



제 2회 자율주행 모형차량 경진대회

Autonomous 최종발표

백재민 | 박준아

Autonomous 최종발표

Index



01

차량설계
에커만 조향&차동기어
베어링 &모터 &LED설계

02

차량구동-인식
차선인식
사물인식

03

차선인식
Flask수동조작
점선 & 실선 인식

04

장애물 탐지방향성1
시뮬레이션 & 실제테스트
아이디어



3D 모델링
내부 설계
외관 설계

차량구동-설계
아두이노 설계
배터리 설계

사물인식
데이터 수집
YOLOv10 custom model

장애물 탐지방향성2
차 & 사람
주차 & 신호등

01 차량 공학적 설계

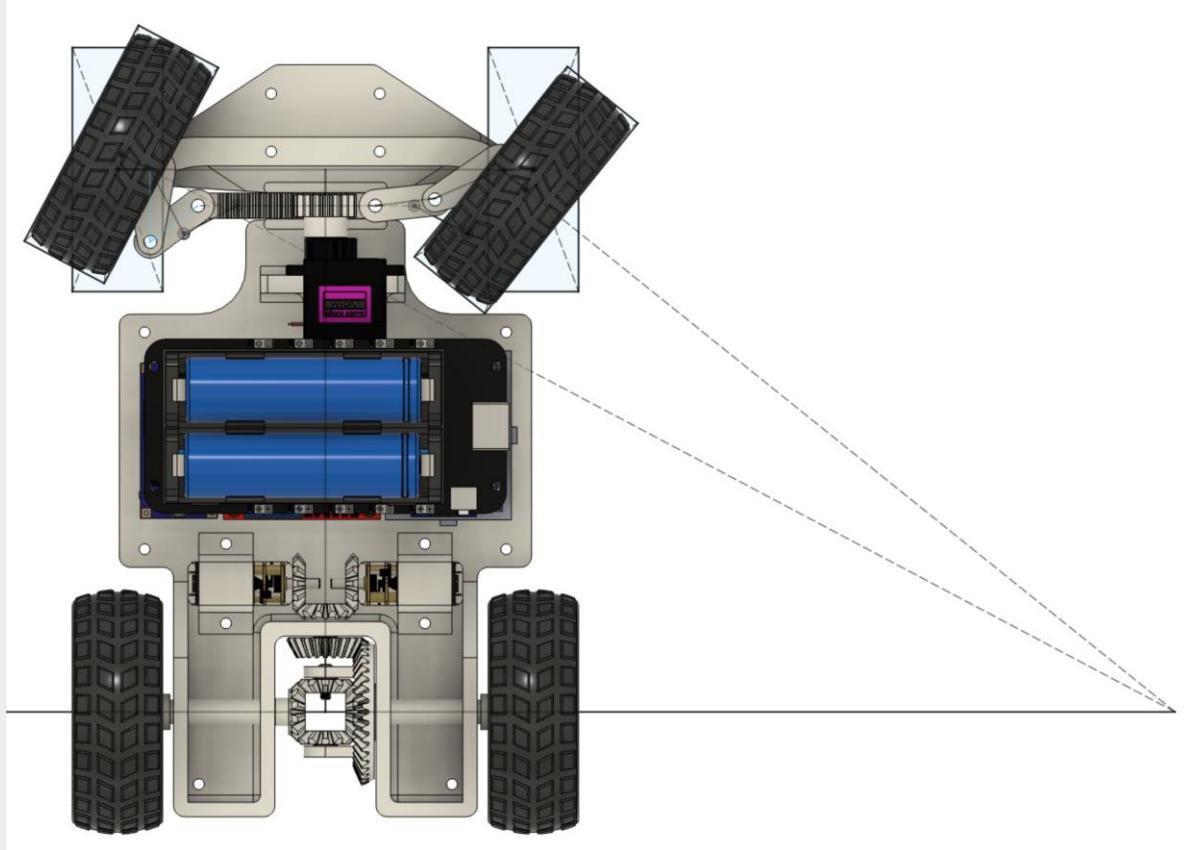
전륜 부분에서는 에커만 조향(Ackermann Steering)을 사용하고,
후륜에서는 차동기어(differential gear) 설계를 적용하여 곡선 주행 시 슬립(slip) 현상을 최소화

전륜 : 에커만 조향

회전 중심을 향하는 방향으로 바퀴가
수직으로 정렬되도록 한다.

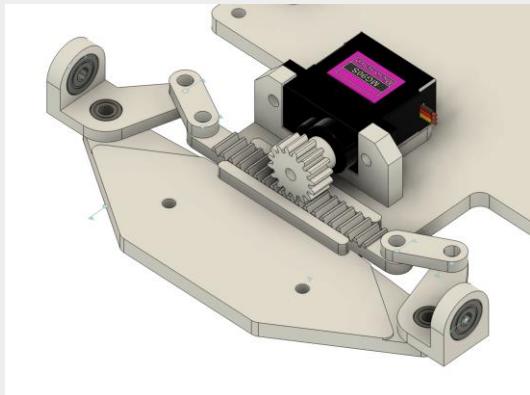
후륜 : 차동기어

곡선 주행 시 외측 바퀴는 빠르게,
내측 바퀴는 더 느리게 회전한다.

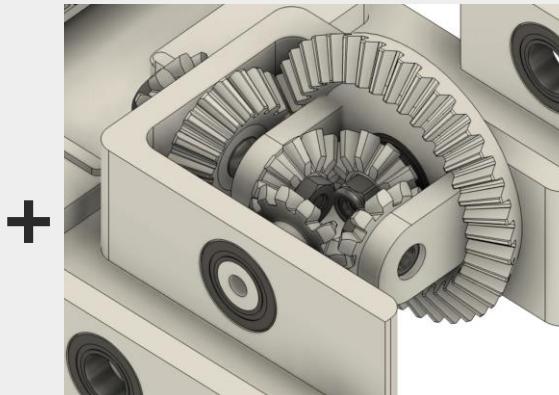


01 차량 공학적 설계

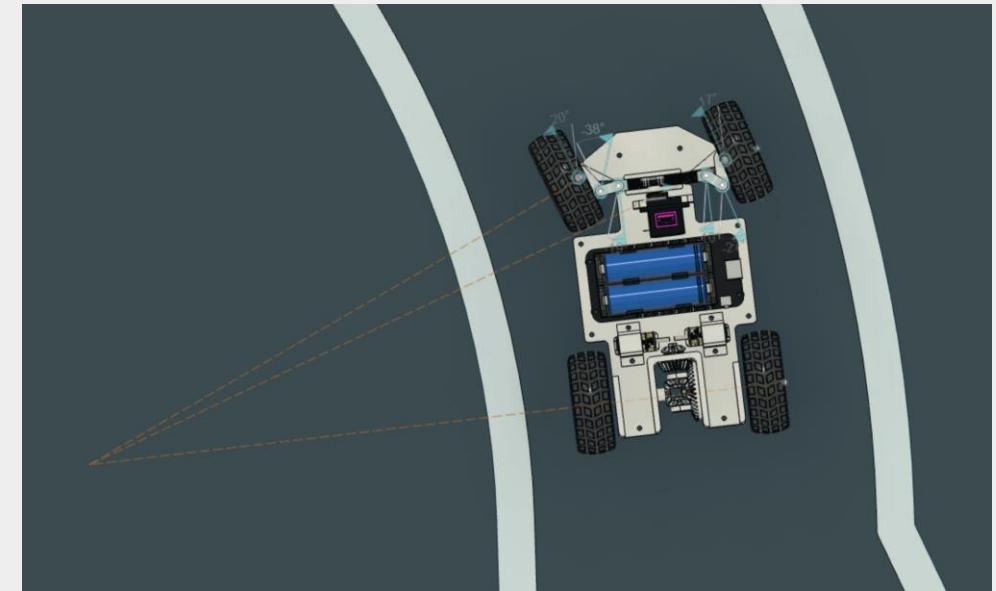
전륜 부분에서는 에커만 조향(Ackermann Steering)을 사용하고,
후륜에서는 차동기어(differential gear) 설계를 적용하여 곡선 주행 시 슬립(slip) 현상을 최소화



에커만 조향



차동기어

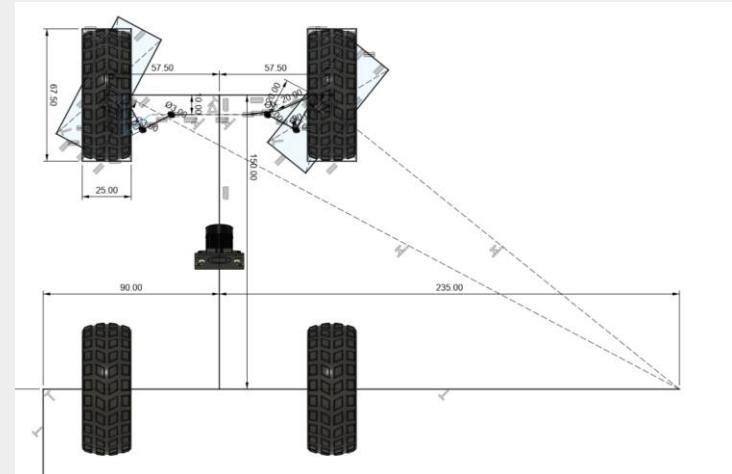
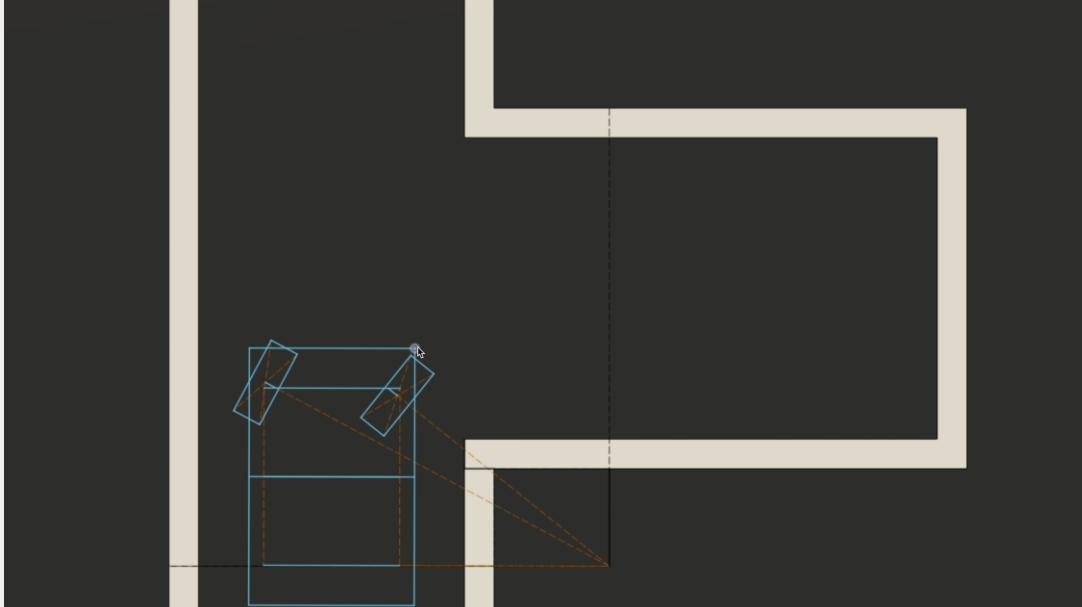


에커만 조향과 차동기어를 직접 3D 모델링하여 경진대회에
적합한 크기로 제작

주행트랙을 3D모델링 하고 주행을 시뮬레이션 해봤을 때
바퀴가 최적의 접지력과 회전력을 유지하도록 하여 운전의
안정성을 높일 수 있었다.

01 차량 공학적 설계

에커만 조향(Ackermann Steering) – 주차



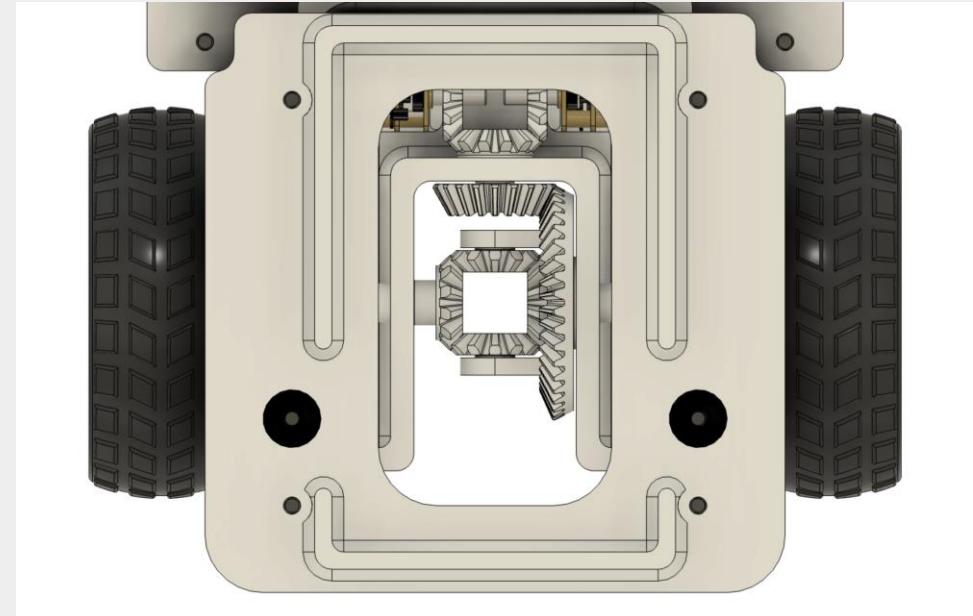
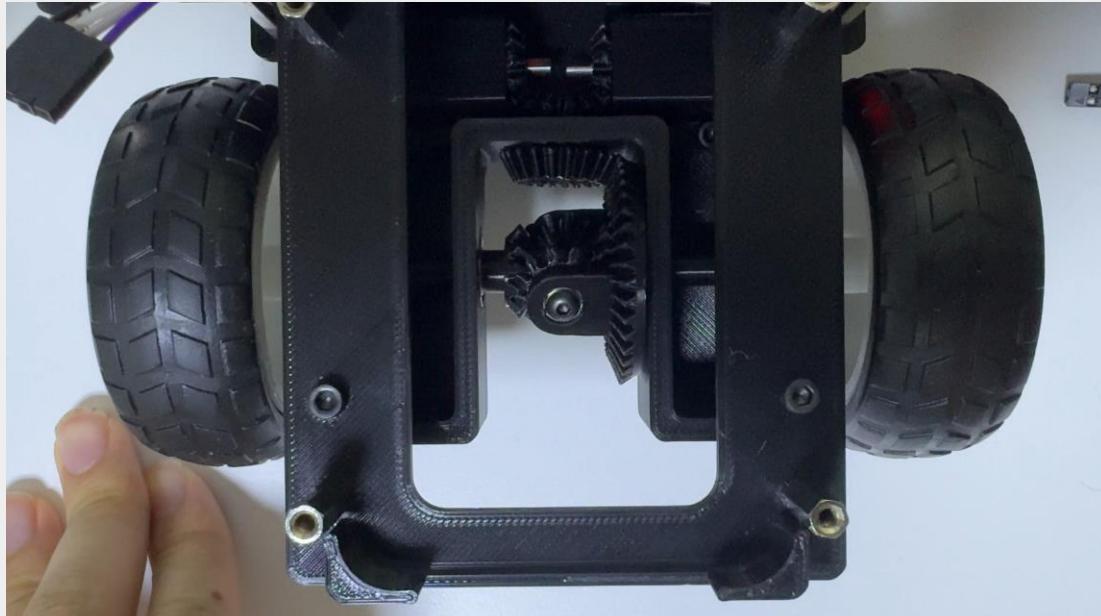
에커만 조향

주차부분의 조향을 원활하게 하기 위해 235mm점을 기준으로 설계를 진행

영상: <https://drive.google.com/file/d/1LY4Bslq5zAYjF-1-Lb-CiBgC1ZaFgqJb/view?usp=sharing>

01 차량 공학적 설계

차동기어(differential gear)



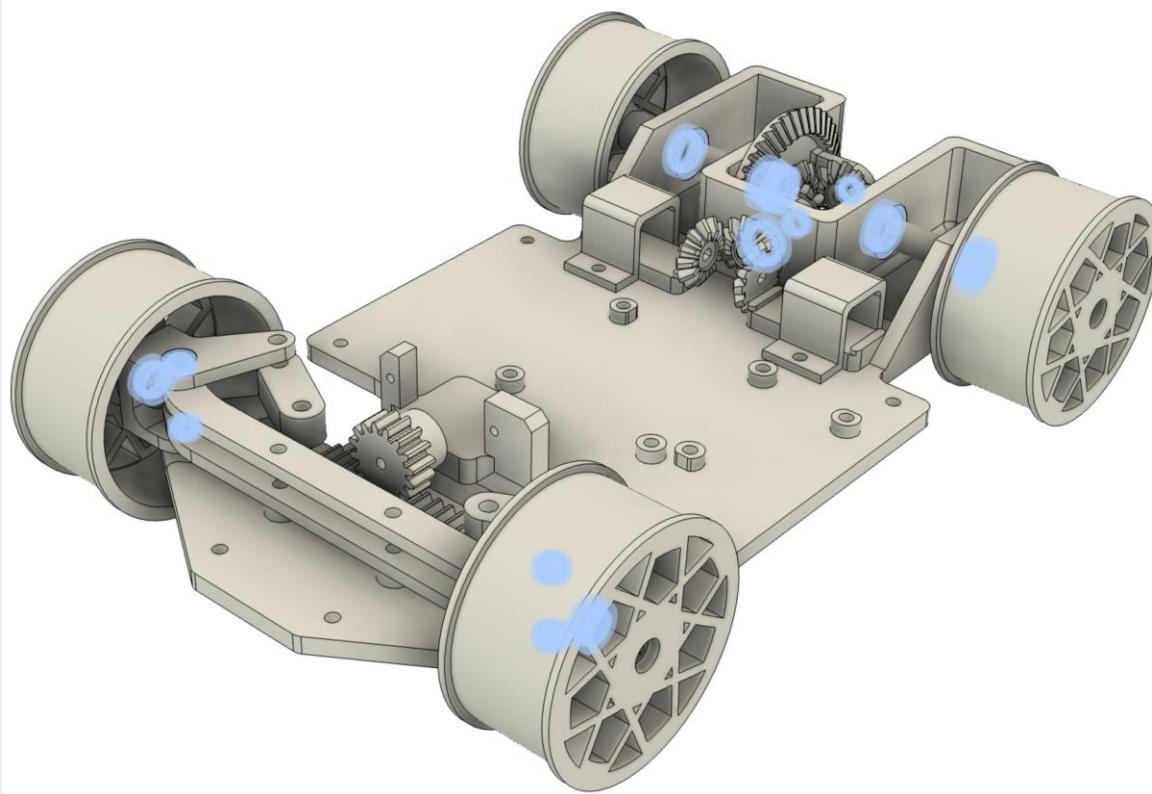
차동기어

차동기어의 베벨기어를 오차를 고려하여 직접 설계하고 베어링을 활용하여 마찰을 최소화, 그 결과 곡선에서의 주행이 훨씬 부드럽게 주행 가능

영상: https://drive.google.com/file/d/1AfJ19zceW9eVdtmJk0rsd53HG0S3NOwt/view?usp=drive_link

01 베어링 설계

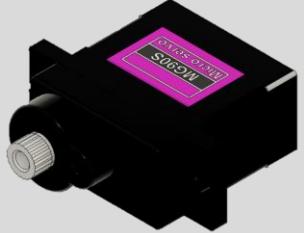
베어링을 활용하여 축 부분을 견고하고 훨씬 부드럽게 회전할 수 있도록 설계



14개의 베어링(파란색 부분)으로 바퀴의 구동부위의 안전성과 부드러움을 향상

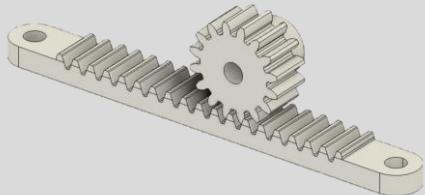
01 모터 설계

서보모터와 랙 앤 피니언 기어를 사용하여 조향, n20 모터 2개와 베벨기어를 사용하여 후륜구동



서보모터(MG90S)

전자적으로 제어되어 정확한 각도로 회전



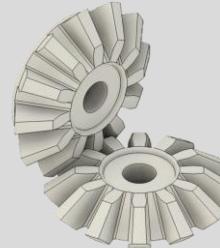
랙 앤 피니언 기어

회전운동을 직선운동으로 변환



n20모터(300RPM)

공간제약으로 인해 소형이지만 충분한 토크를 낼 수 있음

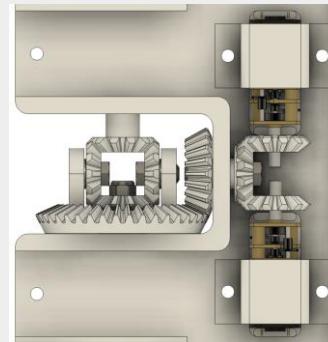


베벨 기어

두 축이 서로 직각으로 교차하는 경우에 회전력을 전달하는데 사용되는 기어



서보모터가 피니언을 돌리면, 랙이 좌우로 움직이게 되어 차량의 바퀴가 좌우로 회전하게 된다.



출력 후 테스트 결과 모터가 1개일 때보다 2개를 이용할 때 더 큰 구동력을 가질 수 있었다.

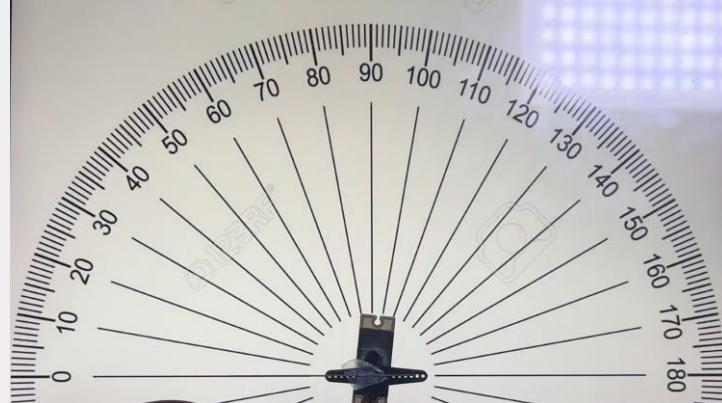
01 모터 설계

서보모터 각도 범위 설계

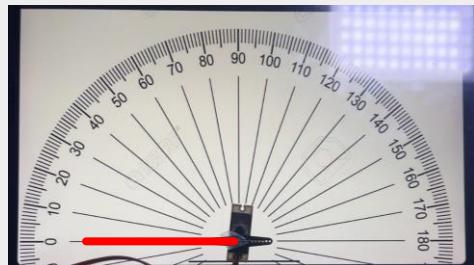


서보모터(MG90S)

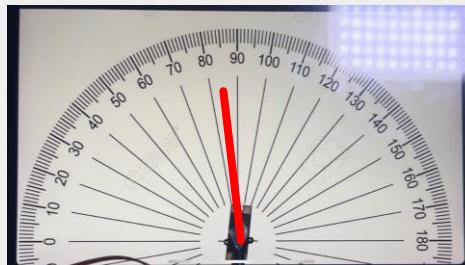
전자적으로 제어되어 정확한 각도로 회전



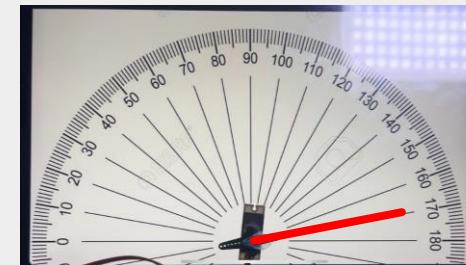
영상:<https://drive.google.com/file/d/1mKsQWh7G1hvbx871F1kROuQ4hWicur/view?usp=sharing>



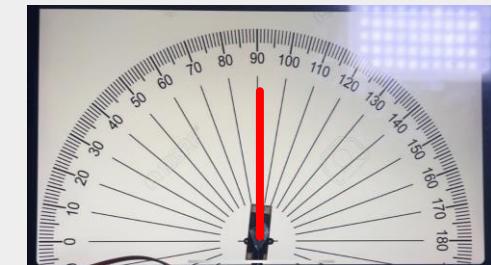
0°



90°



180°



95°

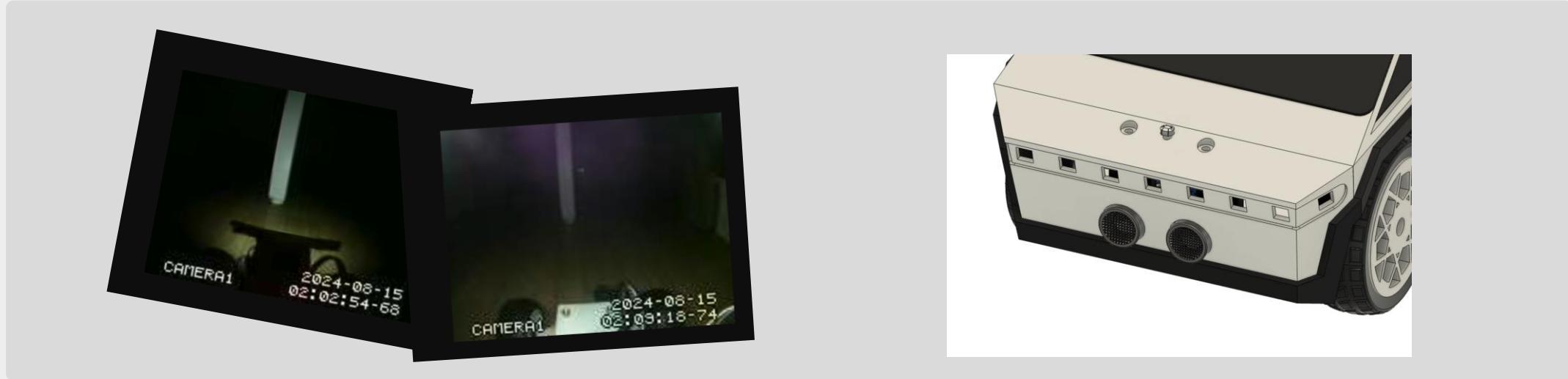
조향 각도의 범위를 10° ~170°로 설정하여 설계를 진행



Autonomous

01 LED 설계

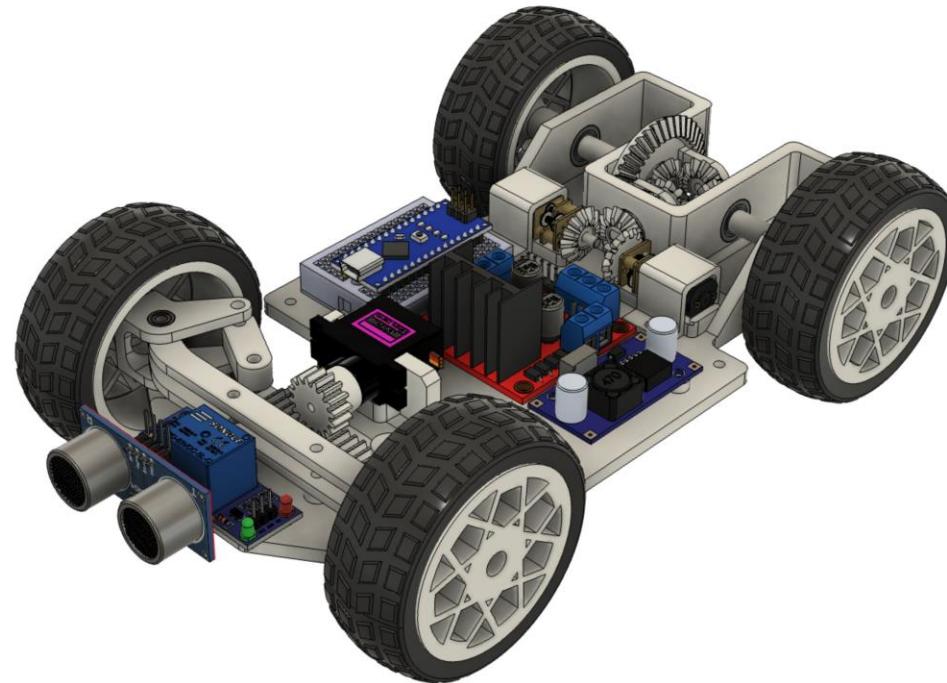
터널에서 쉽게 차량을 인식할 수 있도록 적절한 LED를 사용했다.



9개의 12VLED를 사용하고 전방 뿐만 아니라 측면으로도 LED를 배치하여 터널 속에서 훨씬 더 원활하게
차선을 인식할 수 있도록 설계

01 내부 설계

First Floor



초음파센서, 릴레이모듈, 서보모터, 모터드라이버, n20모터, 아두이노

>선정리용이

01 내부 설계

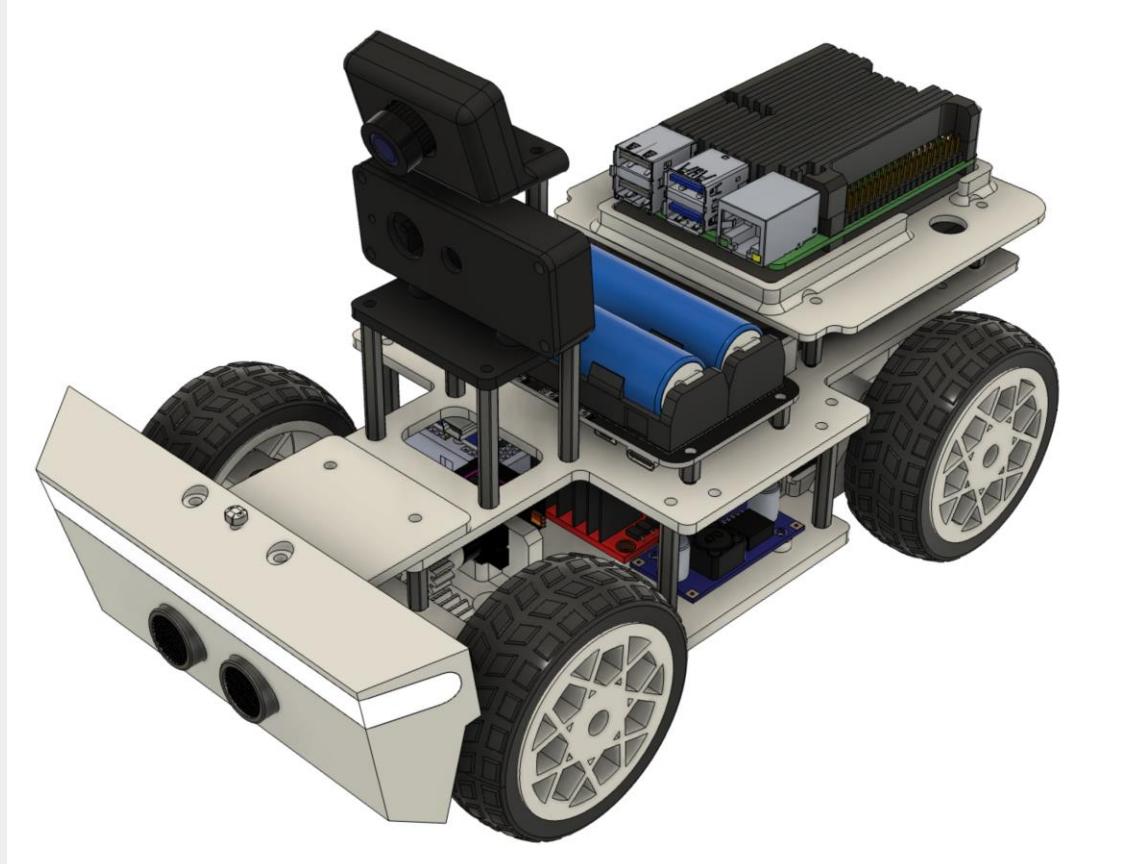
Second Floor



LED, 조도센서, 배터리
> 적절한 전력을 위해 배터리는 2개를 사용

01 내부 설계

Third floor



카메라. 라즈베리파이

->방열을 위해 라즈베리파이를 상단에 배치

01 내부 설계

Cyber Truck



최종 외관 디자인은 사이버트럭을 오마주하여 설계

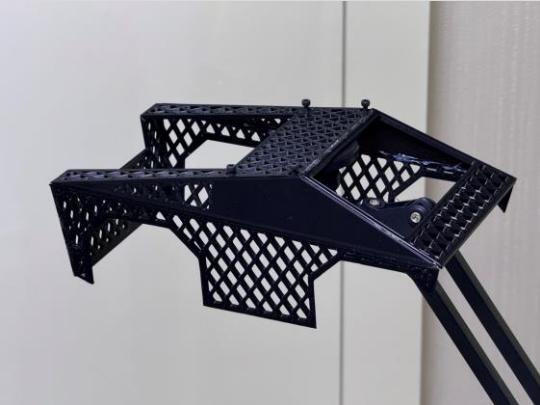


Auto
nomous

Auto
nomous

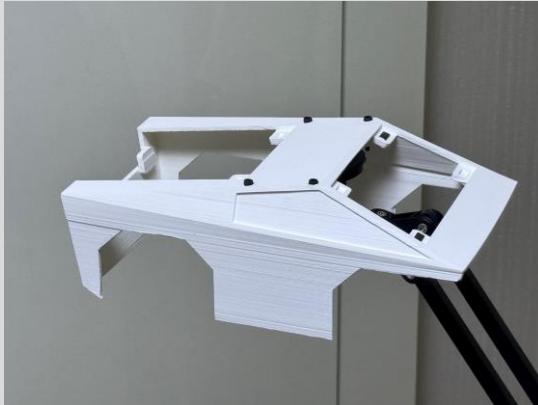
01 외관 설계

3가지 후보를 선정하여 직접 제작



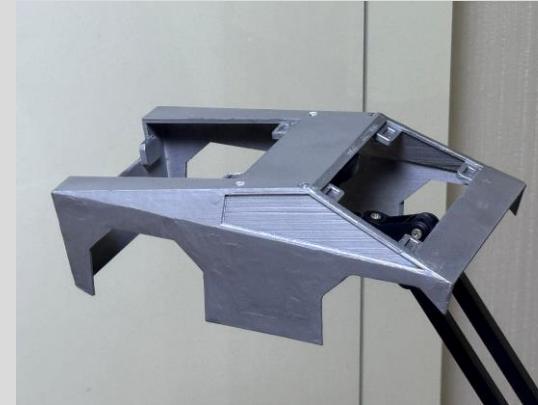
MASH

무게를 줄여 속도를 더 빠르게 할 수
있다는 장점이 있다.



SIMPLE

가장 간단한 모형으로
제일 적합하다고 판단했다.

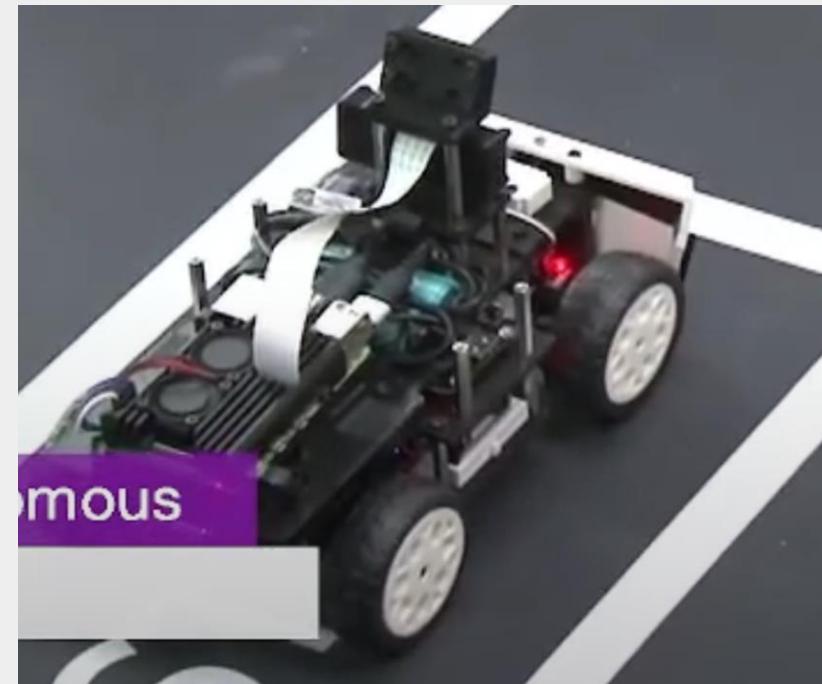
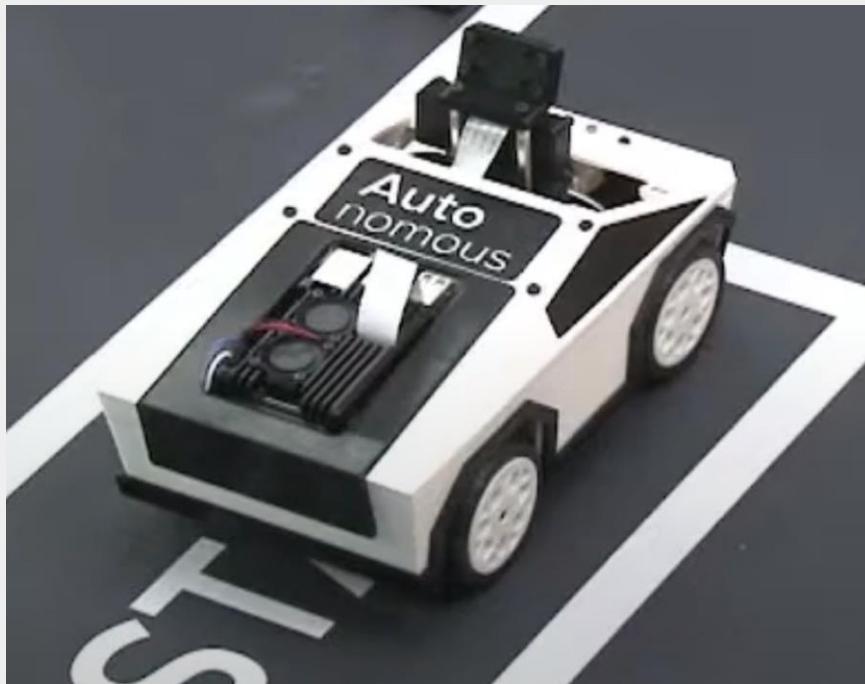


SILEVER

이쁘고 단단하지만 무겁다는 단점이
있다.

01 외관 설계

조금이라도 랩타임을 줄이기 위해 대회에서는 외관 탈거 진행

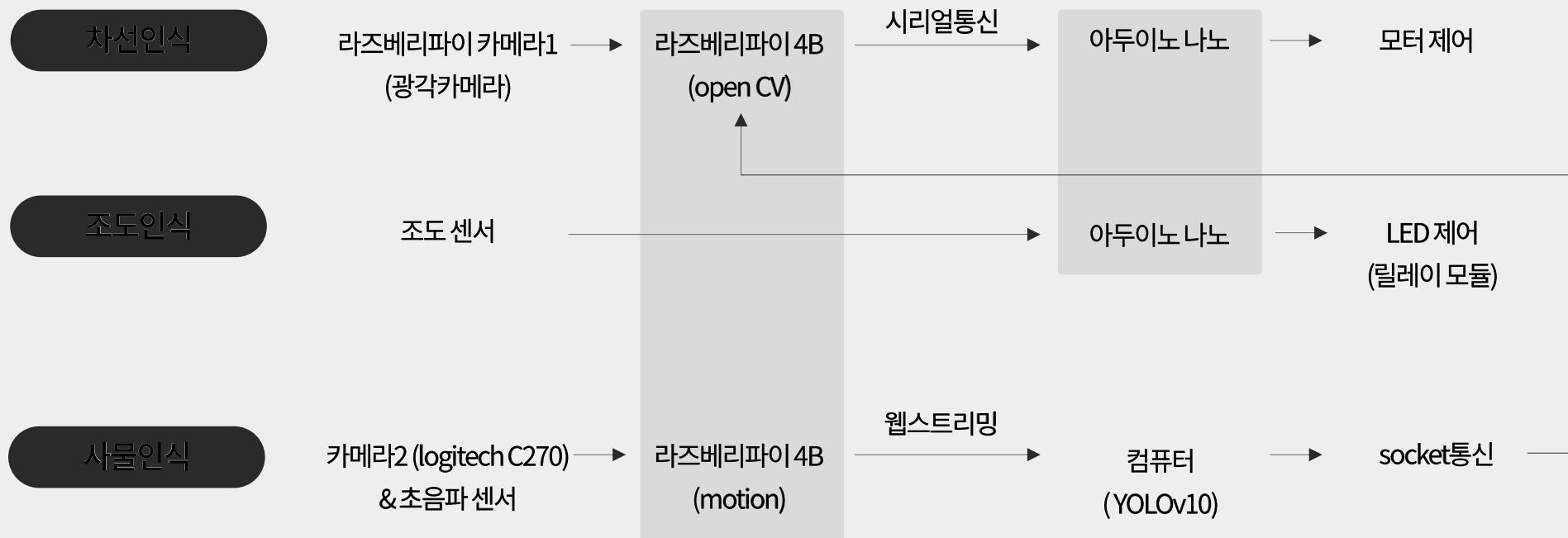


랩타임: 약 24초
(직접 타이머 측정)

랩타임: 약 22초
(직접 타이머 측정)

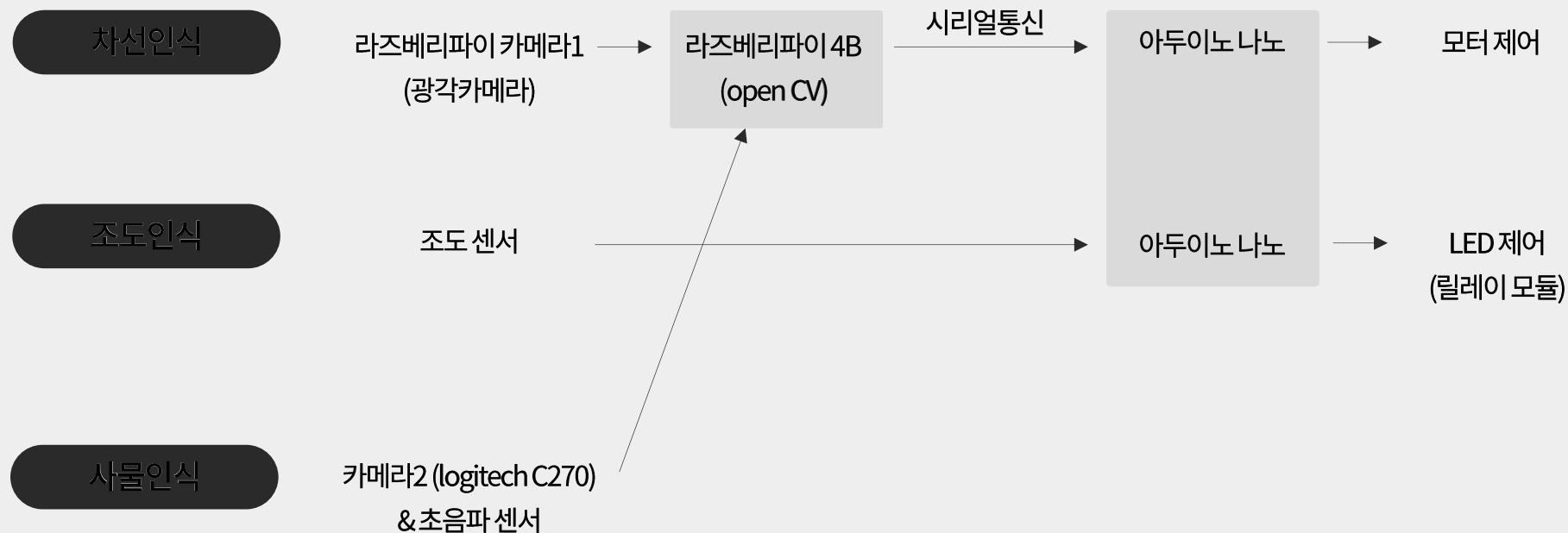
02 차량구동 - 인식 - CASE1

차량구동을 위해서는 차선인식과 사물인식이 필수적이다. 따라서 차선인식에 라즈베리파이와 아두이노나노를 사용하고, 사물인식(신호등 색상포함)은 라즈베리파이를 사용하여 컴퓨터로 영상을 전송하고 이를 다시 라즈베리파이에 결과값을 전송



02 차량구동 - 인식 - CASE2

차량구동을 위해서는 차선인식과 사물인식이 필수적이다. 따라서 차선인식에 라즈베리파이와 아두이노나노를 사용하고, 추가적으로 사물인식(신호등 색상포함)은 초음파센서를 이용한다.



02 차량구동 - 설계

사용부품



라즈베리파이4B



n20 모터



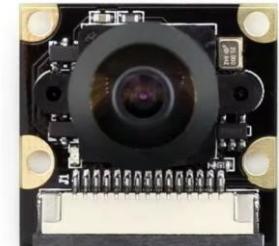
L298n모터드라이버



18650 배터리& UPS



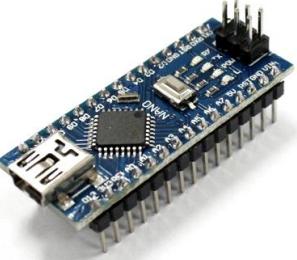
웹카메라



광각카메라



초음파센서



아두이노나노



MG90S 서보모터



스텝업 모듈



조도센서



12V LED



보조배터리

라즈베리파이&아두이노&사물인식과 &차선인식을 고려하여 위의 부품들을 사용해서 설계

02 차량구동 - 설계

아두이노나노 보드 사용 이유

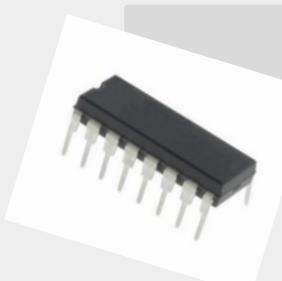
1. 서보모터

서보모터를 라즈베리파이 GPIO로 제어할 때 지터링 현상이 발생할 수 있다. 이를 해결하기 위해 pigpiod 라이브러리를 사용할 수 있지만, 고장이 쉽게 날 수 있다는 단점이 있다.
따라서 하드웨어적으로 지터링 현상을 해결하기 위해 아두이노 보드를 활용하기로 했고, 그중 소형 보드인 아두이노나노를 사용하기로 결정했다.



2. 조도센서

일반적인 조도센서를 라즈베리파이에서 작동시키기 위해서는 아날로그 신호를 디지털 신호로 변환시켜주는 MCP3008과 같은 부품이 필요하다.
하지만 아두이노를 사용한다면 디지털 신호를 받는 핀이 있기 때문에 위와 같은 부품이 필요없다.



3. 방열

방열을 위해 라즈베리파이를 상단에 위치시키기 위해서는 아두이노를 사용해 모터와 LED를 제어하고 하는 방법이 제일 적합하다고 판단했다.

02 차량구동 - 배터리

Ups+보조배터리



라즈베리파이 4B

5V 3A (권장)
5V 1.5A (실제)



n20 모터 2개

6V 1A (권장)
9V 0.7A (실제)



서보모터

5V 1A (권장)
5V 1A (실제)



12V LED

12V 0.7A (권장)
9V 0.7A (실제)



5V 4A UPS

라즈베리파이+기타부품



5V 3A UPS

+



20W 보조배터리

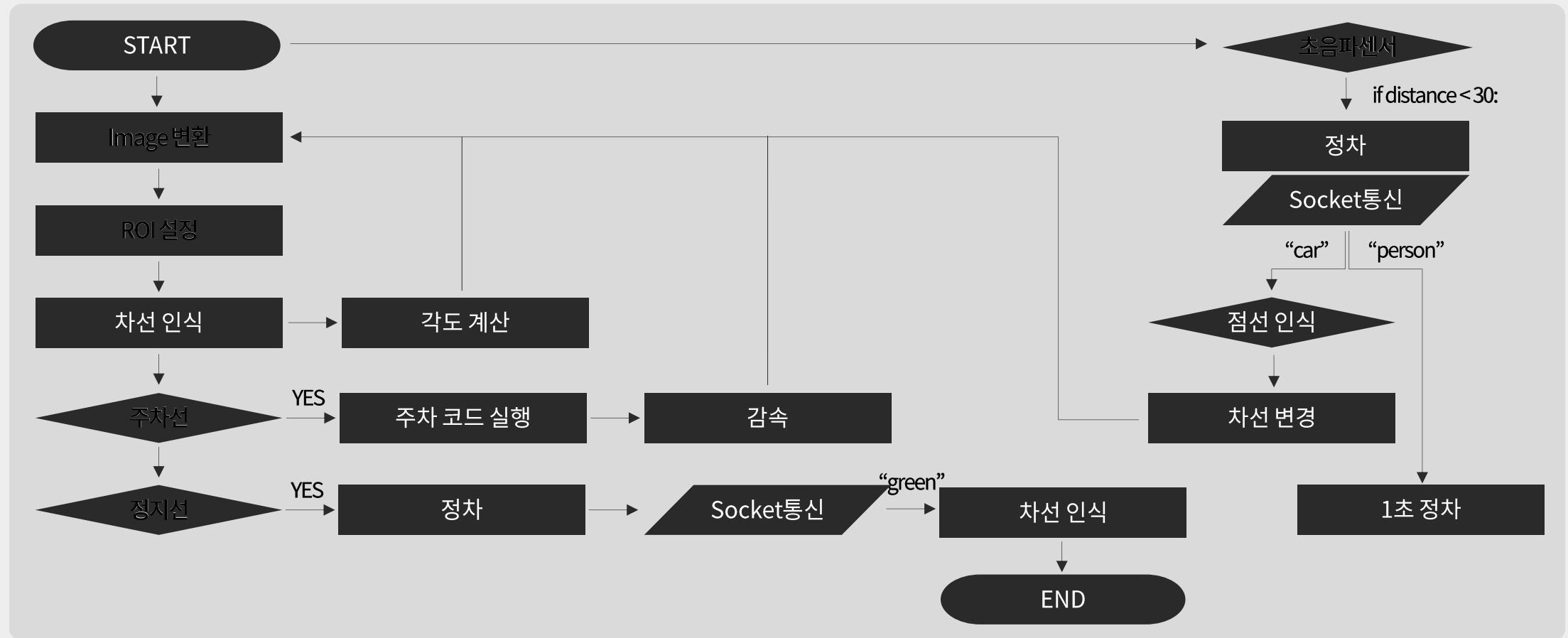
기타부품

라즈베리파이

03 차선인식

라즈베리파이에서 작동하는 코드의 전체 흐름도

정지선을 인식할 때(신호등)와 초음파 센서로 거리를 인식했을 때(차/사람) socket 통신을 진행함으로써 불필요한 통신으로 인한 속도 저하 현상을 방지했다.

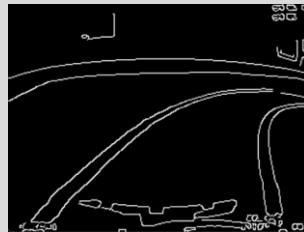


03 차선인식

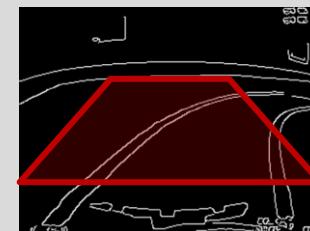
OpenCV를 활용한 Canny Edge 검출 방식과 sliding window 추적 기술을 활용



Original Frame



Canny



Warp



Bird-Eye View



차선용 ROI 설정



Sliding Window

차선인식을 통해 얻은 차선의 중앙값과 차량의 중앙값을 비교하여 deviation값을 계산

03 차선인식

Flask를 활용하여 수동조작을 진행

Motor and Servo Control

DC 모터 및 서보모터 제어

속도 조절:  현재 속도: 0
각도 조절:  현재 각도: 80

LED 토글 (스페이스바)

LED 상태: OFF

```
93
94 k_values = [150, 140, 130, 120, 110, 100, 90, 80, 70, 60, 50, 40]
95 j_values = [-170, -150, -120, -100, -80, -20, 5, 30, 60, 120, 140, 160]
96
```



A black arrow pointing to the right, indicating a continuation or next step.



```

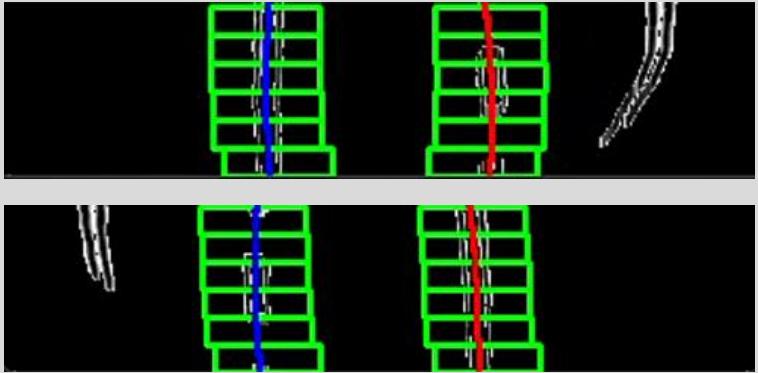
134    def calculate_mean(self):
135        """Calculate mean of the data set.
136
137        Returns:
138            float: Mean of the data set.
139
140        """
141        if len(self.data_set) < 1:
142            raise ValueError("Data set is empty")
143
144        sum = 0
145        for i in range(len(self.data_set)):
146            sum += self.data_set[i]
147
148        return sum / len(self.data_set)
149
150    def calculate_std_deviation(self):
151        """Calculate standard deviation of the data set.
152
153        Returns:
154            float: Standard deviation of the data set.
155
156        """
157        if len(self.data_set) < 2:
158            raise ValueError("Data set is empty or has less than two elements")
159
160        mean = self.calculate_mean()
161
162        sum = 0
163        for i in range(len(self.data_set)):
164            sum += (self.data_set[i] - mean) ** 2
165
166        variance = sum / len(self.data_set)
167
168        return variance ** 0.5
169
170    def calculate_min(self):
171        """Calculate minimum value of the data set.
172
173        Returns:
174            float: Minimum value of the data set.
175
176        """
177        if len(self.data_set) < 1:
178            raise ValueError("Data set is empty")
179
180        min_value = self.data_set[0]
181
182        for i in range(1, len(self.data_set)):
183            if self.data_set[i] < min_value:
184                min_value = self.data_set[i]
185
186        return min_value
187
188    def calculate_max(self):
189        """Calculate maximum value of the data set.
190
191        Returns:
192            float: Maximum value of the data set.
193
194        """
195        if len(self.data_set) < 1:
196            raise ValueError("Data set is empty")
197
198        max_value = self.data_set[0]
199
200        for i in range(1, len(self.data_set)):
201            if self.data_set[i] > max_value:
202                max_value = self.data_set[i]
203
204        return max_value
205
206    def calculate_range(self):
207        """Calculate range of the data set.
208
209        Returns:
210            float: Range of the data set.
211
212        """
213        if len(self.data_set) < 2:
214            raise ValueError("Data set is empty or has less than two elements")
215
216        min_value = self.calculate_min()
217        max_value = self.calculate_max()
218
219        return max_value - min_value
220
221    def calculate_percentile(self, percentile):
222        """Calculate percentile of the data set.
223
224        Args:
225            percentile: Percentile to calculate.
226
227        Returns:
228            float: Percentile of the data set.
229
230        """
231        if len(self.data_set) < 1:
232            raise ValueError("Data set is empty")
233
234        sorted_data_set = sorted(self.data_set)
235
236        index = int((percentile / 100) * len(sorted_data_set))
237
238        if index == len(sorted_data_set):
239            return sorted_data_set[-1]
240
241        if index == 0:
242            return sorted_data_set[0]
243
244        return sorted_data_set[index]
245
246    def calculate_skewness(self):
247        """Calculate skewness of the data set.
248
249        Returns:
250            float: Skewness of the data set.
251
252        """
253        if len(self.data_set) < 3:
254            raise ValueError("Data set is empty or has less than three elements")
255
256        mean = self.calculate_mean()
257        std_deviation = self.calculate_std_deviation()
258
259        sum = 0
260        for i in range(len(self.data_set)):
261            sum += (self.data_set[i] - mean) ** 3
262
263        skewness = sum / (len(self.data_set) * std_deviation ** 3)
264
265        return skewness
266
267    def calculate_kurtosis(self):
268        """Calculate kurtosis of the data set.
269
270        Returns:
271            float: Kurtosis of the data set.
272
273        """
274        if len(self.data_set) < 4:
275            raise ValueError("Data set is empty or has less than four elements")
276
277        mean = self.calculate_mean()
278        std_deviation = self.calculate_std_deviation()
279
280        sum = 0
281        for i in range(len(self.data_set)):
282            sum += (self.data_set[i] - mean) ** 4
283
284        kurtosis = sum / (len(self.data_set) * std_deviation ** 4)
285
286        return kurtosis
287
288    def calculate_z_scores(self):
289        """Calculate z-scores of the data set.
290
291        Returns:
292            list: List of z-scores for each element in the data set.
293
294        """
295        if len(self.data_set) < 1:
296            raise ValueError("Data set is empty")
297
298        mean = self.calculate_mean()
299        std_deviation = self.calculate_std_deviation()
300
301        z_scores = []
302        for i in range(len(self.data_set)):
303            z_scores.append((self.data_set[i] - mean) / std_deviation)
304
305        return z_scores
306
307    def calculate_boxplot(self):
308        """Calculate boxplot of the data set.
309
310        Returns:
311            dict: Dictionary containing the boxplot statistics.
312
313        """
314        if len(self.data_set) < 5:
315            raise ValueError("Data set is empty or has less than five elements")
316
317        min_value = self.calculate_min()
318        max_value = self.calculate_max()
319
320        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
321
322        boxplot = {
323            "min": min_value,
324            "q1": quartiles[0],
325            "median": quartiles[1],
326            "q3": quartiles[2],
327            "max": max_value
328        }
329
330        return boxplot
331
332    def calculate_iqr(self):
333        """Calculate interquartile range of the data set.
334
335        Returns:
336            float: Interquartile range of the data set.
337
338        """
339        if len(self.data_set) < 5:
340            raise ValueError("Data set is empty or has less than five elements")
341
342        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
343
344        iqr = quartiles[2] - quartiles[0]
345
346        return iqr
347
348    def calculate_outliers(self):
349        """Calculate outliers of the data set.
350
351        Returns:
352            list: List of outliers in the data set.
353
354        """
355        if len(self.data_set) < 5:
356            raise ValueError("Data set is empty or has less than five elements")
357
358        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
359
360        iqr = quartiles[2] - quartiles[0]
361
362        lower_bound = quartiles[0] - (1.5 * iqr)
363        upper_bound = quartiles[2] + (1.5 * iqr)
364
365        outliers = []
366        for i in range(len(self.data_set)):
367            if self.data_set[i] < lower_bound or self.data_set[i] > upper_bound:
368                outliers.append(self.data_set[i])
369
370        return outliers
371
372    def calculate_z_scores(self):
373        """Calculate z-scores of the data set.
374
375        Returns:
376            list: List of z-scores for each element in the data set.
377
378        """
379        if len(self.data_set) < 1:
380            raise ValueError("Data set is empty")
381
382        mean = self.calculate_mean()
383        std_deviation = self.calculate_std_deviation()
384
385        z_scores = []
386        for i in range(len(self.data_set)):
387            z_scores.append((self.data_set[i] - mean) / std_deviation)
388
389        return z_scores
390
391    def calculate_boxplot(self):
392        """Calculate boxplot of the data set.
393
394        Returns:
395            dict: Dictionary containing the boxplot statistics.
396
397        """
398        if len(self.data_set) < 5:
399            raise ValueError("Data set is empty or has less than five elements")
400
401        min_value = self.calculate_min()
402        max_value = self.calculate_max()
403
404        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
405
406        boxplot = {
407            "min": min_value,
408            "q1": quartiles[0],
409            "median": quartiles[1],
410            "q3": quartiles[2],
411            "max": max_value
412        }
413
414        return boxplot
415
416    def calculate_iqr(self):
417        """Calculate interquartile range of the data set.
418
419        Returns:
420            float: Interquartile range of the data set.
421
422        """
423        if len(self.data_set) < 5:
424            raise ValueError("Data set is empty or has less than five elements")
425
426        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
427
428        iqr = quartiles[2] - quartiles[0]
429
430        return iqr
431
432    def calculate_outliers(self):
433        """Calculate outliers of the data set.
434
435        Returns:
436            list: List of outliers in the data set.
437
438        """
439        if len(self.data_set) < 5:
440            raise ValueError("Data set is empty or has less than five elements")
441
442        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
443
444        iqr = quartiles[2] - quartiles[0]
445
446        lower_bound = quartiles[0] - (1.5 * iqr)
447        upper_bound = quartiles[2] + (1.5 * iqr)
448
449        outliers = []
450        for i in range(len(self.data_set)):
451            if self.data_set[i] < lower_bound or self.data_set[i] > upper_bound:
452                outliers.append(self.data_set[i])
453
454        return outliers
455
456    def calculate_z_scores(self):
457        """Calculate z-scores of the data set.
458
459        Returns:
460            list: List of z-scores for each element in the data set.
461
462        """
463        if len(self.data_set) < 1:
464            raise ValueError("Data set is empty")
465
466        mean = self.calculate_mean()
467        std_deviation = self.calculate_std_deviation()
468
469        z_scores = []
470        for i in range(len(self.data_set)):
471            z_scores.append((self.data_set[i] - mean) / std_deviation)
472
473        return z_scores
474
475    def calculate_boxplot(self):
476        """Calculate boxplot of the data set.
477
478        Returns:
479            dict: Dictionary containing the boxplot statistics.
479
480        """
481        if len(self.data_set) < 5:
482            raise ValueError("Data set is empty or has less than five elements")
483
484        min_value = self.calculate_min()
485        max_value = self.calculate_max()
486
487        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
488
489        boxplot = {
490            "min": min_value,
491            "q1": quartiles[0],
492            "median": quartiles[1],
493            "q3": quartiles[2],
494            "max": max_value
495        }
496
497        return boxplot
498
499    def calculate_iqr(self):
500        """Calculate interquartile range of the data set.
501
502        Returns:
503            float: Interquartile range of the data set.
504
505        """
506        if len(self.data_set) < 5:
507            raise ValueError("Data set is empty or has less than five elements")
508
509        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
510
511        iqr = quartiles[2] - quartiles[0]
512
513        return iqr
514
515    def calculate_outliers(self):
516        """Calculate outliers of the data set.
517
518        Returns:
519            list: List of outliers in the data set.
519
520        """
521        if len(self.data_set) < 5:
522            raise ValueError("Data set is empty or has less than five elements")
523
524        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
525
526        iqr = quartiles[2] - quartiles[0]
527
528        lower_bound = quartiles[0] - (1.5 * iqr)
529        upper_bound = quartiles[2] + (1.5 * iqr)
530
531        outliers = []
532        for i in range(len(self.data_set)):
533            if self.data_set[i] < lower_bound or self.data_set[i] > upper_bound:
534                outliers.append(self.data_set[i])
535
536        return outliers
537
538    def calculate_z_scores(self):
539        """Calculate z-scores of the data set.
540
541        Returns:
542            list: List of z-scores for each element in the data set.
543
544        """
545        if len(self.data_set) < 1:
546            raise ValueError("Data set is empty")
547
548        mean = self.calculate_mean()
549        std_deviation = self.calculate_std_deviation()
550
551        z_scores = []
552        for i in range(len(self.data_set)):
553            z_scores.append((self.data_set[i] - mean) / std_deviation)
554
555        return z_scores
556
557    def calculate_boxplot(self):
558        """Calculate boxplot of the data set.
559
559        Returns:
560            dict: Dictionary containing the boxplot statistics.
560
561        """
562        if len(self.data_set) < 5:
563            raise ValueError("Data set is empty or has less than five elements")
564
565        min_value = self.calculate_min()
566        max_value = self.calculate_max()
567
568        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
569
570        boxplot = {
571            "min": min_value,
572            "q1": quartiles[0],
573            "median": quartiles[1],
574            "q3": quartiles[2],
575            "max": max_value
576        }
577
578        return boxplot
579
580    def calculate_iqr(self):
581        """Calculate interquartile range of the data set.
582
583        Returns:
584            float: Interquartile range of the data set.
584
585        """
586        if len(self.data_set) < 5:
587            raise ValueError("Data set is empty or has less than five elements")
588
589        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
590
591        iqr = quartiles[2] - quartiles[0]
592
593        return iqr
594
595    def calculate_outliers(self):
596        """Calculate outliers of the data set.
597
598        Returns:
599            list: List of outliers in the data set.
599
600        """
601        if len(self.data_set) < 5:
602            raise ValueError("Data set is empty or has less than five elements")
603
604        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
605
606        iqr = quartiles[2] - quartiles[0]
607
608        lower_bound = quartiles[0] - (1.5 * iqr)
609        upper_bound = quartiles[2] + (1.5 * iqr)
610
611        outliers = []
612        for i in range(len(self.data_set)):
613            if self.data_set[i] < lower_bound or self.data_set[i] > upper_bound:
614                outliers.append(self.data_set[i])
615
616        return outliers
617
618    def calculate_z_scores(self):
619        """Calculate z-scores of the data set.
620
621        Returns:
622            list: List of z-scores for each element in the data set.
623
624        """
625        if len(self.data_set) < 1:
626            raise ValueError("Data set is empty")
627
628        mean = self.calculate_mean()
629        std_deviation = self.calculate_std_deviation()
630
631        z_scores = []
632        for i in range(len(self.data_set)):
633            z_scores.append((self.data_set[i] - mean) / std_deviation)
634
635        return z_scores
636
637    def calculate_boxplot(self):
638        """Calculate boxplot of the data set.
639
639        Returns:
640            dict: Dictionary containing the boxplot statistics.
640
641        """
642        if len(self.data_set) < 5:
643            raise ValueError("Data set is empty or has less than five elements")
644
645        min_value = self.calculate_min()
646        max_value = self.calculate_max()
647
648        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
649
650        boxplot = {
651            "min": min_value,
652            "q1": quartiles[0],
653            "median": quartiles[1],
654            "q3": quartiles[2],
655            "max": max_value
656        }
657
658        return boxplot
659
660    def calculate_iqr(self):
661        """Calculate interquartile range of the data set.
662
663        Returns:
664            float: Interquartile range of the data set.
664
665        """
666        if len(self.data_set) < 5:
667            raise ValueError("Data set is empty or has less than five elements")
668
669        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
670
671        iqr = quartiles[2] - quartiles[0]
672
673        return iqr
674
675    def calculate_outliers(self):
676        """Calculate outliers of the data set.
677
678        Returns:
679            list: List of outliers in the data set.
679
680        """
681        if len(self.data_set) < 5:
682            raise ValueError("Data set is empty or has less than five elements")
683
684        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
685
686        iqr = quartiles[2] - quartiles[0]
687
688        lower_bound = quartiles[0] - (1.5 * iqr)
689        upper_bound = quartiles[2] + (1.5 * iqr)
690
691        outliers = []
692        for i in range(len(self.data_set)):
693            if self.data_set[i] < lower_bound or self.data_set[i] > upper_bound:
694                outliers.append(self.data_set[i])
695
696        return outliers
697
698    def calculate_z_scores(self):
699        """Calculate z-scores of the data set.
700
701        Returns:
702            list: List of z-scores for each element in the data set.
703
704        """
705        if len(self.data_set) < 1:
706            raise ValueError("Data set is empty")
707
708        mean = self.calculate_mean()
709        std_deviation = self.calculate_std_deviation()
710
711        z_scores = []
712        for i in range(len(self.data_set)):
713            z_scores.append((self.data_set[i] - mean) / std_deviation)
714
715        return z_scores
716
717    def calculate_boxplot(self):
718        """Calculate boxplot of the data set.
719
719        Returns:
720            dict: Dictionary containing the boxplot statistics.
720
721        """
722        if len(self.data_set) < 5:
723            raise ValueError("Data set is empty or has less than five elements")
724
725        min_value = self.calculate_min()
726        max_value = self.calculate_max()
727
728        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
729
730        boxplot = {
731            "min": min_value,
732            "q1": quartiles[0],
733            "median": quartiles[1],
734            "q3": quartiles[2],
735            "max": max_value
736        }
737
738        return boxplot
739
740    def calculate_iqr(self):
741        """Calculate interquartile range of the data set.
742
743        Returns:
744            float: Interquartile range of the data set.
744
745        """
746        if len(self.data_set) < 5:
747            raise ValueError("Data set is empty or has less than five elements")
748
749        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
750
751        iqr = quartiles[2] - quartiles[0]
752
753        return iqr
754
755    def calculate_outliers(self):
756        """Calculate outliers of the data set.
757
758        Returns:
759            list: List of outliers in the data set.
759
760        """
761        if len(self.data_set) < 5:
762            raise ValueError("Data set is empty or has less than five elements")
763
764        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
765
766        iqr = quartiles[2] - quartiles[0]
767
768        lower_bound = quartiles[0] - (1.5 * iqr)
769        upper_bound = quartiles[2] + (1.5 * iqr)
770
771        outliers = []
772        for i in range(len(self.data_set)):
773            if self.data_set[i] < lower_bound or self.data_set[i] > upper_bound:
774                outliers.append(self.data_set[i])
775
776        return outliers
777
778    def calculate_z_scores(self):
779        """Calculate z-scores of the data set.
780
781        Returns:
782            list: List of z-scores for each element in the data set.
783
784        """
785        if len(self.data_set) < 1:
786            raise ValueError("Data set is empty")
787
788        mean = self.calculate_mean()
789        std_deviation = self.calculate_std_deviation()
790
791        z_scores = []
792        for i in range(len(self.data_set)):
793            z_scores.append((self.data_set[i] - mean) / std_deviation)
794
795        return z_scores
796
797    def calculate_boxplot(self):
798        """Calculate boxplot of the data set.
799
800        Returns:
800            dict: Dictionary containing the boxplot statistics.
801
802        """
803        if len(self.data_set) < 5:
804            raise ValueError("Data set is empty or has less than five elements")
805
806        min_value = self.calculate_min()
807        max_value = self.calculate_max()
808
809        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
810
811        boxplot = {
812            "min": min_value,
813            "q1": quartiles[0],
814            "median": quartiles[1],
815            "q3": quartiles[2],
816            "max": max_value
817        }
818
819        return boxplot
820
821    def calculate_iqr(self):
822        """Calculate interquartile range of the data set.
823
824        Returns:
825            float: Interquartile range of the data set.
825
826        """
827        if len(self.data_set) < 5:
828            raise ValueError("Data set is empty or has less than five elements")
829
830        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
831
832        iqr = quartiles[2] - quartiles[0]
833
834        return iqr
835
836    def calculate_outliers(self):
837        """Calculate outliers of the data set.
838
839        Returns:
840            list: List of outliers in the data set.
840
841        """
842        if len(self.data_set) < 5:
843            raise ValueError("Data set is empty or has less than five elements")
844
845        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
846
847        iqr = quartiles[2] - quartiles[0]
848
849        lower_bound = quartiles[0] - (1.5 * iqr)
850        upper_bound = quartiles[2] + (1.5 * iqr)
851
852        outliers = []
853        for i in range(len(self.data_set)):
854            if self.data_set[i] < lower_bound or self.data_set[i] > upper_bound:
855                outliers.append(self.data_set[i])
856
857        return outliers
858
859    def calculate_z_scores(self):
860        """Calculate z-scores of the data set.
861
862        Returns:
863            list: List of z-scores for each element in the data set.
864
865        """
866        if len(self.data_set) < 1:
867            raise ValueError("Data set is empty")
868
869        mean = self.calculate_mean()
870        std_deviation = self.calculate_std_deviation()
871
872        z_scores = []
873        for i in range(len(self.data_set)):
874            z_scores.append((self.data_set[i] - mean) / std_deviation)
875
876        return z_scores
877
878    def calculate_boxplot(self):
879        """Calculate boxplot of the data set.
880
880        Returns:
881            dict: Dictionary containing the boxplot statistics.
882
883        """
884        if len(self.data_set) < 5:
885            raise ValueError("Data set is empty or has less than five elements")
886
887        min_value = self.calculate_min()
888        max_value = self.calculate_max()
889
890        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
891
892        boxplot = {
893            "min": min_value,
894            "q1": quartiles[0],
895            "median": quartiles[1],
896            "q3": quartiles[2],
897            "max": max_value
898        }
899
900        return boxplot
901
902    def calculate_iqr(self):
903        """Calculate interquartile range of the data set.
904
905        Returns:
906            float: Interquartile range of the data set.
906
907        """
908        if len(self.data_set) < 5:
909            raise ValueError("Data set is empty or has less than five elements")
910
911        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
912
913        iqr = quartiles[2] - quartiles[0]
914
915        return iqr
916
917    def calculate_outliers(self):
918        """Calculate outliers of the data set.
919
920        Returns:
921            list: List of outliers in the data set.
921
922        """
923        if len(self.data_set) < 5:
924            raise ValueError("Data set is empty or has less than five elements")
925
926        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
927
928        iqr = quartiles[2] - quartiles[0]
929
930        lower_bound = quartiles[0] - (1.5 * iqr)
931        upper_bound = quartiles[2] + (1.5 * iqr)
932
933        outliers = []
934        for i in range(len(self.data_set)):
935            if self.data_set[i] < lower_bound or self.data_set[i] > upper_bound:
936                outliers.append(self.data_set[i])
937
938        return outliers
939
940    def calculate_z_scores(self):
941        """Calculate z-scores of the data set.
942
943        Returns:
944            list: List of z-scores for each element in the data set.
945
946        """
947        if len(self.data_set) < 1:
948            raise ValueError("Data set is empty")
949
950        mean = self.calculate_mean()
951        std_deviation = self.calculate_std_deviation()
952
953        z_scores = []
954        for i in range(len(self.data_set)):
955            z_scores.append((self.data_set[i] - mean) / std_deviation)
956
957        return z_scores
958
959    def calculate_boxplot(self):
960        """Calculate boxplot of the data set.
961
961        Returns:
962            dict: Dictionary containing the boxplot statistics.
963
964        """
965        if len(self.data_set) < 5:
966            raise ValueError("Data set is empty or has less than five elements")
967
968        min_value = self.calculate_min()
969        max_value = self.calculate_max()
970
971        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
972
973        boxplot = {
974            "min": min_value,
975            "q1": quartiles[0],
976            "median": quartiles[1],
977            "q3": quartiles[2],
978            "max": max_value
979        }
980
981        return boxplot
982
983    def calculate_iqr(self):
984        """Calculate interquartile range of the data set.
985
986        Returns:
987            float: Interquartile range of the data set.
987
988        """
989        if len(self.data_set) < 5:
990            raise ValueError("Data set is empty or has less than five elements")
991
992        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
993
994        iqr = quartiles[2] - quartiles[0]
995
996        return iqr
997
998    def calculate_outliers(self):
999        """Calculate outliers of the data set.
1000
1001        Returns:
1002            list: List of outliers in the data set.
1002
1003        """
1004        if len(self.data_set) < 5:
1005            raise ValueError("Data set is empty or has less than five elements")
1006
1007        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
1008
1009        iqr = quartiles[2] - quartiles[0]
1010
1011        lower_bound = quartiles[0] - (1.5 * iqr)
1012        upper_bound = quartiles[2] + (1.5 * iqr)
1013
1014        outliers = []
1015        for i in range(len(self.data_set)):
1016            if self.data_set[i] < lower_bound or self.data_set[i] > upper_bound:
1017                outliers.append(self.data_set[i])
1018
1019        return outliers
1020
1021    def calculate_z_scores(self):
1022        """Calculate z-scores of the data set.
1023
1024        Returns:
1025            list: List of z-scores for each element in the data set.
1026
1027        """
1028        if len(self.data_set) < 1:
1029            raise ValueError("Data set is empty")
1030
1031        mean = self.calculate_mean()
1032        std_deviation = self.calculate_std_deviation()
1033
1034        z_scores = []
1035        for i in range(len(self.data_set)):
1036            z_scores.append((self.data_set[i] - mean) / std_deviation)
1037
1038        return z_scores
1039
1040    def calculate_boxplot(self):
1041        """Calculate boxplot of the data set.
1042
1042        Returns:
1043            dict: Dictionary containing the boxplot statistics.
1044
1045        """
1046        if len(self.data_set) < 5:
1047            raise ValueError("Data set is empty or has less than five elements")
1048
1049        min_value = self.calculate_min()
1050        max_value = self.calculate_max()
1051
1052        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
1053
1054        boxplot = {
1055            "min": min_value,
1056            "q1": quartiles[0],
1057            "median": quartiles[1],
1058            "q3": quartiles[2],
1059            "max": max_value
1060        }
1061
1062        return boxplot
1063
1064    def calculate_iqr(self):
1065        """Calculate interquartile range of the data set.
1066
1066        Returns:
1067            float: Interquartile range of the data set.
1068
1069        """
1070        if len(self.data_set) < 5:
1071            raise ValueError("Data set is empty or has less than five elements")
1072
1073        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
1074
1075        iqr = quartiles[2] - quartiles[0]
1076
1077        return iqr
1078
1079    def calculate_outliers(self):
1080        """Calculate outliers of the data set.
1081
1081        Returns:
1082            list: List of outliers in the data set.
1082
1083        """
1084        if len(self.data_set) < 5:
1085            raise ValueError("Data set is empty or has less than five elements")
1086
1087        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
1088
1089        iqr = quartiles[2] - quartiles[0]
1090
1091        lower_bound = quartiles[0] - (1.5 * iqr)
1092        upper_bound = quartiles[2] + (1.5 * iqr)
1093
1094        outliers = []
1095        for i in range(len(self.data_set)):
1096            if self.data_set[i] < lower_bound or self.data_set[i] > upper_bound:
1097                outliers.append(self.data_set[i])
1098
1099        return outliers
1100
1101    def calculate_z_scores(self):
1102        """Calculate z-scores of the data set.
1103
1104        Returns:
1105            list: List of z-scores for each element in the data set.
1106
1107        """
1108        if len(self.data_set) < 1:
1109            raise ValueError("Data set is empty")
1110
1111        mean = self.calculate_mean()
1112        std_deviation = self.calculate_std_deviation()
1113
1114        z_scores = []
1115        for i in range(len(self.data_set)):
1116            z_scores.append((self.data_set[i] - mean) / std_deviation)
1117
1118        return z_scores
1119
1120    def calculate_boxplot(self):
1121        """Calculate boxplot of the data set.
1122
1122        Returns:
1123            dict: Dictionary containing the boxplot statistics.
1124
1125        """
1126        if len(self.data_set) < 5:
1127            raise ValueError("Data set is empty or has less than five elements")
1128
1129        min_value = self.calculate_min()
1130        max_value = self.calculate_max()
1131
1132        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
1133
1134        boxplot = {
1135            "min": min_value,
1136            "q1": quartiles[0],
1137            "median": quartiles[1],
1138            "q3": quartiles[2],
1139            "max": max_value
1140        }
1141
1142        return boxplot
1143
1144    def calculate_iqr(self):
1145        """Calculate interquartile range of the data set.
1146
1146        Returns:
1147            float: Interquartile range of the data set.
1148
1149        """
1150        if len(self.data_set) < 5:
1151            raise ValueError("Data set is empty or has less than five elements")
1152
1153        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
1154
1155        iqr = quartiles[2] - quartiles[0]
1156
1157        return iqr
1158
1159    def calculate_outliers(self):
1160        """Calculate outliers of the data set.
1161
1161        Returns:
1162            list: List of outliers in the data set.
1162
1163        """
1164        if len(self.data_set) < 5:
1165            raise ValueError("Data set is empty or has less than five elements")
1166
1167        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
1168
1169        iqr = quartiles[2] - quartiles[0]
1170
1171        lower_bound = quartiles[0] - (1.5 * iqr)
1172        upper_bound = quartiles[2] + (1.5 * iqr)
1173
1174        outliers = []
1175        for i in range(len(self.data_set)):
1176            if self.data_set[i] < lower_bound or self.data_set[i] > upper_bound:
1177                outliers.append(self.data_set[i])
1178
1179        return outliers
1180
1181    def calculate_z_scores(self):
1182        """Calculate z-scores of the data set.
1183
1184        Returns:
1185            list: List of z-scores for each element in the data set.
1186
1187        """
1188        if len(self.data_set) < 1:
1189            raise ValueError("Data set is empty")
1190
1191        mean = self.calculate_mean()
1192        std_deviation = self.calculate_std_deviation()
1193
1194        z_scores = []
1195        for i in range(len(self.data_set)):
1196            z_scores.append((self.data_set[i] - mean) / std_deviation)
1197
1198        return z_scores
1199
1200    def calculate_boxplot(self):
1201        """Calculate boxplot of the data set.
1202
1202        Returns:
1203            dict: Dictionary containing the boxplot statistics.
1204
1205        """
1206        if len(self.data_set) < 5:
1207            raise ValueError("Data set is empty or has less than five elements")
1208
1209        min_value = self.calculate_min()
1210        max_value = self.calculate_max()
1211
1212        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
1213
1214        boxplot = {
1215            "min": min_value,
1216            "q1": quartiles[0],
1217            "median": quartiles[1],
1218            "q3": quartiles[2],
1219            "max": max_value
1220        }
1221
1222        return boxplot
1223
1224    def calculate_iqr(self):
1225        """Calculate interquartile range of the data set.
1226
1226        Returns:
1227            float: Interquartile range of the data set.
1228
1229        """
1230        if len(self.data_set) < 5:
1231            raise ValueError("Data set is empty or has less than five elements")
1232
1233        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
1234
1235        iqr = quartiles[2] - quartiles[0]
1236
1237        return iqr
1238
1239    def calculate_outliers(self):
1240        """Calculate outliers of the data set.
1241
1241        Returns:
1242            list: List of outliers in the data set.
1242
1243        """
1244        if len(self.data_set) < 5:
1245            raise ValueError("Data set is empty or has less than five elements")
1246
1247        quartiles = self.calculate_percentile(25), self.calculate_percentile(75)
1248
1249        iqr = quartiles[2] - quartiles[0]
1250
1251        lower_bound = quartiles[0] - (1.5 * iqr)
1252        upper_bound = quartiles[2] + (1.5 * iqr)
1253
1254        outliers = []
1255        for i in range(len(self.data_set)):
1256            if self.data_set[i] < lower_bound or self.data_set[i] > upper_bound:
1257                outliers.append(self.data_set[i])
1258
1259        return outliers
1260
1261    def calculate_z_scores(self):
1262        """Calculate z-scores of the data set.
1263
1264        Returns:
1265            list: List of z-scores for each element in the data set.
1266
1267        """
1268        if len(self.data_set) < 1:
1269            raise ValueError("Data set is empty")
1270
1271        mean = self.calculate_mean()
1272        std_deviation = self.calculate_std_deviation()
1273
1274        z_scores = []
1275        for i in range(len(self.data_set)):
1276            z_scores.append((self.data_set[i] - mean) / std_deviation)
1277
1278        return z_scores
1279
1280    def calculate_boxplot(self):
1281        """Calculate boxplot of the data set.
1282
1282        Returns:
1283            dict: Dictionary containing the boxplot statistics.
1284
1285        """
1286        if len(self.data_set) < 5:
1287            raise ValueError("Data set is empty or has less than five elements")
1288
1289        min_value = self.calculate_min()
1290        max_value = self.calculate_max()
1291
1292        quartiles = self.calculate_percentile(25), self.calculate_percentile(50), self.calculate_percentile(75)
1293
1294        boxplot = {
1295            "min": min_value,
1296            "q1": quartiles[0],
1297            "median": quartiles[1],
1298            "q3": quartiles[2],
1299            "max": max_value
1300        }
1301
1302        return boxplot
1303
1304    def calculate_iqr(self):
1305        """Calculate interquartile range of the data set.
1306
1306        Returns:
1307            float: Interquartile range of the data set.
1308
1309        """
1310        if len(self.data_set) < 5:
1311            raise ValueError("Data set is empty or has
```

수동조작을 이용하여 측정한 각도를 얻고,

선험보기법을 사용하여 deviation값과 서보모터의 각도의 관계 함수를 도출함으로써 서보모터를 1도 단위의 각도로 움직일 수 있도록 했다.

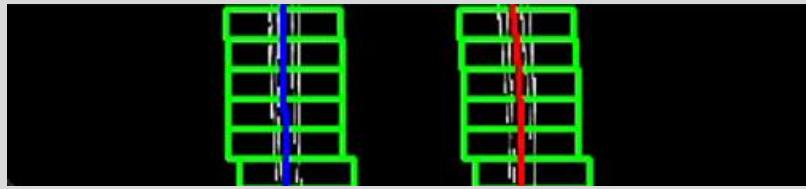
03 차선인식

`Line()` 클래스의 `is_dashed` 속성을 추가하여 검출한 차선이 실선인지 혹은 점선인지 판단했다. 이를 통해 차량 장애물의 개수와 위치에 상관없이 차선변경이 가능하게 구현했다.



점선

위: 오른쪽 점선 (비어있는 window 2개)
아래: 왼쪽 점선 (비어있는 window 3개)



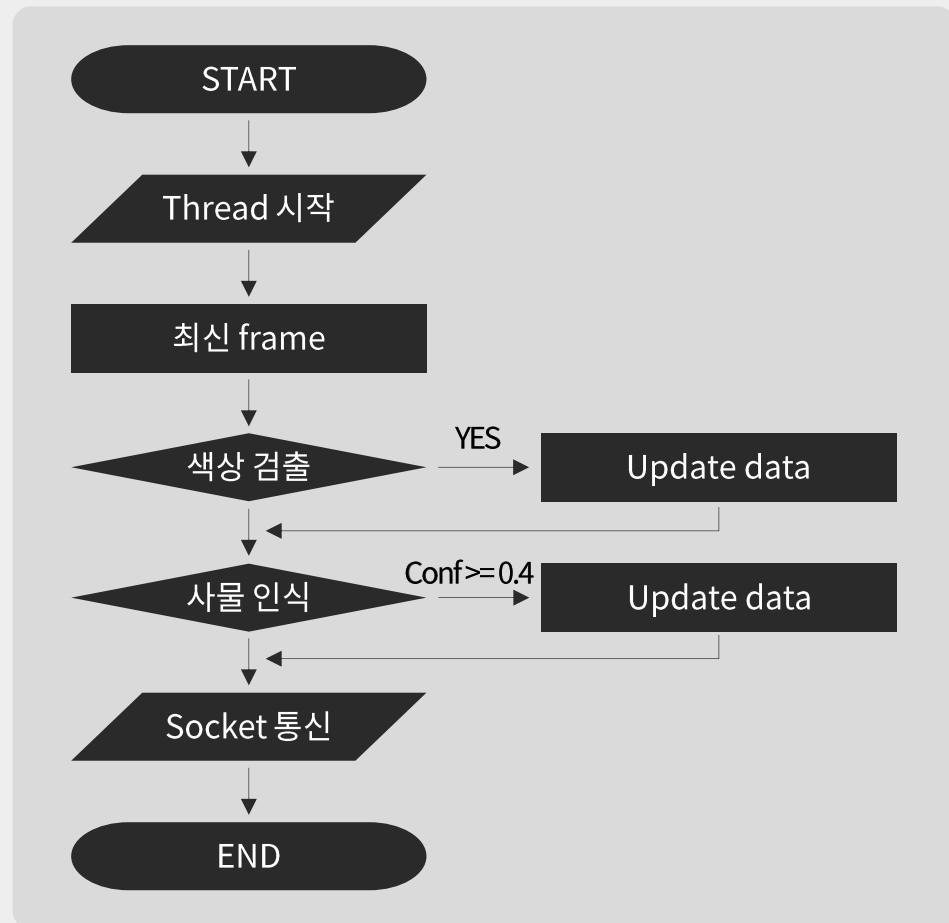
실선

비어있는window 0개

minpix보다 적은 수의 pixel이 검출되는 window의 수를 count, 총 6개 중 2개 이상의 window가 비어 있는 경우를 점선으로 간주한다.

03 사물인식

PC에서 작동하는 코드의 전체 흐름도

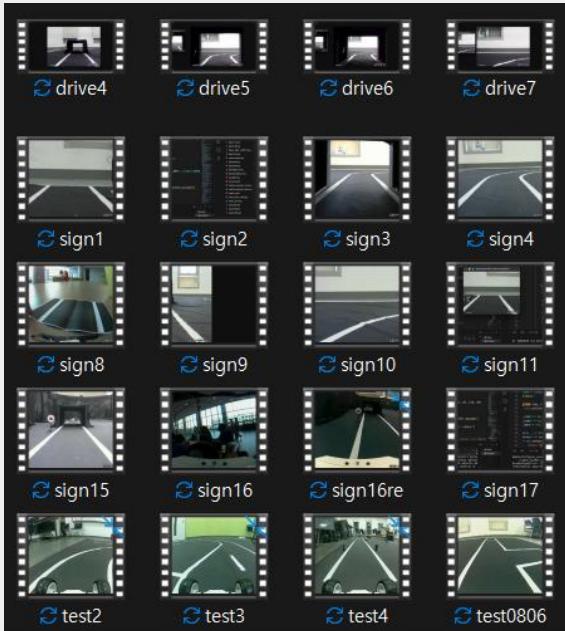


색상 검출을 진행한 후에 사물 인식 결과값을 덮어씌움으로써 사물인식으로 신호등을 검출 못할 경우를 대비하였다. 이는 초기 신호등에서 초록불의 조도가 약해 사물인식의 정확도가 낮았기 때문이다. 사물인식의 정확도가 0.4인 값만 유효하게 설정함으로써 정확도를 높였다. thread를 이용해서 비디오스트리밍과 객체 검출이 서로 방해되지 않고 동시에 수행할 수 있게 하여 딜레이를 최소화했다. video_stream() 함수가 독립된 쓰레드에서 실행되기 때문에, 메인 루프는 비디오스트리밍의 지연 없이 계속해서 객체 검출과 처리 작업을 수행할 수 있다.

```
23 def video_stream():
24     global frame
25     cap = cv2.VideoCapture('http://192.168.2.2:8081/') # 웹캠 URL을 설정
26     while True:
27         ret, latest_frame = cap.read()
28         if not ret:
29             break
30         frame = latest_frame # 최신 프레임 저장
31     cap.release()
32
33 > def find_green(img, size = 10): ...
34
35 > def find_red(img, size = 10): ...
36
37 # 비디오 스트리밍을 위한 쓰레드 시작
38 thread = threading.Thread(target=video_stream)
39 thread.start()
```

03 사물인식

연습주행 영상을 통한 학습데이터 수집



```
video_path = 'ref/12.mp4'
save_directory = 'dataset'

# 저장할 디렉토리가 없으면 생성
if not os.path.exists(save_directory):
    os.makedirs(save_directory)

# 비디오 파일 열어 챙기
cap = cv2.VideoCapture(video_path)
frame_count = 2100

while cap.isOpened():
    ret, frame = cap.read()
    frame = cv2.resize(frame, (320, 240), interpolation=cv2.INTER_AREA)

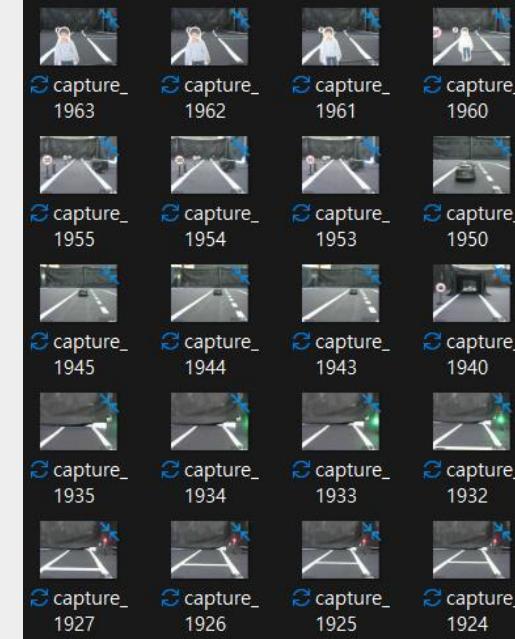
    if not ret:
        print("비디오를 불러오는 데 실패하거나 비디오가 끝났습니다.")
        break

    cv2.imshow('Video', frame)

    # 키 입력 대기
    key = cv2.waitKey(1) & 0xFF

    if key == ord('c'):
        frame_count += 1
        filename = f'{save_directory}/capture_{frame_count}.jpg'
        cv2.imwrite(filename, frame)
        print(f'캡처 파일: {filename}')

    elif key == ord('q'):
        break
```



영상을 캡처하는 코드를 별도로 작성해서 주행 영상을 frame 단위로 custom model을 위한 학습데이터화했다.

03 사물인식

영상편집을 활용한 차량데이터수집

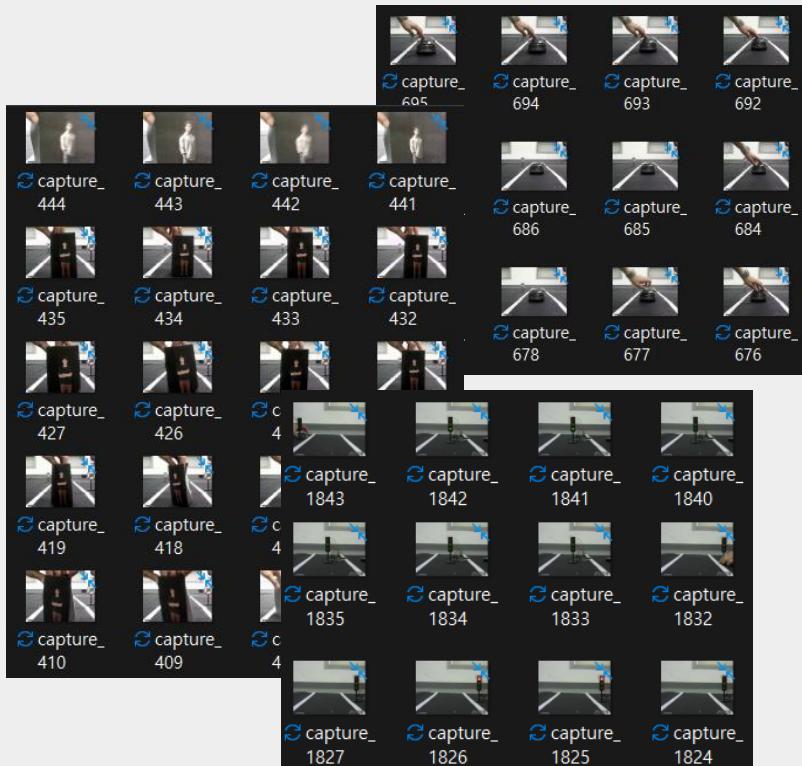


실제 환경에서 다양한 각도로 많은 차량데이터를 추가적으로 얻을 수 있었다.

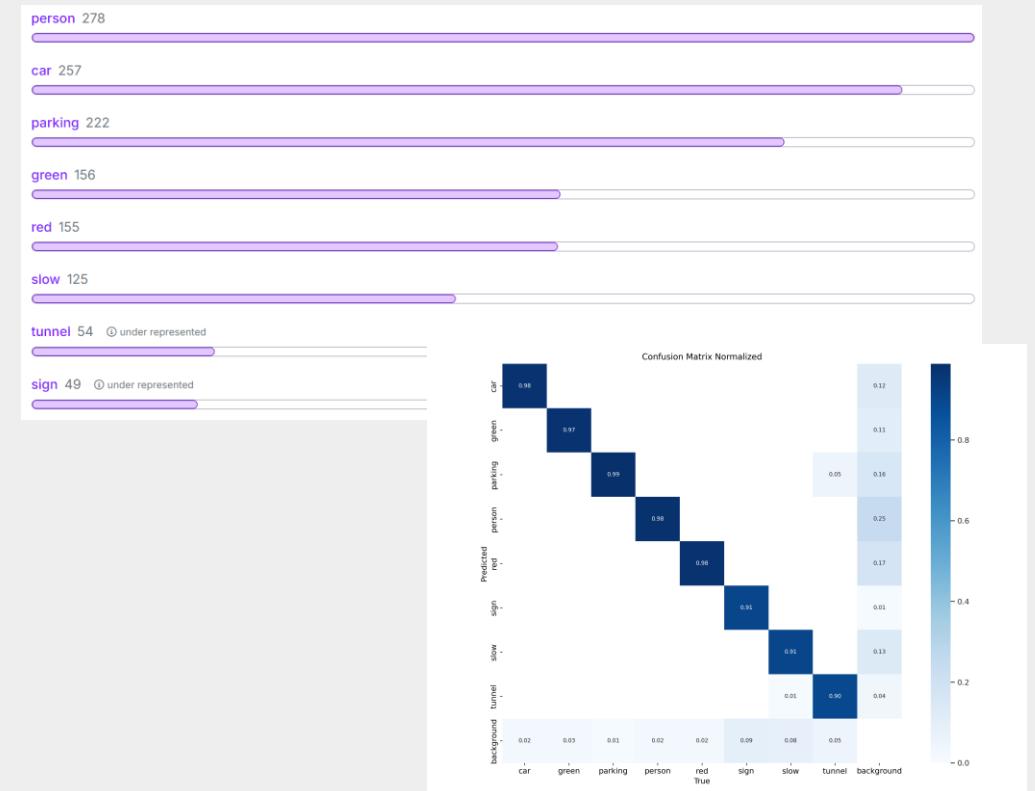
그 결과 차량을 잘 인식하여 차선변경을 잘 진행 시킬 수 있었다.

03 사물인식

주요 클래스들의 학습데이터 추가수집



사람 이미지 출력하여 각도별로 촬영, 차량모형을 각도별로 촬영, 신호등 위치별로 촬영하여 다양한 각도로 많은 데이터를 추가적으로 얻을 수 있었다.

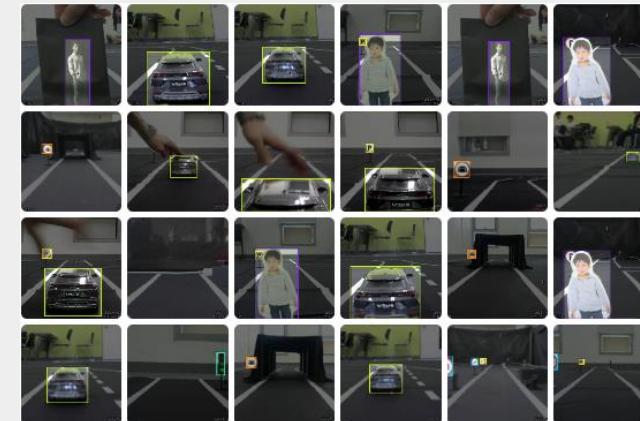


그 결과 주요 클래스인 person, car, green의 학습데이터를 충분히 확보할 수 있었고, 학습 결과 높은 정밀도와 재현율을 보였다.

03 사물인식

Roboflow에서 라벨링과 이미지 전처리 과정을 진행한 후, Google colab에서 YOLOv10 custom model 학습을 진행
차량이 진행할 때 진행 방향에 따라 장애물의 각도가 달리 보일 수 있기 때문에 horizontal shear를 추가하여 학습 데이터의 다양성을 높였다.

Preprocessing	Auto-Orient: Applied
	Resize: Stretch to 640x480
Augmentations	Outputs per training example: 3
	Shear: $\pm 15^\circ$ Horizontal, $\pm 0^\circ$ Vertical



```
results = model.train(data="/content/data.yaml", epochs=100, batch=64, patience=20, save_period=25, resume=True,  
                      flip_lr=0.0, project=drive_dir, iou=0.5, translate=0.4, name='yolov10n_training')
```

Colab 링크: https://colab.research.google.com/drive/1HrctzptCwxvjrN_cWCVcRGWTKBJpng#scrollTo=y-lhbe5CPwYV

03 사물인식

YOLOv10 custom model 학습에 사용한 하이퍼파라미터

박스 위치는 중요하지 않다는 점과 빠르게 움직이고 부분적으로 보이는 물체를 감지하는 것에 초점을 맞춰 값을 조정

Parameter	Default	Value	이유
epochs	100	100	100회 이상인 경우 성능이 미세하게 높았으나, 100회만으로 충분히 원하는 결과를 얻을 수 있었음
batch	16	64	Batch size가 낮을수록 인식률이 안 좋았음. 실험으로 얻은 최상의 결과값임
patience	100	20	epochs 숫자에 맞춰 값을 낮춰 설정함으로써 과적합을 방지함
save_period	-1	25	모델 체크포인트 저장 빈도를 설정하여 런타임 에러로 인한 학습중단에 대비함
resume	False	True	마지막으로 저장한 체크포인트부터 훈련을 재개하여 런타임 에러로 인한 학습중단에 대비함
fliplr	0.5	0.0	좌우반전된 이미지가 불필요하기 때문에 해당 기능을 제외함
iou	0.7	0.5	값이 낮을수록 중복되는 상자를 제거하여 탐지 횟수가 줄어들어 중복을 줄이는데 유용함
translate	0.1	0.4	부분적으로 보이는 물체를 감지하는 학습을 도와 근거리에 있는 사물 인식의 정확성을 높임

03 사물인식

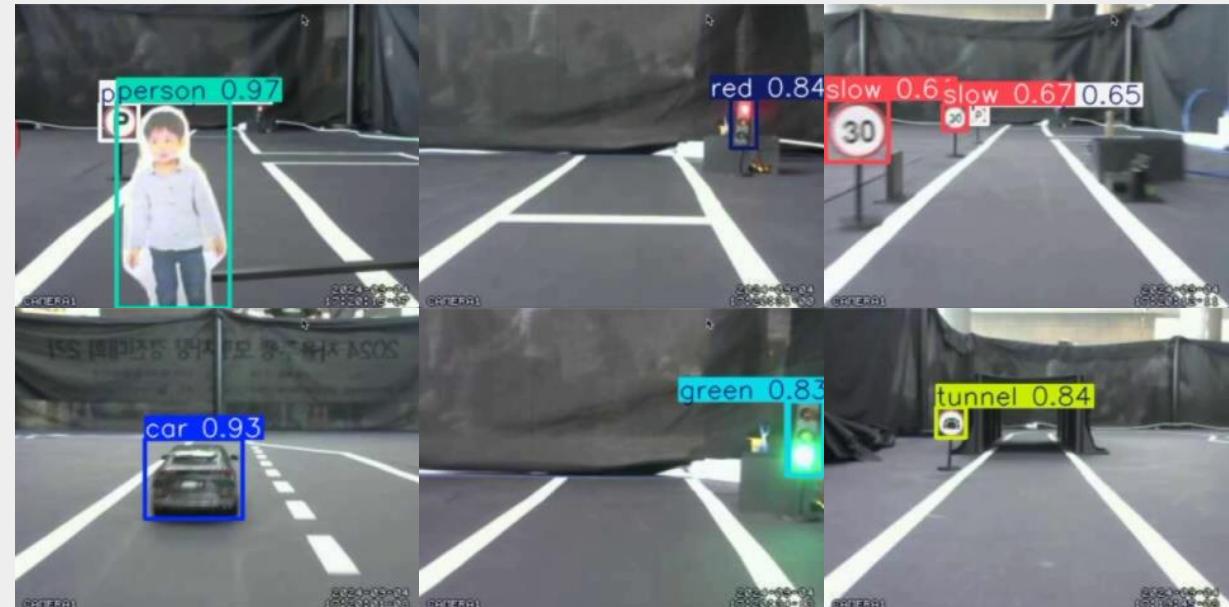
YOLOv10 custom model 학습로그

No.	dataset	hyperparameters	mAP50
1	주행영상+인터넷이미지	Epochs=50, batch=16	0.894
2	주행영상+인터넷이미지	Epochs=50, batch=64	0.762
3	주행영상+자체제작장애물	Epochs=100, batch=32	0.935
4	주행영상+자체제작장애물	Epochs=100, batch=64, iou=0.5, translate=0.4	0.967
5	주행영상+자체제작장애물	Epochs=200, batch=64, iou=0.5, translate=0.4	0.984
6	주행영상+자체제작장애물	Epoch=100, batch=64, iou=0.5, translate=0.4	0.970
7	주행영상+자체제작장애물	Epoch=100, batch=64, iou=0.5, translate=0.4, fliplr=0.0	0.979
8	주행영상+주최측장애물	Epoch=100, batch=64, iou=0.5, translate=0.4, fliplr=0.0	0.984

03 사물인식

YOLOv10 custom 학습모델을 활용한 사물인식 결과

```
while True:  
    # 버퍼를 클리어하여 최신 프레임만 읽음  
    for _ in range(15): # 15 프레임 정도를  
        ret, frame = cap.read()  
        if not ret:  
            break  
        frame = cv2.resize(frame, (320, 240),  
  
    # YOLOv8 모델로 프레임 예측  
    results = model(frame)  
  
    # 결과를 이미지에 그리기
```

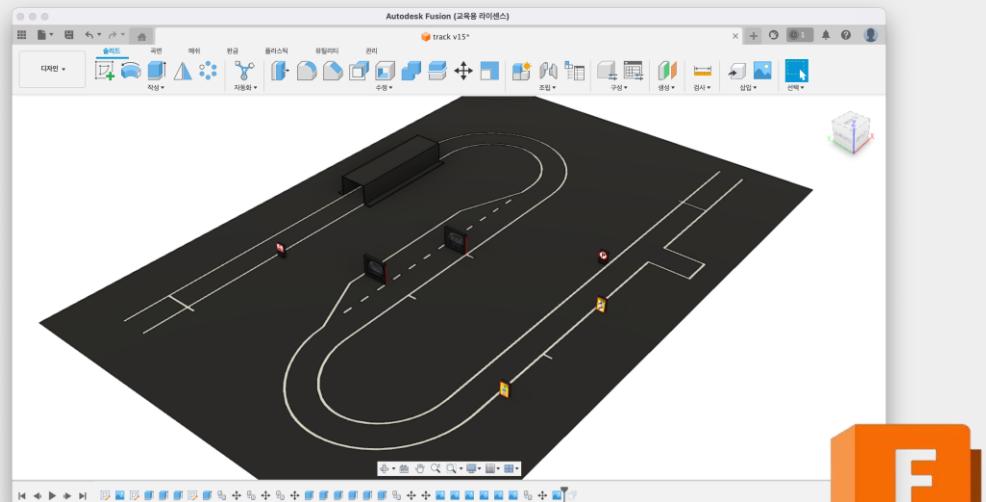


영상: <https://drive.google.com/file/d/1BilczDAHSzV0vkQL8g0XUI47ae7JcTJz/view?usp=sharing>

04 장애물 타파 방향성(연습용 트랙 설치 전)

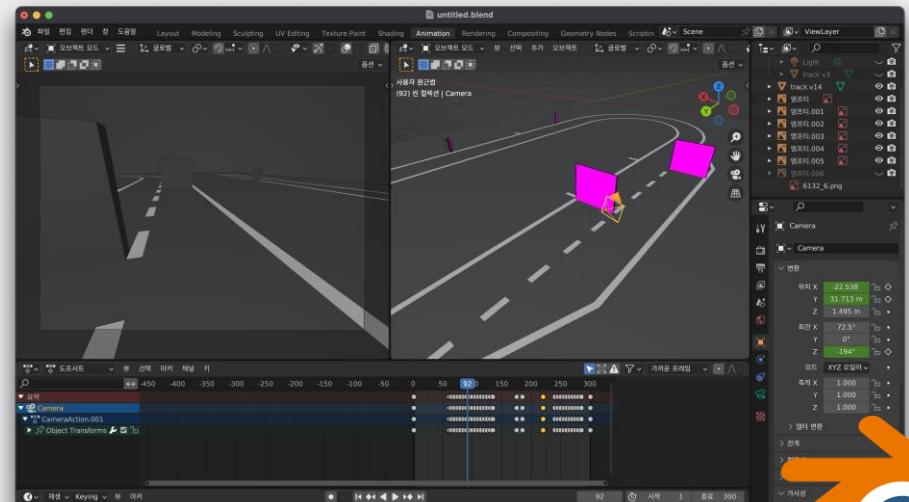
시뮬레이션

Fusion360과 Blender를 활용하여 실제주행과비슷한조건으로 시뮬레이션 제작



Fusion 360

경진대회의 주행트랙과장애물들을
3D모델링



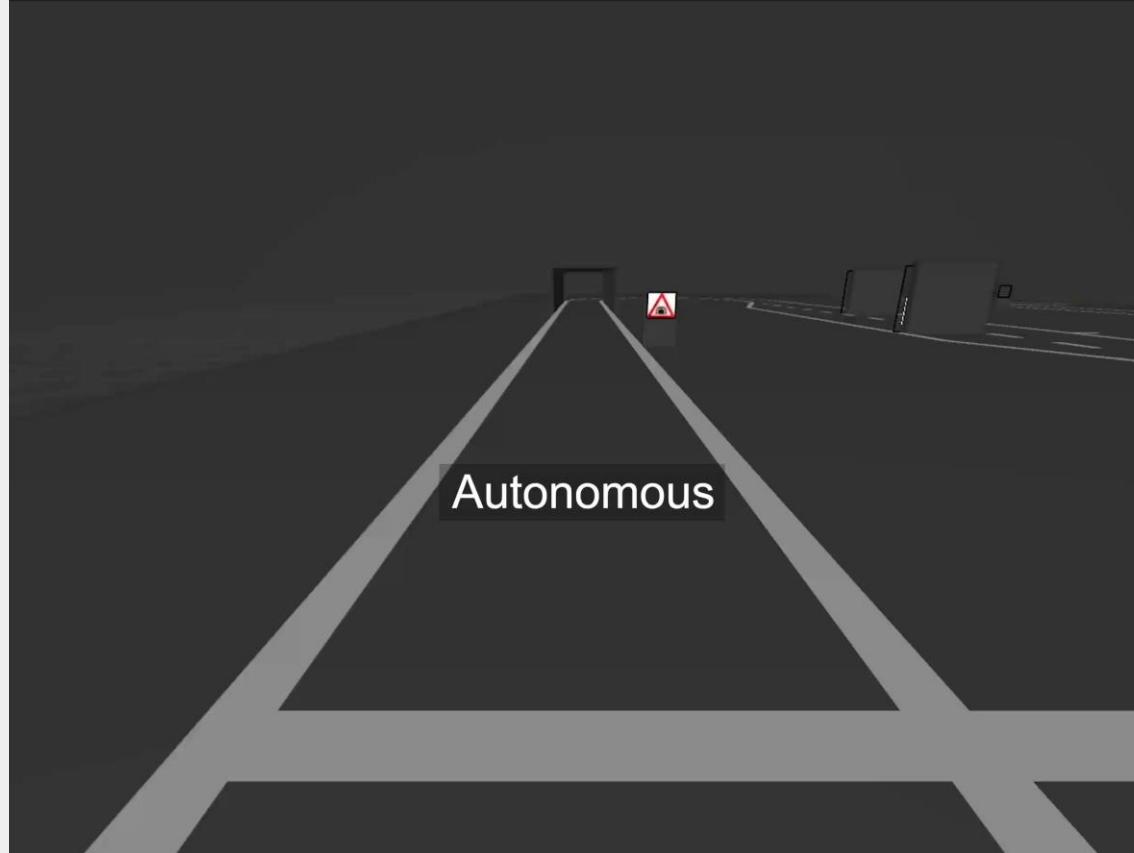
Blender

카메라의 실제 높이와 각도, 광각 등을
고려하여 시뮬레이션 영상 제작



04 장애물 타파 방향성(연습용 트랙 설치 전)

시뮬레이션



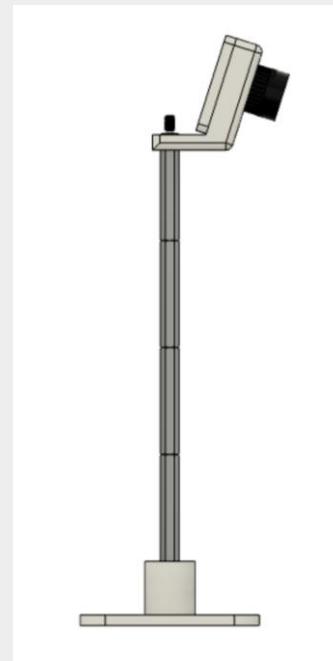
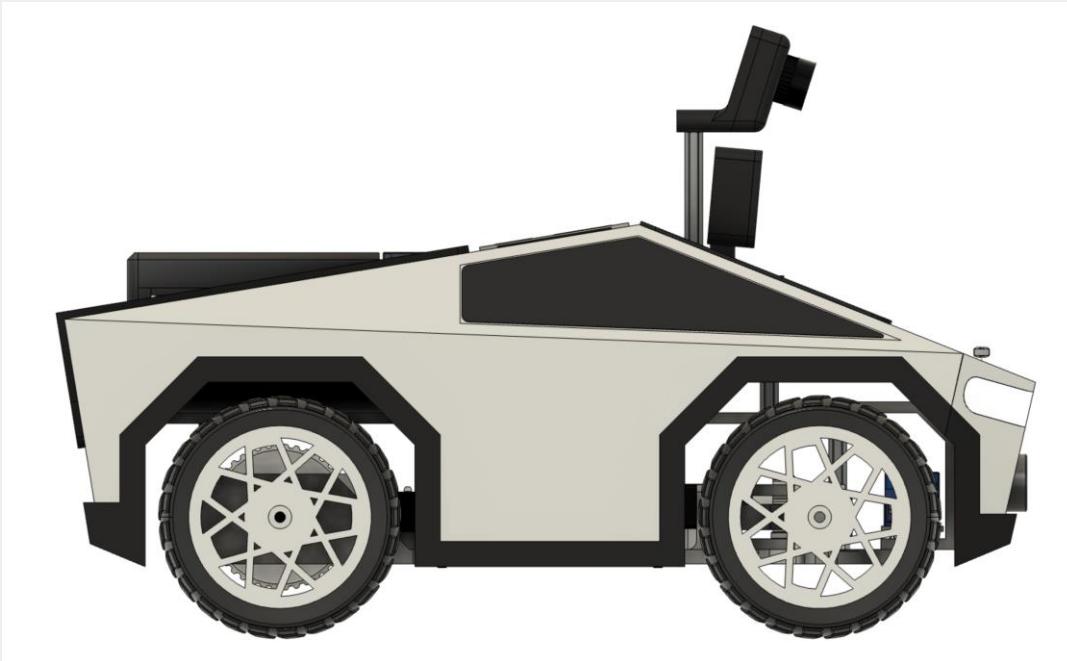
연습용 트랙 설치 전 위와 같이 실제 제작한 영상으로 장애물 타파 방향성을 구현

영상: <https://drive.google.com/file/d/1QKRWUfMPGsWvJDHYlrAPJjik7w9mvPv/view?usp=sharing>

Auto
nomous

04 장애물 탐지 방향성(차량 제작 전)

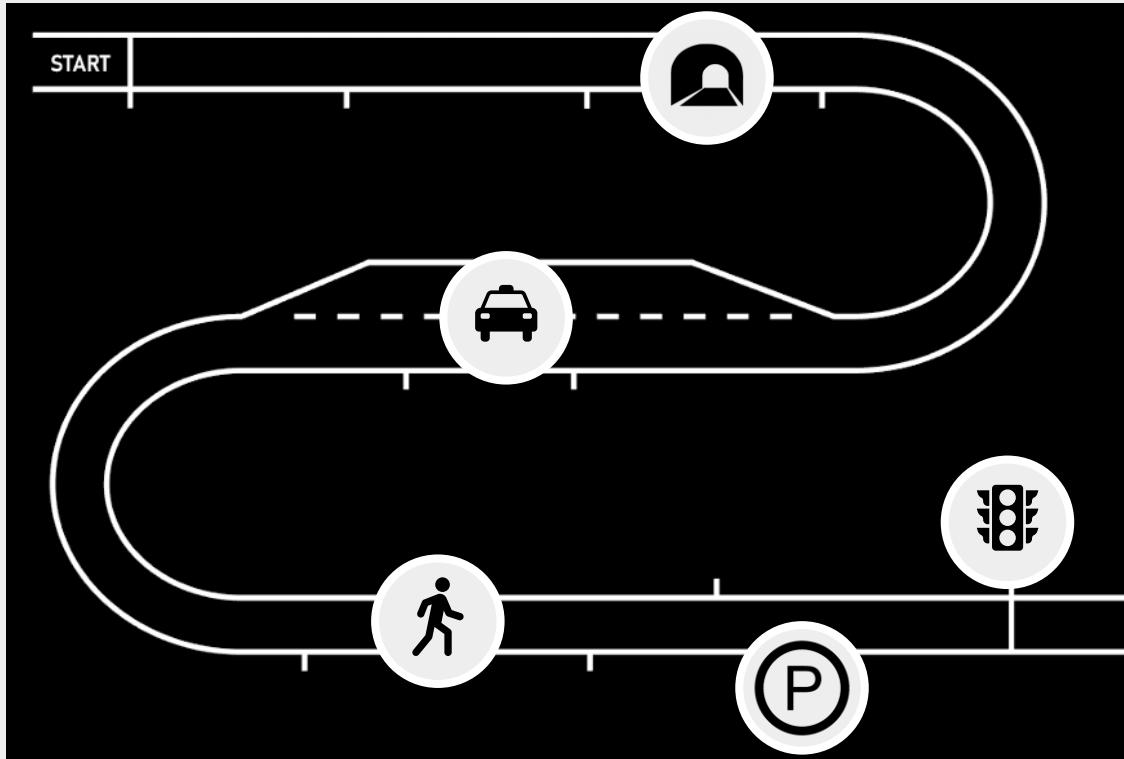
실제테스트



정확성을 높이기 위해 모델링한 차량을 바탕으로 스탠드형 카메라를
제작하여 실제 테스트를 진행

04 장애물 탐파 방향성

장애물 탐파 아이디어



터널

조도센서를 이용하여 어두워지면 LED 점등



정적장애물(차량)

사물인식(YOLO)과 초음파센서를 이용하여 동일 차선 내의 장애물을 인식하면 차선 변경



동적장애물(사람)

사물인식(YOLO)과 초음파센서를 이용하여 근거리의 사람이 인식되었을 때 긴급 정지



주차

주차선(수평 평행선)을 인식하면 주차 코드 실행

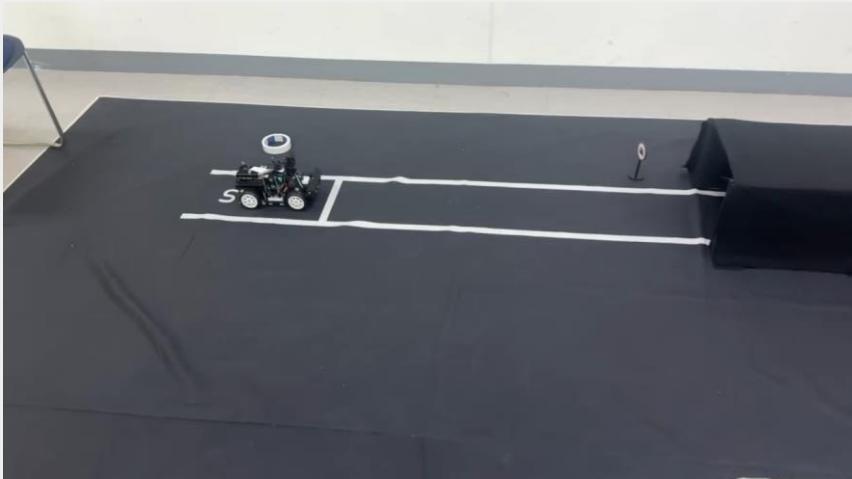


신호등

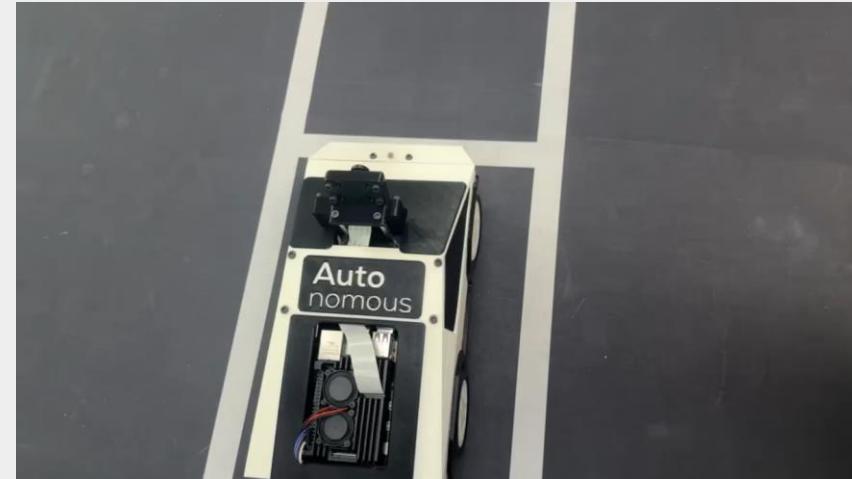
정지선 검출 후 정차, 사물인식(YOLO)과 OpenCV 색상 검출을 통해 초록불이면 진행

04 장애물 타파 방향성 - 터널

조도센서를 이용하여 어두워지면 LED 점등



전면부 모습



후면부 모습

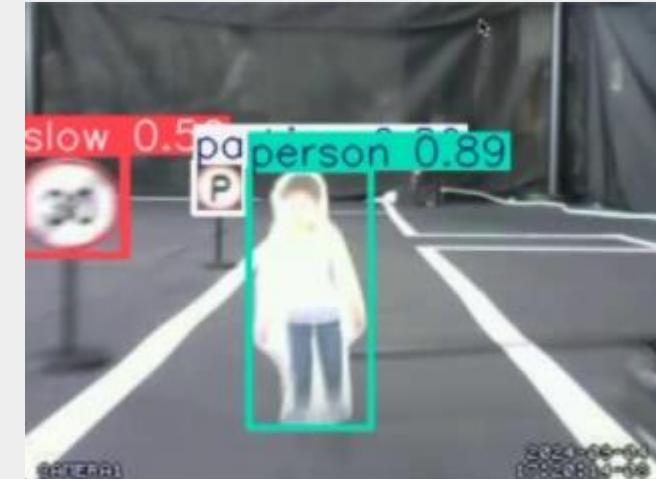
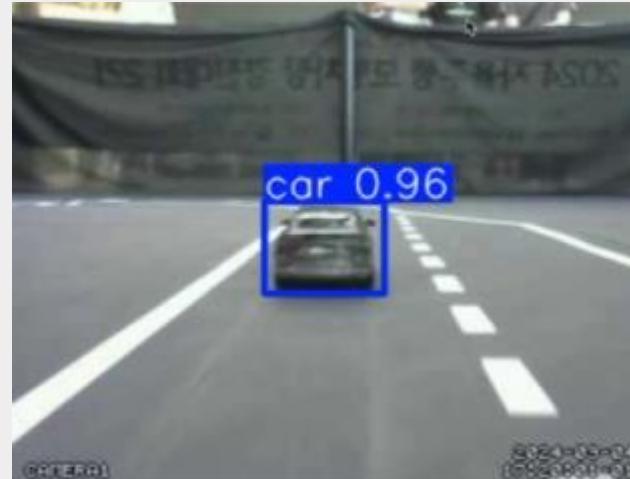
영상(좌) : <https://drive.google.com/file/d/1EwoJAxl3LC1ocNfb7VptGZVOvxzMP5Z5/view?usp=sharing>

영상(우) : <https://drive.google.com/file/d/1lz4IJESvCW45EQDqzXX1Ck8eEUjJinZi/view?usp=sharing>

04 장애물 탐파 방향성 – 차&사람

사물인식(YOLO)과 초음파센서를 이용하여 ‘차’일 경우, 동일 차선내의 장애물을 인식하면 차선변경&근거리의 사람이 인식되었을 때 긴급 정지

```
try:
    distance1 = measure_distance(14, 15) #전방 센서
    # print(distance1)
    if distance1 > 30:
        set_servo_angle(angle)
        set_motors_speed(speed)
    elif 5 < distance1 < 30:
        set_motors_speed(0)
    try:
        data = client_socket.recv(1024).decode()
        if data:
            print(f"Received command: {data}")
            if data == "car":
                if dir == 2:
                    print("go_right")
                    set_servo_angle(160)
                    set_motors_speed(speed)
                    time.sleep(0.3)
                elif dir == 1:
                    print("go_left")
                    set_servo_angle(15)
                    set_motors_speed(speed)
                    time.sleep(0.3)
                elif data == "person":
                    set_motors_speed(0)
                    time.sleep(1)
                    speed = 20
            except socket.error as e:
                print(f"Socket error: {e}")
                break
        else:
            set_motors_speed(0)
            time.sleep(1)
```



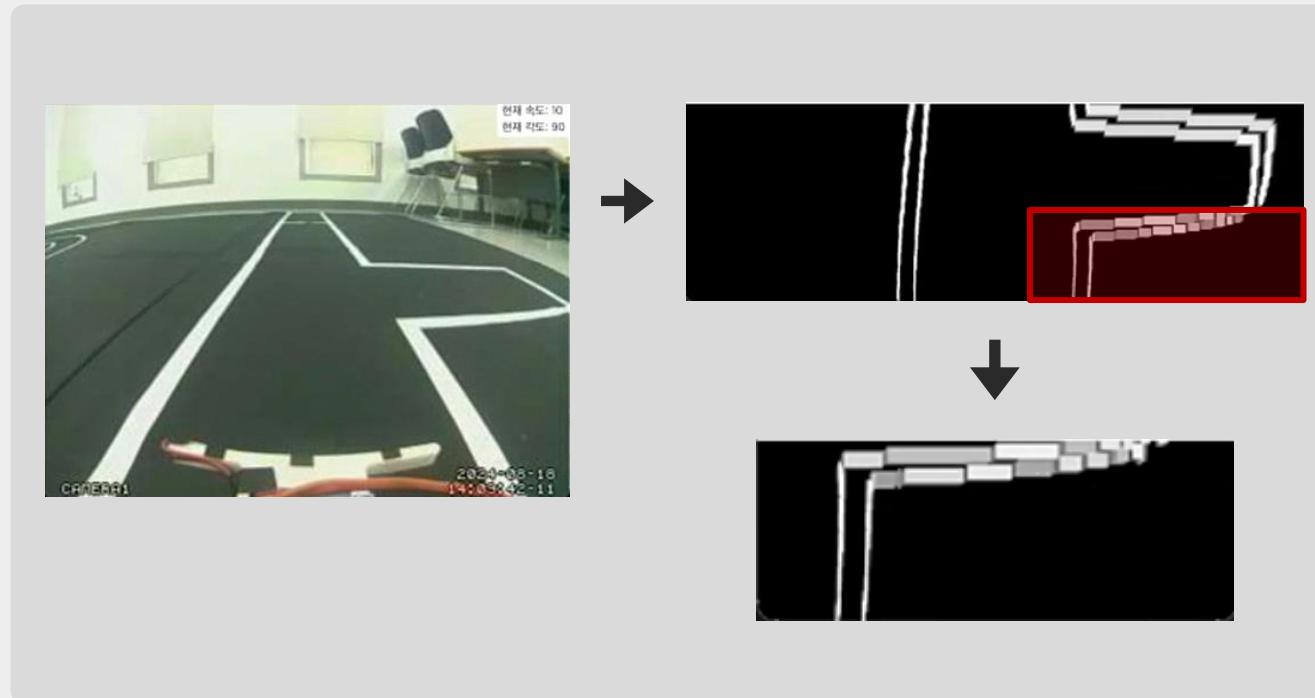
동적장애물의 경우 초음파센서로 어느 정도 거리를 인식하고 사물 인식 결과값을 받아온다.

그것이 ‘차’일 경우, 왼쪽과 오른쪽 중 어느 곳이 점선인지 파악한 다음 차선변경을 하는 코드를 작성했다.

‘사람’일 경우, 사람 검출 시 멈추는 코드 주차와 이후 주차선 검출의 정확도를 높이기 위해 속도를 낮추는 코드를 실행시킨다.

04 장애물 타파 방향성 - 주차

주차선(수평 평행선)을 인식하면 주차코드 실행



차선의 오른쪽 구간을 주차선 검출용 ROI로 설정하고,
HoughlinesP()를 이용하여 수평선을 감지하면 주차코드를 실행한다.

```
def detect_horizon(img): # 주차선 감지
    global Parking
    if Parking:
        return Parking

    # cv2.imshow('warped frame for parking', img)
    lines = cv2.HoughLinesP(img, 1, np.pi / 180, threshold=100, minLineLength=50, maxLineGap=10)

    if lines is not None:
        count = 0
        for line in lines:
            x1, y1, x2, y2 = line[0]
            angle = np.degrees(np.arctan2(y2 - y1, x2 - x1)) % 180

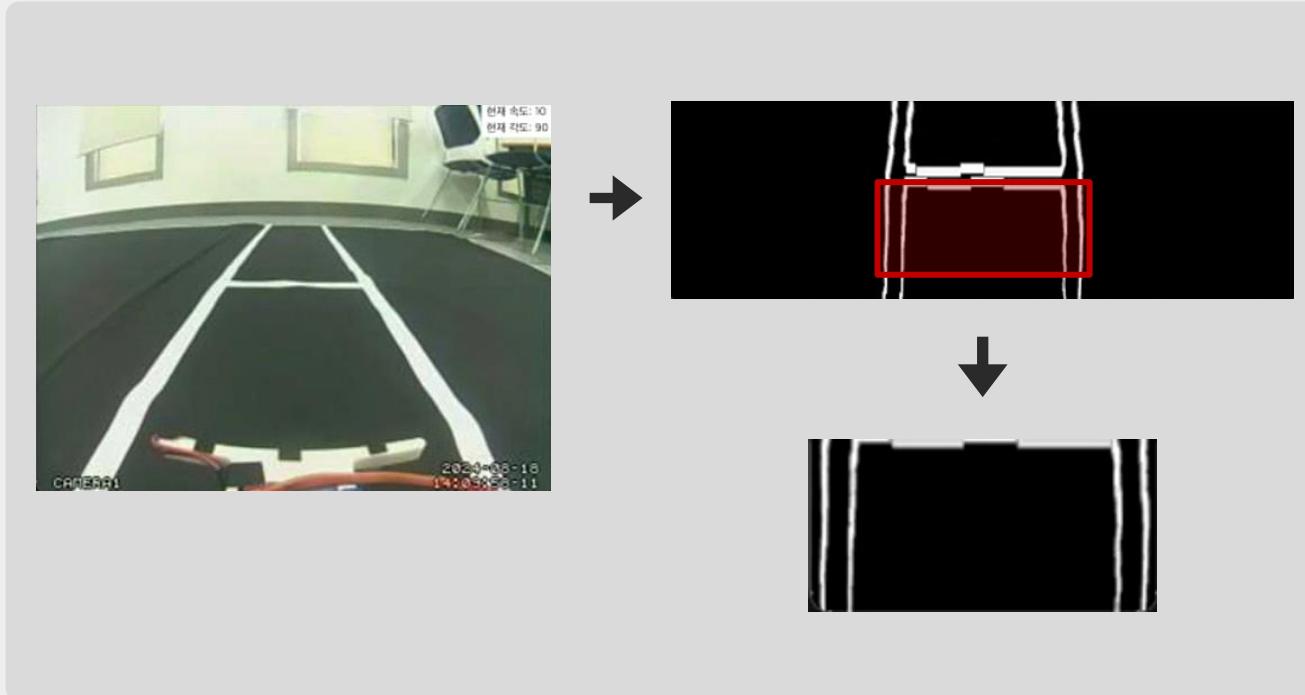
            if 170 <= angle or angle <= 10:
                count += 1

        if count >= 1:
            print("parkingline")
            parking_code()
            Parking = True

    return Parking
```

04 장애물 타파 방향성 - 신호등

정지선을 인식하면 잠시 멈춤



차선의 중앙부분을 정지선 검출용 ROI로 설정하고,
HoughlinesP()를 이용하여 수평선을 감지하면 일시 정지한다.

```
def detect_stopline(img): # 정지선 감지
    global Stop
    if Stop:
        return Stop

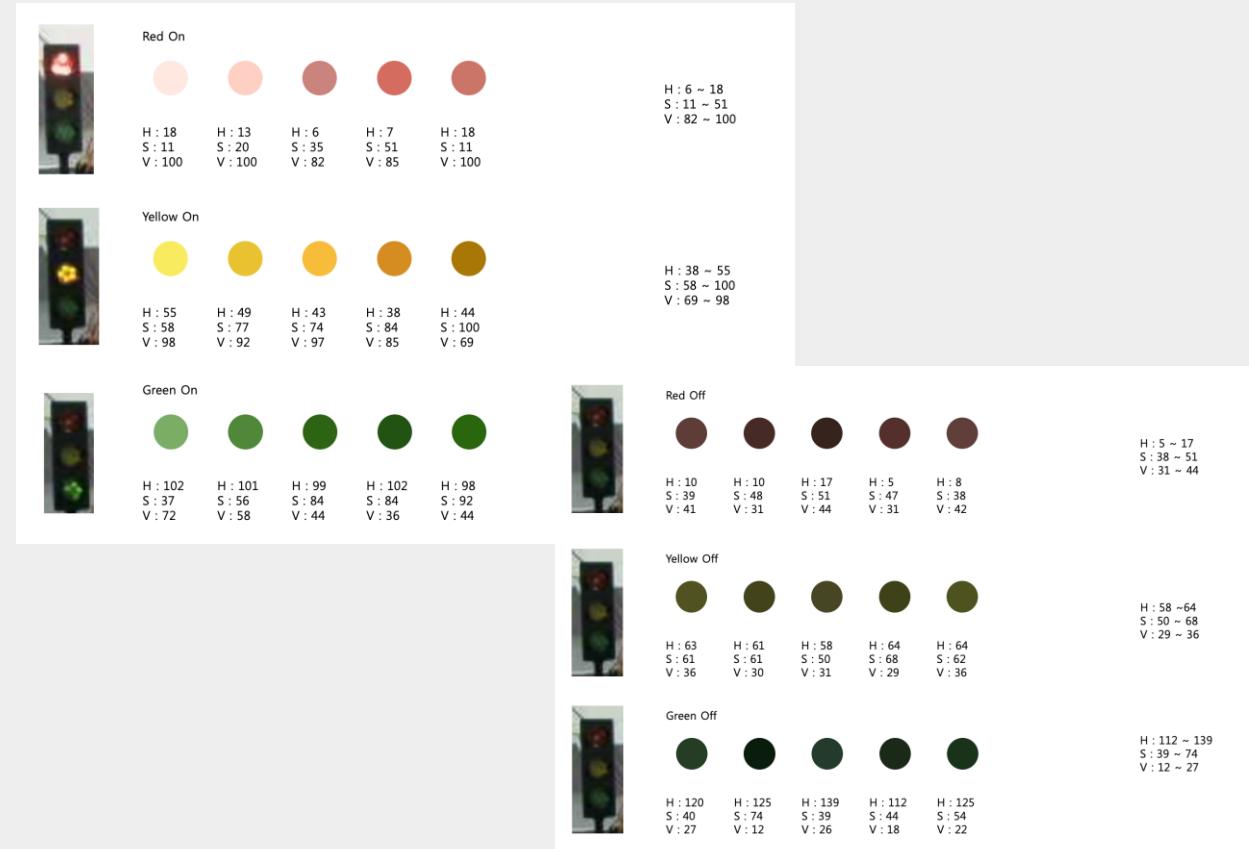
    # cv2.imshow('warped frame for stopline', img)
    lines = cv2.HoughLinesP(img, 1, np.pi / 180, threshold=100, minLineLength=30, maxLineGap=10)

    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line[0]
            angle = np.arctan2(y2 - y1, x2 - x1) * 180 / np.pi
            angle = angle if angle >= 0 else angle + 180
            if 170 <= angle or angle <= 10:
                set_motors_speed(0)
                print("stopline")
                time.sleep(1)
                Stop = True

    return Stop
```

04 장애물 타파 방향성 - 신호등

정지선검출후 정차, 사물인식(YOLO)과 OpenCV 색상검출을 통해 초록불이면 진행



신호등작동영상에서 사진을 캡쳐, 색상을 추출하여 그 범위에 있는 값을 인식하도록 설정

영상: <https://drive.google.com/file/d/1rpoOY6fafw0vga1dLsdxGICITDmCgGc0D/view?usp=sharing>

04 장애물 타파 방향성 - 신호등

신호등 추출 방법 - 색상인식

	Red				Yellow				Green			
	On		Off		On		Off		On		Off	
	low	high	low	high	low	high	low	high	low	high	low	high
H(360)	2	20	5	17	38	55	58	64	142	182	112	139
S(100)	19	52	38	51	58	100	50	68	5	47	39	74
V(100)	84	100	31	44	69	98	29	36	94	100	12	27
H(360)	112	360										
S(100)	0	6										
V(100)	90	100										
H(180)	1	10	2.5	8.5	19	27.5	29	32	71	91	56	69.5
S(255)	48.45	132.6	96.9	130.05	147.9	255	127.5	173.4	12.75	119.85	99.45	188.7
V(255)	214.2	255	79.05	112.2	175.95	249.9	73.95	91.8	239.7	255	30.6	68.85
H(180)	56	180										
S(255)	0	15.3										
V(255)	229.5	255										



```
32
33 def find_green(img, size = 10):
34     hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
35
36     green_lower = np.array([70, 10, 230], np.uint8)
37     green_upper = np.array([95, 120, 255], np.uint8)
38     green_mask = cv2.inRange(hsv, green_lower, green_upper)
39
40     contours, _ = cv2.findContours(green_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
41     for contour in contours:
42         if cv2.contourArea(contour) > size * size:
43             return "green"
44     return None
45
46 def find_red(img, size = 10):
47     hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
48
49     red_lower1 = np.array([0, 25, 150], np.uint8)
50     red_upper1 = np.array([10, 135, 255], np.uint8)
51     red_lower2 = np.array([170, 130, 150], np.uint8)
52     red_upper2 = np.array([180, 255, 255], np.uint8)
53     red_mask1 = cv2.inRange(hsv, red_lower1, red_upper1)
54     red_mask2 = cv2.inRange(hsv, red_lower2, red_upper2)
55     red_mask = cv2.bitwise_or(red_mask1, red_mask2)
56
57     contours, _ = cv2.findContours(red_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
58     for contour in contours:
59         if cv2.contourArea(contour) > size * size:
60             return "red"
61     return None
```

추출해 얻은 값을 변환하고 그 값을 바탕으로 코드를 작성
contourArea(contour)를 계산하는 과정을 추가하여 불필요한 검출을 예방했다.

04 장애물 탐지 방향성 - 신호등

신호등 추출 방법 - 사물인식

```
while True:  
    # 버퍼를 클리어하여 최신 프레임만 읽음  
    for _ in range(15): # 15 프레임 정도를  
        ret, frame = cap.read()  
        if not ret:  
            break  
    frame = cv2.resize(frame, (320, 240),  
  
    # YOLOv8 모델로 프레임 예측  
    results = model(frame)  
  
    # 결과를 시각화하기위해
```



YOLO custom 학습모델을 통해 초록불이 들어올 경우, “green”을 반환하고 일정 시간동안 직진

영상: <https://drive.google.com/file/d/1PJYiHaX7-K4VM-55ozGs7wRrHfJqW-qK/view?usp=sharing>

A u t o n o m o u s 설 계 발 표

THANK YOU

백재민 | 박준아

Auto
nomous

Auto
nomous