코일 선적 알고리즘 개발

(2023학년도 2학기 Logistics Lab 연구참여)

2024.02.26

지도 교수: 김병인

지도 선배: 임상운

참여 학생: 김예은

목차

1. 연구 내용

1.1 문제 설명

기존 코일 선적 스케줄 편성과 작업 지시는 수작업으로 진행되었기 때문에, 작업자의 숙련도에 따라서 작업 편차 및 효율의 하락이 발생하였다. 또한 향후 제품 선적 크레인 자동화 기술 개발을 위하여 스케줄링 자동화 선행 연구가 필요하였기 때문에, 코일 선적 알고리즘 개발이 필요하다.

코일 선적을 위해서는 코일의 중량, 외경, 두께, 폭 등을 고려한 다양한 제약조건을 만족해야 한다. 예를 들면 이웃한 코일 간의 크기 차이가 일정 수준이상 나면 안된다는 것이 있을 수 있다. 또한 제약조건 뿐만 아니라, 하중 Balance를 고려하여 안정적인 선적을 구현할 수 있도록 하여야 한다.

문제 해결을 위한 방법론으로는 수리모델과 휴리스틱 알고리즘이 있다. 수리모델은 실제 상황에서는 소요시간이 매우 길어지고 계산 불가 현상이 발생하여 적용이 어렵다. 따라서 문제 해결을 위해 휴리스틱 알고리즘이 사용되었다.

1.2 연구 참여

연구 참여를 시작하였을 때는 어느정도 알고리즘이 개발되고 있는 중이었다. 그 중 일부를 맡아서 해결해보았다.

1.2.1 코일 좌표 계산 및 시각화

기존 진행 상황에서는 각 열과 단에 선적해야 하는 코일은 알고리즘을 통해 결정되었지만, 정확한 위치는 결정되지 않았다. 1단의 코일은 바닥에 붙어 있으므로 y값은 코일의 반지름과 같고, x값은 이전 코일과 접하도록 설정하면 된다. 그러나 2단과 3단의 코일은 각각 하단의 코일 2개와 접해야 하므로 좌표 계산에 어려움이 있다. sympy의 symbols, Eq, solve를 이용하여 코일의 x, y 좌표를 계산하였다. 코드는 아래와 같다.

```
[ ]  def equation(x1, y1, x2, y2, r1, r2, r3):
        x, y = symbols('x y')

        equation1 = Eq((x2-x)**2 + (y2-y)**2, (r2+r3)**2)
        equation2 = Eq((x1-x)**2 + (y1-y)**2, (r1+r3)**2)

        solutions = solve((equation1, equation2), (x, y))

        real_solutions = [(x, y) for x, y in solutions if x.is_real and y.is_real]
        filtered_solutions = [(x, y) for x, y in real_solutions if x > 0 and y > 0]
        max_y_solution = max(filtered_solutions, key=lambda solution: solution[1])

        return max_y_solution[0], max_y_solution[1]
```

```
### y, z 좌표를 만드는 코드 ###

    # 결과 copy
    result_y = result.copy()
    result_z = result.copy()

    # y좌표 삽입
    for r in tqdm(range(2)):#result_y.shape[0]), desc="Outer Loop"):
        y_co = 0
        b = np.where(result[r][0] == -1)[0][0]

        # 1단 y
        for i in range(b):
            if i == 0:
                result_y[r][0][i] = coil_dia[int(result[r][0][i])]/2 #맞음
                y_co += result_y[r][0][i]
            else:
                if result[r][0][i] != -1 and result[r][1][i-1] not in key_coil_result_new2:
                    y_co += math.sqrt((coil_dia[int(result[r][0][i])]/2 + coil_dia[int(result[r][0][i-1])]/2)**2
                                      - (coil_dia[int(result[r][0][i])]/2 - coil_dia[int(result[r][0][i-1])]/2)**2)
                    result_y[r][0][i] = y_co
                elif result[r][0][i] != -1 and result[r][1][i-1] in key_coil_result_new2:
                    y_co += math.sqrt((coil_dia[int(result[r][0][i])]/2 + coil_dia[int(result[r][1][i-1])]/2)**2
                                      + (coil_dia[int(result[r][0][i-1])]/2 + coil_dia[int(result[r][1][i-1])]/2)**2
                                      - 2*(coil_dia[int(result[r][0][i])]/2 + coil_dia[int(result[r][1][i-1])]/2)
                                      *(coil_dia[int(result[r][0][i-1])]/2 + coil_dia[int(result[r][1][i-1])]/2)*cosine_value
                                      -(coil_dia[int(result[r][0][i])]/2-coil_dia[int(result[r][0][i-1])]/2)**2)
                    result_y[r][0][i] = y_co
                else:
                    continue

        # 1단 z
        for i in range(b):
            result_z[r][0][i] = coil_dia[int(result[r][0][i])]/2

        # 2단
        for j in tqdm(range(b-1)):
            if result_y[r][1][j] != -1:
                y_value, z_value = equation(result_y[r][0][j], result_z[r][0][j], result_y[r][0][j+1], result_z[r][0][j+1],
                                            coil_dia[int(result[r][0][j])]/2, coil_dia[int(result[r][0][j+1])]/2, coil_dia[int(result[r][1][j])]/2)
                result_y[r][1][j] = y_value
                result_z[r][1][j] = z_value
            else:
                continue

        # 3단
        for k in tqdm(range(b-2)):
            if result_y[r][2][k] != -1:
                y_value, z_value = equation(result_y[r][1][k], result_z[r][1][k], result_y[r][1][k+1], result_z[r][1][k+1],
                                            coil_dia[int(result[r][1][k])]/2, coil_dia[int(result[r][1][k+1])]/2, coil_dia[int(result[r][2][k])]/2)
                result_y[r][2][k] = y_value
                result_z[r][2][k] = z_value
            else:
                continue
```

[그림1 코일 좌표 계산 코드 1]

그러나 sympy의 symbols, Eq, solve를 이용하여 좌표를 계산하는 방법은 계산 속도가 매우 느리다는 치명적인 단점이 있었다. 그래서 계산 속도를 향상시키기 위해 scipy.optimize의 fsolve를 사용하여 좌표를 계산하는 방법으로 수정하였다. 처음부터 fsolve를 사용하지 않았던 이유는 fsolve를 사용할 경우 초기 좌표 값을 설정해주어야 하고 초기 좌표 값에 따라 다른 결과를 얻을 수 있다는 점 때문이다. 실제로 방정식을 풀 경우 해는 2쌍이 나오게 되는데, sybols, Eq, solve를 사용할 때에는 y 값이 더 큰 경우를 해로 선택하였다. 초기 해로 x값은 하단의 두 코일 x값 평균, y 값은

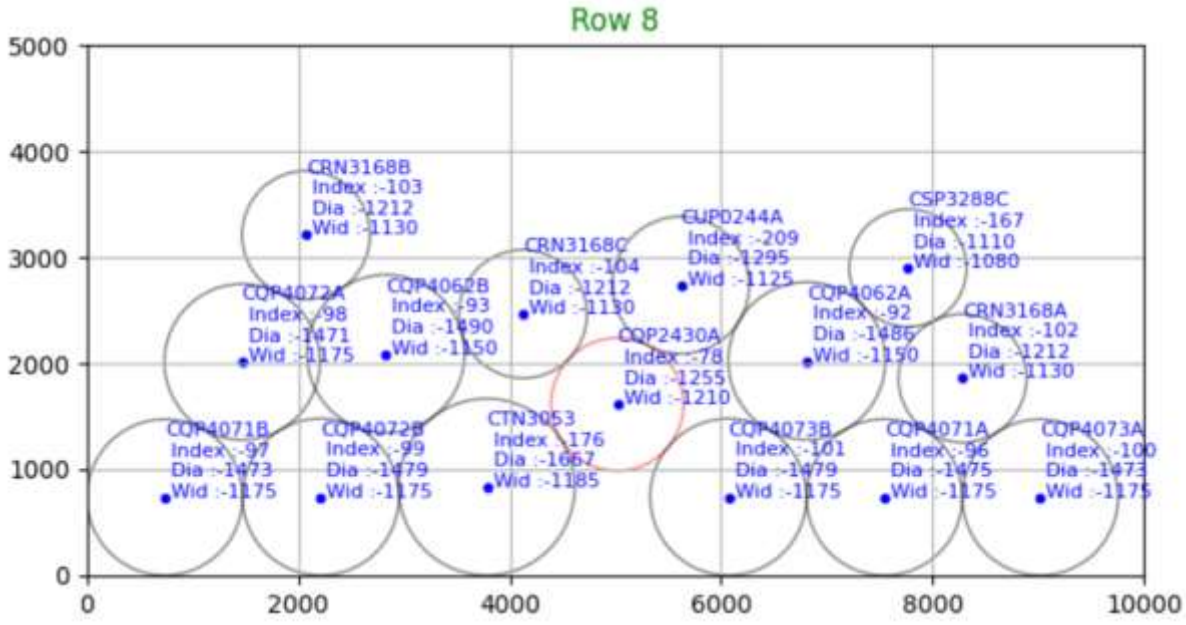
하단의 두 코일 y값 평균에 새로운 코일의 지름을 더한 값을 사용하였다. 초기값을 설정한 덕분에 1시간이 걸리던 계산이 1초 미만으로 줄어들었다. 코드는 아래와 같다.

```
[ ] def equation(vars, x1, y1, x2, y2, r1, r2, r3):
        x, y = vars
        eq1 = (x - x1)**2 + (y - y1)**2 - (r1 + r3)**2
        eq2 = (x - x2)**2 + (y - y2)**2 - (r2 + r3)**2
        return [eq1, eq2]
```

```
### y, z 좌표를 만드는 코드 ###

# 결과 copy
result_y = result.copy()
result_z = result.copy()

# y좌표 설정
for r in tqdm(range(20)):#result_y.shape[0]):
    y_co = 0
    b = np.where(result[r][0] == -1)[0][0]

    # 1단 y
    for i in range(b):
        if i == 0:
            result_y[r][0][i] = coil_dia[int(result[r][0][i])]/2 #반올
            y_co += result_y[r][0][i]
        else:
            if result[r][0][i] != -1 and result[r][1][i-1] not in key_coil_result_new2:
                y_co += math.sqrt((coil_dia[int(result[r][0][i])]/2 + coil_dia[int(result[r][0][i-1])]/2)**2
                                  - (coil_dia[int(result[r][0][i])]/2 - coil_dia[int(result[r][0][i-1])]/2)**2)
                result_y[r][0][i] = y_co
            elif result[r][0][i] != -1 and result[r][1][i-1] in key_coil_result_new2:
                y_co += math.sqrt((coil_dia[int(result[r][0][i])]/2 + coil_dia[int(result[r][1][i-1])]/2)**2
                                  + (coil_dia[int(result[r][0][i-1])]/2 + coil_dia[int(result[r][1][i-1])]/2)**2
                                  - 2*(coil_dia[int(result[r][0][i-1])]/2 + coil_dia[int(result[r][1][i-1])]/2)
                                  *(coil_dia[int(result[r][0][i-1])]/2 + coil_dia[int(result[r][1][i-1])]/2)*cosine_value
                                  -(coil_dia[int(result[r][0][i])]/2-coil_dia[int(result[r][0][i-1])]/2)**2)
                result_y[r][0][i] = y_co
            else:
                continue

    # 1단 z
    for i in range(b):
        result_z[r][0][i] = coil_dia[int(result[r][0][i])]/2

    # 2단
    for j in range(b-1):
        if result_y[r][1][j] != -1:
            y_value, z_value = fsolve(equation, [(result_y[r][0][j] + result_y[r][0][j+1]) / 2, (result_z[r][0][j] + result_z[r][0][j+1]) / 2 + coil_dia[int(result[r][1][j])]],
                                      args = (result_y[r][0][j], result_z[r][0][j], result_y[r][0][j+1], result_z[r][0][j+1],
                                              coil_dia[int(result[r][0][j])]/2, coil_dia[int(result[r][0][j+1])]/2, coil_dia[int(result[r][1][j])]/2))
            result_y[r][1][j] = y_value
            result_z[r][1][j] = z_value
        else:
            continue

    # 3단
    for k in range(b-2):
        if result_y[r][2][k] != -1:
            y_value, z_value = fsolve(equation, [(result_y[r][1][k] + result_y[r][1][k+1]) / 2, (result_z[r][1][k] + result_z[r][1][k+1]) / 2 + coil_dia[int(result[r][2][k])]],
                                      args = (result_y[r][1][k], result_z[r][1][k], result_y[r][1][k+1], result_z[r][1][k+1],
                                              coil_dia[int(result[r][1][k])]/2, coil_dia[int(result[r][1][k+1])]/2, coil_dia[int(result[r][2][k])]/2))
            result_y[r][2][k] = y_value
            result_z[r][2][k] = z_value
        else:
            continue
```

[그림2 코일 좌표 계산 코드 2]

시각화 결과는 아래와 같다. 옆에서 보는 평면도 외에도 위에서 내려다 보는 시각화 결과 및 3D 시각화 등 다양한 시각화 자료를 만들었다.



[그림3 코일 시각화 결과]

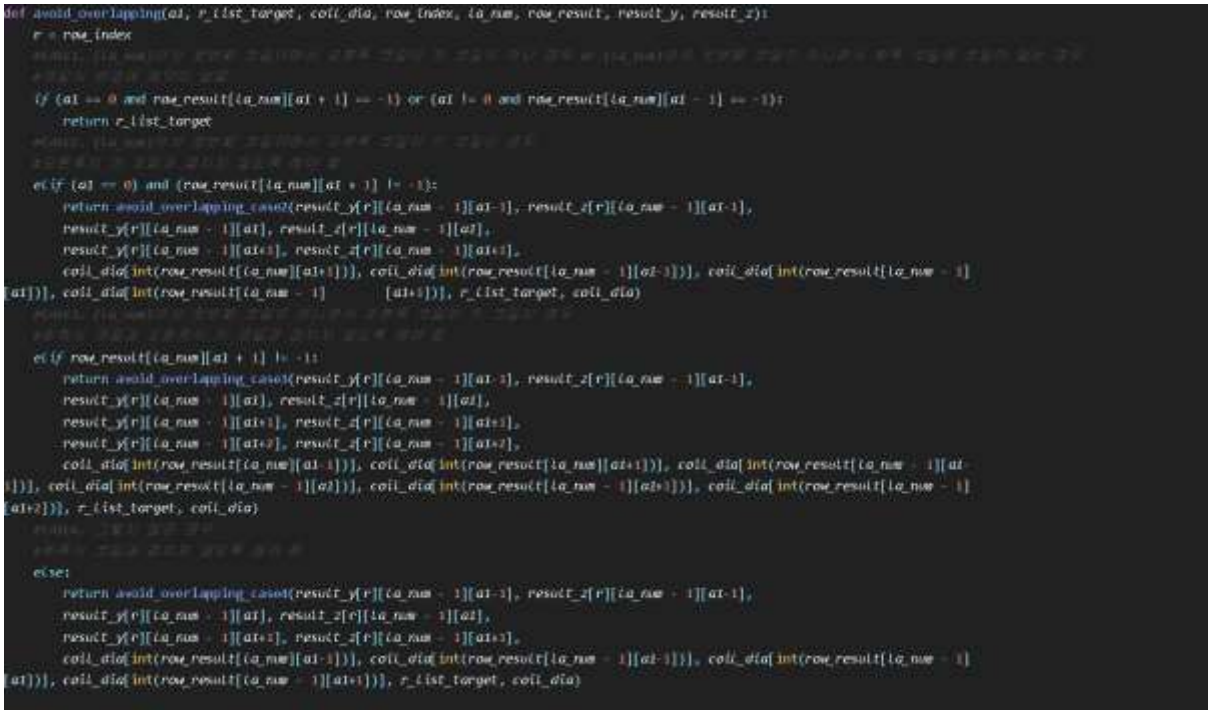
1.2.2 코일 겹침 방지

1.2.1에서 정확한 좌표를 계산하여 시각화한 결과, 새로운 문제를 발견하였다. 그림3의 2단 첫번째와 두번째 코일에서 볼 수 있듯이 두 코일이 겹쳐진다는 것이다. 따라서 이 문제를 해결하기 위해, 주변 코일 외경을 고려하여 겹치지 않는 코일 리스트를 반환하는 함수를 만들고, 해당 함수를 기존 코일 선적 알고리즘에 추가로 사용하였다. 지도 선배와 논의한 결과 좌표를 계산하는 대신 주변 코일들의 반지름만을 이용하여 코일의 겹침을 확인하는 방법이 시간적인 효율적일 수 있다는 의견이 있었다. 따라서 주변 코일들의 반지름 만을 이용하여 2단에 새로 놓일 코일의 겹침을 확인할 수 있는 함수를 만들었다.

```python
def avoid_overlapping(a1, r_list_target, row_result, kc_set_i, la_num):
  if a1 == 0 or row_result[la_num][a1-1] == kc_set_i or row_result[la_num][a1-1] == -1:
      return r_list_target

  avoid_r_list_target = []

  r1 = coil_dia[int(row_result[la_num-1][a1-1])]
  r2 = coil_dia[int(row_result[la_num-1][a1])]
  r3 = coil_dia[int(row_result[la_num-1][a1+1])]
  r4 = coil_dia[int(row_result[la_num][a1-1])]

  angle1 = math.degrees(math.acos(((r1+r2)**2+(r2+r3)**2-4*(math.sqrt(r1*r2)+math.sqrt(r2*r3))**2)/(2*(r1+r2)*(r2+r3))))
  if r2*2 < r1+r3:
      angle1 = 360 - angle1
  angle2 = math.degrees(math.acos(((r1+r2)**2+(r2+r4)**2-(r1+r4)**2)/(2*(r1+r2)*(r2+r4))))

  for i in r_list_target:
      r5 = coil_dia[i]
      angle3 = math.degrees(math.acos(((r2+r5)**2+(r2+r3)**2-(r3+r5)**2)/(2*(r2+r5)*(r2+r3))))
      angle4 = 360 - (angle1 + angle2 + angle3)
      d = math.sqrt(-math.cos(math.radians(angle4))*2*(r2+r4)*(r2+r5)+(r2+r4)**2+(r2+r5)**2)

      if r4 + r5 <= d:
          avoid_r_list_target.append(i)

  return avoid_r_list_target
```

[그림4 코일 겹침 방지 코드 1]

그러나 후속적으로 다양한 데이터를 사용하여 시각화 진행한 결과, 3단에도 코일의 겹침이 확인되었기 때문에 3단에 놓일 코일의 겹침도 방지해야 했다. 그러나 2단은 하단의 코일들이 바닥과 붙어 있어서 코일들의 반지름 만을 이용하여 겹침을 확인할 수 있었지만, 3단의 경우 동일한 방법의 적용이 어려웠다. 따라서 2단과 3단 모두 주변 코일의 좌표를 이용하여 겹침을 확인할 수 있는 함수가 필요하였다. 기존 코드의 경우에는 겹침 방지 함수를 사용하여 선적할 코일들을 모두 구하고, 각 코일들의 좌표를 계산했지만, 주변 코일의 좌표를 이용하여 겹침을 확인하기 위해서는 좌표를 마지막에 계산하는 기존의 방법은 비효율적이었다. 따라서 한 단에 선적할 코일을 구하고 해당 단의 코일들의 좌표를 계산하는 방식으로 코드를 수정하였다.



```
def avoid_overlapping(a1, r_list_target, coil_dia, row_index, ia_num, row_result, result_y, result_z):
    r = row_index

    if (a1 == 0 and row_result[ia_num][a1 + 1] == -1) or (a1 != 0 and row_result[ia_num][a1 - 1] == -1):
        return r_list_target

    elif (a1 == 0) and (row_result[ia_num][a1 + 1] != -1):
        return avoid_overlapping_case2(result_y[r][ia_num - 1][a1-1], result_z[r][ia_num - 1][a1-1],
        result_y[r][ia_num - 1][a1], result_z[r][ia_num - 1][a1],
        result_y[r][ia_num - 1][a1+1], result_z[r][ia_num - 1][a1+1],
        coil_dia[int(row_result[ia_num][a1+1])], coil_dia[int(row_result[ia_num - 1][a1-1])], coil_dia[int(row_result[ia_num - 1]
[a1])], coil_dia[int(row_result[ia_num - 1]       [a1+1])], r_list_target, coil_dia)

    elif row_result[ia_num][a1 + 1] != -1:
        return avoid_overlapping_case3(result_y[r][ia_num - 1][a1-1], result_z[r][ia_num - 1][a1-1],
        result_y[r][ia_num - 1][a1], result_z[r][ia_num - 1][a1],
        result_y[r][ia_num - 1][a1+1], result_z[r][ia_num - 1][a1+1],
        result_y[r][ia_num - 1][a1+2], result_z[r][ia_num - 1][a1+2],
        coil_dia[int(row_result[ia_num][a1-1])], coil_dia[int(row_result[ia_num][a1+1])], coil_dia[int(row_result[ia_num - 1][a1
-1])], coil_dia[int(row_result[ia_num - 1][a2])], coil_dia[int(row_result[ia_num - 1][a1+1])], coil_dia[int(row_result[ia_num - 1]
[a1+2])], r_list_target, coil_dia)

    else:
        return avoid_overlapping_case4(result_y[r][ia_num - 1][a1-1], result_z[r][ia_num - 1][a1-1],
        result_y[r][ia_num - 1][a1], result_z[r][ia_num - 1][a1],
        result_y[r][ia_num - 1][a1+1], result_z[r][ia_num - 1][a1+1],
        coil_dia[int(row_result[ia_num][a1-1])], coil_dia[int(row_result[ia_num - 1][a1-1])], coil_dia[int(row_result[ia_num - 1]
[a1])], coil_dia[int(row_result[ia_num - 1][a1+1])], r_list_target, coil_dia)
```

[그림5 코일 겹침 방지 코드 2]

1.2.3 무게 균형 맞추기

기존 알고리즘은 주변에 선적되는 코일과 크기 차이를 줄이기 위해 크기 순으로 정렬한 후 스케줄링하였기 때문에 배의 앞뒤 무게 균형이 맞지 않았다. 기존 결과를 활용하면서 앞뒤 밸런스를 맞추기 위하여 row를 다시 정렬하였고, 좌우 밸런스를 맞추기 위하여 row를 좌우 반전을 주어 정렬하였다.

```python
def set_center_of_gravity(result_dataframe, ship_hold_len):
    #앞 뒤

    result_row_num = result_dataframe.loc[:, 'coil_co_r'].max()
    result_row_gross = []

    for i in range(1, result_row_num + 1):
        result_row_gross.append(result_dataframe[result_dataframe.loc[:, 'coil_co_r'] == i].loc[:, 'GROSS'].sum())

    result_row_gross = pd.DataFrame(result_row_gross, columns=['gross_sum'])
    result_row_gross['row'] = result_row_gross.index + 1

    #앞 뒤 조절 전 출력
    n = len(result_row_gross)
    if n % 2 == 0:
        f_sum = result_row_gross.iloc[:n // 2]['gross_sum'].sum()
        b_sum = result_row_gross.iloc[n // 2:]['gross_sum'].sum()
    else:
        f_sum = result_row_gross.iloc[:n // 2]['gross_sum'].sum()
        b_sum = result_row_gross.iloc[(n // 2) + 1:]['gross_sum'].sum()
    print("\n조절 전) 앞:", f_sum, "뒤:", b_sum)

    result_row_gross = result_row_gross.sort_values(by="gross_sum")

    if len(result_row_gross) % 2 == 1:
        middle = result_row_gross.iloc[len(result_row_gross) // 2]
        result_row_gross = result_row_gross.append(middle, ignore_index=True)
        result_row_gross = result_row_gross.sort_values(by="gross_sum")

    new_result_row_gross = result_row_gross.copy()

    for i in range(len(result_row_gross) // 2):
        new_result_row_gross.iloc[i], new_result_row_gross.iloc[-(i + 1)] = result_row_gross.iloc[i ' 2], \
                                                                            result_row_gross.iloc[i ' 2 + 1]

    new_result_row_gross = new_result_row_gross.drop_duplicates()

    #앞 뒤 조절 후 출력
    n = len(result_row_gross)
    if n % 2 == 0:
        f_sum = new_result_row_gross.iloc[:n // 2]['gross_sum'].sum()
        b_sum = new_result_row_gross.iloc[n // 2:]['gross_sum'].sum()
    else:
        f_sum = new_result_row_gross.iloc[:n // 2]['gross_sum'].sum()
        b_sum = new_result_row_gross.iloc[(n // 2) + 1:]['gross_sum'].sum()
    print("조절 후) 앞:", f_sum, "뒤:", b_sum)

    for i in range(result_row_num):
        result_dataframe[result_dataframe.loc[:, 'coil_co_r'] == new_result_row_gross.iloc[i, 1]].loc[:, 'coil_co_r'] = i + 1

    #좌 우

    result_row_num = result_dataframe.loc[:, 'coil_co_r'].max()

    result_r_gross = [0 for _ in range(result_row_num)]
    result_l_gross = [0 for _ in range(result_row_num)]

    for i in range(result_row_num):
        for _, j in result_dataframe[result_dataframe.loc[:, 'coil_co_r'] == i + 1].iterrows():
            if j['y_coor'] >= ship_hold_len / 2:
                result_r_gross[i] += j['GROSS']
            else:
                result_l_gross[i] += j['GROSS']

    #좌 우 조절 전 출력
    print("조절 전) 좌:", sum(result_l_gross), "우:", sum(result_r_gross))

    for i in range(1, result_row_num):
        if ((sum(result_r_gross[:i - 1]) > sum(result_l_gross[:i - 1])) and (
                result_r_gross[i] > result_l_gross[i])) or (
                (sum(result_r_gross[:i - 1]) < sum(result_l_gross[:i - 1])) and (
                result_r_gross[i] < result_l_gross[i])):
            result_dataframe[result_dataframe.loc[:, 'coil_co_r'] == i]['y_coor'] = ship_hold_len - \
                                                                                    result_dataframe[result_dataframe.loc[:, 'coil_co_r'] == i][
                                                                                        'y_coor']
            result_r_gross[i], result_l_gross[i] = result_l_gross[i], result_r_gross[i]

    #좌 우 조절 후 출력
    print("조절 후) 좌:", sum(result_l_gross), "우:", sum(result_r_gross))

    return result_dataframe
```

[그림6 무게 균형 코드]

1.2.4 Literature Review

코일 선적 관련 논문 Optimization Search Algorithm of Allocation Planning for strip Coils in Hold for Shipment by Using Operational Know-how외 6편을 읽고, 해당 논문에서 사용한 알고리즘을 정리하며 학습하였다.

[그림7 논문 정리]

2. 후기

산업경영공학과의 과목들을 통해 최적화와 물류에 흥미를 느끼고 코딩을 좋아하는 저에게 이번 연구 참여는 매주 하루라도 더 빨리 과제 결과물을 발표하고 싶을 정도로 흥미로웠습니다. 특히 처음에 좌표 계산과 시각화를 맡게 되어서 직접 결과물을 확인할 수 있어서 연구 참여에 더 빠르게 몰입할 수 있었습니다. 또한 연구 참여 이전에는 논문 읽는 것을 크게 좋아하지 않았지만, 연구 참여를 통해 특정 분야의 과제를 진행하고 관련 논문을 읽으며 다른 사람들은 어떻게 문제를 해결하는지 알아가는 것도 매우 흥미로울 수 있다는 것을 알게 되었습니다.

또한 연구 참여를 통해 프로젝트가 전반적으로 어떻게 진행되는지 경험할 수 있었습니다. 이번이 첫 연구 참여였기 때문에 연구실에 어떻게 연구를 진행하고 논문을 작성하는지에 대한 배경지식이 없었으나 기업 과제를 진행하고 아이디어를 얻어서 논문을 작성할 수도 있다는 것을 알게 되었습니다. 이를 통해 대학원 진학 시 더 적응을 잘 할 수 있을 것이라는 확신을 갖게 되었습니다.

좌표 계산, 시각화, 겹침 방지, 밸런스 등 다양하고 재미있는 부분을 과제로 제공해주시고, 프로젝트의 전반을 함께할 수 있게 해준 김병인 교수님과 임상운 사수님께 감사드립니다.