<주인을 따라오는 스마트 캐리어>

공과대학 전기정보공학부

2021-19847

이민준

Contents

1. Introduction

1.1 주제 선정 동기

누구나 한 번쯤 무거운 짐을 옮기며 싫증을 낸 적이 있을 것이다. 손으로 들고 가는 방식을 개선한답시고 캐리어나 카트에 짐을 싣고 가더라도 편리함은 잠깐일 뿐, 끌고 가는 것마저 일이 된다. 본 스마트 캐리어 프로젝트는 이러한 점에 착안하여 시작하게 되었다. 올 겨울에 기숙사에서 겨울 짐들을 캐리어에 우겨 넣고 집으로 가던 도중 무거운 짐을 끌고 가는 것에 싫증이 났다. 그래서 사용자가 캐리어를 직접 끌지 않아도 사람을 인식하고 추적해 자동으로 따라오는 캐리어를 만드는 목표를 갖게 되었다.

1.2 작품 개발 목표

본 프로젝트에서는 사람의 특징적인 point를 자동으로 인식해 따라가는 캐리어를 만들고, 이를 스마트폰 앱과 연동해 스마트폰으로 캐리어를 조종하는 기능을 구현하여 IoT 기술을 접목하고자 한다.

2. Preparation

2.1 준비물

- Hardware
  - STM32 NUCLEO-F411RE (MCU)
  - HuskyLens PRO (AI CAM)
  - HC-06 Bluetooth Module
  - JGA25-370 Geared DC Motor
  - TB6612FNG Motor Driver
  - Ultrasonic Sensor
  - 아크릴판
- Software
  - Arduino IDE
  - MIT App Inventor
  - Fusion 360

2.2 개발 계획

2.2.1 외형 제작

200mm x 500mm 아크릴판 2개, 200mm x 500mm 아크릴판 2개를 활용해 직육면체 형태의 캐리어 외형을 제작한다. 아크릴판과 아크릴판을 연결하는 부품은 3D 프린터로 출력한다. 모터, MCU 등을 장착할 캐리어 바닥은 레이저 가공을 통해 나사 구멍을 낸다.

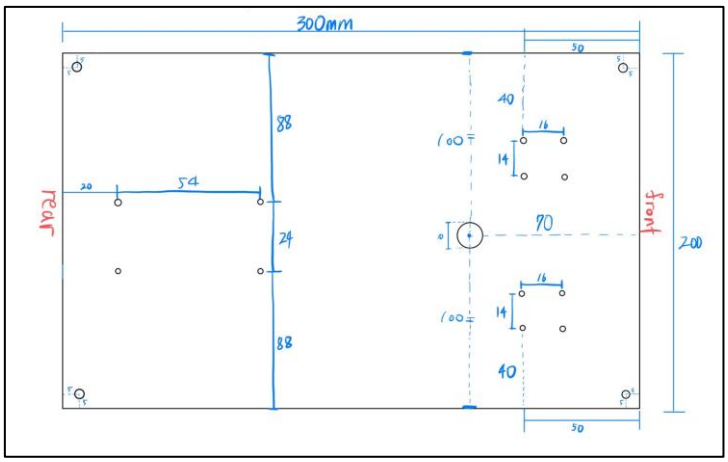


그림 1. 레이저 가공 주문 견적 도면

2.2.2 이미지 인식 및 모터 제어

Object Tracking이 가능한 HuskyLens Pro를 활용해 특정 이미지를 학습시키고, 학습된 값을 MCU로 전송해 모터를 제어한다. 이때 모터 제어는 단순한 Bang-Bang Control이 아닌 PID제어를 도입함으로써 더욱 정밀하게 대상을 쫓아가도록 한다. 또한 초음파 센서를 활용해 갑작스럽게 물체가 나타날 경우 순간적으로 급정거할 수 있도록 한다.

2.2.3 애플리케이션과 연동

기존에 HM-10 블루투스 모듈을 사용하여 iPhone과 연동을 시도하려 하였으나, iOS에서 제공하는 블루투스 앱과 본 프로젝트에서 사용하는 MCU 간의 호환성 문제로 인해 HC-06 블루투스 모듈과 Android 앱을 사용한다. 캐리어 앱은 MIT App Inventor를 활용해 제작한다.

3. Making & Coding

3.1 제작 과정

Task 1. 외형 제작

5개의 아크릴판과 3D 프린터를 활용해 캐리어의 몸체를 제작한다. 3D 모델링은 Fusion 360을 사용하였다.

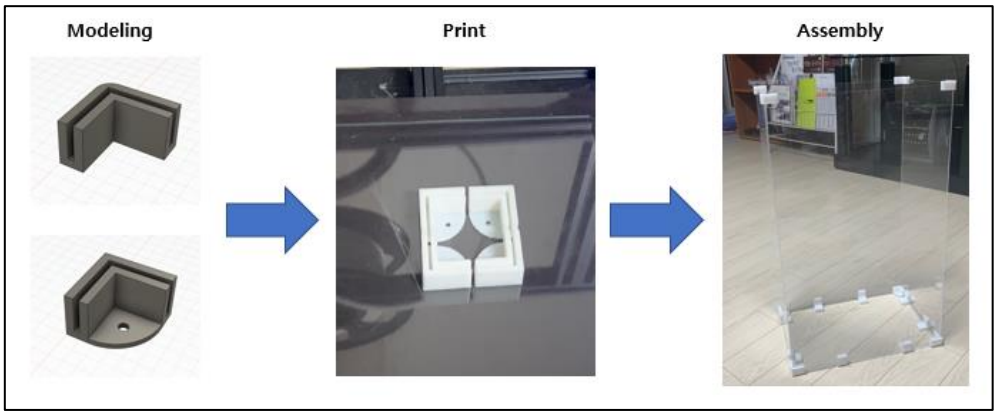


그림 2. 외형 제작 과정

위 몸체에 모터, MCU, 센서 등을 추가해 최종적인 외형을 완성하였다.

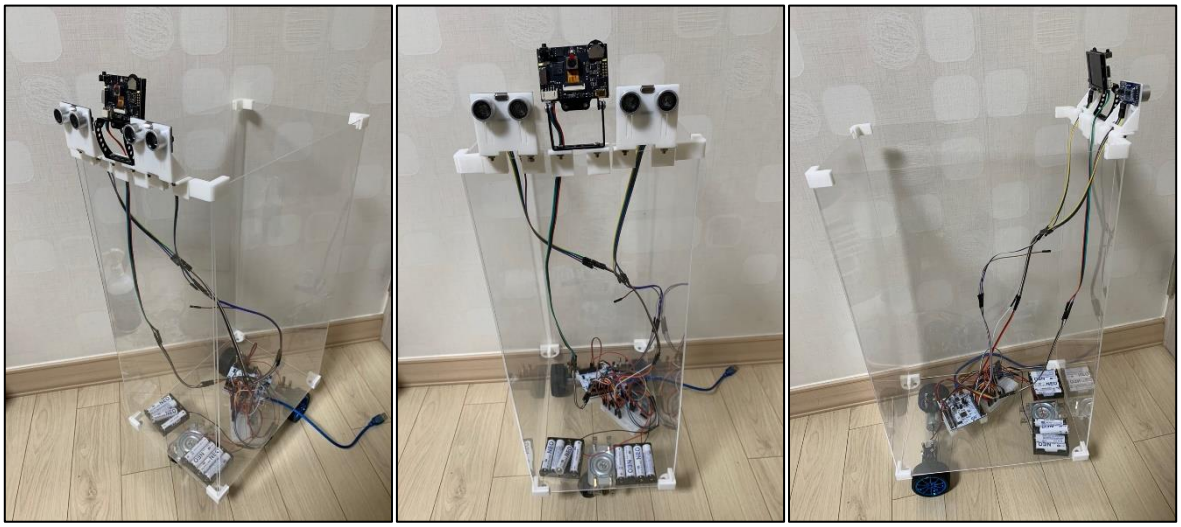


그림 3. 최종 외관 모습

Task 2. 이미지 인식

시중에 있는 스마트 캐리어가 사람을 따라가는 방식은 다양하다. 그 중에서 손목에 착용하는 Watch를 이용하거나, 특징적인 패턴을 몸에 지니어 캐리어가 그 패턴을 인식해 추적하도록 하는 것이 대중적이다. 본 프로젝트에서는 카메라를 활용해 Computer Vision을 접목하는 데 중점을 두어, 사용자의 특정 패턴을 학습하고 그것을 인식해 쫓아가도록 하는 기능을 구현하고자 한다. 사용한 카메라는 HuskyLens PRO 라는 AI Cam으로 Object Recognition, Object Tracking, Color Recognition등 다양한 CV 기능들을 지원한다. 우선 개발 단계이므로 캐리어가 잘 학습하고 따라올 수 있게 아래 그림과 같은 단순한 패턴을 학습시켰다.

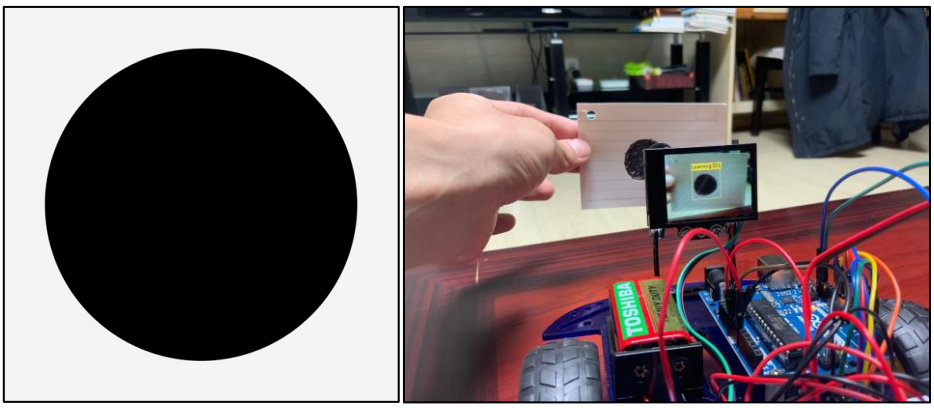


그림 3. 학습시킨 이미지(왼쪽)와 학습시키는 모습(오른쪽)

Task 3. 모터 제어 및 하드웨어 설계

PID 제어 알고리즘을 도입해 모터를 제어하였다. PID 제어란 비례(P), 적분(I), 미분(D) 제어를 결합한 것으로, 제어하고자 하는 대상의 output을 측정하고 이를 set point와 비교하여 error를 계산하고, 이 error를 이용하여 제어에 필요한 제어값을 계산하는 피드백 제어기의 구조로 되어 있다. 아래 식은 PID 제어의 제어값을 구하는 식이다.

$$\mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(t)\,dt + K_d \frac{de}{dt}$$

그림 4. PID 제어 식

각 항들의 역할은 다음과 같다.

비례항: 현재 상태에서의 오차값의 크기에 비례한 제어 작용

적분항: 정상상태 오차를 없애는 작용

미분항: 출력값의 급격한 변화에 제동을 걸어 오버슛을 줄이고 안정성 향상

각 항에 곱해져 있는 상수들은 수작업으로 조정해야 하는데, 가장 보편적인 Ziegler Nichols method를 활용해 gain을 튜닝하였다.

자세한 모터 제어 방식은 다음과 같다. HuskyLens가 이미지를 인식했을 때 형성되는 Bounding Box 중점의 x좌표와 set point 간의 차이(error)를 구해 그것을 PID 제어기에 입력함으로써 출력되는 ***output***을 모터 speed 제어하는 부분에 활용하였다. 좌측 모터와 우측 모터의 기본 속도를 120으로 설정하고(speed range: 0 ~ 255), 좌측 모터에는 120 – ***output***, 우측 모터에는 120 + ***output*** 만큼의 속도를 주었다.

Kinematics와 관련된 내용은 조향 방식과 전/후륜 구동 두 가지 부분으로 나눌 수 있다. 조향 방식은 일반적인 자동차에 사용되는 아커만(Ackerman) 방식을 사용하려 했다. 아커만 방식은 이동 방향과 바퀴가 이루는 각을 변화시켜 조향을 하는 방식이다. 하지만 기계적인 설계가 많이 들어가고 복잡한 제어가 우려되어 차동 제어 방식을 채택했다. 차동제어 방식은 두 바퀴의 속도 차이를 이용해 방향을 조절하는 방식으로 간단한 설계가 가능했다.

또한 전륜 구동과 후륜 구동의 방식을 비교해 보았을 때, 전륜 구동은 급커브가 잘 되지만, 급커브 이후 다시 직선 주행을 해야 할 때 바로잡지 못하고 계속 출렁이는 현상이 나타났다. 휴륜 구동은 급격한 방향전환이 잘 되지 않지만 직선 주행 시 안정적인 모습을 보였다. 캐리어의 특성상 짐을 싣고 가야 하는데 출렁임이 심하면 안정적이지 못하고 넘어질 위험성도 있다. 따라서 후륜 구동의 방식을 채택했다.

Task 4. 블루투스 연동

원래 HM-10 블루투스 모듈과 iPhone의 Dabble앱을 사용할 계획이었다.

그림 5. Dabble 앱 로고(왼쪽)와 Dabble 앱에서 지원하는 Gamepad(오른쪽)

하지만 해당 앱 라이브러리가 STM32 계열 보드를 지원하지 않아서 본인이 직접 Android 앱을 개발하였다. Android 앱 개발은 MIT App Inventor를 활용하였다.

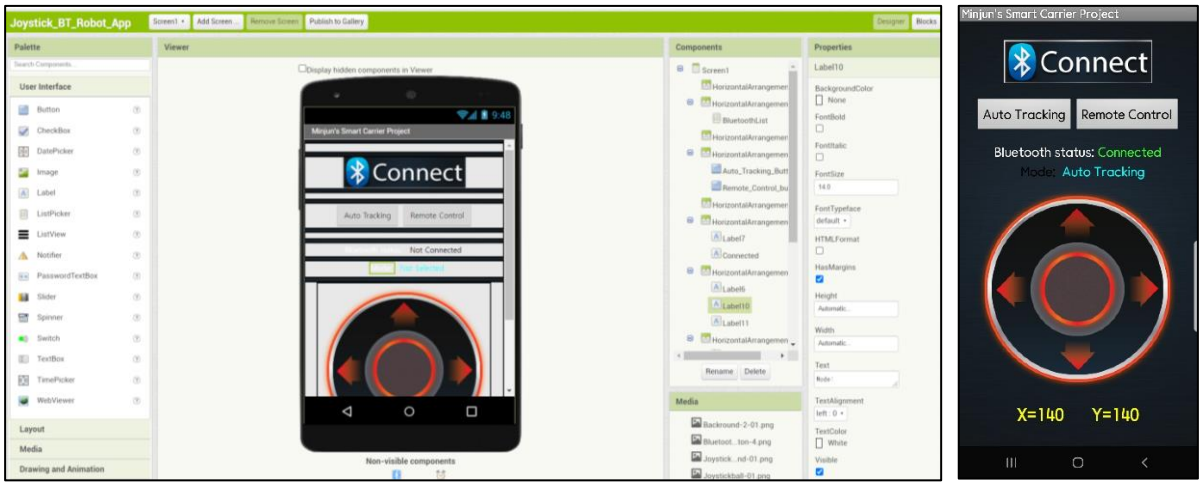


그림 6. 개발 화면(왼쪽)과 실제 앱 화면(오른쪽)

<앱 상세 설명>

- Bluetooth Connect: 휴대폰에 등록된 블루투스 기기의 목록을 보여주며 연결할 기기를 선택할 수 있다.
- Auto Tracking: 사용자를 인식해 자동으로 따라가는 모드 설정
- Remote Control: 휴대폰 화면의 조이스틱으로 캐리어를 조종하는 모드 설정
- Bluetooth status: 휴대폰과 블루투스 연결 상태 표시
- Mode: 현재 모드 표시
- Joystick: Remote Control 모드 시 캐리어 조종

3.2 Coding

Arduino

```
#include "HUSKYLENS.h"
#include <SoftwareSerial.h>

SoftwareSerial BTSerial(6, 7);

#define AP 3
#define A1 4
#define A2 5
#define BP 9
#define B1 10
#define B2 11
#define STBY 8

#define STOP 0;
#define LEFT 1;
#define RIGHT 2;

//huskylens variables
int x_center;
int width;
char mode;
int last_state;

//joystick variables
int xAxis=140, yAxis=140;
int motorSpeedA = 0;
int motorSpeedB = 0;
int break_state=0;

//PID constants
double kp = 0.6;
double ki = 0.005;
double kd = 10;

unsigned long currentTime, previousTime;
double elapsedTime;
double error;
double lastError;
double input, output;
double setPoint = 160;
double pid_i=0;
double rateError;

HUSKYLENS huskylens;
HUSKYLENSResult result;
void motor(int left, int right);
```

라이브러리 불러오기 및 기본적인 변수 선언

```
void setup() {
    Serial.begin(115200);
    BTSerial.begin(9600);
    Wire.begin();
    while (!huskylens.begin(Wire))
    {
        Serial.println(F("Begin failed!"));
        Serial.println(F("1.Please recheck the \"Protocol Type\" in HUSKYLENS (General Settings>>Protocol Type>>I2C)"));
        Serial.println(F("2.Please recheck the connection."));
        delay(100);
    }
    pinMode(A1, OUTPUT);
    pinMode(A2, OUTPUT);
    pinMode(AP, OUTPUT);
    pinMode(B1, OUTPUT);
    pinMode(B2, OUTPUT);
    pinMode(BP, OUTPUT);
    pinMode(STBY, OUTPUT);
}

void loop() {
    digitalWrite(STBY, HIGH);

    if(BTSerial.available()){
      mode = BTSerial.read();
    }

    while(mode == 33){ //!
      if(BTSerial.available()){
        mode = BTSerial.read();
      }
      autoControl();
    }

    motor(0, 0);

    if(mode == 114){
      xAxis=140;
      yAxis=140;
      while(mode == 114){
        joystickControl();
        if(break_state == 1){
          break_state = 0;
          break;
        }
      }
    }
}
```

Setup: 시리얼 통신, 블루투스 통신 시작, 허스키 렌즈 연결 에러 검사, pinMode 설정

Loop: STBY pin HIGH 설정 (TB6612FNG 모터드라이버에서 STBY HIGH 설정 시 모터 구동 가능), 블루투스로 원하는 모드 선택(Auto Tracking(ascii code: 33(“!”))/Remote Control(ascii code: 114(“r”))),

autoControl 함수: 사용자 자동으로 따라가는 기능 구현

joystickControl 함수: joystick 으로 조종 시 그에 따라 움직이는 기능 구현

break_state: joystick 작동 도중 Auto Tracking Mode 로 전환 시 바로 break 를 하기 위해 정의한 변수(Auto Tracking Mode 가 호출되면 joystickControl 함수 내의 break_state 가 1 이 됨)

+ Auto Tracking 모드가 33 인 이유는 조이스틱에서 받아들이는 x, y 값이 60 ~ 130 이므로, 만약 Remote Control Mode 에서 Auto Tracking 모드로 전환 시 조이스틱에서 받아들이는 값의 범위 밖에 있는 값으로 받아들이기 위함. 그렇지 않으면 조이스틱 조종 도중 Auto Tracking 모드로 바뀔 수 있음.

```
void autoControl(){
  if (!huskylens.request()){
          digitalWrite(STBY, LOW);
          previousTime = millis();
          motor(0, 0);
        }
      else if(!huskylens.isLearned()){
        digitalWrite(STBY, LOW);
        previousTime = millis();
        motor(0, 0);
      }
      else if(!huskylens.available()){
        if(BTSerial.available()){
              mode = BTSerial.read();
        }
        previousTime = millis();
        if(last_state == 0){
          motor(0, 0);
        }
        else{
          motor(constrain(120 - output, 70, 255), constrain(120 + output, 70, 255));
        }
      }
      else{
        while (huskylens.available())
        {
            if(BTSerial.available()){
              mode = BTSerial.read();
              Serial.println(mode);
              if(mode == 113){
                break;
              }
            }
            result = huskylens.read();
            input = (double)result.xCenter;

            if(input < 120){
              last_state = LEFT;
            }
            else if(input > 200){
              last_state = RIGHT;
            }
            else{
              last_state = STOP;
            }

            if(result.width > 100){
              previousTime = millis();
              motor(0, 0);
              delay(500);
              last_state = STOP;
            }
            else{
              output = computePID(input);
              digitalWrite(STBY, HIGH);
              motor(constrain(120 - output, 70, 255), constrain(120 + output, 70, 255));
            }
          }
        }
}
```

autoControl 함수를 정의한 부분

HuskyLens 가 연결되지 않았거나 학습한 내용이 없을 때는 모터 작동을 안하고, HuskyLens 가 학습은 되었는데 물체가 감지되지 않으면 이전 결과를 바탕으로 움직임(last_state 가 0 일 때는 STOP), 만약 HuskyLens 가 물체를 인식하면 인식한 물체를 따라 주행하는데, 이때 물체의 Bounding Box 가로 폭이 100 보다 크면 정지함.

```
void joystickControl(){
  while (BTSerial.available() >= 2) {
    xAxis = BTSerial.read();
    delay(10);
    yAxis = BTSerial.read();
  }
  delay(10);
  if(xAxis < 40 or yAxis < 40){
    break_state = 1;
    mode = 33;
  }
  if (xAxis > 130 && xAxis <150 && yAxis > 130 && yAxis <150){Stop();}

  if (yAxis > 130 && yAxis <150){
    if (xAxis < 130){turnRight();
      motorSpeedA = map(xAxis, 130, 60, 0, 255);
      motorSpeedB = map(xAxis, 130, 60, 0, 255);
    }
    if (xAxis > 150) {turnLeft();
      motorSpeedA = map(xAxis, 150, 220, 0, 255);
      motorSpeedB = map(xAxis, 150, 220, 0, 255);
    }
  }
  else{
    if (xAxis > 130 && xAxis <150){
      if (yAxis < 130){forward();}
      if (yAxis > 150){backward();}
      if (yAxis < 130){
        motorSpeedA = map(yAxis, 130, 60, 0, 255);
        motorSpeedB = map(yAxis, 130, 60, 0, 255);
      }
      if (yAxis > 150){
        motorSpeedA = map(yAxis, 150, 220, 0, 255);
        motorSpeedB = map(yAxis, 150, 220, 0, 255);
      }
    }
    else{
      if (yAxis < 130){forward();}
      if (yAxis > 150){backward();}
      if (xAxis < 130){
        motorSpeedA = map(xAxis, 130, 60, 255, 50);
        motorSpeedB = 255;
      }
      if (xAxis > 150){
        motorSpeedA = 255;
        motorSpeedB = map(xAxis, 150, 220, 255, 50);
      }
    }
  }
  analogWrite(AP, motorSpeedA);
  analogWrite(BP, motorSpeedB);
}
```

joystickControl 함수를 정의한 부분

차례대로 joystick 의 x 좌표와 y 좌표를 받아들이고, 각 좌표의 범위에 따라 좌, 우 모터의 속도를 조절

위 코드 내에 있는 forward(), backward(), turnLeft(), turnRight(), Stop() 함수는 간단한 전진, 후진, 좌회전, 우회전에 대한 함수이므로 코드 첨부는 생략.

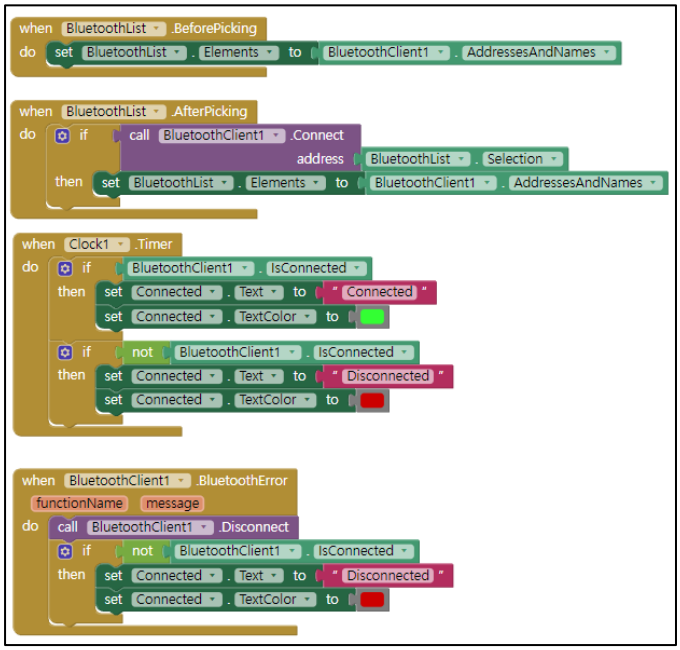
```
double computePID(double inp){
  currentTime = millis();
  elapsedTime = (double)(currentTime - previousTime);

  error = setPoint - inp;
  if(-10 < error and error < 10){
    pid_i += ki*error;
  }
  rateError = (error - lastError)/elapsedTime;
  double out = kp*error + pid_i + kd*rateError;

  lastError = error;
  previousTime = currentTime;

  return out;
}
```

PID 제어값을 연산하는 함수

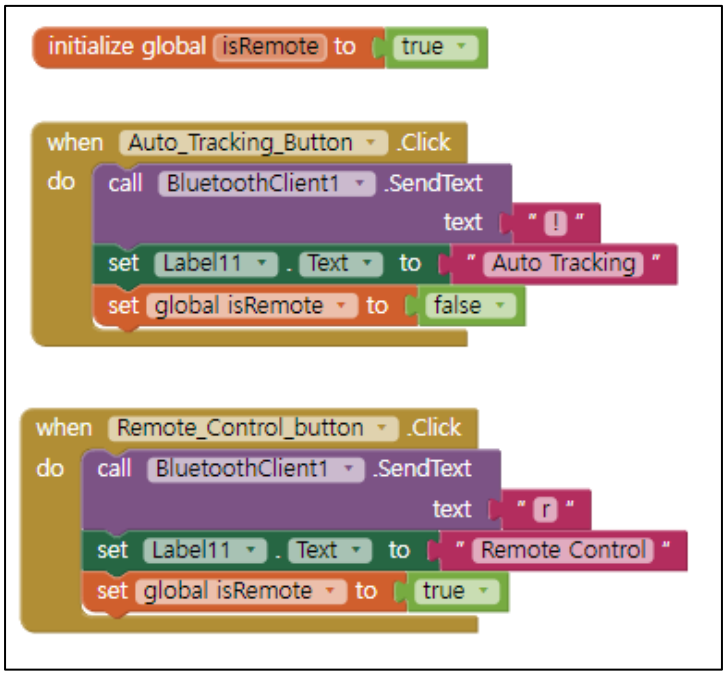
elapsedTime 을 통해 Δt를 구하고, setPoint(160)과 입력받은 값 간의 차이(error)를 구함. 이후 특정 범위 내에 있을 때 적분(I) 제어 값을 누적하고, rateError 를 구해 이후 미분(D) 제어에 활용함.

Out 에 최종적인 PID output 을 연산하고, 이후 lastError 와 previousTime 을 다시 설정한다.

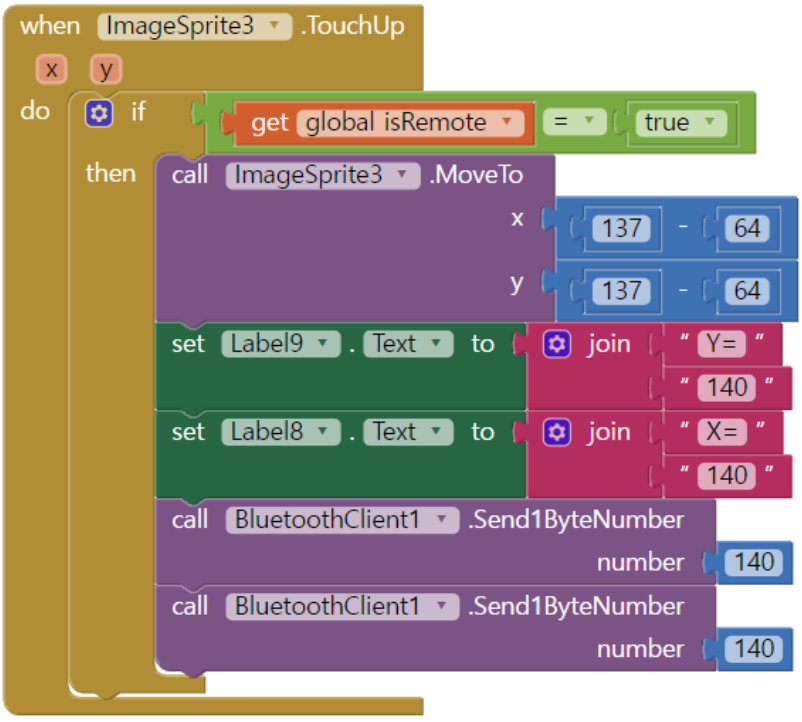
MIT App Inventor



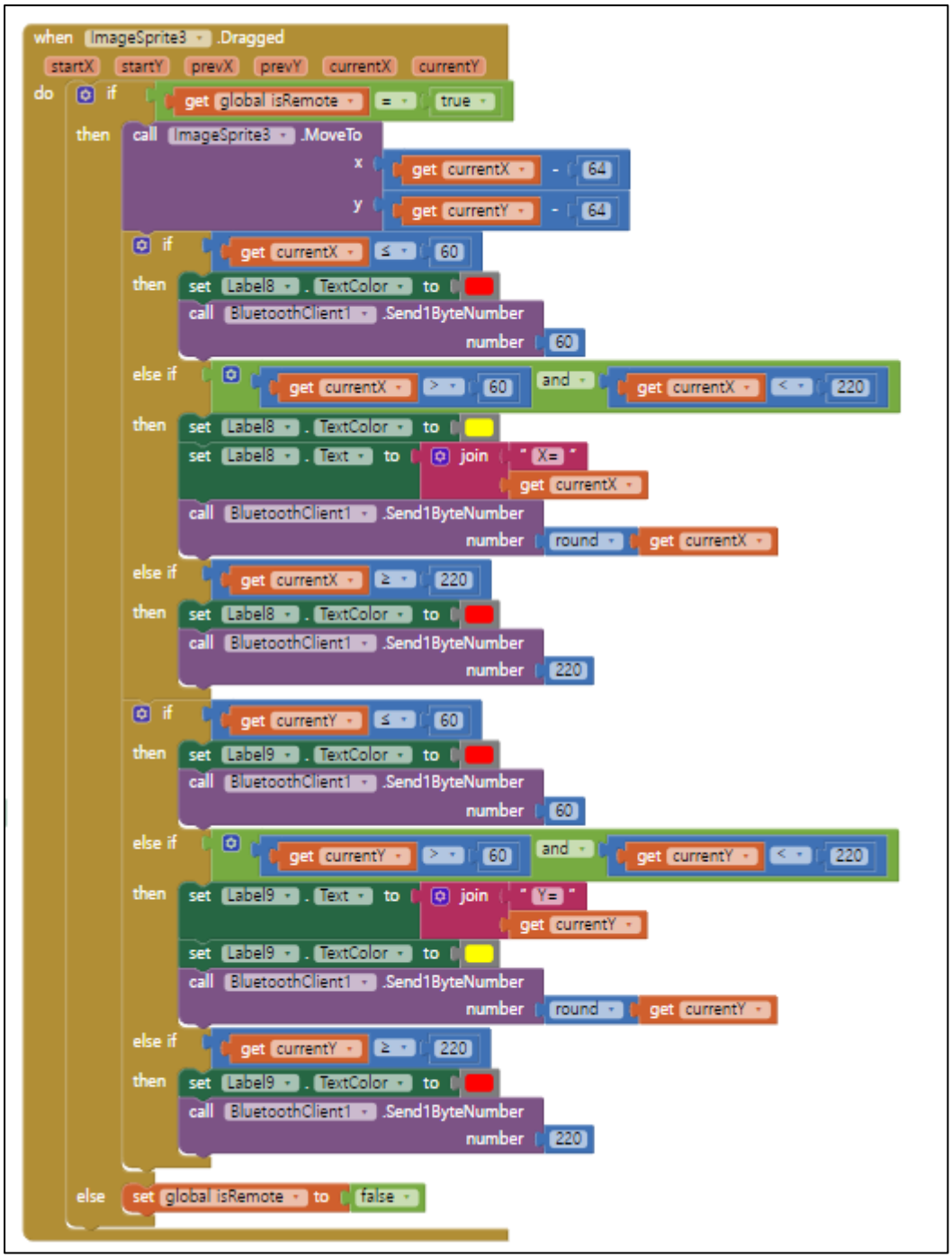
블루투스 연결 상태 확인과 관련된 기능 설정



모드 선택 및 Remote Control Mode 선택 여부에 대한 변수 isRemote 선언



조이스틱에서 손을 떼었을 때 원위치 시키도록 하는 부분



Joystick 을 움직였을 때 좌표를 실시간으로 받아서 블루투스로 전송하는 부분

4. Conclusion

IoT 와 Computer Vision 기술을 접목한 스마트 캐리어를 제작해 보았다. 본 프로젝트에서 제작한 캐리어를 실생활에서 사용하기엔 아직 부족함이 있지만, 제약된 조건(균일한 지형, 일정한 빛) 하에서는 괜찮게 작동하였다. 캐리어가 사람을 따라가도록 하기 위해 기존에 캐리어가 학습했던 검은 점 이미지를 사용자의 등에 붙이고 그것을 인식해 따라가도록 하였다. 하지만 실생활에서 스마트 캐리어를 사용하겠다고 민망함을 무릅쓰고 등에 커다란 점을 붙이고 다닐 사람은 얼마되지 않을 것이다. 따라서 캐리어의 주행을 보조할 만한 다른 장치들을 함께 사용해야 한다. 또한, 카메라가 한 가지 특징적인 point 만 인식하는 것이 아니라 사용자의 여러 feature 를 학습하고 인식해 더욱 정확한 제어를 가능하게 해야 실생활에서 사용이 가능할 것이다. 또한 장애물 감지 시 단순 정지가 아니라 그 상황을 빠져나올 수 있는 알고리즘과 센싱 기술이 필요하다. 현재 만든 캐리어는 prototype 으로 아직 짐을 싣고 실제로 가지고 다닐 정도는 아니다. 만약 실제로 사용된다면 짐을 싣게 될 것인데, 이 때는 하중에 따른 제어방식이 또 달라져야 할 것이다. 이번 프로젝트에 사용한 HuskyLens PRO 는 Object Tracking 기능을 지원했는데, 아쉽게도 한 가지 Object 만 학습시킬 수 있었다. 따라서 향후에는 더욱 좋은 카메라와 CV 알고리즘을 사용해, 주인을 인식하는 것뿐만 아니라 제스쳐 인식으로 명령을 수행하는 기능까지 구현하면 더욱 유용하고 혁신적인 캐리어가 될 것이다.