Steelcase Panel Quality Assurance

User's Manual

Modified Date: 04/29/2016

Document Revision: 1.0

Created by UAH Senior Design Team StPaQuAs

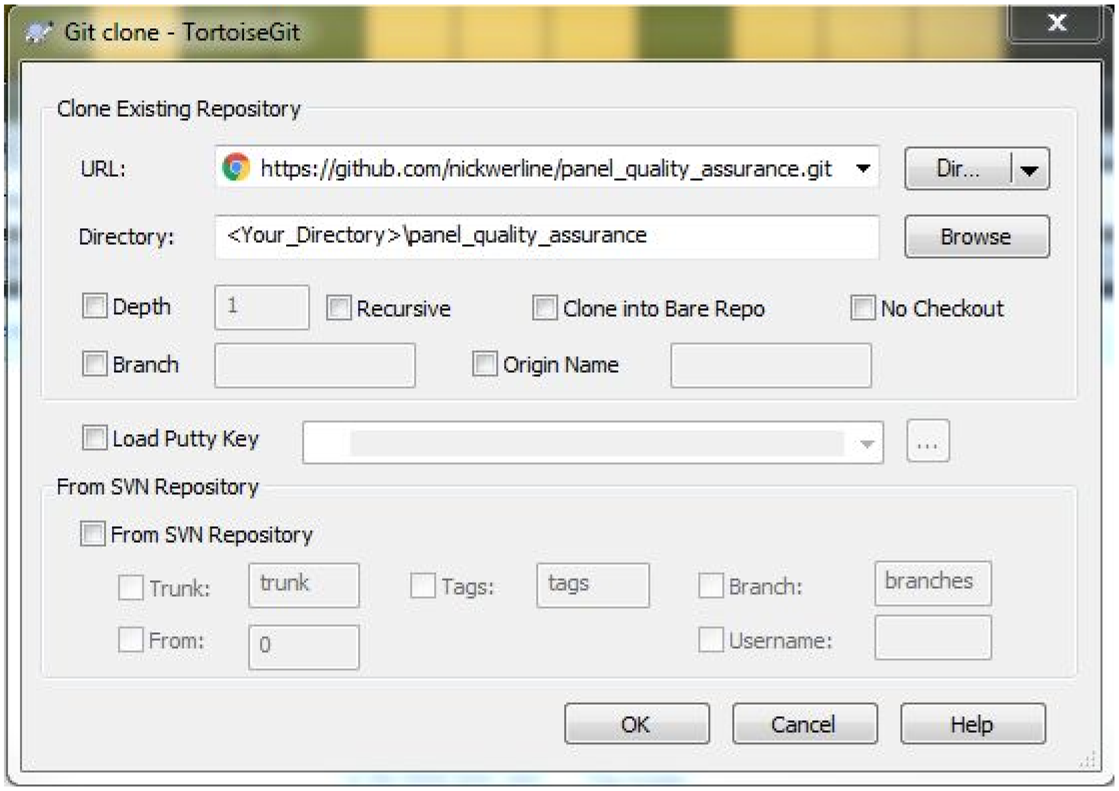
Table of Contents

Installation

Cloning the Repository

Repository Location: **https://github.com/nickwerline/panel_quality_assurance**
You may clone our repository using the git client of your choice. We recommend TortoiseGit (https://tortoisegit.org/download/). If you are using TortoiseGit, navigate to the folder where you want the repository, right click and select "Git Clone…"

From here you will enter the URL of the repository as shown:


Git clone - TortoiseGit

Clone Existing Repository

URL: https://github.com/nickwerline/panel_quality_assurance.git

Directory: <Your_Directory>\panel_quality_assurance

Depth 1 | Recursive | Clone into Bare Repo | No Checkout

Branch | Origin Name

Load Putty Key

From SVN Repository

From SVN Repository

Trunk: trunk | Tags: tags | Branch: branches

From: 0 | Username:

Select 'OK' and when finished you will have a clone of our repository.

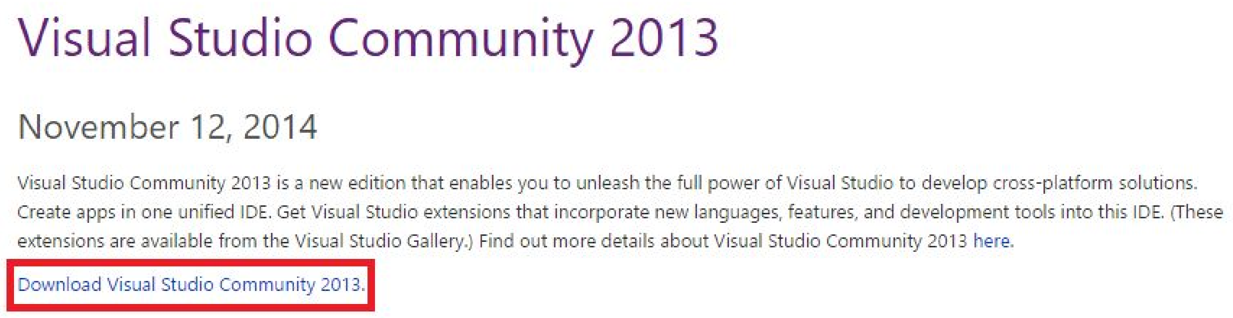
OpenCV Setup

All of the Visual Studio project settings to run our application with OpenCV are already set up in our repository. All you need to do in order to make OpenCV work is set the environment variable OPENCV_DIR=<path_to_repository_clone>/opencv/build/x86/vc12. If a new project is set up to build an application with OpenCV, either copy the project settings from our repository or follow this guilde:

http://docs.opencv.org/3.0-beta/doc/tutorials/introduction/windows_visual_studio_Opencv/windows_visual_studio_Opencv.html#windows-visual-studio-how-to

Installing Visual Studio 2013

In order to build our application, you must have Visual Studio 13. Download Visual Studio 2013 Community from here: https://www.visualstudio.com/en-us/news/vs2013-community-vs.aspx



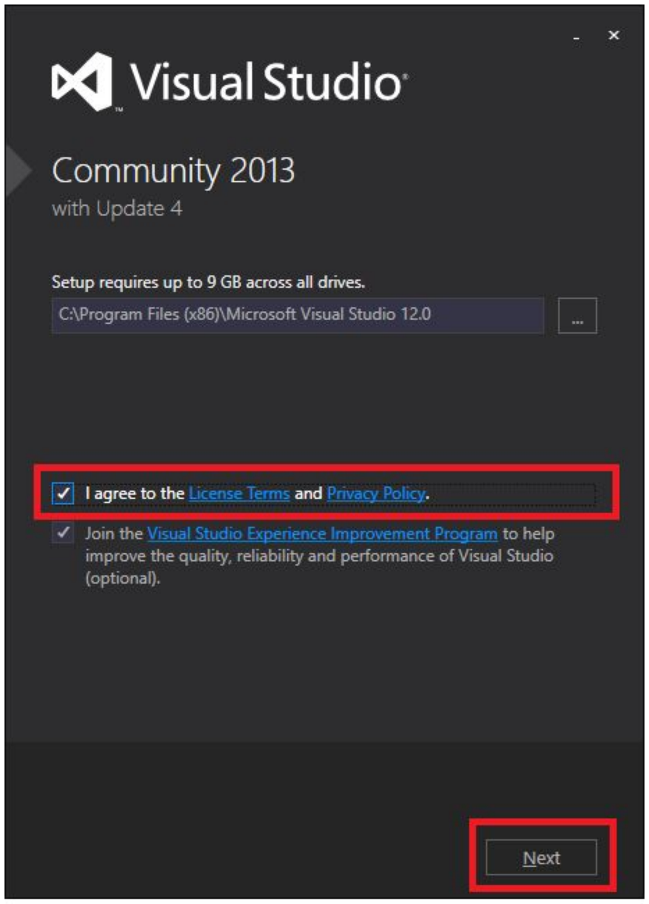
Visual Studio Community 2013

November 12, 2014

Visual Studio Community 2013 is a new edition that enables you to unleash the full power of Visual Studio to develop cross-platform solutions. Create apps in one unified IDE. Get Visual Studio extensions that incorporate new languages, features, and development tools into this IDE. (These extensions are available from the Visual Studio Gallery.) Find out more details about Visual Studio Community 2013 here.

Download Visual Studio Community 2013.

Run the .exe file to install Visual Studio.



Visual Studio

Community 2013
with Update 4

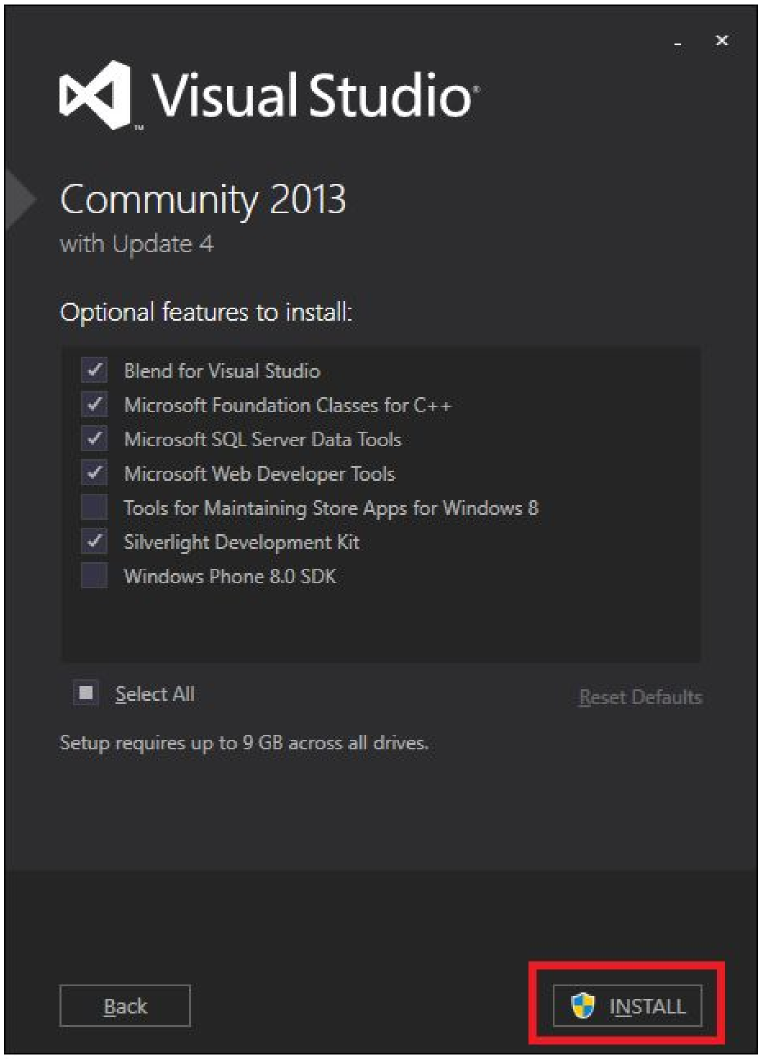
Setup requires up to 9 GB across all drives.

C:\Program Files (x86)\Microsoft Visual Studio 12.0

[✓] I agree to the License Terms and Privacy Policy.

[✓] Join the Visual Studio Experience Improvement Program to help improve the quality, reliability and performance of Visual Studio (optional).

Agree to the License Terms and Privacy Policy then click “Next”.


Visual Studio

Community 2013
with Update 4

Optional features to install:

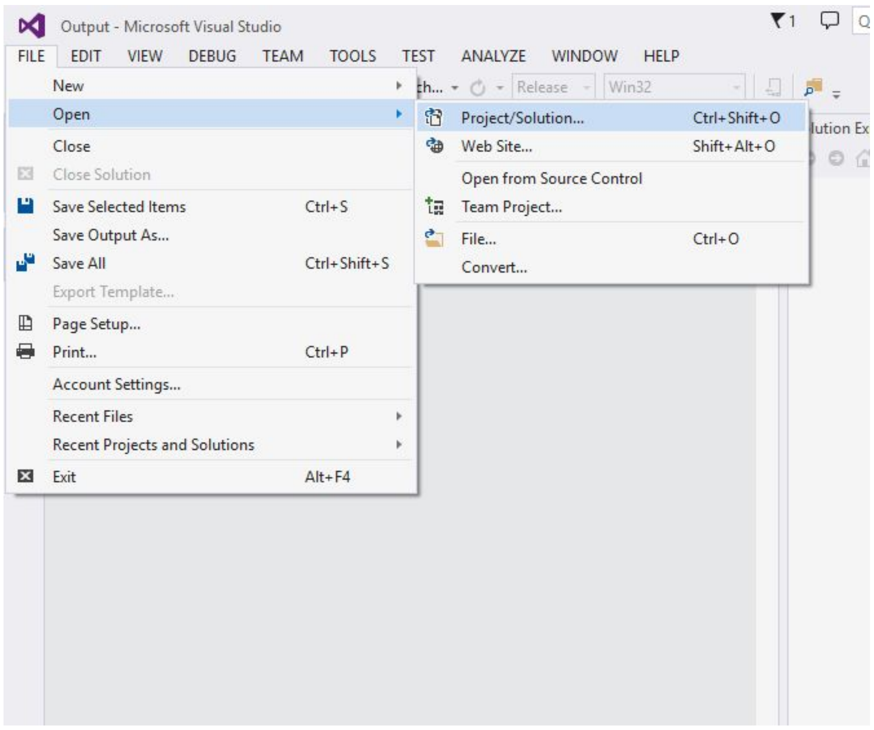
- [✓] Blend for Visual Studio
- [✓] Microsoft Foundation Classes for C++
- [✓] Microsoft SQL Server Data Tools
- [✓] Microsoft Web Developer Tools
- [ ] Tools for Maintaining Store Apps for Windows 8
- [✓] Silverlight Development Kit
- [ ] Windows Phone 8.0 SDK

Select All

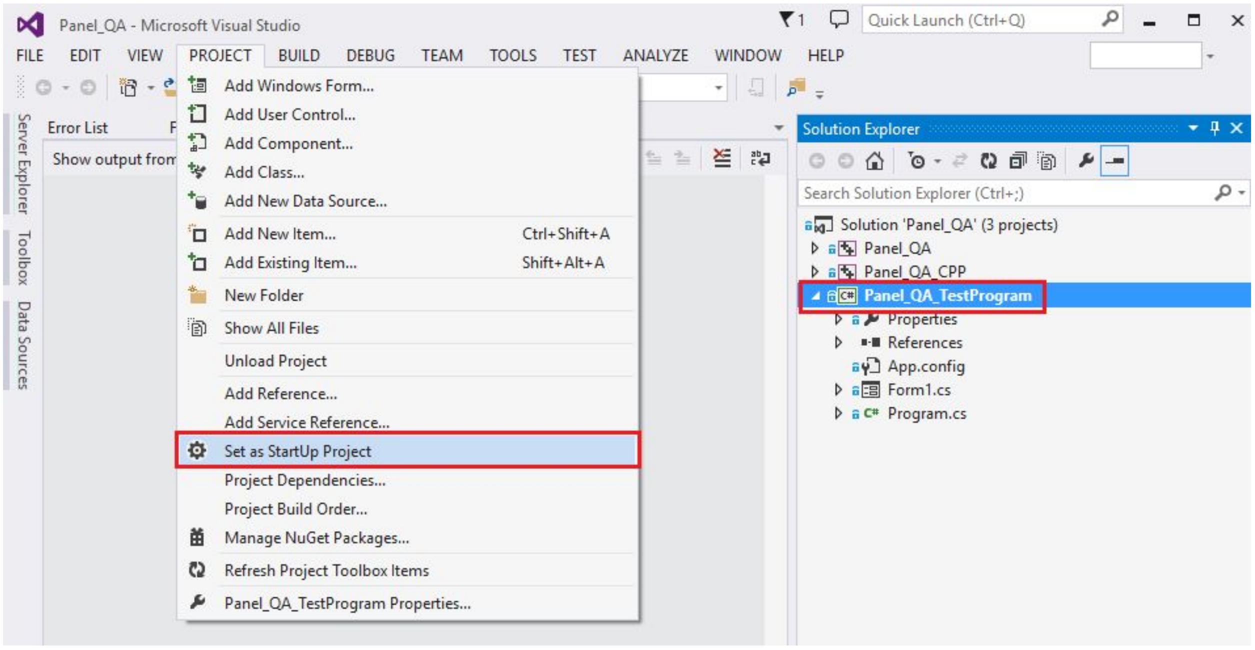
Setup requires up to 9 GB across all drives.

Click “Install”. The installation will take a while. When it is finished, you will be prompted to restart your computer.

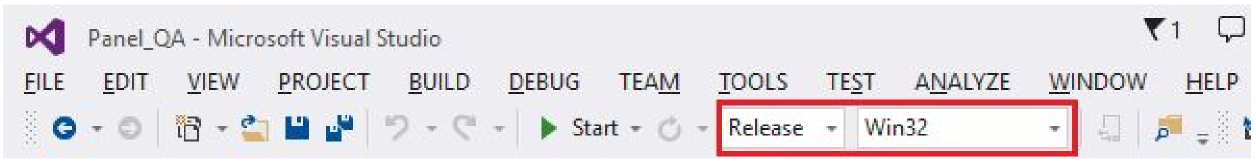
Building the Application



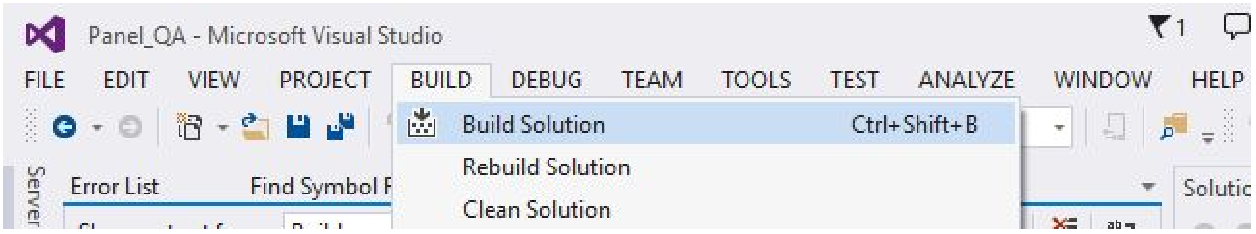
Open Visual Studio, then click “File ->Open->Project/Solution and navigate to the project folder. The solution is located at "..\panel_quality_assurance\Panel_QA\Panel_QA.sln"



Select the “Panel_QA_TestProgram” project. Then click “Project->Set as StartUp Project”


Panel_QA - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Start
Release
Win32

Select “Release” and “Win32” in the drop-down menus.


Panel_QA - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Build Solution Ctrl+Shift+B
Rebuild Solution
Clean Solution
Error List
Find Symbol R
Server
Solutio

Select “Build Solution” under “Build”. The executable will be created and is located at "..\panel_quality_assurance\Panel_QA\Panel_QA_TestProgram\bin\Release\Panel_QA_TestProgram.exe". You can double click this executable to run it.

Accessory Tag Detection

StPaQuAs’ Accessory Tag Detection reads in an image and determines whether or not a tag of the specified color is present on the panel in the image. The currently supported colors are red and blue.

Running Tag Detection

In order to use the tag detection functionality the user must simply enter the path to the image on the Main tab and select the “Detect Tag” button while the desired color is selected.

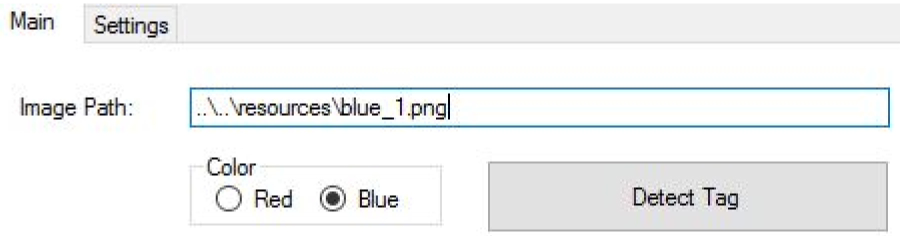
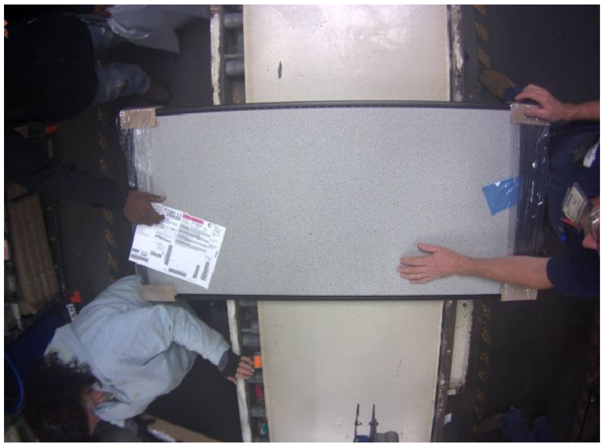
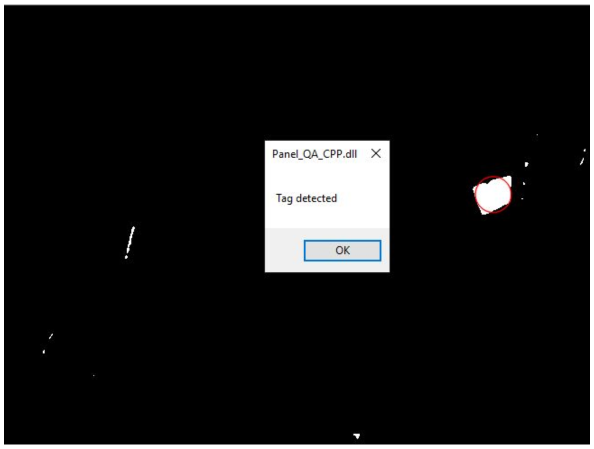


Image Path: ..\..\resources\blue_1.png

Color: Red [✓ Blue]

Here is the input image and the result of “Blue” tag detection:




Panel_QA_CPP.dll

Tag detected

Debugging Tag Detection Parameters

In order to debug the parameters associated with tag detection, the user may enter the image path and use the “Red”, “Blue”, and “Debug” buttons on the Settings tab.

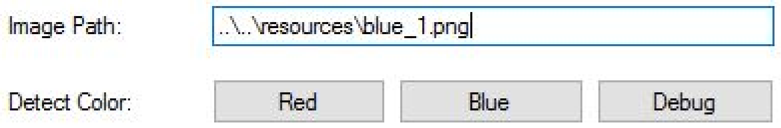
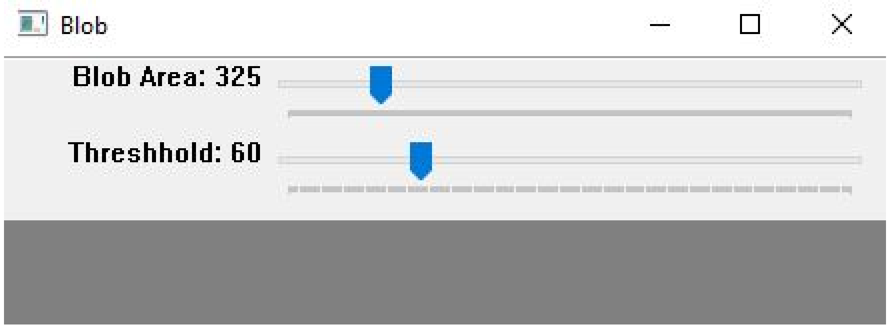


Image Path: ..\..\resources\blue_1.png

The “Red” and “Blue” buttons have basically the same functionality except for different colors. When one of these buttons are selected four dialogs are created: Original Image, Threshold, Keypoints, and a sliders dialog. The sliders dialog contains two sliders: Blob Area and Threshold.

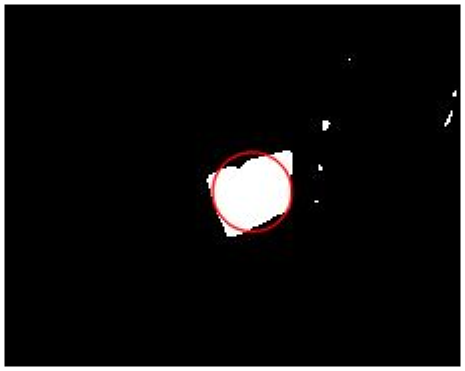

Blob

Blob Area: 325

Threshhold: 60

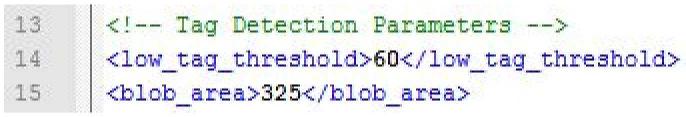
The initial state of the sliders is based on the current settings of the variables associated with them. As these sliders are adjusted, the Threshold and Keypoints images are dynamically updated using the current settings of the sliders.

Threshold is the binary threshold that is applied after the tag color is isolated. This threshold will eliminate low intensity colors from the image. The lower the threshold, the lower the intensity of the color is allowed to be. Keypoints is the same image after dilation is applied. Dilation is basically filling in small holes and smoothing rough edges. This ensures that a full blob is formed for the blob detection stage. The Keypoints image also displays whether or not a blob greater than or equal to the minimum blob area. This setting can be adjusted using the "Blob Area" slider. If a blob is big enough it will be circled, meaning that a tag will be detected. When the user is ready to stop debugging, they must hit escape before they are able to close the dialog.



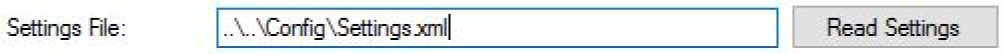
Updating the Settings File

Once the desired settings are found, the user must enter the settings into Settings.xml located in the "Config" directory. This is where all of the settings for Canny edge detection, Hough line detection, tag detection, and feature detection are set. The user will enter the "Blob Area" setting in the blob_area field and the threshold value into the low_tag_threshold field.



```
<!-- Tag Detection Parameters -->
<low_tag_threshold>60</low_tag_threshold>
<blob_area>325</blob_area>
```

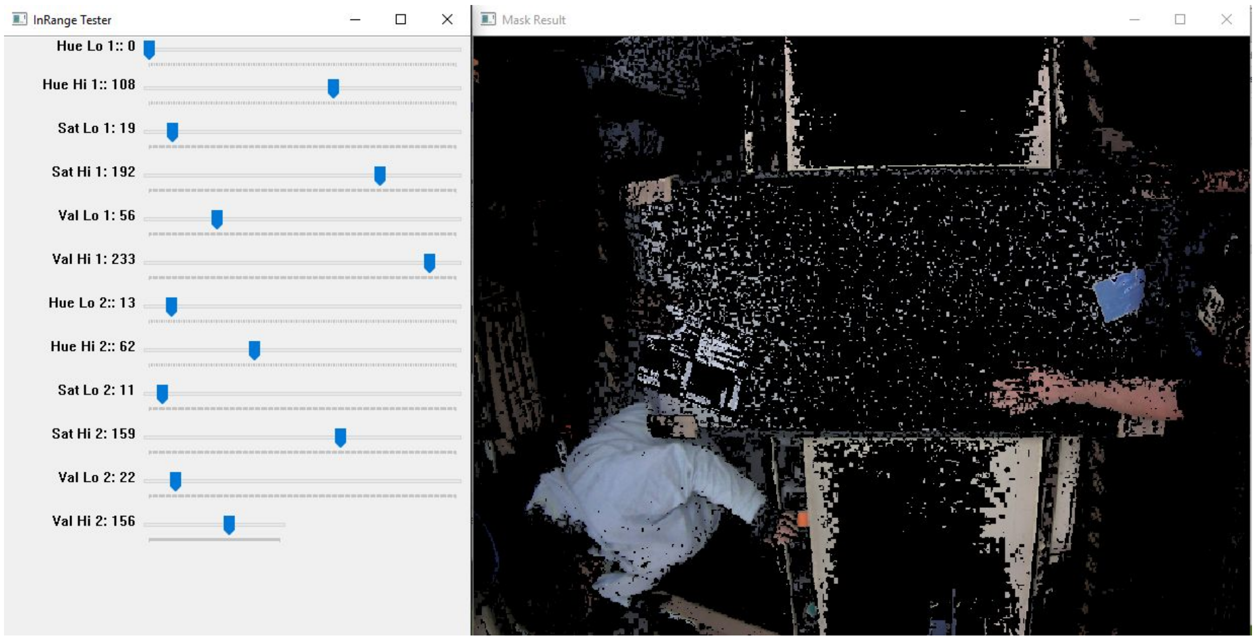
These settings can be loaded into the application by entering the path to the settings file in the "Settings File" field on the settings tab of the dialog and selecting the "Read Settings" button. Settings.xml is automatically loaded when the application is started, but it can be updated this way while the application is running.



Settings File: ..\..\Config\Settings.xml

Debugging Tag Colors

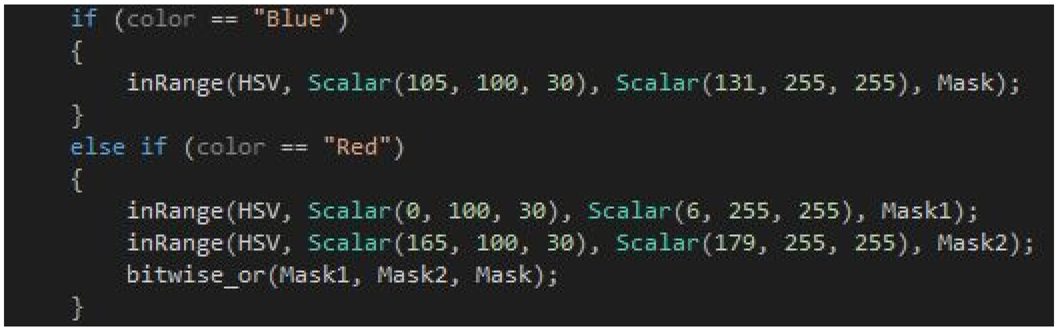
In order to debug tag colors, the user can enter the path to the image in "Image Path" on the Settings tab and select the "Debug" button. This will bring up two dialogs: "InRange Tester" and "Mask Result". The InRange tester includes two ranges of HSV values that can be adjusted to filter out all colors that are outside of the specified range. It includes two ranges because colors such as red contain ranges at the top and the bottom of the spectrum. There is a known issue that once the range is opened up, the user will not be able to make the range smaller again unless they close the dialog first. When the user is ready to stop debugging, they must hit escape before they are able to close the dialog. Another method of finding HSV color ranges is looking them up online. Most of the time this can be very helpful, even if it is only to find the starting point for a color range.



```
Hue Lo 1:: 0
Hue Hi 1:: 108
Sat Lo 1: 19
Sat Hi 1: 192
Val Lo 1: 56
Val Hi 1: 233
Hue Lo 2:: 13
Hue Hi 2:: 62
Sat Lo 2: 11
Sat Hi 2: 159
Val Lo 2: 22
Val Hi 2: 156
```

Adding New Tag Colors

After the HSV range for the desired color is found the new color can be added. For now, a new color can only be added in the code. The function Panel::MaskWithColor() in Panel.cpp contains the HSV ranges for our currently supported colors and this is where the range(s) for a new color must be added. The developer must also add the corresponding dialog buttons/options to go with that color.



```
if (color == "Blue")
{
    inRange(HSV, Scalar(105, 100, 30), Scalar(131, 255, 255), Mask);
}
else if (color == "Red")
{
    inRange(HSV, Scalar(0, 100, 30), Scalar(6, 255, 255), Mask1);
    inRange(HSV, Scalar(165, 100, 30), Scalar(179, 255, 255), Mask2);
    bitwise_or(Mask1, Mask2, Mask);
}
```

Known Issues

We have encountered an issue when the default feature (qr_steelcase.png) can cause a false positive detection because of the red and blue present in the feature image. If image bounding by feature detection is necessary to limit the detection area for tag detection, we recommend trying one of two things:

1. Make code changes to ignore or cover up the feature

2. Make changes to Config/Setting.xml (min_blob_area) to adjust the minimum blob (tag) area (in pixels)

Also if there are other objects in the frame with a similar color and high enough intensity, that may cause a false positive detection.

Constraints

- There must not be any objects of similar color and high enough intensity in the detection frame or a false positive detection may occur.

Length and Width Detection

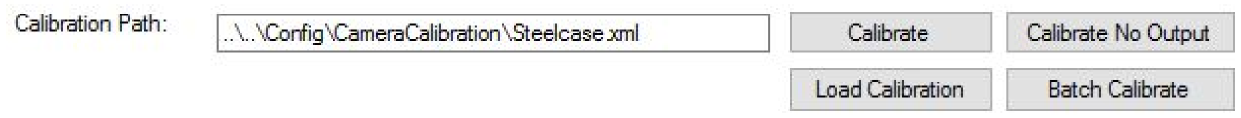
StPaQuAs Length and Width Detection reads in an image and determines the length and width in centimeters of the panel/part present in the image. As of now Length Conversion only shows the dimensions in a message box, but the length and width values can be used to compare against the information from a trim ticket once the application is integrated into Steelcase's system.

Distortion Removal

The distortion introduced by the cameras can be removed using the camera calibration features. A sample configuration file titled "Steelcase.xml" can be found in Config\CameraCalibration. This file gives information on many factors, the ones most likely to be edited:

- BoardSize_Width - gives the number of square corners to find width-wise (default 9)
- BoardSize_Height - gives the number of square corners to find height-wise (default 6)
- Calibrate_Pattern - name of which pattern to detect (default "CHESSBOARD")
- Input - location of file to be read that points to the images to be used
- Calibrate_NrOfFrameToUse - how many images to use
- Write_outputFileName - Location of file to save the calibration parameters

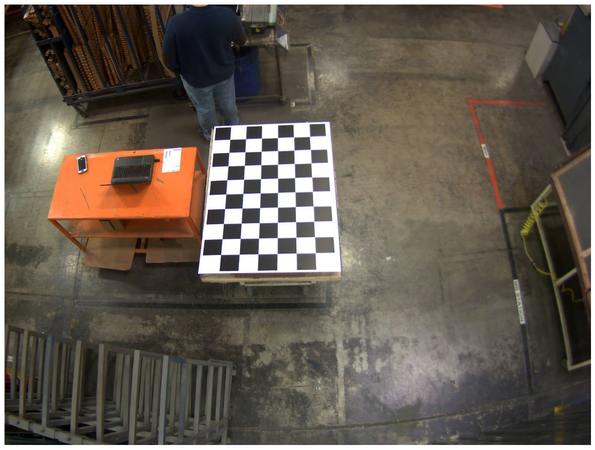
The sample "Input" file is titled "Steelcase_input.xml" and can also be found in Config\CameraCalibration. It has been preloaded with images that have found a good set of corners. The complete set of given checkerboard images can be found in "Config\CameraCalibration\CalibrationImages".


Calibration Path: ..\..\Config\CameraCalibration\Steelcase.xml

Once the xml file for the calibration has been entered into the Calibration Path input box, the user may select either “Calibrate” or “Calibrate No Output”. The former will display each image and the checkerboard corners that were found. This is useful to determine which images are good candidates and which images will not work. Bad candidate images for calibration are ones were the lines are not lined up on checkerboard corners or where no checkerboard was found at all. “Calibrate No Output” will also calculate the camera calibration, but it does not output images. This is useful for generating the output calibration file once the proper checkerboard images have been chosen.

The “Load Calibration” button is used for loading a calibration that was previously created with one of the Calibrate buttons. This file to load was defined as the <Write_outputFileName> in the original .xml file (Steelcase.xml). The example output file can be found in Config\CameraCalibration titled “out_camera_data.xml”. Once a calibration is loaded, the “View With Calibration” button will now apply this camera calibration to the image in the input box.

“Batch Calibrate” is used for running the calibration on many images at once. The images to be calibrated may be placed in a folder whose location is placed in the Calibration Path input box. A folder named “Calibrated_Folder” must be present in “Panel_QA_TestProgram\bin\Release”. This is the location where the calibrated images will be saved to.


Original image

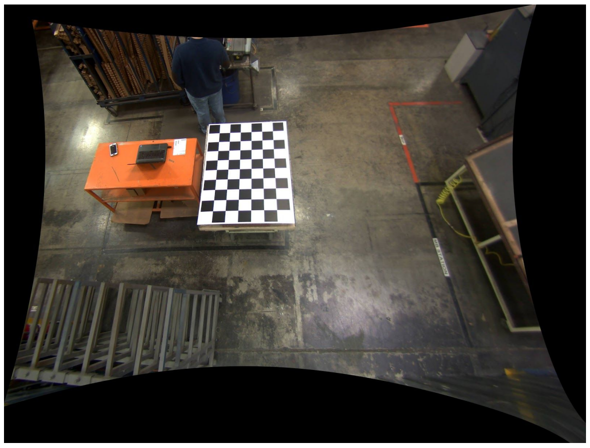
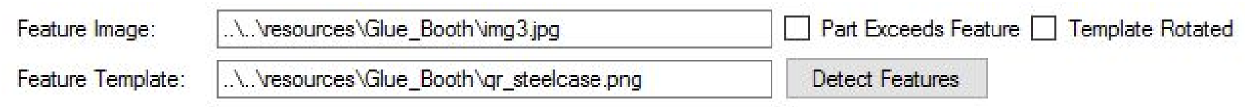


Image after distortion removal

Setting Boundaries and Length Conversion

The “Detect Features” button may be used to crop the image to remove some of the environment noise that is not the panel. The Feature Template is the image that will be detected while the Feature Image is where the image containing the panel will be placed.


Feature Image: ..\..\resources\Glue_Booth\img3.jpg
Feature Template: ..\..\resources\Glue_Booth\qr_steelcase.png

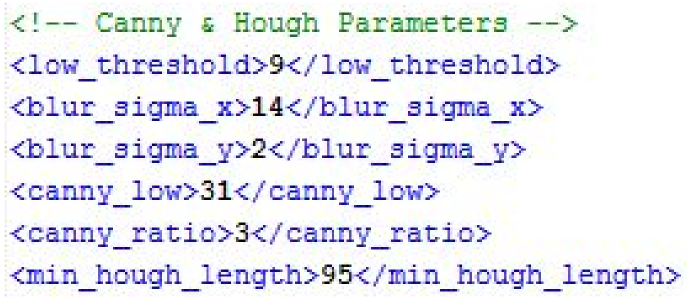
When the “Detect Features” button is pressed, the software will find the features in the image and automatically create a region of interest from which you can crop further images. The region of interest is saved in image_boundary.xml which is found in panel_quality_assurance\Panel_QA\Panel_QA_TestProgram\Config. This file is automatically loaded at the startup of the program. It can be edited manually, but currently the software would have to be closed and re-opened after updating the file. “Detect Features” also calculates the pixel to cm conversion rate and is stored in the same xml file. The <image_boundary> parameters are listed as 4 numbers. These are defined as the x and y starting points for the image, and the following two are how many pixels to extend away from the starting location to define the rectangle. Use the paths in the above image to see a working example of settings boundaries and finding length and width conversion using Feature Detection.

Debugging Canny and Hough Parameters

The parameters used for Canny edge detection and Hough line detection can be debugged using the “Canny Edges” button on the Settings tab. Entering the path to the image and selecting this button will bring up several dialogs with the intermediate steps for measuring length and width including: Canny edges, Hough lines, thresholding, and Gaussian blur. The parameters for these can be adjusted using the sliders similar to blob detection and the InRange tester. As the sliders are adjusted, the intermediate images are dynamically updated to reflect the changes.

Setting Canny and Hough Parameters

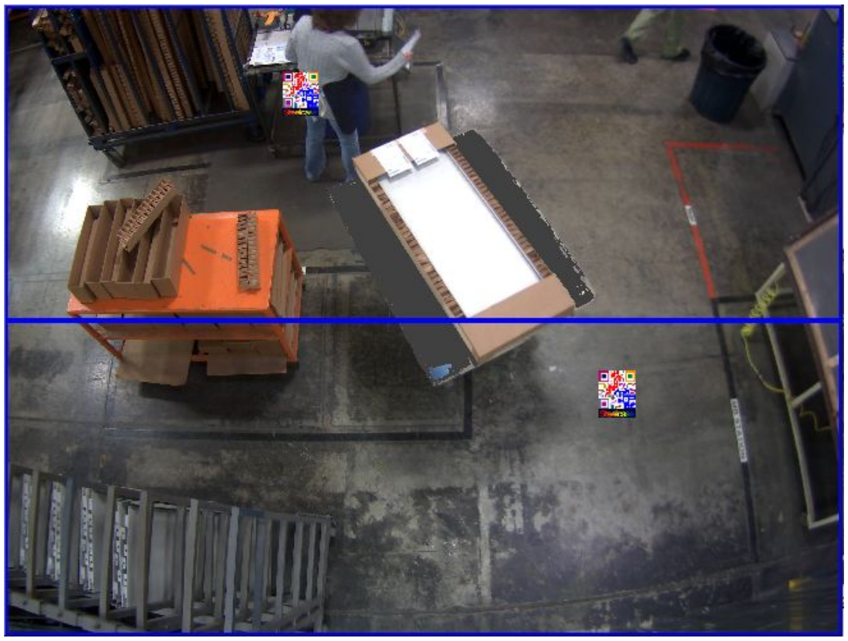
Once the optimal settings are found using the steps above, the settings must be entered into Config/Settings.xml for later use:



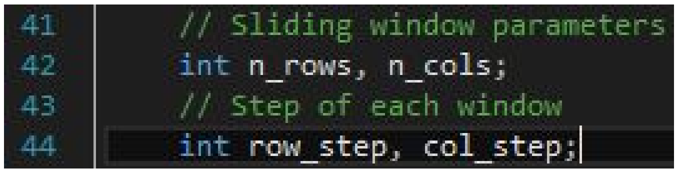
```
<!-- Canny & Hough Parameters -->
<low_threshold>9</low_threshold>
<blur_sigma_x>14</blur_sigma_x>
<blur_sigma_y>2</blur_sigma_y>
<canny_low>31</canny_low>
<canny_ratio>3</canny_ratio>
<min_hough_length>95</min_hough_length>
```

Adjusting the Sliding Window

A sliding window is used during feature matching to allow us to detect two features instead of only one. By default, the sliding windows divides the image into a top half and a bottom half.

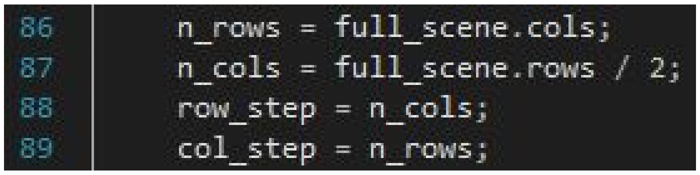


This sliding window can be adjusted in MyFeatureDetector::SlidingWindow() in FeatureDetector.h. It is controlled by the following variables:



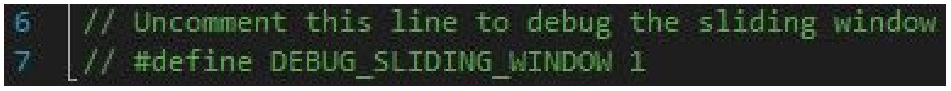
```
    // Sliding window parameters
    int n_rows, n_cols;
    // Step of each window
    int row_step, col_step;
```

Currently the values of those variables are set as follows:



```
    n_rows = full_scene.cols;
    n_cols = full_scene.rows / 2;
    row_step = n_cols;
    col_step = n_rows;
```

To see what changing these values does the user can uncomment the following line and run “Detect Features” from the Settings tab:



```
// Uncomment this line to debug the sliding window
// #define DEBUG_SLIDING_WINDOW 1
```

Running this will display step-by-step dialogs of how the sliding window is operating.

Known Issues

Sometimes feature detection can be sensitive to rotation. The OpenCV algorithm claims to be rotation and scale invariant but we have seen a few cases where that is not completely true.

Constraints

- The panel must be rectangle to get an accurate measurement.
- The camera must be positioned to as close to overhead as possible. The steeper the angle of the camera, the less accurate the measurement will be.
- For Feature Detection boundary setting and length conversion, the features must be fully in view in the image and as clear as possible so that the feature detector is able to find it.
- The features must not be located in the same half horizontally across the image. This can be changed by adjusted, however (See Sliding window).
- For length and width measurement the part must be within the bounds of the top left corner of the top feature and the bottom right corner of the bottom feature unless the “Part Exceeds Feature”.

Future Improvements

There are several things that we would have liked to have completed, but we simply did not have time before the semester ended. We believe this would be a great project to pass on to another design team. Here are some improvements that we suggest for future design teams:

- Expand dimension detection to work with objects that are not rectangle
- Implement settings dialog to replace Settings.xml
- Implement panel color detection
- Implement panel end cap width detection
- Integrate camera calibration into dimension detection process