open CV Final Report.

20130226 류현준

**Week 1**

1. **Launching Ubuntu**

첫 과제, 우분투(Ubuntu ; Open source Operating System based on Linux)를 설치함.

우분투의 기본이 되는 터미널 사용법을 공부하고, 라이브러리 설치 및 시스템 업데이트를

터미널을 이용하여 실행하는 방법을 학습.

1. **Install ROS, QT Creator**

Vision Processing에 줄곧 사용할 ROS와 QT Creator(Cross platform for C++, Java)를 설치함.

Visual Processing에 필요한 라이브러리 및 패키지를 다운로드함. Face detection 패키지를

이용하여 face detection을 해봄.

1. **Camera calibration**

ROS 를 이용하여 분배된 웹캠을 인식하고, checkerboard plate를 이용하여 주어진 웹캠의

카메라 내부표정을 측정하였음. 분배된 카메라의 내부표정은 Figure. 1과 같음.

카메라 내부표정을 이요해 화상에 나오는 이미지를 처리하는 방법을 습득함.

카메라 내부표정과을 이용한 World coordinate 상 좌표, Pixel coordinate 상 좌표의

관계는 Figure 2와 같음.

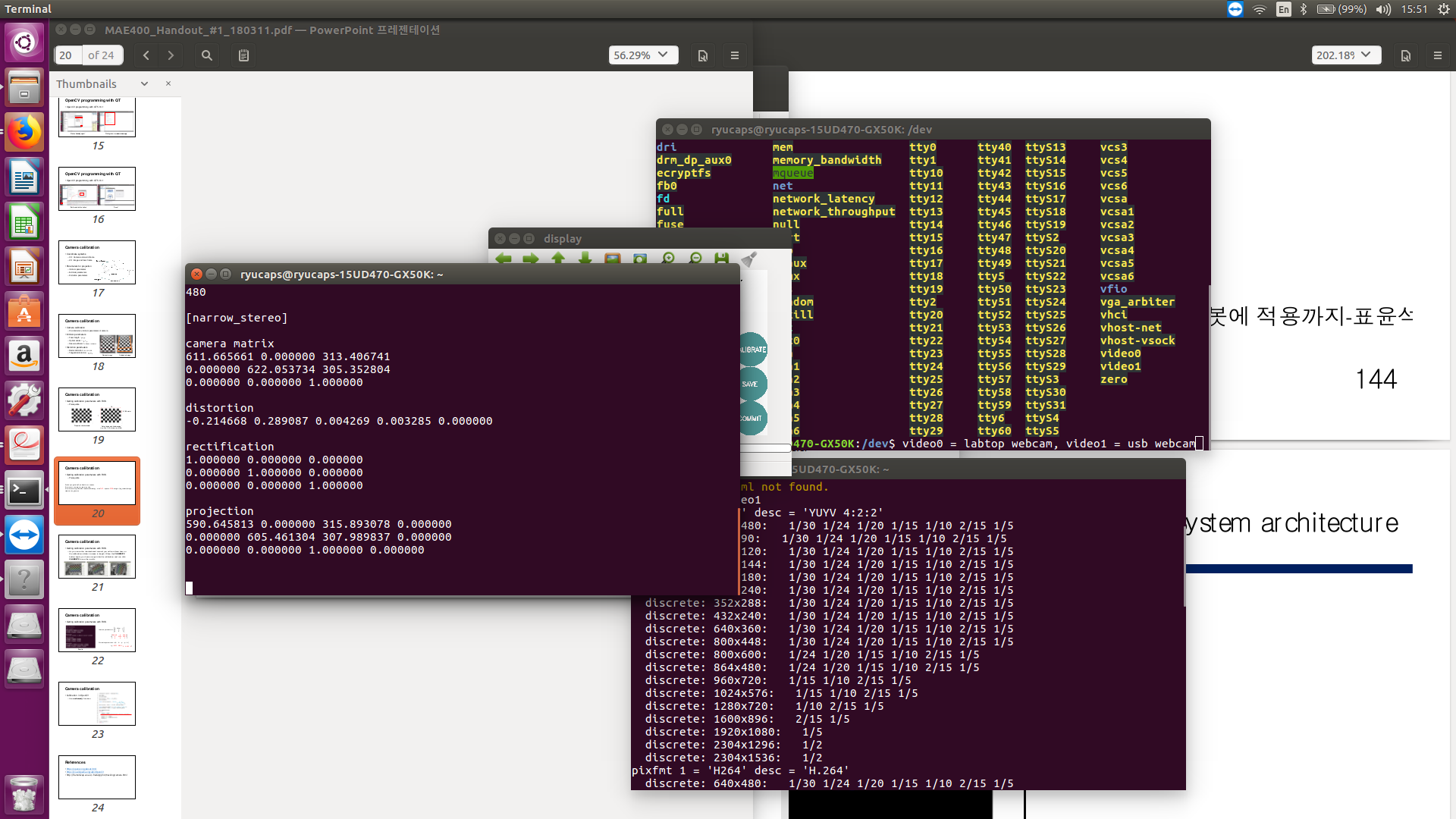


Figure 1

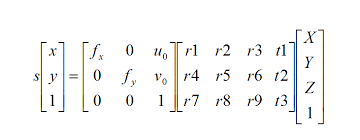
[](https://www.google.co.kr/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwiBtMn_35baAhWF2LwKHcE8D_YQjRx6BAgAEAU&url=https://www.packtpub.com/books/content/camera-calibration&psig=AOvVaw3Y5vKB6demfGej2ZakYSE_&ust=1522592149434269)

Figure 2

**Week 2**

1. **Preparing for Presentation**

3/30일의 발표를 준비함. 본인이 ppt작업을 담당하고 전체적인 형식의 틀을 다짐.

1. **Plan for detecting balls**

Ball detecting mechanism의 기초를 다짐.

1. 카메라를 통해 들어오는 2D 이미지(Figure 3)의
2. 기계동 바닥에 대한 Homography transform matrix를 구함. (Machine의 카메라 위치에

대한 정보 필요)

1. 기계동 로비 바닥에 대한 Homography transform을 통해 2D mapping을 함.
2. 만들어진 2D 이미지에 사전조사한 기계동 바닥의 dimension을 활용하여 거리 정보를

설정함.

1. 설정된 정보를 이용하여 차체가 움직일 경로를 계산함.

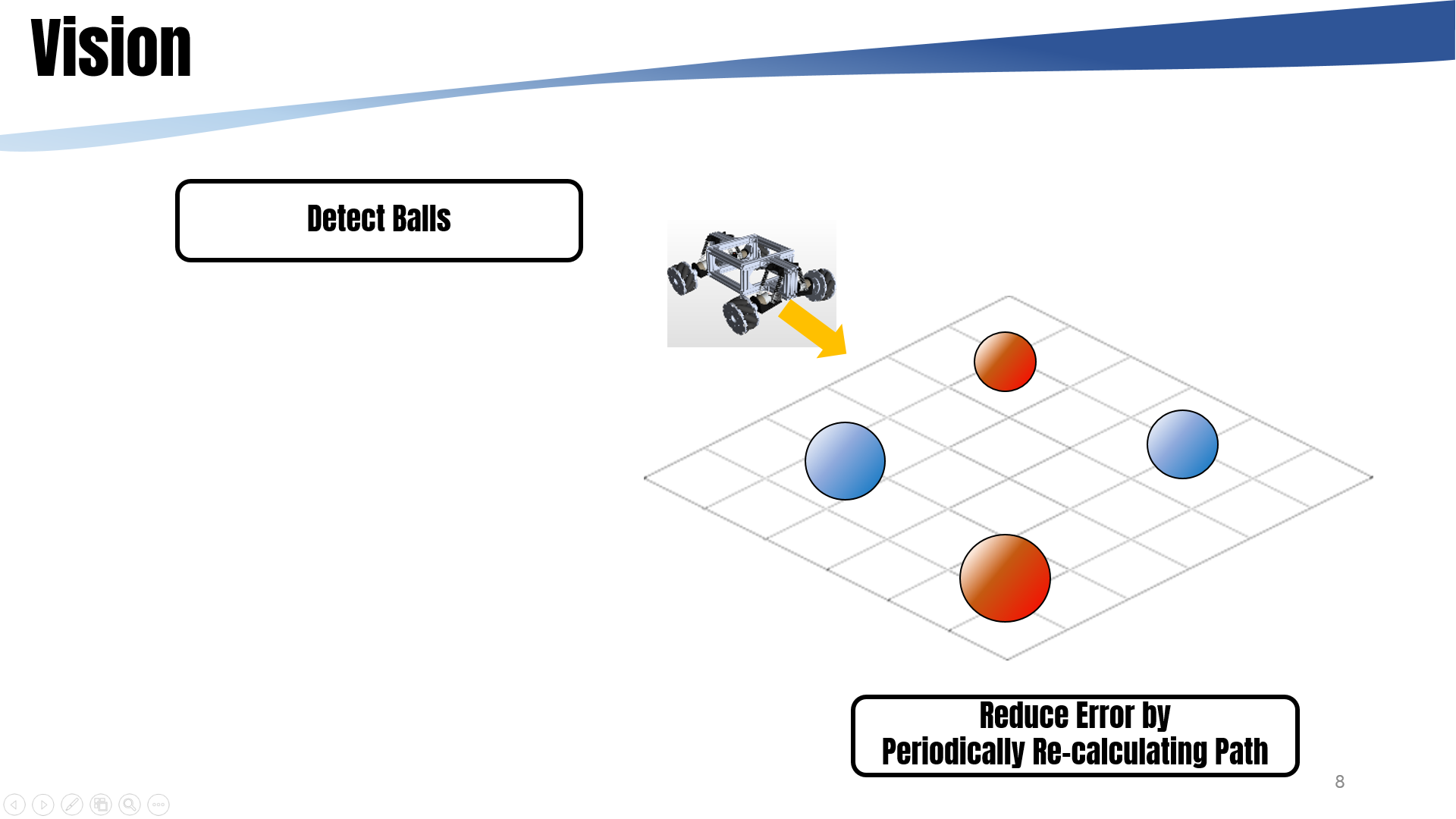


Figure 3

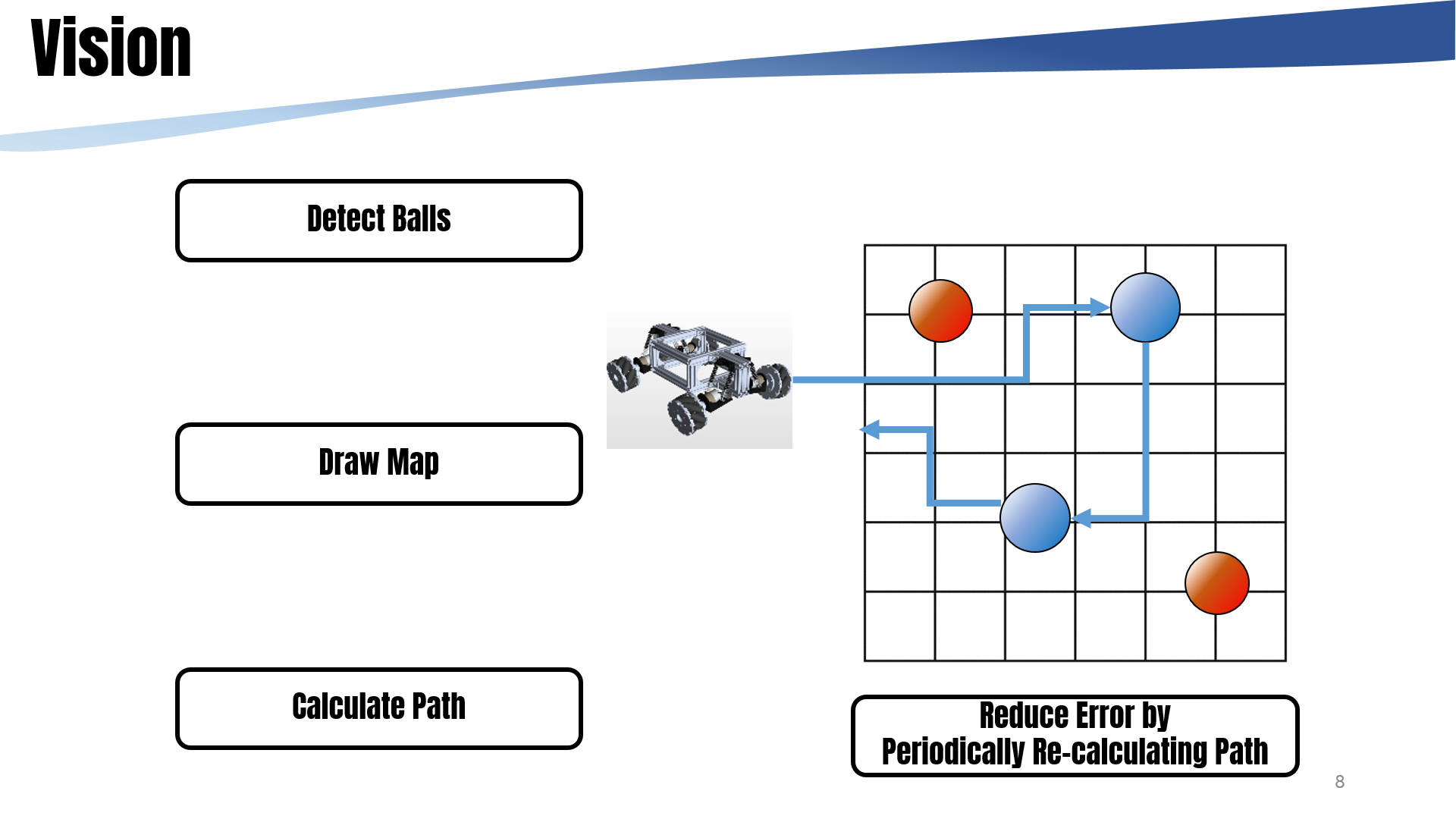


Figure 4

1. **Basics of Image Processing**

이미지 프로세싱의 기초가 되는 테크닉들에 대해 공부함.

블러처리, 라플라시안 변환, 오브젝트 추적, 허프 라인/서클 트랜스포메이션, 서클 디텍션,

엣지 디텍션 등의 기본 테크닉을 실제로 해보고 소스코드를 공부함.

본인이 설정한 Ball detection mechanism을 실행하기 위해선 Hough circle transformation,

Edge detection을 주로 이용할 것으로 예상됨.

Progress report on Open CV 4/1~4/15

**Week 3.**

1. Studied distributed algorithm for ball detecting

Participate on Open CV TA session and got ball detecting algorithm basic code.

The code had these listed problems.

1. Detects the non-circular object too.

* The code algorithm starts with detecting the designated color first. So it detects anything that’s blue or red and draw circular hall on the object
* Possibility of error

1. Cannot detect Blue Ball

* Our main goal is to collect blue balls. But the distributed code cannot detect blue ball correctly.
* Possibility of non-robustness

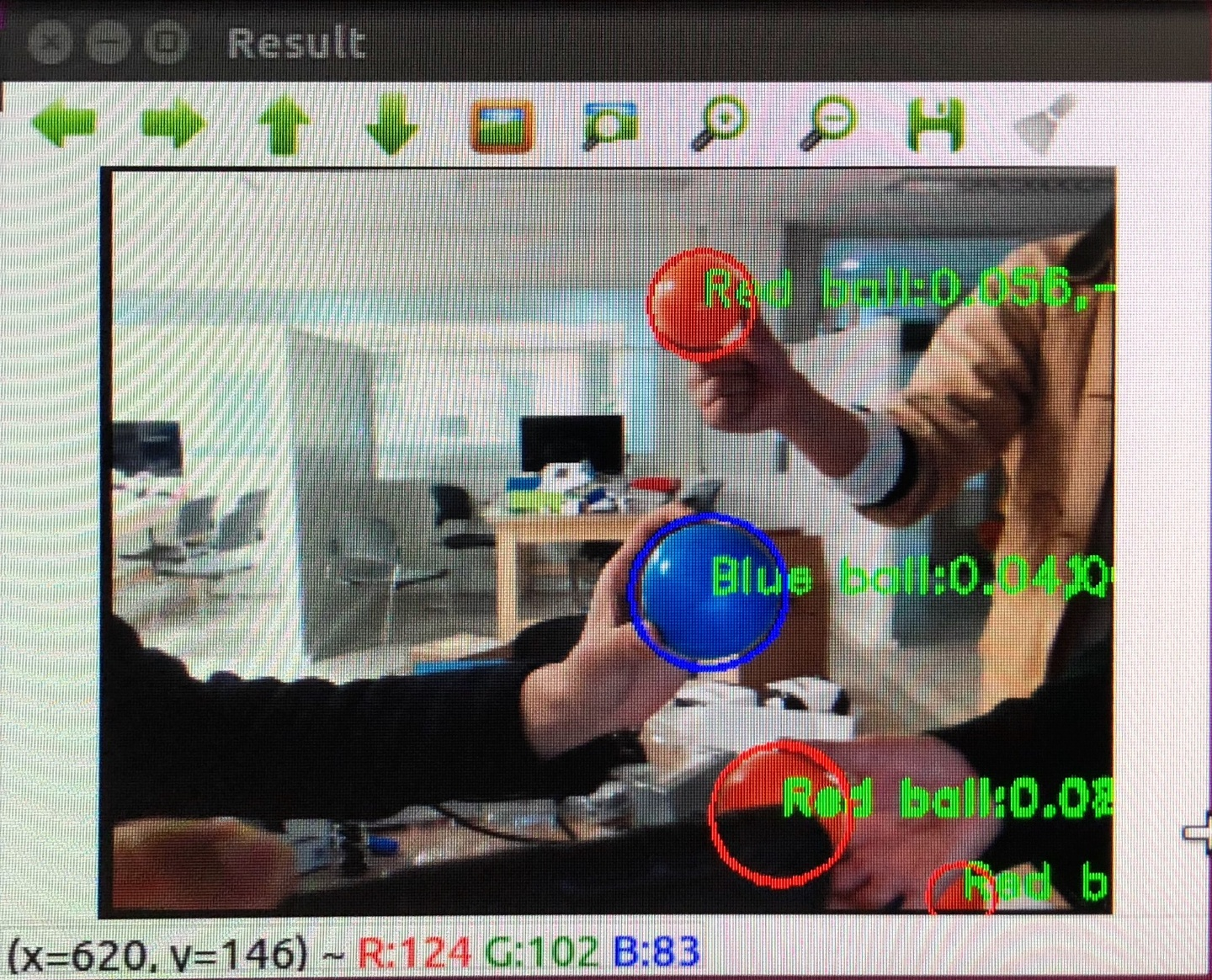
1. Should calculate precise internal parameters

* Our main code has internal parameters of TA’s camera
* Need to calculate precise camera (should get checkerboard plate)

1. Integrate nodes with ROS part.

Had meeting with ROS team and integrated nodes. The result can be seen from the picture.

While integrating the node, talked about problems in distributed code.



**Week 4**.

1. Found ways to solve the problems in distributed code.
2. Blue-cell problem -1

In our real world, if we get image with RGB frame pixels, we get dominant pixel

brightness values in Blue color. That’s the reason why we cannot detect

blue color robust.

If we are to solve the problem, we should get Red and Blue coordinate dot graph.

And get standard deviation in both red and blue color.

After get the STD, get K(constant) and multiply it to every pixels’ blue pixel value and

name it as frame 2

we’ll do the ball detecting process with this frame2.

1. Blue-cell problem2

If we change the ball detecting mechanism to detect circular shape first, before

detecting color, we may solve the problem

if we adopt this method, we can solve the first problem, incapability of detecting

non-circular shape too

conclusion.

We’ll try to adopt both solutions for solving blue-cell problem in next 2 weeks.

And make prototype that can detect balls and calculate possible paths.

Progress report on Open CV 4/16~4/29

**Week 5.**

1. Tried to solve blue-color problem.

By modifying distributed code, tried blue color modification.

Algorithm goes like this

1. Get the frame
2. Get Blue and Red value of each pixels in the frame.
3. Draw 2-D graph which shows Red and Blue value of pixels in dot-graph
4. Get standard deviation of blue value and red value in the graph
5. Divide into and get normalizing factor n.
6. Meanb = mean({blue pixel value}

For each pixel

Realb = extracted blue value

Modb = (Realb – Meanb)\*n

Newb = modb + meanb

Return Newb

End

1. Run rest of the code with modified pixel value

* **But it failed**

1. Reason for failure?

Have tried to detect balls under the light condition similar with Mechanical engineering

Building. (almost yellow)

* Couldn’t detect red ball, instead, blue ball was detected.

Can be concluded that, blue color cannot be serious problem. The real problem was

light source.

* Need for modification of frame based on light source.

**Week 6**.

1. Modification based on light source.

Modification algorithm goes like this.

1. Take picture of white plate under same light condition of final presentation space.
2. Get pixel value of white plate.
3. Set the pixel value as normalizing factor (normB, normG, normR)
4. For each pixels we get from camera, do normalizing with norm factors.

newk = realk\*255/normk

1. Save it as new image.

Haven’t implemented the code perfectly yet. I’ll discuss the code with professor next

Monday.

1. Saturation problem

When our mobile system get closer to object ball, the specular reflection region is

detected as another small ball. This saturation can be serious problem since it can

make error at output to Labview part. So after perfectly implementing code for

modifying color mentioned above, I will try to solve this saturation problem.

1. Calculation time problem

Even if all the problems mentioned above can be solved, it can delay calculation time

In serious manner. So carefully considered implementation is needed.

**Week 7**

1. **Modifying distributed code.**

In order to improve our ball detecting condition. Tried to change the ball detecting code.

The code’s algorithm goes like follow.

* Detect Circles by Hough circle transform
* Circles’ data (center and radius) are saved in a matrix called circles
* Get the data from the matrix and draw circle on black image which is

copied from the original frame.

* Use the image as another set of input which eliminates errors by detecting unnecessary objects in the background.
* Detect artificially drawn blue and red balls.

Our cv team thought that by implementing this algorithm, we can reduce the error and

Detect balls without losing our data from inrange function operation.

1. **Done demonstration of whole detecting process.**

To finish our job, it is necessary to test the detecting code. So our cv team and ROS team

tried detecting and tracking process.

Due to instability in detecting the ball ( for z axis), wheel alignment problem, and motor

Control problem, tracking the ball was not so easy for our whole test.

For open CV part, we should improve our detecting by implementing some other processes.

**Week8**

1. **Done modifying whole code**

Have done modifying the code, tried detecting the ball. Several problems exist.

1. Instability in Hough circle transform itself makes our detecting code somewhat more

Instable. Which means there exists constraints.

* Background should be plane.
* Some region in the frame (mostly of boundary) is ball – undetectable region
* Still detects some none-existing balls.

1. To solve problems above, more robust code is needed.

* Check the circle’s color more robustly
* Test the code on demoplace (Mechanical engineering building 1st floor

lecture room

**Week 9**

1. Modifying of code
2. Tried to deal with detection problem.

* Modified detection algorithm from hough circle to circle

(didn’t work that well)

* Found out that Houghcircle transform is undesireable for this task.

1. Tried to detect ball with aid of optical flow

* By using optical flow, get the location of previously-detected ball location and

detect ball around the saved data.

* Didn’t work well either because of calculation time (clock skew error occurs)

1. Decided to concentrate on basics.

* Whenever open CV team tried to implement another methods to improve

Ball-Detection, it didn’t work well because on instability, long calculation time

and other unexpected problems. So, in the upcoming week, we will try to

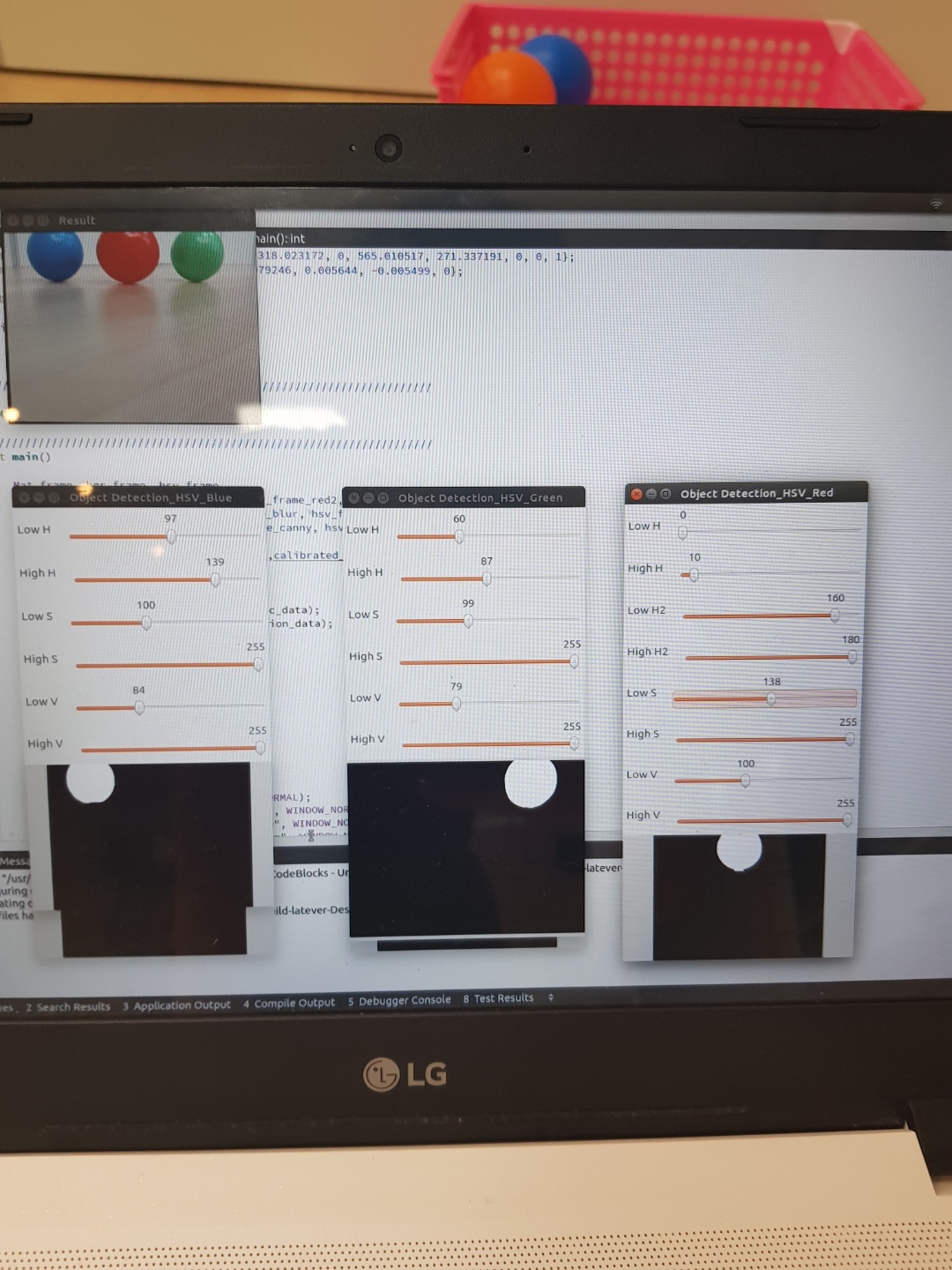
improve the ball detection code just by modifying a bit of the distributed code.

Not implementing something new.

**Week 10**

1. Demo on the ME building 1510
2. Done matching white balance in certain condition

* 기계동 1510호의 조명 상황에 맞춰서 각 ball의 HSV value range를 수정함.
* Ball 의 contour가 잡히지 않던 문제는 Saturation 및 Value 때문인 것으로 판단하고, 값을 수정하기 위해 값을 계산하여 맞춤
* 실제 Ball 의 사이즈보다 contour가 크게 잡히는 문제를 수정함.



1. ROS 및 Solid Works파트와 협업
2. 카메라 높이 설정

* 카메라의 높이 설정을 완료함. Partition 너머가 카메라의 시야에 보이지 않도록 설정하는 높이가 약 30~40cm 사이인 것으로 판명

Final Report (6/17)

주어진 환경에서 정해진 색의 공을 찾아서 정해진 장소로 가져오는 것이 우리 한 학기 과제의 목표였다. Open CV 수업 시간에 뼈대가 되는 코드를 분배 받은 뒤, 해당 코드의 문제점을 1학기동안 수정하는 방식으로 수업이 진행되었다.

해당 코드의 문제점은 대표적으로 세가지로 꼽힐 수 있다.

1. 코드의 아웃풋이 2번 반복된다는 점

* 해당 문제점은 구동계에 인풋을 2번 주게 만든다. 이는 차체를 컨트롤 함에 있어서 번거로운 필터링 과정을 거쳐야 하게 만든다.

1. 공의 사이즈가 실제보다 크게 잡힌다는 점

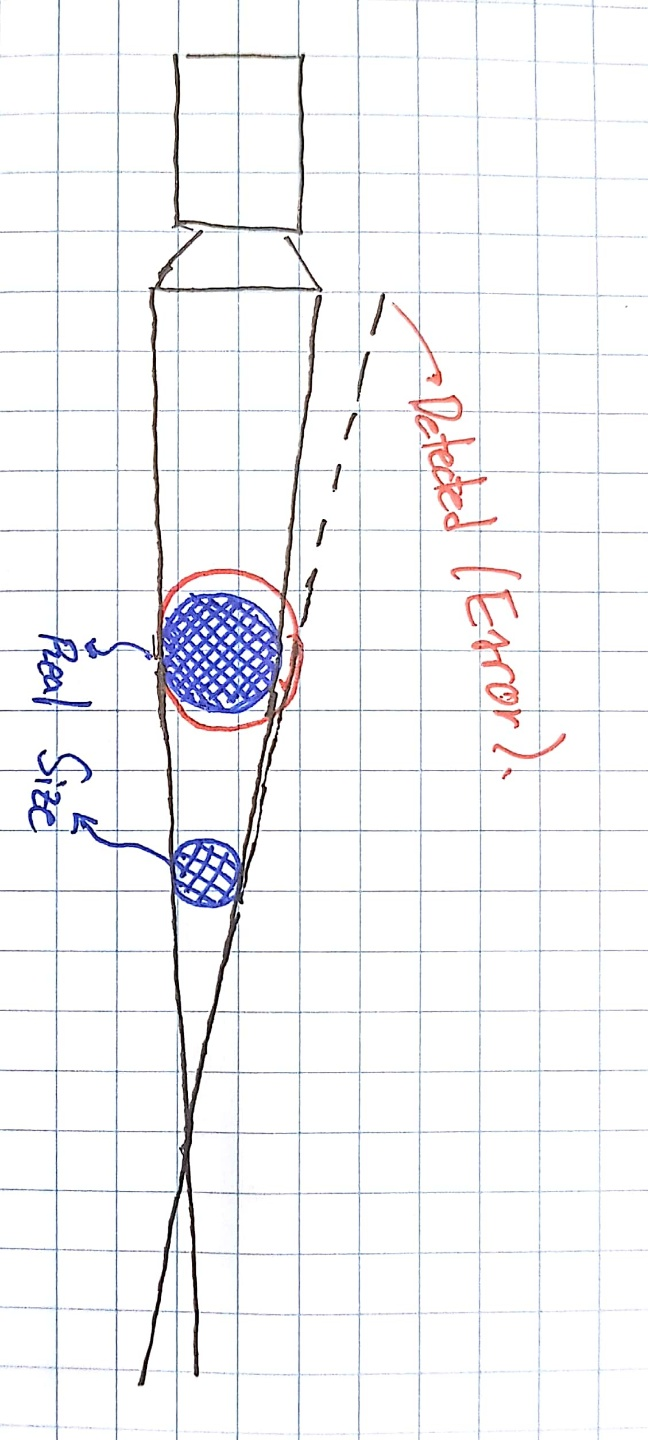
* 해당 문제점은, 탐지된 공의 사이즈를 기준으로 공까지의 거리를 계산하는 본 코드의 알고리즘을 생각 해 보았을 때, 공까지의 거리를 실제 거리보다 가깝게 계산하도록 만들 수 있다.

1. 파란색 공을 탐지하지 못한다는 점

* 파란 공을 찾아서 잡는 것이 어렵기 때문에 근본적으로 본 과제를 수행하는데 문제가 될 것이다.

각각의 문제점을 해결하기 위해서 각기 다른 접근법을 택하기로 하였다. 각 문제점 별 적용한 해결 방법을 간단한 그림과 함께 설명하도록 하겠다.

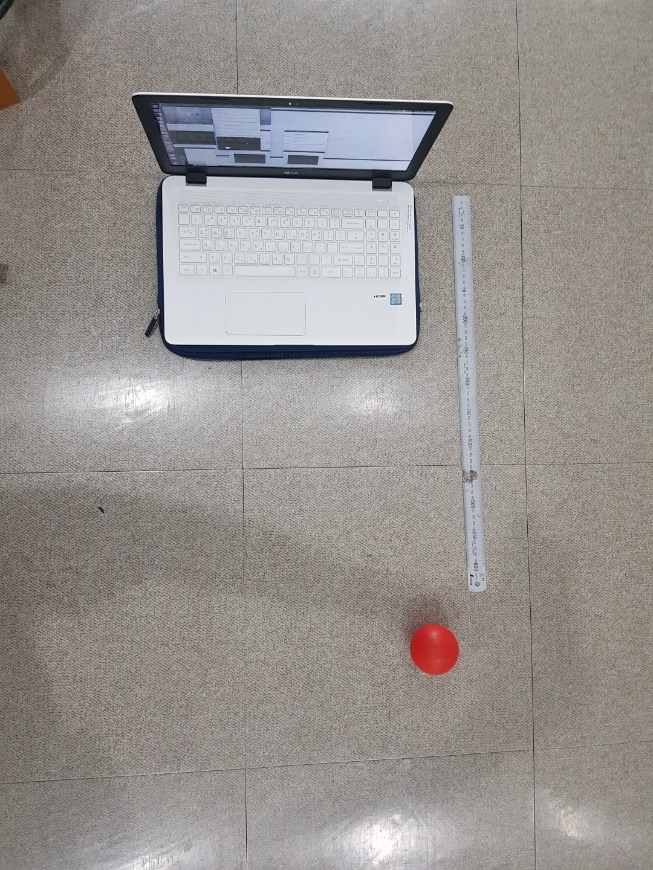
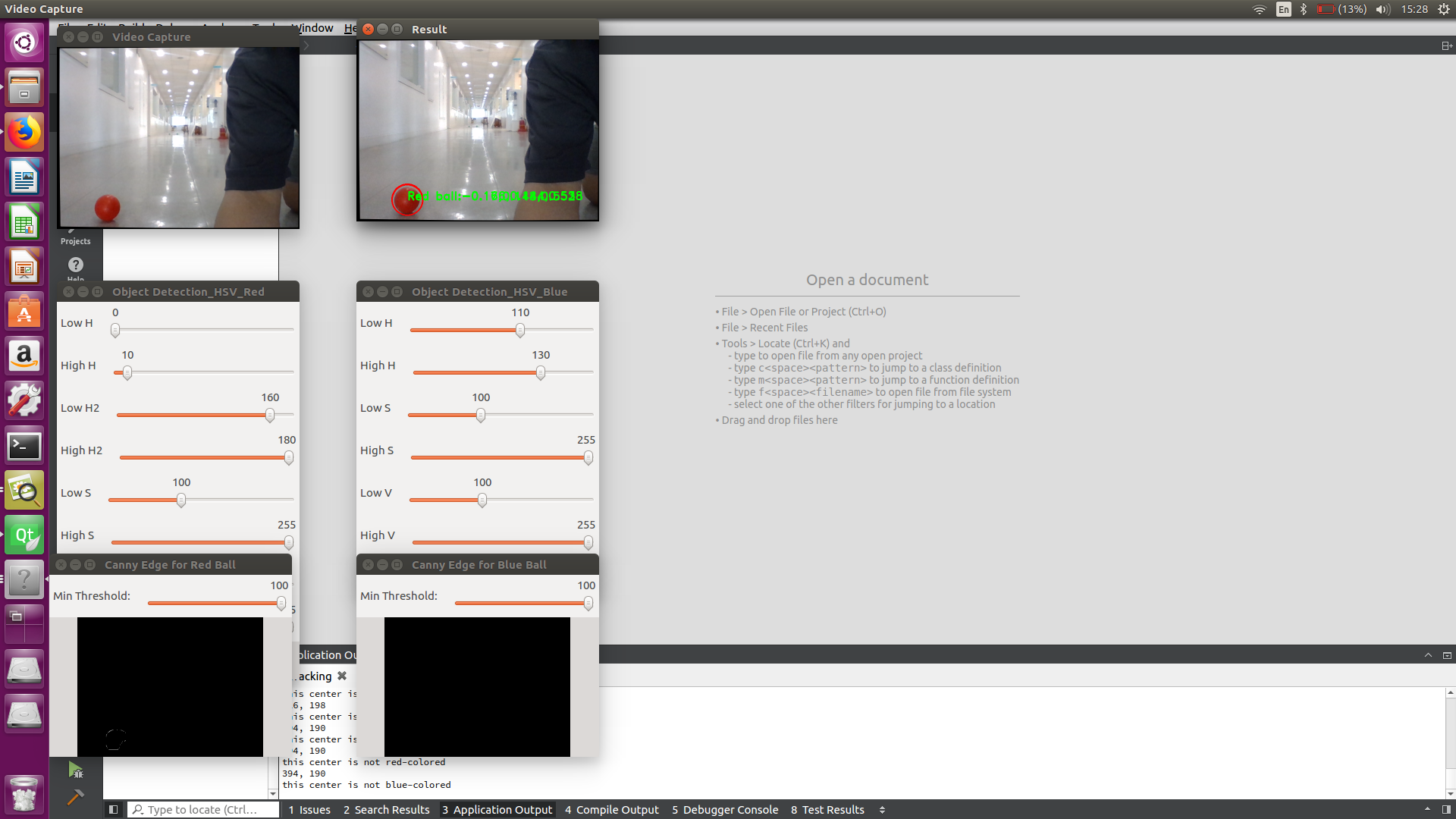
2번의 아웃풋이 똑같은 데이터를 출력하는데, 그 이유는 데이터를 저장하고 출력하는 과정에서 서로 다른 하드웨어에 2번 저장하도록 프로그램이 비 효율적으로 짜여 있었기 때문이다. 해당 코드를 수정하는 방법에는, 두 군데 하드웨어 중 한 곳에서만 데이터를 받아오도록 ros에서 코드를 짜거나, 혹은 들어오는 두 개의 데이터의 평균값을 받아오도록 코드를 수정하면 될 것이다. 우리 조는 해당 문제를 해결하기 위해 전자의 문제 해결 방법을 택하였다.



위 사진과 같이, 실제 공 사이즈와 탐지되는 공의 사이즈 사이에 linear한 관계가 있다는 사실을 실험적으로 관찰할 수 있었다. 우리 조는 공과의 거리가 65~75cm 떨어져 있을 때부터 gripper를 가동하기 시작하므로, 정확한 구동계 컨트롤을 위해서는 65~75cm 사이에서 정확한 거리 정보를 출력해 주는 것이 중요하다. 따라서 65~75cm 사이에서 나는 픽셀 에러를 보정해 주는 것을 시도하였고, 탐지된 공의 사이즈(반지름 : R)로 거리를 계산하기 이전에 해당 픽셀 에러를 빼주는 과정을 거쳤다.

[R(modified) = R-E (Pixel Error)]

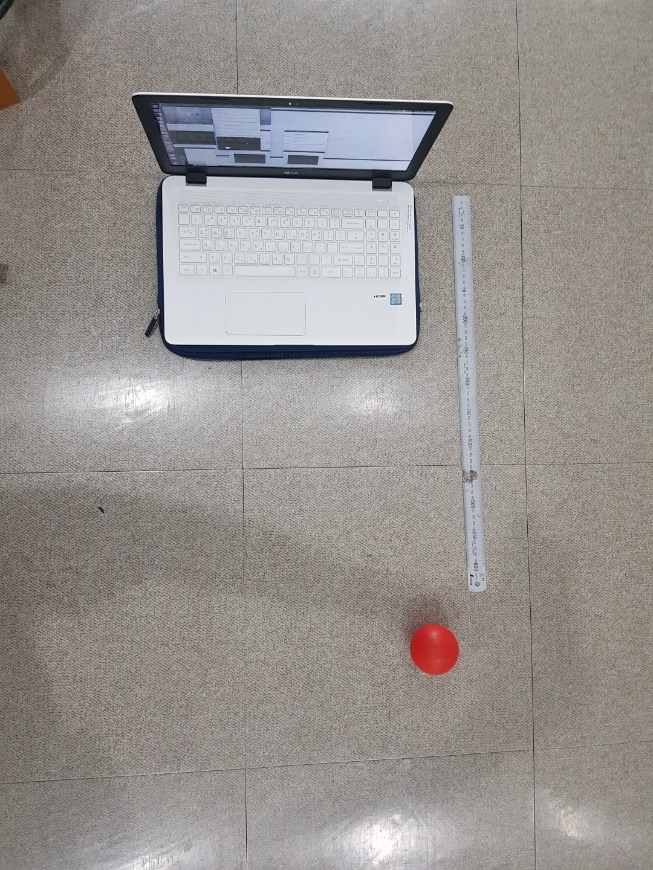
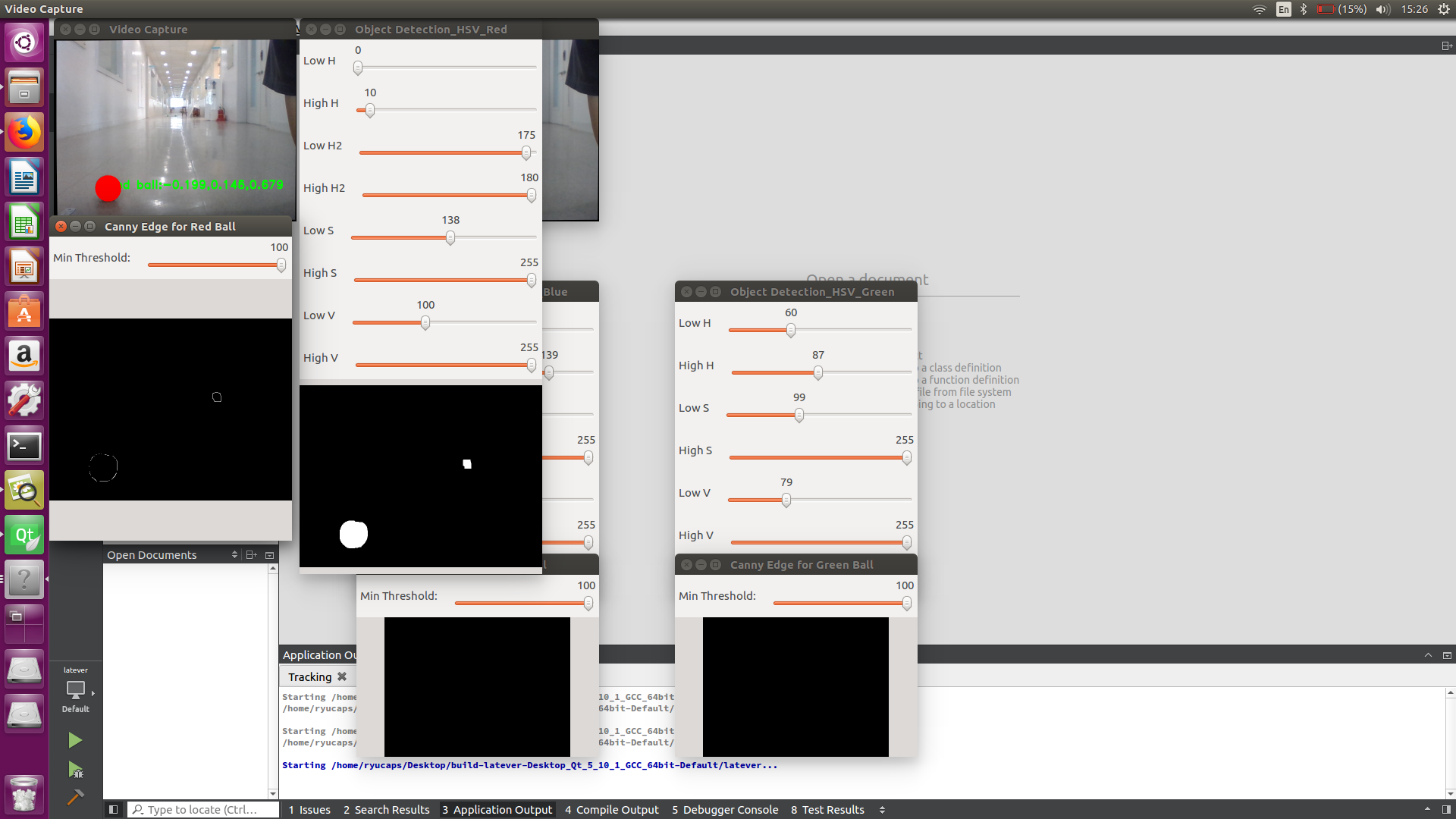
그 결과 기존의 코드에서 68cm 떨어진 물체를 53.8cm로 인식하던 것에 비해,



0.68 m

0.538 m

0.538 m



0.679 m

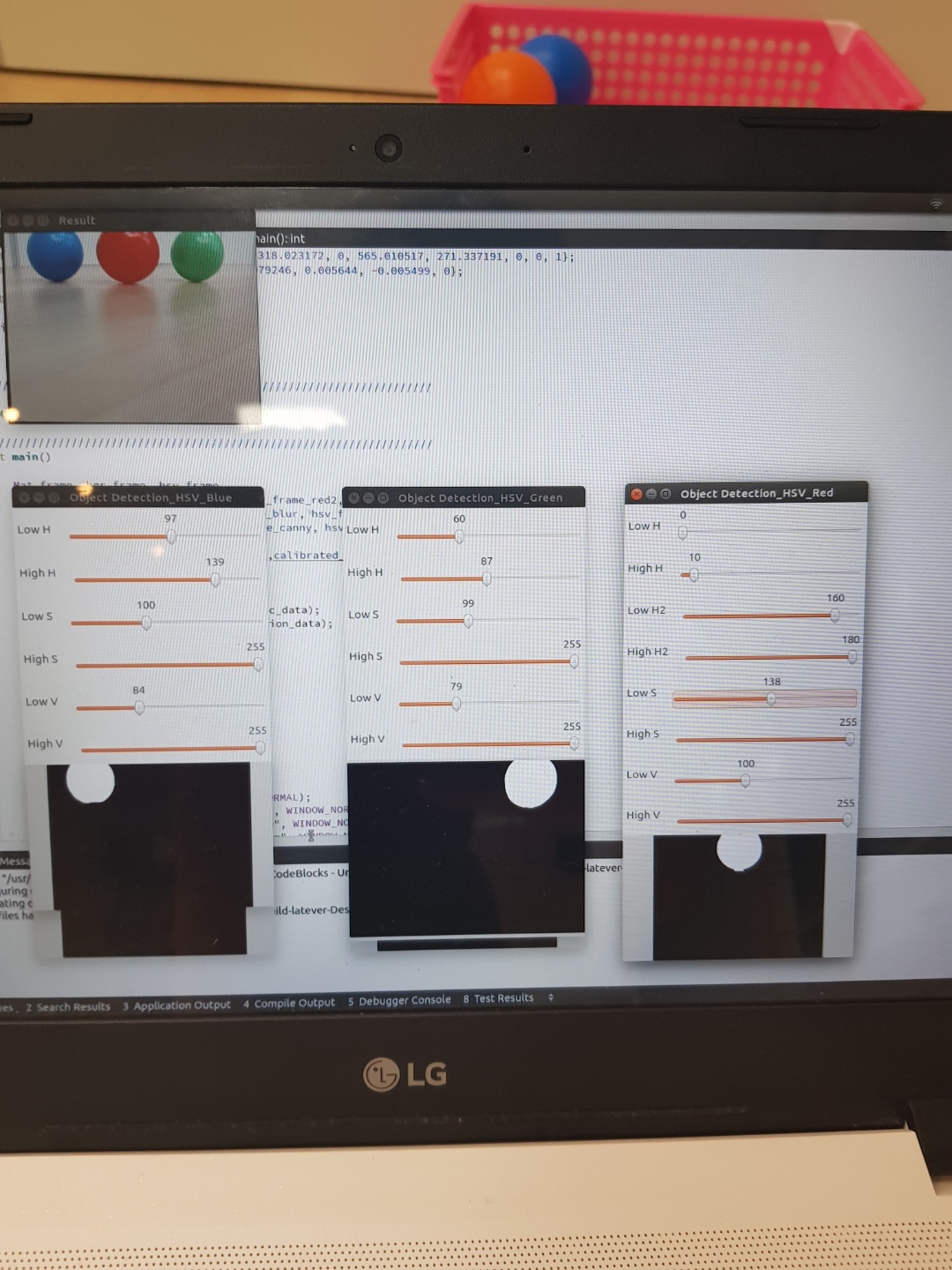
0.68 m

수정한 코드는 67.9cm 떨어진 것으로 인식하여 오차율을 21%에서 1%미만으로 줄일 수 있었다.

파란색 공을 탐지하지 못하는 가장 큰 이유 중 한 가지는 B값만을 기준으로 파란 공을 찾는 본 코드의 알고리즘으로 인한 것인데, 주변 생활의 B값과, 파란공의 B값이 큰 차이를 가지지 않기 때문이다. 이를 보정하기 위해서 white balance라는 visual processing 작업이 추가로 필요했었는데, 이 white balancing이란, Gamut mapping, Gray world assumption, Color card method 등의 다양한 방법을 통해 이미지를 원하는 색조 (일반적으로는 이미지의 색을 자연광 상태에서의 색과 동일하게 맞추는 데에 사용한다) 로 바꾸기 위해 사용하는 알고리즘이다. 하여 해당 알고리즘을 포함하여 본 코드를 수정하였다.

하지만 한가지 문제점이 추가로 발생하였는데, 매 순간 들어오는 프레임 이미지에 white balancing 알고리즘을 추가로 작업하는데 소요되는 시간이 과하게 오래 걸리는 것을 관찰할 수 있었다.

따라서 생각해낸 방법은, white balance를 맞추고 난 뒤 탐지될 파란 공의 B값이 white balance를 맞추기 전엔 얼마였는지 역추적한 후, 본 코드 내에서의 B값 탐지 범위를 조정하는 것이다. 파란색, 빨간색, 연두색 세가지 색상에 모두 똑같은 수정을 거친 결과 세가지 색 모두 안정적으로 탐지 할 수 있게 되었다. 밑의 사진은 탐지된 빨강, 파랑, 연두색 공의 블롭이다. 세가지 색상의 공 모두에서 완벽한 원의 형태가 탐지되고, 주변 환경에서 에러(에러가 될만한 요인 : (1)Saturation, (2)Reflection on floor)를 포함하지 않는 것을 알 수 있다.



**결론**

결과적으로 3조의 open CV 파트는 정확한 거리추정, 색상 탐지 두 가지 지상과제 모두를 완벽에 가깝도록 해결해 낼 수 있었다. 이 후 정확한 거리 정보를 바탕으로 구동계 컨트롤을 Lidar의 도움 없이도 정밀하게 해낼 수 있었고, 서로 다른 색상의 공 또한 완벽한 사이즈로 탐지할 수 있었기에 빨간색 공을 제대로 탐지하지 못하고 파란공과 함께 잡거나, 혹은 파란색 공을 탐지하지 못하여서 놓치는 불상사 없이 과제를 수행 할 수 있는 발판을 마련 하였다. 해당 경험을 바탕으로 창의적 시스템 구현 2를 수행할 때는 정확, 정밀한 시각정보에 Lidar의 동체 추적 정보를 바탕으로 3차원 mapping까지 구현하는 것을 목표로 해 볼 것이다.