12-2015

Detection and recovery from camera bump during automated inspection of automobiles on an assembly line

Shawn Mathew
Clemson University, snmathe@clemson.edu

Detection and recovery from camera bump during automated inspection of automobiles on an assembly line

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

by
Shawn Nainan Mathew
December 2015

Abstract

This thesis details the steps taken to detect and compensate for camera bumps while performing part identification using VI at the BMW manufacturing plant and on the simulation testbed. For the system presented here to work, the user is required to record a video from the camera before the camera is bumped and one after the camera has been bumped. The premise behind the method suggested here is that the transformation between the background in the pre- and post-bump video will be equal to the transformation in the foreground. A Background extraction program is used to generate a background image from the pre- and post-bump videos. Feature tracking and matching is performed on the background images to find the transformation between them. This transformation is then applied to the templates extracted from the pre-bump video. An additional manual compensation step is needed in cases where the transformation in the background is not equal to the transformation in the foreground. The resultant transformation is applied to all the templates of the pre-bump video and VI is seen to successfully identify parts with sufficient accuracy.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis considers the problem of detecting and recovering from a bump-like motion of a stationary camera being used for automated inspection. Cameras are frequently used for automated inspection during manufacturing assembly processes in order to detect defects. For example, the visual system developed in [5] is used to detect defects in textile fabrics. As another example, [6] details a machine vision system to detect defects in printed circuit board components. Cameras can also be used to check whether bottles are being filled to the correct volume or to check that bolts have been properly inserted into a housing. Such a camera is required to be stationary. If it moves, then the measurements it is taking of the parts become distorted. This thesis considers the problem of bump-like motion, in which the camera is stationary for a long period of time, then undergoes a single small movement (a bump), and is then stationary for another long period of time. The goal is to detect that the bump has occurred, and then take corrective measures that allow the inspection system to continue operating despite being repositioned.

On an assembly line, a camera will typically be bolted down or enclosed in an apparatus designed to protect it from the environment. However, on occasion it is possible that a stray tool movement from a human worker could accidentally contact a camera, bumping it, and causing the field-of-view to change. It is expected that the system would then start to operate poorly and that this poor performance would be noted. Further, it is expected that a human worker would observe that the camera had been bumped and manually try to re-position it to its original location and orientation, in order to make the inspection system operational again. However, it is expected that this manual re-positioning would never identically match the original positioning. The goal of this thesis is to detect and correct for this mismatch in order to restore the performance of the inspection system as closely as possible to its original installation. It must be noted that the alternative would be to recalibrate and retrain the system at its new positioning, which takes a great deal more time and thus keep the inspection system off-line for a much longer period of time.

Camera motion has been extensively studied in the computer vision literature. In its most complex variation, the problem can be formulated as tracking the motion of a hand-held camera moving freely through the world. Other variations constrain the problem, for example limiting motion to 2D or other restricted trajectories, or limiting the visible content to things easily tracked, etc. This work considers a fairly restricted variation in that the bump-like motion is assumed to happen once between long periods of no motion. However, the scene content is challenging in that it is assumed that the camera is observing an assembly line where a large portion of the field-of-view is in constant motion.

1.1 Visual Inspector (VI)

This thesis is meant to improve the robustness of the Visual Inspector (VI) [7] system which was developed as a joint research project between Clemson University and the BMW Spartanburg manufacturing plant. VI is a computer vision system that performs an automated inspection of parts on the assembly line to detect defects and missing parts and notify workers about the same. It was designed to be a cheaper alternative to commercially available automated inspection systems. Another feature of VI is its flexibility, in that it can be configured and trained to detect defects and missing parts at several locations along the assembly line.

VI uses template matching to perform the part identification. During configuration of VI at a location on the assembly line, the user installs the camera on a rigid mount as shown in figure 1.1 so that the object being tracked is clearly visibly within the frame as shown in figure 1.2. This section of the assembly line is called the door line.

In order to identify parts like the door emblem, various template images of the emblem are matched with the frame to find the best match. This technique is called template matching. Instead of matching the emblem templates with every frame from the camera, a trigger is defined that would activate VI's template matching process. The trigger is chosen to be an object that is always present with the part to be identified. To further reduce computational costs, the trigger is

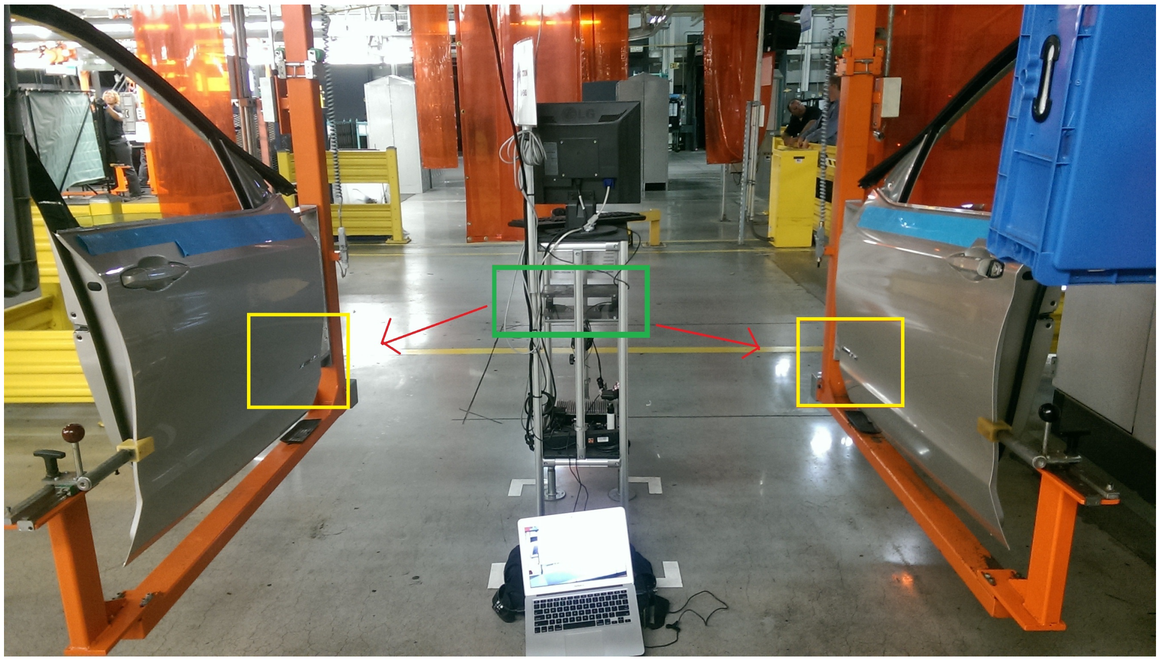


Figure 1.1: The camera setup to scan the car's door emblems. The green box highlights the cameras mounted on the rig, and the yellow boxes highlight the target to be identified.

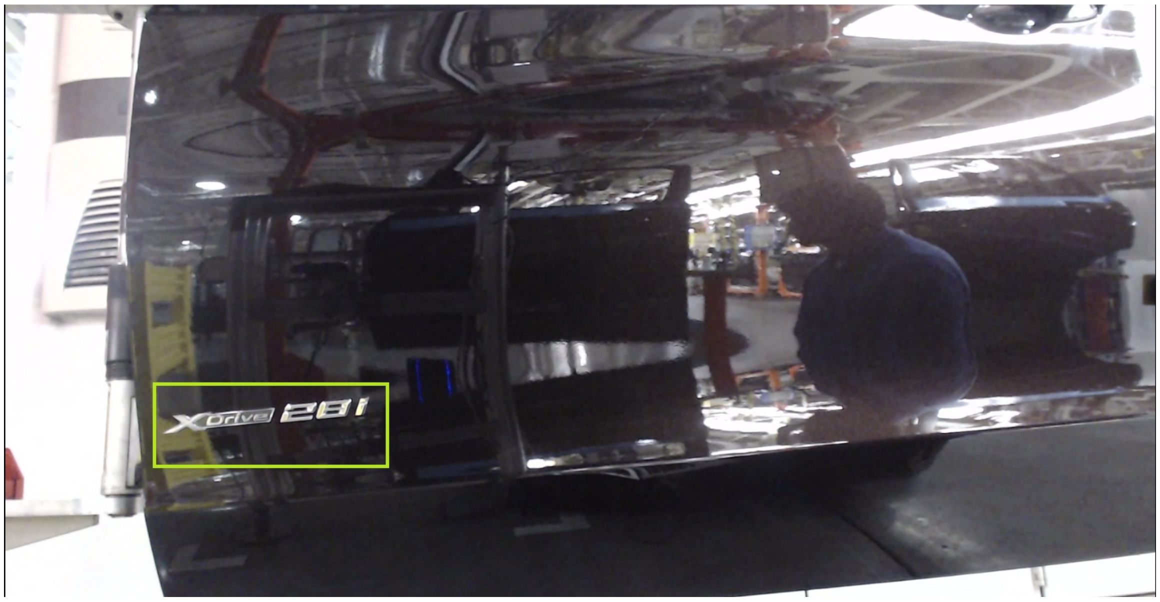


Figure 1.2: A frame from the camera that has been mounted to scan the door emblem, enclosed by the green box, on the door.

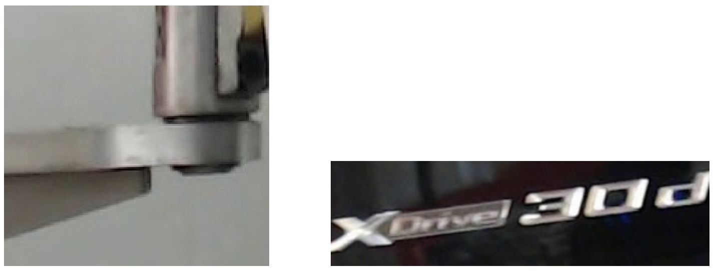

(a) A trigger template (b) A class template

Figure 1.3: Some of the templates used in the part identification process.

only searched for in that section of the frame where the trigger object is known to always appear. This frame section is called the trigger search window and the images used to trigger VI are called the trigger templates. Once VI is triggered, the emblem templates are matched with the camera frame. Once again, to reduce the number of computations, the emblem templates are matched with only that section of the frame where the emblem is known to always appear. This frame section is called the inspection window and the emblem templates are called class templates.

Figure 1.3 shows the templates used to identify the door emblem and figure 1.4 shows the trigger search window and the inspection window. A Setup program is designed to enable the user to select trigger templates, the trigger search window, class templates and the inspection window from a recorded video. Once the user extracts the necessary templates and selects the appropriate windows using this program, the user runs VI. VI performs template matching and displays the identified part. Figure 1.5 shows the output of VI when a part is successfully identified. VI records the identification results and the user can review these results to evaluate the performance of VI using the Review program. While reviewing, the user can add additional templates and make other changes to improve VI's accuracy. A well trained VI system has an accuracy of 99.5%.

1.2 The problem

As seen in section 1.1, VI correctly identifies parts so long as the part appears within the search window. However, if the camera is bumped by a passing worker, as has happened at the BMW Spartanburg plant, the part may no longer appear within the search window, as shown in figure 1.6. Even if the part still appears within the search window it can happen that the camera has rotated or has been moved closer to the object, resulting in scaling. As a result, VI fails to identify

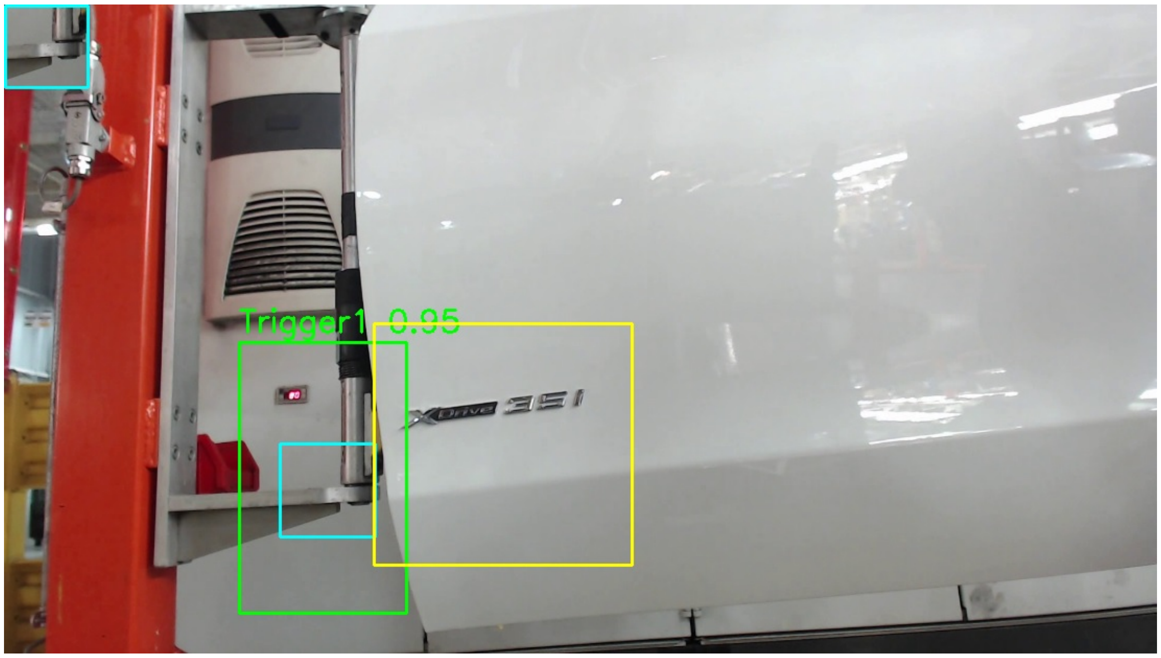


Figure 1.4: A screen grab from the setup program. The green box indicates the trigger search window and the yellow box encloses the inspection window.

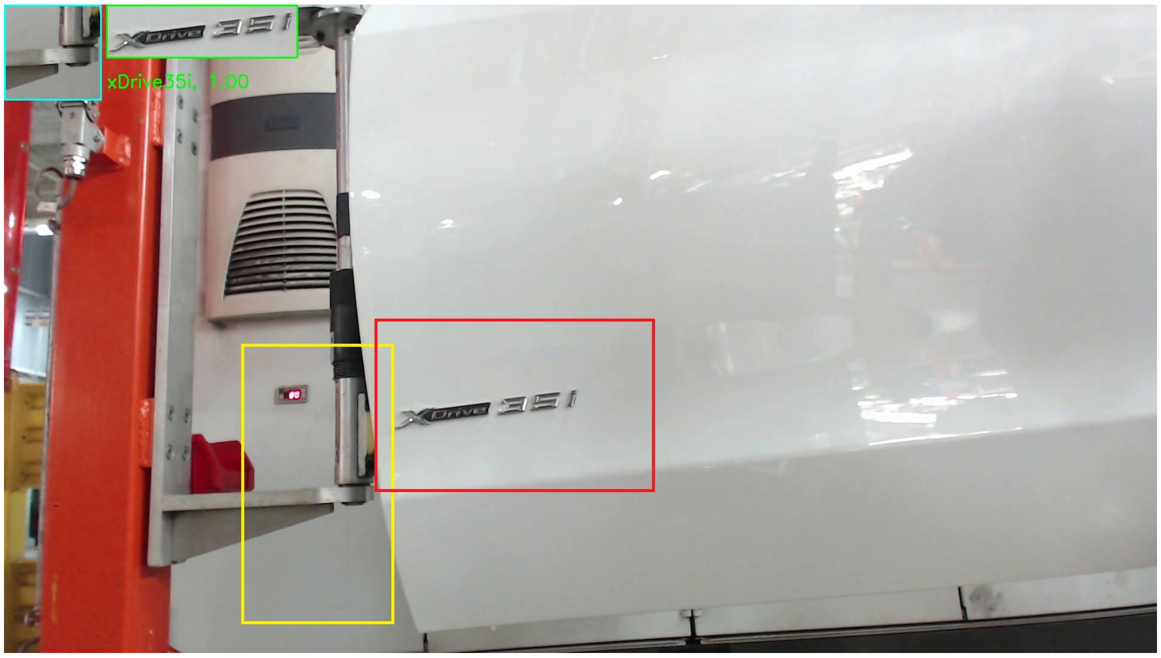


Figure 1.5: A successful identification by VI. The overlapping image enclosed by the blue box shows the trigger template image that best matched with the trigger search window (enclosed by the yellow box). The overlapping image within the green box shows the class template that best matched with the inspection window (enclosed by the red box).

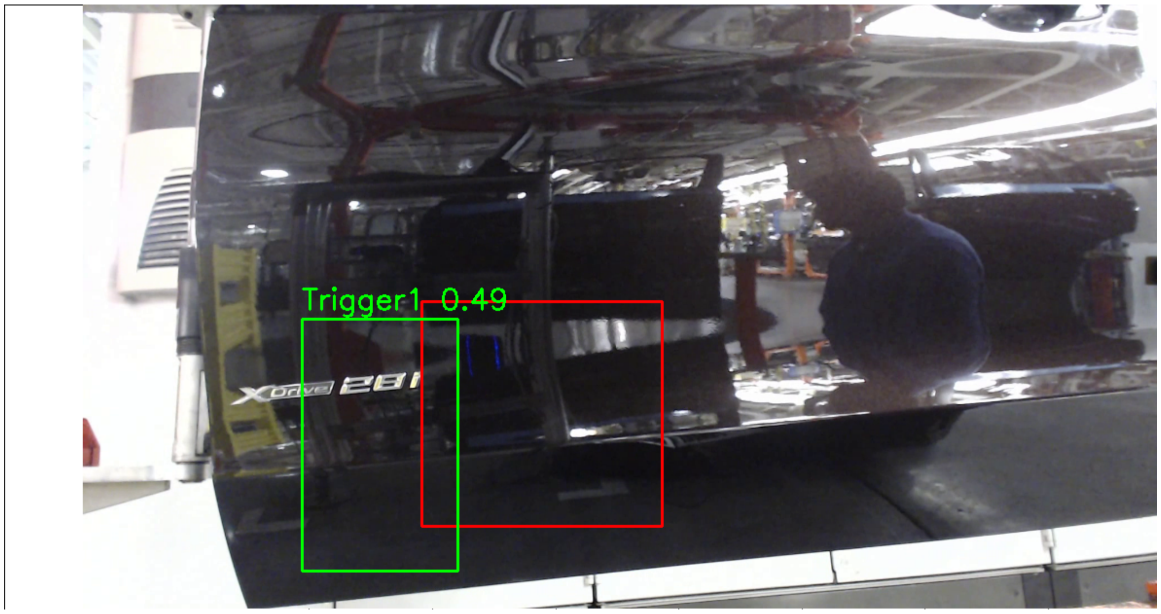


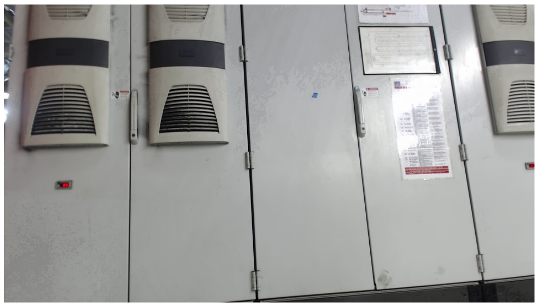
Figure 1.6: An unsuccessful identification by VI due to a camera bump. The green box indicates the trigger search window and the red box encloses the inspection window. The trigger value of 0.49 is the measure of the match. A minimum of 0.7 is required to successfully identify a part.

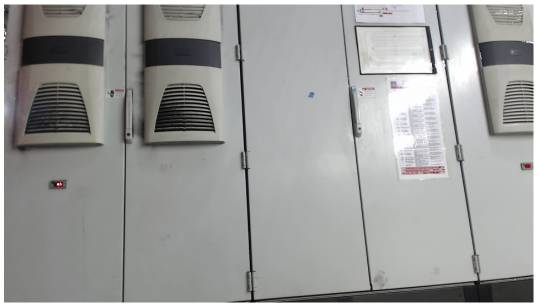
the part. Repeated failures alert the user to inspect VI. When it is identified that the camera has been bumped, the user tries to reposition the camera to its original position. If the user is unable to do so, the current templates would become unusable and the system would have to be re-trained by selecting new templates using the Setup program. Depending on the location at which the camera has been installed, this process could require several hours of recorded video.

It is assumed that in the event of a camera bump, the user tries to re-position the camera as close as possible to its original position. Nonetheless, this thesis corrects bumps of larger degrees as shown in figure 1.7. It fixes this problem by calculating the displacement of the camera image between the pre-bump and post-bump videos and applying this displacement to all the templates. The solution presented here reduces the retraining time to 15-60 minutes depending on the complexity of the parts being identified by VI.

1.3 Related work

The solution to camera stabilization presented here is specialized to the assembly line of a manufacturing plant and relies only on the before and after bump background images to calculate


(a) Background image from position 1


(b) Background image from position 2

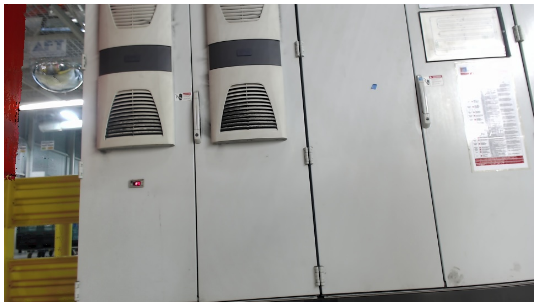
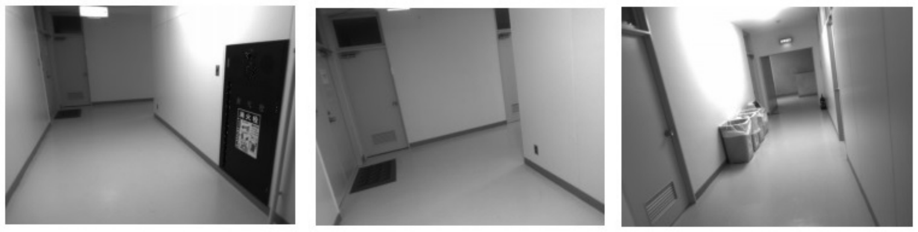

(c) Background image from position 3

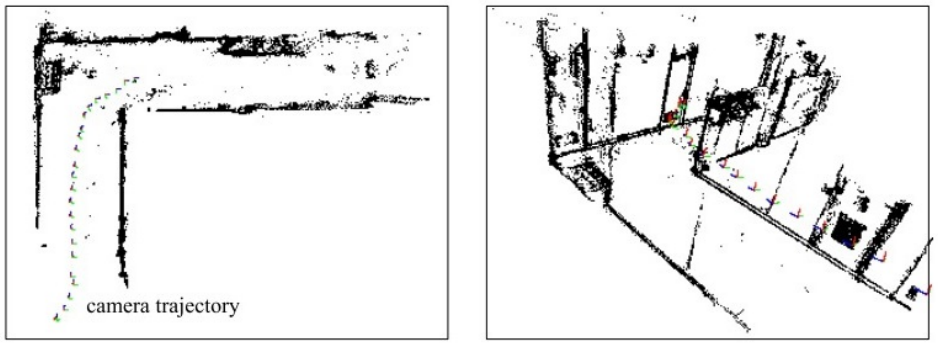
Figure 1.7: The background images from three different camera positions that show the varying degrees of camera bumps. The camera bump between (a) and (b) is very slight while the shift between (b) and (c) is a great deal more.

Figure 1.8: DJI Osmo. Image borrowed from http://www.cnet.com/products/dji-osmo/.

the transformation to be applied to the templates. Further, the camera is bumped from a stationary position to another stationary position which is as close as possible to the original position. Commonly, camera stabilization algorithms and techniques continuously compensate for camera shifts. Camera stabilization is an issue that has always plagued videographers and photographers. Some of the problems include camera shakes when using handheld cameras that result in an image blur and soft images, reduced visual quality of the video due to camera motion, etc. To capture sharp images and steady videos the cameras are mounted on a tripod stand. Newer digital camera models come with settings that automatically compensates for the inadvertent camera shake. However, when the camera is required to move from one position to another on a car, quadcopter or simply by hand, solutions to camera stabilization problems include gyroscope controlled mounts like the DJI Osmo shown in figure 1.8 and software solutions like [8] where video quality is improved by filling in missing parts of the image by using image data of neighboring frames, and [9] which uses particle filter tracking to track the projected affine model of the camera motions. Other solutions to this problem include those that rely on tracking camera motions and background tracking.

Camera motion tracking is a technique that has wide application in several fields like area mapping. [10, 11, 1] discuss simultaneous localization and mapping (SLAM) which is used to create


(a) Some frames from the camera



(b) The 3D map created from the input frames

Figure 1.9: A 3D map created by tracking the motion of a camera along a corridor. Image borrowed from [1].

a global map of an unknown environment by tracking the observer's location. Figure 1.9 shows the 3D map created by tracking the motion of the camera along the corridor by aligning edges in consecutive frames using the ICP algorithm.

Background extraction or subtraction is a common procedure in computer vision. It is used to solve several problems like in traffic cameras [12] for speed detection, lane density, controlling traffic movement, etc. Figure 1.10 shows the results of the background extraction algorithm implemented in [2]. Other approaches to solving the background extraction problems include [13] which uses a probability based algorithm, and [14] which uses linear algebra to extract the background image. Background extraction also has several application in surveillance. For example, [3] discusses a background tracking problem using a self-organizing neural network to detect unattended baggage, as seen in figure 1.11.

As described here, the camera used at the inspection line of the BMW plant is rigidly mounted and is not expected to move freely. Therefore, unlike the camera stabilization solutions

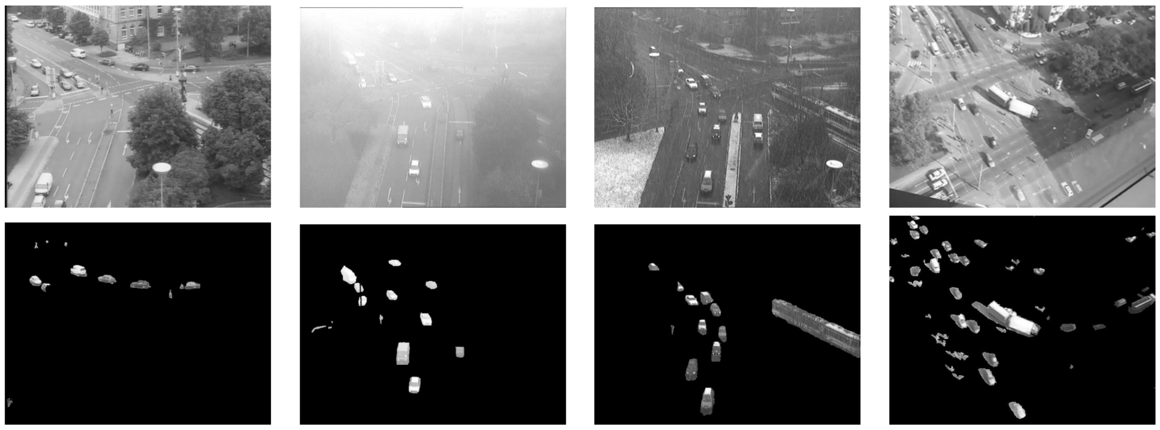


Figure 1.10: Background extraction on video from a traffic camera. Image borrowed from [2].

listed here, the solution presented here is much simpler and relies only on two background images generated before and after the camera bump to calculate the displacement. This greatly simplifies the stabilization problem, requiring only to find the affine transformation between the two images. Moreover, the background extraction problem is also specialized for the assembly line of the plant, in that it requires tracking the intensity of the pixels for the most frequently stable intensity values. Due to the nature of the assembly line, these intensity values are assumed to belong to the background image.

1.4 Novelty

To our knowledge, this is the first work to consider the problem of detecting and recovering from a bump-like motion of a stationary camera being used for automated inspection on an assembly line. This thesis describes an algorithm to accomplish this goal. It demonstrates the algorithm on a simulated testbed and on real data collected at the BMW Spartanburg plant.

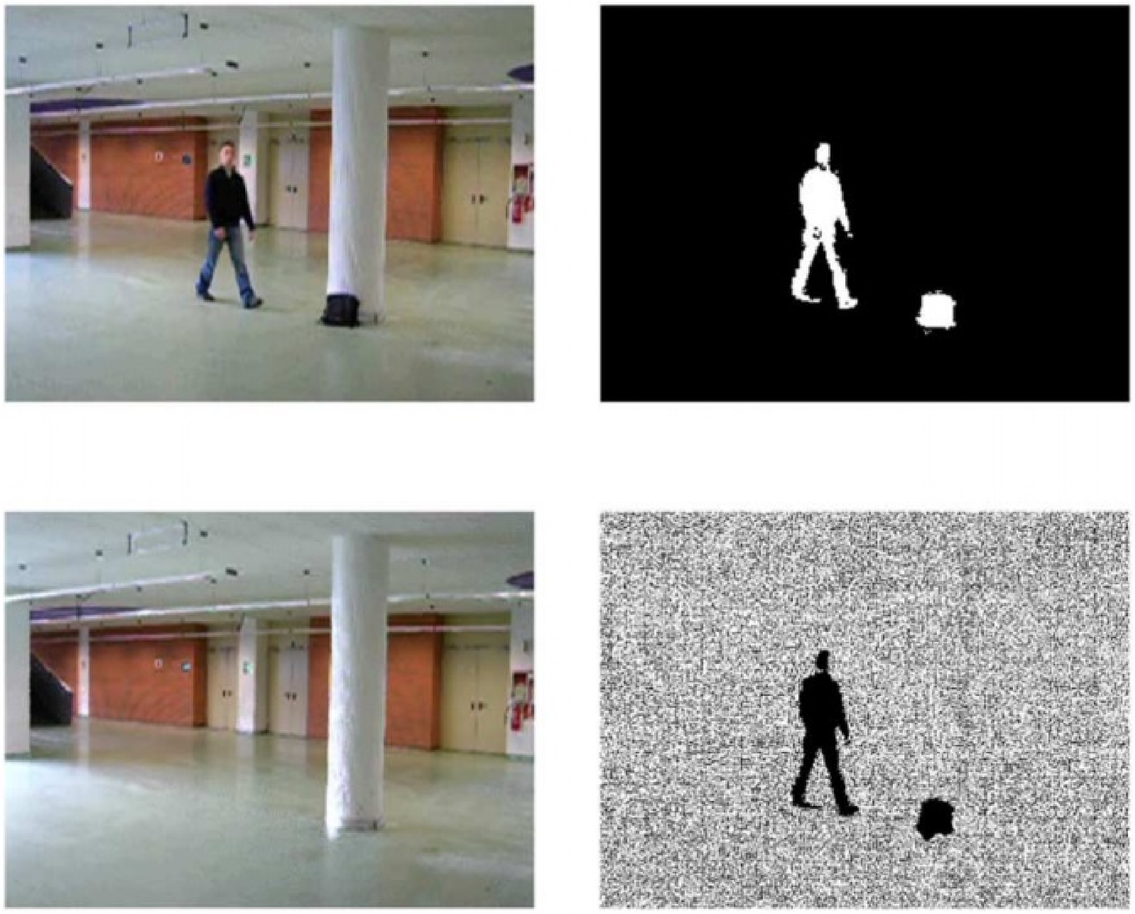


Figure 1.11: Background extraction used in surveillance to detect unattended baggage. Image borrowed from [3].

Chapter 2

Method

This chapter presents an overview of the bump detection and compensation method and details the steps taken to implement the camera bump detection and compensation system, which are background extraction, feature detection and matching, and affine transformation. In the event that these steps are not sufficient to compensate for the camera shift, a semi-automatic method is also detailed where the user is required to readjust the search windows using the Setup program.

2.1 Overview

The following events outline the process of implementing camera bump detection and compensation at the BMW Spartanburg plant.

1. When the user sets up VI initially, a 30 minute long video is recorded from the camera at the location. This serves as the pre-bump video. Once VI is set up, it is expected that the system runs smoothly without incident.

2. If and when a human worker bumps the camera out of its original position VI may start failing. When the user notices a constant failure in VI and ascertains that the camera has been bumped, the user tries to realign the camera to its original position. It is likely that the user will be unable to exactly reposition the camera to its original position and so follows the next steps to detect the camera bump and compensate for it.

3. After taking VI offline, the user records another 30 minute long video from the new position

of the camera. This serves as the post-bump video.

4. The user then runs the background extraction program on both the pre-bump and post-bump videos to generate a background image.

5. The feature matching program is then run on the two background images to calculate the transformation between them. This transformation is applied to all the templates.

6. As an additional step, the user runs the manual compensation program to ascertain if the transformation is sufficient to enable VI to run smoothly. If additional compensation is indeed required, the user uses the Setup program on the post-bump video to reposition the template search windows and captures a frame of the video with the trigger within the search window. Using this, the compensation program determines the additional transformation to be applied to the templates.

7. The user can then test the new templates in VI on the post-bump video to check the performance of the modified system. If the user is satisfied, VI is brought online and the automated inspection can continue. To improve efficiency, the user can generate the background image from the pre-bump video before the bump has occurred. In that case, the bump detection and compensation process would take around 40-60 minutes.

The following section describes each of these steps in detail.

2.2 Bump detection and compensation

When a camera is moved out of its position, the image displayed by the camera goes through geometric transformations of rotation, translation, scaling and skewing. If this transformation is found, then it can be applied to the stored templates and the search windows, and VI can continue to work without having to be retrained. The first problem in finding this transformation is to locate recurring features between the images captured by the camera before and after the camera was bumped. The displacement in these features between the two images is the transformation to be applied to all the templates and search windows. However, due to the nature of the assembly line, it is difficult find good features to track. From figure 2.1 it is clear that the two frames are quite similar. Tracking the camera shift using these images would give incorrect results. It thus becomes

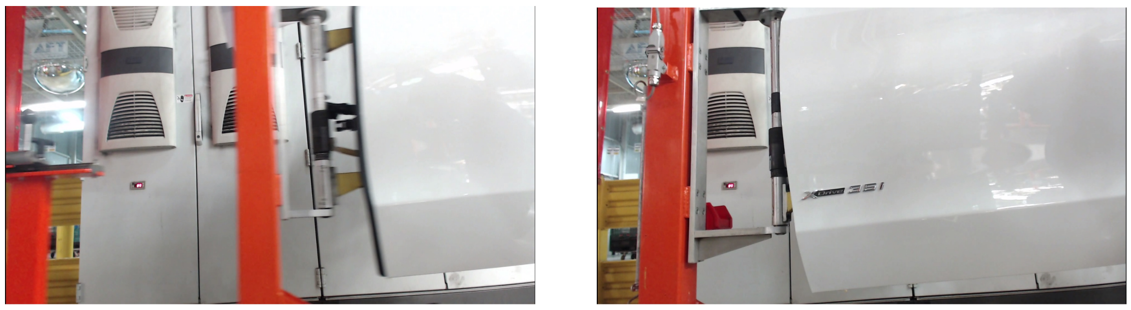

(a) (b)

Figure 2.1: Two frames from the camera installed at the door line.

necessary to eliminate the foreground, which include the parts on the assembly line and the workers moving in and out of the frame, from the background, which only composes of stationary objects like ceiling beams, walls and guardrails.

2.2.1 Background extraction

The approach to background extraction that is presented here is based on the premise that the most frequently stable pixels in an image must belong to the background. This necessitates a video of the assembly line that is sufficiently long enough to capture the background. The length of time depends on the activity in the camera frame. If there is a lot of repeated foreground activity in the camera frame it would take longer to generate a suitable background image. If however, there is very little repeated foreground activity in the camera frame, the background can be quickly generated.

For a pixel intensity to be considered a background pixel, it must have the total longest stable period. A pixel is said to be stable for a period of time if its intensity does not change by a significant amount during that period of time. If the history of the pixel is tracked over a sufficiently long duration it can be assumed that the most frequently stable intensity is a part of the background. This assumption is reasonable because of the constant and repeated motion of the assembly line.

When the user sets up VI initially, the background extraction program is run to generate a background image before a potential camera bump. If and when the camera is bumped, the user runs the background extraction program again to generate a post bump background image. Algorithm 1 is the background extraction algorithm used here, in its elemental form, where I_p is the channel intensity of pixel p. From tests conducted at the camera locations at the BMW manufacturing plant, best results were obtained when the *tolerance* was set to 15 and the *stabilityThreshold* was set to 40.

Algorithm 1 Background Extraction Algorithm (Elemental form)

```
1: for every pixel $p$ in the image do
2:     if change in $I_p < tolerance$ then
3:         $stabilityCounter_p$++
4:     else
5:
6:         if $stabilityCounter_p > stabilityThreshold$ then
7:             $avgIntensity_p$ = the average intensity of $p$ for the stable period.
8:             $Histogram(avgIntensity_p)$ += $stabilityCounter_p$
9:         else
10:            Reset $stabilityCounter_p$ to 0
```

The algorithm in its basic form would require a histogram of 256 intensity values for each of the 3 color channels for every pixel in the image. Thus requiring 1,593 Mbytes of memory space for a 1080p image. However, through tests it was found that only 5 intensity values need to be stored for every pixel. The color channels were combined into a single structure, i.e. as a color, so that if the intensity change of a single channel goes above the tolerance value, it is stored as a separate color. This brought down the memory requirement to 10 Mbytes.

Other enhancements to the basic algorithm include that if the histogram value of a color at a pixel is too small after a threshold amount of time, then that color value is rejected from the histogram, making room for another potential color value. The threshold for rejection was set so that after 2000 iterations of the algorithm, a color must have attained a histogram value of at least 200 to remain a candidate for the background image. Algorithm 2 shows the improved algorithm.

Depending on the activity in the camera frame, it takes 5-30 minutes of video time to generate a suitable background image.

2.2.2 Feature detection and matching

Once the background images are generated, the user runs the feature detection and matching program to find features and corners in the pre-bump and post-bump background image. The Harris corner detector [15] is a very popular corner detector. SIFT [16] is also a popular technique of detecting features in an image. The advantage of using SIFT corners are that they are invariant to rotation, translation and scaling. However, the Harris corner is not sufficient to find good matching

Algorithm 2 Background Extraction Algorithm (Enhanced form)

```
for every pixel $p$ in the image do
    if change in $color_p < tolerance$ then
        $stabilityCounter_p++$
    else

        if $stabilityCounter_p > stabilityThreshold$ then
            $avgColor_p$ = the average color of p for the stable period

            if $avgColor_p$ is a candidate for background then
                $Histogram(avgColor_p) += stabilityCounter$
            else

                if ( then$numberOfCandidates_p$ <5)
                    Add $avgColor_p$ as a candidate
                    $Histogram(avgColor_p) += stabilityCounter$
                else
                    Apply Rejection Principle
            else
                Reset stability counter to 0
```

features between images and the SIFT technique has been improved upon by several other feature detection techniques like ORB (Oriented FAST and Rotated BRIEF) [17] and BRISK (Binary Robust Invariant Scalable Keypoints) [18]. For this application, ORB was found to give best results. The OpenCV library is used to implement the ORB keypoint detector and descriptor generator. OpenCV is a Computer Vision Library released under the BSD license, making it free to use for academic and commercial uses. The ORB interface that is implemented in OpenCV is used for the purposes of detecting keypoints and generating descriptors for those keypoints. ORB is a fusion of the FAST keypoint detector and BRIEF descriptor with many enhancements to improve performance.

Figure 2.2 shows how the FAST [4] corner detection works. The pixel p is a corner if there exists a set of n contiguous pixels in the circle of 16 pixels which are all brighter than $I_p + t$, or all darker than $I_p - t$, where t is an appropriate threshold value. n is chosen to be 12.

Extracting keypoints out of an image gives us information about the position of the keypoint and its surrounding area. However, this information may not be sufficient to compare keypoints in different images. For this, keypoint descriptors are used.

Descriptors summarize certain characteristics about the keypoints. These characteristics could be its orientation, intensity values, etc. depending on the algorithm being used. The BRIEF [19] descriptor creates a binary vector using intensity comparisons in a smoothed patch

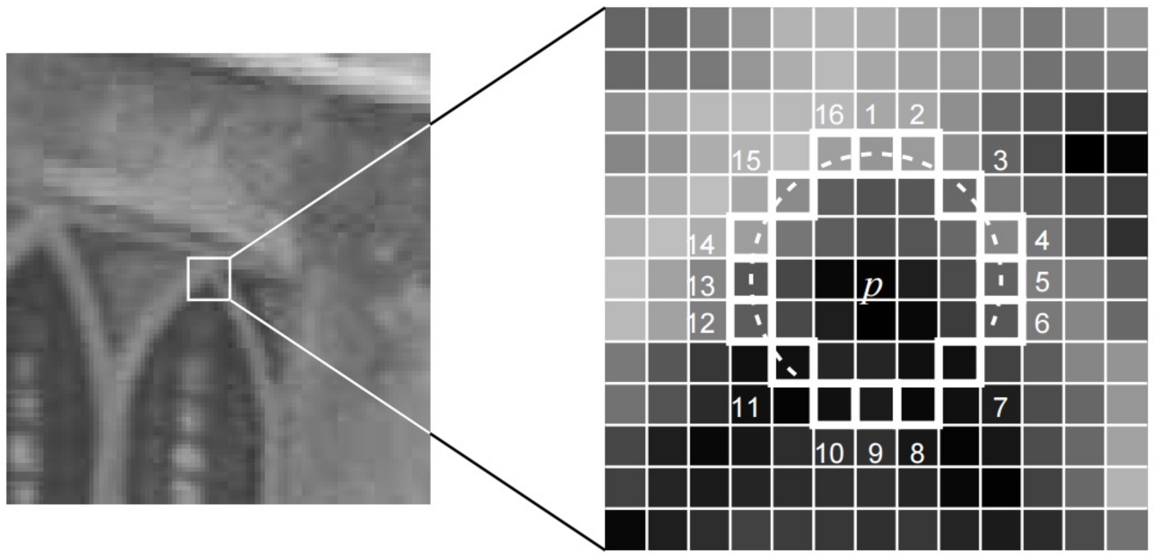


Figure 2.2: A corner detected by the FAST algorithm. Image borrowed from [4].

of the image. For example, a test on patch p of size $S \times S$ may have the following rule.

$$\text{Set bit to } \begin{cases} 1 & if p(x) < p(y) \\ 0 & otherwise \end{cases}$$

where *p(x)* is the pixel intensity in a smoothed version of p at location x. The length of this binary vector can be 128, 256 and 512 and this identifies the tests to be performed.

ORB improves on the FAST and BRIEF implementation by making several enhancements. For example, the FAST algorithm is not rotation invariant. ORB improves on this by calculating the orientation of the vector from the weighted centroid of the patch to the corner point.

After ORB is run on the pre-bump and post-bump background images, descriptor matching is performed to find the correspondence of the keypoints of the two images. For this another OpenCV function called the BFMatcher (Brute Force Matcher) is used. BFMatcher, implementing the brute force algorithm, basically takes a descriptor of a feature in one image and matches it using a distance measure with every feature in the second image until the best match is found. For this application, the Hamming norm is used as the distance measure.

After the keypoints are matched, the three best matches are chosen based on the rule that it must have the shortest distance between the two descriptors and that the points are separated by a

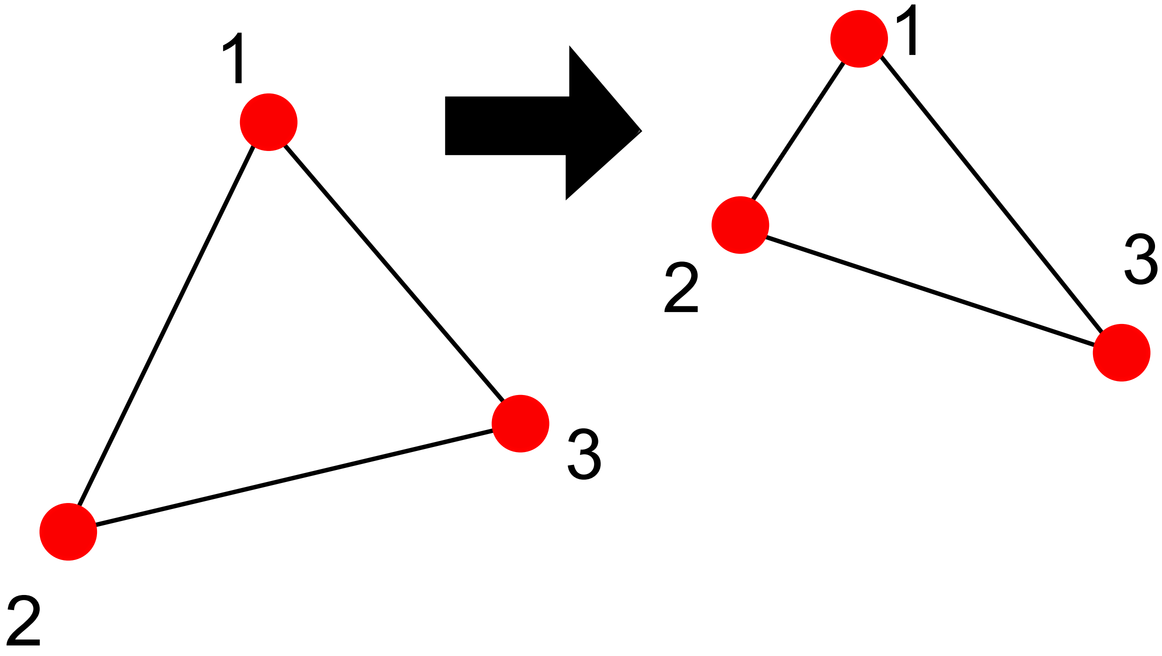


Figure 2.3: An illustration of the affine transformation.

minimum distance threshold. Well separated points give a better result for the affine transformation matrix in the next step. The minimum distance threshold for a 1080p image was chosen as 100. The user can choose an appropriate value depending on the texture in the background. These 3 points are then used to find the affine Transformation matrix that the image has to be transformed by.

2.2.3 Affine transformation

An affine transformation is any transformation that can be expressed in the form of a matrix multiplication (linear transformations like rotation and scaling), followed by a vector addition (translations). To find the affine transformation, three points are required. Figure 2.3 shows the transformation from the first image to the second image. The points 1, 2 and 3 in the first image correspond to points 1, 2 and 3 in the second image. The three best keypoint matches are used to find the affine transformation from one image to the other.

OpenCV implements affine transformation through two functions, `getAffineTransform()` and `warpAffine()`. The three selected points of each image are passed as parameters to the `getAffineTransform()` function implemented in OpenCV. This function generates the 2×3 affine transformation matrix that converts the points in one image to the other. Since the templates generated for the pre-bump conditions are to be modified for the post-bump conditions, this function is used to find the affine transformation matrix that converts the 3 points in the pre-bump image to the 3 points in the post-bump image. This affine transformation is then applied to the pre-bump image using the `warpAffine()` function so that it warps the image to be similar to the post-bump image. This enables the user to visually judge whether the calculated transformation is appropriate or not.

The image generated from the warp is then compared with the post-bump image and a similarity measure is used to how well the two images match. For the similarity measure, the gradient magnitude of the two images are calculated using the 3×3Sobel gradient kernel. The two gradient images are then compared pixel by pixel to attain a similarity measure. The simplified algorithm for this process is shown in Algorithm 3.

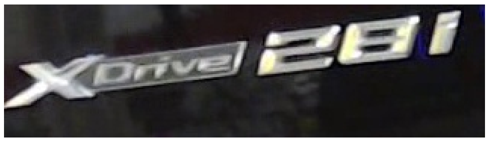
Algorithm 3 Feature Detection, Matching and Affine Transformation Steps

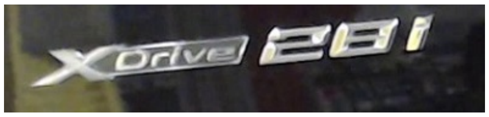
```
1: Generate ORB keypoints for pre-bump and post-bump image.
2: Calculate ORB descriptors for the keypoints.
3: Use BFMatcher (a brute force algorithm) to find matches in the two sets of descriptors.
4: Find 3 best matches from the top 10 matches using the minimum distance threshold.
5: Use the getAffineTransform function to generate the affine transformation matrix using the
   pre-image points as the source and the post-bump image points as the destination.
6: Apply the generated transformation matrix on the pre-bump image using the warpAffine func-
   tion.
7: Compare the gradient magnitudes of the warped image and the post-bump image to find the
   similarity measure.
```

For better results through better selection of keypoints and matches, a RANSAC algorithm [20] is used over this entire process. The gradient magnitude similarity measure is checked for each iteration of the process to find the best set of keypoints and matches, and consequently the best affine transformation matrix. Once a good affine transformation matrix is found from the RANSAC algorithm and the user, it is applied to all the templates. The templates need only be transformed by rotation and scaling, while the location of the search windows need only be transformation by translation as shown in figures 2.4, 2.5 and 2.6.

2.2.4 Manual readjustment

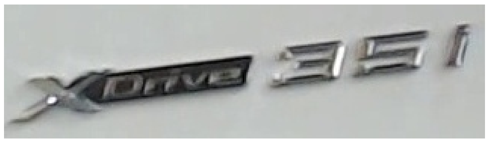
It was observed that the calculated affine transformation is not sufficient for every case. This is because the premise that the transformation in the background image will be equal to the transformation in the foreground does not hold in every situation, as shown in figure 2.7. Position


(a) Pre-bump template


(b) Post bump template

(c) Pre bump template

(d) Post bump template


(e) Pre-bump template

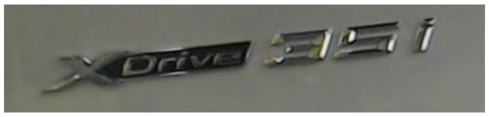
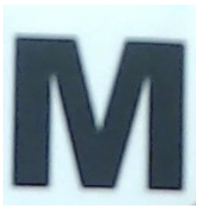

(f) Post bump template

Figure 2.4: The templates from the post-bump camera position are slightly rotated compared to the pre-bump template. This is further illustrated in Figure 3.19.


(a) Pre-bump template

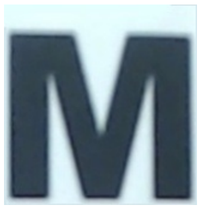
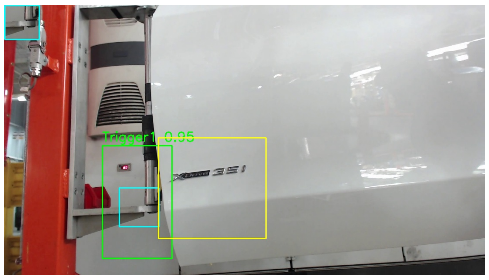
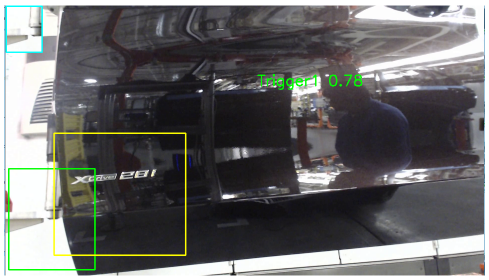

(b) Post bump template

Figure 2.5: The templates from the post-bump camera position are scaled compared to the pre-bump template



(a) Pre-bump location of search window



(b) Post bump location of search window

Figure 2.6: The translation of the search windows between the pre- and post-bump camera positions.

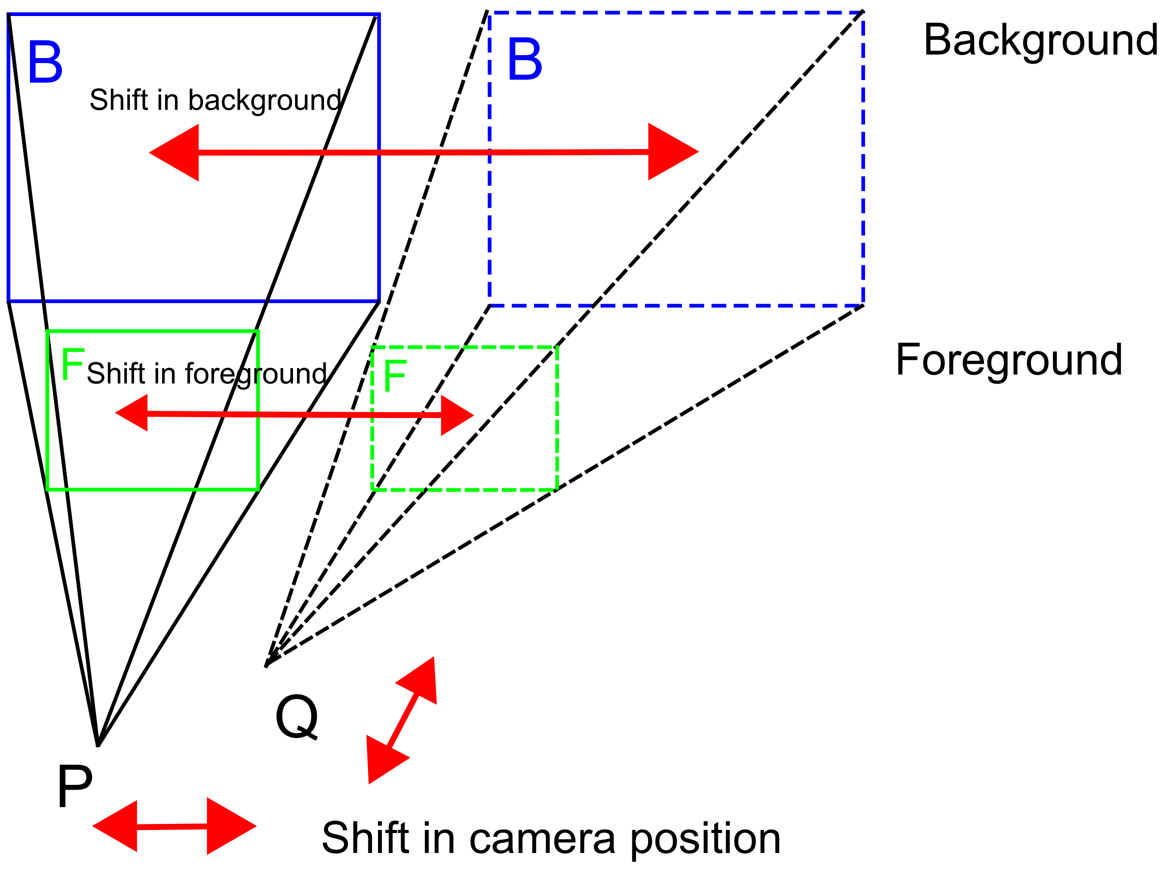


Figure 2.7: An illustration of an extreme shift in camera position.

P and Q are pre- and post-bump positions of the camera. As shown in the figure, the translation of the background does not equal the translation in the foreground. Additionally, the user may move the camera towards or away from the foreground, causing scaling in the foreground but not as much in the background. To compensate for this, user interaction is required to find the correct transformation for the templates. From the previous sections, it is clear that the process until this point corrects for any rotation in the foreground. This is because a rotation in the background must be equal to the rotation in the foreground in the environment of the assembly line. However, the translation and scaling may not be equal due to large difference in the distance between the camera and the foreground and the foreground and the background.

To correct this, the user runs the Setup program, where the user updates the location of the template search window to the correct location. The user also additionally saves a frame of the video where the trigger is within the trigger search window. The difference in the location of

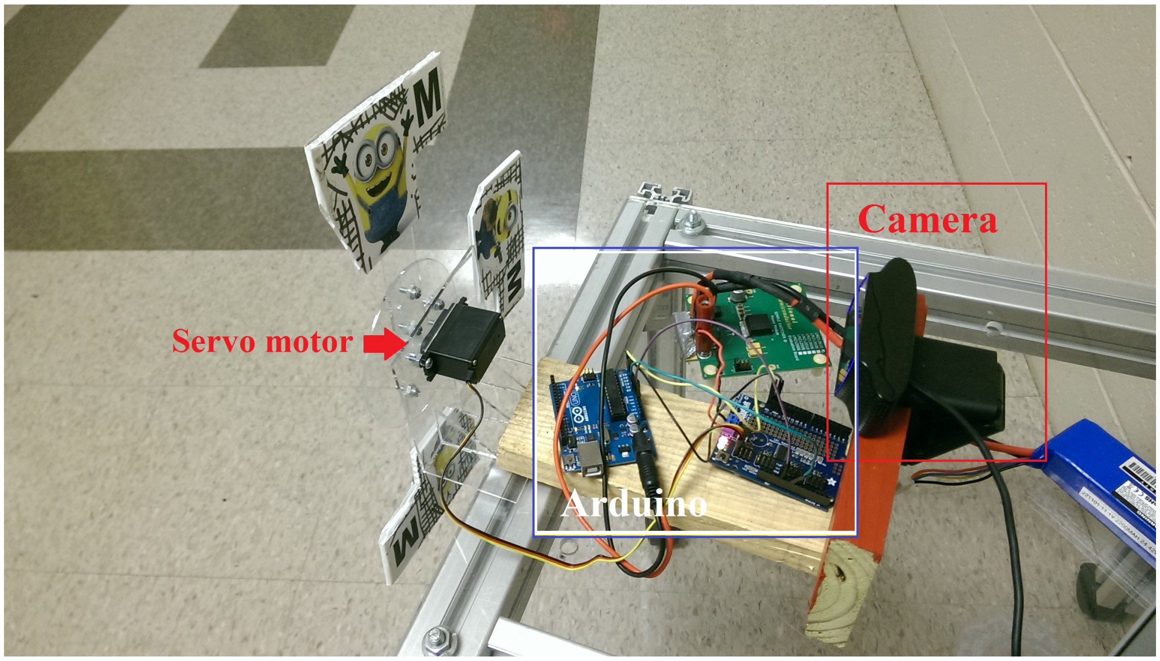


Figure 2.8: The simulation testbed setup.

the inspection window from the result of the warp and the manual interaction is the translation to be applied to all the other search windows. The saved frame is scanned within the updated search window to find the scale transformation to be applied to the templates by using a range of scale factor values until the best template match is found. This is the scale transformation to be applied to all the templates.

2.3 Simulation testbed

In order to test the bump detection and compensation system, pre- and post-bump videos are required. Since it was not feasible to take a VI system offline to record these test videos, a simulation testbed was designed. Figure 2.8 shows the setup of the simulation testbed consisting of a Logitech C920 webcam as the camera, a frame of images mounted on a servo motor, an Arduino Uno and a servo shield to control the servo motor and a power evaluation board to supply power to the micro-controller and servo shield. Figure 2.9 shows a frame captured from the the camera. The yellow box indicates the trigger search window and the red box indicates the inspection window. Figure 2.10 shows the trigger template and figure 2.11 shows the three class templates. It should

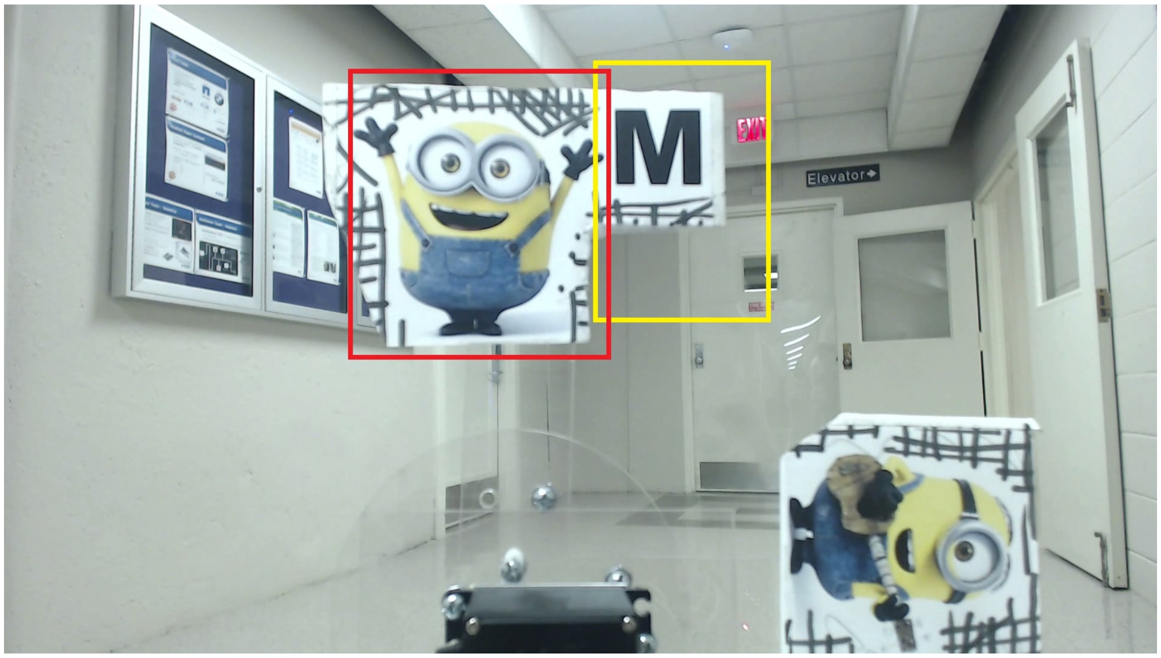


Figure 2.9: A frame captured from the camera on the testbed.

be noted that the objective of this simulation is to test not the part identification by VI, but the bump detection and compensation presented here. It is for this reason that only the three objects in figure 2.11 are used.

The testbed was designed in such a way that the frame holding the three images is rotated by the servo motor clockwise and counter clockwise through 180 degrees in 45 seconds. This was done to easily simulate the slow and repetitive nature of the assembly line at the BMW manufacturing plant. To simulate camera bumps, the camera was left stationary with the servo motor running to record the pre-bump video. The camera was then rotated about its axis by angles up to 10 degrees and translated by up to 2 cm. The camera was then left stationary to record the post-bump video. Table 2.1 shows the details of the videos recorded at various positions on the simulation testbed. The type indicates whether the video is a pre-bump or post-bump video.

2.4 Real data

Table 2.2 shows the details of the videos recorded at various locations in the BMW Spartanburg plant. These videos are used to test the background extraction algorithm presented here.

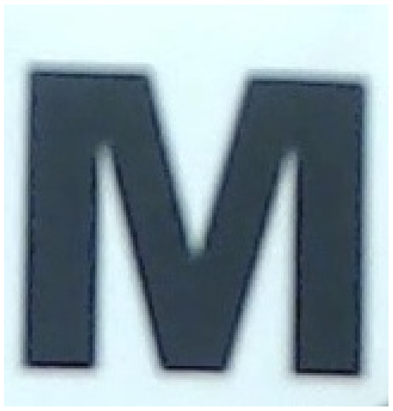


Figure 2.10: The trigger template from the testbed.

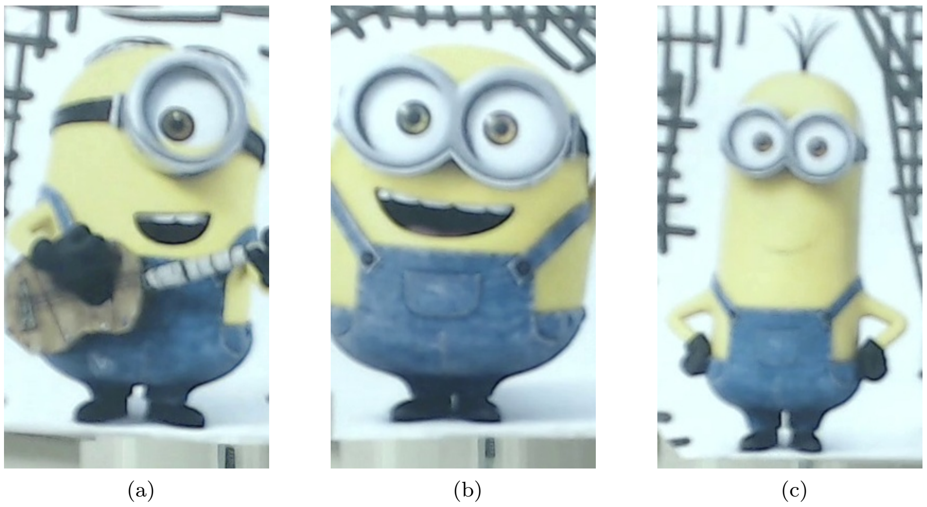


Figure 2.11: The three class templates from the testbed.

Video name	Duration	Number of objects to be identified	Type	Degree of bump
simulation pos 1	6:50 min	20	pre-bump	NA
simulation pos 2	5:12 min	15	post-bump	small bump
simulation pos 3	5:13 min	15	post-bump	large bump
simulation pos 4	5:04 min	12	pre-bump	NA
simulation pos 5	5:05 min	13	post-bump	large bump

Table 2.1: Details of the videos recorded at various positions on the simulation testbed.

Video name	Duration	Number of objects to be identified
engine line	33:33 min	21
spring line	1:02:37 hr	42
door line	7:00 min	4
door line pos 3	7:37 min	5
door line pos 4	5:36 min	4

Table 2.2: Details of the videos recorded at various locations in the BMW Spartanburg plant to test background extraction.

Video name	Duration	Number of objects to be identified	Type	Degree of bump
door line pos 1	9:06 min	5	pre-bump	NA
door line pos 2	7:10 min	4	post-bump	large bump

Table 2.3: Details of the video recorded at various positions on the door line of the BMW Spartanburg plant.

Since it was not feasible to bump the camera at every location at the BMW plant, multiple videos were recorded at the door line of the plant with varying degrees of camera bumps. Table 2.3 shows the details of these videos. Short videos are used since the objective of this thesis is to test the bump detection and compensation technique and not to test VI. Figures 2.12 and 2.13 show samples of the trigger and class templates recorded at the door line of the BMW Spartanburg plant.

2.5 Evaluation metric for the proposed system

To test the performance of the bump detection and compensation method presented here, the post-camera bump results of VI are checked before and after the correction was applied. Table 2.4 shows a sample of the comparison of VI's detection accuracy after the camera was bumped. The table shows the duration of the video, the number of objects to be tracked that appear in the video, VI's identification accuracy while using the original templates and VI's detection accuracy once the correction has been applied to the templates. Ideally, VI should give a near 100% accuracy after the correction has been applied to the templates.

Video name	Duration	Number of objects	VI accuracy before correction was applied	Accuracy after correction was applied
door line pos 2	7:10 min	4	0%	100%

Table 2.4: A sample comparison of VI's detection accuracy after the camera was bumped by a large degree.

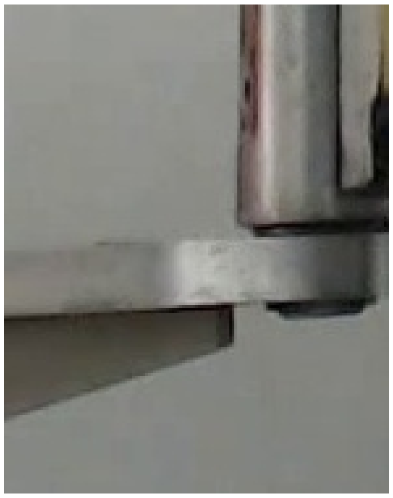


Figure 2.12: A trigger template from the door line of the BMW Spartanburg plant.

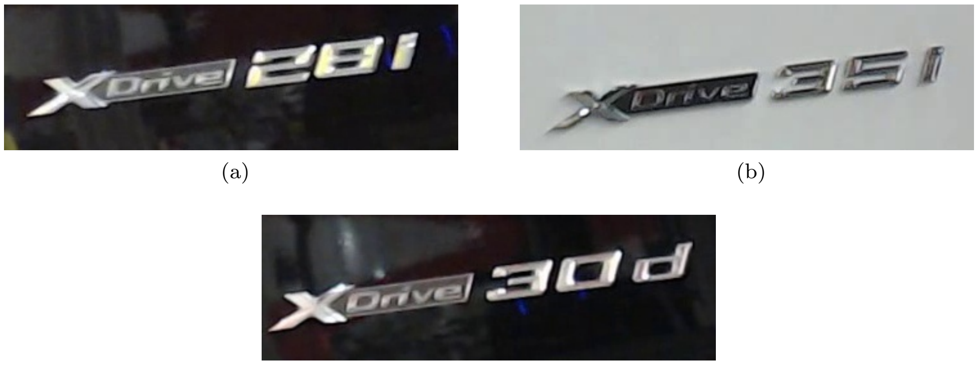


Figure 2.13: Some of the class templates from the door line of the BMW Spartanburg plant.

Chapter 3

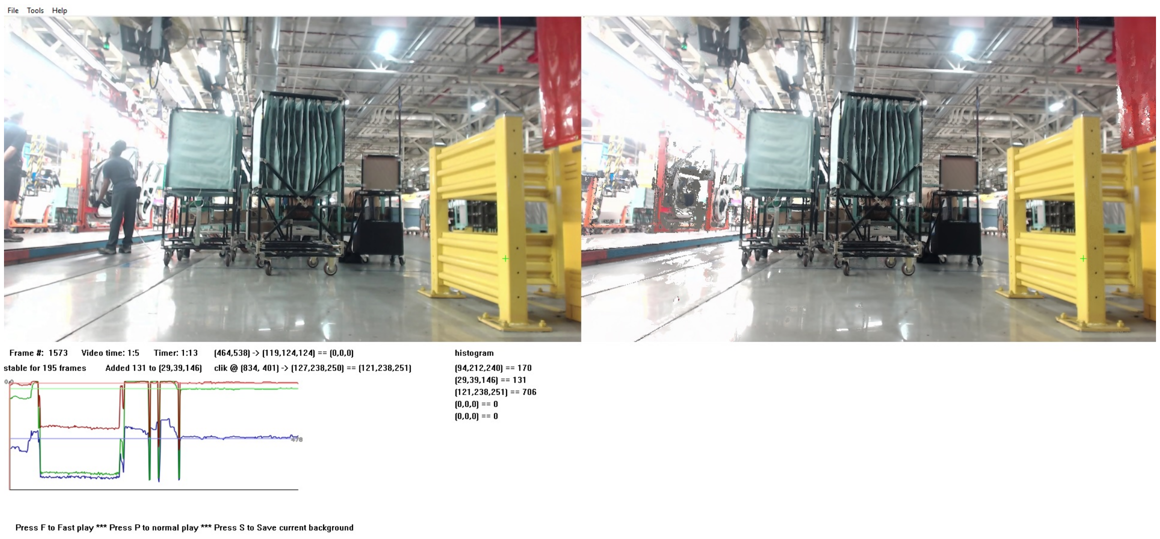
Results

This chapter describes the results of implementing the steps described in chapter 2. The process is implemented and tested on the simulation testbed described in section 2.3 and the videos recorded at the BMW Spartanburg plant.

3.1 The background extraction program and results

The background extraction program was developed using WinAPI to analyze the pixels in the video frame and to display the background image being generated. The program requires the user to load the video from which a background needs to be extracted. Figures 3.1 and 3.2 show screen grabs of the background extraction program at frame numbers 1573 and 2385 respectively of a video recorded at the door line of the BMW Spartanburg plant. The sub-image on the left is the frame from the video currently being analyzed and the sub-image on the right is the generated background at that point in the video.

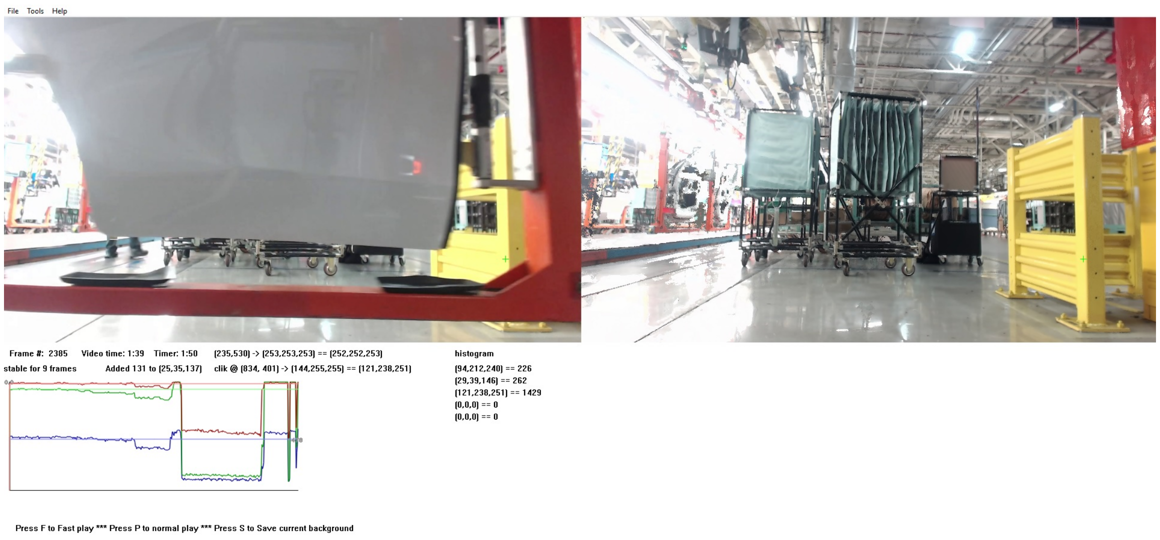
Figures 3.3 and 3.4 show a closer look at the graph plot of the figures 3.1 and 3.2 respectively. The figures have been modified from figures 3.1 and 3.2 to better observe the data. The plots show the intensity of a pixel in the frame against the frame number. The pixel being tracked is highlighted by a green cross-hair in both the current video frame on the left and the background image on the right. The color of a plot line indicates the color channel being tracked. The darker shades of the colors track the current intensity of the pixel in the video frame, while the lighter shades of the color show the current intensity of the corresponding pixel in the background image. To track the



```
Frame #:  1573    Video time: 1:5    Timer: 1:13    (464,538) -> (119,124,124) == (0,0,0)            histogram
stable for 195 frames    Added 131 to (29,39,146)    clik @ (834, 401) -> (127,238,250) == (121,238,251)    (94,212,240) == 170
                                                                                                           (29,39,146) == 131
                                                                                                           (121,238,251) == 706
                                                                                                           (0,0,0) == 0
                                                                                                           (0,0,0) == 0

Press F to Fast play *** Press P to normal play *** Press S to Save current background
```

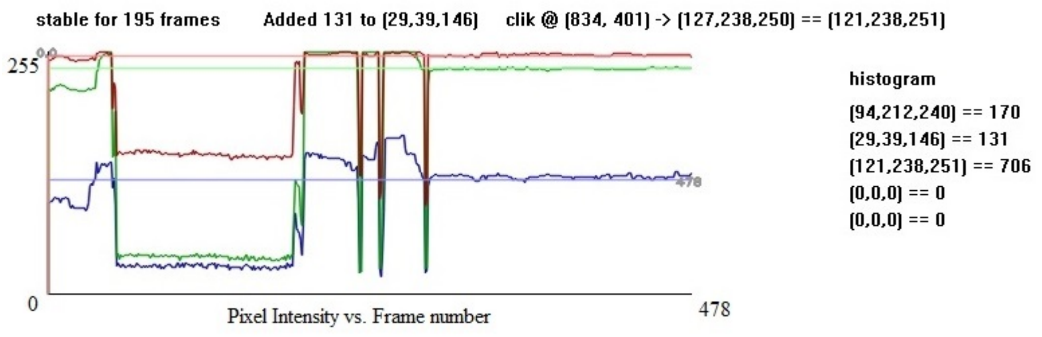
Figure 3.1: A screen grab of the background extraction program running videos recorded at the door line of the BMW plant.



```
Frame #:  2385    Video time: 1:39    Timer: 1:50    (235,530) -> (253,253,253) == (252,252,253)       histogram
stable for 9 frames    Added 131 to (25,35,137)    clik @ (834, 401) -> (144,255,255) == (121,238,251)    (94,212,240) == 226
                                                                                                           (29,39,146) == 262
                                                                                                           (121,238,251) == 1429
                                                                                                           (0,0,0) == 0
                                                                                                           (0,0,0) == 0

Press F to Fast play *** Press P to normal play *** Press S to Save current background
```

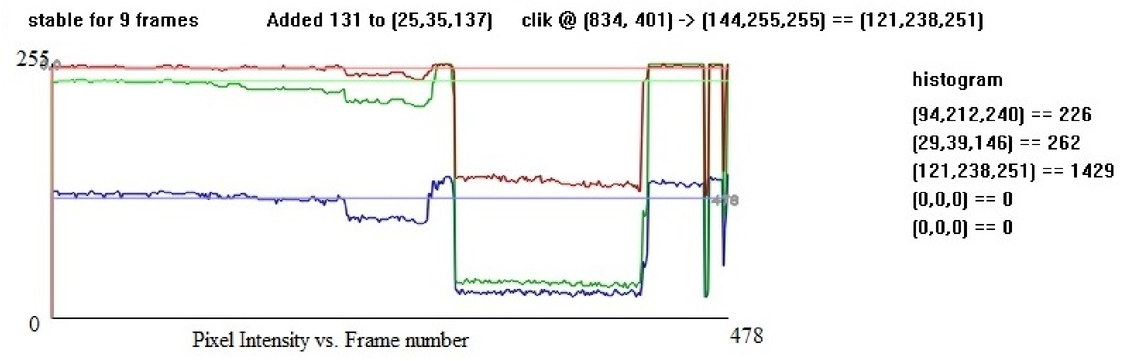
Figure 3.2: Another screen grab of the background extraction program running videos recorded at the door line of the BMW plant.



```
stable for 195 frames        Added 131 to (29,39,146)     clik @ (834, 401) -> (127,238,250) == (121,238,251)

histogram
(94,212,240) == 170
(29,39,146) == 131
(121,238,251) == 706
(0,0,0) == 0
(0,0,0) == 0
```

Figure 3.3: A closer look at the plot in figure 3.1.



```
stable for 9 frames          Added 131 to (25,35,137)     clik @ (834, 401) -> (144,255,255) == (121,238,251)

histogram
(94,212,240) == 226
(29,39,146) == 262
(121,238,251) == 1429
(0,0,0) == 0
(0,0,0) == 0
```

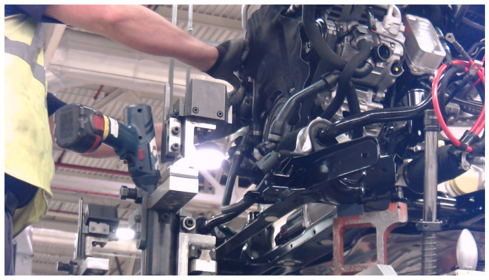
Figure 3.4: A closer look at the plot in figure 3.2.

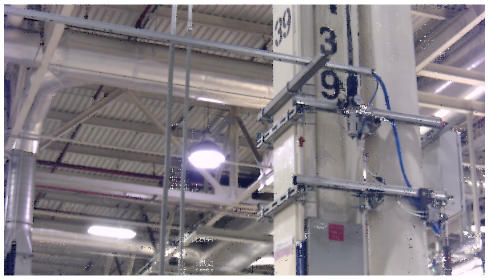
intensity of a pixel, the user can select any pixel in the image with a mouse click. The program shows which pixel is being tracked, the number of frames it has been stable for, the current pixel intensity (in BGR format) of the pixel in the video frame, the pixel intensity at its corresponding location in the background image, and the last addition to the histogram along with the histogram created for that pixel location. In figure 3.4, the pixel at (834, 401) is being tracked. The pixel has a current pixel intensity of (144, 255, 255) in the video frame and an intensity of (121, 238, 251) in the background image. This intensity value has been stable for 9 frames and the last update to the histogram was a value of 131 to (25, 35, 137). The value of 131 is added to (29, 39, 146) in the histogram. This is due to the set tolerance value of 15, that is, any pixel intensity with a value that is ±15 of (29, 39, 146) in any channel would be entered into the histogram bucket of (29, 39, 146). (29, 39, 146) was chosen as a candidate for the background intensity for pixel (834, 401) in an earlier iteration of algorithm 2 from section 2.2.1.

The program saves a background image at 1, 5, 10, 20 and 30 minutes of video time and at the end of the video, as in figure 3.5. Alternatively, the user can save the current background image at any instant by hitting the 'S' key or selecting the option from the menu. This is to allow the user to visually choose a suitable background image. Figure 3.6 shows a frame of a video and the background generated from that video.

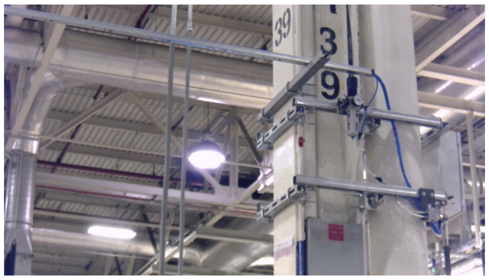
Sub figure (g) in figure 3.6 shows several black regions where a good background pixel was not found. This is because the pixel at that location was not stable for more that 40 frames, which is the set threshold to be considered a candidate for the background pixel intensity. Figure 3.7 shows a plot of the pixel belonging to this region. It is clear that the pixel intensity does not stabilize and hence, the histogram for that pixel location is unable to acquire a candidate intensity for its corresponding location in the background. The cause for the oscillation in the video was a loose wire. The background extraction algorithm implemented here is used despite this shortcoming due to its simplicity. Also, the background image is used only to extract and match features to find the affine transformation. Since the features are to be matched and only the best matches are to be used, these oscillations are ignored. Additionally, the background extraction algorithm is sensitive to illumination changes. This is also not an issue since the feature detection algorithm used here, ORB, is invariant to illumination changes.

The user runs the background extraction program on the video recorded before and after the camera was bumped. Figures 3.8 and 3.9 show the before and after bump background images at


(a) A frame of a video


(b) The background generated after 1 minute of video

(c) The background after 5 minutes of video


(d) The background after 10 minutes of video

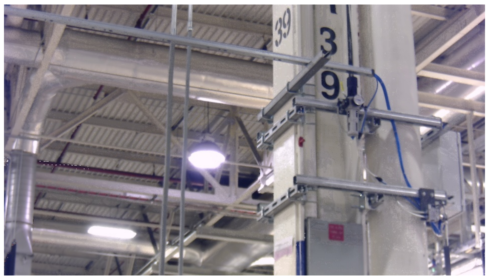
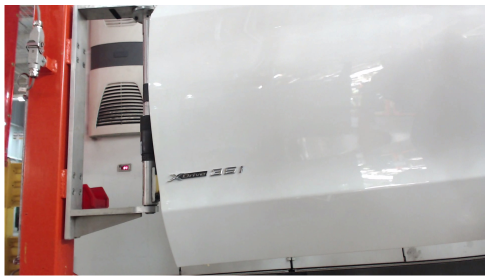
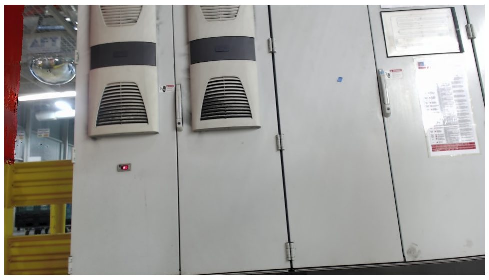
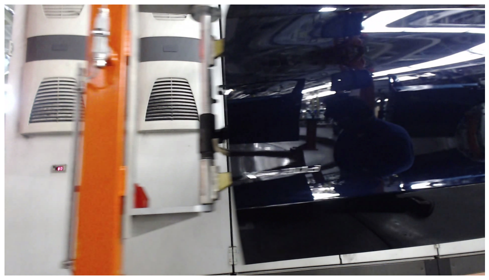
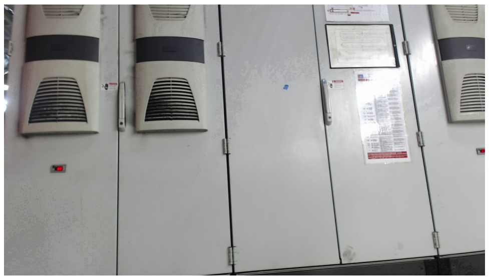

(e) The background at the end of the video

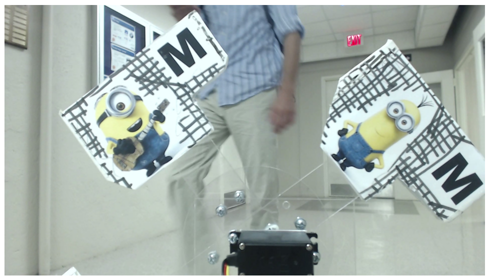
Figure 3.5: The background images generated at the engine line of the BMW plant.

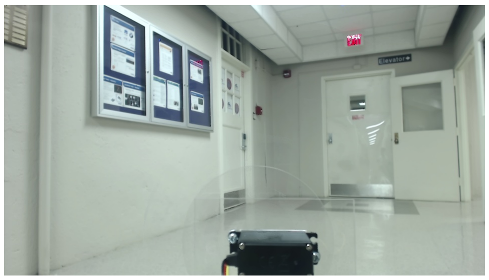

(a) A video frame

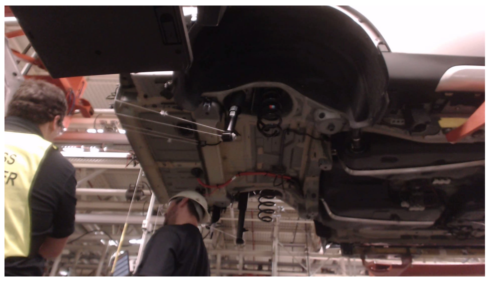

(b) The background generated


(c) A video frame


(d) The background generated


(e) A video frame


(f) The background generated


(g) A video frame

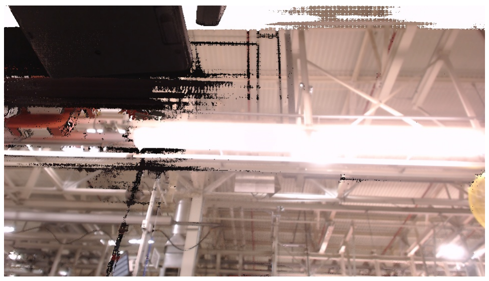

(h) The background generated

Figure 3.6: Backgrounds generated at various locations in the BMW plant and on the simulation testbed.

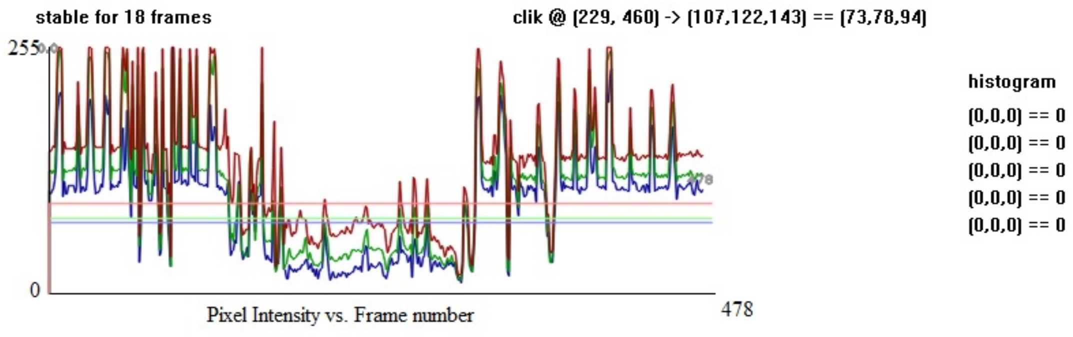


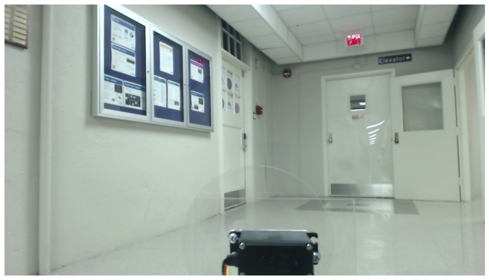
Figure 3.7: The plot of the pixel at (102, 530) in figure 3.6 (g) at which an intensity value is not stable for more than 40 frames, which is the set threshold.

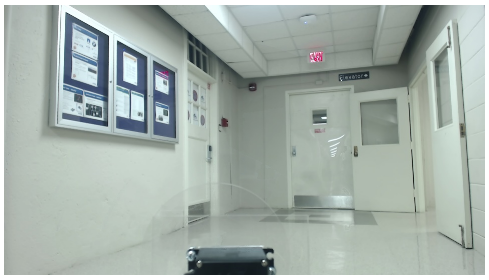
Video name	length of video	Number of objects	Time to generate good background
engine line	33:33 min	21	10 min
spring line	1:02:37 hr	42	30 min
door line	7:00 min	4	5 min
door line pos 1	5:36 min	4	5 min
door line pos 2	7:10 min	4	5 min
door line pos 3	7:37 min	5	5 min
door line pos 4	5:36 min	4	5 min
simulation pos 1	6:50 min	20	5 min
simulation pos 2	5:12 min	15	5 min
simulation pos 3	5:13 min	15	5 min
simulation pos 4	5:04 min	12	5 min
simulation pos 5	5:05 min	13	5 min

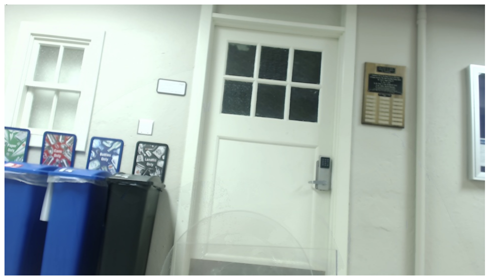
Table 3.1: Time taken by the background extraction program to generate a good background image for various test videos.

several locations. Since it was not feasible to simulate a camera bump at every location at the BMW plant, only the camera at the door line was bumped. The other background images are generated from the simulation testbed.

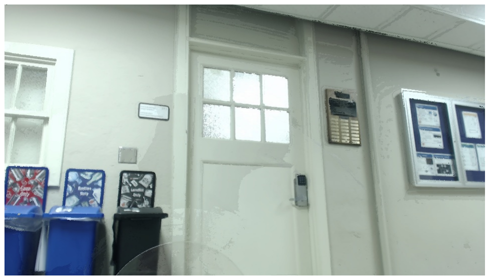
Table 3.1 shows the time taken by the background extraction program to generate a good background image for various test videos. From the table it is clear that the background extraction generally takes 5-10 minutes to generate a good background image. In the case of the spring line video however, due to slow moving cars on the assembly line which would occupy almost the entire frame of the video, as shown in figure 3.6 (g), it required 30 minutes to generate a satisfactory background image as shown in figure 3.6 (h).


(a) Before camera bump


(b) After camera bump


(c) Before camera bump

(d) After camera bump


(e) Before camera bump

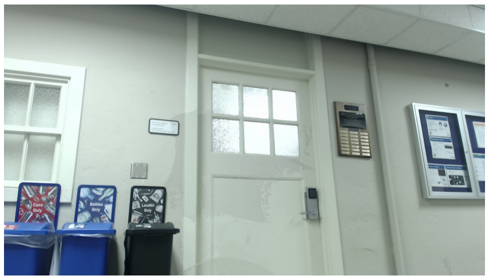
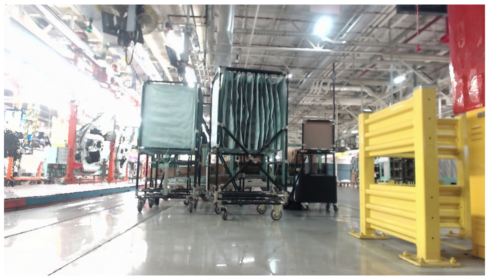

(f) After camera bump

Figure 3.8: Before- and after-bump background images.

(a) Before camera bump

(b) After camera bump


(c) Before camera bump

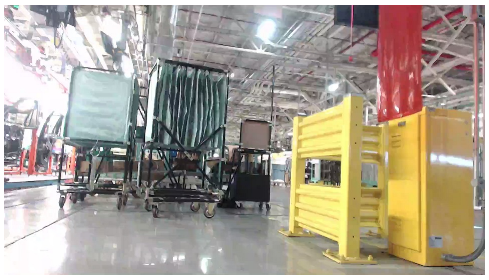

(d) After camera bump

Figure 3.9: Additional before- and after-bump background images.

3.2 The feature matching program, affine transformation and results

After the user generates the after bump background image, the two background images are used as inputs to the feature matching program. The feature matching program, also designed using WinAPI, is used to find the affine transformation between the pre- and post-bump background images using feature tracking and matching. The program requires the user to load the two background images and then finds the best affine transformation matrix that transforms the pre-bump background image to the post-bump background image. The transformation from the pre-bump to post-bump is found, as opposed to post- to pre-bump, because the template images created by VI are associated with the pre-bump video data. Thus, in order for the templates to be usable after the bump, they are transformed in this directions. Another solution to correcting for bumps, instead of transforming template images, would be to affine transform every frame from the new position of the camera to the pre-bump camera position. Thus the original templates can continue to be used. However, transforming every frame of the video would be very computationally intensive and so is not preferred.

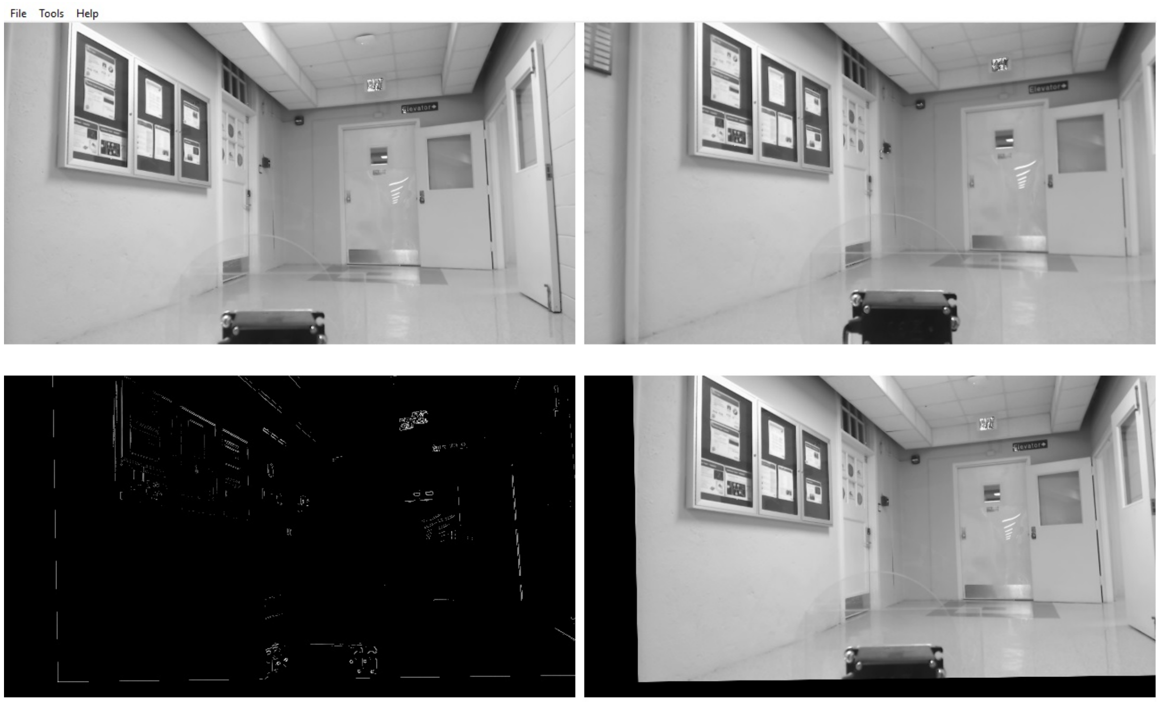


Figure 3.10: A screen grab of the feature matcher program.

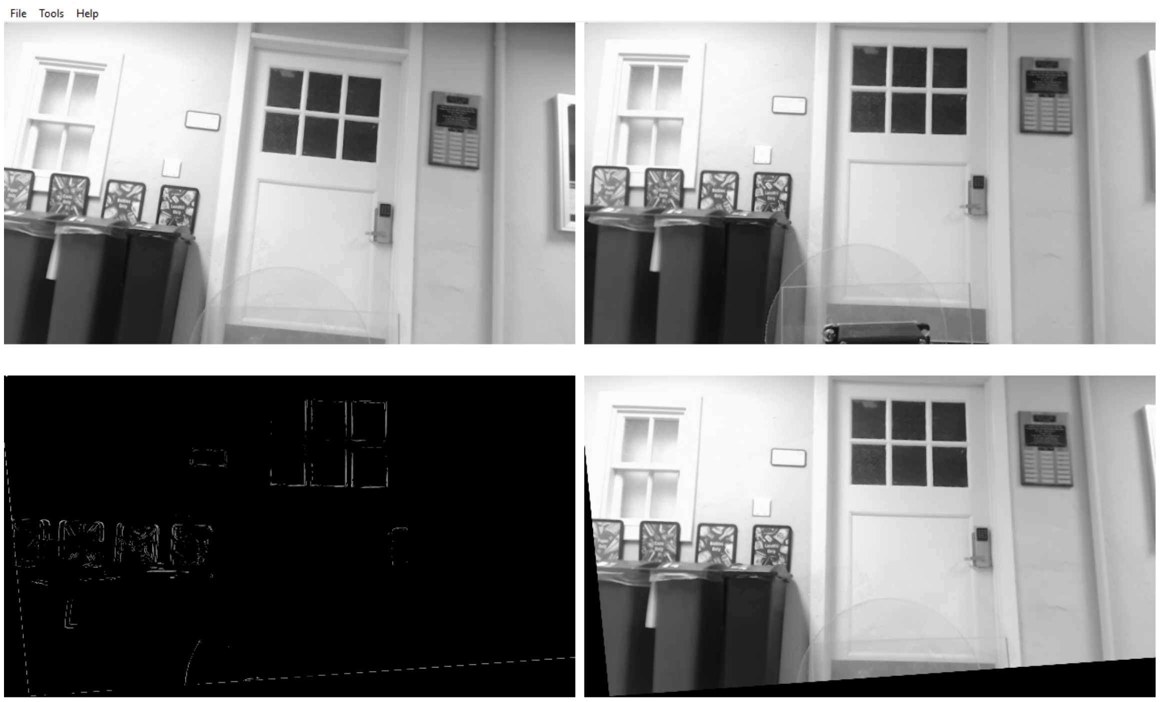


Figure 3.11: Another screen grab of the feature matcher program.

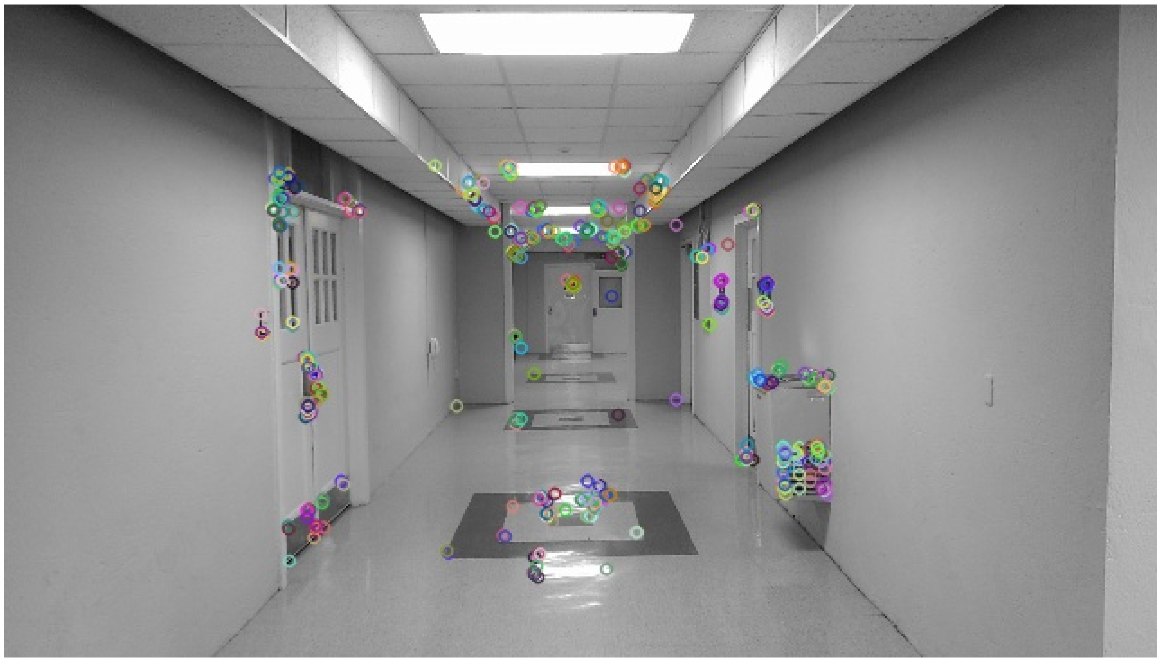


Figure 3.12: ORB keypoints detected in the before-bump image.

Figure 3.10 and 3.11 show screen grabs of the feature matcher program. The feature detection algorithm, ORB, requires grayscale images and so the background images are converted to grayscale. The subimage on the top-left is the pre-bump image, the subimage on the top-right is the post-bump image, the subimage on the bottom-right is the result of transforming the pre-bump image to the post-bump image, and the subimage on the bottom-left is the similarity image of the gradient magnitude of the post-bump image and the resulting image of applying the affine transformation on the pre-bump background image.

The transformation is calculated by first detecting the features or keypoints in the input images and then by finding the best matches between their keypoints. Figure 3.12 and 3.13 show the ORB keypoints detected in the pre- and post-bump background images respectively and figure 3.14 shows the matching keypoints found using the BFMatcher, the brute force matching algorithm implemented by OpenCV. For better visibility, figure 3.15 shows the 15 best keypoint matches. The best matches are found by sorting the matched keypoints based on the Hamming distance measured between the two keypoint descriptors. Figure 3.16 shows another example of keypoint detection and matching applied on background images from the BMW plant.

The best 3 keypoint matches are found using the algorithm described in section 2.2.3 and

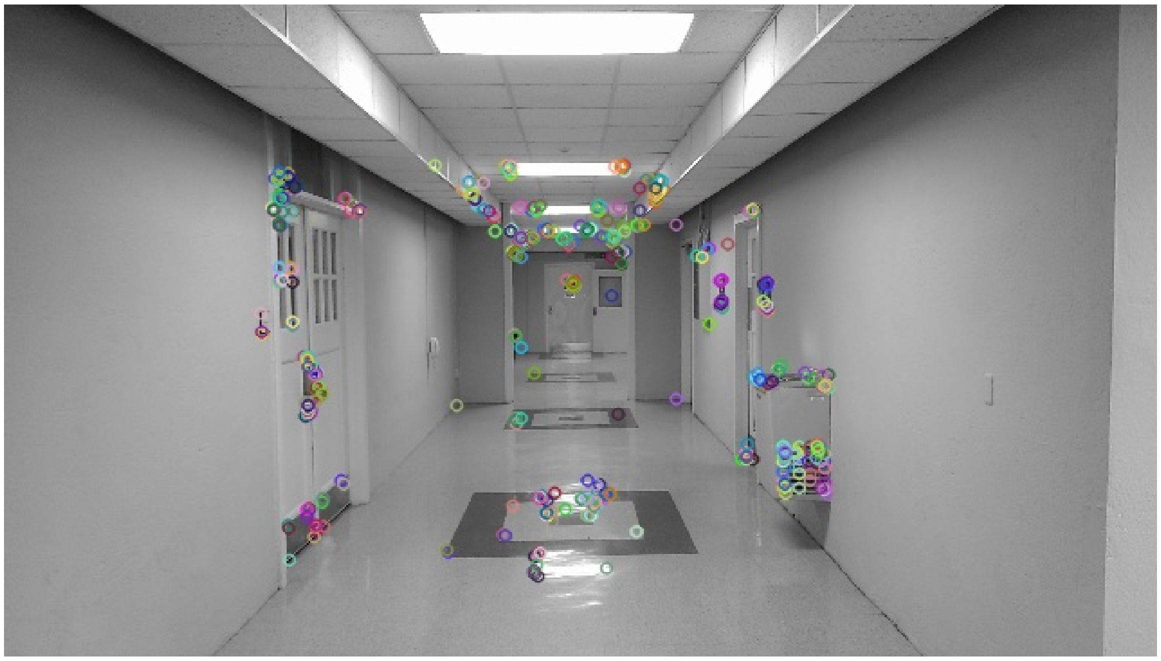


Figure 3.13: ORB keypoints detected in the after-bump image.

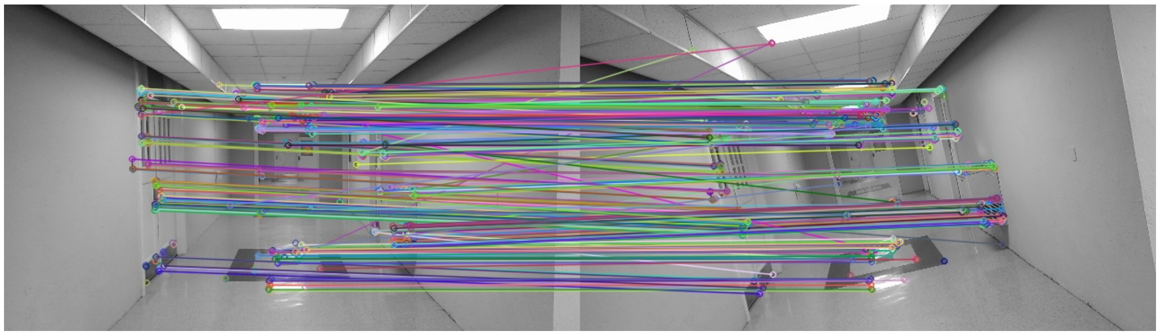


Figure 3.14: All the keypoint matches between figures 3.12 and 3.13.

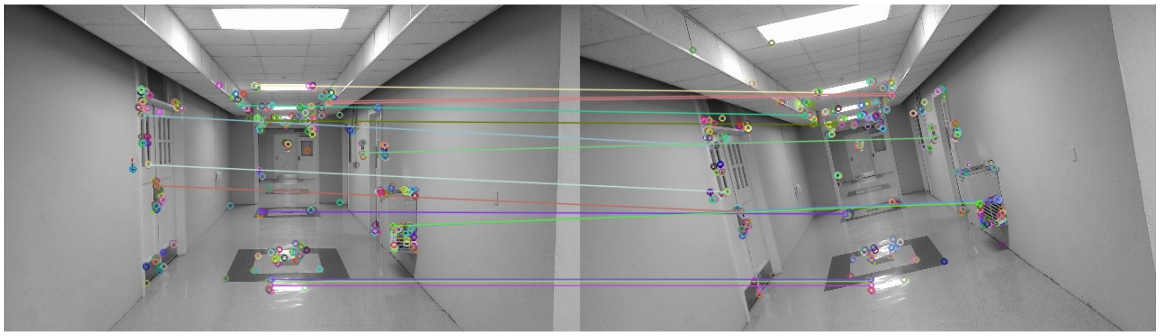
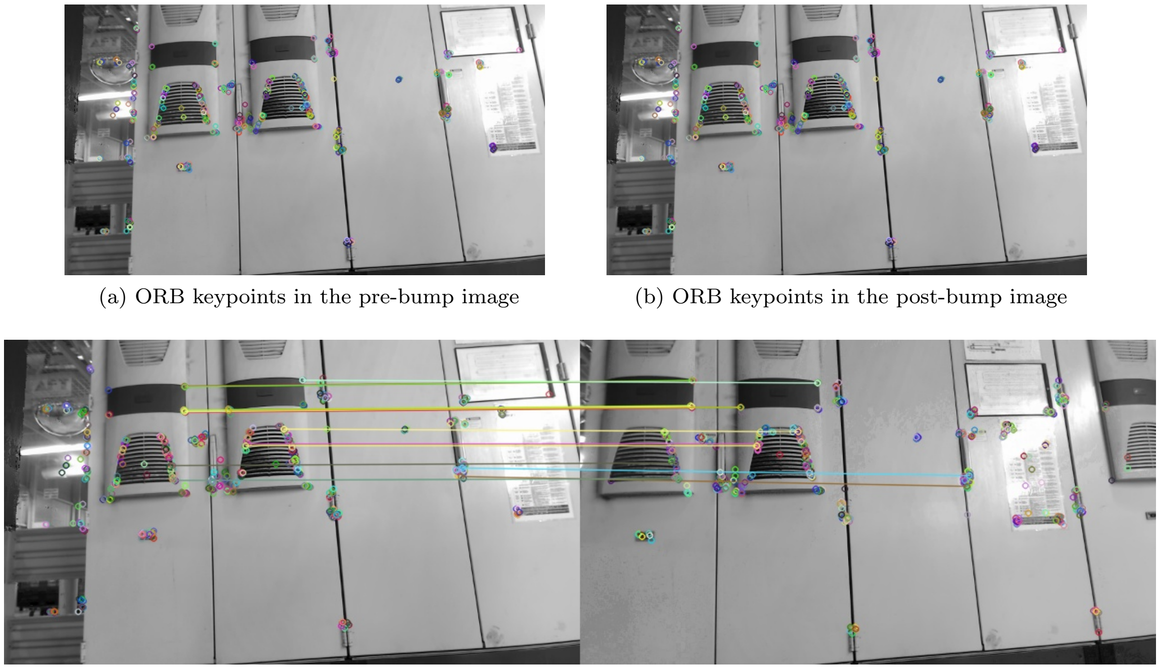


Figure 3.15: The 15 best keypoint matches between figures 3.12 and 3.13



(a) ORB keypoints in the pre-bump image

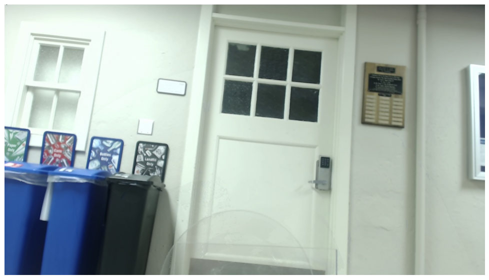
(b) ORB keypoints in the post-bump image

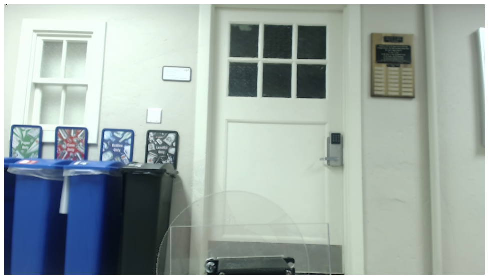
(c) The 15 best keypoint matches

Figure 3.16: Another example of the keypoint detection and matching applied on background images from the BMW plant.

is used to calculate the affine transformation matrix. This transformation is then applied to the pre-bump background image. Figures 3.17 and 3.18, show the result of applying the transformation on several extreme examples. Figure 3.20 shows a more realistic situation where the user has tried to reposition the camera as close as possible to its original orientation. Since the algorithm uses the RANSAC algorithm to find the best keypoint matches, it is quite possible that the user may have to run the feature matching program multiple times to manually and visually find the best match. The transformation or warp results in a black border around the images, as seen in figure 3.21, which affects the template matching algorithm. To solve this, mask images are generated which are binary images displaying a zero value when image data is present and a high value for a black pixel, as seen in figure 3.21. VI is modified to subtract the mask image from the template image before performing template matching.

Once the user finds the best transformation, the transformation is applied to all the templates. Figures 3.21 shows the original template, the warped or transformed template and its corresponding mask image for various examples. As discussed in section 2.2.3, the templates are only


(a) Before camera bump


(b) After camera bump

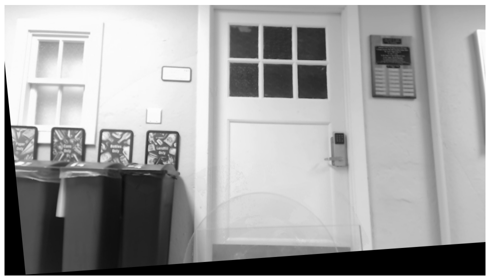
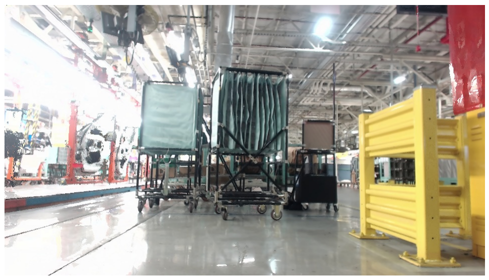
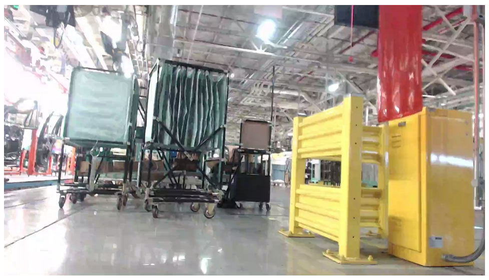

(c) First image transformed to second

Figure 3.17: A transformation of images from the simulation testbed.


(a) Before camera bump


(b) After camera bump

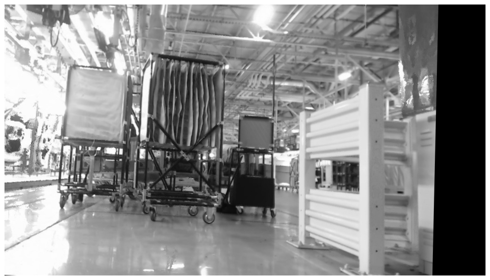
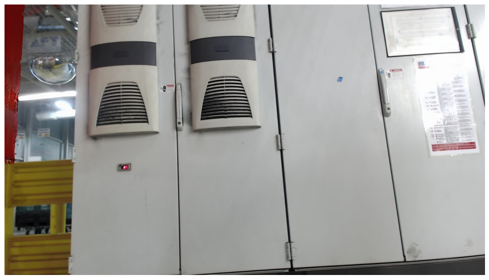
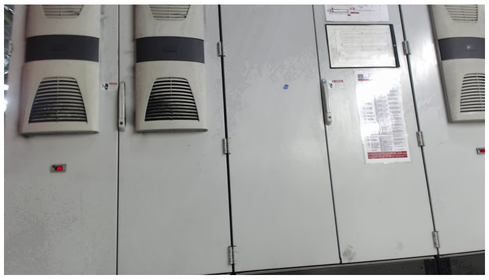

(c) First image transformed to second

Figure 3.18: A transformation of images from the BMW plant.


(a) Before camera bump


(b) After camera bump

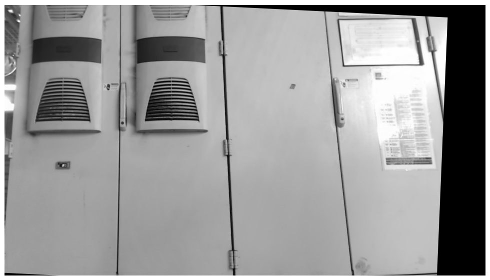
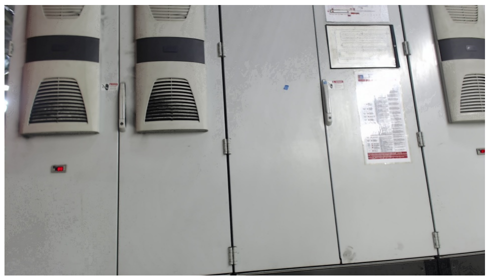
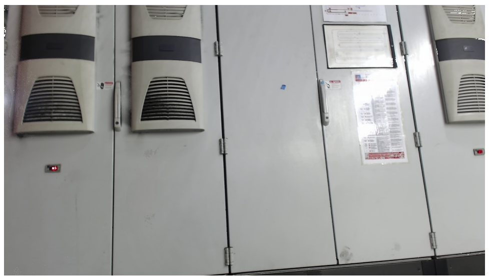

(c) First image transformed to second

Figure 3.19: Another transformation of images from the BMW plant.


(a) Before camera bump


(b) After camera bump

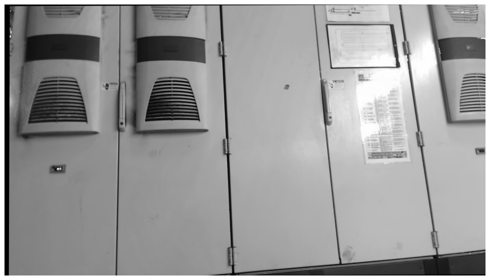

(c) First image transformed to second

Figure 3.20: A transformation of images from the BMW plant that is more likely to happen.

transformed by rotation and scaling and not by translation. Whereas the inspection windows are transformed by translation alone. Figures 3.22 and 3.23 show the effect of applying the affine transformation on templates of the simulation testbed and the door line of the BMW Spartanburg plant.

3.3 The manual compensation program

Once the templates have been transformed, the user runs the manual compensation program. The program, in turn, runs the Setup program and lets the user select the new location of the trigger search window. The difference in the updated window location and the old location is the transformation to be applied to the other search windows. Figure 3.24 shows a screen grab of Setup program showing the old location of the trigger search window (within the solid green box) and the new location (within the dotted green box). Additionally, the user is required to save a frame of the video showing the updated trigger search window enclosing the trigger. This is used to calculate the scalefactor to be applied to all the templates.

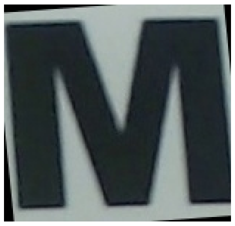
3.4 VI results

After the templates have been transformed the user runs VI to test the warped templates on the video recorded after the bump. The following are the results of implementing the bump compensation algorithm on the simulation testbed and videos from the door line of the BMW plant.

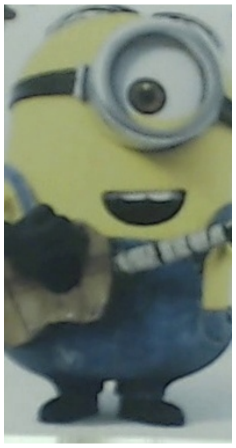
3.4.1 On the simulation testbed

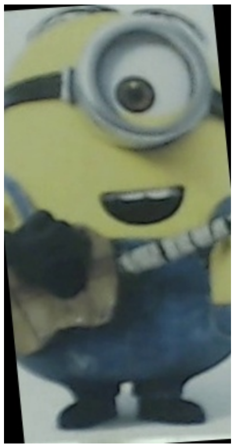
Table 3.2 shows the result of VI before the camera was bumped on the simulation testbed. The table shows the actual parts in the frame, whether VI was triggered and VI's classification result. The original templates shown in figure 3.25 are used for this. The table shows a 100% detection accuracy by VI. Figure 3.26 shows a successful classification by VI. The camera is then bumped and VI is run on the video recorded after the bump while still using the original templates. Figure 3.17 shows the degree of the bump. Table 3.3 shows the result of this which indicates a lowered accuracy of 75%. Figure 3.27 shows that the trigger does not match well with the trigger search window and thus VI does not trigger. This could be due to a camera rotation or that the camera moving closer to the foreground, resulting in scaling, both of which cause the current templates to be unusable.

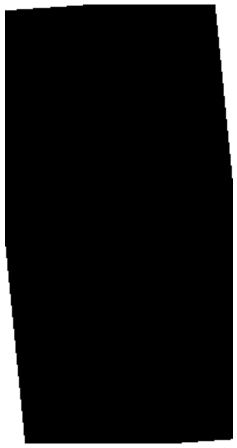
(a) Original template


(b) Warped template

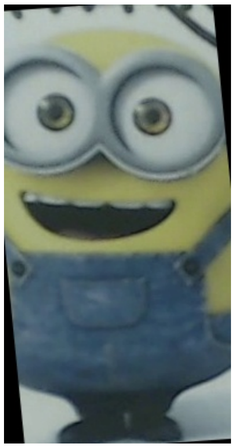
(c) Mask image


(d) Original template


(e) Warped template


(f) Mask image

(g) Original template


(h) Warped template

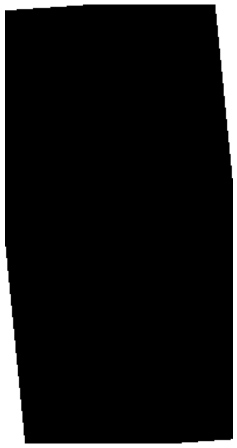
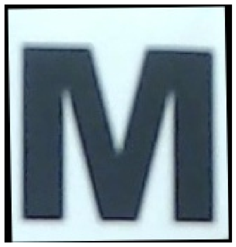

(i) Mask image

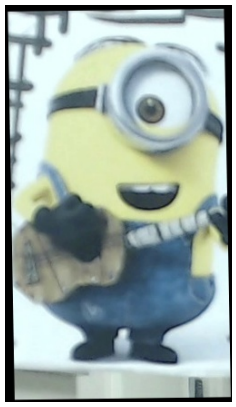
Figure 3.21: The effect of applying the affine transformation found in figure 3.17 on template images at the simulation testbed.

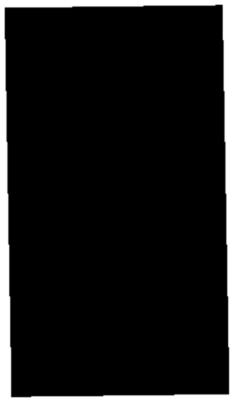
(a) Original template

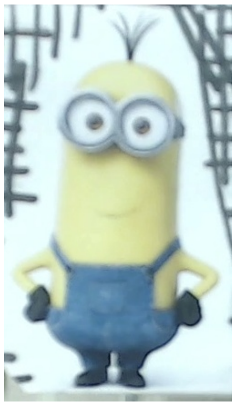

(b) Warped template

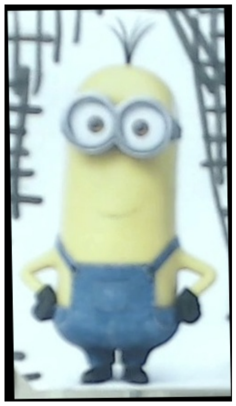
(c) Mask image

(d) Original template


(e) Warped template


(f) Mask image


(g) Original template


(h) Warped template

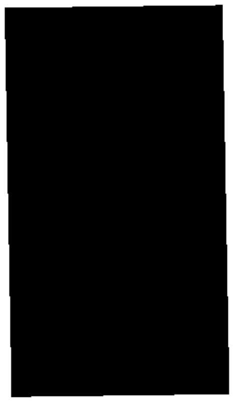
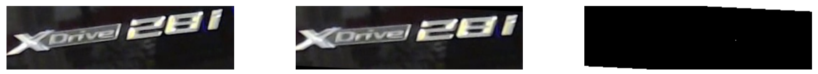

(i) Mask image

Figure 3.22: The effect of applying the affine transformation found in figure 3.10 on template images at the simulation testbed.

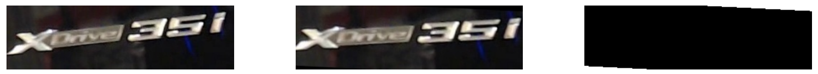
(a) Original template (b) Warped template (c) Mask image



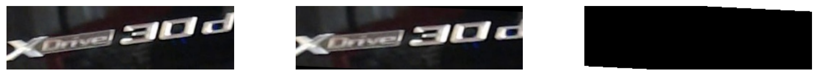
(d) Original template (e) Warped template (f) Mask image

(g) Original template (h) Warped template (i) Mask image

(j) Original template (k) Warped template (l) Mask image

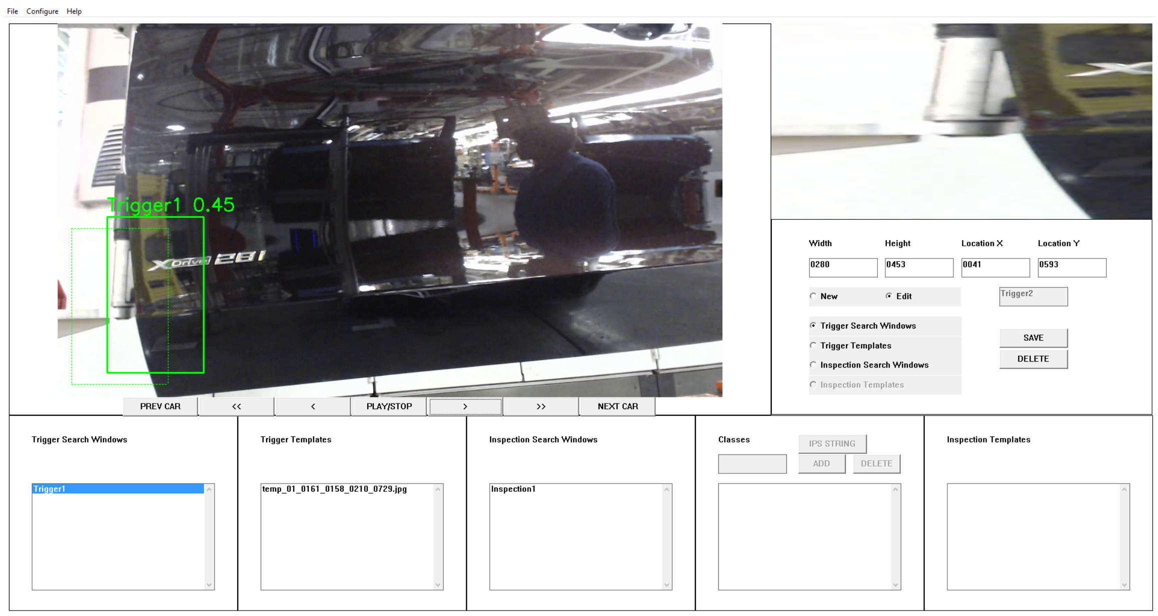


(m) Original template (n) Warped template (o) Mask image



(p) Original template (q) Warped template (r) Mask image

Figure 3.23: The effect of applying the affine transformation found in figure 3.19 on template images of the door line the BMW plant.



Width: 0280
Height: 0453
Location X: 0041
Location Y: 0593

New [✓ Edit]
Trigger2

[✓ Trigger Search Windows]
Trigger Templates
Inspection Search Windows
Inspection Templates

Trigger Search Windows: Trigger1
Trigger Templates: temp_01_0161_0158_0210_0729.jpg
Inspection Search Windows: Inspection1
Classes
Inspection Templates

Figure 3.24: A screen grab of Setup program showing the old and new location of the trigger search window.

The user runs the background extraction program, the feature matching program and the manual compensation program which generates the new warped templates. VI is then run using these modified templates and table 3.4 shows the results. Figure 3.25 shows the warped templates and its corresponding mask images, and figure 3.28 shows a successful classification by VI. The table 3.4 shows a 100% detection accuracy by VI, which is an improvement from 75% after the camera was bumped.

3.4.2 On the door line

Table 3.5 shows the result of VI before the camera was bumped at the door line of the BMW plant. The table shows the actual parts in the frame, whether VI was triggered and VI's classification result. The original templates shown in figure 3.29 are used for this. The table shows a 100% detection accuracy by VI. Figure 3.30 shows a successful classification by VI. The camera is then bumped and VI is run on the video recorded after the bump while still using the original templates. Figure 3.19 shows the degree of the camera bump. Table 3.6 shows the result of VI on the post-bump video and using the original templates and indicates a reduced identification accuracy of 0%. Figure 3.31 shows that the trigger no longer appears within the trigger search

Actual object	Triggered	Classified as
kevin	yes	kevin
bob	yes	bob
stuart	yes	stuart
bob	yes	bob
kevin	yes	kevin
bob	yes	bob
stuart	yes	stuart
bob	yes	bob
kevin	yes	kevin
bob	yes	bob
stuart	yes	stuart
bob	yes	bob

Table 3.2: VI result before camera bump and using original templates on the simulation testbed that gives 100% accuracy.

Actual object	Triggered	Classified as
stuart	no	NA
bob	yes	bob
kevin	yes	kevin
bob	yes	bob
stuart	no	NA
bob	yes	bob
kevin	yes	kevin
bob	yes	bob
stuart	no	NA
bob	yes	bob
kevin	yes	kevin
bob	yes	bob

Table 3.3: VI result after camera bump and using original templates on the simulation testbed that gives a reduced 75% accuracy.

Actual object	Triggered	Classified as
stuart	yes	stuart
bob	yes	bob
kevin	yes	kevin
bob	yes	bob
stuart	yes	stuart
bob	yes	bob
kevin	yes	kevin
bob	yes	bob
stuart	yes	stuart
bob	yes	bob
kevin	yes	kevin
bob	yes	bob

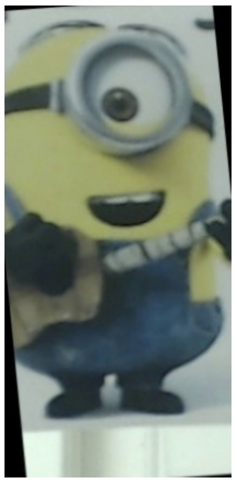
Table 3.4: VI result after camera bump and using the warped templates on the simulation testbed which gives an improved 100% accuracy.

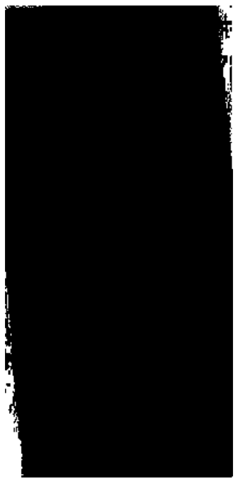
(a) Original template

(b) Warped template

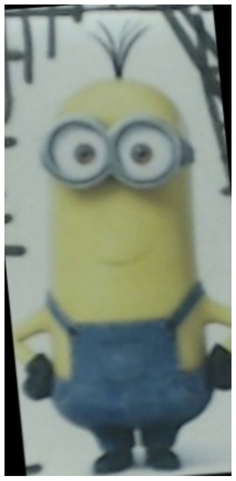
(c) Mask image

(d) Original template


(e) Warped template


(f) Mask image

(g) Original template


(h) Warped template

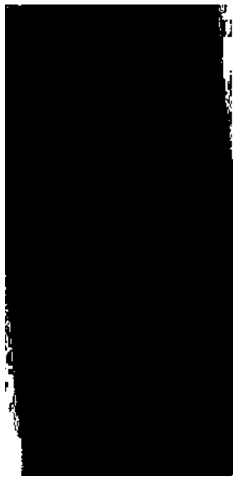

(i) Mask image

Figure 3.25: The templates used to test bump detection and compensation on the simulation testbed.

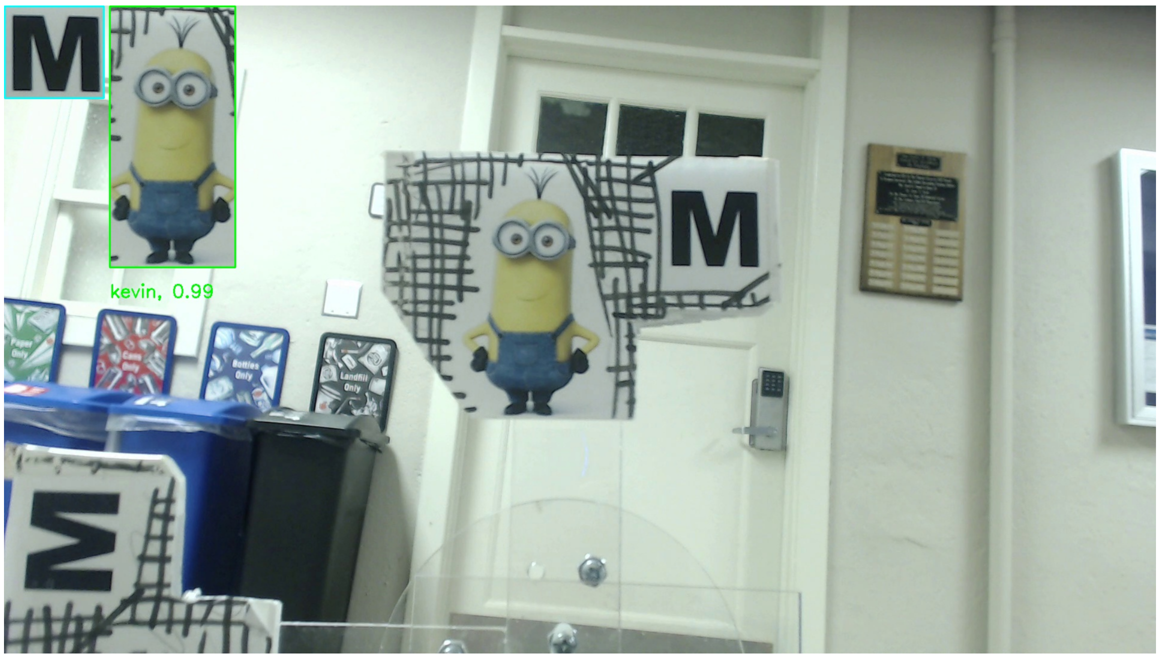


Figure 3.26: A successful classification by VI on the pre-bump simulation testbed video.

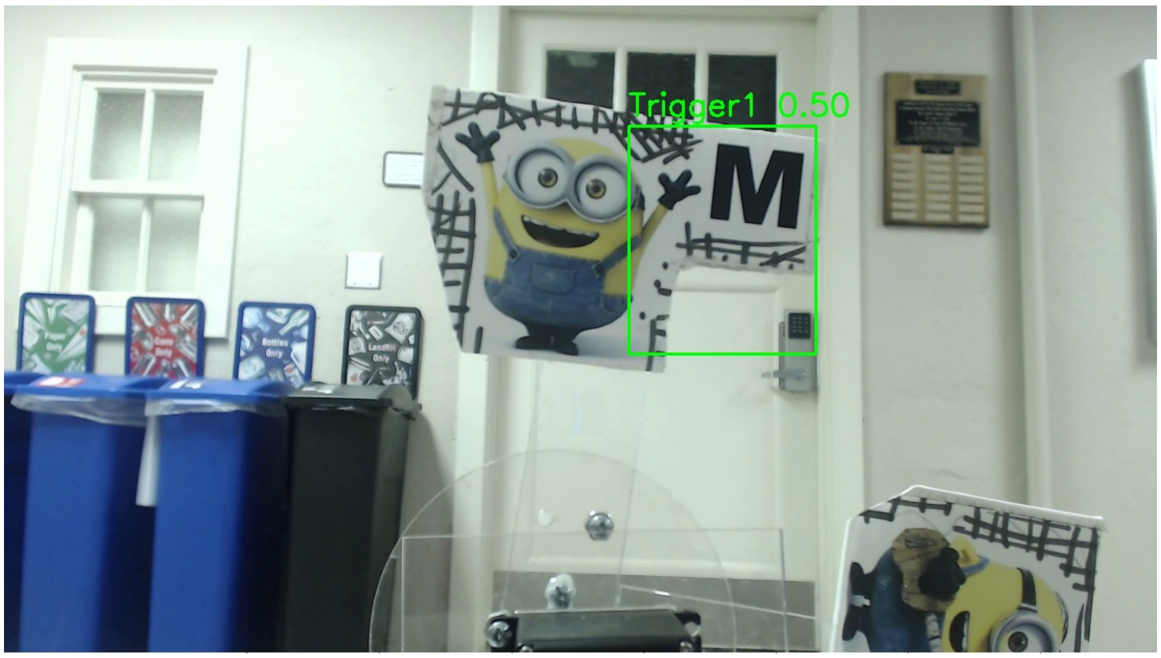


Figure 3.27: A screen grab of the setup program running the post-bump video with the original templates of the simulation testbed. The trigger templates do not have a successful match with the trigger search window.

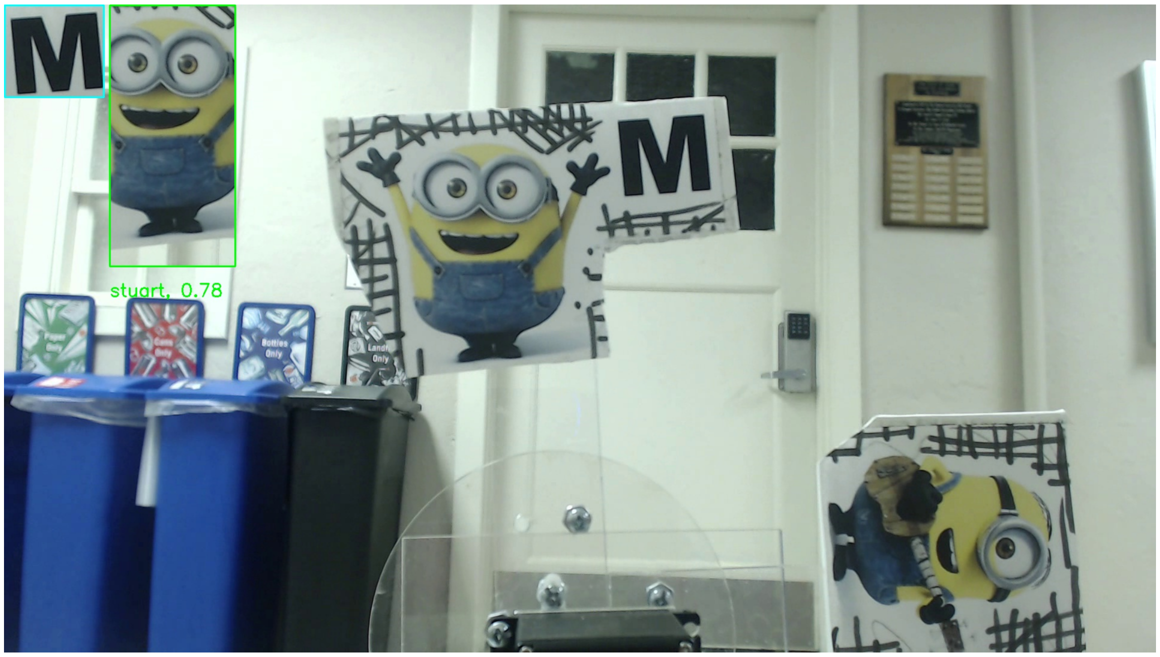


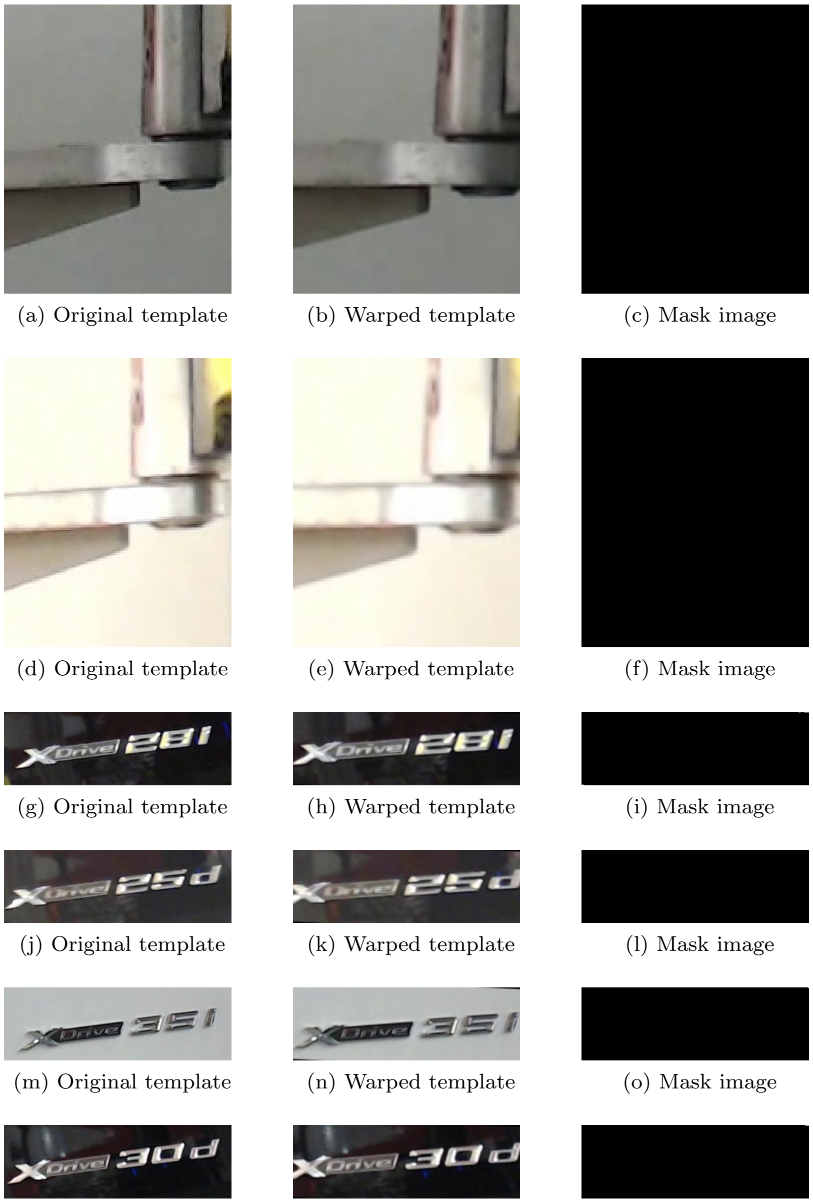
Figure 3.28: A successful classification by VI using the warped templates on the simulation testbed.

Actual object	Triggered	Classified as
xDrive35i	yes	xDrive35i
xDrive30d	yes	xDrive30d
xDrive35i	yes	xDrive35i
xDrive25d	yes	xDrive25d
xDrive28i	yes	xDrive28i

Table 3.5: VI result before camera bump and using original templates on the door line.

window, thus resulting in VI not triggering. The user runs the background extraction program, the feature matching program and the manual compensation program which generates the new warped templates. VI is then run using these modified templates and table 3.7 shows the results. Figure 3.29 shows the warped templates and figure 3.32 shows a successful classification by VI. The table 3.7 shows an improved 100% detection accuracy by VI.

Table 3.8 shows the performance evaluation of the bump correction algorithm applied on several videos recorded at the BMW plant and on the simulation testbed. For video ‘simulation pos 2’, on the simulation testbed, it is seen that VI continued to work with 100% accuracy after the small bump. Thus, the user was able to re-position the camera well enough for the system to not be updated with corrected templates. For video ‘simulation pos 5’, on the simulation testbed, the bump correction was unable to bring VI to a 100% accuracy from 0% after the large bump.



(a) Original template (b) Warped template (c) Mask image

(d) Original template (e) Warped template (f) Mask image

(g) Original template (h) Warped template (i) Mask image

(j) Original template (k) Warped template (l) Mask image

(m) Original template (n) Warped template (o) Mask image

(p) Original template (q) Warped template (r) Mask image

Figure 3.29: The templates used to test bump detection and compensation on the door line of the BMW plant.

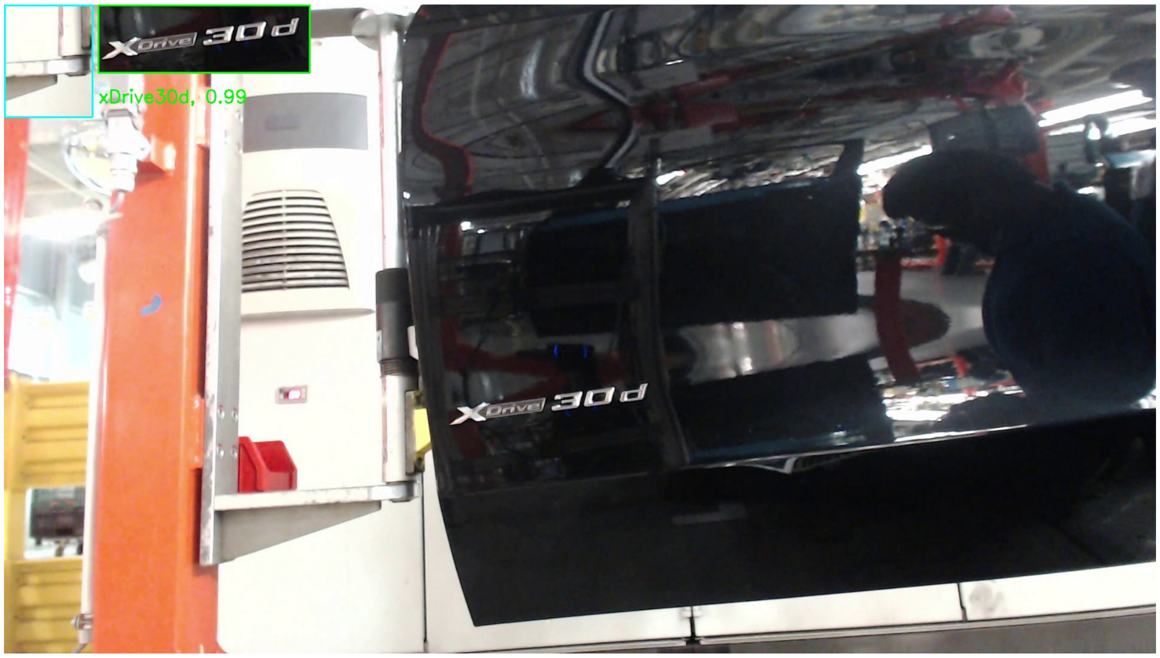


Figure 3.30: A successful classification by VI on the pre-bump door line video.

Actual object	Triggered	Classified as
xDrive28i	no	NA
xDrive30d	no	NA
xDrive35i	no	NA
xDrive28i	no	NA
xDrive28i	no	NA
xDrive28i	no	NA

Table 3.6: VI result after camera bump and using original templates on the door line.

Actual object	Triggered	Classified as
xDrive28i	yes	xDrive28i
xDrive30d	yes	xDrive30d
xDrive35i	yes	xDrive35i
xDrive28i	yes	xDrive28i
xDrive28i	yes	xDrive28i
xDrive28i	yes	xDrive28i

Table 3.7: VI result after camera bump and using the warped templates on the door line.

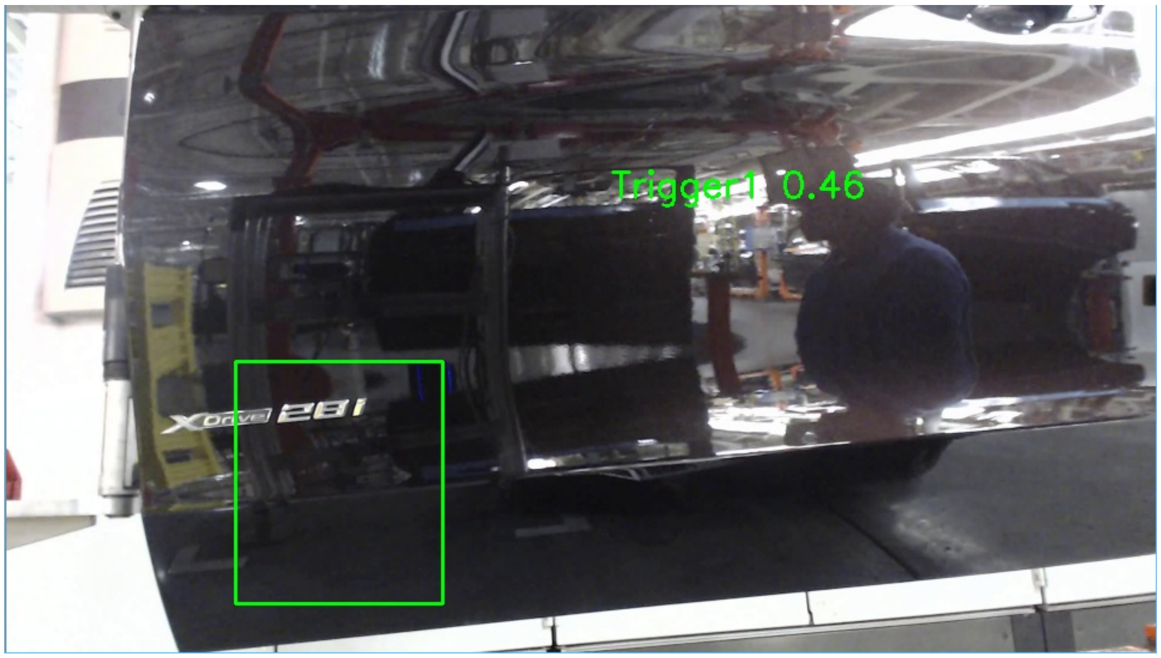


Figure 3.31: A screen grab of the setup program running the post-bump video with the original templates on the door line. The template does not appear within the trigger search window.

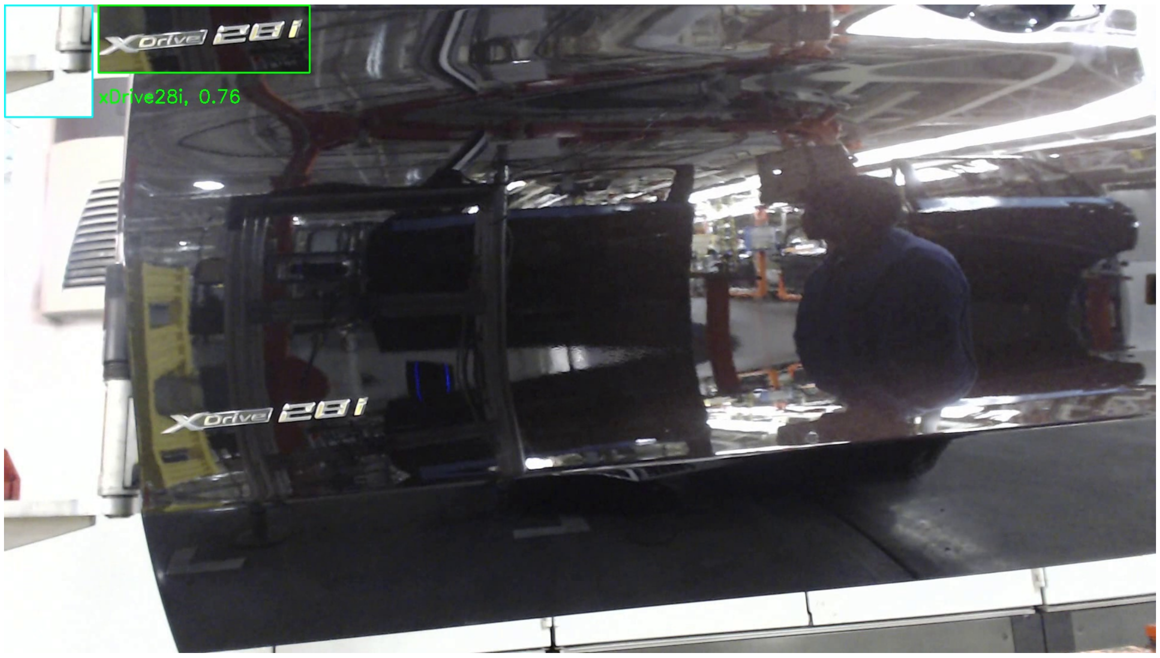


Figure 3.32: A successful classification by VI using the warped templates on the post-bump door line video.

Video	Duration	Number of objects	Condition	VI accuracy before correction	VI accuracy after correction
simulation pos 2	5:12 min	15	small bump	100%	100%
simulation pos 3	5:13 min	15	large bump	75%	100%
simulation pos 5	5:05 min	13	large bump	0%	66.67%
door line pos 2	7:10 min	4	large bump	0%	100%

Table 3.8: The performance evaluation of the bump correction algorithm applied on several videos recorded at the BMW plant and on the simulation testbed.

On closer analysis, it was observed that this is due to skewing in the foreground. Skewing causes unequal scaling in different regions of the frame, resulting in different scale factors for the trigger template and for the class template. A correction for this was found by calculating the scale factor for the class template which resulted in the 66.67% accuracy shown here. However, further work needs to be done to find a better solution that brings VI to a 100% accuracy.

Chapter 4

Conclusions and Further Work

The bump detection and compensation method suggested in this thesis was successfully implemented on the videos recorded at the BMW Spartanburg manufacturing plant and on the simulation testbed. Further work can be done to improve the efficiency of this system.

For example, the background extraction algorithm used here is based on the premise that the most frequently stable pixel intensities are likely a part of the background image. However, there are regions where the pixel intensity does not remain stable for more than the threshold value of 40 frames as seen in figure 3.7. Such pixels can create false features in the image which is detrimental to the method suggested here. A confidence image can be used as a mask so that features are not detected in these regions. Figure 4.1 shows a sample confidence image for figure 3.9 (c). The confidence image is a binary image which has high values at locations which have a good histogram and low values when the histogram does not indicate a clear candidate for a pixel intensity at that location in the background image.

Additionally, as see in figure 3.6 (g) and its plot in figure 3.7, the background extraction algorithm is unable to acquire a candidate for certain regions in the image as the pixel intensity does not remain stable for more than 40 frames. To avoid such black regions, an adaptive threshold may be used at every pixel location so that best possible candidate is chosen as the background pixel.

Depending on the camera location, it is possible that the background image does not contain sufficient texture to find features. One way to create texture in such situations would be by adding a contrasting structure in the background. Alternatively, a more sensitive feature detection algorithm can be used.

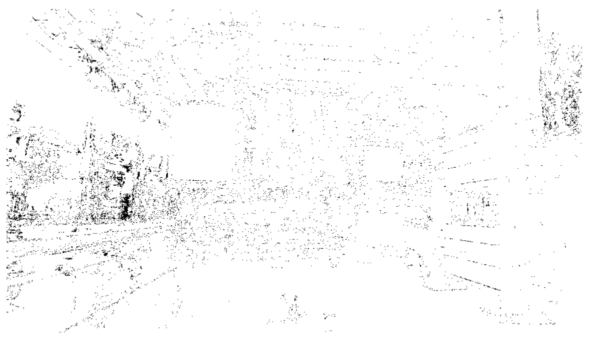


Figure 4.1: A confidence image generated for figure 3.9 (c)

In extreme cases, it is possible that when the camera is very close to the foreground and the camera is bumped severely, the foreground is skewed. A solution to this problem could be to track more known objects in the image. The variation in their relative sizes can help determine the skewing in the image.

In conclusion, a camera bump detection and compensation method was described and implemented successfully on various test cases from the simulation testbed and videos recorded at the BMW Spartanburg plant.

Bibliography

[1] M. Tomono, “Robust 3d slam with a stereo camera based on an edge-point icp algorithm,” in *Robotics and Automation*, 2009, pp. 4306–4311.

[2] S. Cheung and C. Kamath, “Robust techniques for background subtraction in urban traffic video,” International Society for Optics and Photonics, pp. 881–892, 2004.

[3] L. Maddalena and A. Petrosino, “A self-organizing approach to background subtraction for visual surveillance applications,” *Image Processing*, vol. 17, no. 7, pp. 1168–1177, 2008.

[4] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” in *Computer Vision*, 2006, pp. 430–443.

[5] F. Cohen, Z. Fan, and S. Attali, “Automated inspection of textile fabrics using textural models,” *Pattern Analysis and Machine Intelligence*, vol. 13, no. 8, pp. 803–808, 1991.

[6] A. Crispin and V. Rankov, “Automated inspection of pcb components using a genetic algorithm template-matching approach,” *The International Journal of Advanced Manufacturing Technology*, vol. 35, no. 3-4, pp. 293–300, 2007.

[7] J. Schulte and A. Hoover, “Visual inspector user manual,” *Clemson University*, 2015.

[8] Y. Matsushita, E. Ofek, W. Ge, X. Tang, and H. Shum, “Full-frame video stabilization with motion inpainting,” *Pattern Analysis and Machine Intelligence*, vol. 28, no. 7, pp. 1150–1163, 2006.

[9] J. Yang, D. Schonfeld, and M. Mohamed, “Robust video stabilization based on particle filter tracking of projected camera motion,” *Circuits and Systems for Video Technology*, vol. 19, no. 7, pp. 945–954, 2009.

[10] A. Davison, I. Reid, N. Molton, and O. Stasse, “Monoslam: Real-time single camera slam,” *Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[11] A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, “Kinectfusion: Real-time dense surface mapping and tracking,” in *Mixed and Augmented Reality*, 2011, pp. 127–136.

[12] A. Lai, H. Yoon, and G. Lee, “Robust background extraction scheme using histogram-wise for real-time tracking in urban traffic video,” in *Computer and Information Technology*, 2008, pp. 845–850.

[13] C. Chiu, M. Ku, and L. Liang, “A robust object segmentation system using a probability-based background extraction algorithm,” *Circuits and Systems for Video Technology*, vol. 20, no. 4, pp. 518–528, 2010.

[14] E. Durucan and T. Ebrahimi, "Change detection and background extraction by linear algebra," *Proceedings of the IEEE*, vol. 89, no. 10, pp. 1368–1381, 2001.

[15] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15, 1988, p. 50.

[16] D. Lowe, "Object recognition from local scale-invariant features," in *Computer Vision*, vol. 2, 1999, pp. 1150–1157.

[17] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," in *Computer Vision*, 2011, pp. 2564–2571.

[18] S. Leutenegger, M. Chli, and R. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *Computer Vision*, 2011, pp. 2548–2555.

[19] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," *Computer Vision*, pp. 778–792, 2010.

[20] A. Fischler and C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.