개발완료보고서 [스마트 팩토리]

1. 팀 정보

팀명	TEAM IM	팀장	서경철
팀원	김성재	팀원	김태영
팀원	박정대	팀원	선준형

2. 개발완료보고서

0. 작품명 SMART MLCC Cutting Inspection

1. 개요

1.1. 작품 개요

- '전자산업의 쌀' 이라고 불리는 MLCC(적층세라믹콘덴서)는 회로에 일정량의 전류가 흐르도록 제어해주는 전자 제품의 필수 부품이다. 스마트폰에는 약 800~1200개가 들어가고, 초소형·고사양일수록 탑재량은 많아진다. 최근 5세대 이동통신(5G)의 확대를 기반으로 고사양 프리미엄 스마트폰 등 IT기기의 성능 발전 및 생산량 증가로 MLCC의 수요가 증가하고 있다. 뿐만 아니라 현재 MLCC 업계에서는 전기차와 자율주행차 시대를 대비한 자동차 전자장비용 MLCC 생산 능력 확충 경쟁 또한 치열하게 벌어지고 있다. 따라서 이를 제작하기 위한 MLCC 제조 공정의 속도 및 수율 향상에 대한 요구가 급증하고 있다.

그림1. 적층세라믹콘덴서(MLCC)의 크기

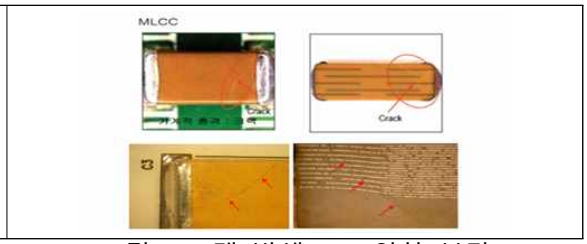


그림2. 크랙 발생으로 인한 불량

- MLCC의 가격은 매우 고가로 제조 공정에서 검사 성능은 MLCC의 품질을 결정하는데 매우 중요하다. MLCC의 검사 정밀도가 떨어지면 제품의 정전 용량에 이상이 생기거나 적층끼리 붙는 합선이 발생해 전자제품의 파손이 일어나므로 검사 수율은 매우 중요하다. MLCC 제조 공정에서 초소형 MLCC의 검사 성능을 높이기 위한 광학계 개발과 IoT 기술을 활용한 자동관리, 딥러닝을 통한 영상처리검사의 수율향상을 목표로 현재의 MLCC 양산 공정에 적용되지 않은 MLCC SHEET 커팅 후 자동영상처리 검사 장치, SMART MLCC Cutting Inspection를 개발하여 불량 검출에 따른 커팅공정 피드백과 전체 생산효율을 높이는 것이 목적이다.

1.2. 개발 목표

- MLCC 절단 후 품질 검사 장치 개발
- 이번 작품의 목표는 다음과 같다.

1) 다양한 크기의 MLCC SHEET를 자동으로 Align하기 위한 가변 생산 시스템 개발

MLCC SHEET를 컨베이어 벨트에 이동시킨 후 영상처리를 진행하려면 항상 일정한 정렬이 필요하다. MLCC를 SHEET 크기에 따라 정렬하기 위해서는 각 규격에 맞게끔 컨베이어 벨트의 좌·우에서 정렬 장치를 작동시켜야 한다. 이를 CATIA Platform 상에서 설계하고 가상 시뮬레이션을 통해 최적화한다.

2) Embedded System을 사용하여 메카트로닉스 장치 구현

작품을 구현하기 위해 각 모듈들은 지속적인 위치제어를 요구한다. 이를 자동으로 교정하는 Calibration 기법도 반드시 필요하다.

3) 영상처리를 통한 검사

영상처리를 위한 파라미터 검출 및 딥러닝을 통한 지속적인 보정을 목표로 한다.

4) 검사율과 검사 이미지를 서버로 전송하는 IoT 기술의 구현
5) 장비를 제작하는데 있어서 CATIA 3D EXPERIENCE PLATFORM을 활용한 DIGITAL TWINS 구현(가상공간에서 검사장비 설계, 제작, 운영 등의 일련과정을 사이버 공간상에서 시뮬레이션)을 통한 기술 개발 최적화

2. 개발 환경 설명

2.1. Hardware 구성

2.1.1. Conveyor Belt & Robot

1) Belt Pulley

벨트 풀리는 XL타입 벨트를 사용하기로 선정하였다. 개발 계획서를 작성할 때 계획했던 롤러형식이 아닌 풀리 형식을 채택해 진행하였다. 풀리 형식을 채택한 배경으로는 벨트의 슬립이 없다는 점이 정밀 기계를 만드는 과정에서 가장 중요하다고 생각했고, 베어링에 걸리는 부하가 적어 오랜 기간 동안 사용할 수 있기 때문이다.

풀리 잇수 = 풀리 지름 * 원주율 / 벨트 피치

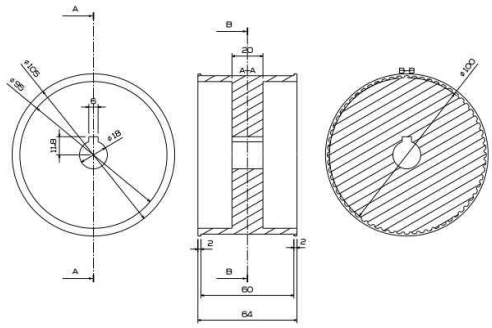


그림 3. Belt Pulley 설계도

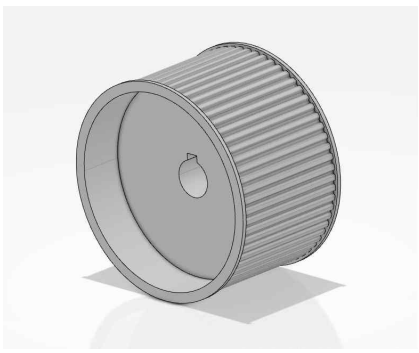


그림 4. Belt Pulley 3D 모델링

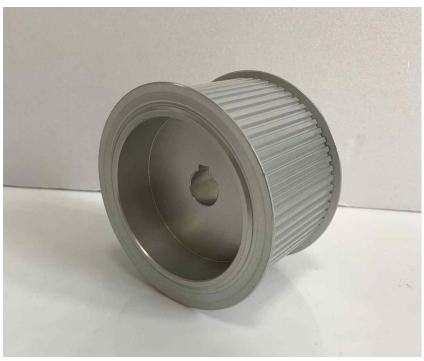


그림 5. Belt Pulley 실제 모습

2) Pulley Aix

풀리의 축을 설계하기 위해선 텐셔너, 스텝 모터 등의 규격 등을 고려해야한다. 이를 만족시키기 위해 양쪽 풀리 축의 원주를 다르게 설계하였다. 풀리와 축을 연결하기 위한 키 홈은 풀리 축 규격에 맞게 설계하였다. 또한 베어링에 끼워져 돌기 위해서 베어링 외륜에 축이 닿지 않도록 설계하였다.

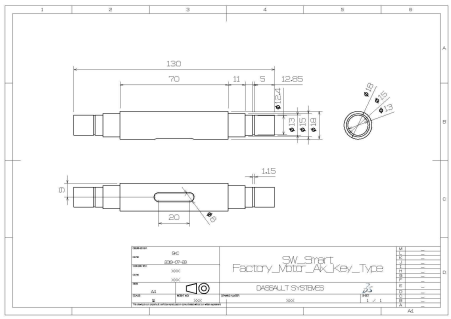


그림 6. 스텝 모터 Pulley Aix 설계도

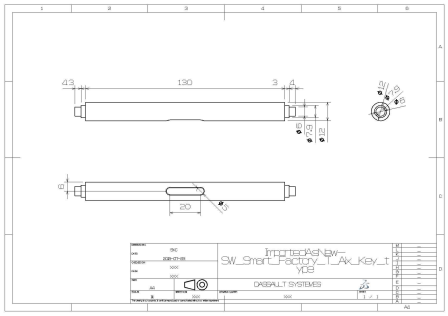


그림 7. 텐셔너 Pulley Aix 설계도

그림 8. 스텝 모터 Pulley Aix 3D 모델링

그림 9. 텐셔너 Pulley Aix 3D 모델링

그림 10. 스텝모터 Pully Aix 실제 모습

그림 11. 텐셔너 Pulley Aix 실제 모습

3) Conveyor Belt

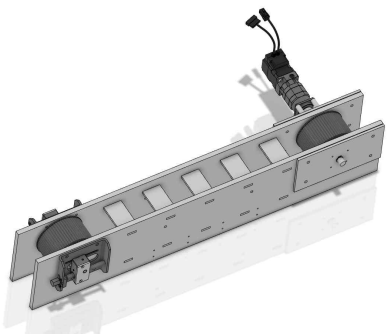


그림 12. Conveyor Belt 3D 모델링

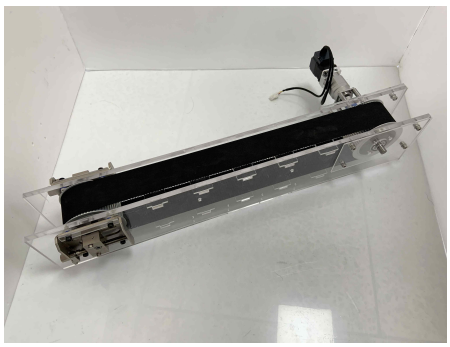


그림 13. Conveyor Belt 실제 모습

4) Robot

그림 14. Robot 3D 모델링

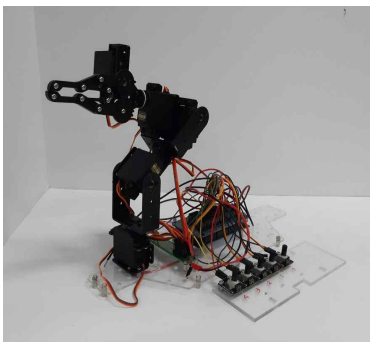


그림 15. Robot 실제 모습

2.1.2. Rotation Stage

MLCC를 고정하기 위한 진공판, MLCC를 회전하여 카메라로 촬영하기 위해 Rotation Stage를 활용하였고 Rotation Stage와 진공판을 고정하기 위한 마운트 제작을 통해 카메라를 움직이는 대신 Rotation Table을 움직여 파노라마 촬영을 하도록 설계하였다.

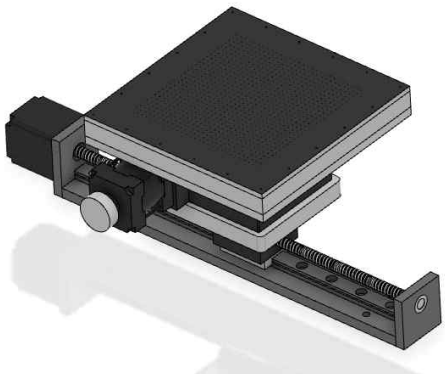


그림 16. Rotation Table 3D 모델링

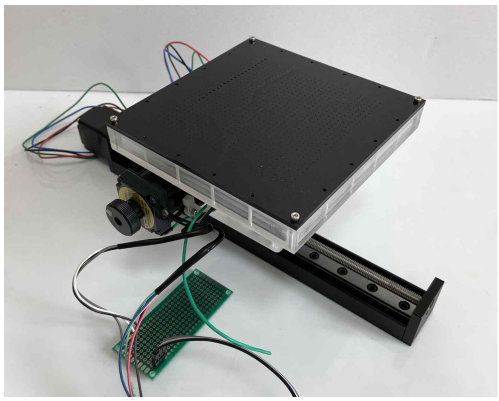


그림 17. Rotation Table 실제 모습

2.1.3. ERROR CHECK Stage

Rotation Table로 옮겨진 MLCC를 검사하기 위해 Rotation Table 위의 MLCC의 중심과 카메라의 중심의 높이를 일치 시켜 설계를 진행하였다. 카메라 렌즈와 Rotation Stage의 중심을 일치시켜주기 위해 카메라 마운트부를 비대칭으로 설계하였다.

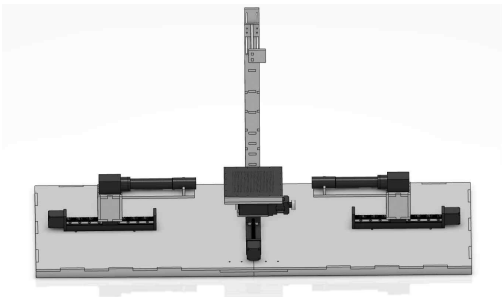


그림 18. MLCC 검사 장비 3D모델링

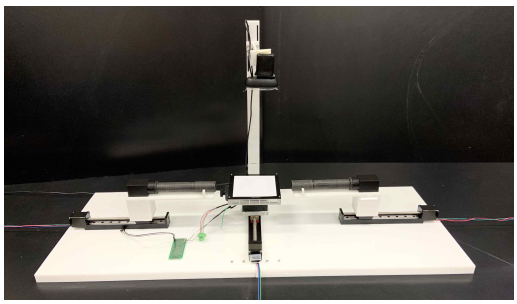


그림 19. MLCC 검사 장비 실제 모습

2.1.4. 전체 구성

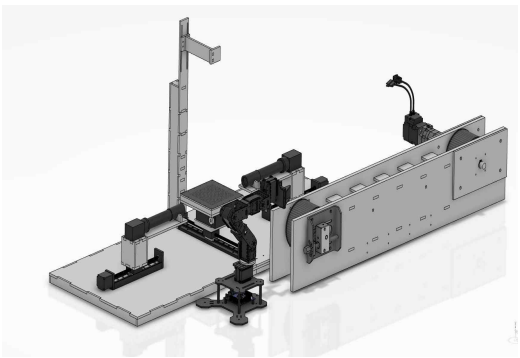


그림 20. 3D 모델링한 전체 모습

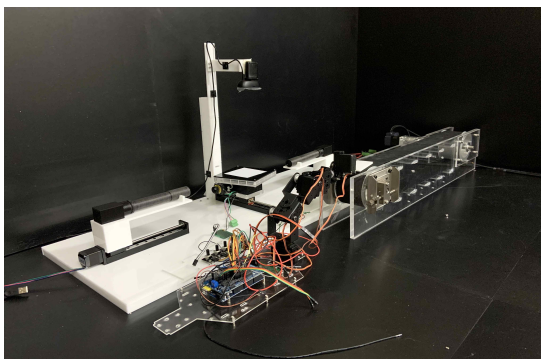


그림 21. 전체 구성 실제 모습

2.2. Software 구성

1) 전체 시스템 구조

SMART MLCC Cutting Inspection의 전체 시스템의 논리적 구조와 실제 구현된 모습을 함께 나타내었다. SMART MLCC Cutting Inspection의 논리적 구조는 그림 22와 같으며, 실제 구현된 모습은 그림 23과 같다.

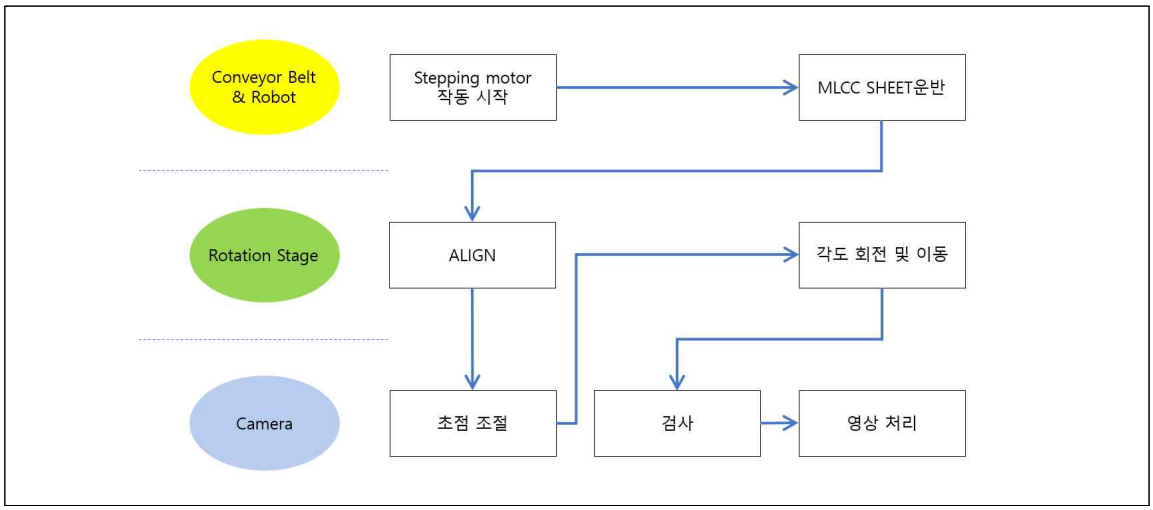


그림 22. SMART MLCC Cutting Inspection 전체 시스템의 논리적 구성도

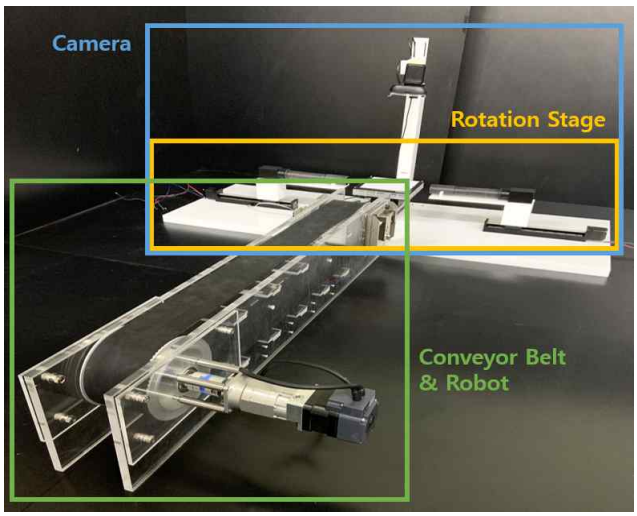


그림 23. SMART MLCC Cutting Inspection 전체 시스템 실제 구현 모습

2) Conveyor Belt & Robot

MLCC Sheet를 Checking Table까지 운반시켜주는 작업을 하는 파트로 Conveyor Belt에 Stepping모터가 장착되어있다.

3) Rotation Stage

MLCC 품질 검사를 진행하기 전, Table 위에 놓인 Sheet를 ALIGN 해주는 작업을 하는 파트로 Rotation Table, Linear Stage, 그리고 Checking Table로 구성되어있다. Rotation Table은 각도 값을 pulse값으로 변환하여 MLCC의 단면을 볼 수 있게 회전시켜준다. Linear Stage는 카메라와 MLCC 간의 거리와 위치를 조정하기 위해 사용하였다.

4) Camera

MLCC 단면을 촬영한 후, 영상처리 작업을 하는 파트로 총 3대의 카메라(카메라 모델명)로 구성되어 있다. MLCC 절단 후 품질 검사 과정에서 외곽선 인식작업은 불량 및 품질 판단에 있어 중요한 역할을 한다. 하지만 카메라는 항상 영상왜곡이 존재한다. 이를 해결하기 위해서는 촬영으로 측정한 좌표 값을 실제 좌표로 변환해주는 작업이 필요하다. 따라서 카메라로 촬영한 영상의 왜곡을 3차 다항식을 통하여 실제 좌표를 구하고 외관선의 폭과 외곽선의 곡률반경을 구하는 영상 왜곡 교정 알고리즘 프로그램을 설계하였다. 이를 통해 카메라 영상에서 차선의 실제 좌표를 측정해 차선을 따라 주행 할 수 있으며 차선의 폭과 차선의 곡률반경 등을 구할 수 있다.

2.3. Software 설계도

2.3.1. MACHINE CONTROL

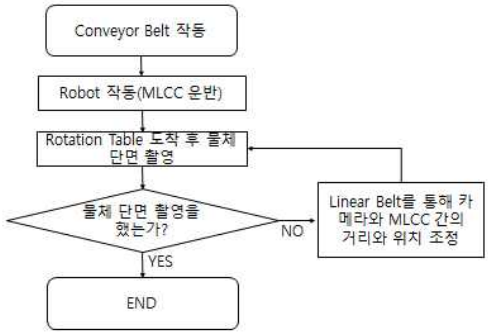


그림 24. MACHINE CONTROL 전체 알고리즘

2.3.2. VISION

1) Calibration

실제 측정된 영상 속 지점의 좌표를 [u,v]라 하고 실제 구하고자 하는 지점의 좌표 값을 [x,y]라고 할 때, 카메라로 촬영한 영상의 차선은 perspective로 인해 영상좌표[u,v]좌표가 실 좌표 [x,y]좌표와 일치하지 않는다. 일반적으로는 [v]좌표가 커질수록 [u]좌표와 [x]좌표간의 오차가 커지는데 영상좌표 [u,v]좌표와 실 좌표 [x,y]관계가 2차의 관계를 갖고 있다고 가정하면, 카메라의 calibration 모델을 이용하지 않고 3차 다항식을 통하여 실제 좌표를 구할 수 있다. 3차 다항식의 각 계수는 행렬을 통해 영상좌표 [u,v]에 관한 행렬과 이에 대응하는 실 좌표 [x,y]행렬간의 관계를 통하여 구할 있고 관계를 통한 후에 영상좌표 [u,v]에서 실 좌표 [x,y]로 변환한다.

2) Align

카메라에서 입력받아 이진화 시킨 영상의 크기를 계산하고 물체를 검색해 물체에 대한 중심을 찾아 중심좌표를 계산한다. 이를 화면상에 라인을 그려 표시한다. 표시된 무게중심을 기준으로 관심영역을 설정한다. 영상에 대한 현재 픽셀값과 주변 이웃한 픽셀값들의 가중평균을 이용해 현재의 픽셀값을 대처하는 방법으로 영상을 부드럽게 만든다. 영상의 Edge를 구해서 모서리 line를 찾아 이를 분리한다. 선형회귀를 사용하여 직선의 방정식을 구한 뒤, 이를 이용하여 평균을 구한 후 회전각을 계산하여 회전판을 제어한다. 전달받은 회전각이 0.1도 이내일 경우에는 종료하게 된다. 이러한 Align의 흐름도를 그림 25와 같이 나타내었다.

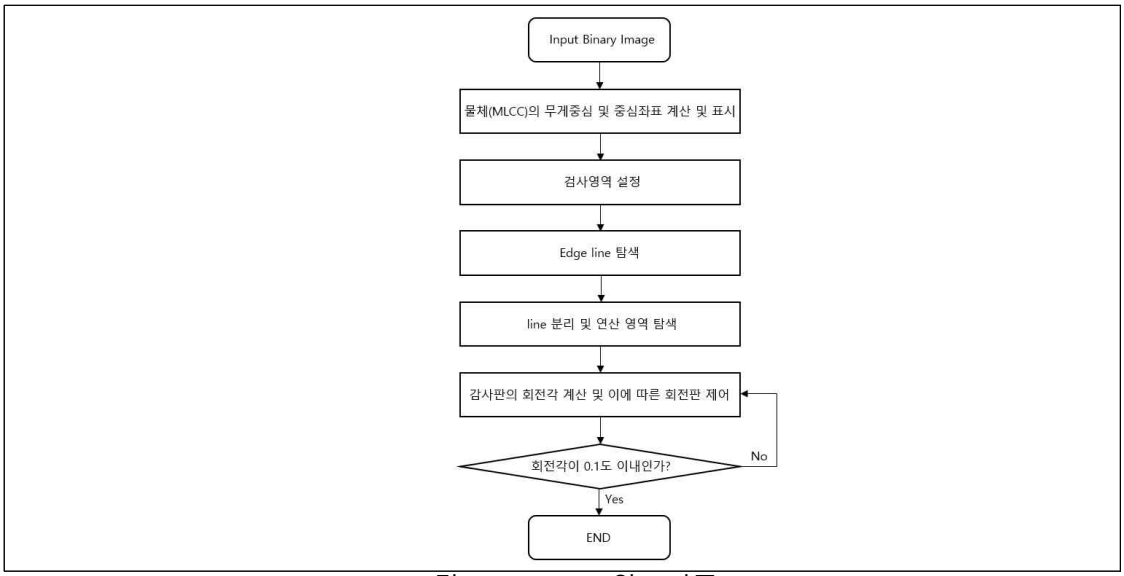


그림 25. ALIGN 알고리즘

3) ERROR CHECK

카메라에서 입력받은 영상에 대하여 이진화를 시킨 영상을 크기를 계산하고 영상의 물체를 검색하여 물체에 대한 중심을 찾아 중심좌표를 계산해 화면상에 라인을 그려 표시한다. 표시된 무게중심을 기준으로 검사영역을 설정한다. 이진화 영상의 가로방향으로 Projection

해 가로방향의 Projection 데이터를 가지고 Projection 데이터의 특정값들 보다 높은 영역의 리브들의 중심들을 찾는다. Projection 데이터를 바탕으로 라벨링을 하며 중심점들을 찾아간다. 라벨링이 끝나게 되면 MLCC의 개수를 표시하게 되며, 리브의 중심을 추출하게 된다. 세로방향으로도 Projection 해 마찬가지로 라벨링 하며 중심점들을 찾는다. 라벨링이 끝나면 MLCC의 개수를 표시하게 되며, 리브의 중심을 추출한다. 가로, 세로 방향으로 리브를 검출하게 되면 리브들의 폭을 구하게 되고 리브들의 폭이 정해진 규정보다 작으면 양품으로 표시하고 그렇지 않다면 불량으로 표시한다. 이러한 흐름도를 그림 26과 같이 나타내었다.

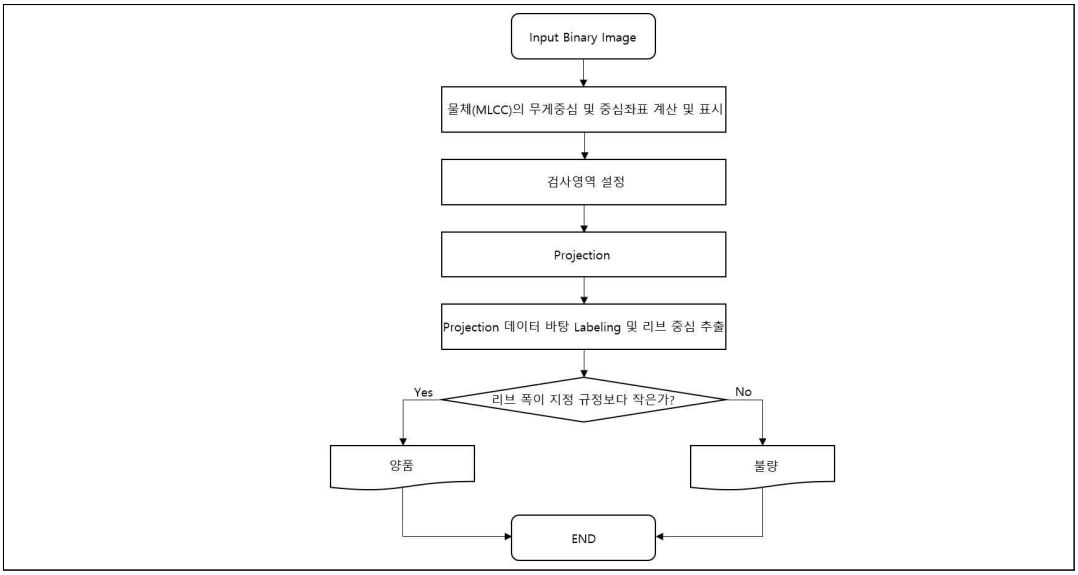


그림 26. ERROR CHECK 알고리즘

2.4. Software 기능

2.4.1. 카메라 교정 및 2차원 영상 교정

카메라를 통하여 구한 이미지의 2차원 영상 좌표 값과 그에 대응하는 3차원 절대 좌표 값 사이에 존재하는 대응관계를 구하는 것을 교정이라고 한다. 카메라 교정을 위해서는 그림 27과 같이 2개의 좌표가 정의된다. 첫 번째는 절대좌표계로 대상 물체의 위치를 나타내는 절대위치를 나타낸다. 두 번째로는 카메라의 영상 상에서 정의되는 영상좌표계가 있다.

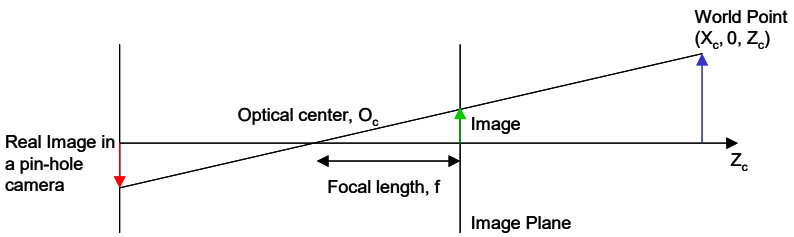


그림 27. 카메라의 좌표 설정

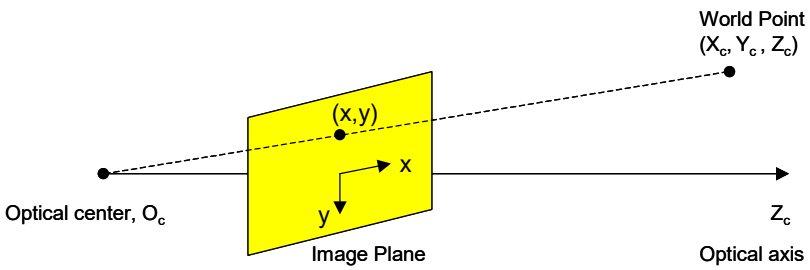


그림 28. 이미지 좌표계

그림 28은 카메라의 좌표를 보여준다. (X_c, Y_c, Z_c)는 절대좌표계의 좌표를 나타내며 f

는 초점 거리 그리고 Oc는 optical center이고 카메라의 상은 그림과 같이 뒤집어져서 맺히게 된다. 이미지 평면과 카메라 상이 맺히는 실제 이미지를 나타나있다. 카메라의 영상 좌표계의 변환은 다음과 같이 나타낼 수 있다.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} \cdot \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \quad (2\text{-}1)$$

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = R \cdot \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} + T \quad (2\text{-}2)$$

(u,v,w)는 카메라의 좌표 점들을 나타내고 (X_c,Y_c,Z_c)는 실제 좌표계의 좌표 점으로 카메라 교정 차트에서 주어지는 값들이다. $(r_1,r_2,\mathrm{L}\ ,r_9)$는 회전 행렬(rotation matrix) 을 나타내고 이는 R로 나타낼 수 있다. (T_x,T_y,T_z)는 평행이동 행렬(translation matrix)을 나타낸다. 이는 T로 나타낸다.

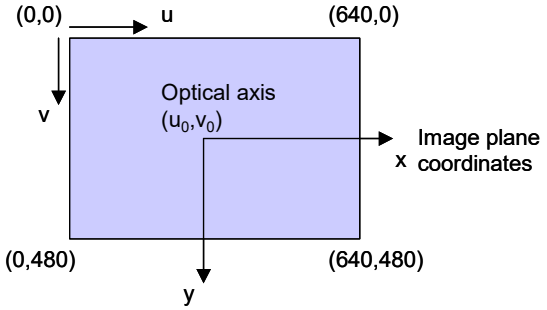


그림 29. 카메라 좌표와 이미지 평면

그림 29는 이미지 좌표와 물체와의 관계를 나타낸다. Optical center을 중심으로 이미지 좌표에 위치한 이미지의 점들과 측정 대상의 물체의 좌표가 기하학적으로 비례하는 관계를 갖으며 이에 따라서 다음과 같은 관계식을 얻는다.

$$x = \frac{fX_c}{Z_c}, \quad y = \frac{fY_c}{Z_c} \quad (2\text{-}3)$$

이때, f는 초점 거리이고 x와 y는 image plane에서의 좌표이다.

전체 이미지의 크기는 640×480 이고 왼쪽 상단부의 좌표가 원점으로 (0,0)의 값을 갖는다. 이미지 좌표의 축은 u,v 좌표계로 나타낼 수 있으며 이미지 좌표는 보통 카메라 좌표의 중심에서 원점을 갖게 되고 이때 중심의 좌표는 (u_o,v_o) 이다.

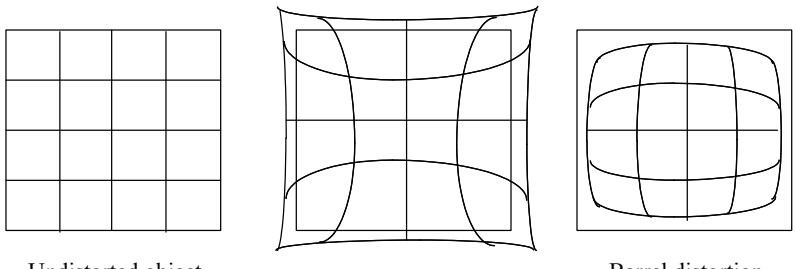


그림 30. 왜곡된 영상의 예

Image plane에서 카메라에 상이 맺히는 경우에 그림 30과 같은 왜곡이 발생한다. 일반적으로 왜곡의 종류에는 그림과 같이 Pin-cushion distortion, Barrel distortion이 있다. Pin-cushion 왜곡은 중심으로부터 왜곡된 영상의 길이가 왜곡되기 전의 영상에 비해 짧아지게 되고 Barrel의 경우에는 길어지는 현상이 일어난다. 왜곡이 발생하는 경우에는 실제 물체의 형태가 왜곡되어서 직선을 추출하거나 정확한 위치에서 원하는 영상 정보를 얻어내는 것이 어렵게 된다. Image plane에서 카메라의 좌표로 변환하고 이 왜곡을 변환시키기 위한 식은 다음과 같다.

$$u = u_0 + k_u \cdot x_d$$
$$v = v_0 + k_v \cdot y_d$$
$$where\ k_u\ and\ k_v\ (pixel/m) \quad (2\text{-}4)$$

k_u, k_v 는 image plane에서 카메라로 좌표가 변환될 때, 각각 u축과 v축 방향으로의 영상 배율이 된다. 영상 왜곡을 고려하면

$$x_d + D_x = x$$
$$y_d + D_y = y \quad (2\text{-}5)$$

이다. 이때, D_x 는 x축에 대한 비선형 항이고 D_y 는 y축에 대한 비선형 항이다.

$$D_x = x_d \cdot k \cdot (x_d^2 + y_d^2) \quad (2\text{-}6)$$

$$D_y = y_d \cdot k \cdot (x_d^2 + y_d^2) \quad (2\text{-}7)$$

이 된다. 이때, x_d, y_d 는 영상 왜곡이 일어났을 때의 카메라 좌표이고, k 는 영상의 왜곡계수이다. 따라서 영상 왜곡 모델은 다음과 같은 식으로 최종적으로 얻어진다.

$$u = Sx \cdot \frac{x_d}{d_x} + u_o \quad (2\text{-}8)$$

$$v = \frac{y_d}{d_y} + v_o \quad (2\text{-}9)$$

이때 Sx는 스캔 파라미터이고 dx는 x축의 영상의 길이, yd는 y축 영상의 길이다. u_o, v_o 는 카메라의 중심점을 나타낸다.

2.4.2. 카메라 모델을 이용한 영상 왜곡 보정

구하고자 하는 값은 transformation matrix T와 rotation matrix R, 초점 거리 f, 왜곡 상수 k, 스캐닝 파라미터 Sx의 15개의 parameter를 찾는다. 찾는 방법은 다음과 같다.

$$\frac{y_d}{x_d} = \frac{y}{x} = \frac{r_4 X_c + r_5 Y_c + r_6 Z_c + T_y}{r_1 X_c + r_2 Y_c + r_3 Z_c + T_x} \quad (2\text{-}10)$$

$$x_d = \frac{d_x}{S_x}(u - u_o) \quad (2\text{-}11)$$

$$y_d = d_y(v - v_o) \tag{2-12}$$

위의 식을 정리하면,

$$[y_d X_c \quad y_d Y_c \quad y_d Z_c \quad y_d \quad -x_d X_c \quad -x_d Y_c \quad -x_d Z_c] \cdot L = x_d \tag{2-13}$$

$$L = \left[\frac{S_x r_1}{T_y} \quad \frac{S_x r_2}{T_y} \quad \frac{S_x r_3}{T_y} \quad \frac{S_x T_x}{T_y} \quad \frac{r_4}{T_y} \quad \frac{r_5}{T_y} \quad \frac{r_6}{T_y} \right]^T \tag{2-14}$$

따라서 위의 식에서 calibration 점을 7개 이상 취할 경우 미지의 벡터 L을 구할 수 있게 된다. 얻어진 L 벡터로부터,

$$\left|T_y\right| = \sqrt{L_5^2 + L_6^2 + L_7^2} \tag{2-15}$$

$$S_x = \sqrt{L_1^2 + L_2^2 + L_3^2} \cdot \left|T_y\right| \tag{2-16}$$

이로부터 R과 T 행렬을 구할 수 있고, 위의 식으로부터 Sx를 구한다. 그리고 f와 k는 앞에서 주어진 식으로부터 계산에 의해 얻어지는 값이다.

2.4.3. 비선형 모델을 이용한 영상 보정 계수 찾기

영상 왜곡 교정 알고리즘 프로그램으로 영상의 왜곡을 교정해 실제 좌표를 구하였다. 이의 방식은 다음과 같다.

측정된 영상의 카메라에서의 좌표를 영상좌표 [u,v], 실제 구하고자 하는 물체의 좌표 값을 실 좌표 [x,y]라고 할 때, 이 관계가 2차의 관계를 갖고 있다고 가정하면, 다음과 같이 3차 항식으로 표시할 수 있다.

$$x = a_1u^3 + b_1u^2v + c_1uv^2 + d_1v^3 + e_1u^2 + f_1v^2 + g_1uv + h_1u + i_1v + j_1 \quad \text{(식1)}$$

$$y = a_2u^3 + b_2u^2v + c_2uv^2 + d_2v^3 + e_2u^2 + f_2v^2 + g_2uv + h_2u + i_2v + j_2 \quad \text{(식2)}$$

따라서 주어진 데이터로부터 $a_1, b_1, \mathrm{L}\ f_2$, 12개의 미지수를 찾으면 된다. 따라서, 측정된 좌표의 개수를 N개라고 하면, 카메라의 좌표와 측정된 물체의 좌표사이에 다음과 같은 행렬을 만들어진다.

$$\begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \\ \vdots & \vdots \\ x_N & y_N \end{bmatrix} = \begin{bmatrix} u_1^3 & u_1^2v_1 & u_1v_1^2 & v_1^3 & u_1^2 & v_1^2 & u_1v_1 & u_1 & v_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_n^3 & u_n^2v_n & u_nv_n^2 & v_n^3 & u_n^2 & v_n^2 & u_nv_n & u_n & v_n & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_N^3 & u_N^2v_N & u_Nv_N^2 & v_N^3 & u_N^2 & v_N^2 & u_Nv_N & u_N & v_N & 1 \end{bmatrix} \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \\ c_1 & c_2 \\ d_1 & d_2 \\ e_1 & e_2 \\ f_1 & f_2 \\ g_1 & g_2 \\ h_1 & h_2 \\ i_1 & i_2 \\ j_1 & j_2 \end{bmatrix} \tag{2-17}$$

$$X = A \cdot x \tag{2-18}$$

이때, A는 $N \times 10$ 행렬이고 N는 교정 패턴의 매칭되는 카메라 좌표의 수이다. X는 우리가 알고 있는 실제 좌표 값이며, 따라서 문제는 x를 찾는 문제로 변하였다. 위의 x값은 다음과 같이 역행렬을 구함으로써 얻어진다.

$$x = A^{-1} \cdot X \tag{2-19}$$

일반적으로 A는 정방행렬이 아니므로 의사 역행렬(pseudo inverse)을 구하여 얻을 수 있다.

$$A^{-1} = A^T \cdot A \cdot A^T \qquad (2\text{-}20)$$

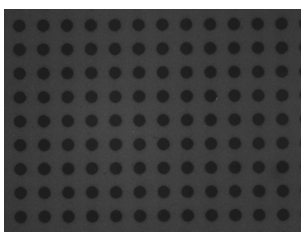
2.4.4. 영상 보정 및 실험 결과

영상보정 차트를 통하여 그림 31.(b),(c)와 같은 영상을 카메라로부터 얻었고, 이 영상 차트의 원의 중심은 각각의 간격이 0.125mm 의 등 간격으로 되어 있으므로 실제 좌표계에서의 영상의 위치를 얻을 수 있다. 영상 보정 차트의 각 점의 위치를 알기 위하여 이미지 처리를 통하여 원의 중심의 위치를 찾아낸다. 영상 좌표에서의 원의 중심 좌표와 실제 좌표의 쌍으로 된 데이터가 얻어야 한다. 얻어진 영상으로부터 정확한 중심의 위치를 얻어내는 것이 중요하므로 먼저 Thresholding을 통하여 영상을 이진화하여 원의 경계를 분리하였다. 이렇게 분리된 원들로부터 Labeling를 통하여 원의 개수를 센다. Labeling을 통하여 얻어진 원들로부터 무게 중심법을 사용하여 각각의 영상에서의 좌표 값을 얻어낼 수 있다. 얻어진 데이터를 위의 식 (1) 에 대입하여 우리가 찾고자 하는 행렬 x를 얻어낸다. 이렇게 얻어진 변수는 모두 20개가 된다. 영상 보정 기법의 성능을 평가하기 위하여 왜곡이 존재하는 영상의 점들과 실제로 영상 보정 후의 값으로부터 오차를 계산하여 성능을 평가하였다. 그림 32는 이때의 오차 분포를 나타내고 오차에 대한 결과는 표 1에 나타내었다. 카메라 모델을 3차로하였기 때문에 비교적 정확한 영상의 보정이 이루어졌으며 10X 배율에서의 최대 오차는 0.207 um이고 평균 오차는 0.126 um이었다. 그리고 한 Pixel 의 오차는 0.54 um 이었다. 따라서 최대오차는 0.207 pixel의 오차가 나고 평균은 0.126 pixel 의 오차가 발생해 영상의 보정으로 인해 발생하는 오차는 0.5 pixel 미만인 것을 확인 할 수 있었다.

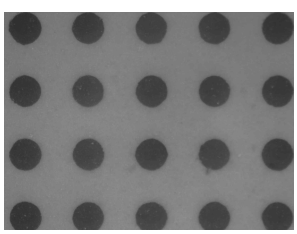
표 1. 오차 분석

Type of error		Error Value
2X	one pixel error	2.50 um
	average error	1.28 um
	max errror	1.84 um
4X	one pixel error	1.25 um
	average error	0.34 um
	max error	0.83 um
10X	one pixel error	0.55 um
	average error	0.126 um
	max error	0.207 um

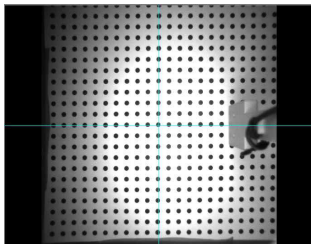
(a) 카메라 교정 calibration chart 0.125mm



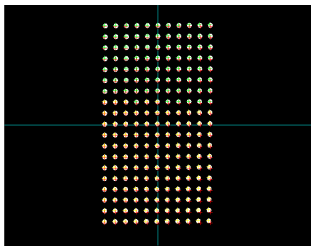
(b) 4X 배율 교정용 calibration chart



(c) 10X 배율 교정 calibration chart



(d) calibration 영상처리



(e) calibration 영상처리 결과

(d) calibration 지그

그림 31. 영상 보정 차트 사진

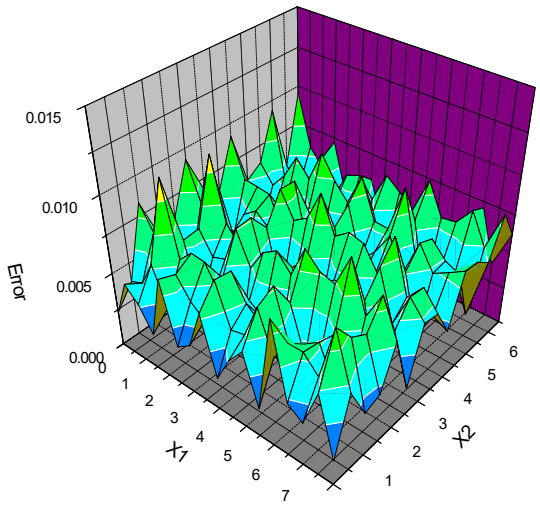


그림 32. 오차 분석 결과

2.5. UI 사용법

UI는 크게 Hardware Test와 Hardware Setup으로 구성되어있다.

1. Hardware Test	[1] Rotation Table Test [2] Center Linear Table Test [3] Right Linear Table Test [4] Left Linear Table Test
2. Hardware Setup	[1] Rotation Table Setup [2] Center Linear Table Setup [3] Right Linear Table Setup [4] Left Linear Table Setup

2.6. 개발환경

구성요소	구성환경		
	H/W	S/W	UI
개발 언어	C	C	C
개발 환경	Windows10	Windows10, Linux	Windows10
개발 보드	Arduino Mega	UBUNTU	Arduino Uno
개발 도구	Arduino, CATIA V6(DIGITAL TWINS 활용)	CodeVisionAVR Visual Studio	Arduino

3. 개발 프로그램 설명

3.1. 파일 구성

3.1.1. CALIBRATION

2Dcalib_matrix	2Dcalib_matrix.cpp	카메라 사용시 발생하는 외곡 수정 코드

3.1.2. ERROR CHECK

imageprocessing	imageprocessing..c	이진화, 노이즈 제거 라인검출, 중심 검출 등 영상처리 코드
Jeston_nano_GPIO	Jeston_nano_GPIO..c	NVIDIA Jetson Nano보드의 포트 사용을 위한 코드
Rotation_inspection	Rotation_inspection.c	Angle Stage 위의 MLCC SHEET 의 기울기를 구하고 Angle Stage회전각을 구하기 위한 코드
Serial_com	Serial_com..c	Rotation_inspection에서 구한 각도를 모터와 통신하기 위한 코드

3.1.3. ROBOT

Robot arm	Robot arm.ino	컨베이어를 통해 옮겨온 MLCC SHEET를 Rotation Stage에 이송

3.2. 함수별 기능

3.2.1. MACHINE CONTROL

- stepper_init()

  Stepping Motor를 구동시키기 위해 사용하는 함수이다. Stepping Motor의 최대 속도를 지정을 해 준 다음, 구동을 시킨다.

- pulse_angle()

  Rotation Table의 회전각을 Pulse 값으로 변환 시켜주는 함수이다. 지정된 회전각에 맞게 Rotation Table을 회전시켜준다.

- swInterrupt()

  Rotation Table을 원점으로 다시 회전시켜주는 함수이다. 돌려져 있는 Rotation Table을 원점으로 조정해준다.

- Pulse dist()

  Linear Stage를 작동시키는 함수이다. Rotation Table에서 제품(MLCC)를 촬영을 하지 못했을 경우, Linear Stage에서 카메라와 제품(MLCC) 간의 거리와 위치를 조정해준다.

3.2.2. VISION

- Calculation_Center_BinaryImage()

  물체(MLCC)의 무게중심을 구하기 위한 함수이다. 카메라로 캡처한 영상을 화면에 표시한 뒤 이진화를 통하여 이미지를 반전 시킨다. 그 후 영상의 가로, 세로 크기만큼 크기를 계산 후 무게중심을 구하게 된다.

- Line()

  MLCC의 무게중심을 표시하는 함수이다. Calculation_Center_BinaryImage()에서 무게중심을 구한 후 영상의 시작 좌표 점과 끝 좌표 점을 계산하여 화면상에 줄을 그려서 표시해준다.

표시된 무게중심을 기준으로 ROI영역을 설정하게 된다.

- Set_ROI()

  검사 영역을 설정하는 함수이다. 영상에 대한 연산량을 줄이기 위하여 관심영역을 설정한다. 관심영역의 크기는 가변적으로 적용할 수 있다.

- GaussianBlur()

  영상에 대한 현재 픽셀 값과 주변 이웃한 픽셀 값들의 가중 평균을 이용해서 현재 픽셀의 값을 대체한다. 현재 픽셀에서 가까울수록 더 큰 가중치를 갖고 멀수록 더 작은 가중치를 갖는다. 그래서 노이즈 값은 상대적으로 이웃 픽셀들 값과 상관성이 작기 때문에 이웃 픽셀 값들의 가중평균의 방식으로 완화시킬 수 있다. Canny Edge를 사용하기 위하여 영상을 부드럽게 만든다.

- Canny()

  영상의 Edge를 구해서 모서리 line를 찾는다. GaussianBlur()를 사용하여 영상을 부드럽게 만든 후 Sobel 연산자를 사용하여 기울기(Gradient) 벡터의 크기(Magnitude)를 계산하고 Gradient 벡터 방향에서 Gradient크기가 최댓값인 화소만 남기고 나머지는 0으로 만든다. 연결된 Edge를 얻기 위해 두 개의 임계값을 사용하고 높은 값의 임계값을 사용하여 Gradient방향에서 낮은 값의 임계값이 나올 때까지 추적하며 Edge를 연결하는 hysteresis thresholding 방식을 사용하여 모서리 부분을 찾는다.

- LinearCurveFitting()

  라인을 선형회귀를 사용하여 직선의 방정식을 구한다. 검사 판의 회전각을 구하기 위하여 필요한 함수이다. 두 개의 직선의 방정식을 이용하여 평균을 구한 후 회전각을 계산하여 회전판을 제어한다.

- Projection_H()

  이진화 영상의 가로방향으로 Projection을 하는 함수이다. 설정된 관심영역에서 수행을 하게 된다.

- Find_Projection_Center_Position()

  가로방향의 Projection 데이터를 가지고 Projection 데이터의 특정값들 보다 높은 영역의 리브들의 중심들을 찾는 함수이다. Projection 데이터를 바탕으로 라벨링을 하며 중심점들을 찾아가게 된다. 라벨링이 끝나게 되면 MLCC의 개수를 표시하게 되며, 리브의 중심을 추출하게 된다.

- Projection_V()

  이진화 영상의 세로방향으로 Projection을 하는 함수이다. 설정된 관심영역에서 수행을 하게 된다.

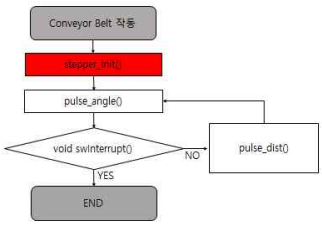
- Find_ProjectData_Center_Position()

  Projection 데이터를 가지고 Projection 데이터의 특정값들 보다 높은 영역의 리브들의 중심들을 찾는 함수이다. Projection 데이터를 바탕으로 라벨링을 하며 중심점들을 찾아가게 된다. 라벨링이 끝나게 되면 MLCC의 개수를 표시하게 되며, 리브의 중심을 추출하게 된다. 가로,세로 방향으로 리브를 검출하게 되면 리브들의 폭을 구하게 되고 리브들의 폭이 정해진 규정보다 작으면 양품으로 표시하고 그렇지 않다면 불량으로 표시한다.

3.3. 주요 함수의 흐름도

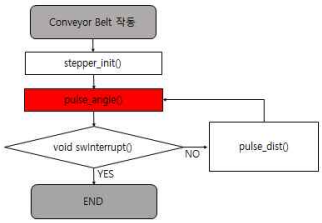
3.3.1. MACHINE CONTROL

1) Rotation Stage



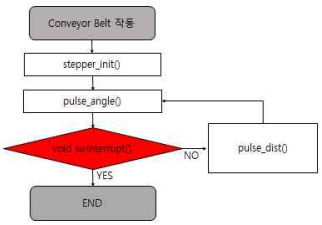
stepper_init()는 Stepping Motor를 구동시키기 위해 사용하는 함수이다.
Stepping Motor의 최대 속도를 지정을 해준 다음, 구동을 시킨다.

```
void stepper_init()
{
   stepper.setMaxSpeed(5000);
   stepper.setAcceleration(1000);
}
```



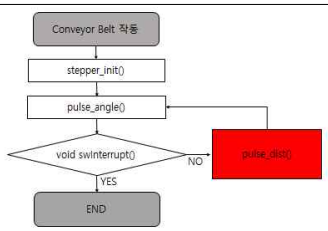
pulse_angle()은 Rotation Table의 회전각을 Pulse 값으로 변환 시켜주는 함수이다.
지정된 회전각에 맞게 Rotation Table을 회전시켜준다.

```
long pulse_angle(long angle)
{
   long pulse = 0;
   pulse = angle * 250;
   stepper.setSpeed(1000);
   stepper.moveTo(pulse);
   stepper.runToPosition();
   motorPosition = stepper.currentPosition();
   Serial.print("angle : ");
   Serial.println(angle);
   Serial.print("pulse : ");
   Serial.println(motorPosition);
}
```



swInterrupt()는 Rotation Table을 원점으로 다시 회전시켜주는 함수이다.
돌려져 있는 Rotation Table을 원점으로 조정해준다.

```
void swInterrupt()
{
   digitalWrite(ledPin, HIGH);
   delayMicroseconds(2000);
   Serial.println("원점");
   //stepper.setCurrentPosition(0);
   pos_home_old = stepper.currentPosition();
}
```



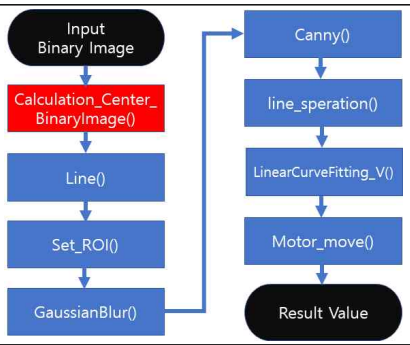
Pulse dist() 함수는 Linear Stage를 작동시키는 함수이다.
Rotation Table에서 제품(MLCC)를 촬영을 하지 못했을 경우, Linear Stage에서 카메라와 제품(MLCC) 간의 거리와 위치를 조정해준다.

```
long pulse_dist(long dist)
{
   long pulse = 0;
```

```
    digitalWrite(hold_pin,HIGH);
    pulse = dist * 338;
    stepper.setSpeed(2000);
    stepper.moveTo(pulse);
    stepper.runToPosition();
    motorPosition = stepper.currentPosition();
    Serial.print("dist : ");
    Serial.println(dist);
    Serial.print("pulse : ");
    Serial.println(motorPosition);
    digitalWrite(hold_pin,LOW);
}
```

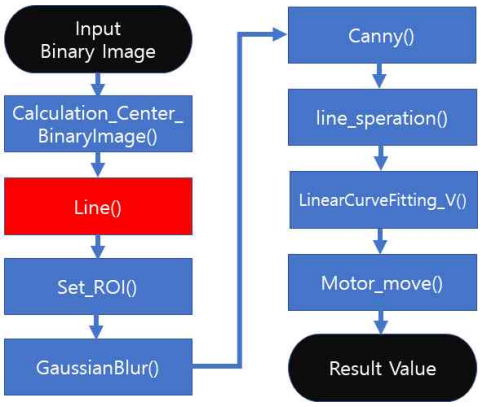
3.3.2. VISION

1) Align



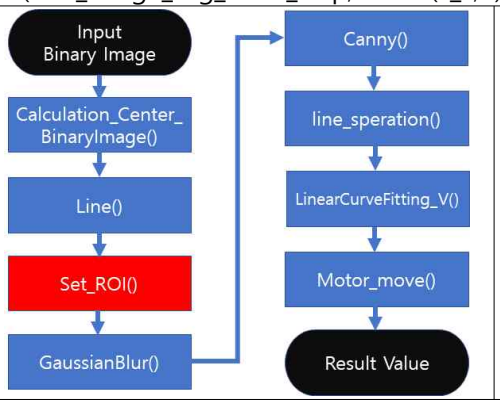
Calculation_Center_BinaryImage()는 물체(MLCC)의 무게중심을 구하기 위한 함수이다. 카메라에서 입력받은 영상에 대하여 이진화를 시킨 영상을 크기를 계산하고 영상의 물체를 검색하여 물체에 대한 중심을 찾아 중심좌표를 계산한다.

```
void Calculation_Center_BinaryImage(BYTE *Bin_Image, CSize image_size, double *x_c, double *y_c)
{
        int x, y;
        int height = image_size.cy;
        int width = image_size.cx;
        UINT area;
        UINT x_sum;
        UINT y_sum;

        area = x_sum = y_sum = 0;
        for (y = 0; y < height; y++)
        {
                for (x = 0; x < width; x++)
                {
                        if (*(Bin_Image + y * width + x) == 255)
                        {
                                area++;
                                x_sum += x;
                                y_sum += y;
                        }
                }
        }

        *x_c = (double)x_sum / (double)area;
        *y_c = (double)y_sum / (double)area;
}
```



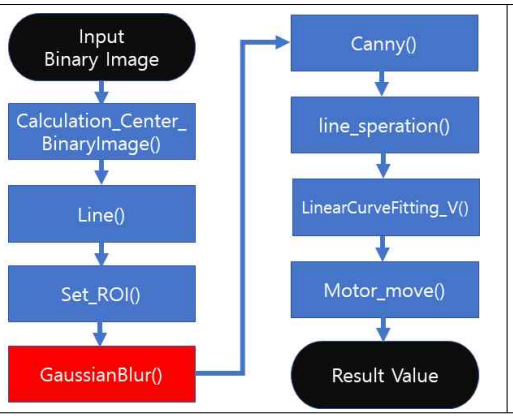
Line()는 무게중심을 표시하는 함수이다. Calculation_Center_BinaryImage()에서 구해진 좌표를 이용하여 영상의 크기를 계산하고 화면상에 라인을 그려서 표시해 준다.

```
 line(mat_image_org_color_disp,  Point(0,y_c),Point(640,y_c),GREEN,1,LINE_AA);
line(mat_image_org_color_disp,  Point(x_c,0),Point(x_c,480),GREEN,1,LINE_AA);
```



Set_ROI()는 무게중심에 있는 검사할 물체에 대하여 검사영역을 설정하게 된다. Line()에 의하여 표시된 무게중심을 기준으로 관심영역을 설정하게 된다.

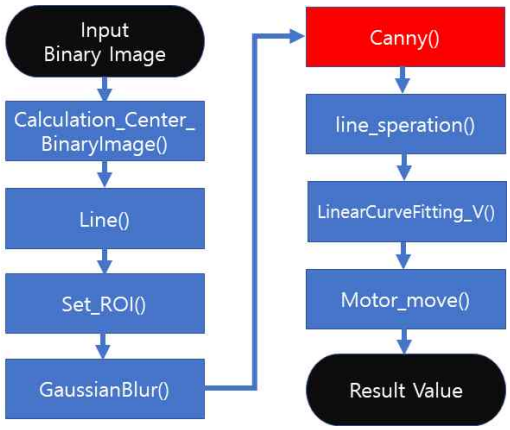
```
CRect Set_ROI(int x1,int y1,int x2, int y2)
{
        CRect temp;
        temp.left      = x1;
        temp.right     = x2;
        temp.top       =  y1;
        temp.bottom    = y2;
        return temp;
}
```



GaussianBlur()는 Canny()를 사용하여 Egde를 구하기 위한 전처기 과정이다. 영상에 대한 현재 픽셀값과 주변 이웃한 픽셀값들의 가중 평균을 이용해서 현재의 픽셀값을 대처하게된다. 이러한 방법으로 영상을 부드럽게 만들 수 있다.

```
int MaskGaussian[3][3]={{1,2,1}, {2,4,2}, {1,2,1}};
 int heightm1=height-1;//중복계산을 피하려고
 int widthm1=width-1;//중복계산을 피하려고
 int mr,mc;
 int newValue;

 //결과 이미지 0으로 초기화
 for(i=0;i<height;i++)
  for(j=0;j<width;j++)
```

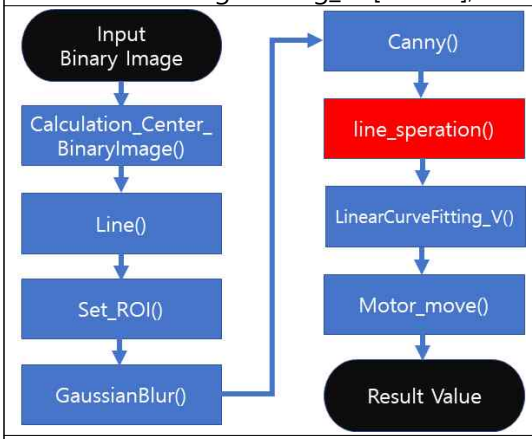
```
    OrgImg[i][j]=0;

 for(i=1; i<heightm1; i++)
 {
  for(j=1; j<widthm1; j++)
  {
   newValue=0; //0으로 초기화
   for(mr=0;mr<3;mr++)
    for(mc=0;mc<3;mc++)
     newValue += (MaskGaussian[mr][mc]*InImg[i+mr-1][j+mc-1]);
   newValue /= 20; //마스크의 합의 크기로 나누기:값의 범위를 0에서 255로 함
   OrgImg[i][j]=(BYTE)newValue;//BYTE값으로 변환
  }
 }
```



영상의 Edge를 구해서 모서리 line를 찾는다. GaussianBlur()를 사용하여 영상을 부드럽게 만든 후 Sobel 연산자를 사용하여 기울기(Gradient) 벡터의 크기(Magnitude)를 계산하고 Gradient 벡터 방향에서 Gradient크기가 최대값인 화소만 남기고 나머지는 0으로 만든다. 연결된 Edge를 얻기 위해 두 개의 임계값을 사용하고 높은 값의 임계값을 사용하여 Gradient방향에서 낮은 값의 임계값이 나올 때까지 추적하며 Edge를 연결하는 hysteresis thresholding 방식을 사용하여 모서리 부분을 찾는다.

```
// Sobel Edge Detection
    for(i=1; i<height-1; i++) {
        index = i*width;
        for(j=1; j<width-1; j++) {
            index2 = index+j;
            // -1 0 1
            // -2 0 2
            // -1 0 1
            dx = pImage[index2-width+1] + (pImage[index2+1]<<1) +
pImage[index2+width+1]
                -pImage[index2-width-1] - (pImage[index2-1]<<1) -
pImage[index2+width-1];

            // -1 -2 -1
            //  0  0  0
            //  1  2  1
            dy = -pImage[index2-width-1] - (pImage[index2-width]<<1) -
pImage[index2-width+1]
                +pImage[index2+width-1] + (pImage[index2+width]<<1) +
pImage[index2+width+1];

             mag = abs(dx)+abs(dy);      // magnitude
            //mag = sqrtf(dx*dx + dy*dy);

            dx_tbl[index2] = dx;
            dy_tbl[index2] = dy;
            mag_tbl[index2] = mag;
        }   // for(j)
    }   // for(i)
```

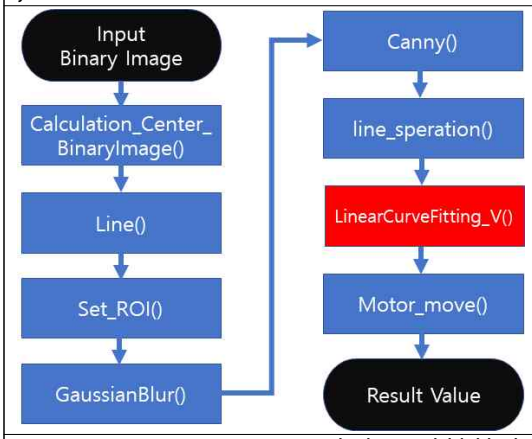
```
    for(i=1; i<height-1; i++) {
        index = i*width;
        for(j=1; j<width-1; j++) {
            index2 = index+j;

            mag = mag_tbl[index2];                    // retrieve the
```

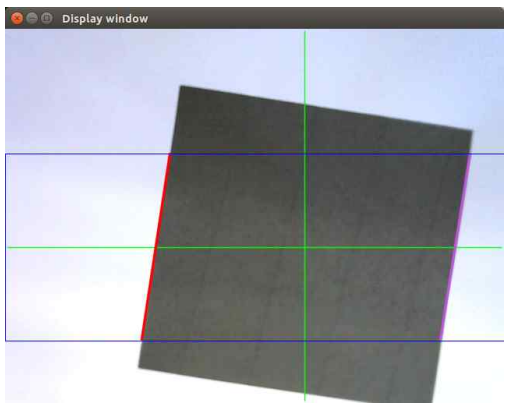


canny()에서 Edge를 구하고 모서리 부분의 line를 검출한 2개의 line를 좌우로 분리하는 함수이다. line를 분리하는 이유는 다음의 함수에서 연산할 영역을 찾기 위함이다.

```
void line_speration(BYTE *edge_image, CSize image_size, CRect ROI, CPoint *l_line_data, CPoint *r_line_data , CPoint center, int *no_line)
{
    int i,j;
    int no_l_line = 0;
    int no_r_line = 0;

    for (i = ROI.top; i<ROI.bottom; i++)
        {
            for (j = ROI.left; j<ROI.right; j++)
            {

                if (*(edge_image + i*image_size.cx + j) ==255 )
                {
                    // find left line
                    if(j < center.x)
                    {
                        l_line_data[no_l_line].x = j;
                        l_line_data[no_l_line].y = i;
                        no_l_line++;

                    }
                     // find right line
                    else
                    {
                        r_line_data[no_r_line].x = j;
                        r_line_data[no_r_line].y = i;
                        no_r_line++;

                     }
                }
            }
        }
        *no_line = no_r_line;

        FILE *fp;
```

```
        fp= fopen("line_l.dat","w");
        for(i=0;i<no_l_line;i++)
        {
        fprintf(fp,"%3d   %3d₩n",no_l_line, l_line_data[i].x,l_line_data[i].y);
        }
        fclose(fp);
}
```

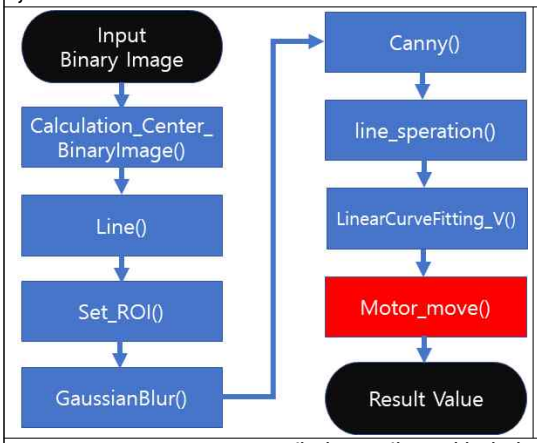


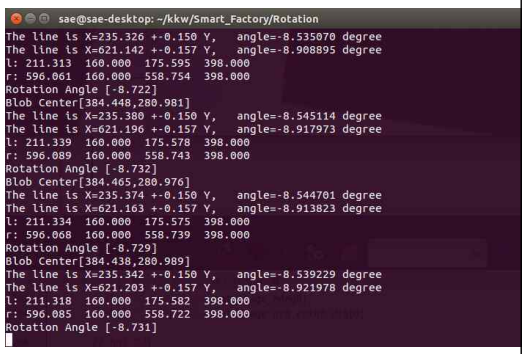


LinearCurveFitting_V()는 라인을 선형회귀를 사용하여 직선의 방정식을 구한다. 검사판의 회전각을 구하기 위하여 필요한 함수이다. 두 개의 직선의 방정식을 이용하여 평균을 구한 후 회전각을 계산하여 회전판를 제어한다.

```
double LinearCurveFitting_V(CPoint* data, int length, double *cc, double *dd)
{
        double sumx = 0, sumy = 0, sumxy = 0, sumx2 = 0;
        double a = 0.0, b = 0.0;
        double angle, min_angle = 360;
        int no_data =0;

        for (int i = 0; i < length; i++)
        {
            if(data[i].x!= -1)
            {
                sumx += data[i].y;
                sumy += data[i].x;
                sumx2 += data[i].y * data[i].y;
                sumxy += data[i].x * data[i].y;
        no_data ++;
            }
        }
        if (length*sumx2 == sumx*sumx) // can not divide by zero
        {
                angle = -90;
        }

        else
        {
        a = (((double)no_data*sumxy - sumx*sumy) / ((double)no_data*sumx2 -
sumx*sumx));
                b = ((sumy - a*sumx) / (double)no_data);
                angle = atan(a) *  180.0 / PI;
        printf("The line is X=%3.3f +%3.3f Y,      angle=%lf degree₩n", b, a,
angle);
                *cc = a;
                *dd = b;
```

```
        }
        return angle;
}
```

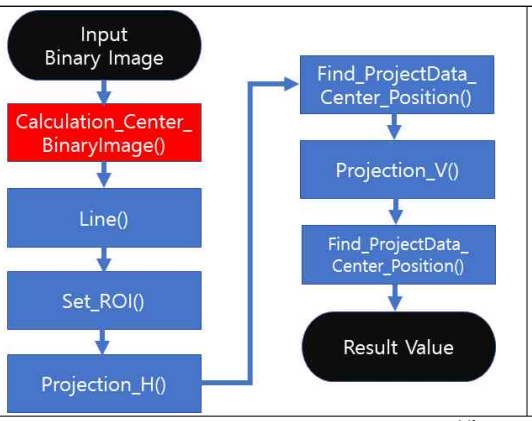


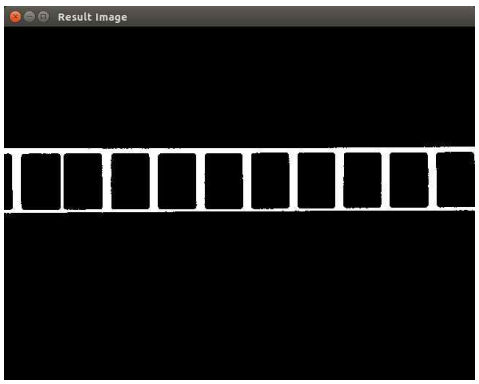


LinearCurveFitting_V()에서 보내온 회전각도를 확인하고 회전판을 회전시킨다. 전달받은 회전각이 0.1도 이내일 경우에는 종료하게 된다.

```
void Motor_move(double m_rotation_angle)
{

      unsigned char buf[10];
          signed short angle;

          angle = (int)(m_rotation_angle*10);
          buf[0] = 'R';

          if(angle >=0)     buf[1] = '+';
          else                  buf[1] = '-';

          buf[2] = (angle >> 8) & 0x00ff; //top byte
          buf[3] = angle & 0x00ff; //bottom byte
          buf[4]=  '*';
          buf[5]= '₩n';
          Serial_Data_Write(buf,6);

          printf("Motor Com. data %s ₩n₩n", buf);

}
```

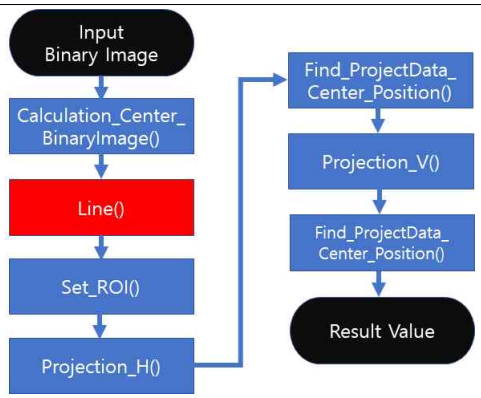
2) ERROR CHECK

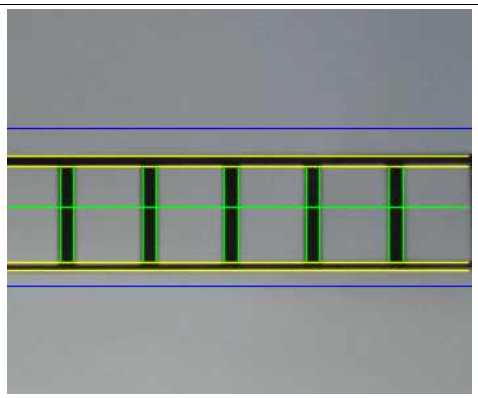




Calculation_Center_BinaryImage()는 물체(MLCC)의 무게중심을 구하기 위한 함수이다. 카메라에서 입력받은 영상에 대하여 이진화를 시킨 영상을 크기를 계산하고 영상의 물체를 검색하여 물체에 대한 중심을 찾아 중심좌표를 계산한다.

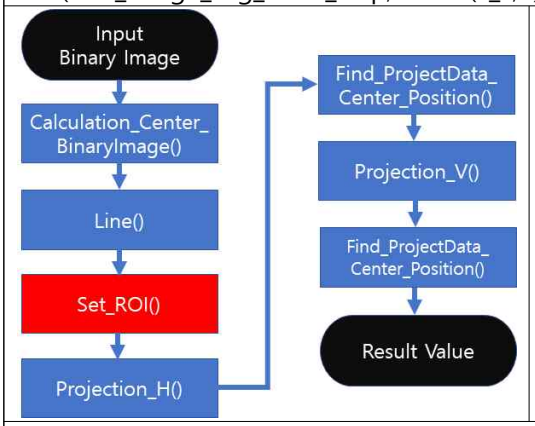
```
void Calculation_Center_BinaryImage(BYTE *Bin_Image, CSize image_size, double *x_c, double *y_c)
{
	int x, y;
	int height = image_size.cy;
	int width = image_size.cx;
	UINT area;
	UINT x_sum;
	UINT y_sum;

	area = x_sum = y_sum = 0;
	for (y = 0; y < height; y++)
	{
		for (x = 0; x < width; x++)
		{
			if (*(Bin_Image + y * width + x) == 255)
			{
				area++;
				x_sum += x;
				y_sum += y;
			}
		}
	}

	*x_c = (double)x_sum / (double)area;
	*y_c = (double)y_sum / (double)area;
}
```





Line()는 무게중심을 표시하는 함수이다. Calculation_Center_BinaryImage()에서 구해진 좌표를 이용하여 영상의 크기를 계산하고 화면상에 라인을 그려서 표시해준다.

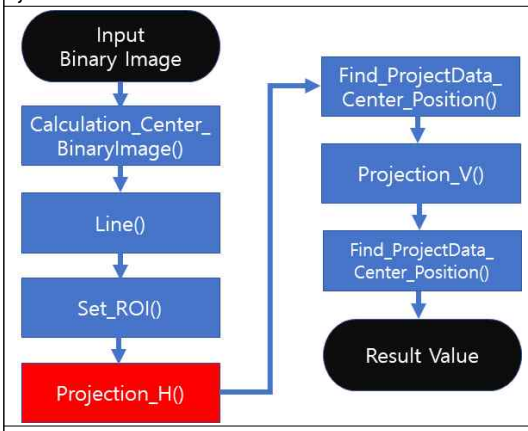
```
line(mat_image_org_color_disp, Point(0,y_c),Point(640,y_c),GREEN,1,LINE_AA);
line(mat_image_org_color_disp, Point(x_c,0),Point(x_c,480),GREEN,1,LINE_AA);
```



Set_ROI()는 무게중심에 있는 검사할 물체에 대하여 검사영역을 설정하게 된다. Line()에 의하여 표시된 무게중심을 기준으로 관심영역을 설정하게 된다.

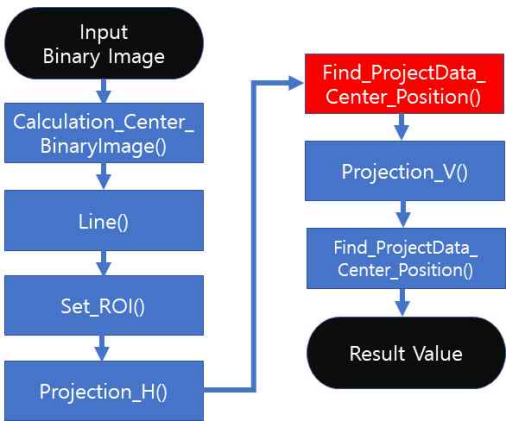
```
CRect Set_ROI(int x1,int y1,int x2, int y2)
```

```
{
        CRect temp;
        temp.left      = x1;
        temp.right     = x2;
        temp.top       =  y1;
        temp.bottom    = y2;
        return temp;
}
```



Projection_H()는 이진화 영상의 가로방향으로 Projection을 하는 함수이다. 설정된 관심영역에서 수행을 하게 된다.

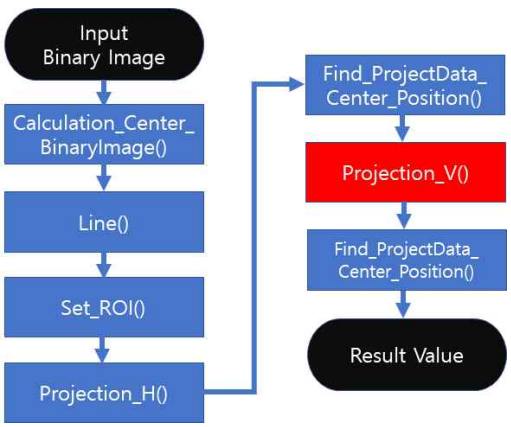
```
void Projection_H(BYTE *In_Image, CSize image_size, CRect ROI, double *projection_h)
{

        int x, y;
        int height = image_size.cy;
        int width = image_size.cx;

        memset(projection_h, 0, height*sizeof(double)) ;

        for (y = ROI.top; y < ROI.bottom; y++)
        {
            for (x = ROI.left; x < ROI.right; x++)
            {
                    (projection_h + y) += (double)(*(In_Image + y*width + x)
)/(255.0);                  }
        }
}
```



가로방향의 Projection 데이터를 가지고 Projection 데이터의 특정값들 보다 높은 영역의 리브들의 중심들을 찾는 함수이다. Projection 데이터를 바탕으로 라벨링을 하며 중심점들을 찾아가게 된다. 라벨링이 끝나게 되면 MLCC의 개수를 표시하게 되며, 리브의 중심을 추출하게 된다.

```
no_label = 0;
        for (x = 0; x < size_projection; x++)
        {
                if (*(projection_temp + x) == 255 && *(Label + x) == 0)
                {
```

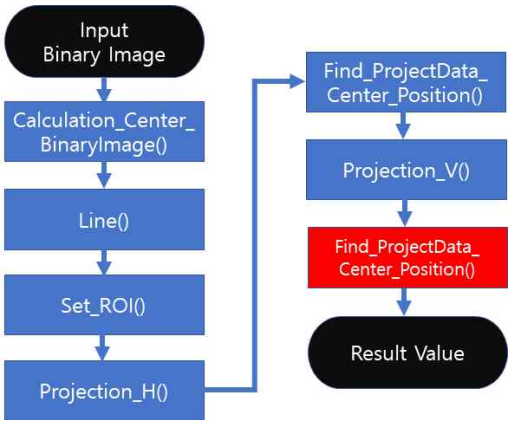
```
                        no_label++;
                        *(Label + x) = no_label;
                            grass(projection_temp, Label, size_projection, x,
no_label);
                }
        }
    start_x = 0;
        no_label = (no_label > m_max_label) ? m_max_label : no_label;
        for (k = 1, index_cnt = 0; k <= no_label; k++)
        {
                position_sum = 0;
                no_data = 0;
                max_pos = 0, min_pos = size_projection;
                for (x = start_x; x < size_projection; x++)
                {
                        if (*(Label + x) == k)
                        {
                                position_sum += (double)x;
                                no_data++;
                                start_x = x;

                                if (max_pos <= x)  max_pos = x;
                                if (min_pos >= x)  min_pos = x;
                        }
                }
                *(position + k - 1) = position_sum / no_data;
                Projection_Blob[index_cnt].cx = min_pos;
                Projection_Blob[index_cnt].cy = max_pos;

                if ( ((max_pos - min_pos) >= m_min_distance) && ((max_pos -
min_pos) <= m_max_distance) )
                {
                  index_cnt++;
                }
        }
```

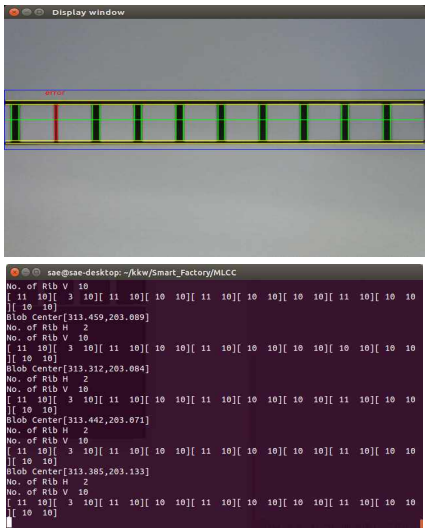


Projection_V()는 이진화 영상의 세로방향으로 Projection을 하는 함수이다. 설정된 관심영역에서 수행을 하게 된다.

```
void Projection_V(BYTE *In_Image, CSize image_size, CRect ROI, double
*projection_v)
{

        int x, y;
        int height = image_size.cy;
        int width = image_size.cx;
```

```
        memset(projection_v, 0, width*sizeof(double));

        for (x = ROI.left; x < ROI.right; x++)
        {
          for (y = ROI.top; y < ROI.bottom; y++)
          {
                *(projection_v + x) += (double)(*(In_Image + y*width + x)
)/(255.0);
          }
        }
}
```





세로방향의 Projection 데이터를 가지고 Projection 데이터의 특정값들 보다 높은 영역의 리브들의 중심들을 찾는 함수이다. Projection 데이터를 바탕으로 라벨링을 하며 중심점들을 찾아가게 된다. 라벨링이 끝나게 되면 MLCC의 개수를 표시하게 되며, 리브의 중심을 추출하게 된다. 가로,세로 방향으로 리브를 검출하게 되면 리브들의 폭을 구하게 되고 리브들의 폭이 정해진 규정보다 작으면 양품으로 표시하고 그렇지 않다면 불량으로 표시한다.

```
no_label = 0;
        for (x = 0; x < size_projection; x++)
        {
                if (*(projection_temp + x) == 255 && *(Label + x) == 0)
                {
                        no_label++;
                        *(Label + x) = no_label;
                           grass(projection_temp, Label, size_projection, x,
no_label);
                }
        }
        start_x = 0;
        no_label = (no_label > m_max_label) ? m_max_label : no_label;
        for (k = 1, index_cnt = 0; k <= no_label; k++)
        {
                position_sum = 0;
                no_data = 0;
                max_pos = 0, min_pos = size_projection;
                for (x = start_x; x < size_projection; x++)
                {
                        if (*(Label + x) == k)
```

```
                    {
                        position_sum += (double)x;
                        no_data++;
                        start_x = x;

                        if (max_pos <= x)  max_pos = x;
                        if (min_pos >= x)  min_pos = x;
                    }
                }
                *(position + k - 1) = position_sum / no_data;
                Projection_Blob[index_cnt].cx = min_pos;
                Projection_Blob[index_cnt].cy = max_pos;

                if ( ((max_pos - min_pos) >= m_min_distance) && ((max_pos -
min_pos) <= m_max_distance) )
                {
                  index_cnt++;
                }
        }
```

3.4. 기술적 차별성

- 3대의 카메라를 사용함으로서 다방면의 위치에서 MLCC 단면 촬영이 가능하며 불량 및 품질 판단의 정확성을 높였다.
- 컨베이어벨트 관련 프로그램 저작권 등록을 완료해 보유하고 있다.
- MLCC 단면을 보기 위해 Rotation Table을 사용해 지정 각도 값을 pulse값으로 변환하여 MLCC의 단면을 볼 수 있게 회전시켜준다.
- Rotation Table에서 뿐만 아니라 Linear Belt에서도 초점 조절을 한 후, 단면 인식이 가능하다.
- 디지털 트윈 기술을 활용해 시뮬레이션을 기반으로 한 기구와 SW를 가상제조 하여 실물 제작을 진행해 제조비용과 작업 스케줄을 절약 및 최적화 할 수 있다.

그림 32. Virtual Factory 설계

- 가상현실과 실제 H/W를 연동하는 프로그램을 개발진행 중이며 이를 통해 작동 환경에 VR 기술을 적용하여 생산 효율을 높일 수 있다.

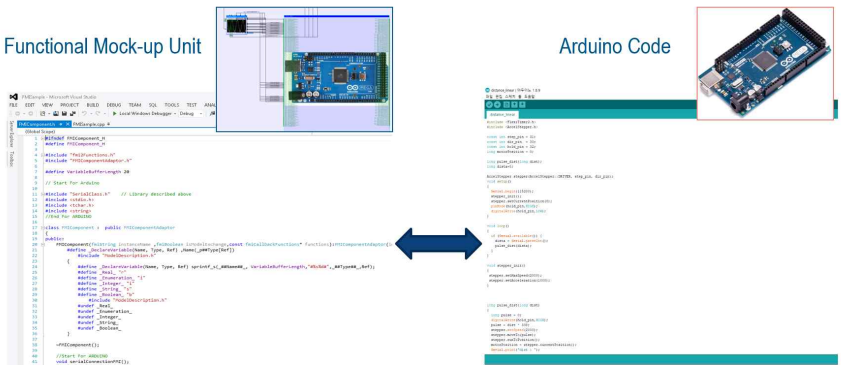


그림 33. Software in the loop(SiL)

4. 개발 중 장애요인과 해결방안

- 벨트 풀리 제작

Stepping Motor를 돌리기 위해선 벨트 풀리를 통해 Conveyor Belt를 작동을 시켜야 하는데, 벨트 풀리를 아크릴로 제작을 했을 경우, 모터의 토크가 풀리에 대응을 하질 못해서 어려움을 겪었다. 회의를 통해 얻은 결과, 아크릴 제작 대신에 가공 제작하기로 했다.

- 카메라 위치 조정

카메라로 MLCC를 정확한 위치에서 촬영을 하려면 카메라의 위치를 조정해야 하는데, 카메라를 높은 위치에 장착을 해야 할지, 낮은 위치에 장착을 해 보았고, 시행착오 끝에 정확한 위치를 선정했다.

- Angle Stage 원점 조정 부정확

Rotation Table에서 원점 조정, 즉 360° 또는 180°를 지정했을 때, 원점으로 조정이 약간의 오차가 발생했는데, 이를 해결하기 위해, 0°일 때 신호를 발생하는 역할을 하는 Encoder 기능을 추가해서 원점 조정에 정확성을 늘렸다.

- 반영구적 작동 불가

MLCC 샘플이 Rotation Table에 도착을 했을 때, 반영구적으로 지정 각도에 맞게 회전이 반영구적으로 되어야 하는데, 초기 단계로 돌아가지 않고 멈춘 상태로 작동이 불가해서 여러 가지 방법을 찾으려 했으나 해결이 안 되서 어려움을 겪었다. 팀원들과의 회의를 통해 프로그램에 반영구적인 함수 'Interrupt'를 사용하기로 했다.

- 멈춤 현상 발생

Rotation Table에서 각도가 0°일 경우, 원점으로 지정 명령을 실행할 때, Rotation Table에 원점으로 지정 명령 실행을 했을 시, Rotation Table이 멈춤 현상이 발생했다. 따라서 회의를 통해 얻은 해결책은 프로그램의 setup부분에서 원점을 설정 한 후, 실행했다.

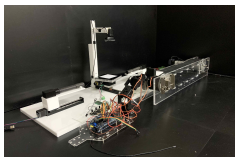
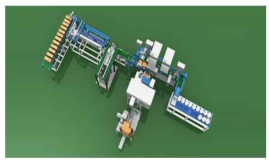
- 카메라 중점 왜곡

카메라로 영상처리를 하려면 카메라의 중점을 맞춰서 영상처리를 하려 했으나, 중점에 맞춰서 할 경우 카메라가 왜곡이 된다. 해결책을 찾아본 결과, 카메라 모델을 이용해서 왜곡된 영상을 변환 뒤 보정하고, 수식을 사용해서 해결했다.

5. 개발결과물의 차별성

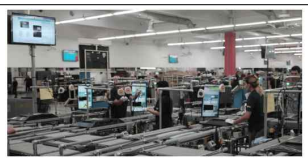
타사와 비교하였을 때 팀 개발 결과물 분석표는 표 5.1과 같다.

표 5.1. 타사와 팀 개발 결과물 분석

결과물 이름	팀 개발 결과물	삼성전기 공장	솔로몬 메카닉스
사진			
주 생산품	MLCC	전장용 MLCC	MLCC
기술적 특징	• 가상공간 구현 가능 • 검사한 이미지를 서버 전송 가능	• 다양한 환경에서 실시간 제조 현장 모니터링 가능	• 자동으로 인식 검인, 화상처리, 서류 공급 등을 수행
비고	-	MLCC 생산 종류 다양	-

SMART MLCC Cutting Inspection에 따른 기대효과는 표 5.2와 같다.

표 5.2. 기대효과

주요 대상	기대 효과
 전자제품 제조업체	고성능 전자제품 제조 가능
 자동차 제조업체	효율이 좋은 자동차 제작 가능
 소비자	높은 사양 전자제품 및 자동차 사용 가능

6. 단계별 개발계획 및 실제 참여인원 및 업무 분장

6.1 참여 인원별 업무 분장 계획

표 2. 참여 인원별 업무 분장 계획

구성원	역할(담당 기술개발 내용)	비중
서경철 (팀장)	◦ 임베디드 시스템 환경 구축 ◦ 메카트로닉스 장치 구현 - CPS 및 DIGITAL TWINS 구축 - XYZ로봇 설계 및 제작 - 컨베이어벨트 설계 및 제작 - 턴테이블 설계 및 제작 - 리니어 스테이지 설계 및 제작 ◦ 영상 처리 알고리즘 구현 - 파노라마 영상 인식 알고리즘 - 불량 확인 알고리즘 ◦ 카메라 왜곡 교정 알고리즘 개발	20%

김성재	◦ 메카트로닉스 장치 구현 - CPS 및 DIGITAL TWINS 구축 - XYZ로봇 설계 및 제작 - 컨베이어벨트 설계 및 제작 - 턴테이블 설계 및 제작 - 리니어 스테이지 설계 및 제작 ◦ 딥러닝 알고리즘 개발 - Visual Studio C++	15%
김태영	◦ 임베디드 시스템 환경 구축 ◦ 영상처리 알고리즘 구현 - 외각선 이진화 추출 알고리즘 - 불량 확인 알고리즘 ◦ MLCC SHEET ALINE 알고리즘 - MLCC 외각선 이진화 추출 - UV좌표를 XY좌표로 변환 ◦ 카메라 왜곡 교정 알고리즘 개발 ◦ UI 개발	15%
박정대	◦ 임베디드 시스템 환경 구축 ◦ Machine control 알고리즘 개발 -ARDUINO, Linear Servo motor ◦ MLCC SHEET ALINE 알고리즘 - MLCC 외각선 이진화 추출 - UV좌표를 XY좌표로 변환 ◦ 카메라 왜곡 교정 알고리즘 개발 ◦ UI 개발	20%
선준형	◦ 메카트로닉스 장치 구현 - XYZ로봇 설계 및 제작 ◦ 딥러닝 알고리즘 개발 -Visual Studio C++ ◦ 지적재산권 조사 및 출원	15%
공통	◦ 계획수립 및 자료조사 ◦ 동영상 제작 ◦ 보고서 작성	15%

6.2 단계별 개발계획

표 3. 단계별 개발 계획

개발내용		수행인	기술 개발 일정																							
			6월				7월				8월				9월				10월				11월			
			1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
기반구축	계획수립 및 자료조사	공동																								
	임베디드 시스템 환경 구축	김태영 박정대 서경철																								
	CPS 및 DIGITAL TWINS 구축	김성재 서경철																								
	지적재산권 조사	선준형																								
비전 (영상처리)	파노라마 영상 인식 알고리즘	서경철																								
	외각선 이진화 추출 알고리즘	김태영																								
	카메라 왜곡 교정 알고리즘	김태영 박정대 서경철																								
	불량 확인 알고리즘	김태영 서경철																								
	Machine control 알고리즘	박정대																								
	MLCC SHEET ALINE 알고리즘	김태영 박정대																								
	UI	김태영 박정대																								
	딥러닝 알고리즘	김성재 선준형																								
메카트로닉스 장치 구현	컨베이어벨트 설계 및 제작	김성재 서경철 선준형																								
	턴테이블 설계 및 제작	김성재 서경철																								
	리니어 스테이지 설계 및 제작	김성재 서경철																								
	검사부 설계 및 제작	김성재 서경철																								
각 모듈별 검사 테스트		공동																								
모듈 통합		공동																								
Debugging		공동																								
종합 테스트 및 분석(동영상 제작)		공동																								
지적 재산권 출원		선준형																								
보고서 작성		공동																								