Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 10 , October - 2026

# IoT-Integrated Simulation-Based Assembly Line Balancing for Smart Manufacturing

*Rajaprasad V G*
Roll No. 24R241
Dept. of Robotics and Automation Engineering
PSG College of Technology
Coimbatore, India

*Krishna Harshit T*
Roll No. 24R257
Dept. of Robotics and Automation Engineering
PSG College of Technology
Coimbatore, India

*Prof. M P Anbarasi*
Faculty Guide
Dept. of Robotics and Automation Engineering
PSG College of Technology
Coimbatore, India

***Abstract*—This paper extends a validated simulation-based assembly line balancing study by adding a simulated Internet of Things (IoT) sensor feed to a FlexSim model. A Python generator produces timestamped cycle-time readings for each of 18 workstations to represent sensor telemetry. The readings are exported to Excel and imported into a FlexSim Global Table, which replaces the historical cycle-time distributions of the baseline as the source of processing times. The line structure and the daily requirement of 200 units are kept so that results remain comparable with the baseline and with three earlier FlexSim models. The paper reports (i) the verified baseline and Models 1–3, (ii) the sensor-feed pipeline with a justification of its design parameters, and (iii) a test of the feed over 30 independent replications, which shows that its statistics and the identity of the bottleneck (WS3) do not change from one replication to the next. A Python re-implementation of the line, which reproduces the FlexSim output of Models 1–3 (164, 189 and 189 units), is then driven by the feed over 30 replications. Sensor-driven inputs change output by less than one unit and mean staytime by about 0.1%, and WS3 remains the bottleneck in every replication. The FlexSim run driven by the sensor feed is still in progress and is reported as pending.**

***Keywords*—Assembly line balancing, smart manufacturing, discrete-event simulation, Internet of Things (IoT) sensor data integration, bottleneck identification, work-in-process, manpower utilization.**

## I. Introduction

Assembly lines are flow-oriented production systems used for high-volume manufacturing. An assembly line consists of workstations at which individual tasks are performed in sequence. Because the tasks are interdependent, precedence relationships determine their order. Assembly line balancing assigns tasks to workstations while satisfying the precedence constraints and ensuring that the required cycle time is not exceeded; classifications and solution methods are surveyed in [4–6]. The main objectives are to minimize the cycle time and the number of workstations, to maximize output, and to improve manpower and equipment utilization. In practice, workstation cycle time varies because of operator performance, working conditions, interruptions and machine condition. A simulation model that represents this behaviour can be used to study production flow and to evaluate balancing decisions before changes are made on the shop floor [7, 8, 17, 18].

A prior study considered an 18-workstation production line and used computerized simulation to investigate line balancing, work-in-process (WIP), manpower utilization, overtime and alternative production configurations [1]. That baseline relied on cycle-time observations fitted to statistical distributions with the ARENA Input Analyzer [1,9]. Smart-manufacturing and Industry 4.0 work proposes that such models should instead be fed by shop-floor data [11, 12, 14–16], and digital-twin studies describe the long-term goal of a continuously updated model [13].

*Contribution of this study.* Relative to the baseline [1], this paper makes the following contributions.

1. It replaces the fitted distributions by a per-workstation, timestamped sensor-style data feed that reaches the FlexSim model [10] through a documented Python → Excel → Global Table pipeline.
2. It keeps the layout, work schedule and 200-unit requirement unchanged, so that any later difference in output or staytime can be attributed to the input source alone.
3. It justifies the two parameters of the feed (50 readings per station and a 3% standard deviation) against the variability of the fitted baseline distributions, and tests the feed over repeated replications and over alternative parameter values (Section IV-F).
4. It defines two ways of linking the Global Table to the model (direct lookup and aggregated mean) and states in advance what each is expected to show.
5. It adds an independent Python discrete-event model of the line, checked against the FlexSim Models 1–3, and uses it to obtain replicated results for the sensor-driven input while the FlexSim run is being completed (Section IV-G).

The feed is synthetic. The study therefore does not claim to validate the method on a real sensor installation; it prepares and tests the integration mechanism that a real feed would use.

## II. Development of the Simulation Model

The production system contains 18 workstations divided into assembly-preparation, assembly and inspection sections. WS1–WS3 form the assembly-preparation section, WS4–WS14 the main assembly line (group A: WS4–WS9; group B: WS10–WS14), and WS15–WS18 the inspection line [1]. Each workstation requires at least one operator. Operators work from 8:00 a.m. to 5:30 p.m. with three scheduled breaks (9:30–10:00 a.m., 1:00–2:00 p.m. and 3:30–4:00 p.m.), and overtime of at most two hours is allowed [1]. Following [1], the normal working time of 10.5 h less 2 h of breaks leaves 8.5 h (30,600 s) available per day. The daily requirement is 200 units, which gives a takt time of $30{,}600/200 = 153$ s per unit. The 8.5 h available time and the 153 s takt time are the values used in [1] and are kept so that results remain comparable. The baseline model uses workstation cycle-time distributions obtained from production data and analysed with the ARENA Input Analyzer [1, 9].

### A. Simulation-Based Assembly Line Balancing

Each workstation is represented by its cycle-time behaviour and its required manpower. The model follows the production flow through the 18 workstations and records output, WIP, workstation utilization, cycle time and waiting conditions. Alternative configurations can be tested by changing overtime, manpower allocation, flexible work schedules, or the source of the cycle-time data itself.

Two time measures are used throughout and must not be confused. The *processing time* of a workstation is the time the operator and machine actually work on one item; in this paper it is the cycle time of Table I, or the value taken from the Global Table. The *staytime* of an item at a workstation is the total time between its arrival at and its



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 10 , October - 2026

departure from that workstation:

$$\text{staytime}_i = \text{waiting}_i + \text{processing}_i, \tag{1}$$

where waiting is the time the item spends queued or blocked because the next station is busy. Staytime therefore equals processing time only when no queue forms. The *mean average staytime* in Tables II and III is the arithmetic mean over the 18 workstations of their mean staytimes.

### B. IoT-Based Data Acquisition Pipeline

A Python script (`generate_sensor_data.py`) simulates IoT sensor telemetry for the 18 workstations. For each of 50 simulated timestamps it draws one cycle-time reading per workstation from a Gaussian distribution centred on that workstation's known mean cycle time (Table I) with a standard deviation of 3% of the mean. Each reading records a timestamp, a workstation identifier and a cycle time in seconds, giving 900 readings ($18 \times 50$). The data are exported to `iot_sensor_feed.xlsx` (columns `Timestamp`, `Workstation`, `CycleTime_s`), which matches the FlexSim Excel import format, and are imported into a Global Table as numeric data. Each workstation therefore has its own series of 50 timestamped readings from which it draws its processing time. Because a shift needs up to about 200 items per station, the lookup index is defined as $((n-1) \bmod 50)+1$ for the $n$-th item, so that each reading is reused about four times.

*Why 50 readings.* The number was chosen as a pilot size for three reasons. First, with 50 readings and a 3% standard deviation the standard error of a station mean is $3\%/\sqrt{50} = 0.42\%$ (0.74 s for WS3 and 0.62 s for WS4), so the aggregated mean lies within about $\pm 0.8\%$ of the Table I mean at 95% confidence. Second, 900 rows make a Global Table small enough to inspect manually. Third, the sample is well above the roughly 30 observations commonly needed for a stable sample mean. It is, however, smaller than the number of items processed in a shift, which is why readings are reused. Table V shows how the standard error changes for 25 to 200 readings.

*Why 3%.* The coefficients of variation (standard deviation divided by mean) of the fitted distributions in Table I range from 0.5% (WS3) to 3.1% (WS18), with a median of 1.6%. A 3% spread is therefore at the upper end of the variability seen in the production data. This makes the feed a deliberately conservative test: if the sensor-driven model is no more variable than the data on which the baseline was fitted, the feed should leave the baseline conclusions unchanged, and if it does change them, the cause is variability rather than a shift in the means. Table V shows how sensitive the bottleneck stations are to this choice.

*Limits of computer-generated data.* The readings are centred on the known means, so the feed cannot reveal drift, bias or errors in those means; it is consistent with the history by construction. A symmetric Gaussian, independent between readings and with the same relative spread at every station, does not reproduce the right-skewed, autocorrelated and sometimes outlying cycle times, the missing values, the clock jitter and the calibration drift of real sensors. The run driven by this feed therefore tests the integration mechanism and the effect of extra reading-to-reading variability. It does not test how well sensor data predict a real line.

The data flow is one-way: known mean cycle times produce the readings, the readings are saved to Excel, the Excel file is loaded into the Global Table, and the workstations read from the Global Table. The Python generator, the Excel export and the Global Table import together form the integration offered by this study.

Fig. 1 shows the import settings. The workbook and sheet are chosen, the first row is used as column headers, and the three columns are imported as values. The result is a Global Table with 900 rows, each containing one sensor reading. Each workstation takes its processing time from this table instead of from a fixed historical distribution.

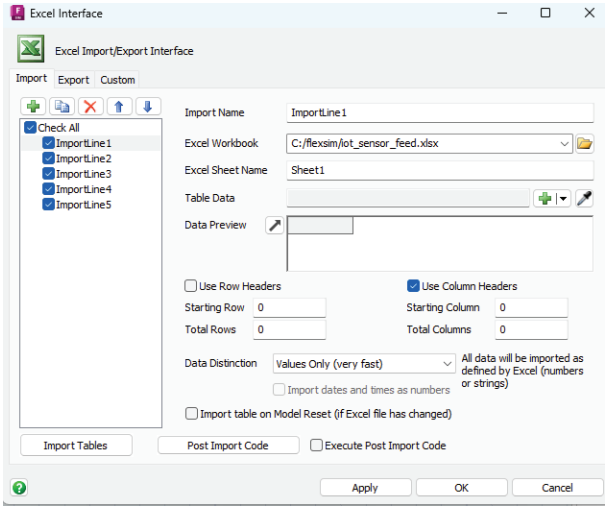



Fig. 1. FlexSim Excel Import/Export Interface configured to import the sensor dataset into the Global Table.

TABLE I
EXISTING WORKSTATION CYCLE-TIME DATA

| WS | Mean (s) | CT distribution |
|---|---|---|
| WS1 | 75.8 | 75.1 + 1.45*BETA(1.43,1.63) |
| WS2 | 85.7 | 84 + 2.96*BETA(1.17,0.951) |
| WS3 | 176.0 | TRIA(173,176,177) |
| WS4 | 146.0 | 143 + 5.72*BETA(1.07,0.9) |
| WS5 | 88.5 | 85 + 6*BETA(0.991,0.708) |
| WS6 | 83.8 | 82.1 + LOGN(1.75,1.44) |
| WS7 | 112.0 | 110 + LOGN(1.93,1.3) |
| WS8 | 113.0 | 109 + 7*BETA(0.893,0.848) |
| WS9 | 86.3 | TRIA(83.1,87.1,88.8) |
| WS10 | 69.1 | 66 + 5*BETA(0.835,0.525) |
| WS11 | 71.8 | TRIA(70.3,70.6,74.5) |
| WS12 | 88.6 | 85 + 6*BETA(1.1,0.777) |
| WS13 | 75.5 | UNIF(73,78) |
| WS14 | 85.1 | NORM(85.1,1.39) |
| WS15 | 73.1 | 71 + GAMM(0.989,2.11) |
| WS16 | 82.2 | UNIF(80,84) |
| WS17 | 86.7 | TRIA(83.2,88,89) |
| WS18 | 58.6 | 56.3 + LOGN(2.45,1.83) |

### C. Simulation Data Preparation

The input data come from two sources. The first is the set of cycle-time distributions kept from the baseline for comparison. The second is the simulated IoT sensor dataset. The preparation steps are: organizing the workstation cycle-time observations; generating and structuring the sensor dataset; importing it into the Global Table; assigning workstation references to the Global Table; defining production schedules; setting manpower requirements; and specifying that 200 units are required each day.

## III. EXISTING WORKSTATION CYCLE-TIME DATA

The cycle-time data from production records are kept as the baseline reference. Table I, reproduced from [1], lists the mean cycle time and the fitted ARENA distribution of each workstation. WS3 has the highest mean cycle time (176 s), and WS4 has the second highest (146 s). WS18 has the lowest (58.6 s). The remaining workstations have mean cycle times between 69 and 113 s. WS3 and WS4 form the basis of the bottleneck analysis, and their means are the values used to centre the simulated IoT readings in Section II-B.

Table I shows that the line is clearly unbalanced. The 18 mean cycle times add up to 1,657.8 s, an average of 92.1 s per station, yet the line can only move at the pace of WS3 (176 s). Line efficiency is



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 10 , October - 2026

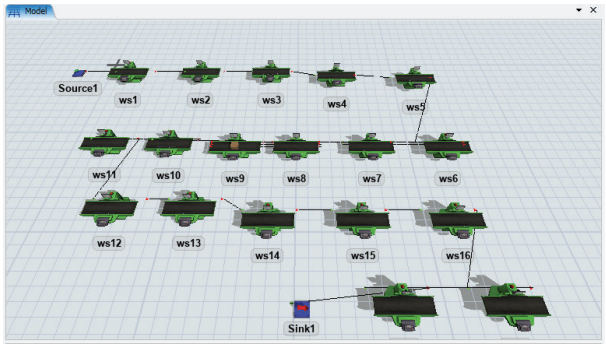



Fig. 2. FlexSim model layout of the 18-workstation assembly line. The layout is Source1 → WS1–WS18 → Sink1 and is used for both the baseline runs and the IoT-integrated runs.

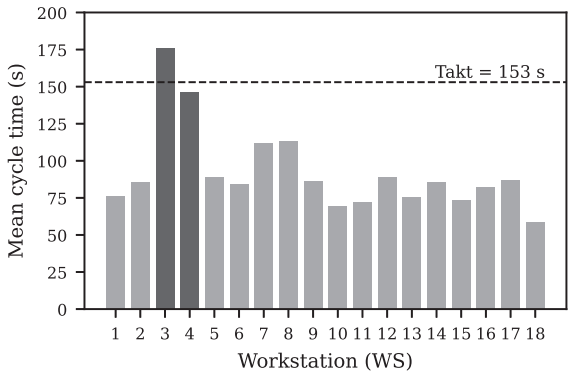



Fig. 3. Mean cycle time of each workstation (Table I) compared with the 153 s takt time.

therefore $1{,}657.8/(18 \times 176) = 52\%$, so nearly half of the operator time is idle. Most of the fitted distributions span only a few seconds, so the station times are almost deterministic, and the imbalance comes from differences between stations rather than within them. WS3 is 23 s slower than the 153 s takt time, and WS4 is 7 s faster. For this reason WS3 and WS4 are identified as the critical stations in Section IV.

### A. Simulation-Based Bottleneck Analysis

The cycle-time distributions are the basis for identifying bottlenecks. Workstations with high mean cycle times relative to the takt time are examined first. The analysis also considers waiting time, WIP accumulation, manpower availability, overtime and work schedules, all of which influence performance. Fig. 2 shows the FlexSim model of the line.

Items are released by Source1, move in sequence through WS1 to WS18, and leave the line at Sink1. Because cycle times differ, WIP builds up before stations such as WS3 and WS4. The baseline runs use the fitted distributions of Table I, whereas the IoT run reads processing times from the sensor Global Table. The stations are arranged in a folded path of connected conveyors to fit the floor area, the item flow is strictly serial, and each conveyor segment acts as a buffer. The same layout is used unchanged for both runs, so any change in output or staytime is due to the input data.

### B. Simulation Scenario Evaluation

Alternative balancing scenarios are evaluated by changing selected operating parameters. In this study the change is the substitution of the distribution-based cycle times by sensor-driven values imported from the IoT Global Table.

## IV. Results

### A. Verification

The ARENA baseline simulation of [1] was verified by comparing its output with the real output when 200 units were released. The real output was 168 units and the simulated output was 166 units. The difference of two units is an error of $2/168 = 1.19\%$, which gives a verification level of 98.81% [1], above the 95% level that [1] regards as acceptable. An error of this size shows that the fitted distributions, layout and work schedule reproduce the line well enough for comparison studies. It also means that the same model structure can be reused for the IoT run without re-calibration, because only the source of the processing times changes. The 166 units come from the ARENA model of [1], whereas the 164 units of Model 1 come from the FlexSim model of this study (Section IV-D). The two-unit difference may reflect differences between the two models and run-to-run variation, which is why replications are required (Section IV-E).

### B. Takt-Time Analysis

The baseline model assumes 8.5 h (30,600 s) of available time per day (10.5 h of normal working time less 2 h of breaks, as in [1]). For a requirement of 200 units the takt time is therefore 153 s per unit [1]. WS3 has a cycle time of 176 s, which is higher than the takt time, whereas WS4 has a mean of 146 s and is close to the takt limit. WS3 and WS4 are therefore the critical workstations in both the baseline and the IoT-integrated evaluation, because the simulated sensor readings for WS3 and WS4 are centred on the same means.

Fig. 3 plots the mean cycle time of every workstation against the 153 s takt time. Only WS3 lies above the line, by 23 s, so it cannot keep pace with one unit every 153 s. WS4 lies below the line by only 7 s, so any increase in its cycle time, such as the variation expected from real sensor readings, can push it above the takt as well. The remaining sixteen stations have mean cycle times between 58.6 s and 113 s and spend a large part of each cycle waiting for upstream work. Fig. 3 supports the adjustment of WS3 and WS4 in Models 2 and 3.

### C. WIP and Manpower Utilization

WIP represents partially completed products that are between production stages. In the baseline, differences in processing times cause WIP to build up at some locations while other workstations remain under-utilised [1]. The baseline study developed six configurations (A–F) based on working hours, manpower, flexibility and scheduling [1]; these remain the reference configurations against which the IoT-integrated run is compared. In configuration A (normal working hours) the ARENA run of [1] ended the day with 34 units of WIP (200 released, 166 completed), 27 of them at WS3. In the FlexSim models the same congestion appears as fewer items released by Source1 (Section IV-E).

The manpower load of a station can be estimated by dividing its cycle time by the takt time. This is a load indicator derived from Table I, not a utilization measured in the simulation. On this basis WS3 is loaded at 115% and WS4 at about 95%, while WS7 and WS8 are at about 73% and 74%. All other stations lie between 38% (WS18) and 58% (WS12), and the line average is about 60%. Unused manpower at stations below the takt is what line balancing tries to recover. WS3 cannot be loaded beyond 100% and therefore limits the line. In 30,600 s of working time, WS3 can complete at most $30{,}600/176 \approx 174$ units. If the line starts empty, the first unit needs about 1,658 s (the sum of all cycle times) to cross the line, so the ceiling falls to about $1 + (30{,}600 - 1{,}658)/176 \approx 165$ units. This is consistent with the 164–168 units seen in the baseline.

### D. FlexSim Model Comparisons (Baseline)

Before the IoT integration, three FlexSim configurations (Models 1–3) were evaluated. They used adjusted processing times at the



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 10 , October - 2026

TABLE II
WORKSTATION STAYTIME COMPARISON: MODELS 1–3 AND IOT-INTEGRATED MODEL

| WS | Model 1 (s) | Model 2 (s) | Model 3 (s) | IoT-Py (s) |
|---|---|---|---|---|
| WS1 | 172.00 | 75.00 | 75.00 | 162.67 |
| WS2 | 174.71 | 85.70 | 85.70 | 170.43 |
| WS3 | 176.00 | 140.00 | 140.00 | 175.96 |
| WS4 | 146.00 | 146.00 | 120.00 | 145.97 |
| WS5 | 88.50 | 88.50 | 88.50 | 88.42 |
| WS6 | 83.80 | 83.80 | 83.80 | 83.75 |
| WS7 | 112.00 | 112.00 | 112.00 | 111.99 |
| WS8 | 113.00 | 113.00 | 113.00 | 113.12 |
| WS9 | 86.30 | 86.30 | 86.30 | 86.36 |
| WS10 | 69.10 | 69.10 | 69.10 | 69.13 |
| WS11 | 71.80 | 71.80 | 71.80 | 71.76 |
| WS12 | 88.60 | 88.60 | 88.60 | 88.47 |
| WS13 | 75.50 | 75.50 | 75.50 | 75.54 |
| WS14 | 85.10 | 85.10 | 85.10 | 85.05 |
| WS15 | 73.10 | 73.10 | 73.10 | 73.04 |
| WS16 | 82.10 | 82.10 | 82.10 | 82.12 |
| WS17 | 86.70 | 86.70 | 86.70 | 86.62 |
| WS18 | 58.60 | 58.60 | 58.60 | 58.61 |

*Mean staytime per item. IoT-Py is the mean of 30 replications of the Python model of Section IV-G driven by the sensor feed (direct lookup, baseline configuration). The FlexSim IoT run is still pending (Section IV-E). Where no queue forms, staytime equals processing time, so for most stations the values equal the cycle times of Table I. WS16 is entered as 82.10 s in the FlexSim models and appears as 82.2 s in Table I.

bottleneck stations (WS1–WS4), while all other stations kept the baseline values. Table II compares staytimes across these models and the IoT model, which will be completed once the IoT run is finished. In Model 1, WS1–WS3 have staytimes of 172.00, 174.71 and 176.00 s. In Model 2 the values for WS1–WS3 drop to 75.00, 85.70 and 140.00 s, and in Model 3 WS4 also drops from 146.00 s to 120.00 s. Output rose from 164 units in Model 1 to 189 units in Models 2 and 3, an increase of 15.24%, while the mean average staytime fell from 102.38 s to 90.05 s and 88.61 s, a reduction of 13.45% for Model 3 [2, 3].

Models 2 and 3 differ only in the processing time of WS4, which allows the effect of relieving the second-slowest station to be separated from the effect of relieving WS3. In all three models the stations WS5 to WS18 are unchanged, so the comparison isolates the effect of adjusting the stations at the start of the line.

The improvement in Models 2 and 3 can be traced to the stations at the front of the line. In Model 1, WS1 and WS2 spend most of their staytime waiting behind WS3. Reducing the staytimes of WS1 to WS3 therefore releases items to the downstream stations and raises output from 164 to 189 units. Reducing WS4 in Model 3 does not raise output further, because WS3, at 140.00 s, remains the slowest station, but it lowers the mean average staytime from 90.05 s to 88.61 s.

Table II shows how staytime at each station changes when the bottleneck stations are adjusted. In Model 1, WS1 and WS2 have staytimes of 172.00 and 174.71 s, far above their own cycle times of 75.8 and 85.7 s. The difference is waiting time (Eq. 1): items are delayed by WS3, not by WS1 or WS2. Model 2 lowers WS3 to 140 s, below the 153 s takt time, but WS4 at 146.00 s then becomes the slowest station. Model 3 cuts WS4 to 120.00 s, so WS3 at 140.00 s is again the slowest, now below takt. WS5 through WS18 stay the same in all three models, so every change in output and mean staytime comes from WS1 through WS4.

### *E. IoT-Integrated Simulation (In Progress)*

The IoT pipeline of Section II-B is in place. The Python generator creates a dataset of 900 readings, which is exported to Excel and imported into FlexSim. At the time of writing, the connection between the processing-time parameters and the sensor table is still being finished, so no FlexSim result for the sensor-driven run is available yet. Replicated results from an independent Python model of the line are given in Section IV-G and are labelled IoT-Py in Tables II and III.

Two methods are being tested for linking the table to workstation behaviour: (i) a direct per-item lookup, where each processing time is taken from the workstation's rows in the Global Table while the simulation runs; and (ii) an aggregated approach, where each processing time is set to the average `CycleTime_s` of the station's readings, calculated with AVERAGEIF. Both methods use the same data. The direct lookup keeps the variability of each reading, whereas the aggregated approach provides a single value per station.

The 3% standard deviation matters mainly for the bottleneck stations. For WS3 (176 s) the deviation is 5.3 s, so every reading is above the 153 s takt time ($z = -4.4$). For WS4 (146 s) the deviation is 4.4 s, so about 5.5% of readings are above takt. This spread is larger than that of the TRIA distribution fitted to WS3, whose standard deviation is about 0.9 s. The IoT run will therefore show whether realistic sensor variability changes the queueing and staytime at WS3 and WS4.

The two approaches are expected to behave differently. With 50 readings per station and a 3% standard deviation, the standard error of the mean is 0.42% (Table IV confirms 0.72 s for WS3 and 0.57 s for WS4). The aggregated approach should therefore reproduce the means of Table I and stay close to the baseline. The direct lookup carries the variation from one reading to the next into the simulation and is the method that can reveal whether sensor variability changes WIP and staytime at the bottleneck stations.

For the final results, each method will be run in at least 10 FlexSim replications with different random streams. Output, mean average staytime and bottleneck station will be reported as means with 95% confidence intervals, and a difference from Models 1–3 will be called real only if the intervals separate. The FlexSim values will then be added beside the IoT-Py values.

Table III condenses these results. The initial input differs between models because Source1 can release an item only when the first conveyor has space. In Model 1 the slow WS3 (176 s) makes the queue behind it fill the upstream buffers, which blocks Source1 and limits it to 175 items. In Models 2 and 3 the first stations are fast enough to keep the buffers clear, so Source1 releases 199 items, close to the 200-unit requirement. The difference between input and final output (11 and 10 items) is WIP that has not yet left the line at the end of the shift.

When the model changes from Model 1 to Model 2, the final output increases from 164 to 189 units, a gain of 25 units or 15.24%. At the same time the mean average staytime falls from 102.38 s to 90.05 s, a decrease of 12.04%. Model 3 does not alter the output but reduces the mean average staytime further to 88.61 s, which is 13.45% lower than Model 1. The longest staytime shifts from 176 s at WS3 to 146 s at WS4 and then to 140 s, back at WS3. This

TABLE III
OVERALL MODEL COMPARISON SUMMARY

| Parameter | Model 1 | Model 2 | Model 3 | IoT-Py |
|---|---|---|---|---|
| Initial input (items released) | 175 | 199 | 199 | 177.6 |
| Final output (units) | 164 | 189 | 189 | 164.2 |
| WIP left at end of shift (items) | 11 | 10 | 10 | 13.4 |
| Mean average staytime (s) | 102.38 | 90.05 | 88.61 | 101.61 |
| Maximum staytime (s) | 176 (WS3) | 146 (WS4) | 140 (WS3) | 176.0 (WS3) |
| Output improvement vs Model 1 | – | 15.24% | 15.24% | 0.1% |

*IoT-Py: mean of 30 replications of the Python model (Section IV-G), direct lookup, baseline configuration (comparable with Model 1). The FlexSim IoT run is still pending. Initial input is the number of items released by Source1 in the shift; WIP at end of shift = initial input − final output.



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 10 , October - 2026

TABLE IV
STATISTICS OF THE SENSOR FEED OVER 30 REPLICATIONS

| Quantity | Mean | SD | Min | Max |
|---|---|---|---|---|
| WS3 sample mean (s) | 175.92 | 0.72 | 174.48 | 177.24 |
| WS4 sample mean (s) | 145.98 | 0.57 | 144.10 | 147.26 |
| WS3 sample SD (s) | 5.27 | 0.39 | 4.46 | 6.29 |
| WS4 sample SD (s) | 4.40 | 0.48 | 3.29 | 5.77 |
| WS3 readings above takt (of 50) | 50.0 | 0.0 | 50 | 50 |
| WS4 readings above takt (of 50) | 2.6 | 1.5 | 0 | 5 |
| Sum of 18 station means (s) | 1657.2 | 1.8 | 1653.0 | 1659.8 |
| Largest station-mean deviation from Table I (%) | 0.86 | 0.20 | 0.54 | 1.30 |
| Output ceiling, 30,600/ max(mean) (units) | 173.9 | 0.72 | 172.6 | 175.4 |

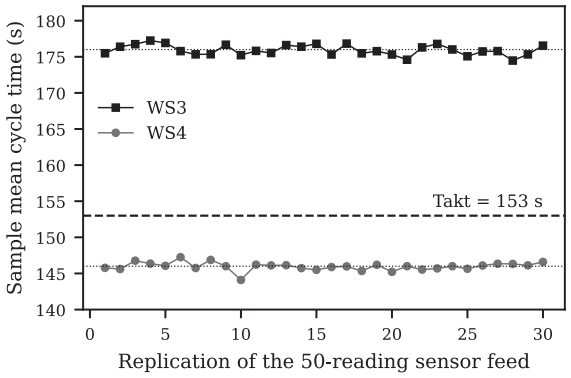



Fig. 4. Sample mean cycle time of WS3 and WS4 for each of the 30 replications of the sensor feed. Dotted lines are the Table I means; the dashed line is the 153 s takt time.

shows that the bottleneck moves to the slowest remaining station whenever a station is improved. Because Models 2 and 3 produce 189 units from an input of 199, the output remains below the required 200 units, so further balancing or a limited amount of overtime is necessary.

### F. Replication Test of the Sensor Feed

The feed was generated 30 times with independent random seeds (1–30), each time with 50 readings per station and a 3% standard deviation, to see whether its statistics change from one replication to the next. Table IV summarizes the 30 replications and Fig. 4 shows the sample means of WS3 and WS4.

The results show that the feed is stable. The sample mean of every station stayed within 1.3% of its Table I mean in all 30 replications, and the sum of the 18 means varied by only $\pm 3$ s around 1,657 s. WS3 was the slowest station in 30 of 30 replications and its mean never fell to the takt time, and the WS4 mean never exceeded the takt time (0 of 30). The number of WS4 readings above takt varied between 0 and 5 of 50, which matches the expected 5.5% per reading ($7/4.38 = 1.6$ standard deviations). The output ceiling implied by the slowest station moved by less than $\pm 2$ units. Replication therefore changes the individual readings but not the conclusions that can be drawn from the feed.

Table V repeats the test with 1,000 replications for other feed sizes and spreads. Increasing the number of readings from 50 to 200 lowers the standard error of the WS3 mean from 0.74 s to 0.38 s but leaves the share of WS4 readings above takt at about 5.5%. The spread has a much larger effect: at 1% WS4 never exceeds takt, whereas at 5% about 17% of its readings do. The 3% setting is

TABLE V
SENSITIVITY TO NUMBER OF READINGS AND SPREAD (1,000 REPLICATIONS)

| Readings per WS | Spread (% of mean) | SE of WS3 mean (s) | SE of WS4 mean (s) | WS4 above takt (%) |
|---|---|---|---|---|
| 25 | 3 | 1.07 | 0.88 | 5.5 |
| 50 | 3 | 0.74 | 0.61 | 5.5 |
| 100 | 3 | 0.53 | 0.44 | 5.6 |
| 200 | 3 | 0.38 | 0.32 | 5.5 |
| 50 | 1 | 0.24 | 0.21 | 0.0 |
| 50 | 5 | 1.24 | 1.05 | 16.7 |

TABLE VI
PYTHON CROSS-CHECK AGAINST FLEXSIM (DETERMINISTIC TIMES)

| Model | Source | Input | Output | Staytime (s) |
|---|---|---|---|---|
| Model 1 | FlexSim | 175 | 164 | 102.38 |
| | Python | 178 | 164 | 101.63 |
| Model 2 | FlexSim | 199 | 189 | 90.05 |
| | Python | 199 | 189 | 90.06 |
| Model 3 | FlexSim | 199 | 189 | 88.61 |
| | Python | 199 | 189 | 88.61 |

therefore the lowest level at which the sensor-driven run can show WS4 crossing the takt, which is the case of interest.

This test applies to the data feed only. It does not replace replications of the FlexSim run itself, which are described in Section IV-E.

### G. Python Cross-Check Model and Sensor-Driven Results

To obtain replicated results for the sensor-driven input, the line was also implemented as a Python discrete-event model (`line_sim.py`). It is a serial line of 18 stations with blocking after service and no buffer between stations. Source1 releases one item every 153 s (the takt time), starting at 153 s, and holds an item back while WS1 is occupied. Staytime is measured from the start of service at a station until the item leaves it, so it includes blocking. The model has no parameter fitted to FlexSim output.

*Cross-check against FlexSim.* With the deterministic times of Models 1–3, the Python model gives the same final output as FlexSim in all three cases (Table VI). The input and the mean average staytime of Models 2 and 3 also agree (90.06 s and 88.61 s against 90.05 s and 88.61 s). Model 1 agrees on output but not exactly on the other measures: the model releases 178 items instead of 175, and its mean average staytime is 101.63 s instead of 102.38 s (0.7% lower). The Python model with the fitted Table I distributions gives 165 units against 166 for the ARENA model of [1] and 168 for the real line, which is a verification level of 98.2% against the real line. The model is therefore adequate for comparing input sources, but it is not a substitute for the FlexSim results, and Model 1 shows that small differences in blocking behaviour remain.

*Sensor-driven runs.* The baseline configuration was then run 30 times for each of three inputs: the fitted Table I distributions, the sensor feed with direct lookup (item $n$ uses reading $((n-1) \bmod 50)+1$), and the sensor feed with the aggregated mean. The feed of replication $r$ is the one with seed $r$ used in Table IV. Table VII gives means with 95% confidence intervals.

The sensor-driven runs lose 0.8 units of output relative to the fitted distributions on average (paired 95% CI $\pm 0.3$ units) and raise the mean average staytime by 0.12 s ($\pm 0.06$ s), or about 0.1%. These differences are statistically detectable because the fitted run is almost deterministic, but they are far too small to change a balancing decision. The direct lookup and the aggregated mean are indistinguishable at this level. WS3 is the bottleneck in all 90 runs. WS4 exceeds the takt time in about 5.5% of its readings, yet its staytime stays at 146.0 s because it lies downstream of WS3 and is starved rather than queued. In this configuration, therefore, adding realistic reading-to-reading variability of 3% does not alter output, staytime



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 10 , October - 2026

TABLE VII
PYTHON MODEL, 30 REPLICATIONS (MEAN ± 95% CI)

| Measure | Fitted | IoT direct | IoT aggregated |
|---|---|---|---|
| Items released | 178.0±0.0 | 177.6±0.3 | 177.6±0.3 |
| Final output | 165.0±0.0 | 164.2±0.3 | 164.2±0.3 |
| Output range | 165–165 | 163–166 | 163–166 |
| Mean avg. staytime (s) | 101.49±0.01 | 101.61±0.06 | 101.59±0.06 |
| Max. staytime (s) | 175.32±0.02 | 175.96±0.28 | 175.92±0.27 |
| WS3 is bottleneck | 30/30 | 30/30 | 30/30 |

or the bottleneck. The result is specific to the one-day horizon, the 3% spread and the Python model, and it should be confirmed with the FlexSim runs.

### H. Summary of Results

The baseline model matched the production closely, producing 166 units against the real 168 units, a verification level of 98.81%. With a takt time of 153 s, WS3 (176 s) is the bottleneck and WS4 (146 s) is close to its limit. As Table III shows, output climbs from 164 units in Model 1 to 189 units in both Model 2 and Model 3, an increase of 15.24%, while the mean average staytime drops from 102.38 s in Model 1 to 90.05 s in Model 2 and 88.61 s in Model 3, a reduction of 13.45%. The longest staytime moves from WS3 to WS4 and then back to WS3. The replication test shows that the sensor feed itself is stable across seeds, and 30 replications of the Python model show that sensor-driven inputs change output by under one unit and staytime by about 0.1% (Table VII). The FlexSim sensor-driven results will be added beside these values once the Global Table linkage is completed.

### I. Discussion and Limitations

The output of the production line is governed by a handful of stations. When WS3 and WS4 are improved, output grows by 15.24% while the remaining sixteen stations stay unchanged, and the bottleneck shifts to the slowest station each time. This illustrates why sensor data are linked to the model: if the cycle time of a station starts to drift, the model can be rerun to determine whether that station becomes the new constraint before any impact reaches the shop floor. The pipeline presented here is a step toward continuous monitoring.

Several limitations apply. The readings are generated by a script from the known mean cycle times, so they are not measurements taken from a line. A Gaussian assumption with a fixed 3% deviation may not match the noise that real sensors produce (Section II-B). The dataset contains 50 timestamps per workstation and is reused cyclically during a shift. The data flow is one-way, with no feedback from the simulation to the sensor source. The model covers one production day, so longer-term drift cannot yet be examined. The Python model does not reproduce Model 1 exactly (Table VI), and the replication tests of Sections IV-F and IV-G do not replace FlexSim replications; the FlexSim runs must be replicated as described in Section IV-E before any difference from the baseline can be separated from random variation.

## V. CONCLUSION

This study builds on a simulation of an 18-workstation assembly line by adding a simulated IoT sensor data pipeline. Cycle-time telemetry for each workstation is produced in Python, exported to Excel, and imported into a FlexSim Global Table that is intended to provide the processing time of each workstation. The 18-workstation layout, the daily requirement of 200 units and the baseline verification level of 98.81% are carried over from the earlier study [1], which allows a direct comparison.

The completed tests support the following conclusions. The baseline Models 1–3 showed that adjusting the bottleneck workstations (WS1–WS4) raised output by 15.24% and cut the mean average staytime by up to 13.45% [2, 3]. The sensor feed, tested over 30 replications and over alternative sizes and spreads, is statistically stable: station means stay within 1.3% of the historical means, WS3 is always the slowest station, and WS4 exceeds the takt time in about 5.5% of readings at the chosen 3% spread. An independent Python model that reproduces the FlexSim output of Models 1–3 shows, over 30 replications, that sensor-driven inputs change output by less than one unit and mean staytime by about 0.1%, and leave WS3 as the bottleneck. No conclusion is drawn from the FlexSim sensor-driven run itself, because that run is not yet complete.

Future work will finish the sensor-driven FlexSim run with both the direct-lookup and the aggregated approaches, using at least 10 replications of each, so that true differences can be separated from random variation. Real sensor data taken from a production line, other variability models (skewed, autocorrelated) and a feedback path from the simulation to the data source can then be examined. Extending the pipeline to accept streaming data would allow the model to be updated continuously and would show, before any change reaches the shop floor, whether a drifting station has become the bottleneck.

## REFERENCES

[1] I. B. Mohd and A. Ibrahim, "A case study of production line balancing with simulation," *Int. J. Ind. Manage. (IJIM)*, vol. 2, pp. 15–26, Jun. 2016, Universiti Malaysia Pahang, Malaysia, ISSN 2289-9286.

[2] K. Harshit T and R. V. G, "FlexSim simulation results: Model 1–3 comparison tables and graphs," unpublished simulation data, Dept. Robotics Autom. Eng., PSG College of Technology, Coimbatore, India, 2026.

[3] K. Harshit T and R. V. G, "FlexSim simulation results: Final quantitative results for Models 1–3," unpublished simulation data, Dept. Robotics Autom. Eng., PSG College of Technology, Coimbatore, India, 2026.

[4] N. Boysen, M. Fliedner, and A. Scholl, "A classification of assembly line balancing problems," *Eur. J. Oper. Res.*, vol. 183, no. 2, pp. 674–693, 2007.

[5] A. Scholl and C. Becker, "State-of-the-art exact and heuristic solution procedures for simple assembly line balancing," *Eur. J. Oper. Res.*, vol. 168, no. 3, pp. 666–693, 2006.

[6] C. Becker and A. Scholl, "A survey on problems and methods in generalized assembly line balancing," *Eur. J. Oper. Res.*, vol. 168, no. 3, pp. 694–715, 2006.

[7] A. M. Law, *Simulation Modeling and Analysis*, 5th ed. New York, NY, USA: McGraw-Hill, 2015.

[8] J. Banks, J. S. Carson II, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, 5th ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.

[9] W. D. Kelton, R. P. Sadowski, and N. B. Swets, *Simulation with Arena*, 5th ed. New York, NY, USA: McGraw-Hill, 2010.

[10] FlexSim Software Products, Inc., "FlexSim 3D simulation modeling and analysis software – user manual," Orem, UT, USA. [Online]. Available: https://www.flexsim.com

[11] L. D. Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[12] F. Tao, Q. Qi, A. Liu, and A. Kusiak, "Data-driven smart manufacturing," *J. Manuf. Syst.*, vol. 48, pp. 157–169, 2018.

[13] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," *IEEE Trans. Ind. Informat.*, vol. 15, no. 4, pp. 2405–2415, Apr. 2019.

[14] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, "Intelligent manufacturing in the context of Industry 4.0: A review," *Engineering*, vol. 3, no. 5, pp. 616–630, 2017.

[15] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Bus. Inf. Syst. Eng.*, vol. 6, no. 4, pp. 239–242, 2014.

[16] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for Industry 4.0-based manufacturing systems," *Manuf. Lett.*, vol. 3, pp. 18–23, 2015.



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 10 , October - 2026

[17] A. Negahban and J. S. Smith, "Simulation for manufacturing system design and operation: Literature review and analysis," *J. Manuf. Syst.*, vol. 33, no. 2, pp. 241–261, 2014.

[18] M. Jahangirian, T. Eldabi, A. Naseer, L. K. Stergioulas, and T. Young, "Simulation in manufacturing and business: A review," *Eur. J. Oper. Res.*, vol. 203, no. 1, pp. 1–13, 2010.



(This work is licensed under a Creative Commons Attribution 4.0 International License.)