REPORT

산업공학종합설계

최종 보고서

과목명 : 산업공학 종합설계

주제 : 피킹 작업 개선을 위한 쿠팡 물류센터의 자동화

담당교수님 : 이홍희 교수님

12120446 이원용

12140507 윤종찬

12150352 이도은

제출일 : 2019.12.2

목차

1. 서론 및 배경

4차 산업혁명의 핵심 디지털 기술의 발전과 함께 물류의 디지털화가 진행되고 있으며, 물류 혁신의 트렌드가 가속화되고 있다. 과거에는 보관, 분류 단계까지의 업무 효율화 관점에서 진행되었지만 오늘날 온라인 물류 (B2C: Business to Consumer)의 폭발적인 성장에 대응하기 위해 보관, 분류 단계를 넘어 피킹, 상차, 하역, 검수, 포장 업무까지 물류 센터 자동화의 영역 안에 들어오고 있다.

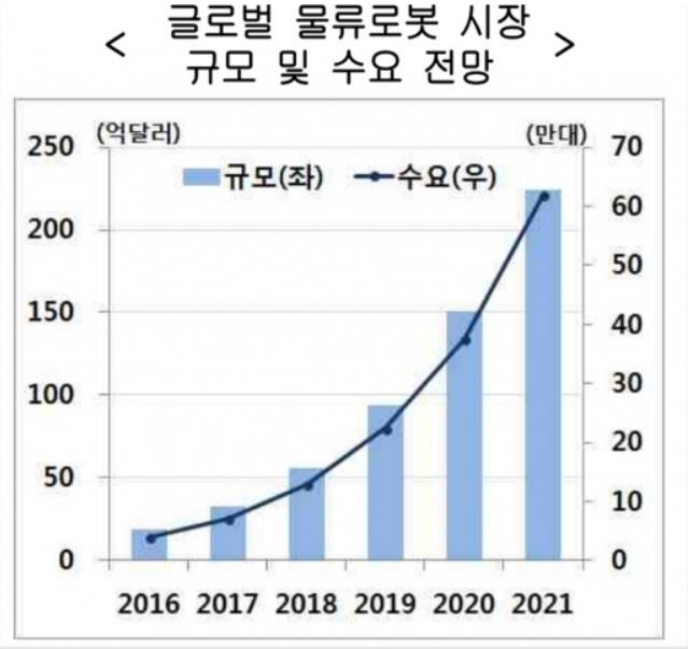


그림1 물류 로봇 시장의 규모 및 수요 전망

앞으로의 물류와 관련된 로봇, 기계들의 규모와 시장이 점차적으로 증가할 것으로 전망되며 더 이상 사람이 아닌 자동화된 시스템으로 물류가 운영될 것이라고 예상하고 있다. 하지만 실제로 유통물류센터의 운영 형태는 시장의 성장 속도를 따라가지 못하고 수작업 위주의 운영을 고수하고 있어 많은 작업이 작업자에게 의존하고 있다. 여기에 더해진 인건비의 상승으로 유통 업계에서는 물류 자동화 도입이 확산되고 있다.

자동화 설비를 도입하면서 인건비는 그대로 유지하여 성공한 사례로 롯데슈퍼를 이야기할 수 있다. 롯데슈퍼는 온라인 전용 배송 시스템에 자동화 물류 시스템을 도입하여 '오토프레시 의왕센터'를 운영 중에 있다. 피킹 로봇 1100대가 5분만에 주문 50건을 처리하면서 빠른 프로세스를 구축하였다. 고효율 창고 관리 시스템인 'GTP(Goods-To-Person) 피킹 시스템'을 이용해 로봇과 피킹 작업자가 협업하고 완전한 자동화는 아니지만, 이를 통해 주문처리시간 단축을 통한 생산성 증대와 주문처리 정확도 향상, 기존 센터 대비 저장 공간의

확대의 이점을 가졌다. 이는 완전한 자동화 시스템은 아니지만 피킹 작업을 보다 효율적으로 빠르게 진행하여 인건비도 줄이고 배송 속도도 향상시킬 수 있도록 해주었다.

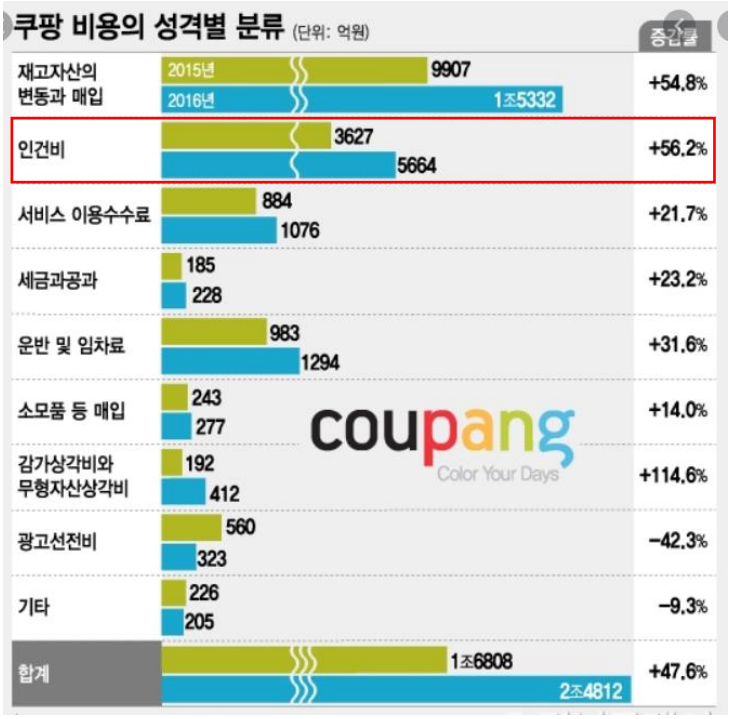


그림 2 쿠팡 비용의 성격 별 분류

이와 반대로 인건비의 상승으로 수익에 영향을 받은 회사로 쿠팡이 있다. 쿠팡은 현재 물류센터에서 '랜덤 스토우'라는 방식을 사용하고 있다. 이는 물류센터에 보관한 제품이 도착하면 입고 담당자는 '랜덤 스토우' 기반의 물류 소프트웨어가 설치된 단말기가 지시하는 진열대의 위치에 제품을 갖다 놓는다. 소프트웨어는 해당 제품의 주문 빈도와 센터 전체 내 배치 분포, 위치 별 재고량 등을 토대로 최적의 진열 장소를 계산해 입고 담당자에게 알려준다. 배치된 제품들을 토대로 주문 제품 목록을 받은 출고 담당자는 직원의 위치와 제품의 위치를 파악하여 여러 동선을 재빨리 계산하고 가장 가까운 진열대의 위치를 직원에게 알려준다. 랜덤 스토우 방식이 다양한 제품들을 한 번에 가져올 수 있는 효율성을 가지지만, 사람이 하는 일이다 보니 제품을 피킹하는 과정에서 발생하는 시간 낭비와 더 많은 제품을 더 빠르게 배송하고자 추가적으로 고용하는 직원으로 인한 인건비가 문제로 꼽히고 있다.

이를 해결하고자 롯데슈퍼의 오토스토어 모델을 착안하여 쿠팡의 물류센터 프로세스에 적합할 수 있는 개선 알고리즘을 구축하여 비용을 줄이고 효율을 높일 수 있도록 연구를 진행하고자 한다.

2. 연구목표와 방법론

1) 연구목표

본 연구의 목표는 쿠팡 물류센터에서 사용하는 랜덤 스토우 방식의 개선을 통한 인건비 절감과 동시에 처리 개수의 증대를 목적으로 한다. 이를 위해 롯데 슈퍼의 오토스토어 피킹 시스템 모델과 기존의 인력모델을 바탕으로 개선된 알고리즘을 비교하여 쿠팡 시스템에 맞는 최적의 방법을 찾아내고자 함이다.

2) 방법론 설정

본 연구에서는 랜덤 스토우 자동화에 대해 설비적인 측면보다 구현 가능할 것이라고 생각되는 기술들을 사용하여 피킹 로봇의 시간을 계산하고, 이에 따른 인건비를 산출할 수 있는 이론을 활용한 뒤, Python 프로그램을 이용하여 알고리즘을 구축한다.

① 재고유지비를 이용한 인건비 산출

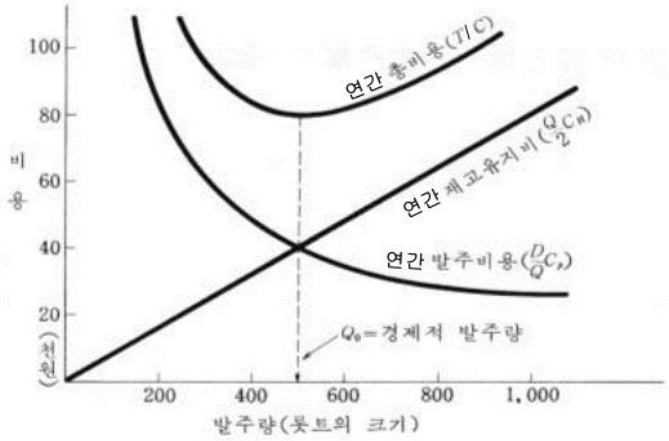


그림 3 재고관리에 따른 비용 그래프

1) 연간 발주 비용

= 연간 발주횟수 × 1회당 발주비용

= $\frac{\text{연간 수요량}(D)}{\text{1회 발주비용}(Q)}$ × 1회당 발주비용(C_P)

2) 연간 재고 유지비

= 평균 재고량 × 단위당 재고유지비

= $\frac{Q}{2}$ ×단위당 재고유지비(C_H)

TIC = 연간 발주비 + 연간 재고유지비

= $\frac{D}{Q}C_o + \frac{Q}{2} \times C_H$

여기서 인력모델의 경우는 발주비용과 관련이 없으므로 연간 재고 유지비를 변형하여 사용한다. 즉, 평균 재고량 $\frac{Q}{2}$ 를 작업자 수로 치환하고, 단위당 재고유지비 C_H를 시급에 따른 임금으로 치환하면 인력모델에 대한 인건비 계산할 수 있다. 더불어 주간 근로자와 야간 근로자로 나눠 그에 따른 인건비 계산을 위해 주간 시급은 8350원, 야간 시급은 12,525원으로 고정하여 계산한다.

② 피킹 로봇의 작업 처리 시간

작업을 수행하기 위한 로봇의 모델링에 사용되는 시간을 정의한다. 로봇의 피킹 시간과 재고를 선반에 채워 넣는 시간은 평균 5초로 가정한다. 로봇의 선반 내 진열대 간의 이동시간과 컨베이어 벨트 및 재고 조달 라인 간 이동시간은 1m/sec로 가정한다. 또한 재고조달을 위한 보충재고 피킹 시간은 평균 10초로 가정한다.

변수	값(초)
가로이동시간	0.5
로봇 및 피킹용 팔 출발동작시간	03.
로봇 및 피킹용 팔멈춤 동작시간	0.3
피킹용 팔 수직 이동시간	0.5
제품추출 및 토트 적재시간	5
토트 교환시간	5
토트 내보내는 시간	5

표 1 시뮬레이션을 위한 로봇의 설정 사양

먼저 비용에 관련된 변수를 설정(인력, 시간)하고 조건에 따라 계산하여 기존 모델의 시간당 비용 및 시간당 물류처리 개수를 산정한다. 그리고 설계된 자동화 시스템의 시간당 물류처리 개수와 비교하여 기존모델대비 투자효율을 분석하는 것으로 한다.

이를 위해 자동화 모델을 설정하고 알고리즘 구축을 진행한다. 그 후 알고리즘의 결과를 통해 기존 인력으로 진행되는 모델과의 결과와 대조하여 시간당 처리하는 제품의 개수와 그에 따른 비용을 비교하는 것으로 한다.

3. 분석 및 모델링

개선된 자동화 모델을 구축하기에 앞서 기존의 인력모델과 롯데슈퍼의 피킹 모델을 조사하고 분석하여 이를 토대로 실제 Python을 이용하여 자동화 모델을 모델링하고자 한다.

1) 인력 모델

: 1개의 선반에 1명의 작업자를 배치해 다음 process를 통해 진행된다.

① 작업자에게 지급된 PDA에 표시된 제품의 위치와 정보를 확인한다.

② 소프트웨어가 계산한 제품 간 경로를 확인하여 이를 따라 이동한다.

③ 해당 위치에 이동 후 제품의 바코드를 스캔해 송장 정보를 등록하고 토트에 적재한다.

④ 적재된 토트의 무게가 15kg에 도달할 경우 토트를 컨베이어 벨트로 운송한다.

⑤ 운송 후 ①~④의 과정을 반복하기 위해 다시 상품 진열대의 위치로 이동한다.

구분	UPH	근무시간 (Hour)	일 평균 처리량(개)	시급 (원/Hour)	총 임금
D.worker1	50	8	400	8,350	₩66,800
D.worker2	49	8	392	8,350	₩66,800
D.worker3	53	8	424	8,350	₩66,800
D.worker4	47	8	376	8,350	₩66,800
D.worker5	45	8	360	8,350	₩66,800
D.worker6	47	8	376	8,350	₩66,800
D.worker7	52	8	416	8,350	₩66,800
D.worker8	51	8	408	8,350	₩66,800
D.worker9	49	8	392	8,350	₩66,800
D.worker10	45	8	360	8,350	₩66,800
N.worker1	54	8	432	12,525	₩100,200
N.worker2	48	8	384	12,525	₩100,200
N.worker3	48	8	384	12,525	₩100,200
N.worker4	51	8	408	12,525	₩100,200
N.worker5	49	8	392	12,525	₩100,200
N.worker6	53	8	424	12,525	₩100,200
N.worker7	51	8	408	12,525	₩100,200
N.worker8	48	8	384	12,525	₩100,200
N.worker9	53	8	424	12,525	₩100,200
N.worker10	51	8	408	12,525	₩100,200
		총 처리량(개)	7952		₩1,670,000

표 2 작업자별 상품 처리 개수 및 임금

작업자의 하루 근무시간은 8시간(추가 연장 미 고려)이며, 주간과 야간 근무로 구분된다. 주간 근무자의 인건비는 8,350원X8시간 = 66,800원이며, 야간 근무자의 경우 12,525원X8시간 = 100,200원으로 계산된다. 주간과 야간이 동일한 작업자 수로 진행될 경우 (66,800X10) + (100,200X10) = 1,670,000원의 인건비가 발생한다.

각 작업자의 UPH평균은 49.7 unit이며 이는 제품 한 개를 피킹 하는데 약 1.2분이 소요된다는 것을 나타낸다.

2) 롯데슈퍼 피킹 모델

: 롯데슈퍼 오토프레시 안에는 총 3000여개의 상온상품이 약 7200개의 상품 보관 상자(Bin)에 나뉘어 있으며, 총 19대의 운반 적용(피킹) 로봇이 초속 3.1m로 움직여 실시간으로 상품의 입출고를 관리하고 있다. 아직은 사람과 기계의 협업으로 이루어지고 있지만 근무 인원의 증가 없이도 지속적으로 늘어나고 있는 온라인 배송을 처리할 수 있다. 구체적인 프로세스는 아래와 같다.

① 고객이 어플리케이션으로 주문한 상품의 위치를 각각 피킹 로봇에 부여한다.

② 주문대로 피킹 로봇이 1차로 상온 상품을 찾기 위해 움직인다.

③ 주문한 상품이 담긴 Bin에 도착하면 그리퍼가 부착된 판넬이 하강하여 고정된다.

④ 고정된 상태에서 주문한 상품을 들어올려 그리퍼 안에 담는다. (단,30kg가 최대)

⑤ 1차적으로 담긴 상자는 컨베이어 벨트를 이동하면서 냉동, 신선, 대형 상품을 담는다.

⑥ 담긴 상자를 사람이 포장하여 출고 Port로 옮겨 출고한다.

⑦ 위 ①-⑥의 과정을 반복한다.

그림4 오토스토어의 구조

롯데슈퍼 오토스토어 시스템에서 평균적으로 로봇 1대가 주문 1건을 처리하는데 7분이 걸린다. 1건당 담아야 할 상품의 개수는 평균 7개라고 한다. 이를 1시간동안 가동했을 때, 처리할 수 있는 건수는 60분/7분 = 약 8.5건이다. 여기서 1건당 평균 제품 개수를 곱하게 되면 1시간동안 처리할 수 있는 개수는 8.5*7 = 약 60(개), 8시간동안 가동했을 경우 60*8 = 480개를 1대가 처리할 수 있다. 이를 구성되어 있는 피킹 로봇 19대가 8시간동안 처리하는 건수를 계산하게 되면, 480 *19대 = 약 9,120개를 처리할 수 있는 것이라고 할 수 있다.

구분	UPH	가동시간(Hour)	일 평균 처리량(개)
로봇1	60	8	480
로봇2	57	8	456
로봇3	60	8	480
로봇4	55	8	440
로봇5	53	8	424
로봇6	62	8	496
로봇7	63	8	504
로봇8	58	8	464
로봇9	60	8	480
로봇10	59	8	472
로봇11	61	8	488
로봇12	60	8	480
로봇13	57	8	456
로봇14	57	8	456
로봇15	60	8	480
로봇16	60	8	480
로봇17	62	8	496
로봇18	56	8	448
로봇19	60	8	480
평균 처리량(건)	62	총 처리량(건)	8960(건)

표3 로봇 19대의 하루평균 처리건수

여기서 출고 Port와 입고 Port에서 근무하는 인원 5명과 전체 시스템에 문제가 생기는 것을 방지하기 위해 모니터를 보며 주시하는 근무자 2명을 포함해 총 7명의 근무인원을 둔다. 인력모델에 비해 3명을 줄여 공정을 유지할 수 있다.

3) 자동화 모델

: 파이썬 프로그래밍을 통해 피킹 로봇 한 대의 평균 이동시간, 피킹 시간을 변수로 설정하고 상품을 토트에 적재하고 컨베이어 벨트에 올려 전달되는 시간의 합을 계산하여 피킹 작업을 수행하는 사이클 타임을 반환하여 모델 간 비교를 실시한다. 기존 인력모델의 조건과 동일하게 10개의 선반에서 2개의 라인 당 1대의 피킹 로봇을 배치하고 각 로봇이 랜덤한 위치의 선반에 있는 물품들을 피킹한다. 피킹 도중 현재 토트 내의 무게와 다음 피킹 물품의 중량 합이 15kg를 초과할 경우 현재 토트를 컨베이어 벨트로 내보내고 빈 토트로 교환하여 피킹을 수행하는 프로세스로 구성한다.

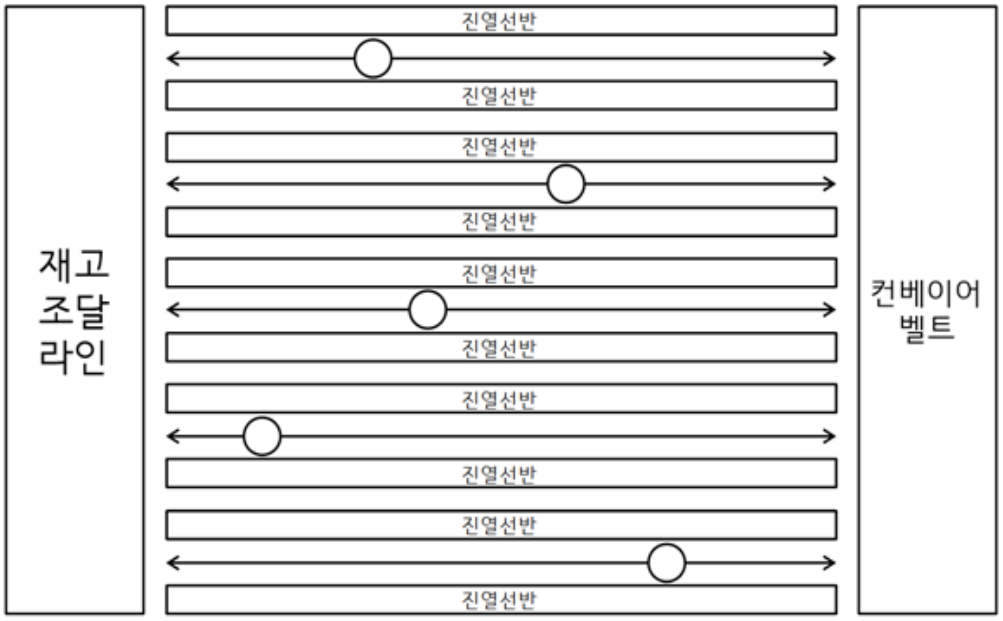


그림5 자동화 모델의 Layout

4. 설계 및 구현

설계 및 구현은 기존의 모델을 구현한 것이 아닌 쿠팡에 적용할 수 있도록 새로 개선한 알고리즘을 Python을 통해 구현한다. 설계를 통해 구현한 과정은 아래와 같다.

① 선반 클래스 작성
② 토트 클래스 작성
③ 로봇 클래스 작성
④ 시뮬레이션 1세트 클래스 (로봇 1대와 선반 2개)
⑤ 피킹 제품 랜덤 생성
⑥ 선반 10개 생성 후 피킹 제품 목록 랜덤 추출
⑦ 피킹 목록 중 해당 선반에 목록 전달을 위한 분할 함수
⑧ 분할된 피킹 제품에 대한 정보들을 전달하여 5세트 시뮬레이션
⑨ 결과 확인

① 선반 클래스 작성

```python
#선반 하나의 객체
import random

class Rack:
    def __init__(self, rack_num, row, col): #(1, 6, 100)으로 선반 객체를 생성하면 '6 X 100 인 1번 Rack' 이 생성
        self.shelf = {}
        self.rack_num = rack_num
        self.row = row
        self.col = col
        for i in range(0,row):
            for j in range(0,col):
                self.shelf[rack_num,i,j] = round(random.random()*15, 2) # 선반 번호, 행, 열 에 해당하는 위치에 15kg 이하 중량 임의 생성
```

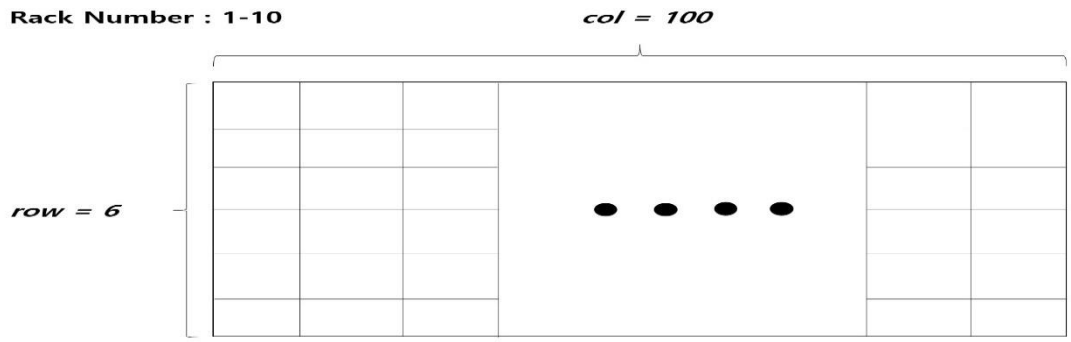


그림 6 선반 객체

구성요소는 선반 번호, 열의 수, 행의 수로 이루어져 있다. 쿠팡의 물류 창고 모델링을 위해 6X100 형태의 선반을 만들고 해당하는 선반 번호를 붙인다. 또한 선반이 생성되면서 1-15 사이의 난수를 각 셀에 두어 결과적으로 제품이 담긴 선반을 생성할 수 있다.

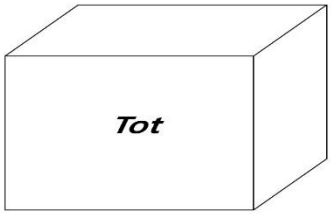
(예) (1, 3, 100):13, 1번 선반 3,100 자리에 13kg의 제품)

② 토트 클래스 작성

```
# 토트 박스 하나의 객체
class Tot:
    def __init__(self):
        self.limit_weight = 15 # 토트 박스 내 제품 중량 합의 한계
        self.comp_weight = 0 # 비교하기 위해 사용되는 중량
        self.now_weight = 0 # 현재 박스 내 제품의 중량 합
        self.tot_dict = {} # 토트의 정보를 저장하기 위한 딕셔너리

    # 현재 중량을 반환하는 메서드
    def getNow(self):
        return self.now_weight

    # 현재 토트에 새로운 물건의 중량(weight)을 추가할 때, 추가하면 한계를 넘어가는지 안 넘어가는지 판단
    # 넘어가지 않으면 True를 반환, 넘어 간다면 False를 반환
    def fullOrNot(self, weight):
        self.comp_weight = self.now_weight + weight
        if self.limit_weight > self.comp_weight:
            return True
        else:
            return False
```



Limit Weight : 15kg

Now Weight :
The weight of current tot weight

Compare Weight :
Current Weight + Weight of new picking product

Full or Not Method :
If tot is not full -> return True
If tot is full -> return False

그림 7 토트 객체

토트 객체는 중량 제한이 15kg이고 현재 중량과 비교하기 위한 중량이 존재한다. 비교하기 위한 중량은 현재 중량에 피킹된 제품의 중량을 더한 값으로, 토트가 가득 찼는지 차지 않았는지 알려주는 Full or Not 메서드에서 사용된다.

③ 로봇 클래스 작성

```
# 로봇 객체
class Robot:
    def __init__(self, rack):
        self.rack = rack
        self.horz_time = 0.5  # 가로 이동 시간
        self.start_time = 0.3  # 출발 동작 시간
        self.stop_time = 0.3  # 멈춤 동작 시간
        self.arm_start_time = 0.3  # 피킹 팔 출발 동작 시간
        self.arm_stop_time = 0.3  # 피킹 팔 멈춤 동작 시간
        self.arm_vert_time = 0.5  # 피킹 팔 수직 이동 시간
        self.pick_time = 5  # 제품 추출 시간
        self.drop_time = 5  # 토트 박스 담는 시간
        self.tot_change_time = 5 # 토트 박스 교환 시간
        self.tot_drop_time = 5 # 토트 내보내는 시간
        self.haveTot = self.setTot(20) # 로봇의 초기 빈 토트의 갯수를 세팅

    # 초기 토트를 리스트 안에 보관하는 메서드
    def setTot(self, t_num):
        totList = []
        for i in range(0, t_num):
            a = Tot()
            totList.append(a)

        return totList

    # 토트 위치는 마지막 행과 동일한 높이라고 가정
    # 따라서 drop 위치는 마지막 행보다 한 칸 위 (row = 6 이고 마지막 칸 index = 5 이므로 drop point = 4)
    def drop_point(self):
        return self.rack.row - 2

    # 로봇 팔의 초기 위치를 세팅
    # 로봇 팔의 초기 위치는 전체 행의 갯수의 중앙으로 가정
    # 따라서 초기 위치는 (전체 행의 수)/2, (row = 6, 마지막 칸 indedx = 5 이므로 init point = 5/2 = 2.5)
    def init_point(self):
        return (self.rack.row - 1)/2
```

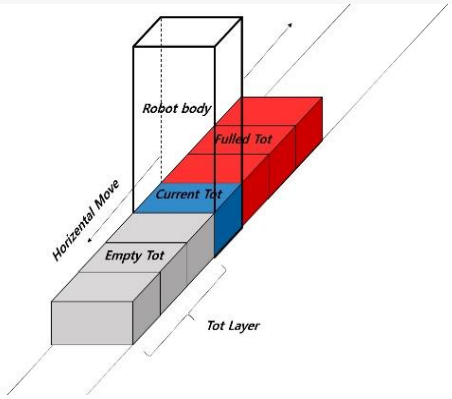


그림 8 로봇의 수평 이동

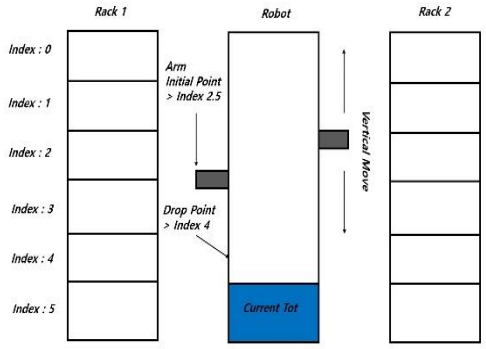


그림9 로봇 팔의 수직 이동

그림8은 로봇의 수평 이동에 대한 그림이다. 로봇의 몸체는 수평 이동만을 한다. 초기에 로봇에는 미리 지정한 토트 개수만큼 빈 토트를 세팅한다. 로봇의 수평 이동 시간은 (선반 한 칸당 수평 이동 시간=0.5), (몸체 출발 시간=0.3), (몸체 멈춤 시간=0.3)으로 구성되어 있다. 또한 토트가 가득 찼을 때 (토트를 교환하는 시간=5)과 작업 완료 시 토트를 내보내는 (토트를 내보내는 시간=5)로 구성되어 있다.

이어서 그림9를 보면 몸체가 멈춘 후에 양쪽의 로봇 팔이 수직 이동에 대한 정보를 알 수 있다. 한 번의 피킹이 끝나면 로봇 팔은 초기 위치로 돌아온다는 설정을 하기 위해 초기 위치는 선반의 중간 위치인 index = 2.5로 늘 위치시킨다. 초기 위치에서 수직 이동 이후에는 피킹을 실시하고 토트에 담을 수 있도록 토트보다 한 칸 위인 index = 4에서 제품을 drop한다. 이러한 수직 운동에 해당하는 시간은 (로봇 팔 출발 시간=0.3), (로봇 팔 멈춤 시간=0.3), (선반 한 칸당 수직 이동 시간=0.5), (제품 추출 시간=5), (토트에 담는 시간=5)로

구성되어 있다.

④ 시뮬레이션 1세트 클래스 (로봇 1대와 선반 2개)

```
# 선반 2개에 로봇 1개의 처리 시뮬레이션
import random

class MRset:
    def __init__(self, p_num, rack1, rack2, picking_product):
        self.p_num = p_num # picking 해야할 제품의 갯수
        self.rack1 = rack1 # 선반 1
        self.rack2 = rack2 # 선반 2
        self.rackSet = {**self.rack1.shelf, **self.rack2.shelf} # 선반 1,2 를 전체적으로 다루기 위해 지정
        self.robot = Robot(rack1) # 선반 1,2 를 담당할 로봇
        self.picking_product = picking_product # picking 해야할 제품 (위치:중량 형태의 딕셔너리로 전달)
        self.time = self.simulate() # 선반 2개와 로봇 1개의 처리 시간을 가짐

    # Simulation about 2 racks used 1 machine
    # 시간의 계산은 본체 이동 시간 + 피킹 마다 걸리는 시간 + 토트 박스 내보내는 시간으로 실시
    def simulate(self):
        time_sec = 0
        time_sec += self.rack1.col * self.robot.horz_time # 한 칸 당 이동 시간이 동일하므로 (열 갯수) * (수평 이동 시간)
        finished_lst = [] # 피킹이 완료된 제품을 담는 리스트

        #Robot Point setting
        init_point = self.robot.init_point() # 로봇 팔의 초기 위치는 0-5의 평균값 (중앙 위치) : 2.5
        drop_point = self.robot.drop_point() # 토트의 높이는 한 칸의 높이와 같으므로 drop point 행의 수 - 한 칸 : 5-1 = 4


        #Tot
        tot_layer = self.robot.haveTot
        tot_num = 1 # 처음 토트의 Number : 1
        now_tot = tot_layer.pop(0) # 토트 목록 중 첫 번째 tot를 선택
        full_tot = {} # 꽉 찬 토트 정보를 저장하는 딕셔너리
        full_tot_layer = [] # 꽉 찬 토트를 저장하는 리스트

        #Picking Product
        pick_product = self.picking_product
        l_pick_product = list(pick_product) # 리스트로 변환해서 피킹해야하는 제품의 위치만을 저장

        #보고서 준비를 위한 리스트 변환 및 보고서 시작 (보기 편하게 양식 만들어 출력)
        lrack1 = list(self.rack1.shelf)
        lrack2 = list(self.rack2.shelf)
        rack_num1 = lrack1.pop(0)[0]
        rack_num2 = lrack2.pop(0)[0]
        print("[ rack",rack_num1,"/rack",rack_num2," 결과 보고서 ]₩n")
        print("< Picking Product Information > ₩n")
        print("Picking 목록 : ",pick_product)
        print("Picking 갯수 : ",len(pick_product))
        print("Product Weight 합 : ",sum(pick_product.values()),"₩n")
        print("< Tot Box Information > ₩n")
```

시뮬레이션은 로봇 1대가 선반 2개를 처리하는 시간을 반환하는 것을 목적으로 작성되었다. 이 때 전달되는 인자는 피킹 제품 개수, 선반 1개, 다른 선반, 피킹 제품 목록이 전달된다. 시간을 반환하게 되므로 time_sec 변수를 0으로 초기화하고 계산을 쉽게 하기 위해 time_sec에 모든 칸을 수평이동한 시간을 먼저 더해준다.

```
        # 피킹하는 작업에 대한 시간 계산
        while len(finished_lst) < self.p_num:

            curTot = now_tot # 현재 토트 = 사용 중인 토트
            now_product = l_pick_product.pop(0) # 피킹하고자 하는 제품의 위치 정보의 제일 앞에서 pop
            curProd_weight = self.rackSet[now_product] # 피킹하고자 하는 제품의 중량을 rackSet에서 뽑아서 저장

            # 마지막 위치 기반 시간 계산
            cur_point_y = now_product[1]

            time_sec += (self.robot.start_time
                    + self.robot.stop_time
                    + self.robot.arm_start_time # 로봇 팔 출발
                    + (self.robot.arm_vert_time
                    * abs(init_point - cur_point_y)) # | 초기 위치 - 현재 위치 | * 팔 수직 이동 시간
                    + self.robot.arm_stop_time # 로봇 팔 멈춤
                    + self.robot.pick_time # 순수 피킹 시간
                    + self.robot.arm_start_time # 피킹 후 로봇 팔 출발
                    + (self.robot.arm_vert_time
                    * abs(drop_point - cur_point_y)) # | 현재 위치 - 토트 박스 위치 | * 팔 수직 이동 시간
                    + self.robot.arm_stop_time # 토트 박스 위치에서 팔 멈춤
                    + self.robot.drop_time  # 토트 박스에 넣는 시간
                    + self.robot.arm_start_time # 토트에 넣은 후 출발
                    + (self.robot.arm_vert_time
                    * abs(drop_point - init_point)) # | 토트박스 위치 - 초기 위치 | * 팔 수직 이동 시간
                    + self.robot.arm_stop_time # 초기 위치에서 팔 멈춤
                    )

            finished_lst.append(now_product) # 피킹이 끝난 제품을 확인하기 위한 리스트

            if curTot.fullOrNot(curProd_weight): # 토트가 가득 찼는지를 판단하는 메서드 (가득차지 않았으면 True)
                curTot.tot_dict[now_product] = curProd_weight  # 피킹한 제품의 위치와 중량을 저장하는 딕셔너리
                curTot.now_weight += curProd_weight #  토트내 중량 최신화

            else: # 토트가 가득 찼을 때
                full_tot[tot_num] = now_tot.now_weight
                full_tot_layer.append(now_tot) # 가득 찬 토트를 저장하는 딕셔너리에 토트 번호와 중량 저장
                print(tot_num,"번 토트 : ", now_tot.tot_dict) # 토트가 가득 찼을 때 토트 정보 출력
                tot_num += 1 # 토트 번호 업그레이드
                now_tot = tot_layer.pop(0) # 현재 토트를 토트 레이어에서 다음 토트로 최신화
                time_sec += self.robot.tot_change_time # 토트 교환 시간
                curTot = now_tot # 현재 토트 최신화
                curTot.tot_dict[now_product] = curProd_weight
                curTot.now_weight += curProd_weight # 최신화된 토트에 중량 추가


        full_tot[tot_num] = now_tot.now_weight # 마지막 토트가 가득 차지 않았더라도 작업이 끝났으므로 토트 딕셔너리에 저장
        time_sec += self.robot.tot_drop_time * len(full_tot) # 토트 하나당 내보내는 시간 X 토트 갯수

        for elmt in finished_lst: # pciking한 위치에 대해 중량을 0으로 바꿔 줌
            if elmt[0]%2 == 1 : self.rack1.shelf[elmt] = 0
            elif elmt[0]%2 == 0: self.rack2.shelf[elmt] = 0
```

다음은 실제적으로 피킹하는 시간을 계산한다. 시간 계산은 피킹하는 위치에 이동을 했을 때만을 고려한다면 다음의 과정을 따르게 된다.

로봇 출발 시간 → 로봇 수평 이동 시간 → 로봇 멈춤 시간
→ 로봇 팔 출발 시간 → 로봇 팔 수직 이동 시간 → 로봇 팔 멈춤 시간
→ 피킹 시간 → 피킹 후 로봇 팔 출발 시간 → 피킹 후 로봇 팔 수직 이동 시간 → 토트 박스 drop 위치에서 로봇 팔 멈춤 시간 → 토트 박스에 drop하는 시간
→ 토트 박스 drop 후 로봇 팔 출발 시간 → 초기 위치로 수직 이동 시간
→ 초기 위치에서 로봇 팔 멈춤 시간

이러한 과정 중 로봇의 수평 이동 시간은 모두 고려되어 time_sec 변수에 저장되어 있으므로 해당 부분을 빼준 시간들을 각 제품에 피킹 시마다 time_sec 변수에 연달아 더해준다. 또한 토트가 가득 찼을 때마다 토트를 교환하는 시간을 더해준다. 이러한 과정이 모두 완료되면 피킹을 완료한 선반의 위치에 중량 값을 모두 0으로 바꾸어 비어 있음을 표시한다.

```
        # 결과 보고서 형식으로 프린트
        print(tot_num,"번 토트는", now_tot.tot_dict)
        print("토트 정보 : ", full_tot)
        print("토트 박스 수 : ", len(full_tot), "₩n")
        print("__라인 처리 시간__ ")
        self.hms(time_sec)

        # 피킹이 정확한 위치에 됐는지 확인하는 보고서
        check_dict = {}
        for position in finished_lst:
            if position[0]%2 == 1: check_dict[position] = self.rack1.shelf[position]
            elif position[0]%2 == 0: check_dict[position] = self.rack2.shelf[position]
        print("₩n 빈 위치 : ",check_dict)
        print("="*125,"₩n")

        return time_sec

    # 초 단위를 시, 분, 초로 분할
    def hms(self, time):
        hour = time // 3600
        time = time - hour * 3600
        mi = time // 60
        time = time - mi * 60
        print(hour, "시간 ", mi, "분 ", time, "초")
```

토트 하나가 처리가 완료되면 그에 대한 토트 정보를 출력한다. 몇 번 토트에 얼마나 제품이 담겼는지, 토트 내의 제품은 어느 위치에서 피킹되었는지, 총 토트의 개수는 몇 개가 사용되었는지를 출력하고 해당 라인을 처리하는데 걸리는 시간을 초 단위에서 시간, 분, 초로 나타내어 출력한다.

⑤ 피킹 제품 랜덤 생성

```
# 1피킹할 제품을 랜덤으로 생성
def prob_Product(num, setOfRack): # (피킹하고자 하는 제품 갯수, 선반 - 딕셔너리 형태로 전달)
    lrackSet = list(setOfRack)
    l_prod = []

    # 피킹하고자 하는 제품 갯수를 랜덤으로 피킹
    while len(l_prod) < num:
        rlst = random.choice(lrackSet)
        if not rlst in l_prod:
            l_prod.append(rlst)

    # 로봇의 작동 순서에 따라 정렬
    reverse_l_prod = reverseTup(l_prod)
    reverse_l_prod.sort()
    result = reverseTup(reverse_l_prod)


    d_prod = {}
    # 딕셔너리 형태로 재생성
    while len(d_prod) < num:
        keyy = result.pop(0)
        d_prod[keyy] = setOfRack[keyy]

    return d_prod

# 리스트 반대 정렬을 위해 사용되는 메서드
# 피킹 시 로봇이 열이 커지는 방향으로 이동하므로 열, 행, 선반 번호 순으로 정렬해 줌
def reverseTup(lst):
        reverse_lst = []
        for elmt in lst:
            l_elmt = list(elmt)
            tmp = l_elmt[0]
            l_elmt[0] = l_elmt[2]
            l_elmt[2] = tmp
            t_elmt = tuple(l_elmt)
            reverse_lst.append(t_elmt)
        return reverse_lst
```

인자로 피킹하고자 하는 제품의 개수와 선반을 전달하면 전체 선반에서 피킹해야 하는 목록을 랜덤으로 생성하고 피킹 순서대로 정렬한다.

⑥ 선반 10개 생성 후 피킹 목록 랜덤 추출

```
# 10개의 선반 생성
r1 = Rack(1, 6, 100)
r2 = Rack(2, 6, 100)
r3 = Rack(3, 6, 100)
r4 = Rack(4, 6, 100)
r5 = Rack(5, 6, 100)
r6 = Rack(6, 6, 100)
r7 = Rack(7, 6, 100)
r8 = Rack(8, 6, 100)
r9 = Rack(9, 6, 100)
r10 = Rack(10, 6, 100)

# 저장소 내의 모든 선반
allOfRack = {**r1.shelf,**r2.shelf,**r3.shelf,**r4.shelf,**r5.shelf
             ,**r6.shelf,**r7.shelf,**r8.shelf,**r9.shelf,**r10.shelf
             }

# 저장소 내에서 50개의 피킹할 제품을 임의 생성
product = prob_Product(50, allOfRack)
```

①에서 작성한 선반 클래스를 이용하여 10개의 선반을 만들고, 모든 10개의 선반에서 피킹할 제품을 50개 랜덤 생성한다.

⑦ 피킹 목록 중 해당 선반에 목록 전달을 위한 분할 함수

```
# 각 선반에 할당된 제품을 나누는 메서드
# 예를 들어 1,2 번 선반에는 선반 번호가 1,2에 해당하는 피킹 제품들이 전달
def division_product(rack_num, product):
    list_of_product = list(product)
    list_of_product.sort()
    list_of_dvsn_product = {}
    prod_num = len(product)

    for i in range(1,10,2):
        add_to_product = {}
        for elmt in list_of_product:
            if elmt[0] == i or elmt[0] == i+1:
                add_to_product[elmt] = product[elmt]

        reverse_l_prod = reverseTup(list(add_to_product))
        reverse_l_prod.sort()
        result = reverseTup(reverse_l_prod)


        d_prod = {}

        while len(result) > 0:
            keyy = result.pop(0)
            d_prod[keyy] = product[keyy]

        list_of_dvsn_product[i,i+1] = d_prod

    return list_of_dvsn_product

dv_prod = division_product(10, product)
```

전체 피킹 목록을 선반 별로 분할하기 위한 함수를 사용한다. 이 과정을 거치면 전체 피킹 목록 중 1번 선반에 해당되는 제품들의 정보는 1번 선반에 전달되고, 2-10번까지 각 선반 당 피킹 정보를 전달하게 된다.

⑧ 분할된 피킹 제품에 대한 정보들을 전달하여 5세트 시뮬레이션

```
# (선반 한 쌍에 전달되는 피킹 제품 수, 선반1, 선반2, 선반 한 쌍에서 피킹되어야 하는 제품 딕셔너리) 형태로 simulation 진행
# 즉, 선반 2개 로봇 1개에 대한 시뮬레이션을 5세트 실시
r12 = MRset(len(dv_prod[1,2]), r1, r2, dv_prod[1,2])
r34 = MRset(len(dv_prod[3,4]), r3, r4, dv_prod[3,4])
r56 = MRset(len(dv_prod[5,6]), r5, r6, dv_prod[5,6])
r78 = MRset(len(dv_prod[7,8]), r7, r8, dv_prod[7,8])
r910 = MRset(len(dv_prod[9,10]), r9, r10, dv_prod[9,10])

# 10개 선반에 대해 시스템 종료 시간 출력 (가장 늦게 끝나는 set이 시스템 종료 시간이므로 가장 오래 걸리는 선반과 시간을 따로 출력
)
max_line = ()
d_time = {r12:r12.time, r34:r34.time, r56:r56.time, r78:r78.time, r910:r910.time}
late_time = max(r12.time, r34.time, r56.time, r78.time, r910.time)

for key in d_time.keys():
    if d_time[key] == late_time: max_line = key

print("해당 라인 : ",max_line.rack1.rack_num,",",max_line.rack2.rack_num)
print("₩n__시스템 종료 시간__")
max_line.hms(late_time)
```

시뮬레이션 5세트에 각 선반들이 피킹해야 하는 제품 목록을 전달하여 시뮬레이션을 실행한다. 각 처리는 동시 시작하기 때문에 가장 시간이 오래 걸린 시뮬레이션 세트의 처리 시간을 시스템 전체의 처리 시간으로 산출한다.

⑨ 결과 확인

```
[ rack 1 /rack 2  결과 보고서 ]

< Picking Product Information >

Picking 목록 :  {(1, 4, 26): 12.46, (2, 4, 37): 12.88, (1, 3, 44): 13.29, (2, 5, 51): 6.97, (1, 4, 72): 2.51, (2, 2, 74): 5.23, (1, 3,
75): 6.06, (2, 4, 86): 7.86, (1, 5, 94): 3.86}
Picking 갯수 :  9
Product Weight 합 :  71.12

< Tot Box Information >

1 번 도드 :  {(1, 4, 26): 12.46}
2 빈 토트 :  {(2, 4, 37): 12.88}
3 번 도드 :  {(1, 3, 44): 13.29}
4 빈 토트 :  {(2, 5, 51): 6.97, (1, 4, 72): 2.51, (2, 2, 74): 5.23}
5 번 도드 :  {(1, 3, 75): 6.06, (2, 4, 86): 7.86}
6 빈 토트는 {(1, 5, 94): 3.86}
로드 성보 :  {1: 12.46, 2: 12.88, 3: 13.29, 4: 14.71, 5: 13.92, 6: 3.86}
토트 박스 수 :  5

__라인 처리 시간__
0.0 시간  3.0 분  52.6000000000005 초

 본 위치 :  {(1, 4, 26): 0, (2, 4, 37): 0, (1, 3, 44): 0, (2, 5, 51): 0, (1, 4, 72): 0, (2, 2, 74): 0, (1, 3, 75): 0, (2, 4, 86): 0,
(1, 5, 94): 0}
==========================================================================================================================
```

[rack 3 /rack 4 결과 보고서]

< Picking Product Information >

Picking 목록 : {(4, 3, 1): 7.03, (4, 0, 3): 6.2, (3, 1, 4): 11.09, (3, 4, 10): 8.93, (3, 4, 15): 1.71, (3, 5, 16): 14.21, (3, 4, 22): 12.21, (4, 4, 39): 10.52, (4, 5, 46): 5.38, (4, 1, 52): 9.55, (3, 0, 53): 13.1, (3, 1, 55): 6.43, (4, 4, 58): 10.14, (3, 2, 83): 8.6, (4, 5, 87): 6.05}
Picking 갯수 : 15
Product Weight 합 : 131.14999999999998

< Tot Box Information >

1 번 도트 : {(4, 3, 1): 7.03, (4, 0, 3): 6.2}
2 번 토트 : {(3, 1, 4): 11.09}
3 번 도트 : {(3, 4, 10): 8.93, (3, 4, 15): 1.71}
4 번 토트 : {(3, 5, 16): 14.21}
5 번 도트 : {(3, 4, 22): 12.21}
6 번 토트 : {(4, 4, 39): 10.52}
7 번 도트 : {(4, 5, 46): 5.38, (4, 1, 52): 9.55}
8 번 토트 : {(3, 0, 53): 13.1}
9 번 도트 : {(3, 1, 55): 6.43}
10 번 토트 : {(4, 4, 58): 10.14}
11 번 도트는 {(3, 2, 83): 8.6, (4, 5, 87): 6.05}
토트 정보 : {1: 13.23, 2: 11.09, 3: 10.64, 4: 14.21, 5: 12.21, 6: 10.52, 7: 14.93, 8: 13.1, 9: 6.43, 10: 10.14, 11: 14.649999999999999}
토트 박스 수 : 11

__라인 처리 시간__
0.0 시간 6.0 분 16.5 초

 빈 위치 : {(4, 3, 1): 0, (4, 0, 3): 0, (3, 1, 4): 0, (3, 4, 10): 0, (3, 4, 15): 0, (3, 5, 16): 0, (3, 4, 22): 0, (4, 4, 39): 0, (4, 5, 46): 0, (4, 1, 52): 0, (3, 0, 53): 0, (3, 1, 55): 0, (4, 4, 58): 0, (3, 2, 83): 0, (4, 5, 87): 0}
==========

[rack 5 /rack 6 결과 보고서]

< Picking Product Information >

Picking 목록 : {(5, 1, 11): 6.55, (6, 0, 35): 11.79, (5, 0, 42): 1.14, (6, 1, 64): 9.72, (5, 4, 65): 1.14, (6, 0, 76): 9.33, (5, 1, 79): 4.69, (6, 4, 81): 14.8}
Picking 갯수 : 8
Product Weight 합 : 58.16

< Tot Box Information >

1 번 토트 : {(5, 1, 11): 6.55}
2 번 도트 : {(6, 0, 35): 11.79, (5, 0, 42): 1.14}
3 번 토트 : {(6, 1, 64): 9.72, (5, 4, 65): 1.14}
4 번 도트 : {(6, 0, 76): 9.33, (5, 1, 79): 4.69}
5 번 토트는 {(6, 4, 81): 14.8}
트트 정보 : {1: 6.55, 2: 12.93, 3: 10.860000000000001, 4: 14.02, 5: 14.8}
토트 박스 수 : 5

__라인 처리 시간__
0.0 시간 3.0 분 38.200000000000045 초

 빈 위치 : {(5, 1, 11): 0, (6, 0, 35): 0, (5, 0, 42): 0, (6, 1, 64): 0, (5, 4, 65): 0, (6, 0, 76): 0, (5, 1, 79): 0, (6, 4, 81): 0}
==========

[rack 7 /rack 8 결과 보고서]

< Picking Product Information >

Picking 목록 : {(7, 1, 15): 4.95, (8, 1, 39): 8.0, (8, 5, 41): 12.24, (8, 2, 64): 3.15, (7, 3, 69): 4.64, (7, 0, 74): 0.22, (7, 4, 75): 7.42, (8, 0, 87): 3.72}
Picking 갯수 : 8
Product Weight 합 : 44.339999999999996

< Tot Box Information >

1 번 토트 : {(7, 1, 15): 4.95, (8, 1, 39): 8.0}
2 번 투트 : {(8, 5, 41): 12.24}
3 번 토트 : {(8, 2, 64): 3.15, (7, 3, 69): 4.64, (7, 0, 74): 0.22}
4 번 토트는 {(7, 4, 75): 7.42, (8, 0, 87): 3.72}
토트 정보 : {1: 12.95, 2: 12.24, 3: 8.01, 4: 11.14}
토트 박스 수 : 4

__라인 처리 시간__
0.0 시간 3.0 분 25.700000000000045 초

 빈 위치 : {(7, 1, 15): 0, (8, 1, 39): 0, (8, 5, 41): 0, (8, 2, 64): 0, (7, 3, 69): 0, (7, 0, 74): 0, (7, 4, 75): 0, (8, 0, 87): 0}
==========

```
[ rack 9 /rack 10  결과 보고서 ]

< Picking Product Information >

Picking 목록 :  {(10, 3, 0)  9.2, (9, 3, 1): 11.04, (9, 1, 9): 0.45, (9, 0, 10): 14.9, (9, 2, 38): 10.53, (10, 1, 42): 2.32, (9, 3, 4
2): 12.73, (9, 5, 47): 7.33, (9, 3, 70)  13.48, (10, 4, 71): 12.15}
Picking 갯수 :  10
Product Weight 합 :  94.13000000000001

< Tot Box Information >

1 번 도트 :  {(10, 3, 0): 9.2}
2 번 토트 :  {(9, 3, 1): 11.04, (9, 1, 9): 0.45}
3 번 도트 :  {(9, 0, 10): 14.9}
4 번 토트 :  {(9, 2, 38): 10.53, (10, 1, 42): 2.32}
5 번 도트 :  {(9, 3, 42): 12.73}
6 번 투트 :  {(9, 5, 47): 7.33}
7 번 도트 :  {(9, 3, 70): 13.48}
8 번 투트는 {(10, 4, 71): 12.15}
트트 정보 :  {1: 9.2, 2: 11.489999999999998, 3: 14.9, 4: 12.85, 5: 12.73, 6: 7.33, 7: 13.48, 8: 12.15}
토트 박스 수 :  8

__라인 처리 시간__
0.0 시간  4.0 분  31.000000000000057 초

 빈 위치 :  {(10, 3, 0): 0, (9, 3, 1): 0, (9, 1, 9)  0, (9, 0, 10): 0, (9, 2, 38): 0, (10, 1, 42): 0, (9, 3, 42): 0, (9, 5, 47): 0, (9,
3, 70): 0, (10, 4, 71): 0}
==========================================================================================================

해당 라인 :  3 , 4

__시스템 종료 시간__
0.0 시간  8.0 분  16.5 초
```

이렇게 선반 10 개에 대한 로봇 5 대의 시스템 처리 시간을 산출하고 토트에 대한 정보와 제품이 올바르게 피킹 되었는지 확인할 수 있는 보고서가 작성된다.

---------- 50 개 10 회 실험 ----------

시뮬레이션	피킹 개수	최대 시간 라인 정보	시간
1	50	3,4 선반 15 개 피킹	6 분 16.50 초
2	50	7,8 선반 14 개 피킹	5 분 09.60 초
3	50	9,10 선반 13 개 피킹	5 분 28.70 초
4	50	7,8 선반 13 개 피킹	5 분 11.70 초
5	50	5,6 선반 13 개 피킹	5 분 45.20 초
6	50	1,2 선반 13 개 피킹	5 분 28.70 초
7	50	1,2 선반 12 개 피킹	5 분 12.30 초
8	50	5,6 선반 17 개 피킹	6 분 25.80 초
9	50	7,8 선반 14 개 피킹	5 분 58.10 초
10	50	5,6 선반 12 개 피킹	5 분 04.30 초

표 4 로봇 5대의 10회 실험에 대한 결과

10회 실험의 결과로 5대의 로봇이 50개를 피킹하는데 있어서 적게는 5분 4초, 길게는 6분 25초가 걸리는 것을 알 수 있다. 기존의 인력모델과 롯데슈퍼의 피킹 시간을 비교하면 확연하게 줄어든 것을 알 수 있으며 이를 통해 피킹 시간의 감소와 로봇의 도입에 따른 고용인원의 감소를 기대할 수 있다.

5. 결론

본 연구는 앞선 요소들을 개선하고자 다양한 변수를 고려하여 장기적인 관점으로 봤을 때 어떤 방식이 쿠팡 물류센터에 효율적일 수 있을지 찾기 위한 연구였다.

현재 쿠팡 물류센터의 일부 자동화를 위한 개선을 진행하기에 앞서 기존 인력을 사용한 모델과 자동화가 도입되어 시행되고 있는 롯데슈퍼의 모델을 분석한 뒤, 쿠팡에 적합한 형태로 구성해보았다. 개선하고자 했던 부분인 제품 처리 건수, 제품 피킹 시간, 고용인원의 결과는 아래와 같다.

	쿠팡 인력(기존)	롯데 슈퍼	자동화된 쿠팡
처리 개수(UPH)	약 50개	약 60개	약 240개
시간(UNIT당)	1.2min	0.96min	0.25min
하루 총 처리개수	약 8000개	약 9000개	약 9600개
고용인원	10명+a	1~2명 (유지/보수 인원)	
인건비(초기비용)	1,670,000+a	도입비용+78,000/일	도입비용+78,000/일

표 5 개선된 모델과 기존 인력, 롯데슈퍼 모델의 비교

		쿠팡 인력	롯데 슈퍼	자동화된 쿠팡
초기비용	고(10점)		●	●
	중(20점)			
	저(30점)	●		
인건비	고(10점)	●		
	중(20점)			
	저(30점)		●	●
처리개수	고(30점)			●
	중(20점)		●	
	저(10점)	●		
	총 점수	50	60	70

표 6 각 요소에 대한 점수 비교

우선, 위의 두 가지 표를 정리해보면 아래와 같다.

첫째, 처리 개수를 비교해보면 1시간에 정해진 개수를 처리한다는 가정하에 약 50개~60개로 동일하다. 1시간당 인력 10명, 피킹 로봇 19대, 피킹 로봇 5대가 정해진 개수를 찾는 것으로 본다.

둘째, 시간 당 1명, 1대당 1개의 제품을 피킹하는 데 걸리는 시간을 비교해본다면 사람은 1개를 피킹하는 데 1.2분이 소요되며 롯데슈퍼의 피킹 로봇은 0.96분, 자동화된 쿠팡의 피

킹 로봇은 0.25분이 소요된다. 자동화된 쿠팡의 시스템을 기준으로 인력모델에서 0.95분을 단축시켜 사실상 동일한 시간을 부여한 후 개수의 제한을 두지 않는다면 자동화된 피킹 로봇이 더 많은 제품을 피킹할 수 있다는 것을 도출해낼 수 있다. 또한, 롯데슈퍼는 19대의 로봇으로 진행되지만 자동화된 쿠팡 시스템은 5대의 로봇으로 피킹하기 때문에 적은 로봇 대수로 더 빠르게 제품을 피킹한다는 것을 알 수 있다.

셋째, 고용인원은 인력모델의 경우 임금의 변동, 작업자 고용수의 유동성으로 인해 고정인원인 10명에 +a 가 되는 반면에 롯데슈퍼의 인력은 고정된 관리인원 7명으로도 충분히 운영이 가능하다. 더불어 자동화된 쿠팡의 경우에도 물류센터 내에서 관리하는 인원을 입출고 담당자와 컨베이어벨트, 재고조달라인 6-8명으로 고정하여 운영이 가능하다. 즉, 기존 인력모델에 비해 자동화된 시스템에서는 유동적인 인원보다는 고정적 인원을 배치하기 때문에 인건비의 낭비를 줄일 수 있다는 것을 도출해낼 수 있다.

마지막으로 표6을 살펴보면 각 요소에 대한 점수를 기재하였다. 기존 인력 모델은 50점, 롯데슈퍼는 60점, 자동화된 쿠팡모델은 70점으로 가장 높은 점수가 계산되었다. 이를 통해 쿠팡에 가장 적합한 모델은 인력과 롯데슈퍼의 오토스토어 방식 그대로가 아닌 현재 우리가 고려하여 설계한 알고리즘이 가장 적합한 것을 알 수 있다.

결론적으로 완벽한 자동화 시스템이 아닌 롯데슈퍼의 시스템의 일부를 가져와 쿠팡의 현 물류시스템에 맞게 구축한다면, 초기 비용은 많으나 고용 인원의 유동성에 따른 인건비를 절감할 수 있고 정해진 시간 내에 피킹할 수 있는 제품의 수가 증가하여 물류센터 내의 효율성을 높일 수 있을 것이다.

6. 참고 문헌

● https://blog.lgcns.com/2014?category=515093, LG CNS 물류 자동화 트렌드, 2019

● http://m.snstimes.kr/view.asp?intNum=21631&ASection=001001,플랫폼 비즈니스 사례 분석, 이상일, SNS타임즈, 2019

● https://www.hankookilbo.com/News/Read/201902071634713068,쿠팡 물류 혁신의 비결, 뒤죽박죽 넣는 '랜덤스토우', 임소형, 한국일보, 2019

● https://blog.naver.com/PostView.nhn?blogId=ulogistics05&logNo=221559619005&parentCategoryNo=&categoryNo=9&viewDate=&isShowPopularPosts=true&from=search, '로봇이 피킹 'AutoStore' 구축…,물류매거진 취재부, 2019

● https://dashboard.jobs.go.kr/index/summary?pg_id=PSCT030300&data2=SCT030300&ct_type=run, 대한민국 일자리 상황판, 임금상승률, 2019

● https://www.mk.co.kr/news/business/view/2019/04/258740/, '인건비.적자 1조 쿠팡', 정열, 매일경제, 2019

● http://www.nlic.go.kr/nlic/InfraFacilityPalletizer.action, 국가물류통합정보센터, 물류설비, 2019

● 양정삼, 2013, 『스마트 공장의 여명』, 『한국 CAD/CAM 학회지』v.19,No.1,pp22-25

● 생산통제 14장 재고관리 Part1 강의노트. 2019

● 이창민, 2007, 『다탄성 Insole의 Workload 감소 효과에 관한 연구』, 『Journal of the Ergonomics Society of Korea 』, Vol 26, No.2, pp.157-165

● 김정훈 외 2명, 2016, 『편의점 유통물류센터의 AGV 도입에 대한 시뮬레이션 분석』, 『Jonal of the Korea Academia-Industrial cooperation Society』, Vol.17,No.6,pp61-69

● 홍현주.(2004).AGV시스템의 모델링 및 교통제어를 위한 Simulation Tool 개발.퍼지 및 지능시스템학회 논문지 Vol. 14,N0.4 . pp. 499-505