

Cascade 방식을 활용한 공위치 제어장치

김상무, 최태오, 서제교, 신한룡

서울과학기술대학교, 서울특별시 노원구 공릉로232

A ball position controller using a Cascade method

Kim sangmoo, Choi theo, Seo jegyo, Shin hanryung

Seoul National University of Science and Technology 232, Seoul

요약

최근 거친 파도에 밀려 추진력을 상실한 선박의 복원력 부실성으로 인하여 컨테이너 화물의 낙하사고가 많이 발생하고 있다. 대부분의 선박은 화물의 위치 및 속도를 제어하는 별도의 제어기 시스템이 없다. 존재하는 경우에도 대부분 위치에 대한 간접적인 피드백 시스템 구조를 통하여 위치를 제어하고 있다. 본 연구는 이러한 문제를 해결하기 위하여 Cascade controller를 물체의 자세 제어에 이용하여 속도와 위치를 모두 제어할 수 있는 제어기 시스템을 만드는 것을 목적으로 한다. 물체 자세제어 장치의 가장 대표적인 '볼 밸런싱' 장치를 활용하여 Cascade controller로 문제를 해결 할 수 있는지 판단하였다. 실험 결과, 원하는 속도로 물체를 원하는 위치에 위치시킬 수 있었으며 이에 따라 여러 화물들의 위치와 자세를 제어하며 이동시킬 수 있게 되었다. 이는 더 나아가 유체와 파괴되기 쉬운 화물과 물체들도 이동시킬 때 안정적으로 운송하는 것에 큰 기여를 할 것이라 판단된다.

주제어 : 위치제어, 속도 제어, cascade 제어

ABSTRACT

Recently, there have been many drop accidents of container cargo due to the poor resilience of ships that have lost propulsion due to rough waves. Most ships do not have a separate controller system to control the position and speed of cargo. Even if it exists, most of the locations are controlled through an indirect feedback system structure. To solve this problem, this research aims to create a controller system that can control both speed and position by using a Cascade controller to control the posture of an object. It was determined whether Cascade controller could solve the problem using the most representative 'ball balancing' device of the object posture control device. As a result of the experiment, it was possible to position the object in the desired position at the desired speed, thereby controlling the position and posture of various cargoes and moving them. It is believed that this will further contribute to stable transportation when moving fluids and easily destroyed cargo and objects.

Key : Position control, speed control, cascade control

I. 서론



21년 2월 경 일본 북부를 지나던 머스크 사(社)의 애안트호벤호에서 컨테이너 박스 260개가 선박 밖으로 낙하하는 사고가 있었다. 거친 파도에 밀려 추진력을 상실한 선박의 복원력이 약해졌고 260개의 컨테이너가 바다 속으로, 갑판 위에 있던 65개의 컨테이너는 흔들리며 선체에 손상을 입혔다. 이와 같은 사고를 방지하고 운송 중인 물체를 보호하기 위한 방안을 탐색해보았다. 그 중 이동 중이거나 흔들리는 평면 위에서 구르는 물체를 범위 밖으로 벗어나지 않게 하는 목적으로 개발된 장치가 있었다.

물체(공)의 현 위치와 기준점의 상대적인 위치에 따라 평면을 기울여 기준점으로 이동시키는 방식으로 작동하는 공 위치 제어장치 (이하 "밸런싱볼")가 그것이다. 기존의 밸런싱볼은 물체의 위치에 대한 상대적인 평면 기울기의 크기와 방향으로 이를 제어한다.

그러나 이와 같은 방법으로 제어할 시에는 물체의 위치는 보정할 수 있으나 물체의 이동 속도는 제어할 수 없다. 위치를 보정하는 과정에서 발생하는 통제되지 않은 속도는 물체의 마모 혹은 파손을 야기할 수 있다.

이를 해결하기 위해 고안해낸 "Cascade 방식을 활용한 공 위치 제어장치"는 PID 제어를 이중으로 사용한다. 위치 제어와 속도 제어를 병행하여 제어장치는 기존의 밸런싱 볼보다 정밀하게 작동한다. 새롭게 고안된 제어 방식을 통해 파손의 위협으로부터 물체를 보호하는 효과를 기대해 볼 수 있다.

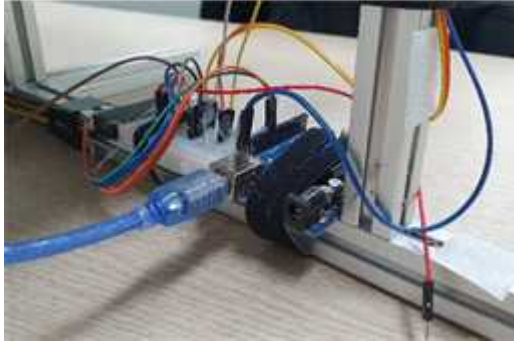
II. 하드웨어 설명



먼저 프로파일로 기본 프레임을 설계했다. 그리고 판이 전 방향으로 기울 수 있도록 universal joint를 이용하여 판과 프로파일을 연결했다.




브레드 보드와 아두이노는 프레임 하부에 가동 범위에 방해되지 않게 안전하게 설치했다. 대부분의 부품은 글루건을 통해 단단하게 프로파일 단면에 부착했다.




프레임 상부에는 영상인식을 위한 카메라 HuskyLens Pro를 부착했다. HuskyLens Pro에 대한 설명은 뒤에 부록을 참고하길 바란다.

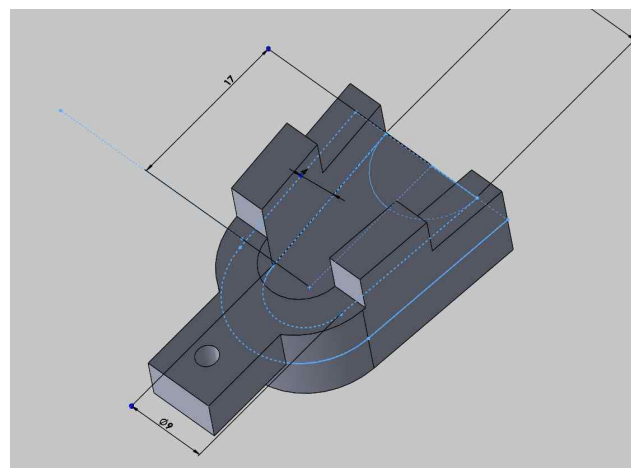
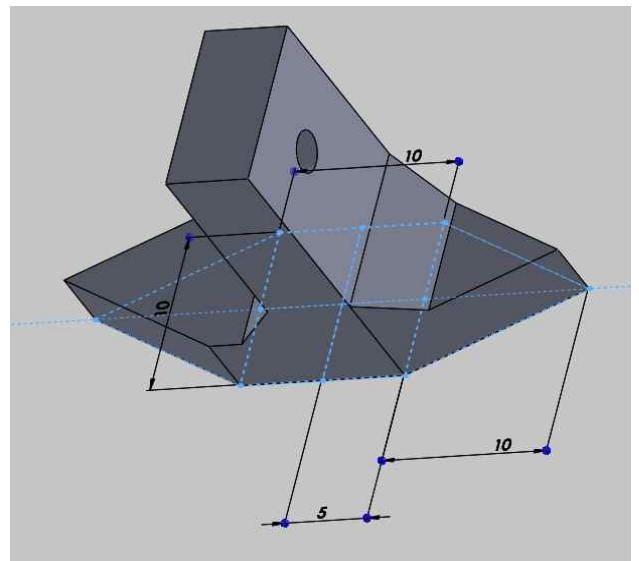
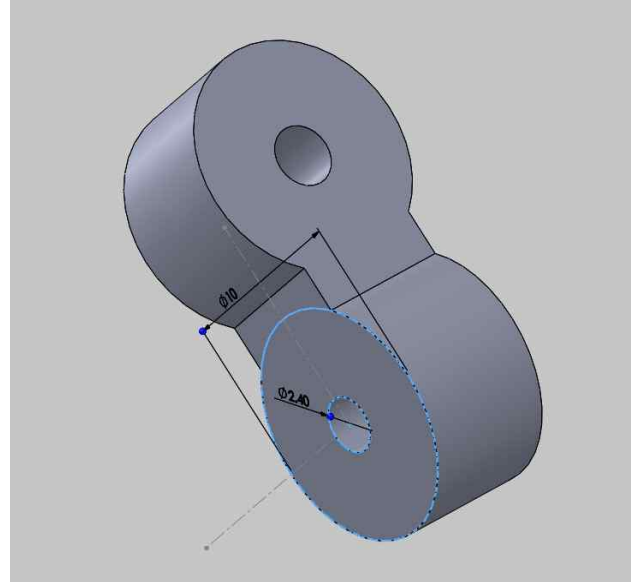


판의 기울기를 제어하는 과정에서는 x축, y축에 각각 서보모터를 사용했다.

	HS-311	TORQUE	3.0 / 3.5 kg (4.8V/6.0V)
		SPEED	0.19 / 0.15 (4.8V/6.0V)
		SIZE	41 x 20 x 37 mm
		WEIGHT	48.5 g

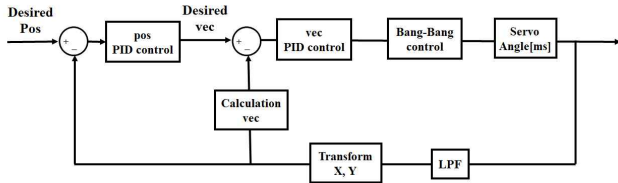
	HS-300	TORQUE	3.02 kg (4.8V/6.0V)
		SPEED	0.19 / 0.15 (4.8V/6.0V)
		SIZE	41 x 20 x 37 mm
		WEIGHT	47.1 g

서보 모터는 x, y축에 서보 암이 각각 수직이 되도록 부착했다. 서보암의 각도에 따라 공에 가해지는 가속도 값이 결정되는 구조이다. CAD 수업 때 배운 내용을 바탕으로 솔리드웍스로 서보암 부착물과 판에 이을 joint를 설계하고 3D 프린터 기로 인쇄했다.



Ⅲ. 소프트웨어 설명

i) 전체 구성도



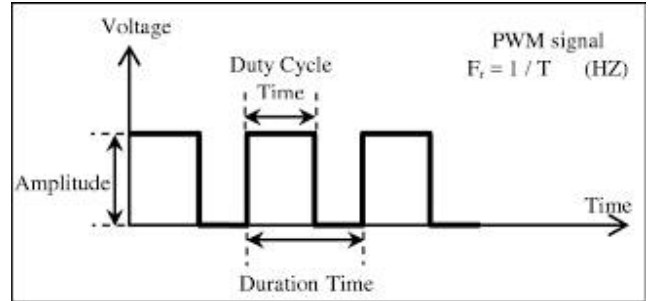
전체 구성도는 위와 같은 형태로 작동된다.

사용자는 Desired Pos 즉, 원하는 특정 위치를 입력하게 된다. 입력된 값은 센서로부터 받아오는 질점의 좌표값을 통하여 비교하게 된다.

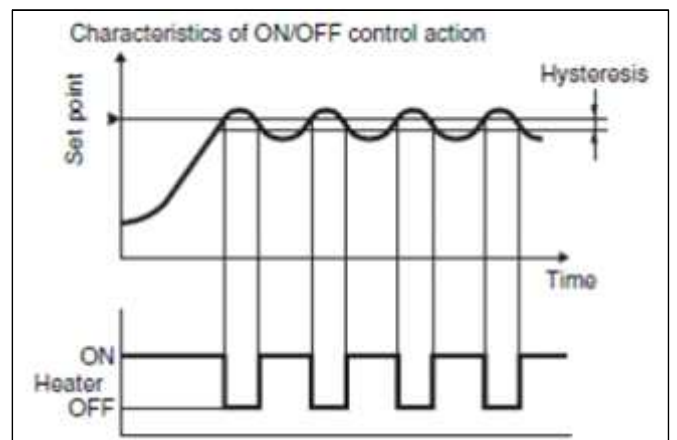
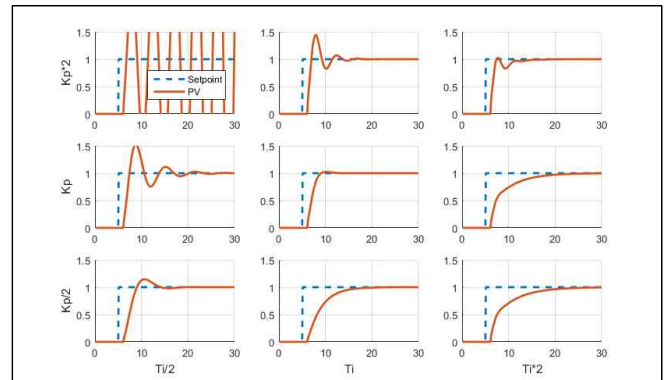
센서에서 받는 값은 질점이 카메라 센서에서 Detecting되는 상대적 위치 값을 픽셀 값으로 반환한다. 이에 따라, 본 장치의 좌표계와 일치시키고, 축척값을 조정하기 위하여 Transform 행렬을 거치게 된다. Transform 행렬에 연산되기 위해서는 Noise가 없는 값으로 연산해야하므로 Low-pass filter를 마이크로 컨트롤러의 코딩으로서 구현하여 Noise가 없는 값을 추정하여 연산에 활용하였다.

Desired Position 과 sensing Position 과의 차이값은 1th PID controller의 error항으로 연산에 이용된다. 1th PID controller에서 반환되는 값은 Desired Position 에 가기 위한 Desired speed 에 해당하는 값으로 계산된다.

반환된 Desired speed는 Transform 행렬을 통해 얻은 위치 값을 통한 이산적인 현재 추정 속도와 비교된다. Desired speed 와 sensing speed 와의 차이 값은 2th PID controller의 error항으로 연산에 이용된다. 2th PID controller에서 반환되는 값은 Desired speed 에 가기 위한 Servo에 인가하는 PWM의 Duty[ms]에 해당하는 값으로 계산된다.



이때, PID controller의 gain에 대한 연산은 humanity methods를 활용하였다. 이에 따라 완벽한 PID controller의 gain을 구할 수 없었기에 최종적인 steady state 상에서 약간의 Hysteresis가 관측되었다. 이에 따라 ON - OFF controller의 일종인 Bang-Bang controller를 도입하여 일정 steady state 에 수렴하였다고 판단되었을 때, Duty[ms]를 0으로 만들어 수렴되도록 설정하였다.



ii) 전체 클래스 구성도

x좌표와 y좌표에서 각각의 위치값과 속도를 받아들이 제어하는 과정을 클래스 `coor`로 작성하였다. 허스키렌즈로 받아들이 각 좌표의 위치값을 클래스 내의 멤버변수 `r`에 저장한다. 이후 클래스 `matrix`에서 행렬식을 통해 픽셀 단위의 위치값을 `cm` 단위로 변환한다.

변환한 위치값을 `coor`의 멤버함수 `calcul`에서 `dt(ms)` 전의 위치값 `r_past`와 비교하여 속도를 계산하고 이를 멤버변수 `V`에 저장한다. `r`과 `V`의 값을 연속적인 값들로 표현하기 위해 멤버함수 `lowpassfilter`를 사용한다. 상수를 데이터에 곱하면 필터값에 더하면 연속적인 값을 가지게 되어 동작 시 모터의 급회전을 방지한다. 필터링된 위치값과 속도값은 멤버변수 `r_f`, `V_f`에 각각 저장된다.

필터링을 거친 값을 통해 멤버함수 `computePID()`에서 모터에 필요한 제어값을 구한다. `computePID`는 멤버변수 `V_f`(혹은 `r_f`), PID제어에 필요한 `P`, `I`, `D` Gain(`Kp`, `Ki`, `Kd`), 그리고 값의 수렴을 위한 `bangbang_control_range` 등을 매개변수로 사용한다. 이 함수에서 모터를 제어하기 위해 필요한 값을 구하여 `coor`의 매개변수 `u`에 저장한다. 모터 제어 시 이 매개변수 `u`가 제어값으로 사용된다.

허스키렌즈에서 받은 픽셀 단위의 값을 `cm` 단위로 변환하여 출력하는 과정이 클래스 `matrix`에 포함된다. 위치값에서 속도값을 구하고, 각 값을 필터링하여 제어에 필요한 값을 구하는 것이 클래스 `coor`의 역할이다. 두 클래스의 헤더파일을 생성하고 이를 `include`하여 각 변환값과 제어값을 얻어 제어장치가 작동하게 된다.

IV. 결론

해상으로 혹은 공중으로 화물을 운송하는 과정에서 화물의 흔들림과 이동은 불가피하다. 이 과정에서 앞에서 언급한 것과 같이 화물의 파손과 유실이 발생하게 된다. 이를 해결하기 위한 수단으로 위치 제어 장치가 제안될 수 있으나 이 역시 완전한 해결책이 될 수는 없다. 위치를 제어하는 과정에서 발생하는 급가속으로 인한 위험성이 존재하기 때문이다. 보다 정밀하고 정확한 제어를 위해 `Cascade` 방식을 제시하여 장치를 개발했다. 이를 적용한 장치는 위치와 속도를 함께 제어하여 상기 목적에 더 적합하게 사용될 수 있다. `Cascade` 방식을 활용한 위치 제어 장치를 통해 물체는 외부 환경으로 기인되는 이동의 반경과 속도가 제어된다. 이를 통해 화물의 운송에서 안정성을 보장할 수 있다. 나아가 유체 혹은 위험물과 같이 운반이 어려운 화물의 경우, 그 해결방안으로 제시될 수 있을 것이다.

VI. 참고문헌

[1] dfrobot HUSKYLENS, 2022, viewed 15 December 2022.

(https://wiki.dfrobot.com/HUSKYLENS_V1.0_SKU_SEN0305_SEN0336)

[2] g-enews, 2022, viewed 15 December

(https://news.g-enews.com/ko-kr/news/article/news_all/202102261456026912ce58317c16_1/article.html?md=20210226180537_U)

[3] 아날로그반도체공학, 2022, viewed 15 Dec

[4] 공학수학, 2022, viewed 15 December

V. 발생한 장애요인 및 해결방안

- HUSKYLENS 문제 -

Prob)

현재 HUSKYLENS는 계속된 전원의 불안정한 공급으로 모듈의 파워가 꺼졌다 켜짐을 반복하고 있다. 이는 HUSKYLENS에 불안정한 전력의 공급으로 과전류가 흘러 모듈의 고장을 막기 위해 전원이 꺼지는 것으로 판단된다.

Sol)

파워 서플라이의 불안정한 전력의 공급을 막기 위해 회로에 zener diode를 달아 허용전압 이상의 전압을 막아주고 free-wheeling diode를 달아 높은 전류로 인한 모듈의 손상을 막아준다.

- 수렴 위치 흔들림 -

Prob)

현재 ball balancing plate는 HUSKYLENS에서 오는 좌표의 x, y값을 Lowpass filter를 통해 필터링해서 받아오고 있다. 또한 bang-bang control을 통해 원하는 desired value에 수렴하도록 제어를 했다. 하지만 정확하지 않는 값을 얻게 되고 이 과정에서 원하는 위치에 수렴하지 않는 문제가 발생한다.

Sol)

해결방안으로는 두 가지가 있다. 첫 번째는 PID control에서의 I gain을 조정하는 것이다. 현재는 원하는 위치에 수렴하지 못해 bang-bang control로 control action을 ON/OFF하는 방식을 사용한다.

하지만 이 방법은 공의 역학적 움직임을 전혀 고려하지 않는 control로 이를 I gain을 조절함으로써 부드럽게 맞출 수 있다. 두 번째로는 제어 control system의 dt값을 줄이는 것이다. 제어 control의 dt값을 줄이면 제어 cycle수가 늘어남으로 더욱 정밀한 제어가 가능해진다.

- 서플라이 전원 불안정 -

Prob)

현재 사용하는 power supply는 정확한 9V의 전압과 전류를 제공하지 못하여 전체 product에 문제가 생긴다.

Sol)

Power supply는 교류를 직류로 변환하고(AC to DC) 전기 기구의 전기 공급을 책임지는 중요한 장치로, 신뢰도가 높은 더 좋은 power supply를 사용한다.

- 전력문제 -

Prob)

프로젝트 초기에 계획했던 9V의 알칼리배터리의 사용은 모터 2개를 제어하기에 충분치 않았고, 자꾸 꺼지는 이슈가 발생했다.

Sol)

배터리의 전력은 V의 값은 같아도, I가 매우 작아 원하는 소비전력이 나오지 못한다. 그래서 모터 torque를 생성해 낼 만큼의 무한정의 큰 I값을 가진 power supply를 사용하였다.

- Universal joint의 xy 종속 -

Prob)

universal joint는 두 개의 회전축을 연결하면서 동시에 각각의 회전축을 대해 상대운동이 가능하도록 하는 연결장치이다. 하지만 이것의 수직축과 회전축을 연결하는 shaft가 서로 독립적으로 움직이지 못하고 진동이 발생하였다 .

Sol)

universal joint의 x축과 y축의 종속 문제는 joint 부분을 ball joint로 교체함으로써 해결이 가능하다. ball joint중에 x,y축만의 2-D(2개의 자유도)만을 사용하여 각 축이 독립적으로 움직이게 할 수 있다

VI. 부록

- float lowpassfilter()

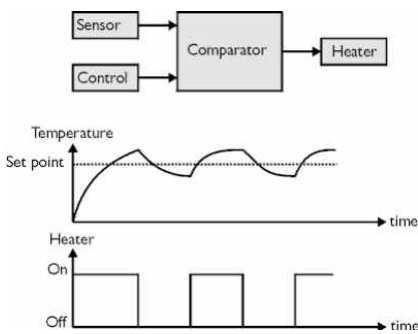
(filter) 변수에 새로운 (data)변수가 (lowpass_constant)변수만큼의 가중치로 low pass filtering을 연산하는 필터링 함수

Input data	control function	Output
filter data lowpass_constant	$X_{filter} = (1 - \alpha)x_{data} + \alpha X_{filter}$ $\alpha = lowpass\ constant$	filter

Code
<pre>float lowpassfilter (float filter, float data, float lowpass_constant) { filter = data * (1 - lowpass_constant) + filter * lowpass_constant; return filter; }</pre>

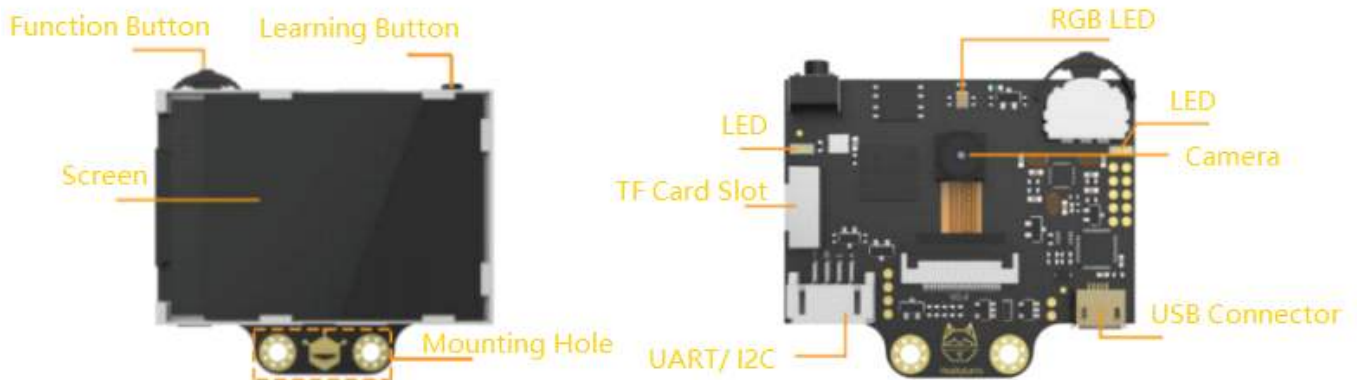
- computePID()

사용자가 원하는 위치 r (Desired Value)에 대하여 실제 위치 y (Measured Value)의 오차를 구해내는 함수이며 PID control을 통하여 보상을 위한 PWM(Pulse width Modulation)의 Duty[ms] 값을 반환하는 함수

Input data	control function	Output
double r double data, double dt, double u, double Kp, double Ki, double Kd, double bangbang_control_range	<p>1. PWM Duty 생성</p> <p>Desired Value와 Measured Value간의 차이값을 (error)값에 저장한다. 그리고 PID제어기를 활용하여 Control Value(Duty) u를 산출한다.</p> $u [Hz] = K_P * error + K_I * \int_{t_{i-1}}^{t_i} error dt + K_D \frac{d(error)}{dt}$ <p>2. Bang-Bang control</p> <p>Measured Value가 Desired Value에 일정 범위 내에 수렴하는 경향을 보이면, 두 값이 같다고 판정하여 빠르게 steady state에 접근시킨다.</p> 	u (Duty[ms])

Input data	control function	Output
double x double y double A[3][1] double D[3][1] double E[3][1] double C[3][3] double B[3][3] int i , j	<p>1.matrix의 생성</p> $A = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ $B = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$ $C = \begin{bmatrix} x.s.f & 0 & 0 \\ 0 & y.s.f & 0 \\ 0 & 0 & 1 \end{bmatrix}$ <p>(x•s•f = x_scale_factor, y•s•f = y_scale_factor)</p> <p>dx = -180, dy = -180 x_scale_factor = 0.165 y_scale_factor = 0.164 dx, dy = 좌표값 변환 값 x_scale_factor, y_scale_factor = 좌표축 변환 값</p> <p>2. matrix 계산</p> $D = B \times A = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 + dx \\ 0 + dy \\ 1 \end{bmatrix}$ $E = C \times D = \begin{bmatrix} x.s.f & 0 & 0 \\ 0 & y.s.f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} dx \\ dy \\ 1 \end{bmatrix} = \begin{bmatrix} (0 + dx)(x.s.f) \\ (0 + dy)(y.s.f) \\ 1 \end{bmatrix}$ <p>기존 HUSKYLENS에서 픽셀로 계산되던 x,y 좌표값을 HUSKYLENS의 중앙으로 좌표축을 변환하고 그 값을 (0,0)으로 정한다.</p>	E[0][0] E[1][0]

- HUSKYLENS DATA Sheet



Processor: Kendryte K210

Image Sensor:

SEN0305 HuskyLens: OV2640 (2.0Megapixel Camera)

SEN0336 HuskyLens PRO: OV5640 (5.0MegaPixel Camera)

Supply Voltage: 3.3~5.0V

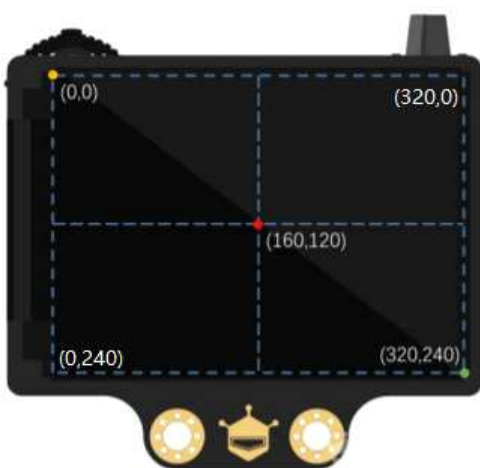
Consumption(TYP): 320mA@3.3V, 230mA@5.0V

Communication Port: UART; I2C

Display: 2.0-inch IPS screen with 320*240 resolution

Built-in Algorithms: Face Recognition, Object Tracking, Object Recognition, Line Tracking, Color Recognition, Tag Recognition, Object Classification

Dimension: 52mm x 44.5mm (2.05*1.75 inch)



VII. 코드 설명

term_project_Final_V.3.4.1.ino

```
//Cascade 클래스
#include "coor.h"
//Matrix 클래스
#include "matrix.h"

Servo x_servo;
Servo y_servo;
HUSKYLENS ball_tracker;

//cascade 행렬 객체선언
coor x;
coor y;
//matrix 선언
mat xyplot;

void setup() {
  Serial.begin(115200);
  //HuskyLens setup
  Wire.begin();
  ball_tracker.begin(Wire);
  //servo attach <coor.h>
  x_servo.attach(X_SERVO);
  y_servo.attach(Y_SERVO);
  //뱅뱅컨트롤 값
  x.bangrange = 5.5;
  y.bangrange = 2.5;
  //튜닝용 Kp 값
  x.u_kp = 0.7;
  y.u_kp = 0.6;
  //중심좌표(판 기준)
  x.c = 150;
  y.c = 140;
}

void loop() {
  //시간 측정 시작
  x.millisTime_i = millis();
  y.millisTime_i = millis();
  //huskyLens 설정
  ball_tracker.request();
  ball_tracker.isLearned();
  //물체 감지하는 동안 발동
```

```

while (ball_tracker.available()) {
    //huskylens 값 읽기
    HUSKYLENSResult result = ball_tracker.read();
    //Matrix update
    xyplot.update(result.xCenter, result.yCenter);
    x.r = xyplot.getx();
    y.r = xyplot.gety();
    //Cascade
    x.cascade();
    y.cascade();
    //서보암 조종
    Control_Servo(x.u, y.u);
    //시간 끝
    x.currentTimeMillis_f = millis();
    y.currentTimeMillis_f = millis();
    x.dt = x.currentTimeMillis_f - x.currentTimeMillis_i;
    y.dt = y.currentTimeMillis_f - y.currentTimeMillis_i;
}
}

//서보모터 Microseconds 제어 함수
void Control_Servo(double ctrlx, double ctrly) {
    //Microseconds 을 이용한 정밀 제어
    x_servo.writeMicroseconds(1350 + ctrlx);
    y_servo.writeMicroseconds(1520 + ctrly);
}

```

Coor.cpp

```
#include <Arduino.h>
#include "coor.h"

//속도 계산 함수
void coor::calcul(double now, double past) {
    V =(now - past) / dt;
    // 잡음 방지
    if (V == 0) {
        V = V_past;
    }
}

//Low Pass Filter
double coor ::lowpassfilter(double filter, double data, double constant) {
    filter = filter * (1 - constant) + data * constant;
    return filter;
}

//PID
double coor ::computePID(double r, double data, double dt, double u, double Kp, double
Ki, double Kd, double bangbang_control_range) {
    double error = r - data;
    double P = Kp * error;
    double I = Ki * error * dt;
    double D = Kd * (-data + data_past) / dt;
    I = constrain(I, I_min, I_max);
    data_past = data;
    I_past = I;
    u = P + I + D;
    //bangbang control
    if (abs(error) <= bangbang_control_range) {
        u = 0;
    }
    u = constrain(u, -500, 500);
    return u;
}

void coor ::cascade() {
    r_f = lowpassfilter(r_f, r, lowpass_constant); //위치 필터
    calcul(r, r_past); //속도 계산
    V_f = lowpassfilter(V_f, V, lowpass_constant); //속도 필터
    //Cascade
    u_v = computePID(c, r_f, dt, u_v, 0.042, 0, 0.91, bangrange); //1 차(위치)
    u = computePID(-u_v, V_f, dt, u, u_kp, 0, 0.09, 1); //2 차(속도)

    r_past = r;
    V_past = V;
}
```

Coord.h

```
#include "HUSKYLENS.h"
#include "SoftwareSerial.h"
#include <Servo.h>
//I constrain
#define I_max 300
#define I_min 0
//서보 핀
#define X_SERVO 9
#define Y_SERVO 10
//lowpass 상수
#define lowpass_constant 0.1

//x 축과 y 축 coordinate 클래스
class coord {
public:
    //변수 선언
    float millisTime_i; // 시간
    float millisTime_f;
    float dt = 0; // 루프 시간
    float I_past = 0;
    float data_past = 0;
    double c; // 목표 좌표 (중심좌표)
    double r; // 현재 좌표
    double r_past = 0;
    double r_f; // 필터링 된 좌표
    double V = 0; // 속도
    double V_past = 0;
    double V_f = 0;
    double u;
    double u_v = 0;
    double bangrange, u_kp;

    void cascade(); // cascade 메인 함수
    void calcul(double, double); //속도 계산 함수
    double lowpassfilter(double, double, double); //Low Pass Filter
    double computePID(double, double, double, double, double, double, double, double);
//PiD
};
```

matrix.cpp

```
#include <Arduino.h>
#include "matrix.h"

mat::mat(double x, double y) {
    A[0][0] = x; //클래스 변수 x 선언
    A[0][1] = y; //클래스 변수 y 선언
}

void mat::matcal() {
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 1; j++)
        {
            D[i][j] = B[i][0] * A[0][0] + B[i][1] * A[1][0] + B[i][2] * A[2][0]; //BXA
matrix 계산
        }
    }

    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 1; j++)
        {
            E[i][j] = C[i][0] * D[0][0] + C[i][1] * D[1][0] + C[i][2] * D[2][0]; //CXD
matrix 계산
        }
    }
}
```

matrix.h

```
// 좌표 변환값
#define dx -180
#define dy -180
// 좌표축 변환값
#define x_scale_factor 0.165
#define y_scale_factor 0.164

class mat {
public:
    double x;
    double y;
    double A[3][1] = { {0},{0},{1} }; // A matrix 설정
    double D[3][1];
    double E[3][1];
    double C[3][3] = { { x_scale_factor , 0 , 0 } , { 0 , y_scale_factor , 0 } , { 0 , 0 , 1 } };
}; // 좌표축 변환값 C matrix 설정
    double B[3][3] = { { 1 , 0 , dx } , { 0 , 1 , dy } , { 0 , 0 , 1 } }; // 좌표 변환값 B matrix
    설정
    int i, j;

    mat(double, double);
    void matcal();
    void update();
};
```