Images" เพื่อนำภาพทุกภาพในโฟลเดอร์นี้มาวนใช้ ดังรูปที่ 3.5



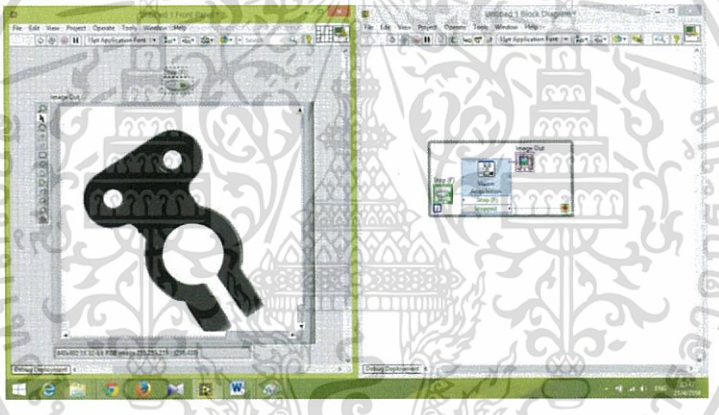
รูปที่ 3.5 NI Vision Acquisition Express : Configure Acquisition Settings

ถัดไปกดปุ่ม next โปรแกรมจะข้ามหัวข้อ Configure Image Logging Settings เนื่องจากเราใช้ภาพที่บันทึกไว้ก่อนหน้านี้แล้ว โดยข้ามไปหัวข้อ Select Controls/Indicators เพื่อเลือก input/output ตามรูปที่ 3.6



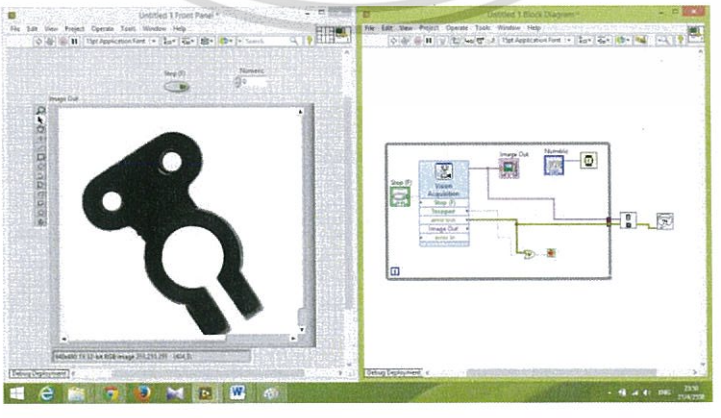
รูปที่ 3.6 NI Vision Acquisition Express : Select Controls/Indicators

เมื่อกดปุ่ม Finish เพื่อจบการตั้งค่า LabVIEW จะได้ภาพบน front panel และ block diagram ดังรูปที่ 3.7 ซึ่งเราสามารถลอง run ภาพบน front panel ได้เลย



รูปที่ 3.7 front panel และ block diagram ที่ได้จากการเขียน Vision Acquisition

จากนั้นเพิ่มฟังก์ชั่นใน block diagram ดังรูปที่ 3.8



รูปที่ 3.8 block diagram เมื่อเพิ่มฟังก์ชั่นต่างๆ

-  Or function ถ้า input ทั้งสองเป็น False ผลลัพธ์ที่ได้จะเป็น False ถ้ามีตัวใดตัวหนึ่งเป็น true ผลลัพธ์ที่ได้จะเป็น true ทันที

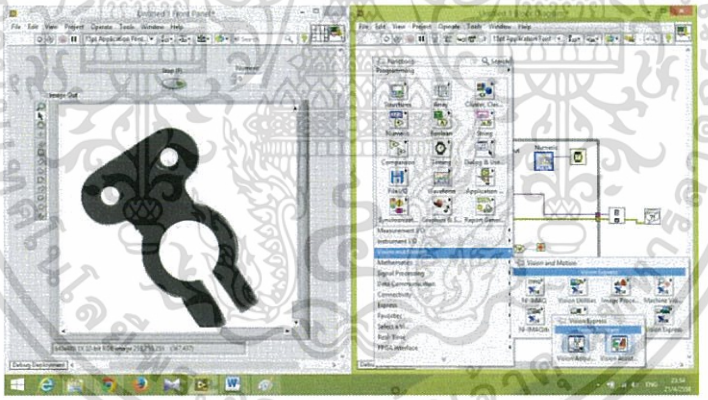
-  Wait function เพื่อสร้าง delay ในการเปลี่ยนรูปเมื่อ run โปรแกรม

-  IMAQ Dispose VI เพื่อลบข้อมูลภาพออกจากหน่วยความจำทุกครั้งเมื่อจบโปรแกรม เพื่อคืนหน่วยความจำให้ระบบ

-  Simple Error Handler VI เพื่อเป็นตัวบอกว่าเกิด error โดยจะแสดงว่าผิดพลาดส่วนไหนใน block diagram

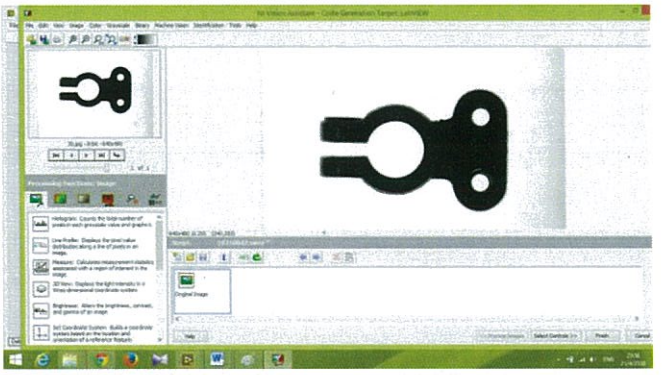
3.1.2.3 การวัดขนาดชิ้นงาน

ในการวัดขนาดชิ้นงานนั้น เราจะใช้ Vision Assistant Express VI ดังรูปที่ 3.9



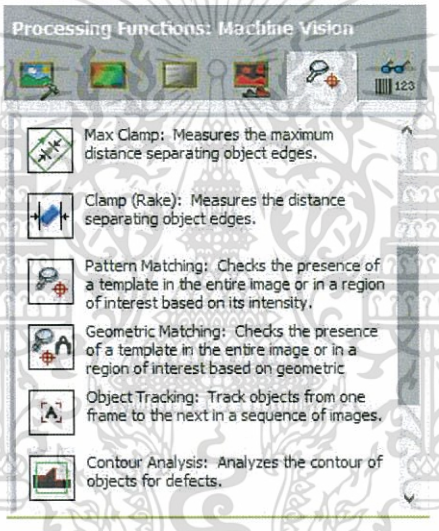
รูปที่ 3.9 Vision Assistant Express VI

3.1.2.3.1 เปิดรูปเพื่อแสดงต้นแบบใน Vision Assistant โดยการเลือก เมนู File>>Open Image และเลือกรูปทั้งหมดที่ต้องการนำมาตรวจสอบดังรูป 3.10



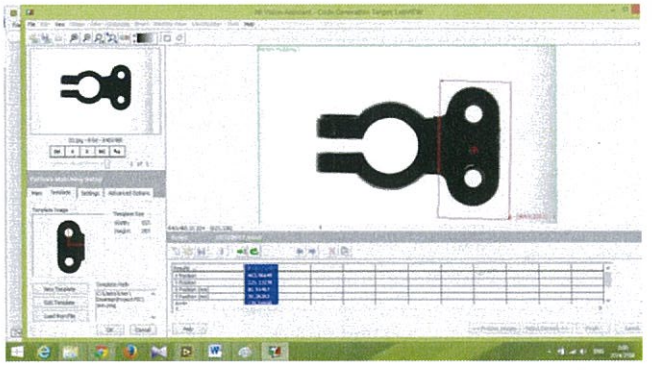
รูปที่ 3.10 ภาพที่จะนำมาวัดขนาด

3.1.2.3.2 ระบุตำแหน่งชิ้นงานด้วย Pattern Matching ดังรูป 3.11



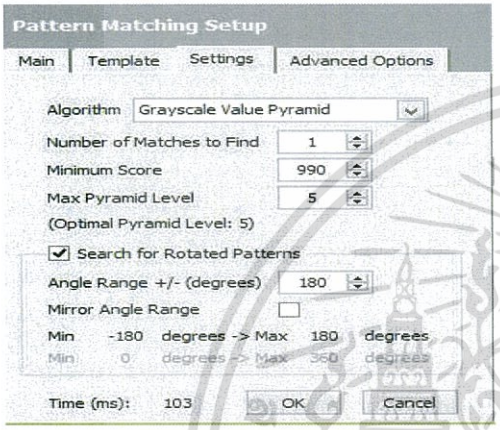
รูปที่ 3.11 Pattern Matching

จากนั้นเลือกหัวข้อ template แล้วกด New template แล้วลากกรอบสี่เหลี่ยมให้ได้ดังรูปที่ 3.12 จากนั้นกด finish แล้ว save template นั้น

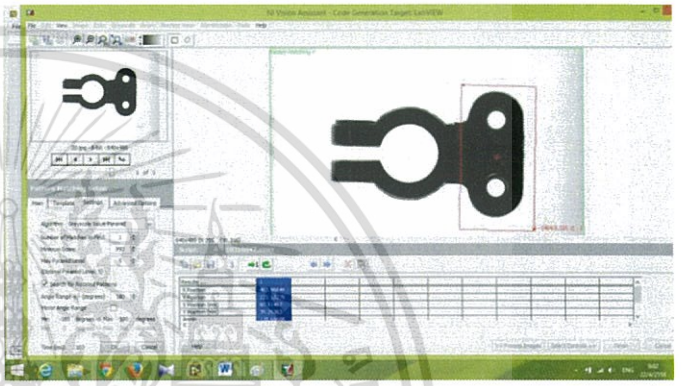


รูปที่ 3.12 Template Image

จากนั้นเลือกหัวข้อ Settings ให้ทำการหาส่วนที่เหมือนกับ template จำนวน 1 จุด (เลือกจุดที่เหมือนมากที่สุดเพียงจุดเดียว) โดยให้คะแนนความเหมือนที่ 990 (คะแนนเต็ม 1000) และสามารถทำการหาในแนวองศาต่างๆ ทีเอียงได้ ±180 องศา (ยิ่งหมุนมากก็จะยิ่งใช้เวลาการคำนวณซ้ำมากขึ้น) ดังรูป 3.13 และเมื่อกดปุ่ม OK จะได้ดังรูป 3.14 จากนั้นทำในลักษณะเดิมแต่เปลี่ยน template ให้ครอบด้านหน้าดังรูปที่ 3.15



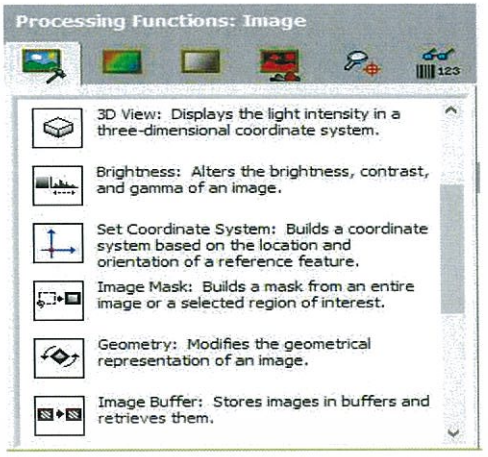
รูปที่ 3.13 Pattern Matching Setup : Settings



รูปที่ 3.14 รูปสำเร็จของ pattern matching 1

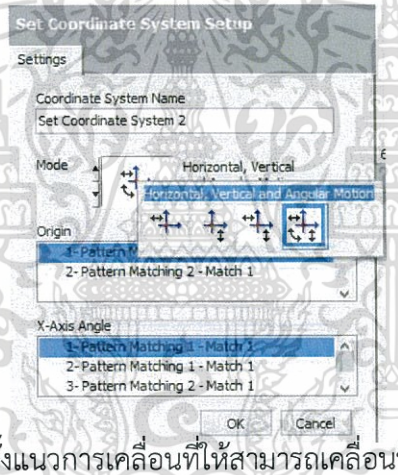
รูปที่ 3.15 รูปสำเร็จของ pattern matching 1 และ 2

3.1.2.3.3 ตั้งแกนอ้างอิงตำแหน่งด้วย Set Coordinate System ดังรูปที่ 3.16



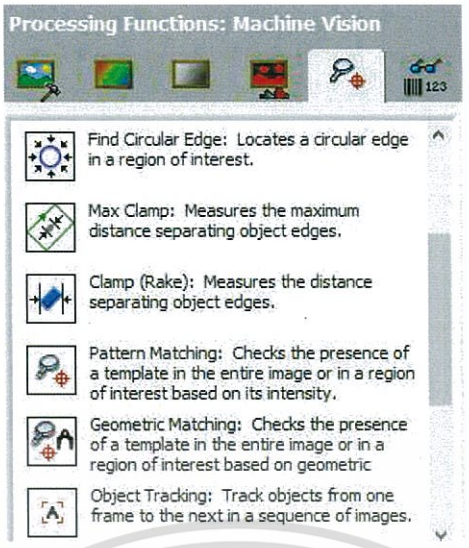
รูป 3.16 Set Coordinate System

จากนั้นเลือก mode ให้ set แกน ให้สามารถเคลื่อนที่ไปได้ทั้งแนวนอนแนวตั้งหรือหมุนได้ดังรูปที่ 3.17



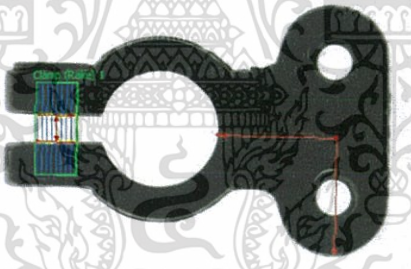
รูปที่ 3.17 การตั้งแนวการเคลื่อนที่ให้สามารถเคลื่อนที่ไปได้ทุกแนว

3.1.2.3.4 วัดความกว้างด้วย Clamp โดยเลือก Clamp (Rake) ดังรูปที่ 3.18



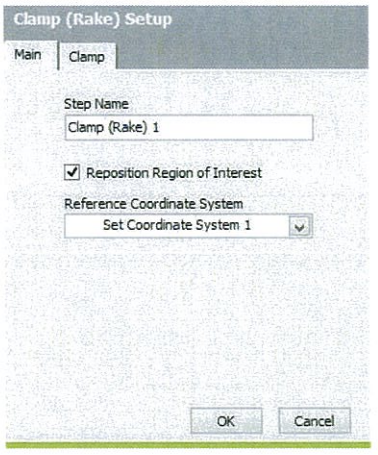
รูปที่ 3.18 Clamp (Rake)

เลือกหัวข้อ Clamp แล้วกดที่ process แล้วเลือกอันสุดท้ายเพื่อวัดความกว้างของตัวหนีบ จากนั้นลากกรอบเพื่อวัดความกว้างของตัวหนีบดังรูปที่ 3.19



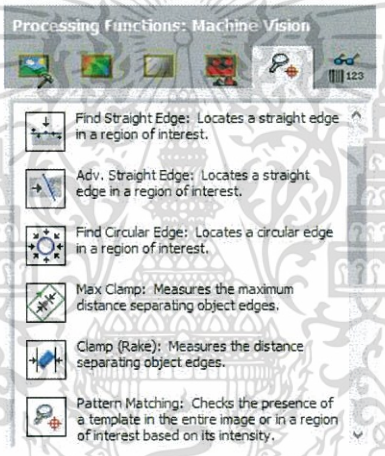
รูปที่ 3.19 ภาพเมื่อทำการลากกรอบ clamp

จากนั้นเลือกหัวข้อ main แล้ว set ค่าดังรูปที่ 3.20



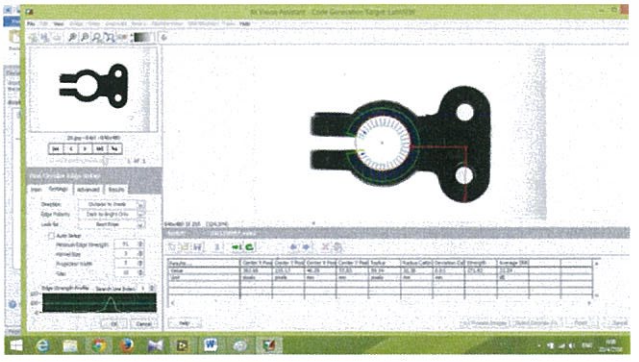
รูปที่ 3.20 การ set ค่าที่ให้สามารถเปลี่ยนตำแหน่งไปตามแกนอ้างอิงได้

3.1.2.3.5 หาวงกลมด้วย Find Circular Edge ดังรูปที่ 3.21



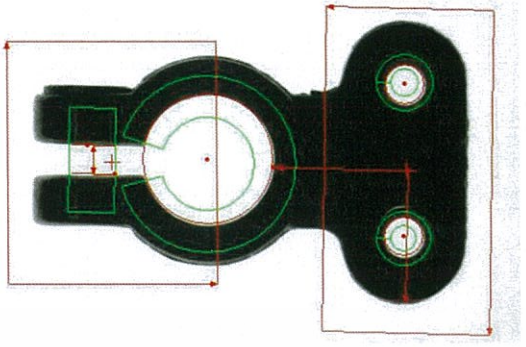
รูปที่ 3.21 Find Circular Edge

เลือกหัวข้อ setting โดยให้ Direction เป็น Outside to Inside, Edge Polarity เป็น Dark to Bright Only, Look for เป็น Best Edge จากนั้นตีกรอบวงกลมให้เป็นดังรูปที่ 3.22



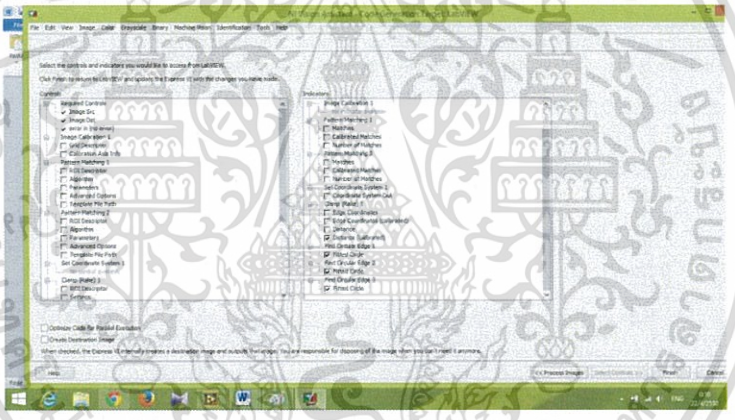
รูปที่ 3.22 การสร้าง Find Circular Edge

ถัดมาทำเหมือนเดิมแต่เปลี่ยนเป็นวงกลมอีก 2 จุดที่เหลือดังรูปที่ 3.23



รูปที่ 3.23 การสร้าง Find Circular Edge ทั้ง 3 วงกลม

3.1.2.3.6 เลือก Select control/indicator และเขียน code ของ input/output ดังรูปที่ 3.24 แล้วกด finish แล้วจะได้ฟังก์ชั่น Vision Assistant เพิ่มขึ้นมา



รูปที่ 3.24 Select control/indicator

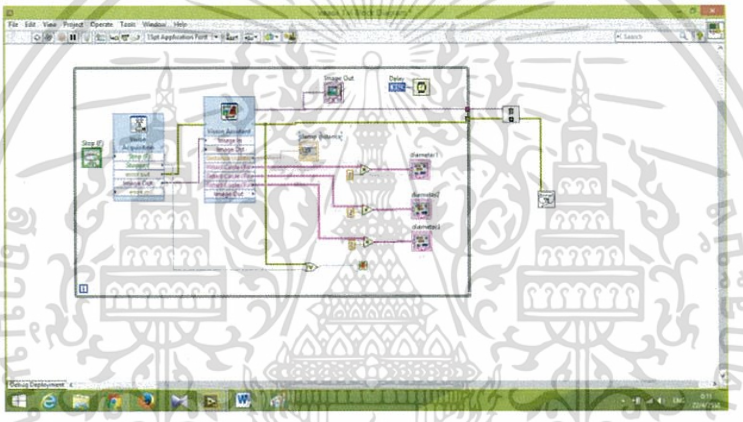
จากนั้นเพิ่มฟังก์ชั่นต่างๆและลากสายดังรูปที่ 3.25

รูปที่ 3.25 Block Diagram

- Multiply Function เพื่อคูณผลที่ได้โดยกำหนดให้คูณ 2 เพื่อให้เป็นเส้นผ่านศูนย์กลาง

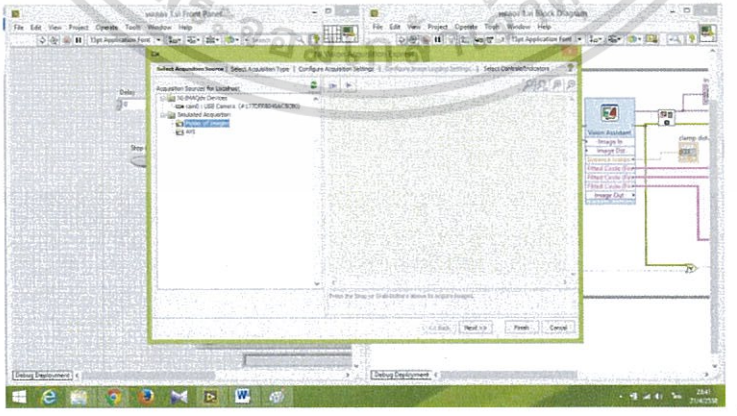
โดยที่ Output Terminal ของแต่ละฟังก์ชั่นใน Vision Assistant Express VI มีหลายรูปแบบเราสามารถรู้ได้โดยการคลิกขวาตรง Terminal แล้วเลือก Create >> Indicator เพื่อทำความเข้าใจกับรูปแบบของข้อมูลในนั้นก่อนที่จะนำไปแสดงผลหรือคำนวณต่อไป

3.1.2.3.7 การแสดงหรือซ่อน Overlay โดยเลือกฟังก์ชั่นตามต่อไปนี้ Functions Palette:>>Vision and Motion>>Vision Utilities>>Overlay เพื่อลบ Overlay เมื่อ Run ข้อมูล ดังรูปที่ 3.26



รูปที่ 3.26 IMAQ Clear Overlay VI

จากนั้นวาง IMAQ Clear Overlay VI ดังรูปที่ 3.27

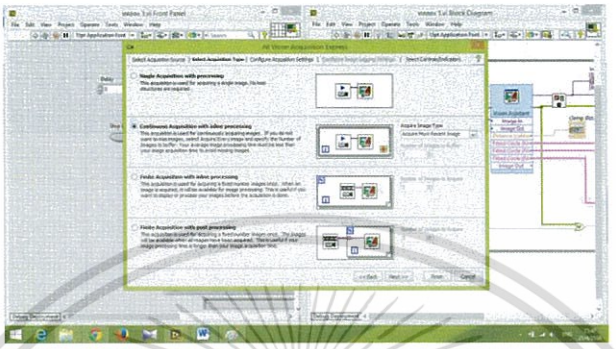


รูปที่ 3.27 Block Diagram เมื่อเพิ่มฟังก์ชั่น IMAQ Clear Overlay VI

3.1.2.3.8 สอบเทียบภาพ

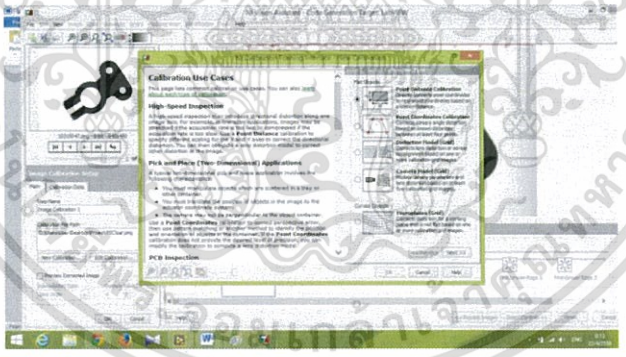
เลือกฟังก์ชั่น Vision Assistant ที่เคยสร้างไว้ใน Block Diagram จากนั้น

เลือก Image Calibration ดังรูปที่ 3.28 โดยเราจะแทรกฟังก์ชั่นนี้ใน script หลังจาก Original Image



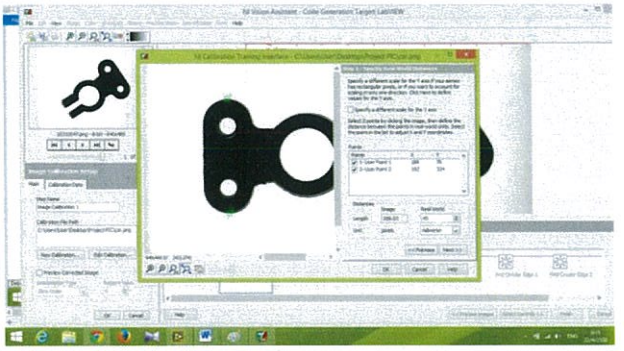
รูปที่ 3.28 Image Calibration

เลือกรูปภาพที่ตรงตามมาตรฐานที่ต้องการ จากนั้นกด New Calibration แล้วเลือก Point Distance Calibration ดังรูปที่ 3.29



รูปที่ 3.29 NI Calibration Training Interface – New Calibration

จากนั้นกด Next >> Next จะให้วัดความยาวของชิ้นงานโดยเลือกตำแหน่งที่ 1 และตำแหน่งที่ 2 ดังรูปที่ 3.30 และเลือกความยาว Real World เป็น 45 โดยหน่วยเป็น millimeter แล้วกด OK



รูปที่ 3.30 การวัดขนาดภาพจริง

ถัดมาเมื่อเราลอง Run script เพื่อตรวจสอบความถูกต้องอีกครั้ง ซึ่งเราจะพบว่าฟังก์ชั่น Set Coordinate System จะต้องแก้ไข โดยเข้าไปเลือก X-Axis Angle ให้ใหม่ เมื่อเรียบร้อยให้กดปุ่ม Finish เพื่อจบการตั้งค่าและออกไปที่โค้ด LabVIEW

กด Finish เพื่อบันทึกการสอบเทียบภาพ ถือเป็นการเสร็จสิ้นการออกแบบซอฟต์แวร์ตรวจสอบคุณภาพชิ้นงาน

3.2 เครื่องมือที่ใช้ในการทดลอง

รูปที่ 3.31 กล้อง wbcam

3.2.1 กล้อง webcam; wc-207:16M.Pixel Web Camera with Infrared and Microphone

รายละเอียดของกล้อง webcam

- 16M. Pixel with CMOS
- Infrared Function
- Built-in Microphone
- Interface : USB version 2.0
- Cord Length : 172 cm.
- Dimension : 14.2×32.8×2.25 cm.
- Plug & Play
- System requirements : Window ME , 2000 ,XP , Vista, 7 , Macintosh OSX-later

3.2.2 LabVIEW 2014

รูปที่ 3.32 โปรแกรม LabVIEW 2014

LabVIEW เป็นโปรแกรมที่ใช้ติดต่อสื่อสารกับเครื่องมือต่างๆที่อยู่ภายนอกผ่านบอร์ด Data Acquisition ใช้งานเป็น monitoring หรือในการควบคุมการวัดค่าต่างๆ เช่น strain อุณหภูมิ หรือสัญญานอื่นๆ โดยมีตัวเซนเซอร์รับสัญญานเข้ามา โดยเอาต์พุตที่ได้จากเซนเซอร์เหล่านี้จะมีค่าเป็นแรงดันหรือกระแส ซึ่ง LabVIEW สามารถอ่านค่าที่ผ่านเข้ามาทางDAQ Card แล้วบันทึกค่าเป็นไฟล์ข้อมูลได้ ดังนั้นการนำ Lab-VIEWไปใช้จะต้องพิจารณาถึงวัตถุประสงค์และ application ที่จะใช้ก่อนว่ามี input เป็นอะไร และต้องการ outputอะไร จากนั้นจึงทำการเลือก hardware ให้ตรงตามต้องการ - การใช้งาน LabVIEW ผู้ใช้ควรมีพื้นฐานด้านการเขียนโปรแกรมพอสมควร เนื่องจากการติดต่อสื่อสารระหว่างโปรแกรมกับเครื่องมือต่างๆที่อยู่ภายนอกนั้น ผู้ใช้ต้องเขียนโปรแกรมคำสั่งการทำงานเพื่อเรียกข้อมูลการวัดแล้วนำมาprocess ให้เป็น

monitering หรือการเขียนคำสั่งเพื่อการควบคุมระบบเช่น ให้โปรแกรมสามารถตรวจสอบค่า Strain ที่อ่านได้ว่าถ้ามีค่าไม่เกินกว่าที่กำหนดแล้วจึงค่อยส่งคำสั่งไปควบคุมให้อุปกรณ์อื่นๆ ทำงานต่อได้ เป็นต้น

โปรแกรม LabVIEW มีองค์ประกอบสำคัญ3 ส่วน คือ

1. Front panel เป็นส่วนตั้งค่าการวัดและอ่านค่าตัวเลขหรือกราฟที่ออกมาจากblock diagram จึงทำหน้าที่เสมือนเครื่องมือวัดจริงโดย inputที่ป้อนเข้าไปจะเป็นตัวควบคุม ส่วน output ที่ออกมาจะเป็นตัวแสดงผล

2. Block diagram ทำหน้าที่เสมือนเป็น Sourcecode โดยใช้โปรแกรมภาษากราฟฟิก องค์ประกอบของ block diagram นี้จะแทนโปรแกรม Node เช่น for loop, casestructure และฟังค์ชั่นทางคณิตศาสตร์เป็นต้น

3. Icon/Connector ภายใน Front panel จะประกอบด้วย icon ต่างๆและมีสายเชื่อมต่อถุงกันในแต่ละicon ซึ่งเมื่อเชื่มต่อกันแล้ว จะสามารถเปลี่ยน Virtualinstrument (VI) นี้ให้เป็น Sub VI หรือ Objectที่นำกลับมาใช้ใน block diagram ได้อีก

ความสามารถของโปรแกรม LabVIEW

เนื่องจากบริษัทNational Instrument(NI) ซึ่งเป็นผู้พัฒนาโปรแกรม LabVIEWมี Product ในการพัฒนาอยู่มากมายทั้ง Hardwareและ Software จึงทำให้โปรแกรม LabVIEW มีความสามารถในการติดต่อ Hardware อย่างหลากหลายเช่น

Hardware

การใช้ โปรแกรม LabVIEW เพื่อเชื่อมต่อกับฮาร์ดแวร์ภายนอกทำได้โดยผ่านทางการ์ด DAQ (data acquisition) การเชื่อมต่อสามารถเชื่อมต่อกับพอร์ต(port) ได้หลายชนิด เช่น พอร์ตขนาน (paralelport), พอร์ตอนุกรม (serial port), GPIB, และHPIBเป็นต้น จึงมีแนวความคิดในการออกแบบวงจรขึ้นมา โดยกำหนดคุณสมบัติให้เป็นบอร์ดแบบภายนอกเชื่อมต่อกับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม(RS-232) มีจำนวนอินพุต-เอาต์พุต 16 ช่อง (channel) อินพุตทำงานได้ทั้งโหมดดิจิตอลอินพุตและอนาลอกอินพุต สำหรับเอาต์พุตกำหนด ให้เป็นแบบดิจิตอลเอาต์พุต ออกแบบให้สร้างง่ายและต้นทุนต้องไม่สูงมากจนเกินไป

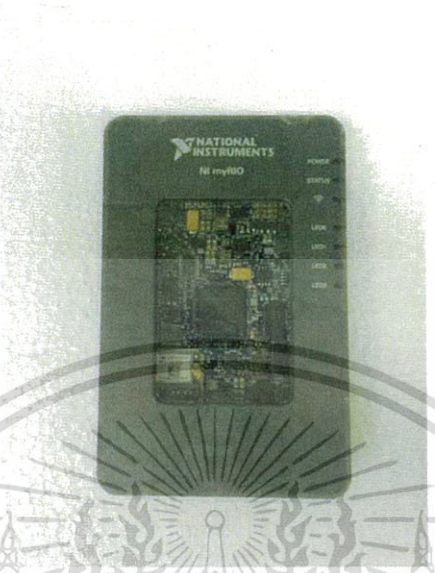
Software

1. Protocolต่างๆในทางอุตสาหกรรม LabVIEW ก็สามารถติดต่อสื่อสารได้รวมทั้ง PLC ยี่ห้อต่างๆ และงาน SCADA LabVIEW ก็สามารถทำได้เหมือนโปรแกรม SCADA ทั่วไป และบริษัท NIยังมี PLC ของตนเองขายอีก

2. ความสามารถในการทำ Image Processing ก็ทำได้ไม่แพ้ ImageProcessing ในท้องตลาด

3. สามารถติดต่อกับ Database มาตรฐานรวมทั้งการควบคุมการทำงานกับโปรแกรม MS-OFFICE และอื่นๆใน windows

3.2.3 NI myRIO



รูปที่ 3.33 NI myRIO

NI myRIO หรือ “เอ็นไอ มายริโอ้” เป็นอุปกรณ์รุ่นล่าสุดที่ใช้สถาปัตยกรรม RIO ที่ย่อมาจาก Re-configurable I/O และเป็นเทคโนโลยีที่ถูกอ้างถึงในหลายๆผลิตภัณฑ์ของ NI โดยผลิตภัณฑ์ที่ใช้เทคโนโลยี RIO และ FPGA นั้นสามารถถูกโปรแกรมได้ด้วยซอฟต์แวร์ NI LabVIEW สามารถปรับเปลี่ยนอินพุต/เอาต์พุตได้ ภายในประกอบด้วย Dual-core ARM Cortex A9 real-time โปรเซสเซอร์ และ Xillnx FPGA โดยสามารถเชื่อมต่อกับอุปกรณ์ภายนอกผ่านบัสต่างๆ และเซนเซอร์ภายนอก

ด้วยขนาดกะทัดรัดสามร พกพาได้ง่าย มีช่องสัญญาณอินพุต เอาต์พุต หลายช่องสัญญาณ และราคาไม่แพง ทำให้ NI myRIO เป็นอุปกรณ์ที่เหมาะสำหรับใช้เป็นเครื่องประกอบการสอบและการทำวิจัยสำหรับนิสิตและนักศึกษา เพื่อให้นักศึกษาได้สามารถพิสูจน์ทฤษฎีที่เรียนมาด้วยการลงมือทำจริง NI myRIO สามารถประยุกต์ใช้เพื่อการพัฒนาแอพพลิเคชั่นได้หลากหลายแขนงทั้งในแอพพลิเคชั่น ด้านวิทยาศาสตร์และวิศวกรรม อาทิ ด้านโครงงานวิศวกรรม ด้านอิเล็กทรอนิกส์เชิงกล ด้านระบบไฟฟ้าควบคุม ด้านหุ่นยนต์อัตโนมัติ หรือ ด้านระบบควบคุมแบบสมองกลฝังตัว เป็นต้น

3.3 การจัดเก็บผลการทดลอง

1.ทำการ Calibration ภาพจากขนาดจริงของชิ้นงาน เพื่อนำมาเปรียบกับ รูปที่จะนำมาตรวจสอบคุณภาพ ทั้ง 18 รูป

2.ปรับค่า Delay ให้มีค่า ระหว่าง 3000-5000 เพื่อให้เมื่อกด Run แล้ว สามารถจดข้อมูลที่ได้ คือ ความยาว Clamp และ เส้นผ่านศูนย์กลางทั้ง 3 ได้ครบก่อนที่จะไปยังรูปถัดไป

3.กด Run ข้อมูลใน โปรแกรม LabVIEW

4.กดที่สวิตช์ไฟใน Front Panel เพื่อเริ่มการ Run ข้อมูล

5.บันทึกผลการตรวจสอบคุณภาพ ทั้ง 18 รูป โดยบันทึก Clamp และ เส้นผ่านศูนย์กลางทั้ง 3 เป็นหน่วย มิลลิเมตร โดยถ้ารูปไหนขึ้นเป็น 0 แสดงว่า รูปนั้นไม่ตรงตามมาตรฐานของชิ้นงานจริง

6.เมื่อเราจดบันทึก ครบทุกรูปจึงกด สวิตช์ไฟ อีกครั้งเพื่อหยุด แล้วกลับไปที่รูปแรก

7.จากนั้นกด ปุ่มสีแดงทางด้านบนซ้าย เพื่อหยุดการ Run ข้อมูล

บทที่ 4

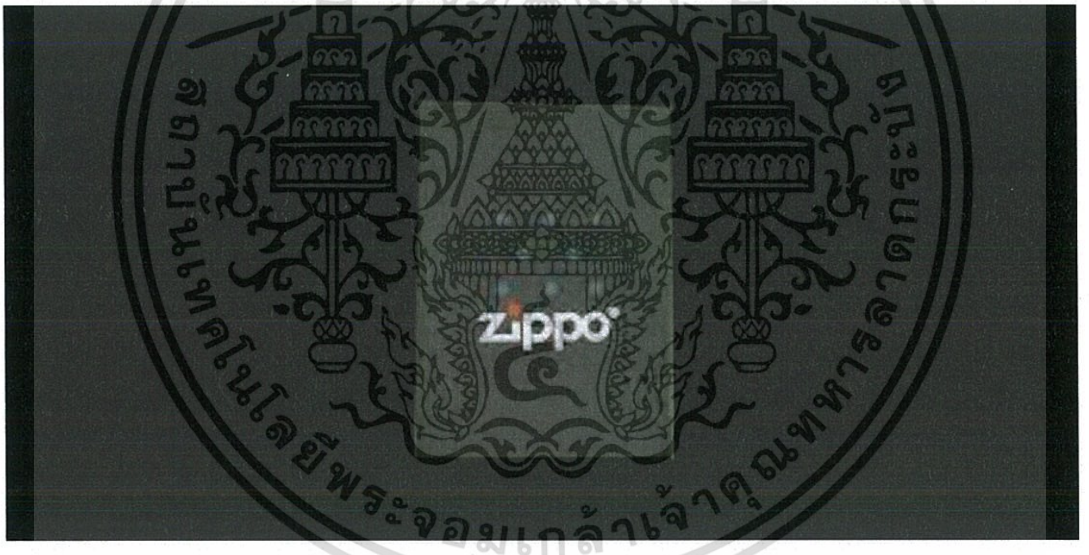
ผลการทดลอง

สำหรับการทดสอบ เพื่อตรวจสอบคุณภาพชิ้นงาน เราจะทำการหาความกว้างของที่หนีบ และเส้นผ่านศูนย์กลางของ วงกลมทั้ง 3 ในชิ้นงาน ดังต่อไปนี้

4.1 ตรวจสอบชิ้นงาน ทั้งหมด 3 ชิ้น และ 6 ชิ้น

นำชิ้นงานทั้งหมด 9 ชิ้น มาเทียบกับ ชิ้นงานมาตรฐาน

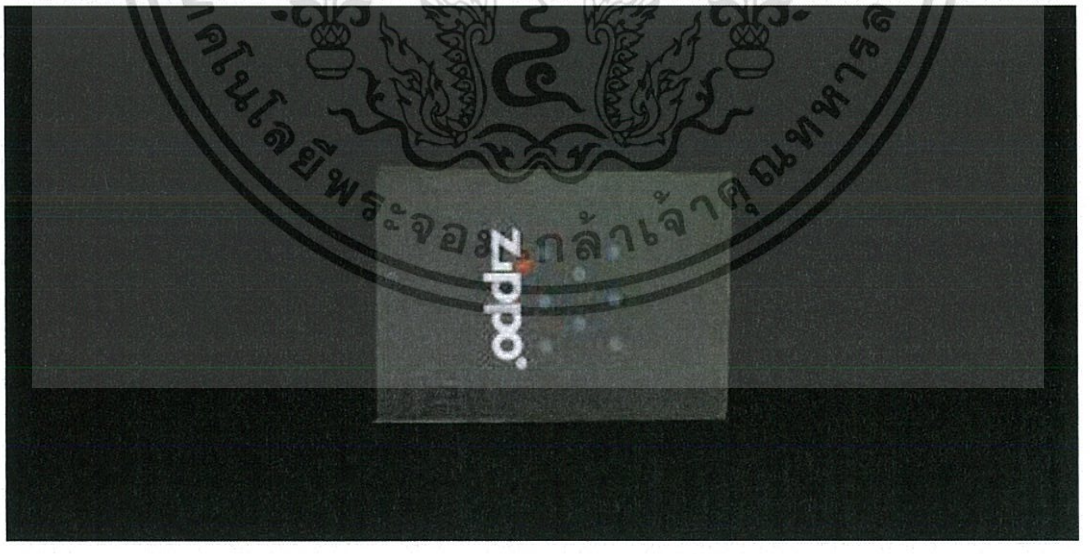
มีความยาว 8.2เซนติเมตร มีความกว้าง 5.8 เซนติเมตร และสูง 2.8 เซนติเมตร



รูปที่ 4.1 ชิ้นงานมาตรฐาน

รูปที่ 4.2 ชิ้นงานที่ตรวจสอบชิ้นที่ 1

มีความยาว 0.0เซนติเมตร มีความกว้าง 0.0 เซนติเมตร และสูง 0.0 เซนติเมตร



รูปที่ 4.3 ชิ้นงานที่ตรวจสอบชิ้นที่ 2

มีความยาว 8.2เซนติเมตร มีความกว้าง 5.8 เซนติเมตร และสูง 2.8 เซนติเมตร

รูป 4.4 ชิ้นงามมาตรฐาน 2

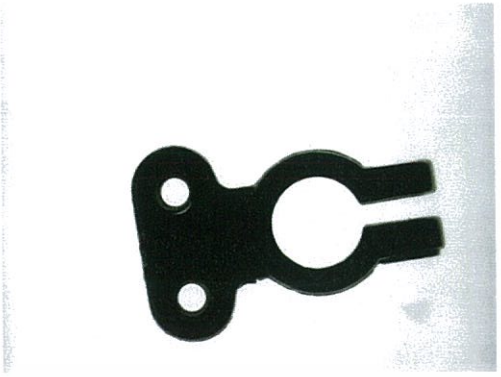
ความยาวของตัวหนีบ=4.450 มิลลิเมตร Diameter1(ใหญ่)=20.764 มิลลิเมตร

Diameter2(บน) =6.283 มิลลิเมตร Diameter3(ล่าง)=6.320 มิลลิเมตร

รูปที่ 4.5 ชิ้นงานที่ตรวจสอบชิ้นที่ 1

ความยาวของตัวหนีบ=0.000 มิลลิเมตร Diameter1(ใหญ่)=0.000 มิลลิเมตร

Diameter2(บน) =0.000 มิลลิเมตร Diameter3(ล่าง)=0.000 มิลลิเมตร



รูปที่ 4.6 ชิ้นงานตรวจสอบชิ้นที่ 2

ความยาวของตัวหนีบ=4.450 มิลลิเมตร Diameter1(ใหญ่)=20.764 มิลลิเมตร

Diameter2(บน) =6.283 มิลลิเมตร Diameter3(ล่าง)=6.320 มิลลิเมตร

รูปที่ 4.7 ชิ้นงานตรวจสอบชิ้นที่ 3

ความยาวของตัวหนีบ=0.000 มิลลิเมตร Diameter1(ใหญ่)=0.000 มิลลิเมตร

Diameter2(บน) =0.000 มิลลิเมตร Diameter3(ล่าง)=0.000 มิลลิเมตร

รูปที่ 4.8 ชิ้นงานตรวจสอบชิ้นที่ 4

ความยาวของตัวหนีบ=4.450 มิลลิเมตร Diameter1(ใหญ่)=20.764 มิลลิเมตร

Diameter2(บน) =6.283 มิลลิเมตร Diameter3(ล่าง)=6.320 มิลลิเมตร

รูปที่ 4.9 ชิ้นงานตรวจสอบชิ้นที่ 5

ความยาวของตัวหนีบ=4.450 มิลลิเมตร Diameter1(ใหญ่)=20.764 มิลลิเมตร

Diameter2(บน) =6.283 มิลลิเมตร Diameter3(ล่าง)=6.320 มิลลิเมตร

รูปที่ 4.10 ชิ้นงานตรวจสอบชิ้นที่ 6

ความยาวของตัวหนีบ=4.450 มิลลิเมตร Diameter1(ใหญ่)=20.764 มิลลิเมตร

Diameter2(บน) =6.283 มิลลิเมตร Diameter3(ล่าง)=6.320 มิลลิเมตร

บทที่ 5

สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

การทดสอบเครื่องตรวจสอบคุณภาพชิ้นงานอัตโนมัติ จะดำเนินการโดยป้อนชิ้นงานลงสายพาน ลำเลียง และกล้องจะทำหน้าที่ในการวัดขนาดความกว้าง ความยาวและวงกลม ที่มีสีแดง สีเขียว และสีน้ำเงิน ในเครื่องคัดแยกชิ้นงานอัตโนมัติ ในแต่ละรอบการทำงาน จะเห็นได้ว่าค่าความผิดพลาดในการตรวจสอบคุณภาพของชิ้นงาน 4.4% ค่าความผิดพลาดในการคัดแยกสีของชิ้นงาน 2.3% เวลาที่ใช้ในการทดสอบคือ 29.96 วินาที

5.2 ข้อเสนอแนะ

ในระบบการตรวจสอบคุณภาพของชิ้นงานนั้น ต้องมีความแม่นยำสูงและไม่ซับซ้อนในการหาค่าต่างๆ ที่ต้องการในชิ้นงาน เพื่อให้ผู้ผลิตใช้งานได้ไม่ยุ่งยาก และให้ผู้บริโภคได้สินค้าตามที่ต้องการ โดยใช้พื้นที่น้อย และเคลื่อนย้ายสะดวก

บรรณานุกรม

[1] กิจไพบูลย์ ชีวพันธุศรี, NATIONAL INSTRUMENTS LabVIEW , พิมพ์ที่บริษัท วี.พริ้นท์ (1991) , ปีที่พิมพ์ 2557

[2] Turek, Fred D. (June 2011) , "Machine Vision Fundamentals, How to Make Robots See". *NASA Tech Briefs* 35 (6): 60–62 , Retrieved 2011-11-29.

[3] Murray, Charles J (February 2012) , "3D Machine Vison Comes into Focus". *Design News* , Retrieved 2012-05-12.

[4] Wilson, Andrew (December 2011) , "Product Focus - Looking to the Future of Vision" ,*Vision Systems Design* 16 (12) , Retrieved 2013-03-05.

[5] National Instruments Thailand Co., Ltd. NI myRIO , http://thailand.ni.com/.

ภาคผนวก

โปรแกรม Labview ที่ใช้ในโครงงาน

Function Programming

Add Function

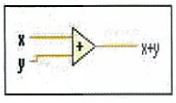
Owning Palette: Numeric Functions

Requires: Base Development System

Computes the sum of the inputs.

If you wire two waveform values or two dynamic data type values to this function, **error in** and **error out** terminals appear on the function. You cannot add two time stamp values together. The dimensions of two matrices you want to add must be the same. Otherwise, this function returns an empty matrix. The connector pane displays the default data types for this polymorphic function.

Details Example



Add to the block diagram Find on the palette

x can be a scalar number, array or cluster of numbers, array of clusters of numbers, and so on.

y can be a scalar number, a fixed-point number, an array or cluster of numbers, an array of clusters of numbers, a time stamp, and so on.

x+y is the sum of **x** and **y**.

Note You can manually configure this function to output data of a type you want. To specify the output data type, right-click the function and select **Properties** to display the **Object Properties** dialog box. On the **Output Configuration** page, click the **Representation** icon and select the data type you want. A blue coercion dot appears on the output terminal of the function to indicate that you have configured the output data type.

Divide Function

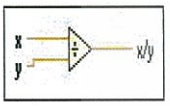
Owning Palette: Numeric Functions

Requires: Base Development System

Computes the quotient of the inputs.

If you wire two waveform values or two dynamic data type values to this function, **error in** and **error out** terminals appear on the function. The connector pane displays the default data types for this polymorphic function.

Details Example



Add to the block diagram Find on the palette

x can be a scalar number, array or cluster of numbers, array of clusters of numbers, and so on.

y can be a scalar number, array or cluster of numbers, array of clusters of numbers, and so on.

x/y is a double-precision, floating-point number if both **x** and **y** are integers. In general, the output type is the widest representation of the inputs if the inputs are not integers or if their representations differ.

Note You can manually configure this function to output data of a type you want. To specify the output data type, right-click the function and select **Properties** to display the **Object Properties** dialog box. On the **Output Configuration** page, click the **Representation** icon and select the data type you want. A blue coercion dot appears on the output terminal of the function to indicate that you have configured the output data type.

Square Function

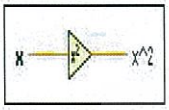
Owning Palette: Numeric Functions

Requires: Base Development System

Computes the square of the input value.

The connector pane displays the default data types for this polymorphic function.

Details Example



Add to the block diagram Find on the palette

x can be a scalar number, array or cluster of numbers, array of clusters of numbers, and so on.

x^2 is of the same numeric representation as **x**.

Note You can manually configure this function to output data of a type you want. To specify the output data type, right-click the function and select **Properties** to display the **Object Properties** dialog box. On the **Output Configuration** page, click the **Representation** icon and select the data type you want. A blue coercion dot appears on the output terminal of the function to indicate that you have configured the output data type.

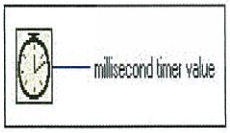
Tick Count (ms) Function

Owning Palette: Timing VIs and Functions

Requires: Base Development System

Returns the value of the millisecond timer.

The base reference time (millisecond zero) is undefined. That is, you cannot convert **millisecond timer value** to a real-world time or date. Be careful when you use this function in comparisons because the value of the millisecond timer wraps from (2^32)-1 to 0.



Add to the block diagram Find on the palette

Not And Function

Owning Palette: Boolean Functions

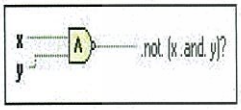
Requires: Base Development System

Computes the logical NAND of the inputs. Both inputs must be Boolean values, numeric values, or error clusters. If both inputs are TRUE, the function returns FALSE. Otherwise, it returns TRUE.

Note This function performs bitwise operations on numeric inputs.

The connector pane displays the default data types for this polymorphic function.

Details Example



Add to the block diagram Find on the palette

x must be a Boolean value or a number. **x** can be a scalar, array or cluster of numbers or Boolean values, array of clusters of numbers or Boolean values, and so on. If **x** is an error cluster, only the **status** parameter of the error cluster passes to the input terminal.

y must be a Boolean value or a number. **y** can be a scalar, array or cluster of numbers or Boolean values, arrays of clusters of numbers or Boolean values, and so on. If **y** is an error cluster, only the **status** parameter of the error cluster passes to the input terminal.

.not. (x .and. y)? is the logical NAND of **x** and **y**.

And Function

Owning Palette: Boolean Functions

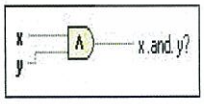
Requires: Base Development System

Computes the logical AND of the inputs. Both inputs must be Boolean values, numeric values, or error clusters. If both inputs are TRUE, the function returns TRUE. Otherwise, it returns FALSE.

Note This function performs bitwise operations on numeric inputs.

The connector pane displays the default data types for this polymorphic function.

Details Example



Add to the block diagram Find on the palette

x must be a Boolean value or a number. **x** can be a scalar, array or cluster of numbers or Boolean values, array of clusters of numbers or Boolean values, and so on. If **x** is an error cluster, only the **status** parameter of the error cluster passes to the input terminal.

y must be a Boolean value or a number. **y** can be a scalar, array or cluster of numbers or Boolean values, arrays of clusters of numbers or Boolean values, and so on. If **y** is an error cluster, only the **status** parameter of the error cluster passes to the input terminal.

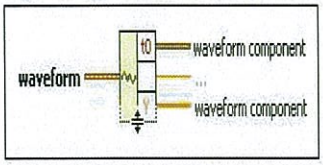
x .and. y? is the logical AND of **x** and **y**.

Get Waveform Components (Analog Waveform) Function

Owning Palette: Waveform VIs and Functions

Requires: Base Development System

Returns the analog waveform you specify. You specify components by clicking on the center of the output terminal and selecting the component you want.



waveform is the waveform from which you want to retrieve components.

t0 returns the trigger time of the waveform.

dt returns the time interval in seconds between data points in the waveform.

Y returns the data values of the waveform.

attributes returns the names and values of all waveform attributes. You also can use the Get Waveform Attribute VI to retrieve the names and values of all attributes or the value of a single attribute.

Obtain Notifier Function

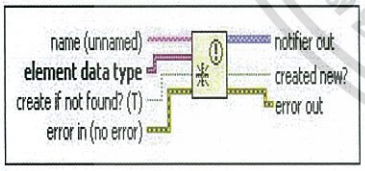
Owning Palette: Notifier Operations Functions

Requires: Base Development System

Returns a reference to a notifier.

Use this reference when calling other Notifier Operations functions.

Details



Add to the block diagram Find on the palette

name contains the name of the notifier that you want to obtain or create. The default is an empty string to create an unnamed notifier.

element data type is the type of data that you want the notifier to contain. You can wire any data type to this input.

create if not found? specifies whether you want to create a new notifier if one with the same name as **name** does not exist. If TRUE (default), the function creates a notifier if one with the same name does not exist.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

notifier out is a reference to the existing named notifier or the new notifier created by this function.

created new? is TRUE if the function created a new notifier.

error out contains error information. This output provides standard error out functionality.

Obtain Semaphore Reference VI

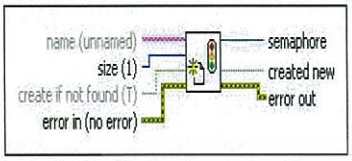
Owning Palette: Semaphore VIs

Requires: Base Development System

Obtains a reference to an existing semaphore or creates a new semaphore and returns a reference to that semaphore.

Use this VI in conjunction with the other Semaphore VIs to implement a semaphore in LabVIEW.

Details Example



Add to the block diagram Find on the palette

name contains the name of the semaphore you want to look up or create. The default is an empty string to create an unnamed semaphore. If you wire **name**, LabVIEW searches for an existing semaphore with the same name and returns a unique reference to the existing semaphore. If a semaphore with the same name does not already exist and **create if not found** is TRUE, LabVIEW creates a new, named semaphore and returns a unique reference to that semaphore.

size specifies how many tasks can acquire the semaphore at the same time. If a named semaphore already exists, wiring a value to this parameter does not resize the semaphore. **size** must be greater than or equal to 1. The default is 1.

create if not found specifies whether you want to create a new semaphore if one with that name does not exist. The default is TRUE, which specifies that LabVIEW creates a semaphore if the semaphore does not exist. If **create if not found** is FALSE and LabVIEW cannot find a semaphore with the name you specify, LabVIEW returns error code 1534.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

semaphore is a reference to an existing or newly created semaphore. If you use this VI to obtain multiple references to the same named semaphore, each reference number is unique.

created new is TRUE if the VI created a new semaphore.

error out contains error information. This output provides standard error out functionality.

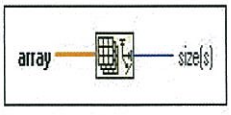
Array Size Function

Owning Palette: Array Functions

Requires: Base Development System

Returns the number of elements in each dimension of **array**.

The connector pane displays the default data types for this polymorphic function.



Add to the block diagram Find on the palette

array can be an *n*-dimensional array of any type.

size(s) is a 32-bit integer if **array** is one-dimensional (1D). If **array** is multidimensional, the returned value is a 1D array in which each element is a 32-bit integer that represents the number of elements in the corresponding dimension of **array**. The order of elements in the return array corresponds to row-major order. Thus, vol is the first index, followed by page, row, and column. These names are index identifiers and have no other meaning.

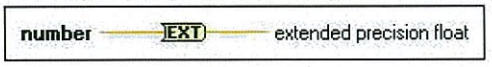
To Extended Precision Float Function

Owning Palette: Conversion VIs and Functions

Requires: Base Development System

Converts a number to an extended-precision, floating-point number.

The connector pane displays the default data types for this polymorphic function.



Add to the block diagram Find on the palette

number can be a scalar number, array or cluster of numbers, array of clusters of numbers, and so on.

extended precision float is of the same data type structure as **number**.

Open Application Reference Function

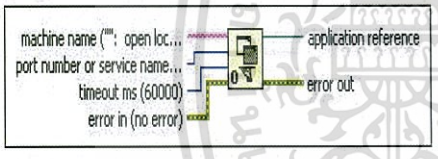
Owning Palette: Application Control VIs and Functions

Requires: Base Development System

Returns a reference to a VI Server application running on the specified computer.

If you specify an empty string for **machine name**, it returns a reference to the local LabVIEW application in which this function is running. If you do specify a **machine name**, it attempts to establish a TCP connection with a remote VI Server on that machine on the specified port.

Details



Add to the block diagram Find on the palette

machine name is the address of the computer that runs an application instance to which you want to establish a connection. This address can be in dotted decimal notation (such as `130.164.15.250`) or domain name notation (such as `foo.ni.com`). An empty string causes this function to return a reference to the local application instance.

port number or service name can accept a numeric or a string input. The default is a numeric. **port number or service name** is the port on which the remote LabVIEW application is listening. If you specify a service name, LabVIEW queries the NI Service Locator for the port number that the server registered. The default is to use the default VI Server listener port number (3363).

To establish communication between a VI and another LabVIEW application, you must know both the **machine name** and the **port number or service name** on which the VI Server in the other LabVIEW application is listening. If you have more than one LabVIEW application on the same machine, one or more of those applications may be listening on a port other than the default VI Server listener port. In this case, make sure to supply the **port number or service name**. Use the VI Server page to set the port number and service name for a VI Server or use the Server:Port and Server:Service Name properties to set the port number or service name programmatically.

Note The VI Server settings in the Options dialog box apply to the default application instance, or VIs not in a project. To set VI Server settings for a project application instance, right-click the target in the Project Explorer window.

timeout ms specifies the time, in milliseconds, that the function waits to complete and return an error. The default value is 60,000 ms or 1 minute. A value of -1 indicates to wait indefinitely.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

application reference is the reference to the specified application.

error out contains error information. This output provides standard error out functionality.

String Length Function

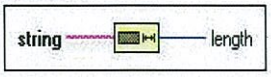
Owning Palette: String Functions

Requires: Base Development System

Returns in **length** the number of characters (bytes) in **string**.

The connector pane displays the default data types for this polymorphic function.

Example



Add to the block diagram Find on the palette

---

string can be a string or any data structure that contains only strings, such as an array or clusters of strings.

length has the same structure as **string**.

Example

Refer to the String Length VI in the `labview\examples\Strings` directory for an example of using the String Length function.

Open example Find related examples

Concatenate Strings Function

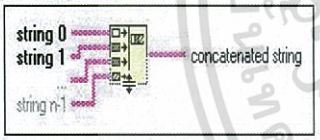
Owning Palette: String Functions

Requires: Base Development System

Concatenates input strings and 1D arrays of strings into a single output string. For array inputs, this function concatenates each element of the array.

Add inputs to the function by right-clicking an input and selecting **Add Input** from the shortcut menu or by resizing the function.

Details Example



Add to the block diagram Find on the palette

---

string 0..n-1 are the strings you want to concatenate.

concatenated string contains the concatenated input strings in the order you wire them to the node from top to bottom.

Concatenate Strings Details

You can use this function to concatenate the output from Picture Functions VIs so that they draw on a single picture control. The pictures are drawn in order from top to bottom.

Example

Refer to the Concatenate Strings VI in the `labview\examples\Strings` directory for an example of using the Concatenate Strings function.

Open example Find related examples

Stop Function

Owning Palette: Application Control VIs and Functions

Requires: Base Development System

Stops the VI in which it executes, just as if you clicked the **Abort Execution** button on the toolbar. Before you call this function with a TRUE input, be sure to complete all final tasks for the VI first, such as closing files, setting safe values for devices being controlled, and so on.

If you wired the input, stop occurs only if the input value is TRUE. The default is to stop as soon as the node that is currently executing finishes.

Details

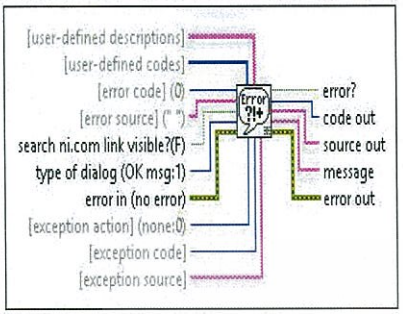
Add to the block diagram Find on the palette

General Error Handler VI

Owning Palette: Dialog & User Interface VIs and Functions

Requires: Base Development System

Indicates whether an error occurred. If an error occurred, this VI returns a description of the error and optionally displays a dialog box.



Add to the block diagram Find on the palette

[user-defined descriptions] is an array of descriptions of user-defined codes. If an incoming error matches one in **user-defined codes**, the VI returns the corresponding description from **user-defined descriptions** in **message**.

[user-defined codes] is an array of numeric error codes you can use to define error codes and messages for your own VIs. The VI searches this array after searching an internal database of error codes. Error codes -8999 through -8000, 5000 through 9999, and 500,000 through 599,999 are reserved for you to define your own error messages.

[error code] is a numeric error code. If **error in** indicates an error, the VI ignores **error code**. If not, the VI tests it. A nonzero value signifies an error.

[error source] is an optional string you can use to describe the source of **error code**.

search ni.com link visible? determines whether the `Search ni.com for error` hyperlink appears in the dialog box. Set to TRUE to display the hyperlink. Clicking the hyperlink opens search results for the error code on `ni.com` in the default web browser. The hyperlink appears only for LabVIEW-defined error codes in the development environment. The default is TRUE.

type of dialog determines what type of dialog box to display, if any. Regardless of its value, the VI outputs the error information and **message** describing the error.

0	**no dialog**—Displays no dialog box. This is useful if you want to have programmatic control over handling errors.
1	**OK message** (default)—Displays a dialog box with a single OK button. After the user acknowledges the dialog box, the VI returns control to the main VI.
2	**continue or stop message**—Displays a dialog box with buttons, which the user can use to either continue or stop. If the user selects Stop, the VI calls the Stop function to halt execution.
3	**OK message + warnings**—Displays a dialog box with any warnings and a single OK button. After the user acknowledges the dialog box, the VI returns control to the main VI.
4	**continue/stop + warnings**—Displays a dialog box with any warnings and buttons, which the user can use to either continue or stop. If the user selects Stop, the VI calls the Stop function to halt execution.

error in describes error conditions that occur before this node runs.

This input contains **status**, **code**, and **source**, which provide standard error in cluster element functionality.

[exception action] is a way for you to create exceptions to error handling. The VI performs the **exception action** if the **error code** and **error source** match the **exception code** and **exception source**. If you use the default value for an **exception source**, only the **exception code** must match for the VI to perform the **exception action**.

0	**No exception** (default)—Performs no error exception handling, even if you wire an **exception code** or an **exception source**.
1	**Cancel error on match**—Treats what is normally an error as no error. If the VI cancels an error, **error?** is FALSE, **code out** is 0, and **source out** is an empty string.
2	**Set error on match**—Upgrades a warning to an error. This parameter sets an error if the VI detects no error, as described in the **status** and **error code** parameters, but the **code** value of **error in** matches **exception code** and the error source value matches **exception source**. If the VI sets an error, **error?** is TRUE, **code out** is the **code** value from **error in**, and **source out** is the **source** value from **error in**.

[exception code] is the error code that you want to treat as an exception. The default is 0.

[exception source] is the error message that you want to use to test for an exception. The default is an empty string.

error? indicates whether an error occurred. If this VI finds an error, it sets the parameters in the error cluster.

code out is the error code indicated by **error in** or **error code**.

source out indicates the source of the error. The **source out** string is a more descriptive string than the **source** string in the **error in** input.

message describes the error code that occurred, the source of the error, and a description of the error. If the VI does not return a description of the error, you can take several actions to find the error code description. If more than one description exists for the same error code, the VI displays all the descriptions, separated by `or`.

error out contains error information. This output provides standard error out functionality.

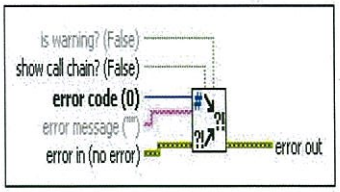
error out contains error information. This output provides standard error out functionality. **error out** returns `no error` by default. If you wire a value to **specific error code to clear** and **error in** contains errors that do not match the error code you wired, **error out** will return an error.

Error Cluster From Error Code VI

Owning Palette: Dialog & User Interface VIs and Functions

Requires: Base Development System

Converts an error or warning code to an error cluster. This VI is useful when you receive a return value from a shared library call or when you return user-defined error codes.



Add to the block diagram Find on the palette

If **is warning?** is TRUE, **status** returns FALSE to indicate that a warning occurred. The default is FALSE.

If **show call chain?** is TRUE, **source** includes the chain of callers from the VI that produced the error or warning to the top-level VI. The default is FALSE, which indicates to include only the calling VI. This VI uses the Call Chain function to obtain the chain of callers.

error code is the code you want to convert to an error cluster. The default is 0, which indicates that no error occurred.

error message is the error description to appear in the **error out** cluster. LabVIEW displays the description in **error message** when the General Error Handler VI receives this **error out** cluster. If **error message** is empty, LabVIEW uses **error code** to determine the error description from existing LabVIEW error codes.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

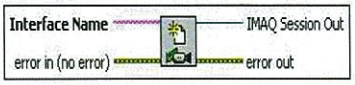
error out contains error information. With the following exception, this output provides standard error out functionality.

If the value of **error in** is `no error`, this VI contains error information that corresponds to the **error code** input.

IMAQ Init VI

Owning Palette: NI-IMAQ
Installed With: NI Vision Acquisition Software

Loads an NI-IMAQ configuration file and configures the image acquisition device.



Interface Name is the name of the interface to be loaded. The name must match the configuration file name used in Measurement & Automation Explorer (MAX).

Note **Interface Name** always identifies a single port of an image acquisition device. A port identifies a single independent data stream from a camera. All NI image acquisition devices support at least one port. Devices that support multiple ports can sustain independent and asynchronous acquisitions from the cameras on each port.

The port number may be explicitly identified by using the :: operator to append the port number suffix to the interface name. Port numbers are zero-based. For example, img0::1 opens port number 1 of the image acquisition device identified by img0. Interface names that do not have a port number suffix default to port 0. img0::0 and img0 are equivalent in meaning.

error in (no error) describes error conditions that occur before this VI or function runs. The default is `no error`. If an error occurred before this VI or function runs, the VI or function passes the **error in** value to **error out**. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in **error out**. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use **error in** and **error out** to check errors and to specify execution order by wiring **error out** from one node to **error in** of the next node.

- **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
- **code** code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning. The default is an empty string.

IMAQ Session Out identifies the initialized device.

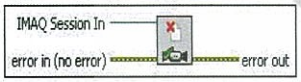
error out contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** indicator on the front panel and select **Explain Error** from the shortcut menu for more information about the error.

- **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
- **code** is the error or warning code. If status is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning. The default is an empty string.

IMAQ Close VI

Owning Palette: NI-IMAQ
Installed With: NI Vision Acquisition Software

Stops the acquisition if one is in progress, releases resources associated with the acquisition, and closes the specified IMAQ session.



IMAQ Session In identifies the device.

error in (no error) describes error conditions that occur before this VI or function runs. The default is `no error`. If an error occurred before this VI or function runs, the VI or function passes the **error in** value to **error out**. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in **error out**. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use **error in** and **error out** to check errors and to specify execution order by wiring **error out** from one node to **error in** of the next node.

- **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
- **code** code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning. The default is an empty string.

error out contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** indicator on the front panel and select **Explain Error** from the shortcut menu for more information about the error.

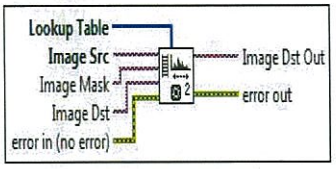
- **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
- **code** is the error or warning code. If status is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning. The default is an empty string.

IMAQ UserLookup 2 VI

Owning Palette: Processing
Requires: NI Vision Development Module

Performs a user-specified lookup-table transformation by remapping the pixel values in an image.

U8 U16 I16



Lookup Table is a grayscale replacement table. This input is an array containing a maximum of 256 elements if **Image Src** is an 8-bit image or a maximum of 65,536 elements if **Image Src** is a 16-bit image. Individual pixels within the image are not modified when the lookup table is missing a value that corresponds to those pixels.

Image Src is a reference to the source image.

Image Mask is an 8-bit image that specifies the region of the small image that will be copied. Only pixels in the **Image Src (Small)** image that correspond to a non-zero pixel in the mask image are copied. All other pixels keep their original values. The entire image is processed if **Image Mask** is not connected.

Image Dst is a reference to the destination image.

error in (no error) describes the error status before this VI or function runs. The default is `no error`. If an error occurred before this VI or function runs, the VI or function passes the **error in** value to **error out**. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use **error in** and **error out** to check errors and to specify execution order by wiring error out from one node to error in of the next node.

- **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning. The default is an empty string.

Image Dst Out is a reference to the destination image. If **Image Dst** is connected, **Image Dst Out** is the same as **Image Dst**. Otherwise, **Image Dst Out** refers to the image referenced by **Image Src**.

error out contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** indicator on the front panel and select **Explain Error** from the shortcut menu for more information about the error.

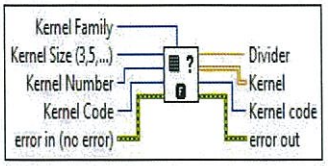
- **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning. The default is an empty string.

IMAQ GetKernel VI

Owning Palette: Filters
Requires: NI Vision Development Module

Reads a predefined kernel. This code consists of three separate units: **Kernel Family**, **Kernel Size**, and **Kernel Number**. If you already know the code, you can enter it directly with **Kernel Code**.

Details Examples



Kernel Family determines the type of matrix. This value corresponds to the thousandth unit in the researched code. The matrix types are as follows:

Gradient (1)	Specifies the kernel family as gradient
Laplacian (2)	Specifies the kernel family as Laplacian
Smoothing (3)	Specifies the kernel family as smoothing
Gaussian (4)	Specifies the kernel family as Gaussian

Kernel Size (3,5,...) determines the horizontal and vertical matrix size. The values are 3, 5, and 7, corresponding to the convolutions 3 × 3, 5 × 5, and 7 × 7 supplied in the matrix catalog. This value corresponds to the hundredth unit in the researched code.

Kernel Number is the matrix family number. It is a two-digit number, between 0 and *n*, belonging to a family and a size. A number of predefined matrices are available for each type and size.

Kernel Code is a code you can use to directly access a convolution matrix. Each code specifies a specific convolution matrix. You can use this input if it is connected and is not 0. The kernel located in the file then is transcribed into a 2D array that is available from the output **Kernel**. You can use the codes to specify a predefined kernel.

error in (no error) describes the error status before this VI or function runs. The default is `no error`. If an error occurred before this VI or function runs, the VI or function passes the **error in** value to **error out**. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use **error in** and **error out** to check errors and to specify execution order by wiring error out from one node to error in of the next node.

- **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning. The default is an empty string.

Divider is the normalization factor associated with the retrieved kernel.

Kernel is the resulting matrix. It corresponds to a kernel encoded by a code specified from the inputs **Kernel Family**, **Kernel Size**, and **Kernel Number** or a from a code directly passed through the input **Kernel Code**. You can connect this output directly to the input **Kernel** in the IMAQ Convolute VI.

Kernel code indicates the code that was used to retrieve the kernel.

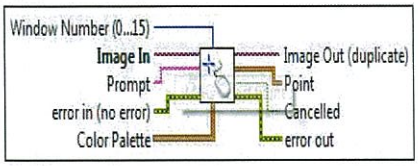
error out contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** indicator on the front panel and select **Explain Error** from the shortcut menu for more information about the error.

- **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning. The default is an empty string.

IMAQ Select Point VI

Owning Palette: Select Region of Interest
Requires: NI Vision Development Module

Allows the user to specify the position of a point in the image. IMAQ Select Point displays the image in the specified window and provides a point tool. IMAQ Select Point returns the coordinates of the point selected when the user clicks **OK** in the window.



Window Number (0...15) specifies the window in which to display the image. When you select the default value of ?1, this VI uses a modal dialog window, centered in the screen. When you select a regular NI Vision window number (0?15), the VI displays **Image In** in the specified window and temporarily sets the NI Vision window to modal mode. When the user clicks **OK** or **Cancel** in the window, the attributes of the window are set back to their initial values.

Image In is a reference to the image on which the user selects the **Point**.

Prompt specifies a message string to display in the title bar of the window. Use this control to provide the user with instructions about selecting the object.

error in (no error) describes the error status before this VI or function runs. The default is `no error`. If an error occurred before this VI or function runs, the VI or function passes the **error in** value to **error out**. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs to display the description of the error code. Use **error in** and **error out** to check errors and to specify execution order by wiring error out from one node to error in of the next node.

- **status** is TRUE (X) if an error occurred before this VI or function ran or FALSE (checkmark) to indicate a warning or that no error occurred before this VI or function ran. The default is FALSE.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning. The default is an empty string.

Color Palette is used to apply a color palette to the image window. **Color Palette** is an array of clusters constructed by the user or supplied by the IMAQ GetPalette VI. This palette is composed of 256 elements for each of the three color planes (red, green, and blue). A specific color is the result of applying a value between 0 and 255 to each of the three color planes. If the three planes have identical values, a gray level is obtained (0 specifies black and 255 specifies white). If the image type requires a color palette and it is not supplied, a grayscale color palette is generated and written.

Tip For best results, set your video adapter to high color or true color.

- **Red** is the value of the red color plane.
- **Green** is the value of the green color plane.
- **Blue** is the value of the blue color plane.

Image Out (duplicate) is a reference to **Image In**. This VI does not modify the image connected to the **Image In** input.

Point is a cluster that specifies the coordinates of the point chosen by the user.

- **X** is the x-coordinate of the point.
- **Y** is the y-coordinate of the point.

Cancelled returns TRUE if the user ends the selection by clicking **Cancel** in the window.

error out contains error information. If **error in** indicates that an error occurred before this VI or function ran, **error out** contains the same error information. Otherwise, it describes the error status that this VI or function produces. Right-click the **error out** indicator on the front panel and select **Explain Error** from the shortcut menu for more information about the error.

- **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.
- **code** is the error or warning code. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** is 0 or a warning code.
- **source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning. The default is an empty string.

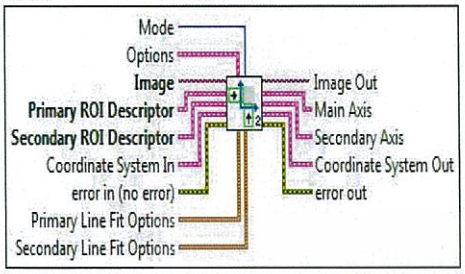
IMAQ Find CoordSys (2 Rects) 2 VI

Owning Palette: Coordinate System
Requires: NI Vision Development Module

Computes a coordinate system based on the position of an object in two search areas of an image. The location and orientation of the coordinate system found is used to create the reference position of a Coordinate System or to update the current location and orientation of an existing Coordinate System. The function has the capability to overlay the following on the returned image: the position of the search area, the search lines, the edges found, and the location and orientation of the Coordinate System found.

Details Examples

U8 U16 RGB U32 I16 RGB U64 SGL HSL U32



U16 **Mode** specifies the function that is performed by this VI. You can choose from the following values:

Find Reference (0)	Specifies that the location and orientation of the found edges set the reference system of **Coordinate System Out**. This is typically the first mode to use when setting up a coordinate system.
Update CoordSys (1)	Specifies that the location and orientation of the found edges set the measurement system of **Coordinate System Out**. The reference system of **Coordinate System In** remains unchanged in **Coordinate System Out**. Use this mode with each new inspection image to update the location and orientation of the edges defining the coordinate system.

Options is a cluster defining the parameters of the edge detection algorithm and the information that is overlaid on the result image.

U16 **Direction** specifies the order and direction in which the edges are searched. The arrow in the **Direction** symbol specifies the search direction for the **Main Axis**, while the perpendicular sign determines the position of the perpendicular axis.

Value	Main Axis	Secondary Axis
0	Horizontal – Left to Right	Bottom to Top
1	Horizontal – Left to Right	Top to Bottom
2	Vertical – Top to Bottom	Left to Right
3	Vertical – Top to Bottom	Right to Left
4	Horizontal – Right to Left	Top to Bottom
5	Horizontal – Right to Left	Bottom to Top
6	Vertical – Bottom to Top	Right to Left
7	Vertical – Bottom to Top	Left to Right

Primary Edge Options specifies the parameters that are used to compute the edge gradient information and detect the edges along the Primary ROI Descriptor.

U32 **Edge Polarity** specifies the polarity of the edges to be found.

All Edges (0)	(Default) Searches for all edges
Rising Edges (1)	Searches for rising edges
Falling Edges (2)	Searches for falling edges

U32 **Kernel Size** specifies the size of the edge detection kernel. The default is 3.

U32 **Width** specifies the number of pixels averaged perpendicular to the search direction to compute the edge profile strength at each point along the search ROI. The default is 3.

SGL **Minimum Edge Strength** specifies the minimum edge strength (gradient magnitude) required for a detected edge. The default is 10.

Interpolation Type specifies the interpolation method used to locate the edge position.

Choose from the following options:

Zero Order (0)	Rounds to the nearest integral edge location
Bilinear (1)	Uses bilinear interpolation to compute the edge location
Bilinear Fixed (4)	(Default) Uses the fixed-point computation of bilinear interpolation to determine the edge location

Data Processing Method is the method used to process the data extracted for edge detection.

Average (0)	(Default) Averages the data extracted for edge detection
Median (1)	Takes the median of the data extracted for edge detection

Secondary Edge Options specifies the parameters that are used to compute the edge gradient information and detect the edges along the Secondary ROI Descriptor.

U32 **Edge Polarity** specifies the polarity of the edges to be found.

All Edges (0)	(Default) Searches for all edges
Rising Edges (1)	Searches for rising edges
Falling Edges (2)	Searches for falling edges

Kernel Size specifies the size of the edge detection kernel. The default is 3.

Width specifies the number of pixels averaged perpendicular to the search direction to compute the edge profile strength at each point along the search ROI. The default is 3.

Minimum Edge Strength specifies the minimum edge strength (gradient magnitude) required for a detected edge. The default is 10.

Interpolation Type specifies the interpolation method used to locate the edge position.

Choose from the following options:

Zero Order (0)	Rounds to the nearest integral edge location
Bilinear (1)	Uses bilinear interpolation to compute the edge location
Bilinear Fixed (4)	(Default) Uses the fixed-point computation of bilinear interpolation to determine the edge location

Data Processing Method is the method used to process the data extracted for edge detection.

Average (0)	(Default) Averages the data extracted for edge detection
Median (1)	Takes the median of the data extracted for edge detection

Secondary Edge Options specifies the parameters that are used to compute the edge gradient information and detect the edges along the Secondary ROI Descriptor.

Edge Polarity specifies the polarity of the edges to be found.

All Edges (0)	(Default) Searches for all edges
Rising Edges (1)	Searches for rising edges
Falling Edges (2)	Searches for falling edges

Kernel Size specifies the size of the edge detection kernel. The default is 3.

Width specifies the number of pixels averaged perpendicular to the search direction to compute the edge profile strength at each point along the search ROI. The default is 3.

Minimum Edge Strength specifies the minimum edge strength (gradient magnitude) required for a detected edge. The default is 10.

Interpolation Type specifies the interpolation method used to locate the edge position.

Choose from the following options:

Zero Order (0)	Rounds to the nearest integral edge location
Bilinear (1)	Uses bilinear interpolation to compute the edge location
Bilinear Fixed (4)	(Default) Uses the fixed-point computation of bilinear interpolation to determine the edge location

Data Processing Method is the method used to process the data extracted for edge detection.

Average (0)	(Default) Averages the data extracted for edge detection
Median (1)	Takes the median of the data extracted for edge detection

Show Search Area determines whether to overlay the ROI on the image.

Show Search Lines determines whether the search lines used to locate the edges are overlaid on the image.

Show Edges Found determines whether the locations of the edges found are overlaid on the result image.

Show Result determines whether the coordinate system found is overlaid on the result image.

Search Area Color specifies the color to use to overlay the search area.

Search Lines Color specifies the color to use to overlay the search lines.

Edge Locations Color specifies the color to use to overlay the edge locations.

Result Color specifies the color to use to overlay the result.

Overlay Group Name specifies the overlay group name for the step overlays.

Image is a reference to the image in which the coordinate system is to be located.

Primary ROI Descriptor is a descriptor that defines the Region of Interest (ROI) within which the edge detection is performed.

Global Rectangle contains the coordinates of the bounding rectangle.

Contours are each of the individual shapes that define an ROI.

ID refers to whether the contour is the external or internal edge of an ROI.

Type is the shape type of the contour.

Coordinates indicates the relative position of the contour.

Secondary ROI Descriptor is a descriptor that defines the Region of Interest (ROI) within which the edge detection is performed.

Global Rectangle contains the coordinates of the bounding rectangle.

Contours are each of the individual shapes that define an ROI.

ID refers to whether the contour is the external or internal edge of an ROI.