# **Computer Vision Decoration program flow**

## **Step 1: Identify blocks on the conveyor (“Main1\_DetectBlocksOnConveyor”)**

This function Identified whether blocks exists on the conveyor to be picked up. This step did not detect shape or colour of the block, hence if blocks with letters were placed on the conveyor, it was detected. The program flow was as follows:

1. The camera on top of the conveyor was calibrated with the function below. Definition of the function has been included in Appendix A.

CameraCalibrationConveyor;

1. Image was read into the file and calibrated based on the parameters calculated from the previous function.

conveyorImg = imread('Proper\_Pics\conveyor5.jpg');

conveyorImg = undistortImage(conveyorImg,...

cameraParamsConveyor);

1. A colour mask was applied to the image to filter out the blocks from the background of the colour (the definition of the mask function has been included in appendix B). The result was inverted (to have the blocks appear as white and background as black) and segmented to only include the section of the conveyor reachable by the IRB-120. Additional image processing methods were also applied to remove any white noises from the image.

[BW,maskedRGBImage] = createConveyorMask3(conveyorImg);

BW = ~BW;

BW = segmentSection(BW, 555, 1155, 10, 586);

BW = bwareaopen(BW,2000); % remove white noise

BW = bwmorph(BW, 'hbreak');

1. The centroid and area of the blocks detected were calculated using “regionprops”. If blocks were detected, any areas larger or smaller than the average blocks were removed (potentially came from additional white noise) and the remaining centroids were converted to real life coordinates and returned (definition of “conveyorPxlToReal” function has been included in Appendix C). If no blocks were detected, the code simply returns a message indicating and returns an empty array, indicated no detection of blocks.

% use regionprops to calculate centroids

blockProps = regionprops(BW, ‘Centroid’,’Area’);

centroids = vertcat(blockProps.Centroid);

areas = vertcat(blockProps.Area);

if size(centroids,1) > 0

removeIdx = find(areas > 7000 | areas < 1000);

areas(removeIdx) = [];

centroids(removeIdx,:) = [];

% convert to real life

conveyorCentroids = conveyorPxlToReal(centroids(:,1), centroids(:,2));

figure();

imshow(BW); hold on;

plot(centroids(:,1), centroids(:,2), ‘c\*’, ‘MarkerSize’, 10);

else

display(“No blocks detected on the conveyor”);

conveyorCentroids = [];

end

## **Step 2: Identify the block picked up (“Main2\_ IdentifyBlock”)**

This function

## **Appendix A**

% Auto-generated by cameraCalibrator app on 13-Aug-2019

%-------------------------------------------------------

% Define images to process

imageFileNames = {‘D:\Kevinly Santoso\Documents\UNI\y4s2\MTRN4230\Assignments\Assignment 2\Conveyor\_Calibration\conveyor\_\_08\_07\_18\_46\_52.jpg’,...

‘D:\Kevinly Santoso\Documents\UNI\y4s2\MTRN4230\Assignments\Assignment 2\Conveyor\_Calibration\conveyor\_\_08\_07\_18\_46\_57.jpg’,...

‘D:\Kevinly Santoso\Documents\UNI\y4s2\MTRN4230\Assignments\Assignment 2\Conveyor\_Calibration\conveyor\_\_08\_07\_18\_47\_04.jpg’,...

‘D:\Kevinly Santoso\Documents\UNI\y4s2\MTRN4230\Assignments\Assignment 2\Conveyor\_Calibration\conveyor\_\_08\_07\_18\_47\_17.jpg’,...

‘D:\Kevinly Santoso\Documents\UNI\y4s2\MTRN4230\Assignments\Assignment 2\Conveyor\_Calibration\conveyor\_\_08\_07\_18\_47\_21.jpg’,...

‘D:\Kevinly Santoso\Documents\UNI\y4s2\MTRN4230\Assignments\Assignment 2\Conveyor\_Calibration\conveyor\_\_08\_07\_18\_47\_24.jpg’,...

‘D:\Kevinly Santoso\Documents\UNI\y4s2\MTRN4230\Assignments\Assignment 2\Conveyor\_Calibration\conveyor\_\_08\_07\_18\_47\_33.jpg’,...

‘D:\Kevinly Santoso\Documents\UNI\y4s2\MTRN4230\Assignments\Assignment 2\Conveyor\_Calibration\conveyor\_\_08\_07\_18\_47\_36.jpg’,...

‘D:\Kevinly Santoso\Documents\UNI\y4s2\MTRN4230\Assignments\Assignment 2\Conveyor\_Calibration\conveyor\_\_08\_07\_18\_48\_11.jpg’,...

‘D:\Kevinly Santoso\Documents\UNI\y4s2\MTRN4230\Assignments\Assignment 2\Conveyor\_Calibration\conveyor\_\_08\_07\_18\_48\_17.jpg’,...

};

% Detect checkerboards in images

[imagePoints, boardSize, imagesUsed] = detectCheckerboardPoints(imageFileNames);

imageFileNames = imageFileNames(imagesUsed);

% Read the first image to obtain image size

originalImage = imread(imageFileNames{1});

[mrows, ncols, ~] = size(originalImage);

% Generate world coordinates of the corners of the squares

squareSize = 25; % in units of ‘millimeters’

worldPoints = generateCheckerboardPoints(boardSize, squareSize);

% Calibrate the camera

[cameraParamsConveyor, imagesUsed, estimationErrors] = estimateCameraParameters(imagePoints, worldPoints, ...

‘EstimateSkew’, false, ‘EstimateTangentialDistortion’, false, ...

‘NumRadialDistortionCoefficients’, 2, ‘WorldUnits’, ‘millimeters’, ...

‘InitialIntrinsicMatrix’, [], ‘InitialRadialDistortion’, [], ...

‘ImageSize’, [mrows, ncols]);

% View reprojection errors

%h1=figure; showReprojectionErrors(cameraParamsConveyor);

% Visualize pattern locations

%h2=figure; showExtrinsics(cameraParamsConveyor, ‘CameraCentric’);

% Display parameter estimation errors

displayErrors(estimationErrors, cameraParamsConveyor);

% For example, you can use the calibration data to remove effects of lens distortion.

undistortedImage = undistortImage(originalImage, cameraParamsConveyor);

% See additional examples of how to use the calibration data. At the prompt type:

% showdemo(‘MeasuringPlanarObjectsExample’)

% showdemo(‘StructureFromMotionExample’)

## **Appendix B**

function [BW,maskedRGBImage] = createConveyorMask3(RGB)

%createMask Threshold RGB image using auto-generated code from colorThresholder app.

% [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using

% auto-generated code from the colorThresholder app. The colorspace and

% range for each channel of the colorspace were set within the app. The

% segmentation mask is returned in BW, and a composite of the mask and

% original RGB images is returned in maskedRGBImage.

% Auto-generated by colorThresholder app on 20-Aug-2019

%------------------------------------------------------

% Convert RGB image to chosen color space

I = rgb2ycbcr(RGB);

% Define thresholds for channel 1 based on histogram settings

channel1Min = 34.000;

channel1Max = 65.000;

% Define thresholds for channel 2 based on histogram settings

channel2Min = 110.000;

channel2Max = 125.000;

% Define thresholds for channel 3 based on histogram settings

channel3Min = 134.000;

channel3Max = 150.000;

% Create mask based on chosen histogram thresholds

sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...

(I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...

(I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);

BW = sliderBW;

% Initialize output masked image based on input image.

maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.

maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

end

## **Appendix C**

Using 4 coordinates that were already known both in pixel world and real world, a formula was calculated to linearly convert from a pixel coordinate to a real world coordinate.

function realPts = conveyorPxlToReal(xPxl, yPxl)

%% Calibrate the conveyor camera

C2\_world = [20, 146]; % bottom left

C3\_world = [-270, 151]; % top left

C4\_world = [-266, 667]; % top right

C5\_world = [24, 677]; % bottpm right

C2\_pxl = [485, 541];

C3\_pxl = [500, 159];

C4\_pxl = [1181, 165];

C5\_pxl = [1188, 549];

Mx = (C2\_world(1)-C3\_world(1))/(C2\_pxl(2) - C3\_pxl(2));

xReal = Mx\*yPxl - Mx\*C2\_pxl(2) + C2\_world(1);

My = (C5\_world(2)-C3\_world(2))/(C5\_pxl(1)-C3\_pxl(1));

yReal = My\*xPxl - My\*C3\_pxl(1) + C3\_world(2);

display('This is the point you converted to the real world (conveyor): ');

realPts = [xReal, yReal, ones(size(xReal,1),1)\*22.1];

end