US011774933B2

(12) **United States Patent**
Mast

(10) **Patent No.: US 11,774,933 B2**
(45) **Date of Patent: Oct. 3, 2023**

(54) **STAGE AUTOMATION SYSTEM**

(71) Applicant: **Exato IP LLC**, Lancaster, PA (US)

(72) Inventor: **Ryan Mast**, Lancaster, PA (US)

(73) Assignee: **Exato IP LLC**, Lancaster, PA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 149 days.

(21) Appl. No.: **17/469,583**

(22) Filed: **Sep. 8, 2021**

(65) **Prior Publication Data**

US 2022/0043415 A1 Feb. 10, 2022

Related U.S. Application Data

(63) Continuation of application No. PCT/US2020/046476, filed on Aug. 14, 2020, which is a continuation of application No. 16/751,984, filed on Jan. 24, 2020, now Pat. No. 11,385,610.

(60) Provisional application No. 63/075,616, filed on Sep. 8, 2020, provisional application No. 62/938,118, filed on Nov. 20, 2019, provisional application No. 62/887,998, filed on Aug. 16, 2019.

(51) **Int. Cl.**
G05B 19/042 (2006.01)

(52) **U.S. Cl.**
CPC .. ***G05B 19/042*** (2013.01); *G05B 2219/25011* (2013.01)

(58) **Field of Classification Search**
CPC G05B 19/042; G05B 2219/25011; G05B 2219/2664; G05B 19/0426
USPC 294/82.12; 472/78
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2011/0055231 A1 3/2011 Huck et al.
2015/0231524 A1 8/2015 Fisher
2017/0060979 A1 3/2017 Webster et al.
(Continued)

FOREIGN PATENT DOCUMENTS

CN 208130517 U 11/2018

OTHER PUBLICATIONS

English Machine Translation of Chen Wenhui, "Stage Convenient to Scene Switch", Nov. 23, 2018, Publication Number #208130517 (Year:2018).

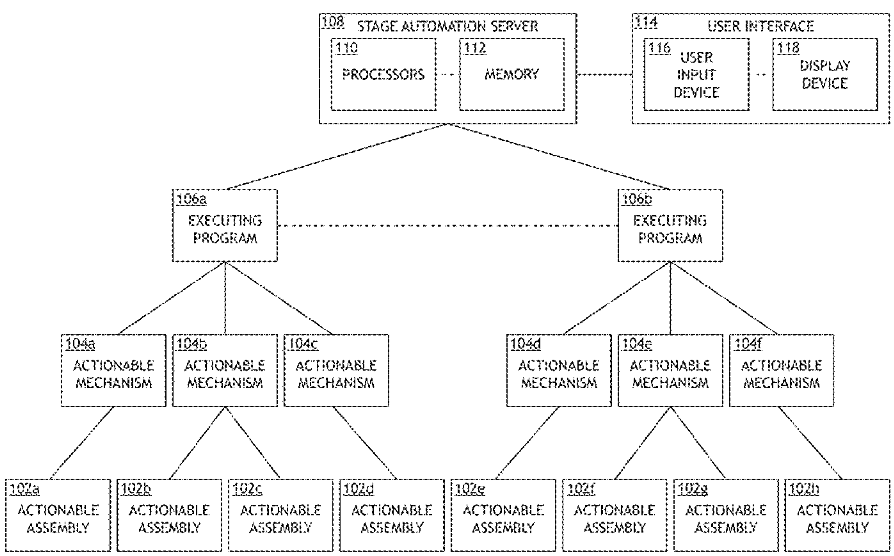
(Continued)

Primary Examiner — Mohammad Ali
Assistant Examiner — Sheela Rao
(74) *Attorney, Agent, or Firm* — Suiter Swantz pc llo

(57) **ABSTRACT**

A stage automation system, may include: at least one processor device implementing a physics engine; at least one memory device; one or more instructions stored in the memory device that, when executed by the at least one processor device implementing a physics engine, configure the at least one processor device for: receiving at least one target case corresponding to a user-interface specified position of a virtual object representative of a real-world object within a virtual space representative of a real-world space; providing position data defining the target case as a seed value to a physics engine; computing at least one effect-of-gravity solution from the seed value; comparing the effect-of-gravity position solution to the target case to determine if the effect-of-gravity position is within one or more position tolerance values relative to the target case; and providing user notification indicative of the comparison between the effect-of-gravity solution and the target case.

16 Claims, 44 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

2019/0352144	A1*	11/2019	Huusko	G06T 7/248
2022/0185639	A1*	6/2022	Post	B66D 3/18

OTHER PUBLICATIONS

Extended European Search Report dated Jul. 6, 2023; European Application No. 20854298.5.

* cited by examiner

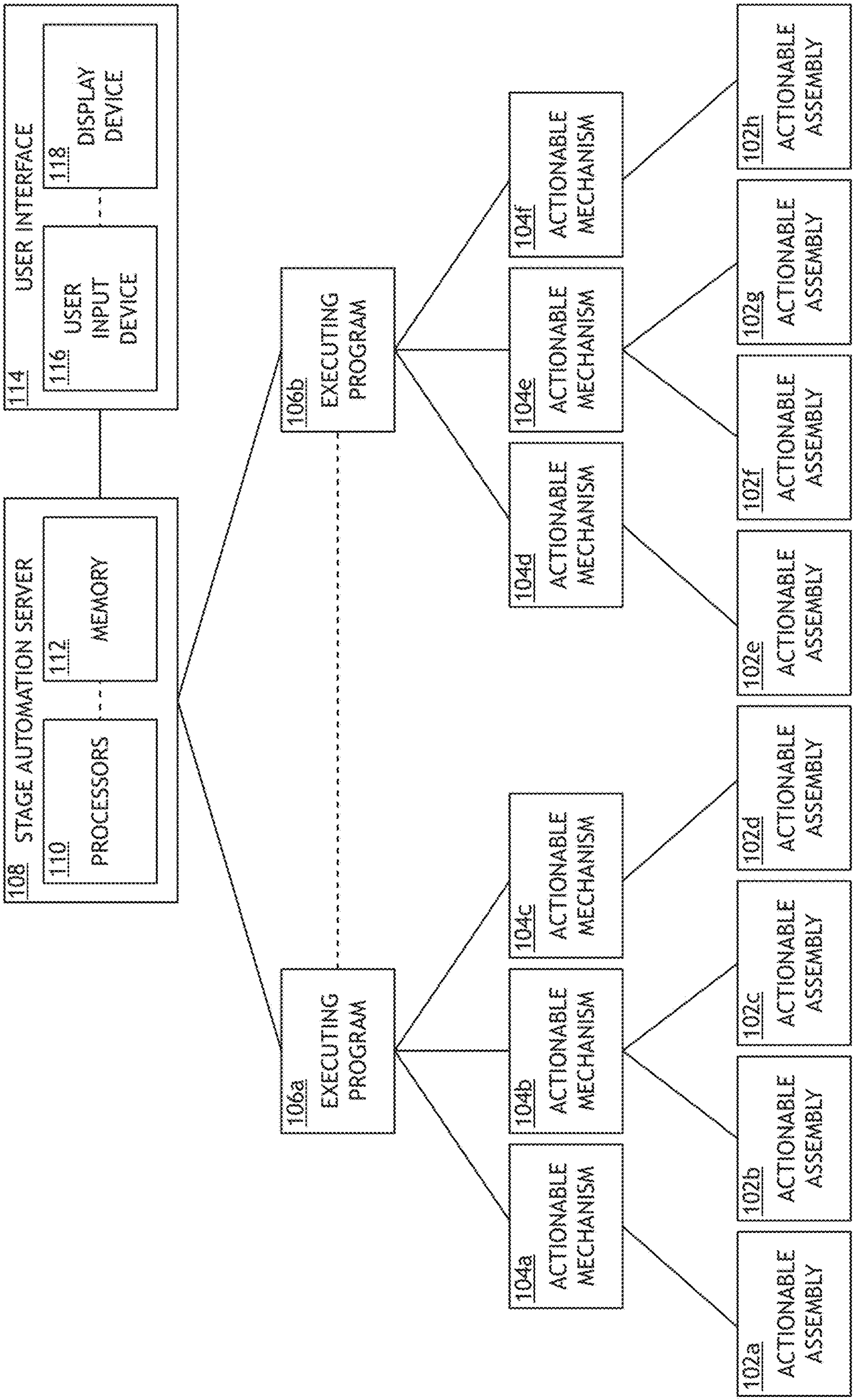

114 USER INTERFACE
116 USER INPUT DEVICE
118 DISPLAY DEVICE
108 STAGE AUTOMATION SERVER
110 PROCESSORS
112 MEMORY
106a EXECUTING PROGRAM
106b EXECUTING PROGRAM
104a ACTIONABLE MECHANISM
104b ACTIONABLE MECHANISM
104c ACTIONABLE MECHANISM
104d ACTIONABLE MECHANISM
104e ACTIONABLE MECHANISM
104f ACTIONABLE MECHANISM
102a ACTIONABLE ASSEMBLY
102b ACTIONABLE ASSEMBLY
102c ACTIONABLE ASSEMBLY
102d ACTIONABLE ASSEMBLY
102e ACTIONABLE ASSEMBLY
102f ACTIONABLE ASSEMBLY
102g ACTIONABLE ASSEMBLY
102h ACTIONABLE ASSEMBLY

FIG. 1

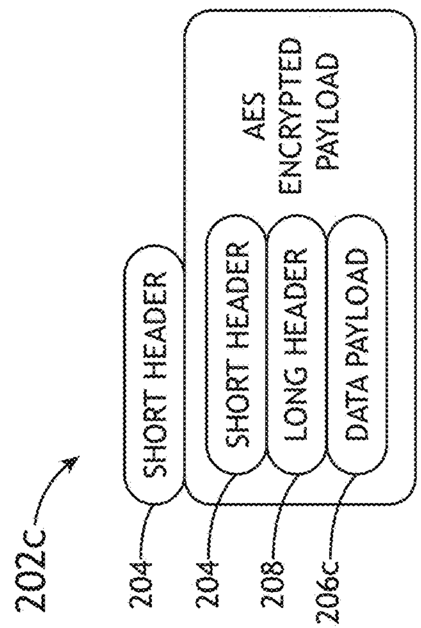


FIG. 2C

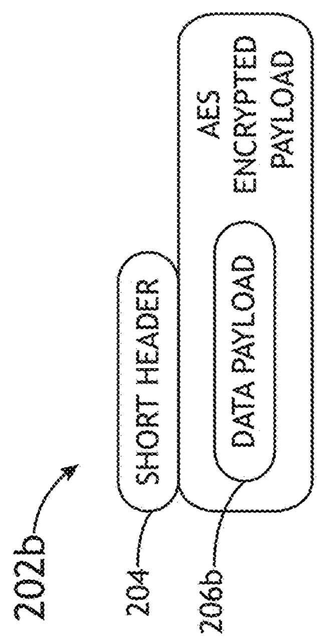


FIG. 2B

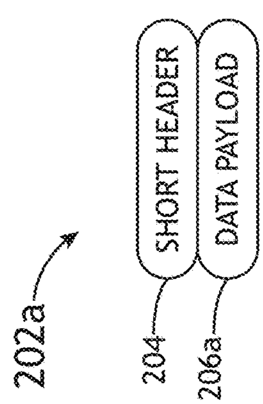


FIG. 2A

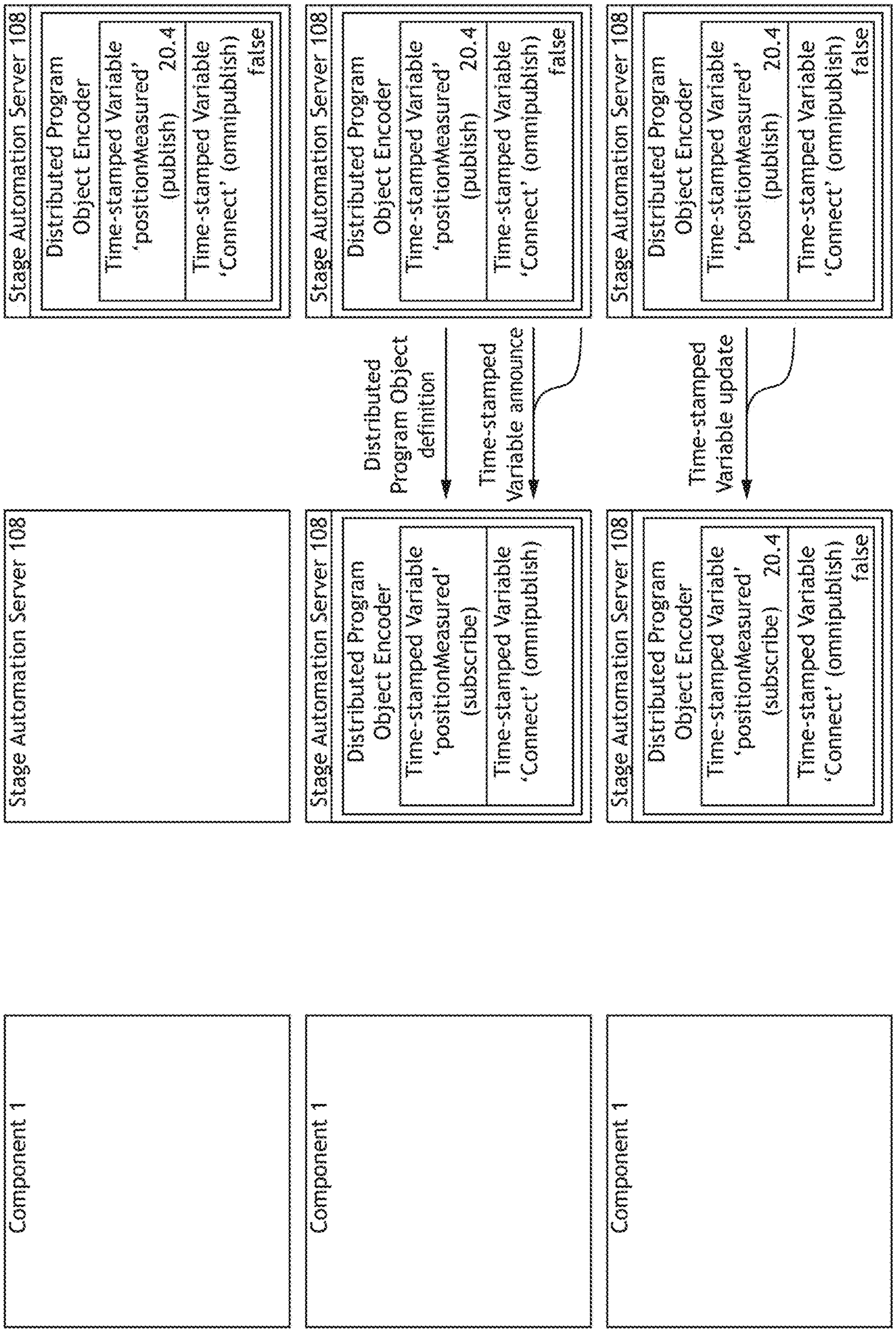

Component 1
Stage Automation Server 108
Stage Automation Server 108
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (publish) 20.4
Time-stamped Variable 'Connect' (omnipublish) false
Component 1
Stage Automation Server 108
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (subscribe)
Time-stamped Variable 'Connect' (omnipublish)
Distributed Program Object definition
Time-stamped Variable announce
Stage Automation Server 108
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (publish) 20.4
Time-stamped Variable 'Connect' (omnipublish) false
Component 1
Stage Automation Server 108
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (subscribe) 20.4
Time-stamped Variable 'Connect' (omnipublish) false
Time-stamped Variable update
Stage Automation Server 108
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (publish) 20.4
Time-stamped Variable 'Connect' (omnipublish) false

FIG. 3A

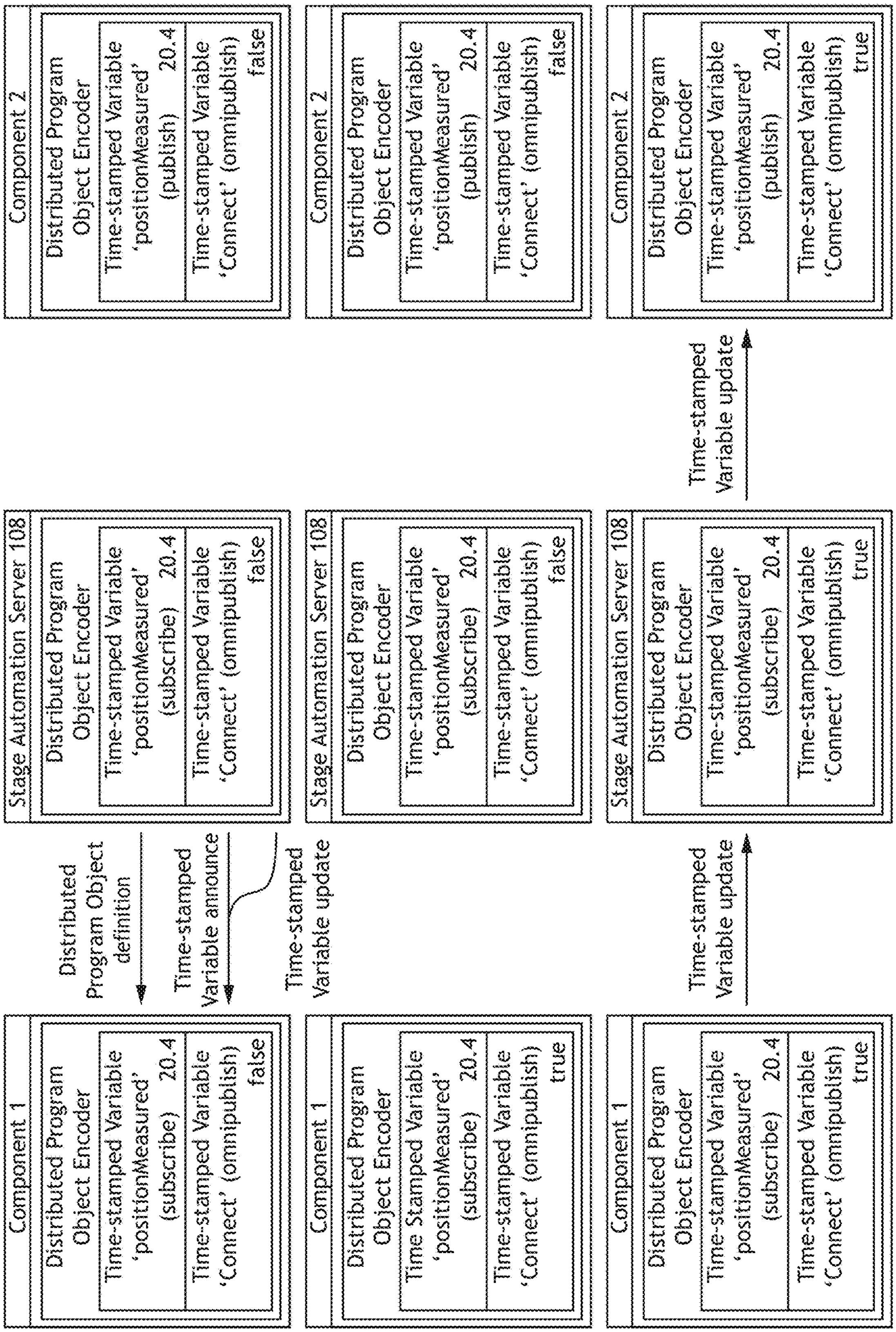

Component 1
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (subscribe) 20.4
Time-stamped Variable 'Connect' (omnipublish) false
Distributed Program Object definition
Time-stamped Variable announce
Time-stamped Variable update
Stage Automation Server 108
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (subscribe) 20.4
Time-stamped Variable 'Connect' (omnipublish) false
Component 2
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (publish) 20.4
Time-stamped Variable 'Connect' (omnipublish) false
Component 1
Distributed Program Object Encoder
Time Stamped Variable 'positionMeasured' (subscribe) 20.4
Time-stamped Variable 'Connect' (omnipublish) true
Stage Automation Server 108
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (subscribe) 20.4
Time-stamped Variable 'Connect' (omnipublish) false
Component 2
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (publish) 20.4
Time-stamped Variable 'Connect' (omnipublish) false
Component 1
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (subscribe) 20.4
Time-stamped Variable 'Connect' (omnipublish) true
Time-stamped Variable update
Stage Automation Server 108
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (subscribe) 20.4
Time-stamped Variable 'Connect' (omnipublish) true
Time-stamped Variable update
Component 2
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (publish) 20.4
Time-stamped Variable 'Connect' (omnipublish) true

FIG. 3B

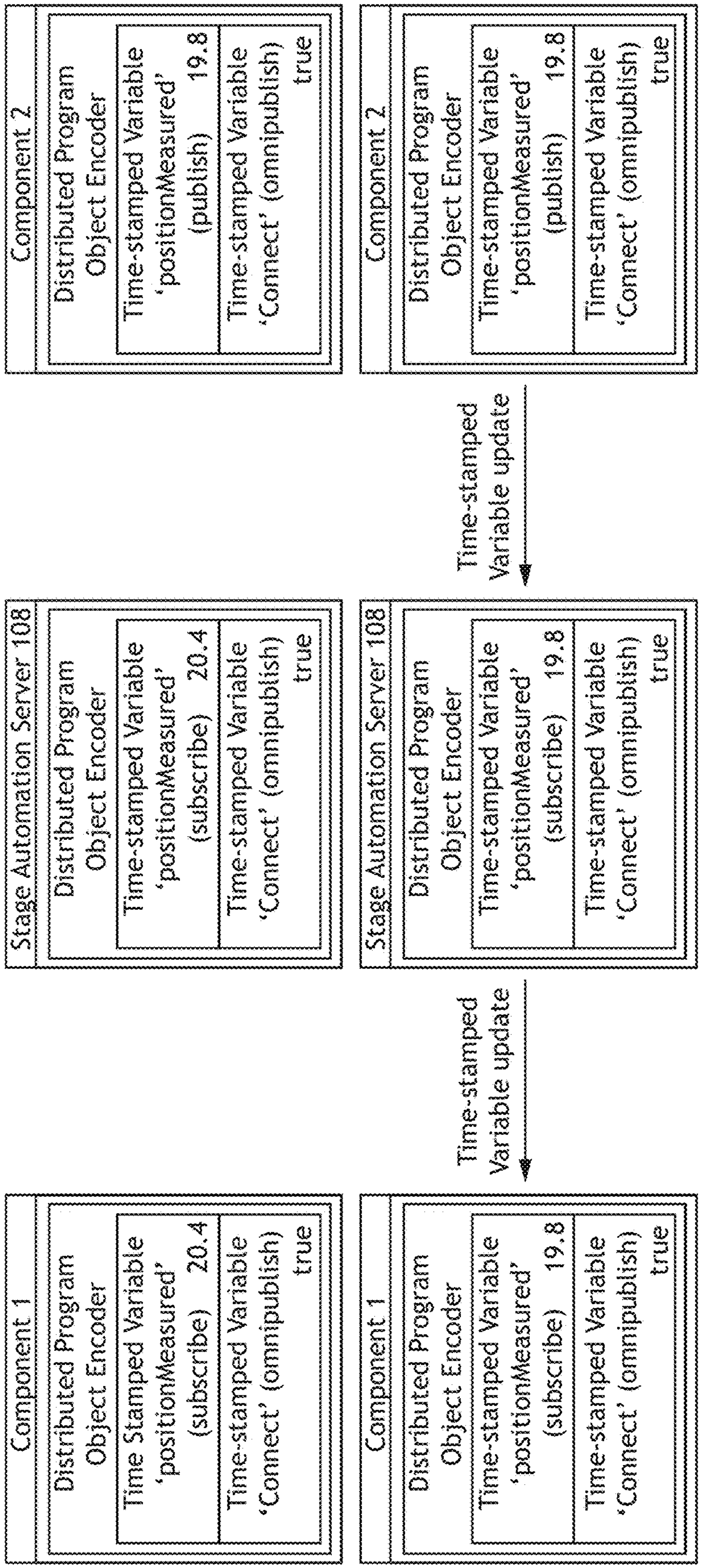

Component 1
Distributed Program Object Encoder
Time Stamped Variable 'positionMeasured' (subscribe) 20.4
Time-stamped Variable 'Connect' (omnipublish) true
Stage Automation Server 108
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (subscribe) 20.4
Time-stamped Variable 'Connect' (omnipublish) true
Component 2
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (publish) 19.8
Time-stamped Variable 'Connect' (omnipublish) true
Component 1
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (subscribe) 19.8
Time-stamped Variable 'Connect' (omnipublish) true
Time-stamped Variable update
Stage Automation Server 108
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (subscribe) 19.8
Time-stamped Variable 'Connect' (omnipublish) true
Time-stamped Variable update
Component 2
Distributed Program Object Encoder
Time-stamped Variable 'positionMeasured' (publish) 19.8
Time-stamped Variable 'Connect' (omnipublish) true

FIG. 3C

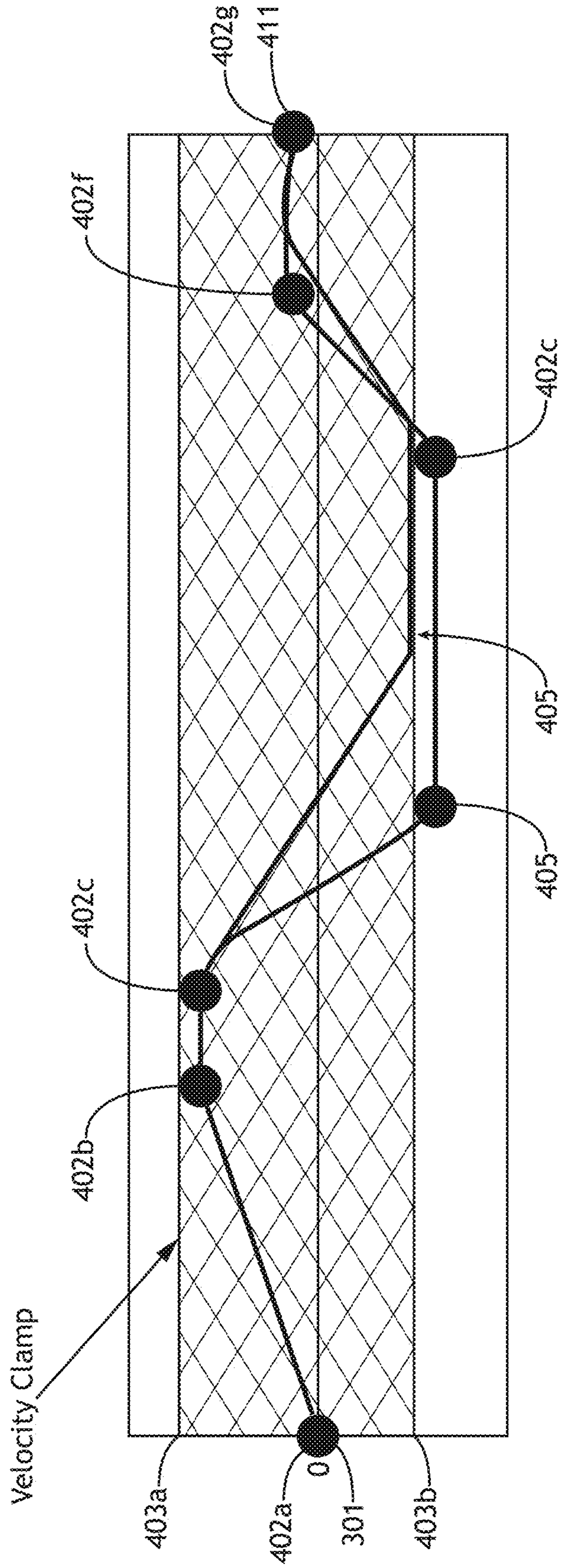

402g
411
402f
402c
405
405
402c
402b
Velocity Clamp
0
403a
402a
301
403b

FIG. 4

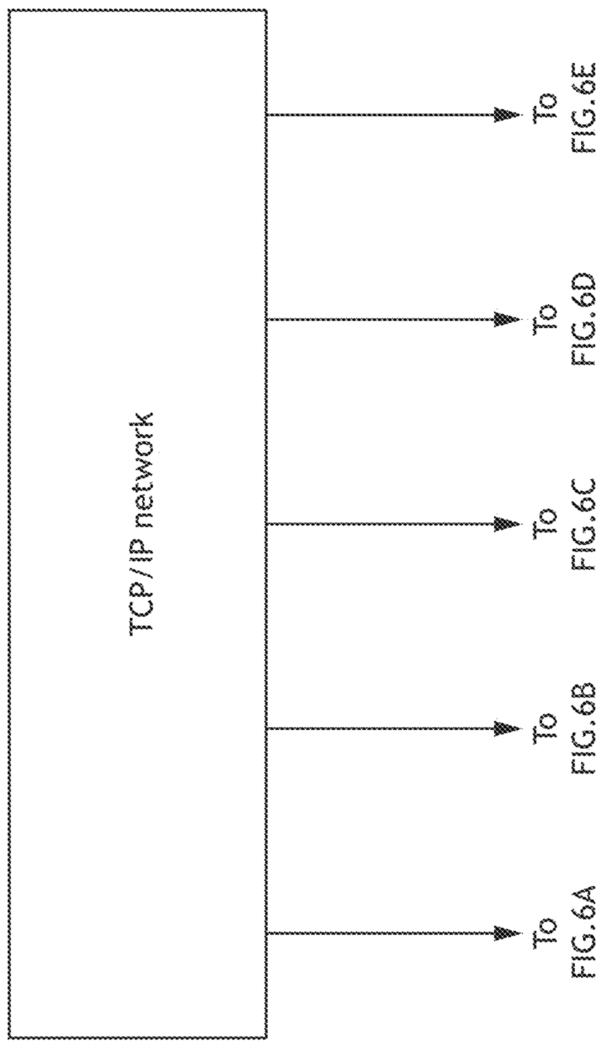


FIG 5

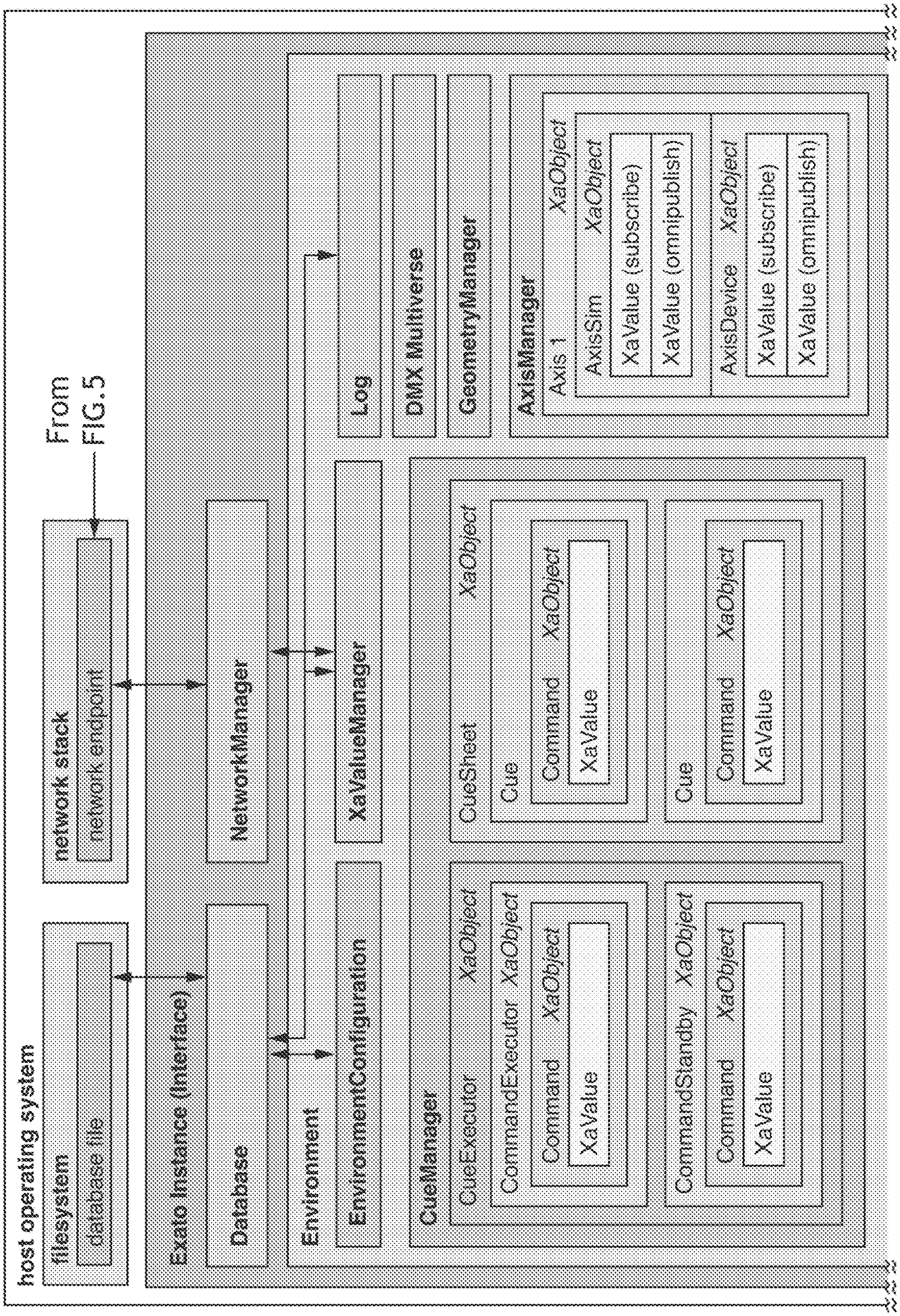

host operating system
filesystem
database file
network stack
network endpoint
From FIG.5
Exato Instance (Interface)
Database
NetworkManager
Environment
EnvironmentConfiguration
XaValueManager
Log
DMX Multiverse
GeometryManager
AxisManager
Axis 1
XaObject
AxisSim
XaObject
XaValue (subscribe)
XaValue (omnipublish)
AxisDevice
XaObject
XaValue (subscribe)
XaValue (omnipublish)
CueManager
CueExecutor
XaObject
CommandExecutor
XaObject
Command
XaObject
XaValue
CommandStandby
XaObject
Command
XaObject
XaValue
CueSheet
XaObject
Cue
XaObject
Command
XaObject
XaValue
Cue
XaObject
Command
XaObject
XaValue

FIG 6A-1

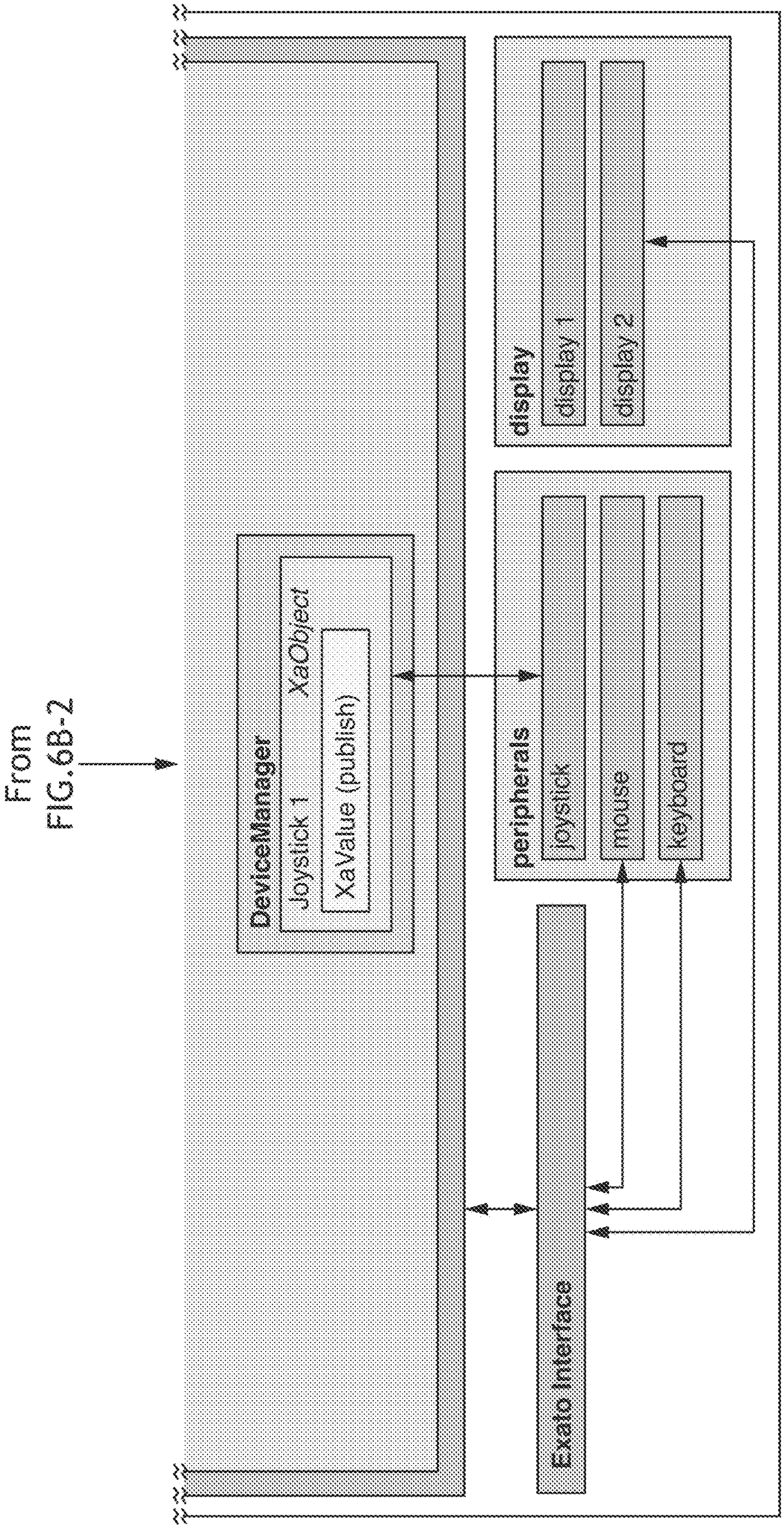

From
FIG.6B-2
DeviceManager
Joystick 1
XaObject
XaValue (publish)
peripherals
joystick
mouse
keyboard
display
display 1
display 2
Exato Interface

FIG 6A-2

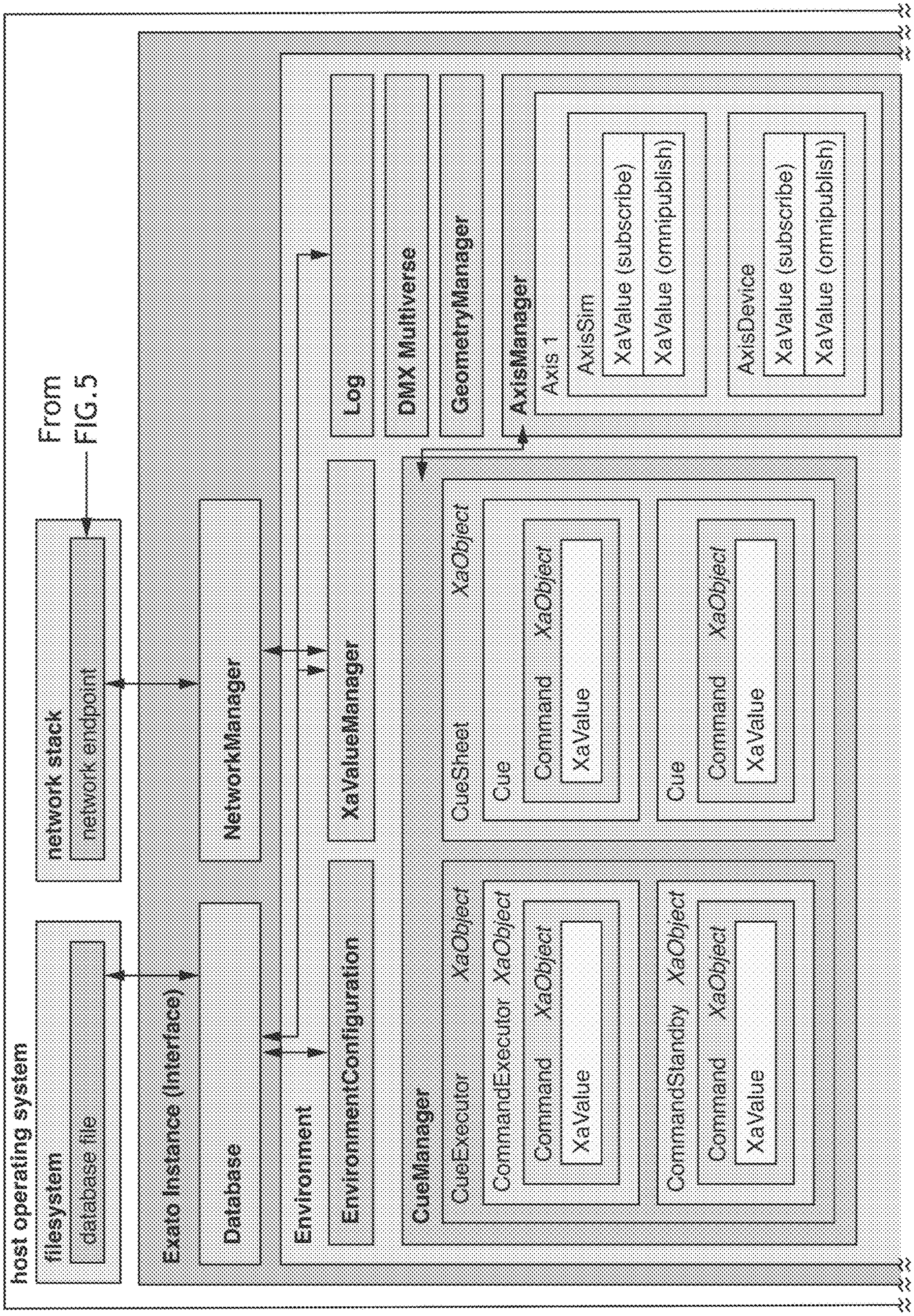

host operating system
filesystem
database file
network stack
network endpoint
From FIG.5
Exato Instance (Interface)
Database
NetworkManager
Environment
EnvironmentConfiguration
XaValueManager
Log
DMX Multiverse
GeometryManager
AxisManager
Axis 1
AxisSim
XaValue (subscribe)
XaValue (omnipublish)
AxisDevice
XaValue (subscribe)
XaValue (omnipublish)
CueManager
CueExecutor
XaObject
CommandExecutor
XaObject
Command
XaObject
XaValue
CommandStandby
XaObject
Command
XaObject
XaValue
CueSheet
XaObject
Cue
Command
XaObject
XaValue
Cue
Command
XaObject
XaValue

FIG 6B-1

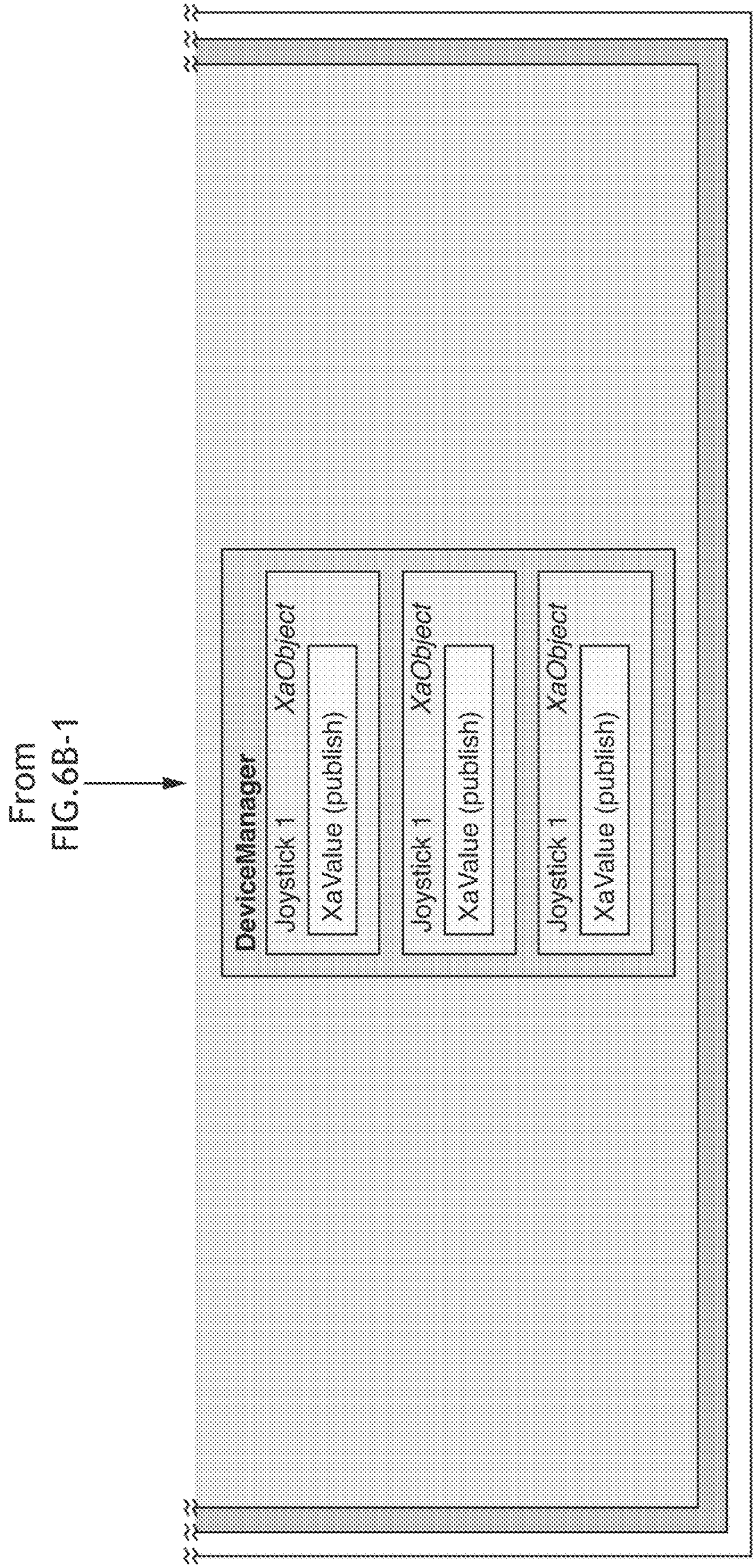

From
FIG.6B-1
DeviceManager
Joystick 1
XaObject
XaValue (publish)
Joystick 1
XaObject
XaValue (publish)
Joystick 1
XaObject
XaValue (publish)

FIG 6B-2

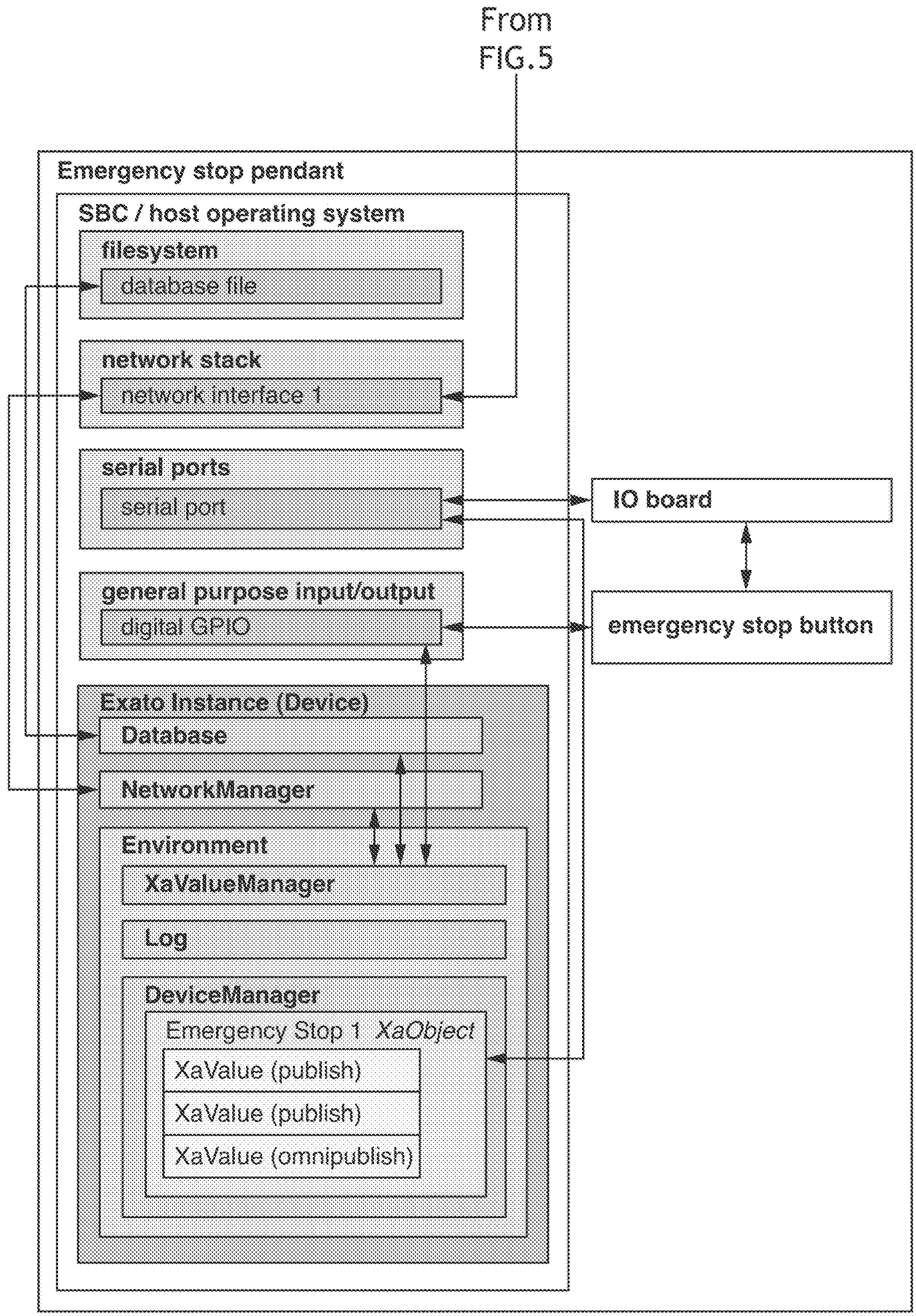

From
FIG.5
Emergency stop pendant
SBC / host operating system
filesystem
database file
network stack
network interface 1
serial ports
serial port
IO board
general purpose input/output
digital GPIO
emergency stop button
Exato Instance (Device)
Database
NetworkManager
Environment
XaValueManager
Log
DeviceManager
Emergency Stop 1 XaObject
XaValue (publish)
XaValue (publish)
XaValue (omnipublish)

FIG 6C

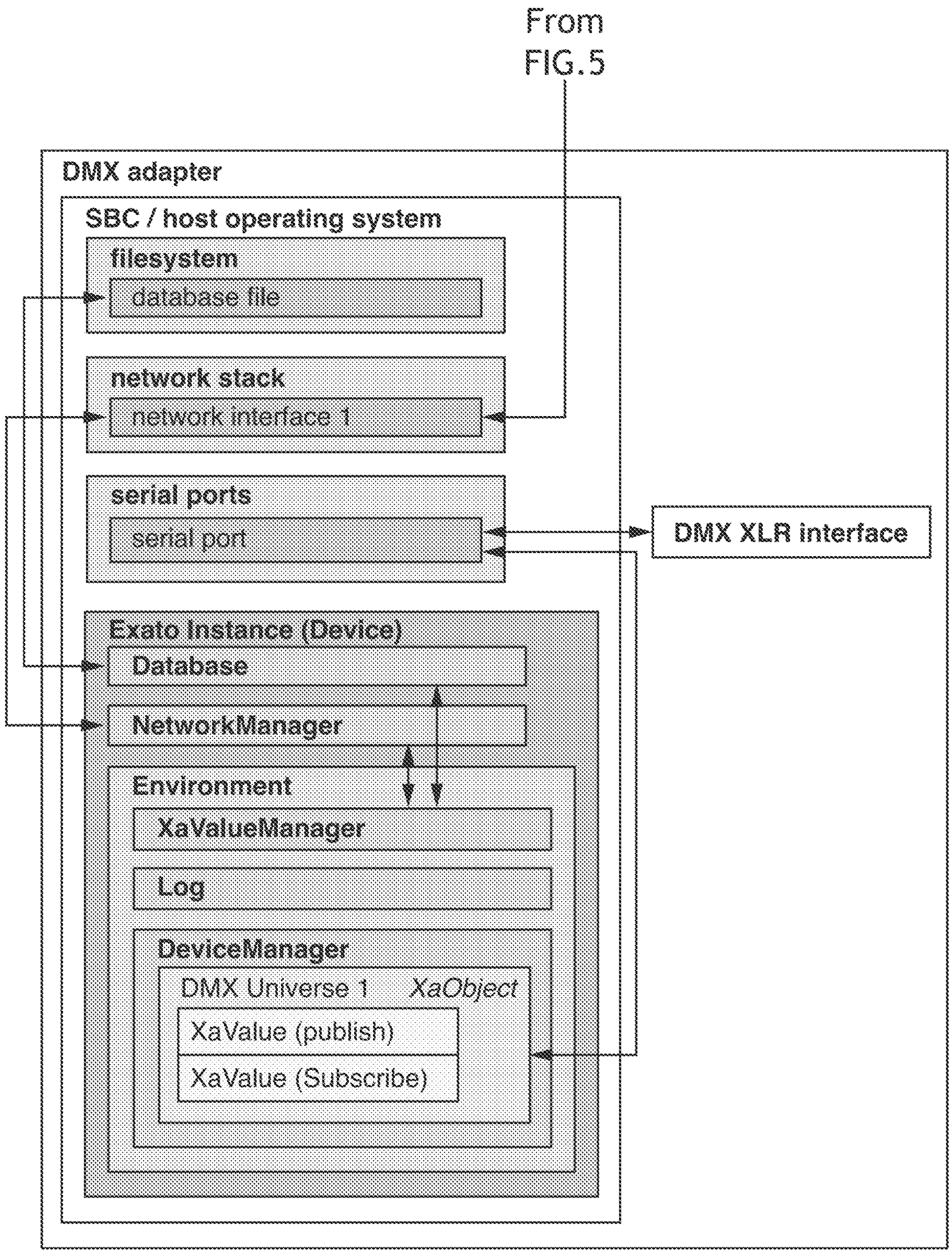

From
FIG.5
DMX adapter
SBC / host operating system
filesystem
database file
network stack
network interface 1
serial ports
serial port
DMX XLR interface
Exato Instance (Device)
Database
NetworkManager
Environment
XaValueManager
Log
DeviceManager
DMX Universe 1
XaObject
XaValue (publish)
XaValue (Subscribe)

FIG 6D

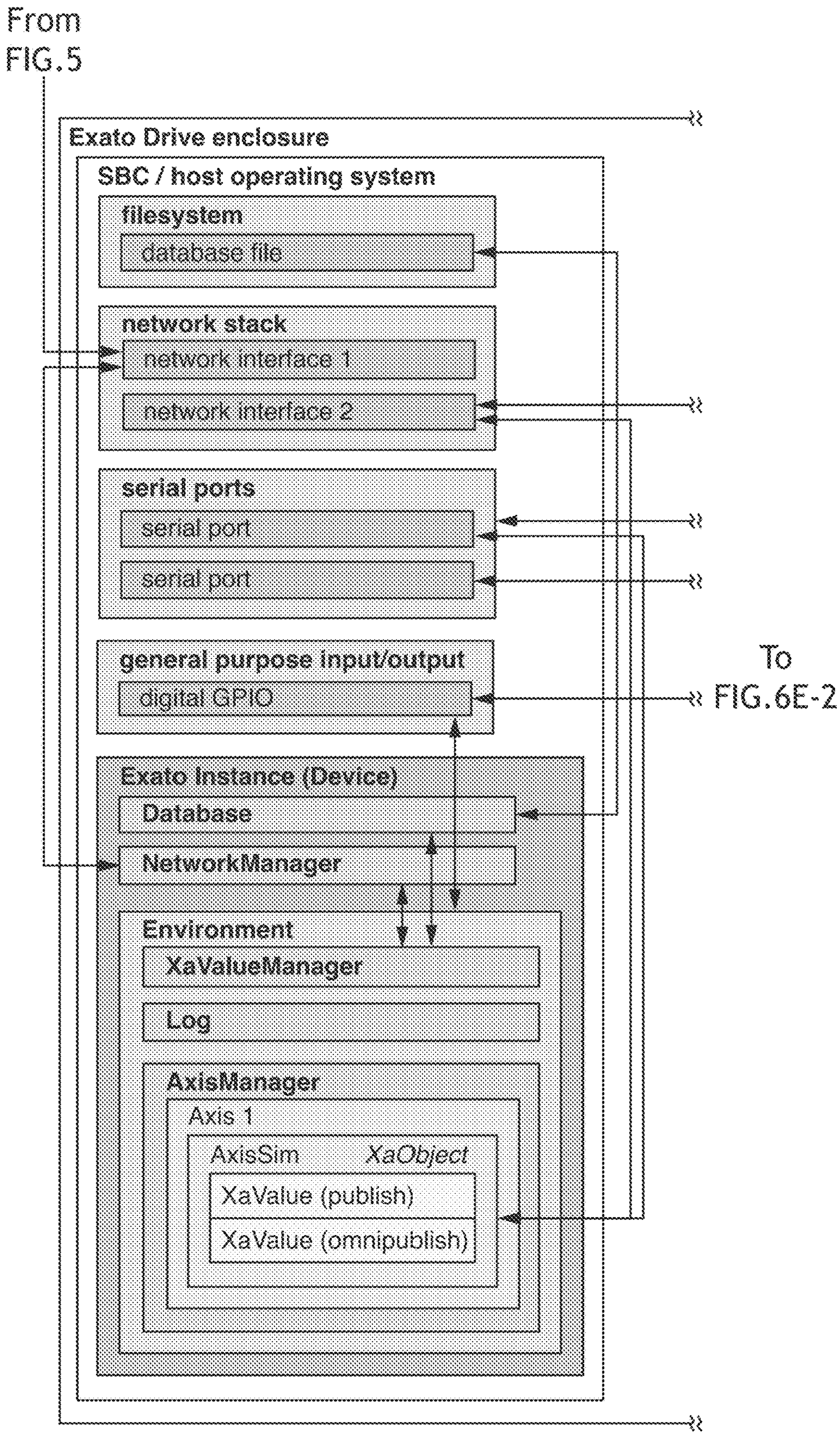

From FIG.5
Exato Drive enclosure
SBC / host operating system
filesystem
database file
network stack
network interface 1
network interface 2
serial ports
serial port
serial port
general purpose input/output
digital GPIO
To FIG.6E-2
Exato Instance (Device)
Database
NetworkManager
Environment
XaValueManager
Log
AxisManager
Axis 1
AxisSim
XaObject
XaValue (publish)
XaValue (omnipublish)

FIG 6E-1

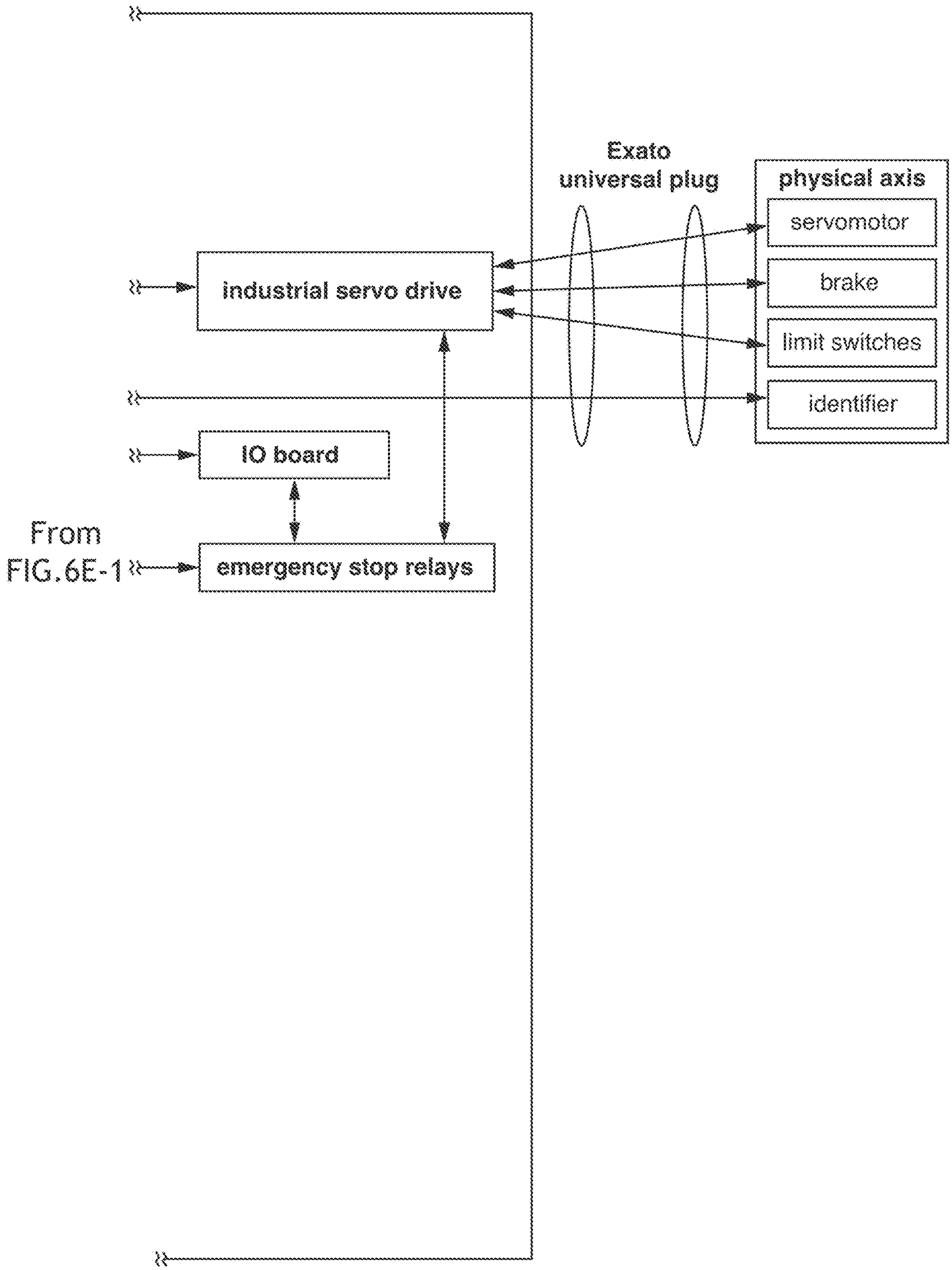

Exato
universal plug
physical axis
servomotor
brake
limit switches
identifier
industrial servo drive
IO board
From
FIG.6E-1
emergency stop relays

FIG 6E-2

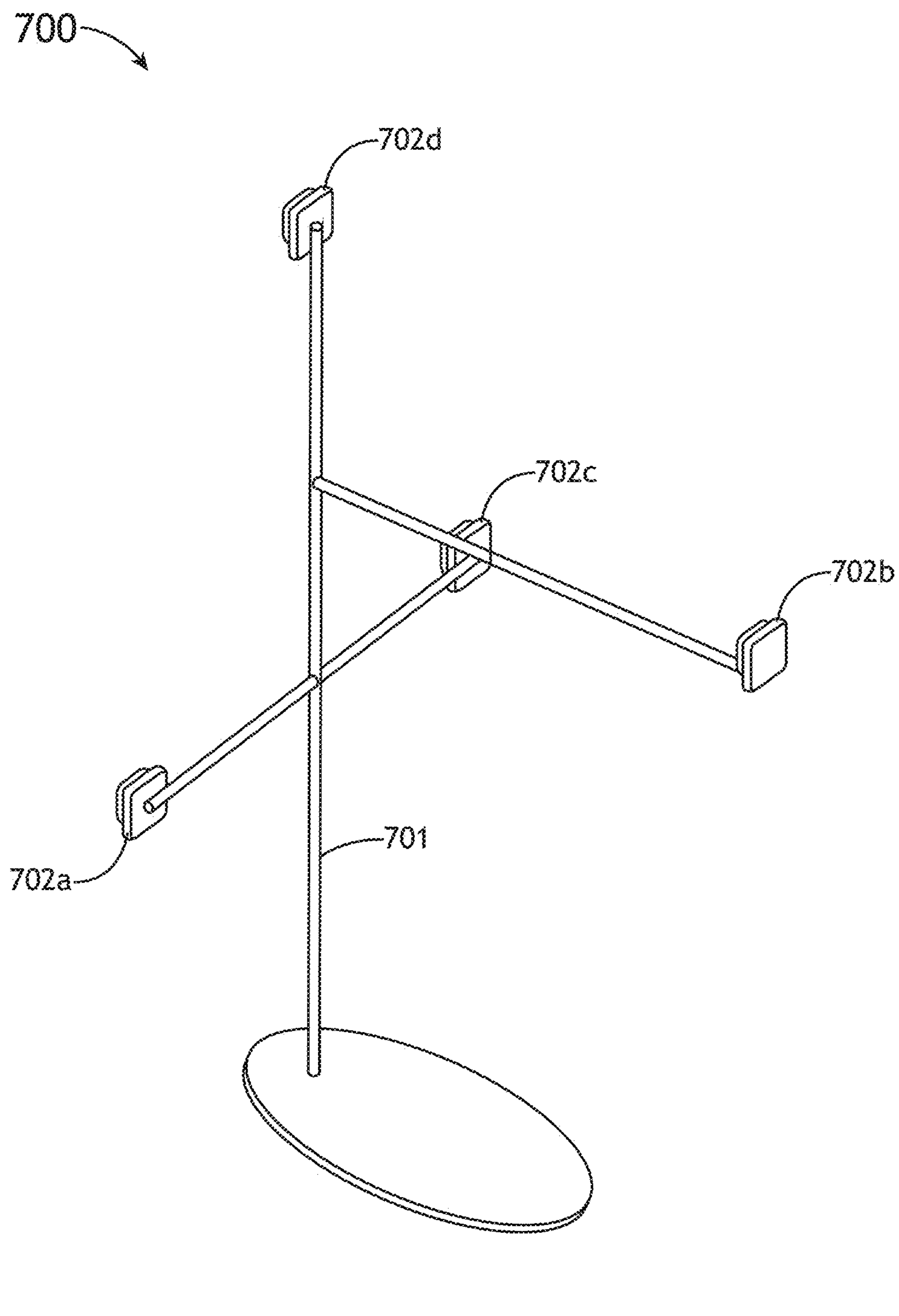


FIG. 7A

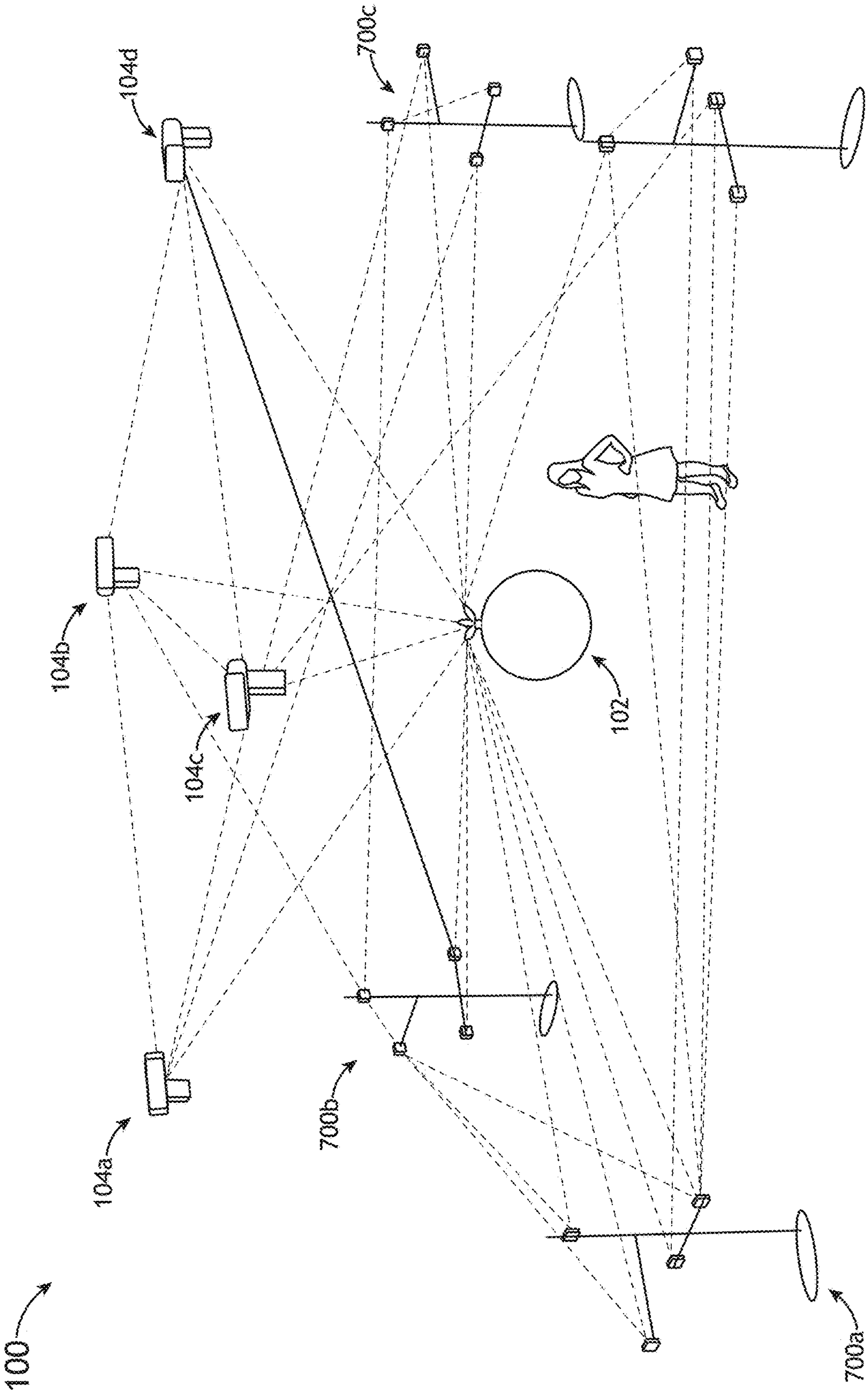


FIG. 7B

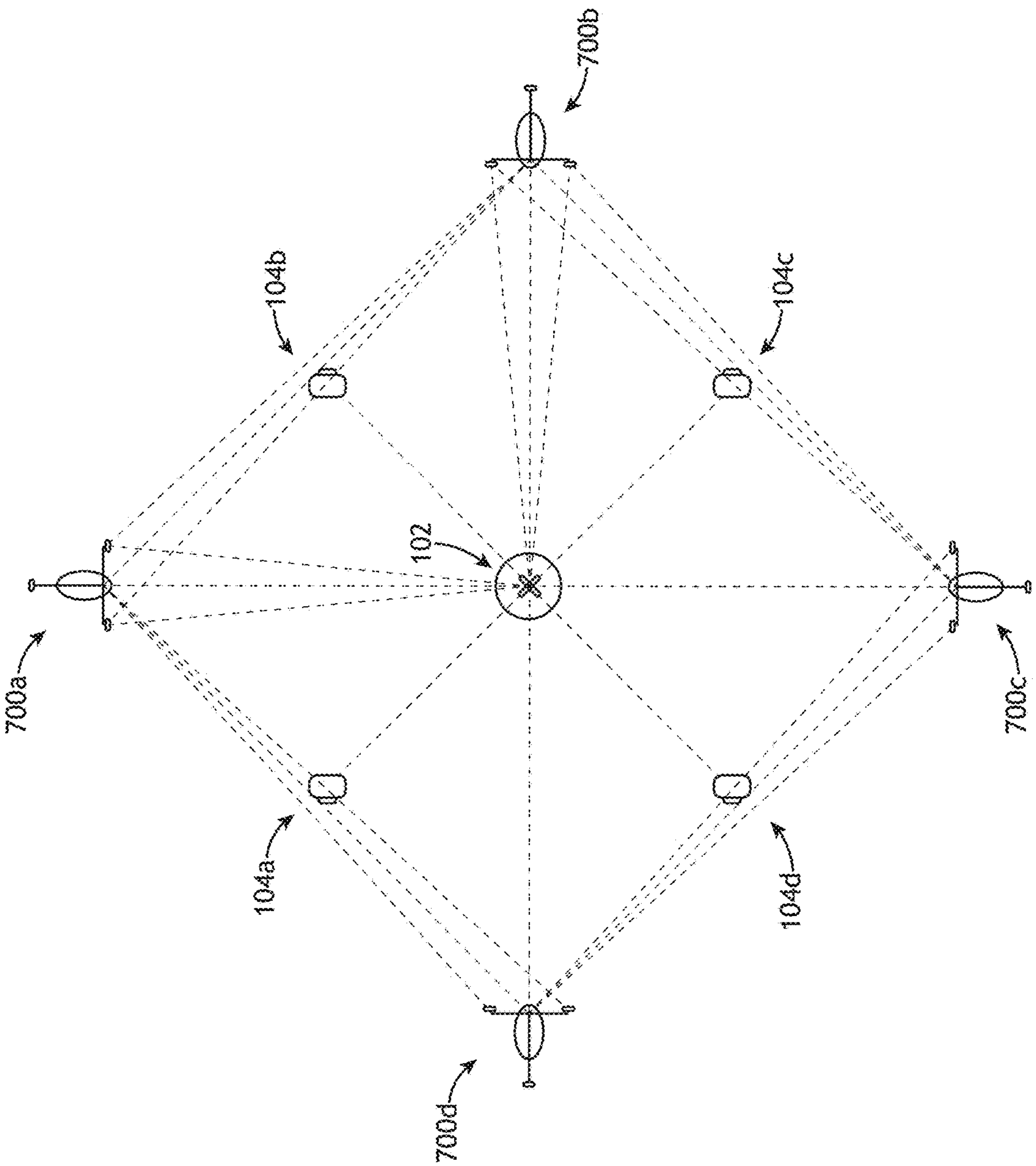


FIG. 7C

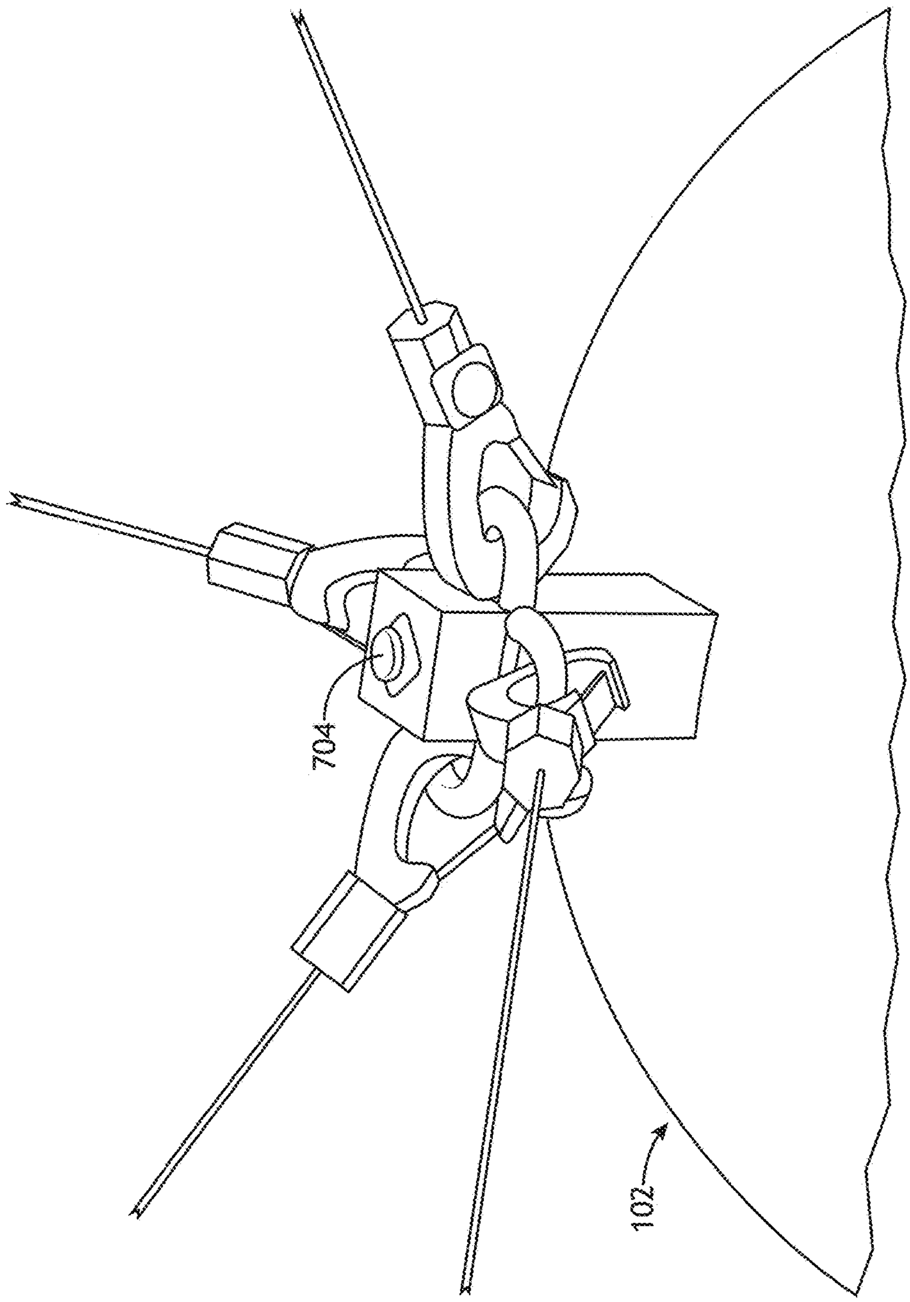


FIG. 7D

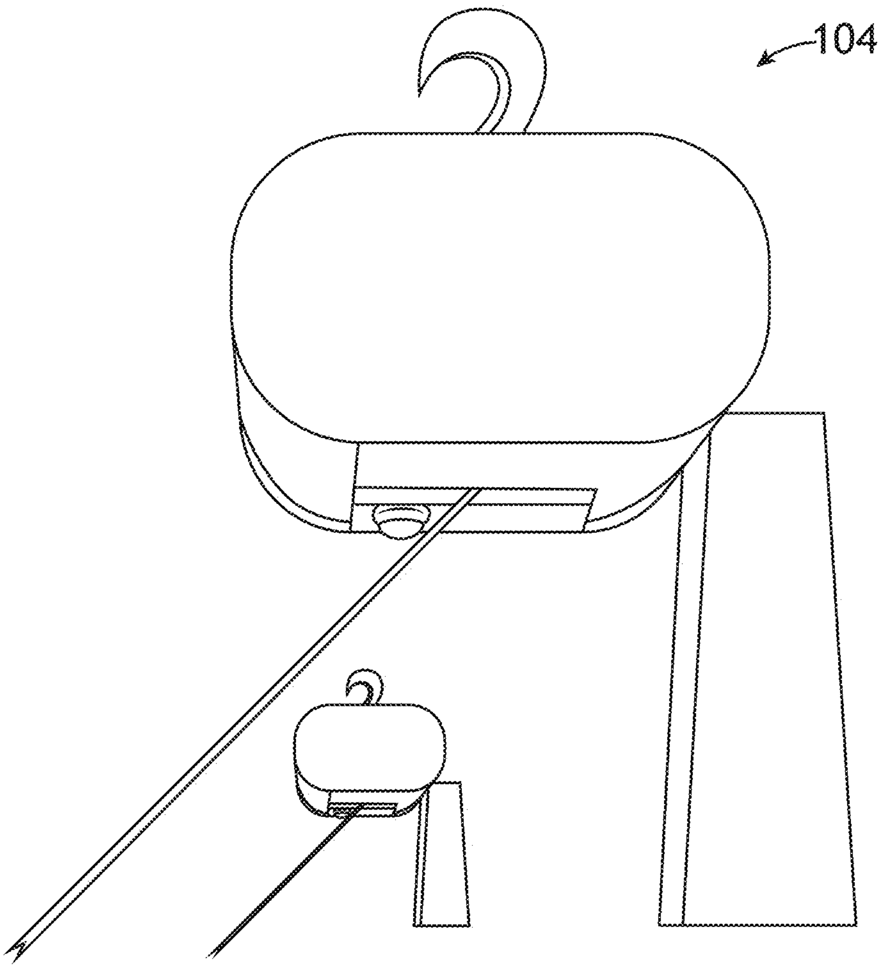


FIG.7E

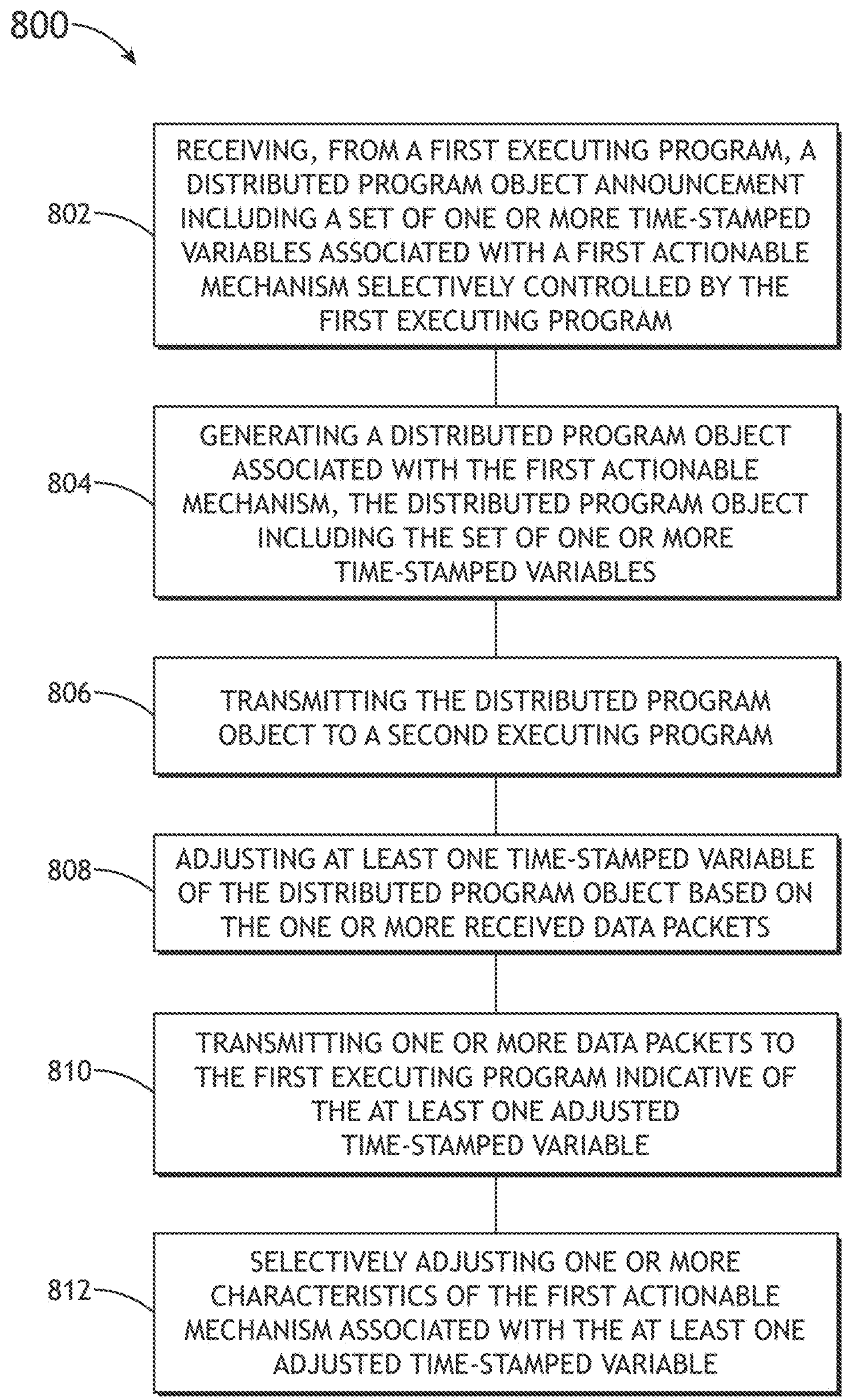

800
802
RECEIVING, FROM A FIRST EXECUTING PROGRAM, A DISTRIBUTED PROGRAM OBJECT ANNOUNCEMENT INCLUDING A SET OF ONE OR MORE TIME-STAMPED VARIABLES ASSOCIATED WITH A FIRST ACTIONABLE MECHANISM SELECTIVELY CONTROLLED BY THE FIRST EXECUTING PROGRAM
804
GENERATING A DISTRIBUTED PROGRAM OBJECT ASSOCIATED WITH THE FIRST ACTIONABLE MECHANISM, THE DISTRIBUTED PROGRAM OBJECT INCLUDING THE SET OF ONE OR MORE TIME-STAMPED VARIABLES
806
TRANSMITTING THE DISTRIBUTED PROGRAM OBJECT TO A SECOND EXECUTING PROGRAM
808
ADJUSTING AT LEAST ONE TIME-STAMPED VARIABLE OF THE DISTRIBUTED PROGRAM OBJECT BASED ON THE ONE OR MORE RECEIVED DATA PACKETS
810
TRANSMITTING ONE OR MORE DATA PACKETS TO THE FIRST EXECUTING PROGRAM INDICATIVE OF THE AT LEAST ONE ADJUSTED TIME-STAMPED VARIABLE
812
SELECTIVELY ADJUSTING ONE OR MORE CHARACTERISTICS OF THE FIRST ACTIONABLE MECHANISM ASSOCIATED WITH THE AT LEAST ONE ADJUSTED TIME-STAMPED VARIABLE

FIG. 8

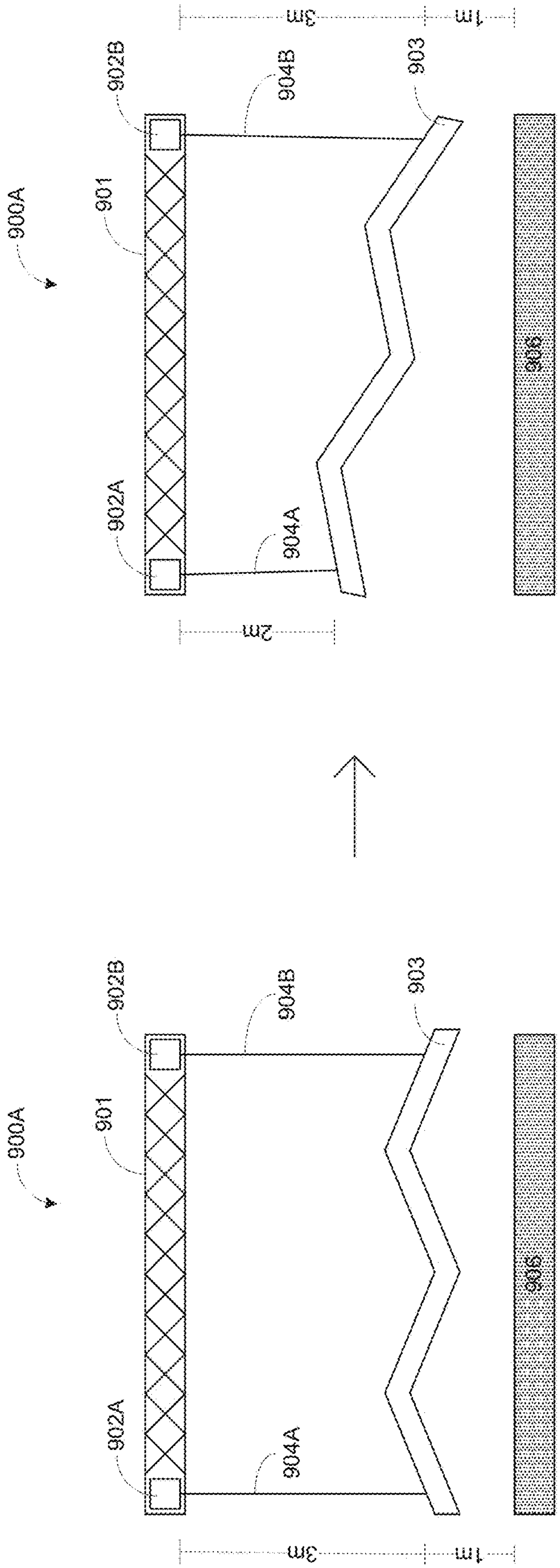

900A
901
902A
902B
903
904A
904B
905
3m
1m
2m
900A
901
902A
902B
903
904A
904B
905
3m
1m

FIG. 9A

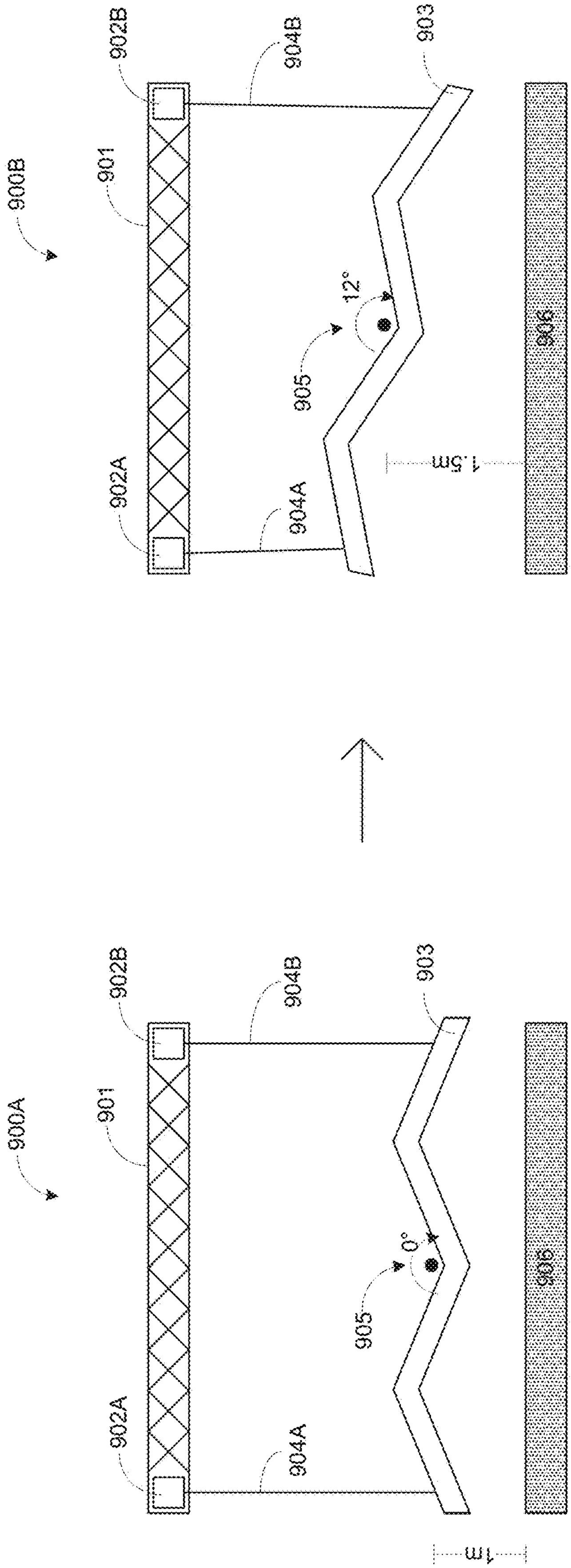

900B
901
902A
902B
903
904A
904B
905
906
12°
1.5m
900A
901
902A
902B
903
904A
904B
905
906
0°
1m

FIG. 9B

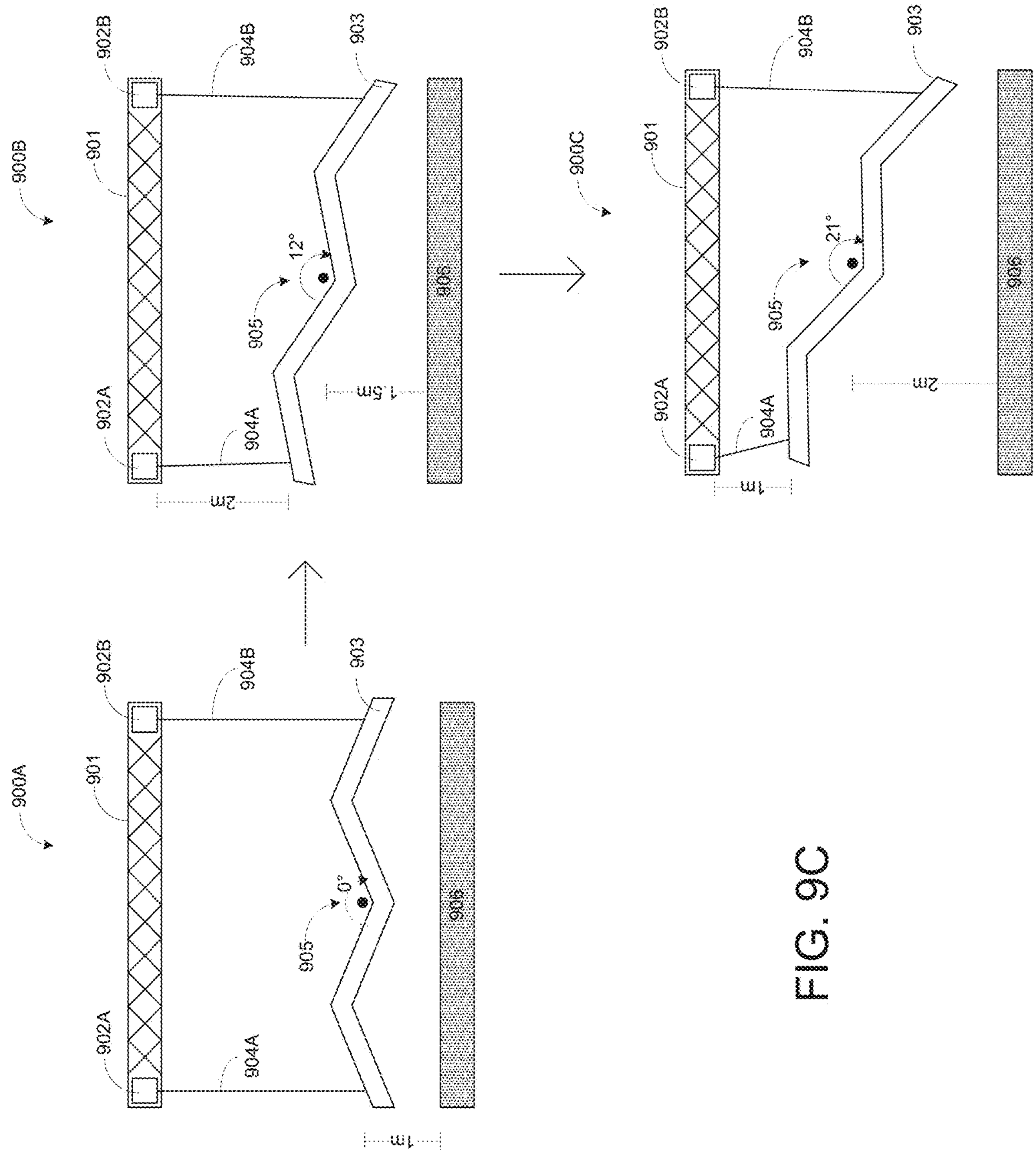

900A
900B
900C
901
902A
902B
903
904A
904B
905
906
0°
12°
21°
1m
2m
1.5m
2m
1m
2m

FIG. 9C

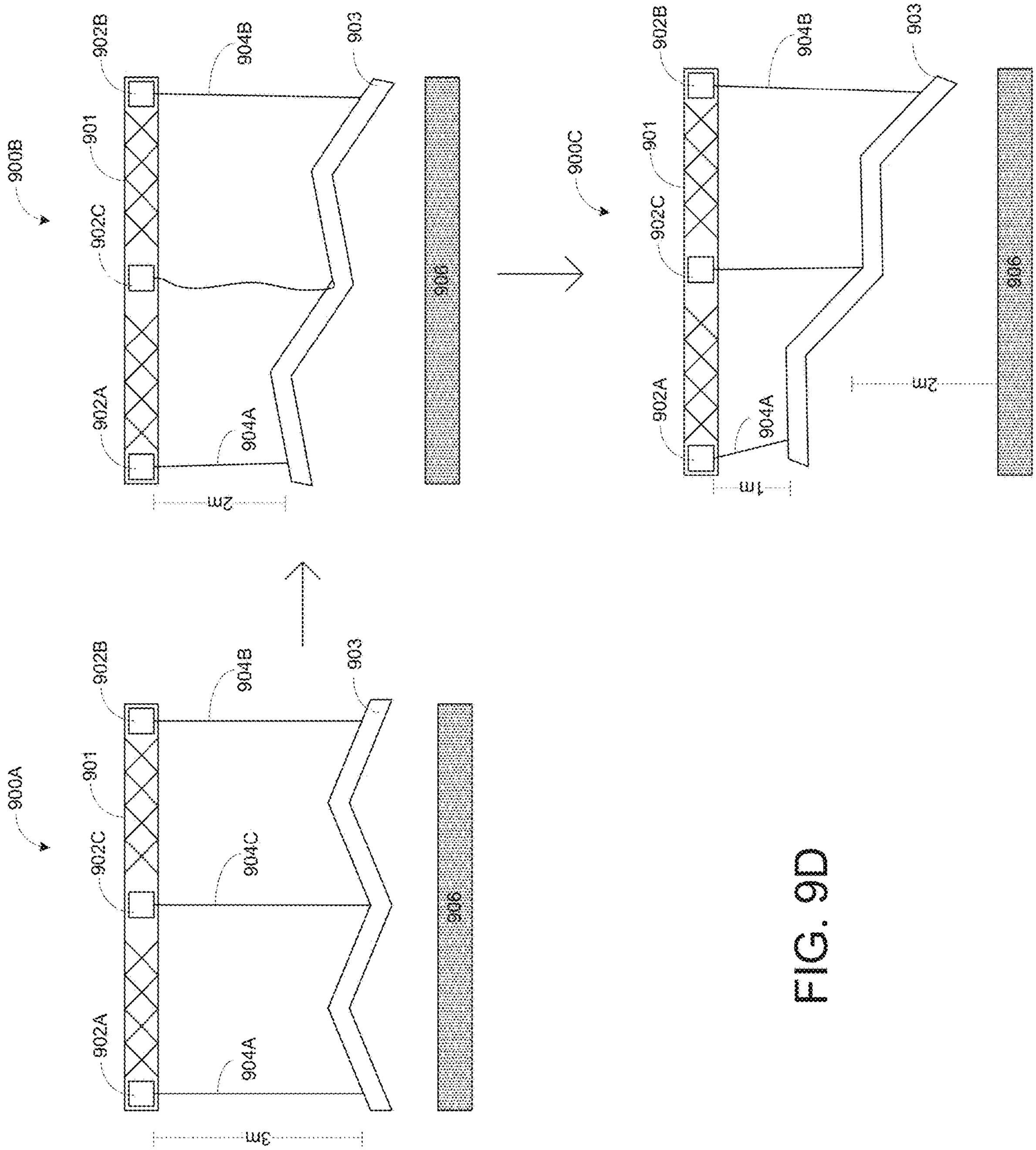

900A
900B
900C
901
902A
902B
902C
903
904A
904B
904C
906
3m
2m
1m

FIG. 9D

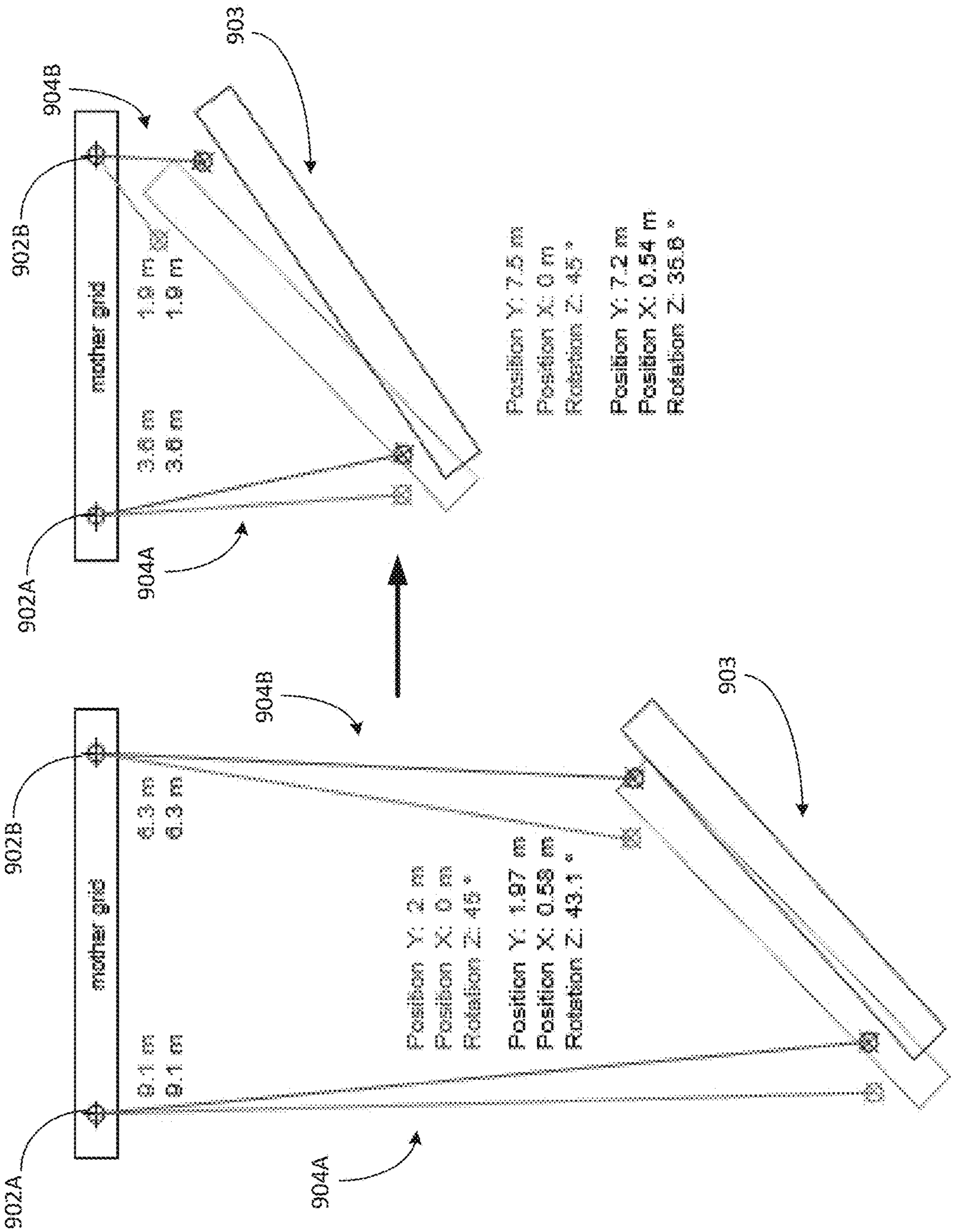

903
904B
902B
mother grid
1.9 m
1.9 m
3.8 m
3.8 m
902A
904A
Position Y: 7.5 m
Position X: 0 m
Rotation Z: 45 °
Position Y: 7.2 m
Position X: 0.54 m
Rotation Z: 35.6 °
903
904B
902B
mother grid
6.3 m
6.3 m
9.1 m
9.1 m
902A
904A
Position Y: 2 m
Position X: 0 m
Rotation Z: 45 °
Position Y: 1.97 m
Position X: 0.58 m
Rotation Z: 43.1 °

FIG. 10

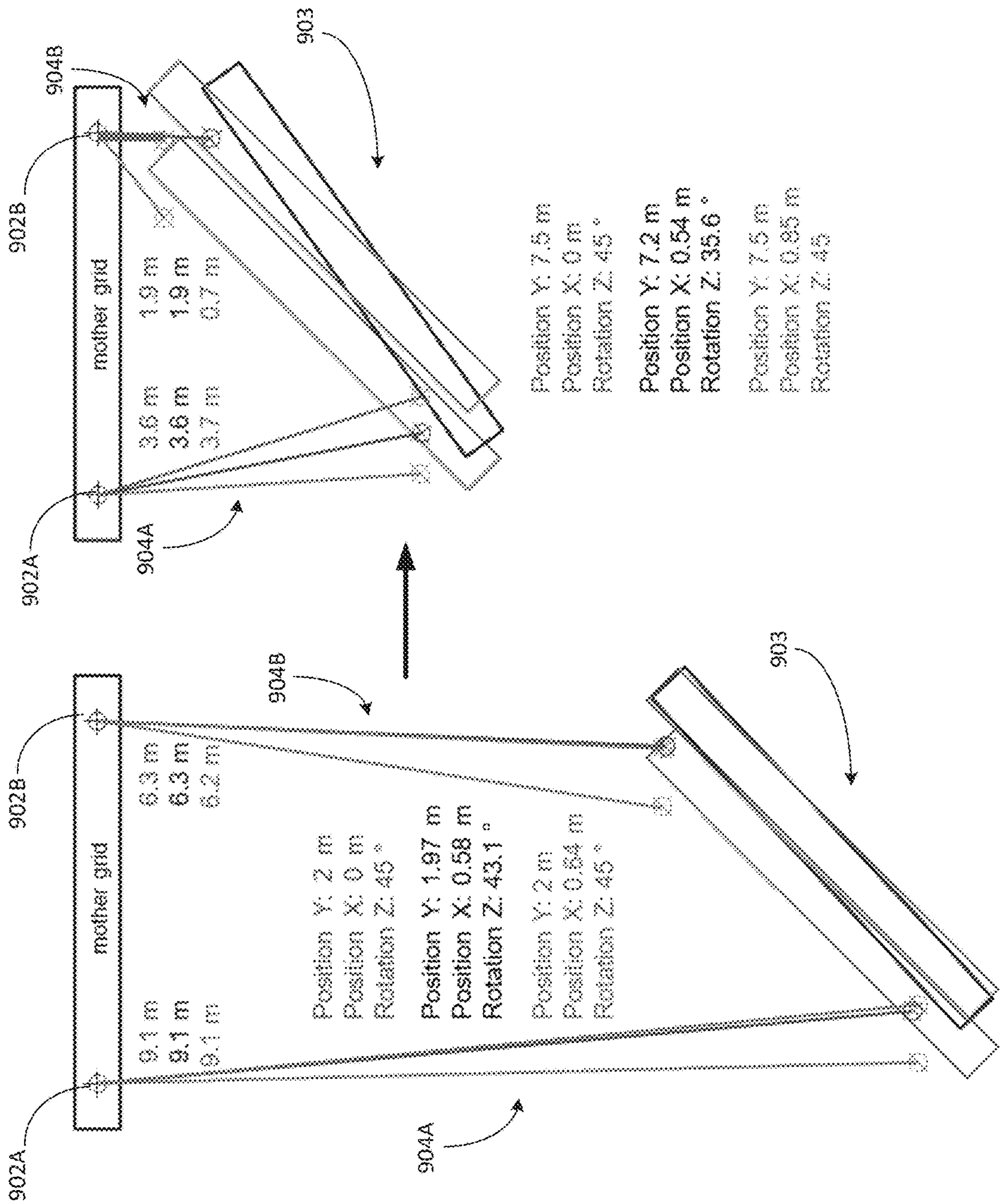

902A
902B
904A
904B
903
mother grid
9.1 m
9.1 m
9.1 m
6.3 m
6.3 m
6.2 m
Position Y: 2 m
Position X: 0 m
Rotation Z: 45 °
Position Y: 1.97 m
Position X: 0.58 m
Rotation Z: 43.1 °
Position Y: 2 m
Position X: 0.54 m
Rotation Z: 45 °
902A
902B
904A
904B
903
mother grid
3.6 m
3.6 m
3.7 m
1.9 m
1.9 m
0.7 m
Position Y: 7.5 m
Position X: 0 m
Rotation Z: 45 °
Position Y: 7.2 m
Position X: 0.54 m
Rotation Z: 35.6 °
Position Y: 7.5 m
Position X: 0.85 m
Rotation Z: 45

FIG. 11

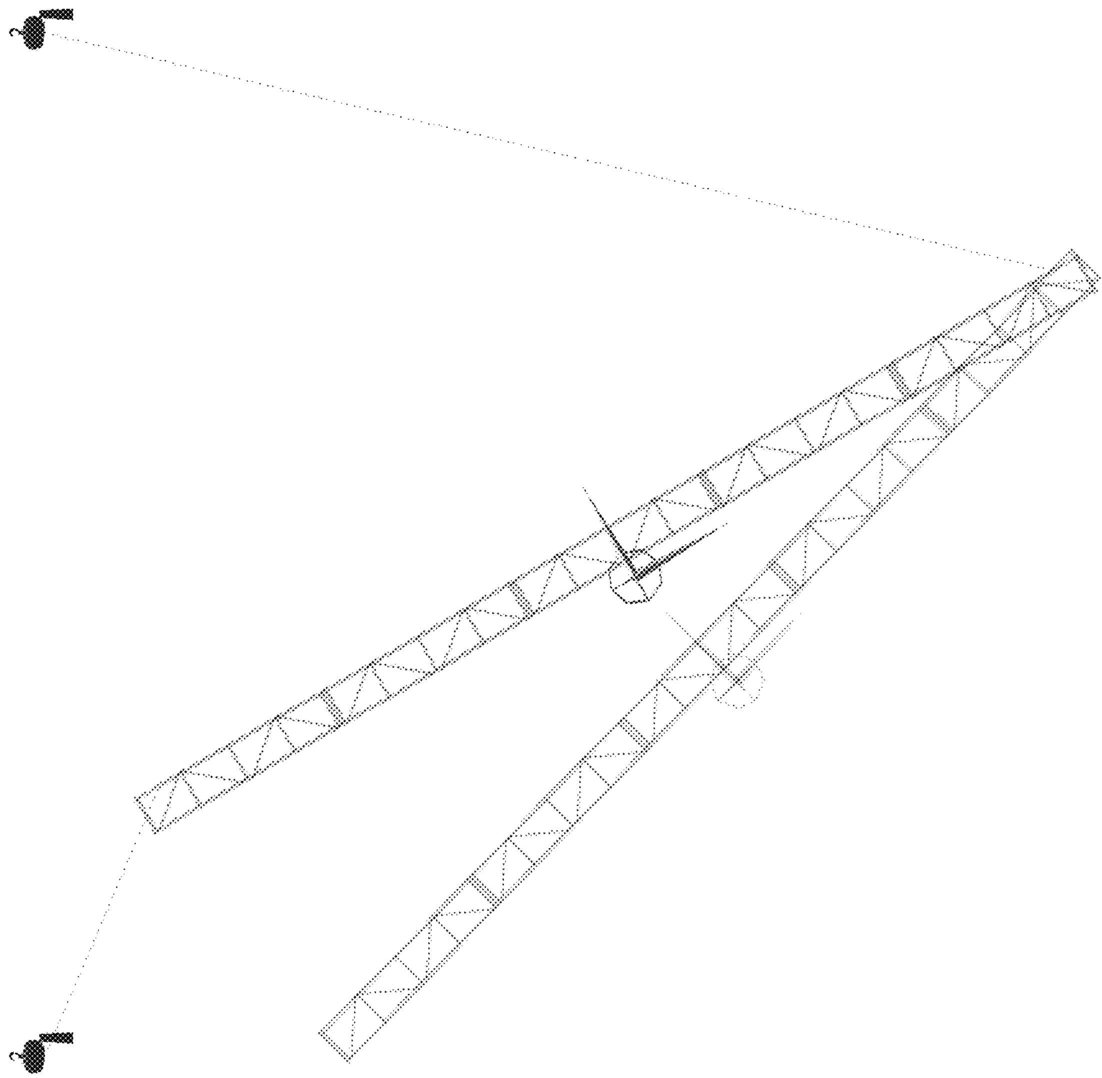


FIG. 12A

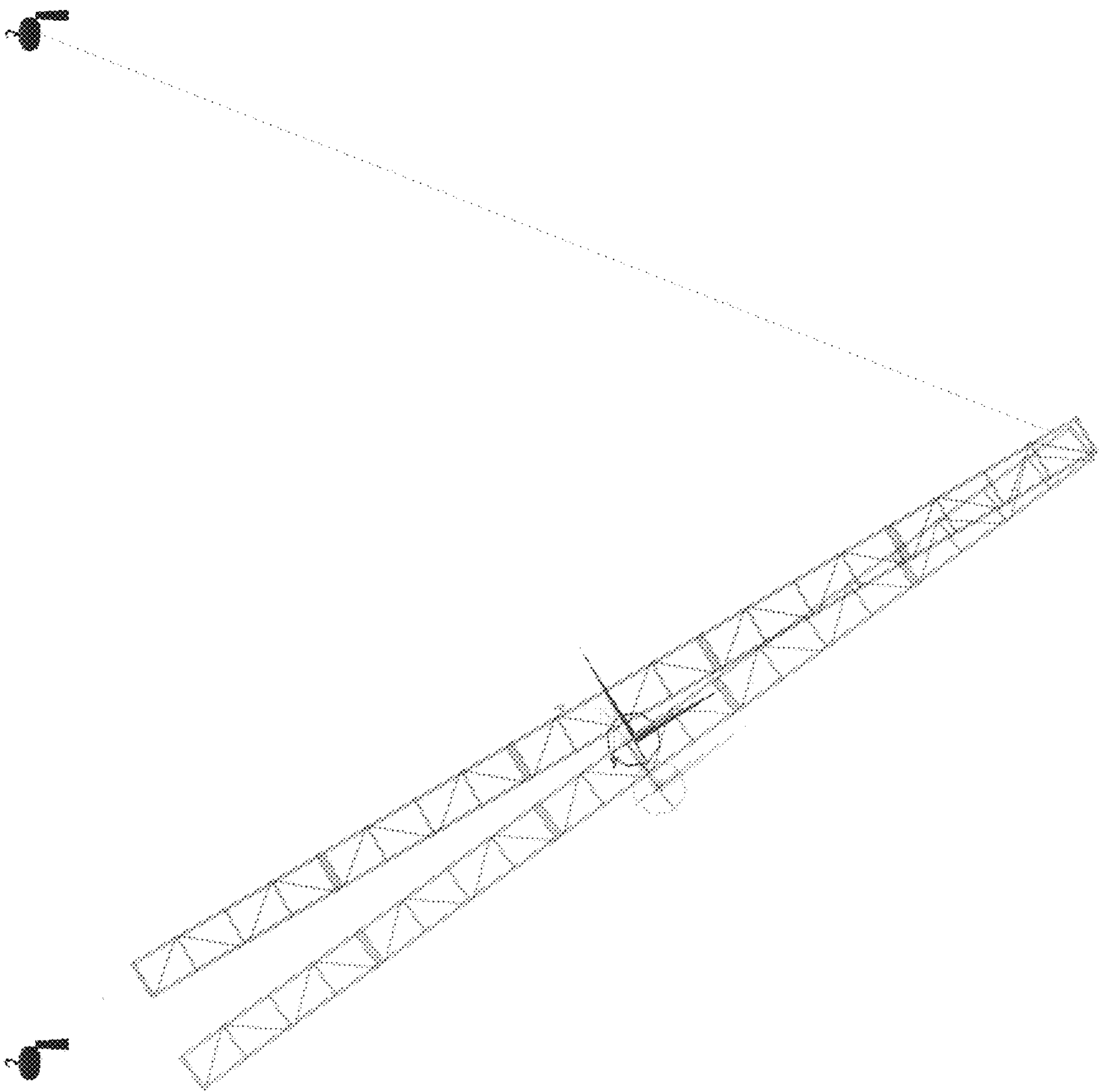


FIG. 12B

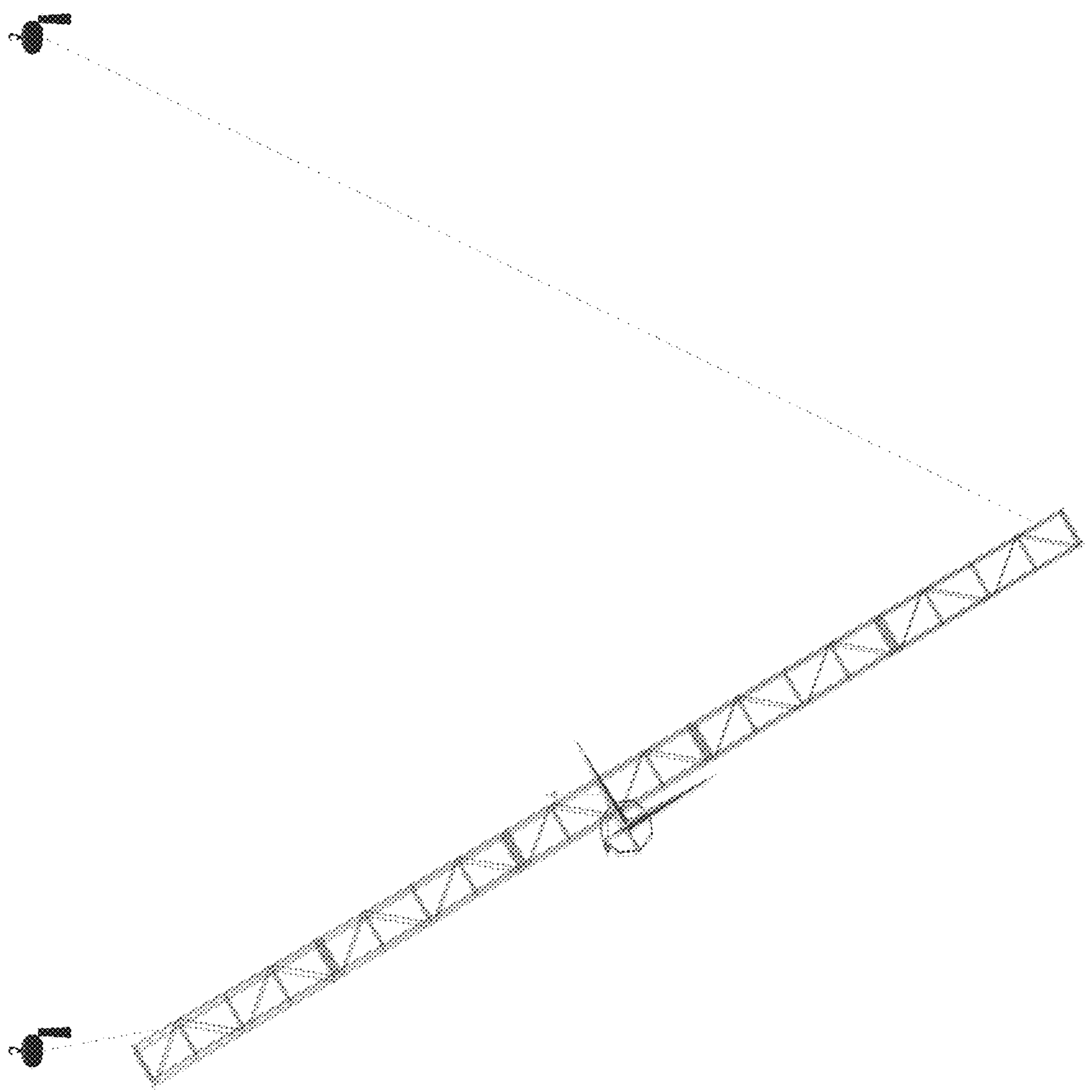


FIG. 12C

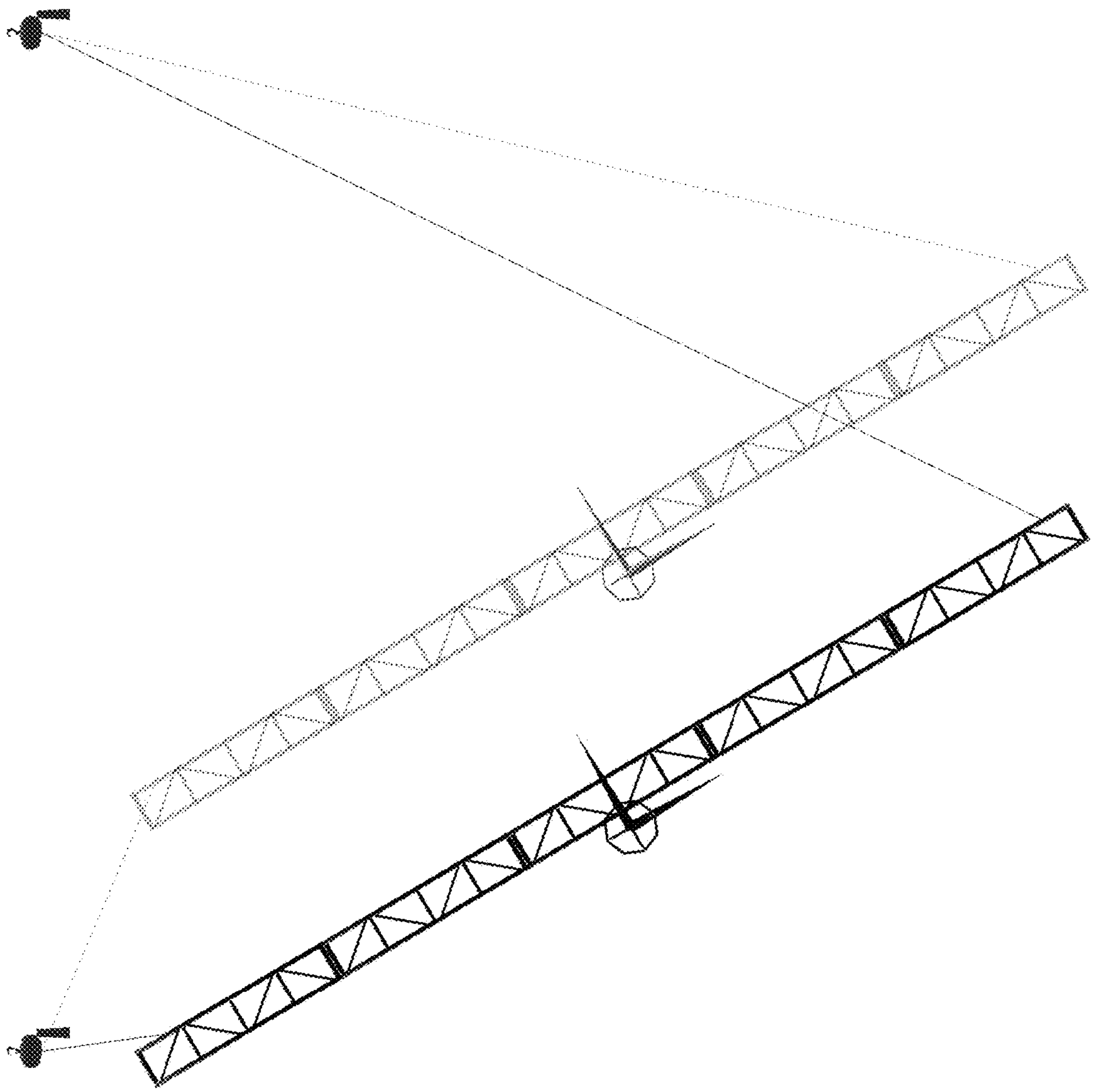


FIG. 12D

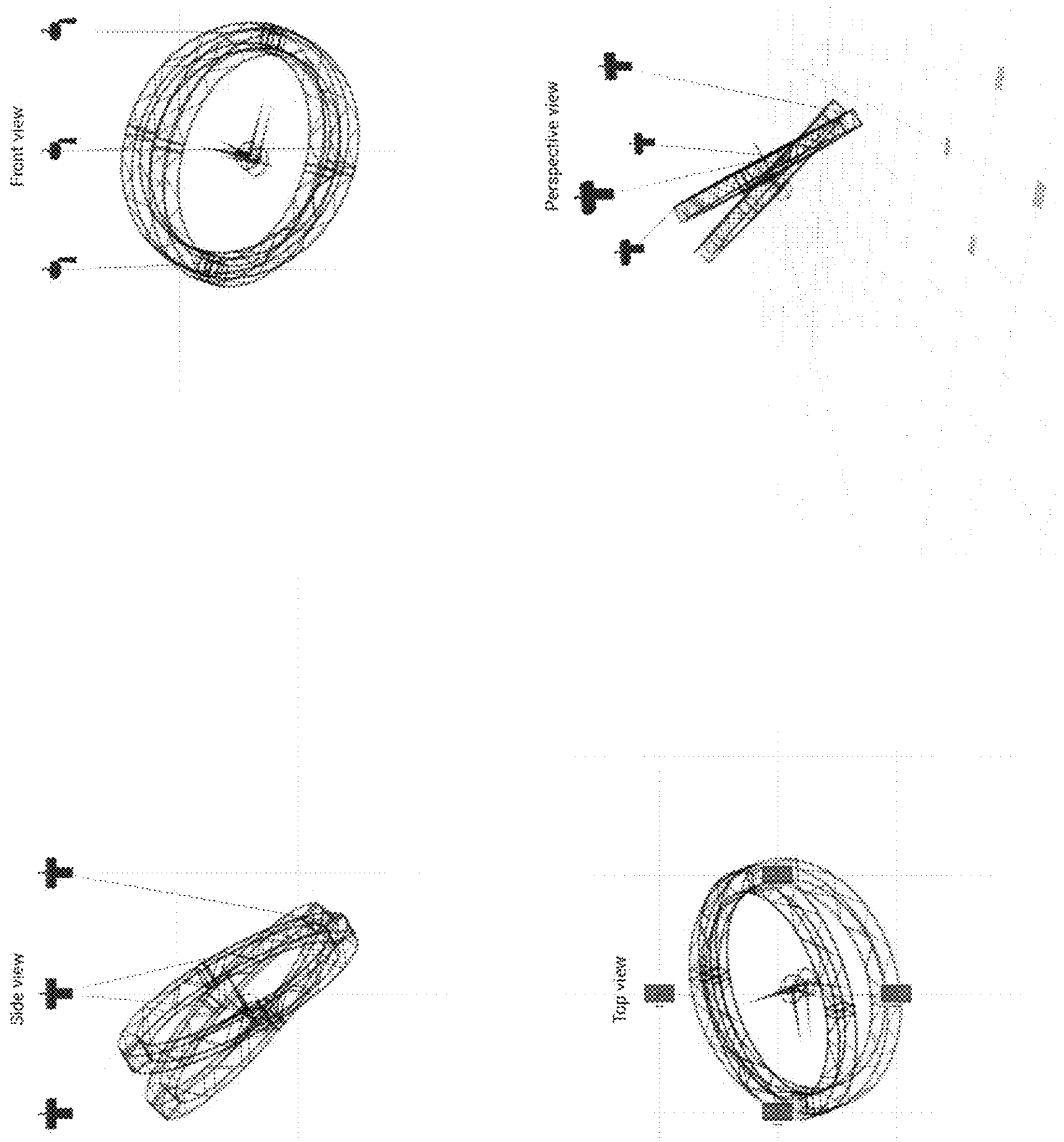


FIG. 13A

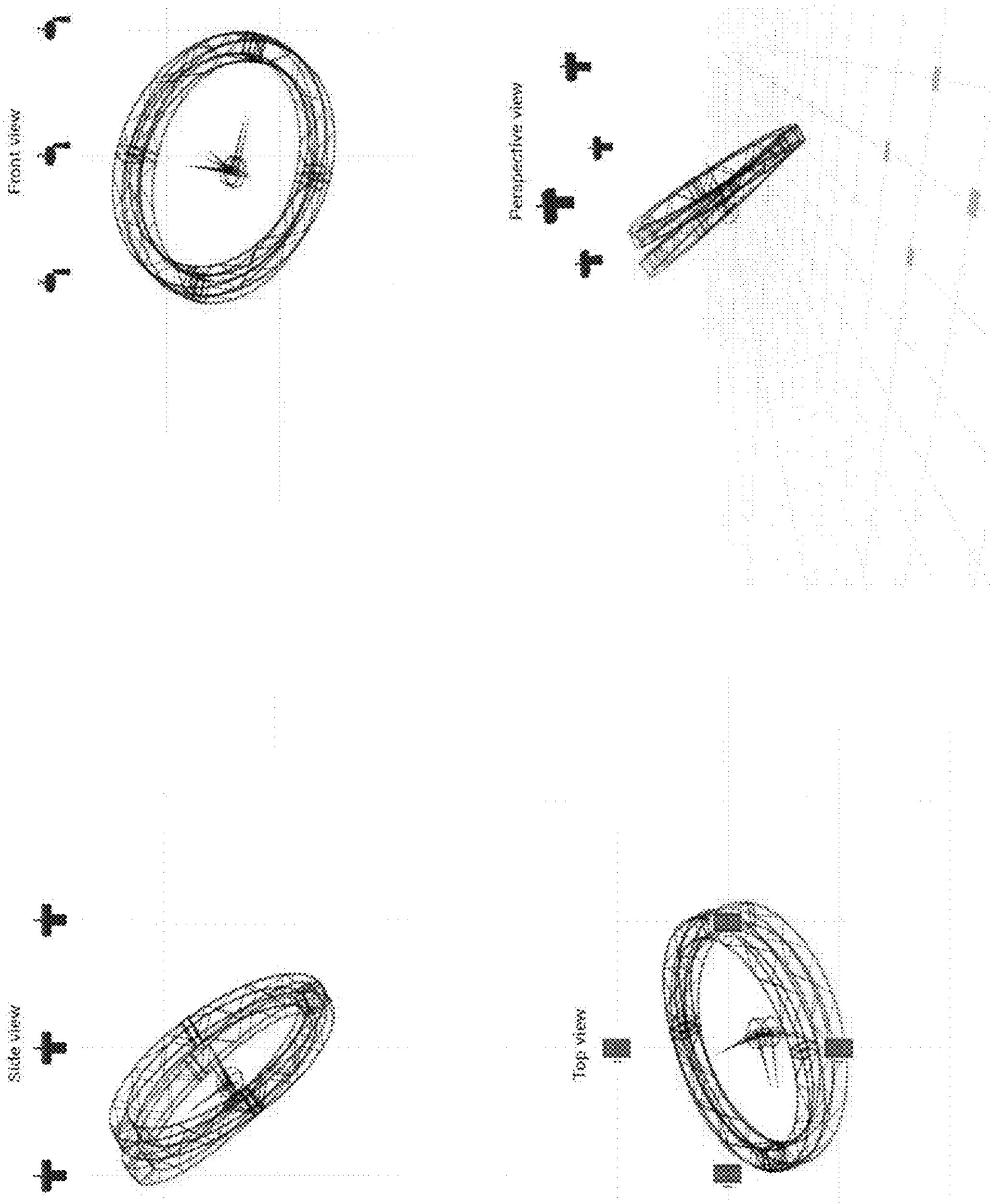


FIG. 13B

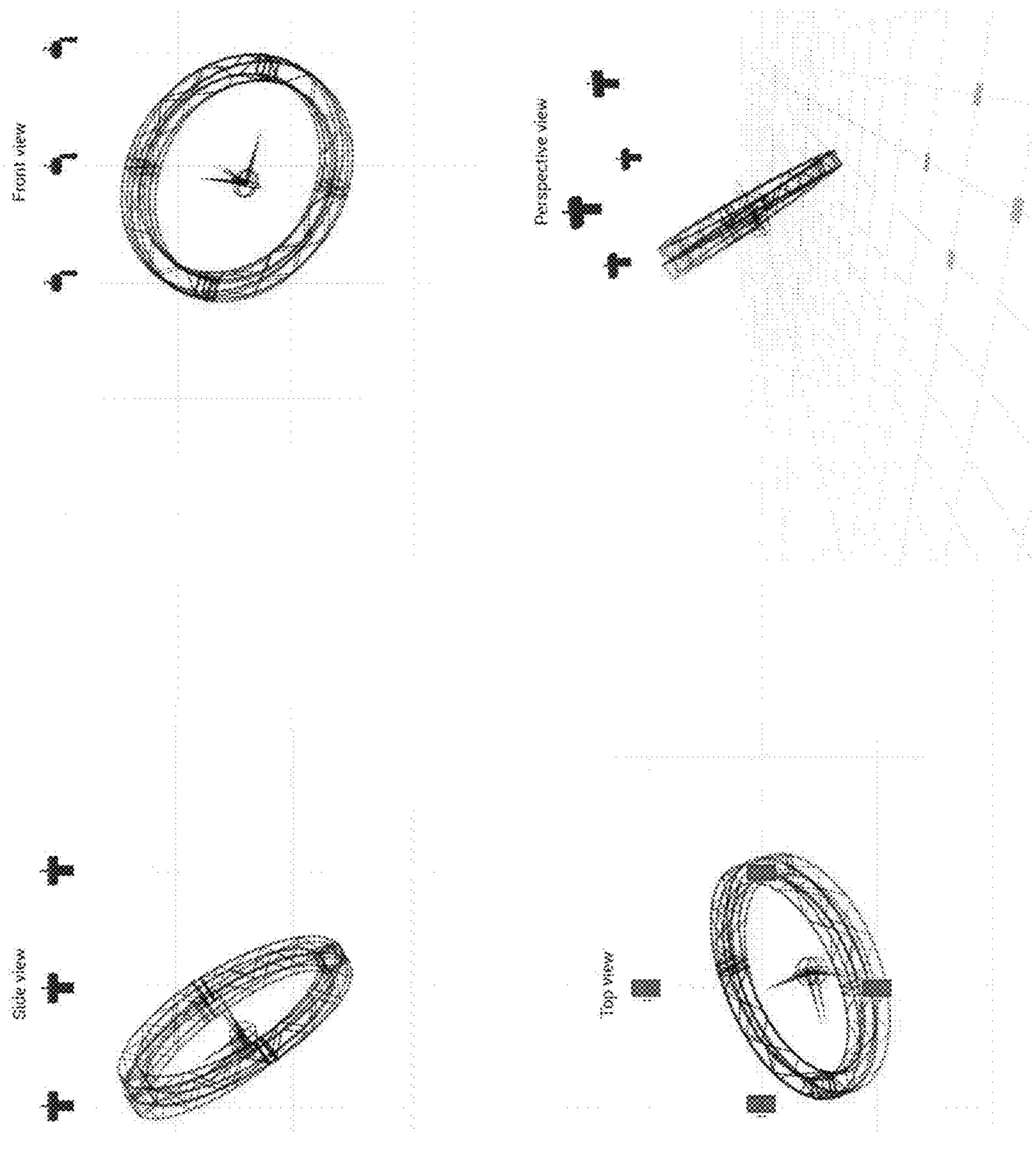


FIG. 13C

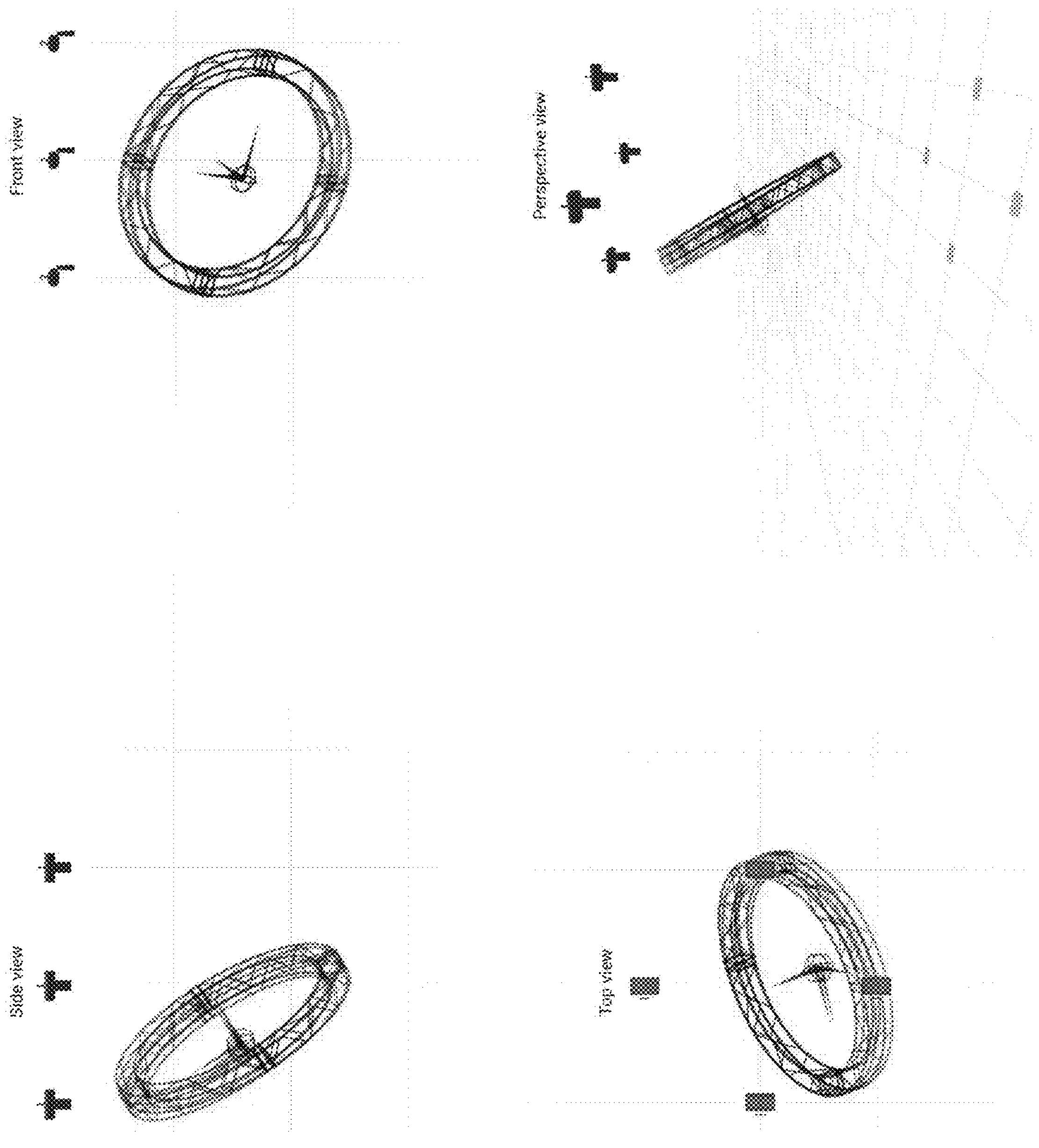


FIG. 13D

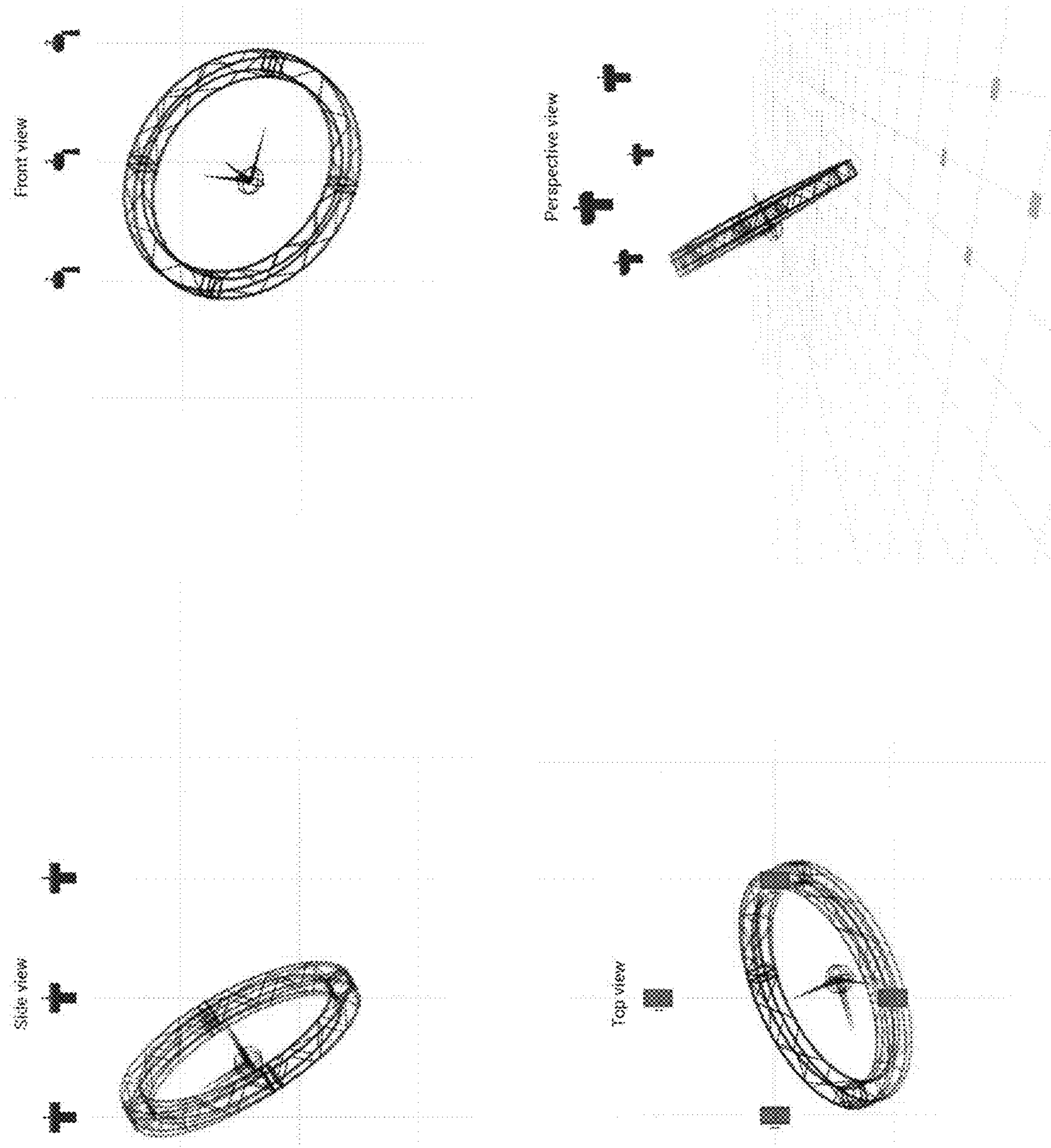


FIG. 13E

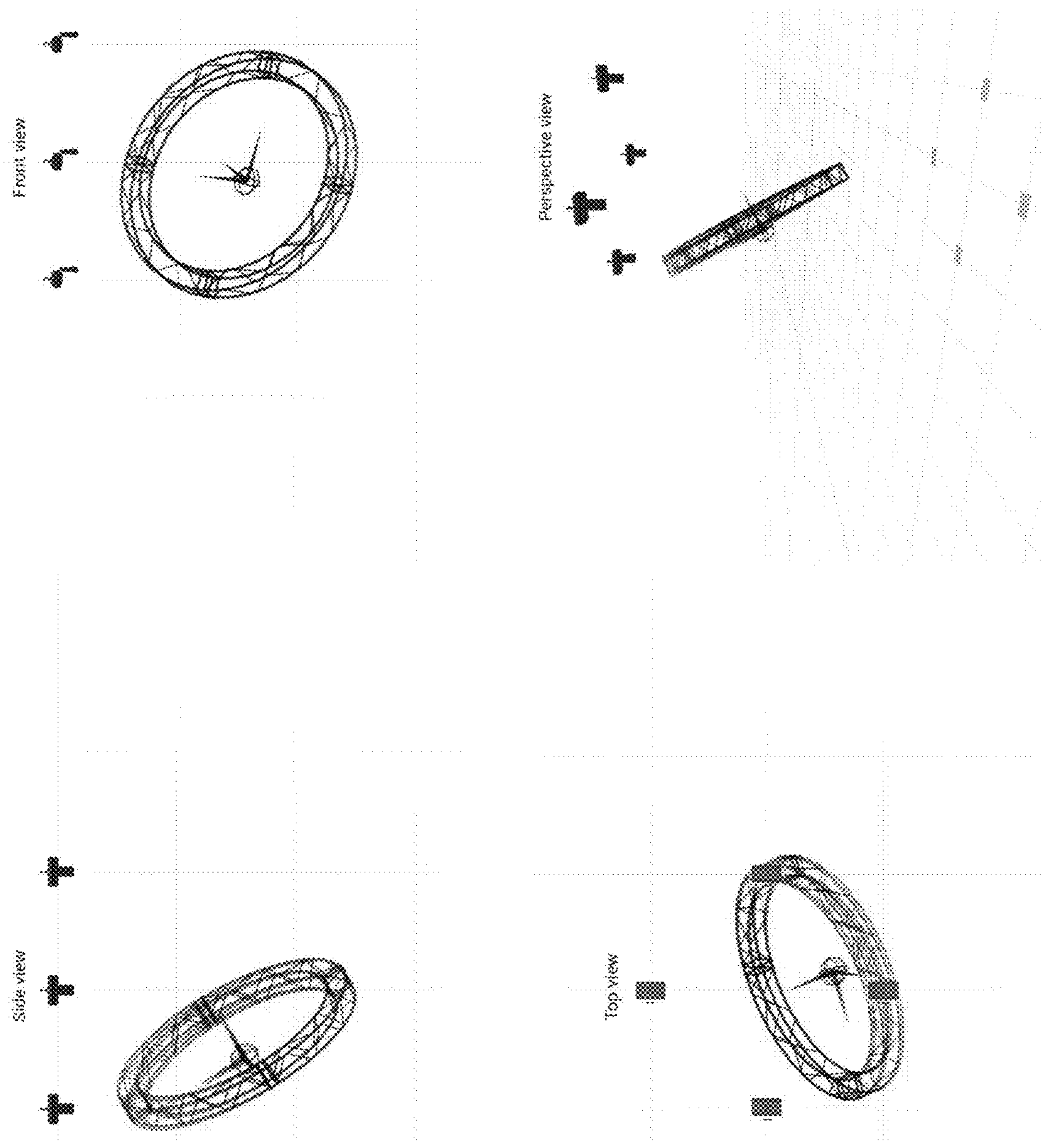


FIG. 13F

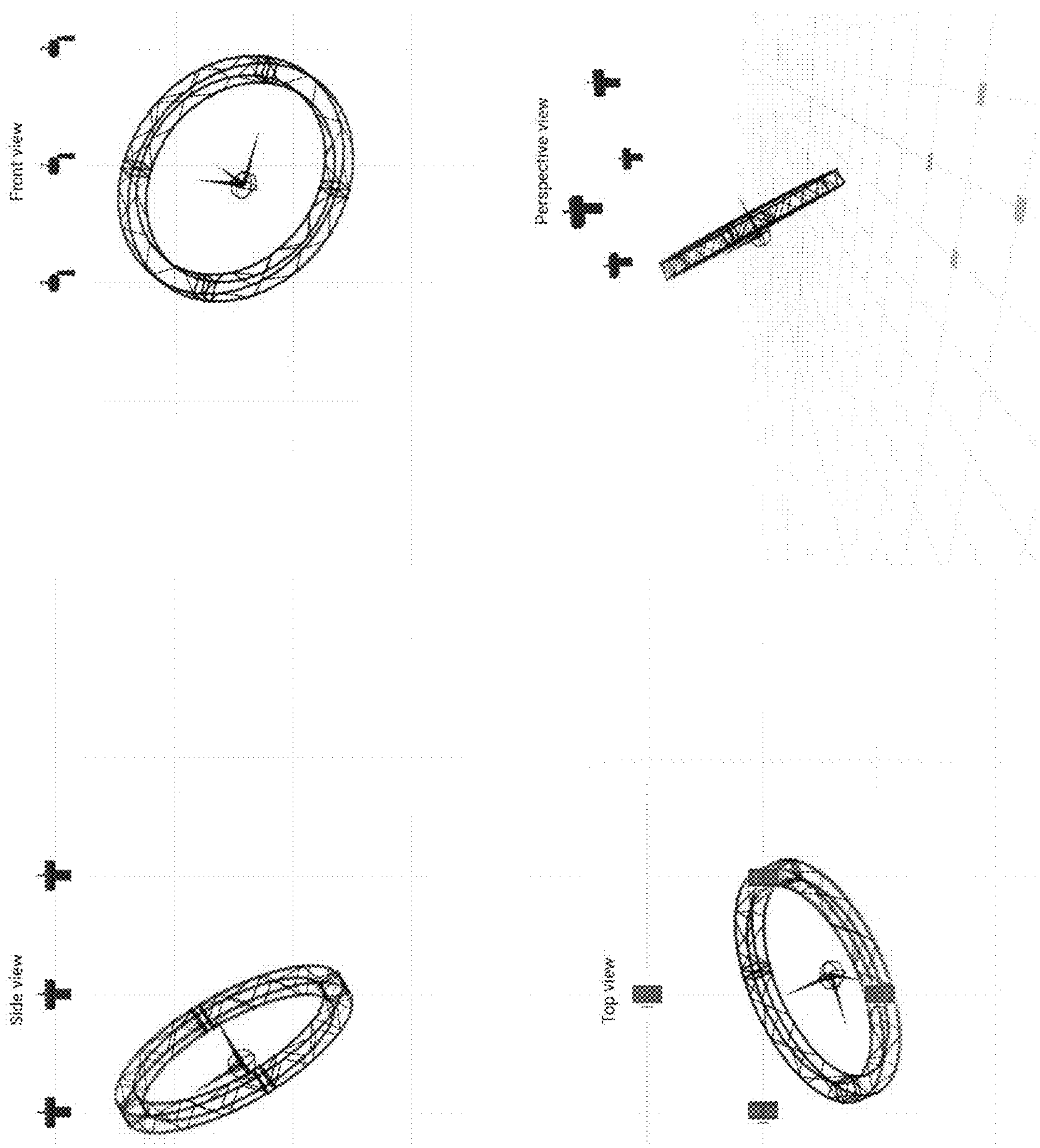


FIG. 13G

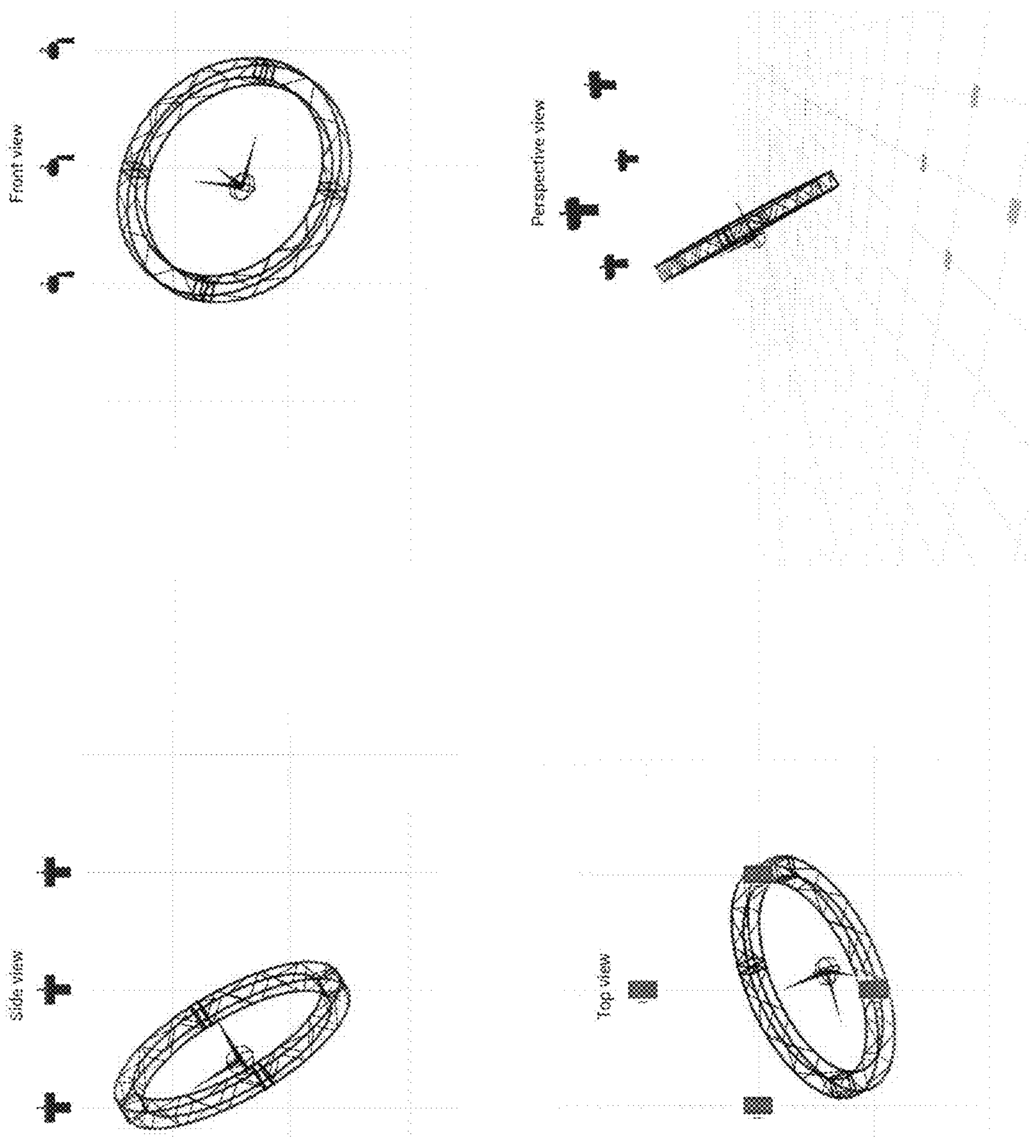


FIG. 13H

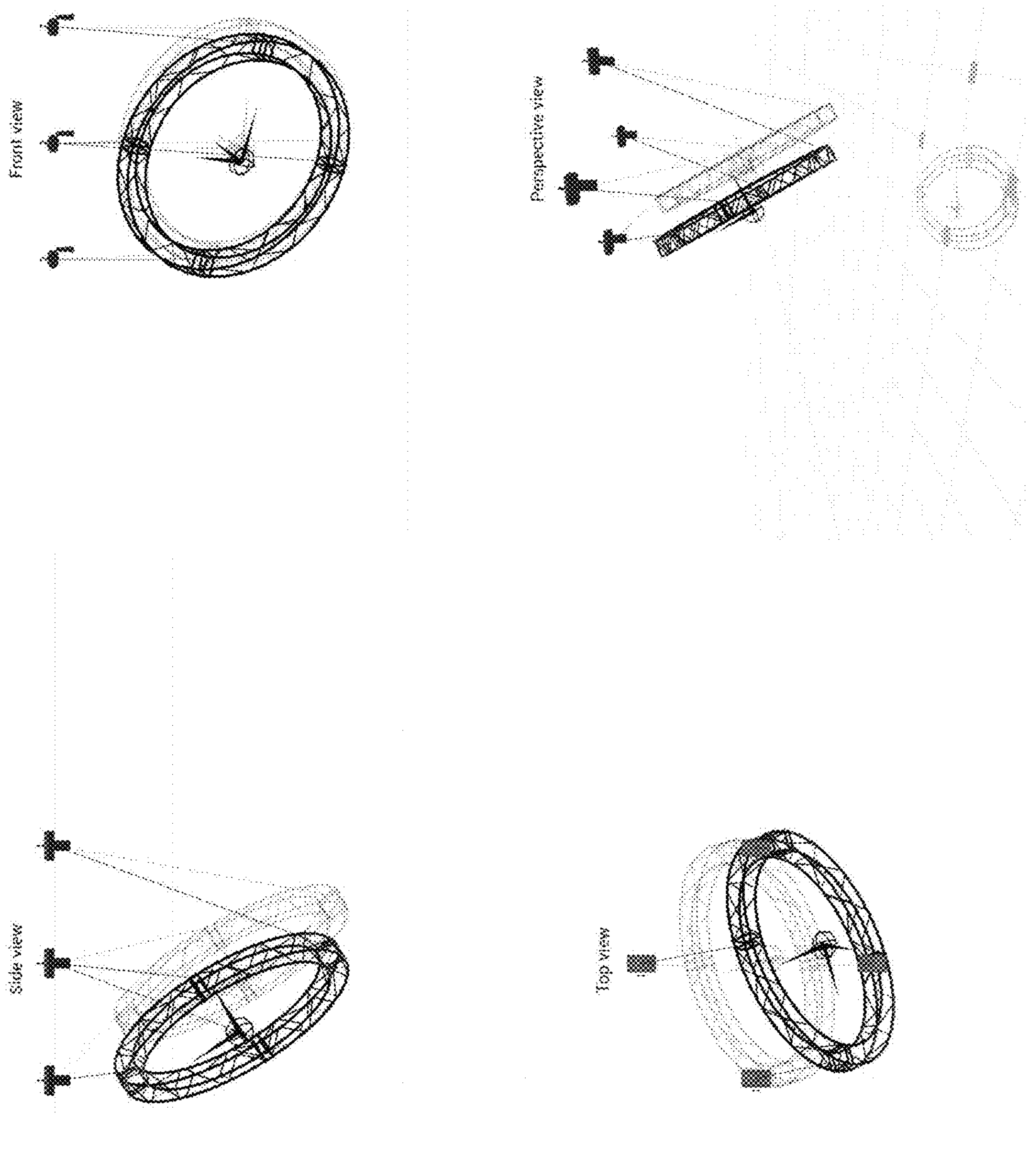


FIG. 13I

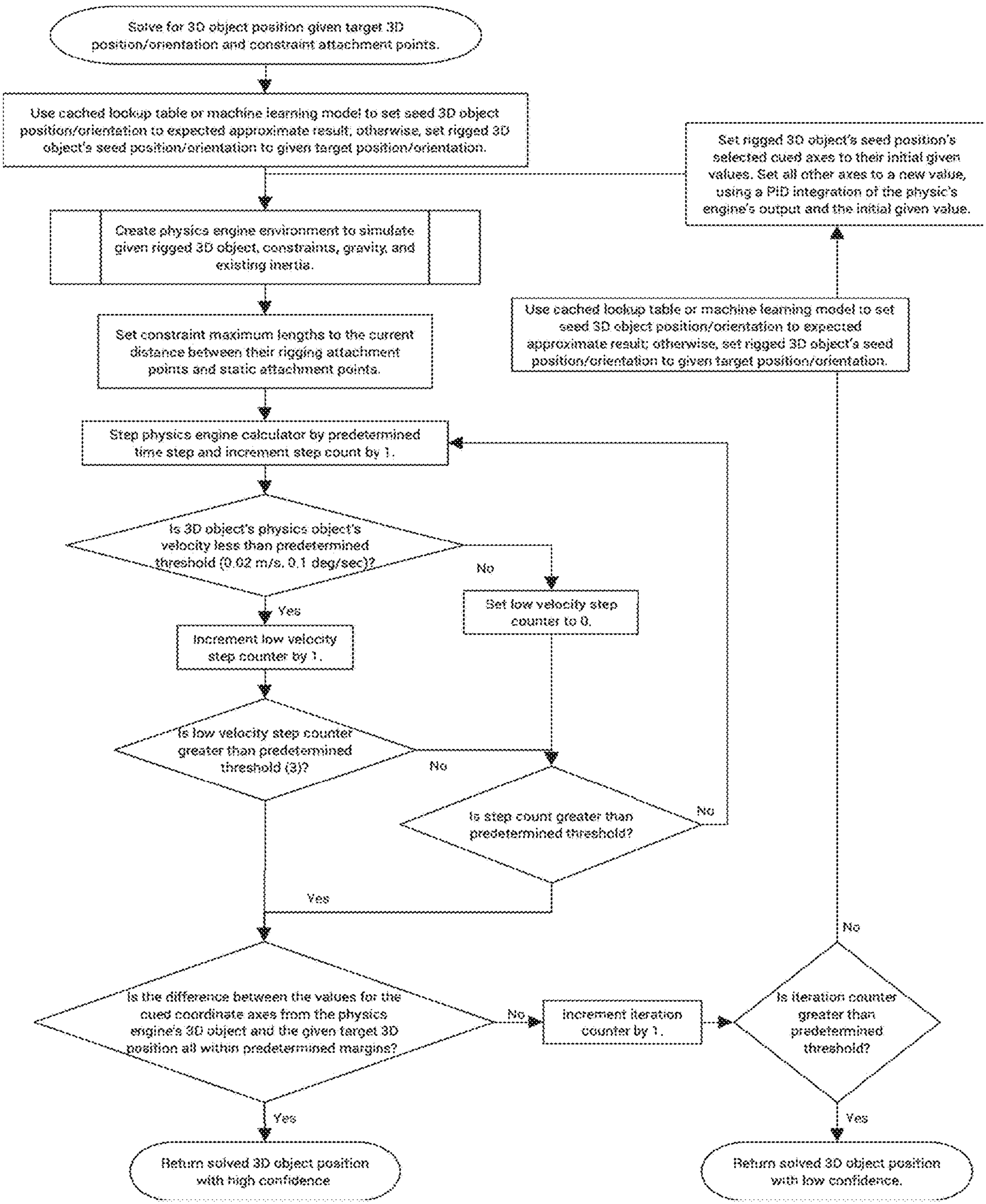

Solve for 3D object position given target 3D position/orientation and constraint attachment points.
Use cached lookup table or machine learning model to set seed 3D object position/orientation to expected approximate result; otherwise, set rigged 3D object's seed position/orientation to given target position/orientation.
Set rigged 3D object's seed position's selected cued axes to their initial given values. Set all other axes to a new value, using a PID integration of the physic's engine's output and the initial given value.
Create physics engine environment to simulate given rigged 3D object, constraints, gravity, and existing inertia.
Use cached lookup table or machine learning model to set seed 3D object position/orientation to expected approximate result; otherwise, set rigged 3D object's seed position/orientation to given target position/orientation.
Set constraint maximum lengths to the current distance between their rigging attachment points and static attachment points.
Step physics engine calculator by predetermined time step and increment step count by 1.
Is 3D object's physics object's velocity less than predetermined threshold (0.02 m/s, 0.1 deg/sec)?
No
Set low velocity step counter to 0.
Yes
Increment low velocity step counter by 1.
Is low velocity step counter greater than predetermined threshold (3)?
No
Is step count greater than predetermined threshold?
No
Yes
No
Is the difference between the values for the cued coordinate axes from the physics engine's 3D object and the given target 3D position all within predetermined margins?
No
Increment iteration counter by 1.
Is iteration counter greater than predetermined threshold?
Yes
Yes
Return solved 3D object position with high confidence.
Return solved 3D object position with low confidence.

FIG. 14

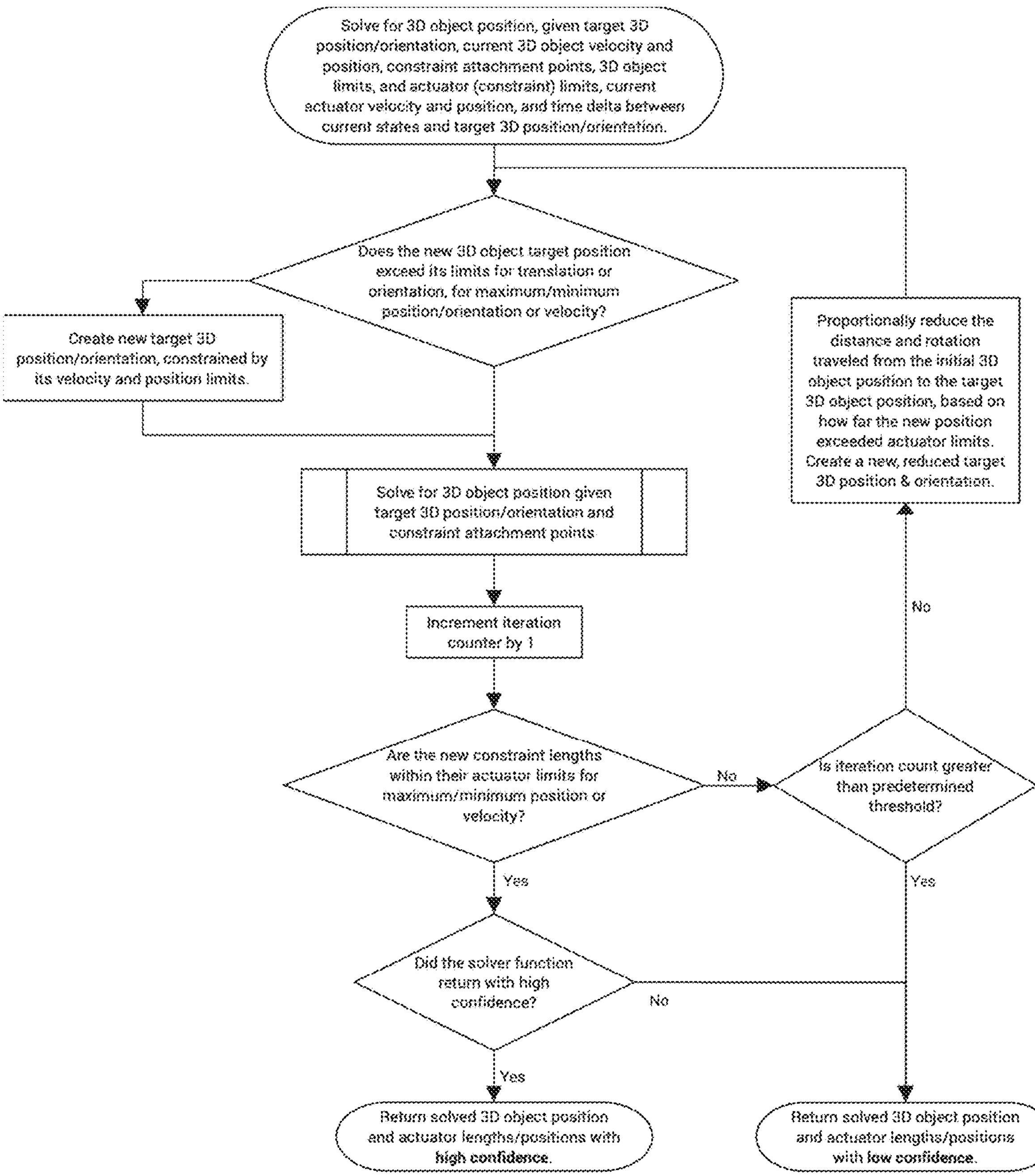

Solve for 3D object position, given target 3D position/orientation, current 3D object velocity and position, constraint attachment points, 3D object limits, and actuator (constraint) limits, current actuator velocity and position, and time delta between current states and target 3D position/orientation.
Does the new 3D object target position exceed its limits for translation or orientation, for maximum/minimum position/orientation or velocity?
Create new target 3D position/orientation, constrained by its velocity and position limits.
Solve for 3D object position given target 3D position/orientation and constraint attachment points
Increment iteration counter by 1
Are the new constraint lengths within their actuator limits for maximum/minimum position or velocity?
No
Is iteration count greater than predetermined threshold?
No
Proportionally reduce the distance and rotation traveled from the initial 3D object position to the target 3D object position, based on how far the new position exceeded actuator limits. Create a new, reduced target 3D position & orientation.
Yes
Yes
Did the solver function return with high confidence?
No
Yes
Return solved 3D object position and actuator lengths/positions with high confidence.
Return solved 3D object position and actuator lengths/positions with low confidence.

FIG. 15

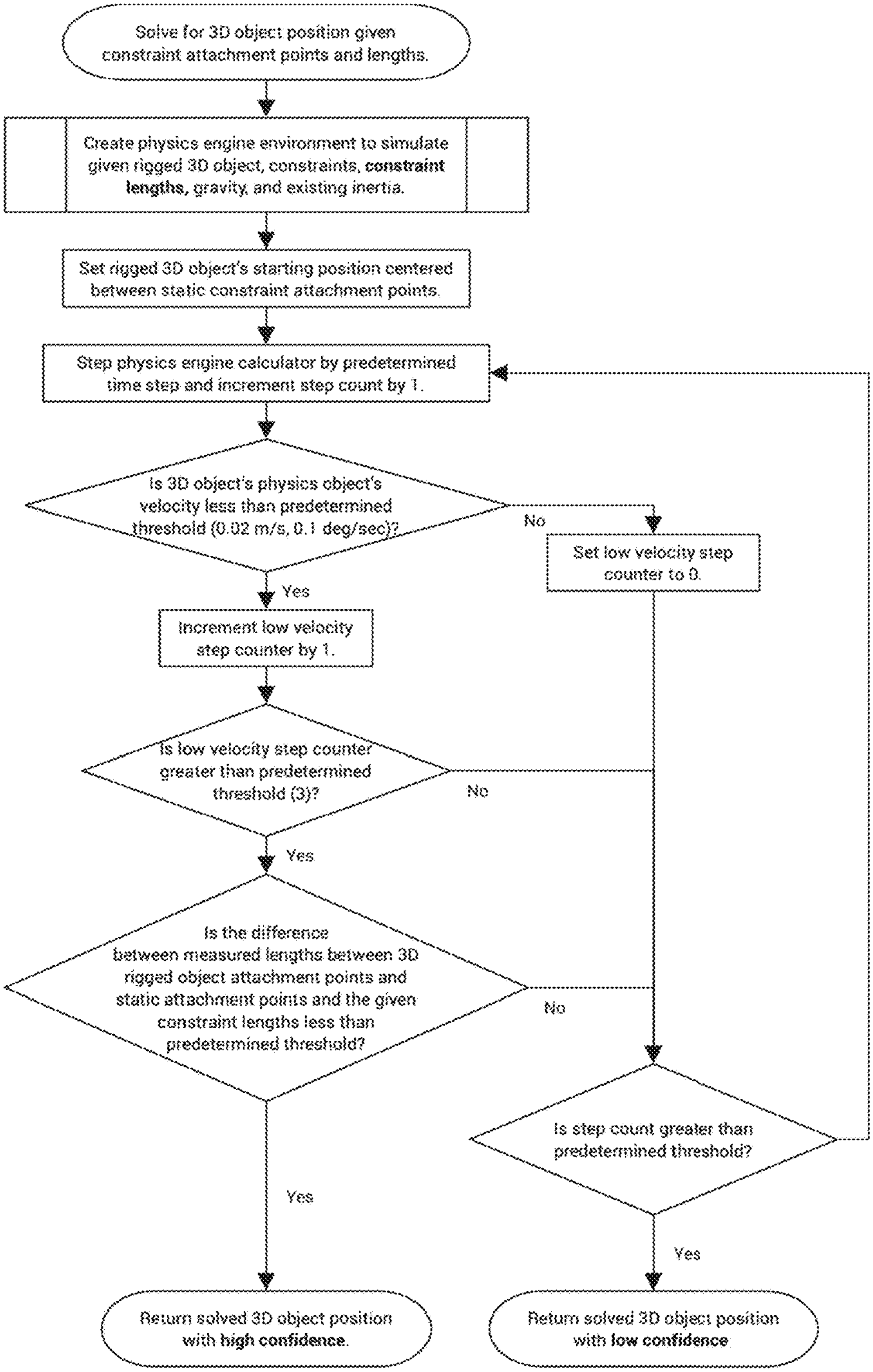

Solve for 3D object position given constraint attachment points and lengths.
Create physics engine environment to simulate given rigged 3D object, constraints, constraint lengths, gravity, and existing inertia.
Set rigged 3D object's starting position centered between static constraint attachment points.
Step physics engine calculator by predetermined time step and increment step count by 1.
Is 3D object's physics object's velocity less than predetermined threshold (0.02 m/s, 0.1 deg/sec)?
No
Set low velocity step counter to 0.
Yes
Increment low velocity step counter by 1.
Is low velocity step counter greater than predetermined threshold (3)?
No
Yes
Is the difference between measured lengths between 3D rigged object attachment points and static attachment points and the given constraint lengths less than predetermined threshold?
No
Is step count greater than predetermined threshold?
Yes
Yes
Return solved 3D object position with high confidence.
Return solved 3D object position with low confidence

FIG. 16

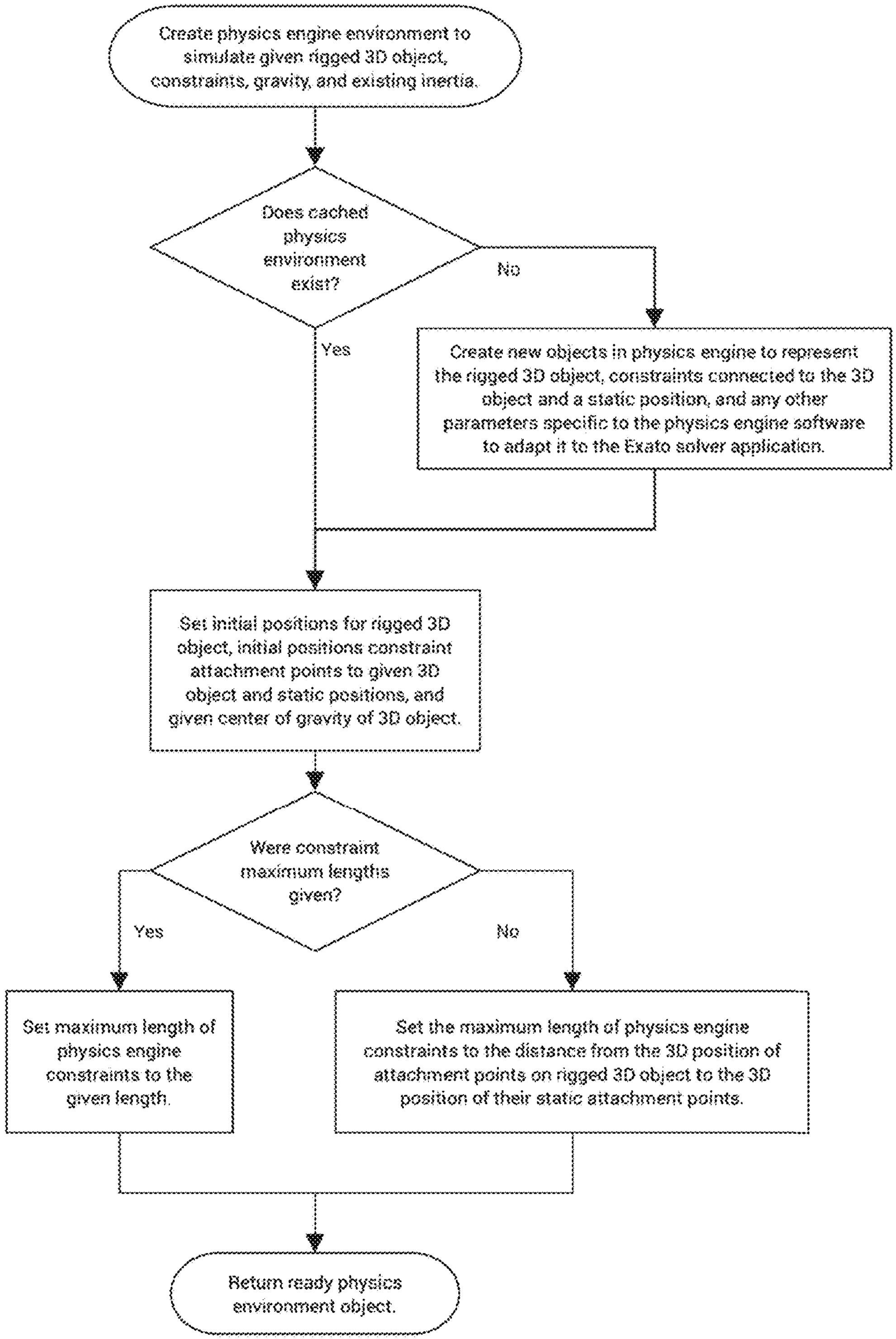

Create physics engine environment to simulate given rigged 3D object, constraints, gravity, and existing inertia.
Does cached physics environment exist?
No
Yes
Create new objects in physics engine to represent the rigged 3D object, constraints connected to the 3D object and a static position, and any other parameters specific to the physics engine software to adapt it to the Exato solver application.
Set initial positions for rigged 3D object, initial positions constraint attachment points to given 3D object and static positions, and given center of gravity of 3D object.
Were constraint maximum lengths given?
Yes
No
Set maximum length of physics engine constraints to the given length.
Set the maximum length of physics engine constraints to the distance from the 3D position of attachment points on rigged 3D object to the 3D position of their static attachment points.
Return ready physics environment object.

FIG. 17

STAGE AUTOMATION SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application claims priority under 35 U.S.C. § 119/120 to:

1) U.S. Provisional Patent Application Ser. No. 62/887,998, entitled STAGE AUTOMATION SYSTEM, filed Aug. 16, 2019, naming Ryan Mast as an inventor,
2) U.S. Provisional Patent Application Ser. No. 62/938,118, entitled STAGE AUTOMATION SYSTEM, filed Nov. 20, 2019, naming Ryan Mast as an inventor,
3) U.S. patent application Ser. No. 16/751,984 U.S. Pat. No. 11,385,610 B2, entitled STAGE AUTOMATION SYSTEM, filed Jan. 24, 2020, naming Ryan Mast as an inventor,
4) International Patent Application Serial No. PCT/US20/46476 entitled STAGE AUTOMATION SYSTEM, filed Aug. 14, 2020, naming Ryan Mast as an inventor, and
5) U.S. Provisional Patent Application Ser. No. 63/075,616, entitled STAGE AUTOMATION SYSTEM, filed Sep. 8, 2020, naming Ryan Mast as an inventor, each of which are incorporated herein by reference in the entirety to the extent not inconsistent herewith.

TECHNICAL FIELD

The present invention generally relates to a system and method for stage automation, and more particularly, a hardware and protocol agnostic system and method for stage automation.

BACKGROUND

In the context of live events, such as concerts, plays, and sporting events, a plurality of various components and machinery may be required to work in tandem with one another in order to properly execute all of the movements and features of the live event. For example, in the context of a concert, a stage automation system may include lights, stage elevators, and winches configured to move stage props which all must execute individualized commands in tandem (e.g., coordination) with one another throughout the duration of the concert. However, many stage automation systems include hardware from disparate manufacturers, which may be configured to run on disparate software programs and communication protocols. In this regard, conventional stage automation systems do not have an efficient mechanism with which to communicate and control each hardware device within the stage automation system.

For example, many traditional stage automation systems communicate with each other across a network (e.g., TCP/IP network, serial/bus-style network such as RS-485 or CAN-bus) by reading and writing indexed "registers" across the network. To get information on the position and velocity of a certain servo drive, a controller of traditional automation systems would need to be programmed for to perform common functions on a drive. For instance, turn on the motor, release the brake, and move forward for various servo drives, the controller may need to set three separate registers on a first servo drive manufactured by a first manufacturer, and two separate registers in a particular sequence on a second servo drive manufactured by a second manufacturer. In this example, in order for the controller to interact with both the first servo drive and the second servo drive, it would need to have a map of what register indexes perform certain functions, whether each servo drive operates in 16/32/64-bit, what endian type each servo drive uses for storing and transmitting numerical data, and include specific programming for how to perform specific functions on each of the first servo drive and the second servo drive.

As shown in the example above, traditional stage automation systems including hardware and software from disparate manufacturers require excessive, tedious programming to enable efficient coordination and communication between the various devices. Additionally, programming must be performed for each hardware/software device of the stage automation system, making conventional systems difficult to modify and/or expand. Therefore, there exists a need in the art for a system and method which cure one or more of the shortfalls of previous approaches identified above.

SUMMARY

A stage automation system is disclosed. In embodiments, the stage automation system includes a first executing program configured to selectively control a first actionable mechanism, and a second executing program configured to selectively control a second actionable mechanism. The stage automation system may further include a stage automation server configured to: receive, from the first executing program, a distributed program object announcement including a set of one or more time-stamped variables associated with the first actionable mechanism; generate a distributed program object associated with the first actionable mechanism, the distributed program object including the set of one or more time-stamped variables; transmit the distributed program object announcement to the second executing program; receive one or more data packets from the second executing program; adjust at least one time-stamped variable of the distributed program object stored in memory based on the one or more received data packets; and transmit one or more data packets to the first executing program. In embodiments, the one or more data packets are configured to cause the first executing program to adjust the at least one time-stamped variable of the distributed program object associated with the first actionable mechanism, and the first executing program is configured to adjust one or more characteristics of the first actionable mechanism associated with the at least one adjusted time-stamped variable.

A stage automation system is disclosed. In embodiments, the stage automation system includes a stage automation server communicatively coupled to one or more executing programs via a network protocol. In embodiments, the stage automation server is configured to: receive, from a first executing program, a distributed program object announcement including a set of one or more time-stamped variables associated with an actionable mechanism running on the first executing program; generate a distributed program object including the set of one or more time-stamped variables in memory; transmit the distributed program object to at least one additional executing program; receive one or more data packets from the at least one additional executing program; adjust at least one time-stamped variables of the set of one or more time-stamped variables of the distributed program object stored in memory based on the one or more received data packets; and transmit one or more data packets to the first executing program. In some embodiments, the one or more data packets are configured to cause the first executing program to adjust at least one time-stamped variable of the one or more time-stamped variables associated with the actionable mechanism and adjust one or more characteristics

of the actionable mechanism associated with the at least one adjusted time-stamped variable.

A method is disclosed. In embodiments, the method includes: receiving, from a first executing program, a distributed program object announcement including a set of one or more time-stamped variables associated with a first actionable mechanism selectively controlled by the first executing program; generating a distributed program object associated with the first actionable mechanism, the distributed program object including the set of one or more time-stamped variables; transmitting the distributed program object to a second executing program; receiving one or more data packets from the second executing program; adjusting at least one time-stamped variable of the distributed program object based on the one or more received data packets; transmitting one or more data packets to the first executing program indicative of the at least one adjusted time-stamped variable; and selectively adjusting one or more characteristics of the first actionable mechanism associated with the at least one adjusted time-stamped variable.

BRIEF DESCRIPTION OF THE DRAWINGS

The numerous advantages of the disclosure may be better understood by those skilled in the art by reference to the accompanying figures in which:

FIG. **1** illustrates a simplified block diagram of a stage automation system, in accordance with one or more embodiments of the present disclosure.

FIG. **2**A illustrates an announcement data packet of a stage automation system, in accordance with one or more embodiments of the present disclosure.

FIG. **2**B illustrates a first stage handshake data packet of a stage automation system, in accordance with one or more embodiments of the present disclosure.

FIG. **2**C illustrates a second stage handshake data packet of a stage automation system, in accordance with one or more embodiments of the present disclosure.

FIGS. **3**A-**3**C illustrate a conceptual diagram of a time-stamped variable with a publish variable mode and a time-stamped variable with an omni-publish variable mode, in accordance with one or more embodiments of the present disclosure.

FIG. **4** illustrates a graph **400** depicting position and velocity limits of a component executing commands, in accordance with one or more embodiments of the present disclosure.

FIG. **5** illustrates a conceptual diagram of a stage automation system, in accordance with one or more embodiments of the present disclosure.

FIGS. **6**A-**1**-**6**E-**2** illustrate conceptual diagrams of a stage automation system, in accordance with one or more embodiments of the present disclosure.

FIGS. **7**A-**7**E illustrate a three-dimensional (3D) tracking system of a stage automation system, in accordance with one or more embodiments of the present disclosure.

FIG. **8** illustrates a flowchart of a method for operating a stage automation system, in accordance with one or more embodiments of the present disclosure.

FIGS. **9**A-**13**I illustrate user interface views representative of object position modeling computations by a stage automation system.

FIGS. **14**-**17** illustrate process flow diagrams showing various processing operations for object position modeling computations.

DETAILED DESCRIPTION OF THE INVENTION

Reference will now be made in detail to the subject matter disclosed, which is illustrated in the accompanying drawings.

Referring generally to FIGS. **1**-**8**, the present disclosure is generally directed to a system and method for coordinated stage automation. In particular, embodiments of the present disclosure are directed to a stage automation system which is configured to unite actionable mechanisms (e.g., mechanical devices) manufactured by varying manufacturers (and therefore operated on varying software/executing programs) under a common communication umbrella. Additional embodiments of the present disclosure are directed to a method of controlling a stage automation system including a plurality of motion devices operating on varying executing programs.

It is contemplated herein that embodiments of the present disclosure may facilitate the efficient communication and cooperation of varying actionable mechanisms within a stage automation system. By communicating with each actionable mechanism and executing program within a stage automation system with a common communication protocol, embodiments of the present disclosure may enable highly coordinated and scalable stage automation systems.

The following descriptions may recite the following terms which may include, but are not limited to, the following characteristics:

- Physics engine. A physics engine may include software that may run on a CPU, GPU, or dedicated FPGA/ASIC that simulates how real-world objects react to physical forces. Most physics engines can calculate how objects will move with gravity, attached constraints (like ropes, pistons, joints, etc.), applied forces, friction, and collisions with other 3D objects.
- Constraint. For most of this description, this will be describing a linear constraint that may be attached to a 3D object. In most situations, this will be ropes or chains from winches or chain motors hooked to a flown object. Essentially, this is a machine that controls how far a reference point associated with the flown object is moved away from a given position either by lifting it up (e.g., via a winch), or by pushing it (e.g., with a piston).
- Vector3. This is 3 numeric values that describe a 3D position in space (e.g. an x/y/z position), or 3 values that make up an orientation described by Euler angles. Given two Vector3 positions and the time between them, we can calculate a 3D velocity which is also represented by a Vector3. For instance, if position 1 is [0,0,0] meters, position 2 is [1, 4, 10] meters, and the time between those samples is 2 seconds, we know that the velocity is [0.5, 2, 5]/second.
- Euler angles. Angles in the X, Y, & Z planes, respectively.
- Quaternion. A mathematical representation of the rotation or orientation of a 3D object that is far more complex and complete than Euler angles. It may be used for transitioning smoothly between two orientations in an animation.
- Axis & axes. In stage automation and industrial automation, a machine that creates motion is commonly called an "axis" or "axis of motion." In geometry, a direction for position or orientation like X, Y, or Z, is also generally also called an "axis," confusingly. For clarity in this document, a machine that creates motion (like a chain motor or hydraulic piston) is referred to as an

"actuator," so any usage of the term "axis" will refer to a geometric direction like position or Euler angles X, Y, or Z.

Limits. Limits are values that restrict how a machine operates. In stage automation, an actuator may have limits for maximum velocity, acceleration, and minimum/maximum position. This keeps the actuator from applying too much force, from going to fast, or from colliding with other objects in the space. Additionally, a 3D object may have separately-set limits on its position, velocity, and speed as calculated in 3D space. In the stage automation system, an object's velocity and acceleration at any given moment may be calculated as a length of a Vector3; that is, its velocity is calculated by its movement overall, not by individual X,Y,Z axes. A 3D object may also be functionally limited by the actuator's individual limitations, which may or may not be more constrictive than the limits set for the 3D object itself. If a 3D object tries to go faster than the actuators can actually move it in any given moment, the stage automation system may adjust down the overall velocity of the move at that moment to the maximum percentage of the requested move it can accomplish.

FIG. **1** illustrates a simplified block diagram of a stage automation system **100**, in accordance with one or more embodiments of the present disclosure. The stage automation system **100** may include, but is not limited to, one or more actionable assemblies **102**, one or more actionable mechanisms **104**, one or more executing programs **106**, a stage automation server **108**, and a user interface **114**.

In one embodiment, the stage automation system **100** may be used in the context of live performances in order to implement hardware and software-agnostic automation. For example, the stage automation system **100** may be utilized to control hardware (e.g., actionable assemblies **102**, actionable mechanisms **104**) in the context of concerts, plays, sporting events, and the like. In embodiments, each implementation of the stage automation system **100** may be defined/described as an "environment," such that the stage automation assembly **100** implemented in the context of a first concert is identified by a first environment, and the stage automation assembly **100** implemented in the context of a second concert is identified by a second environment. In embodiments, an environment may include at least a stage automation server **108** and one or more executing programs **106**. An environment may be defined by a unique UUID, a date and time the respective environment was initialized, and a running clock in seconds from when the environment was initialized. Data associated with each environment may be stored in a memory **112** of the stage automation server **108**. An environment may be further defined by a UUID of a configuration file which set up the particular environment. Generally, as it is used herein, the term "environment" may be regarded as an individual "session" of the stage automation system **100**.

The stage automation server **108** may include a local server/controller and/or a remote server/controller. For example, each environment may include a local server/controller which controls and stores data associated with the environment throughout the respective performance or event. Following the conclusion of the performance/event, the local server/controller (e.g., local stage automation server **108**) may be communicatively coupled to a remote server/controller (e.g., cloud-based stage automation server **108**) such that data associated with the environment may be uploaded and stored in a centralized, cloud-based server (e.g., cloud-based stage automation server **108**). It is noted herein that synchronizing local servers/controllers with a centralized, cloud-based server (e.g., cloud-based stage automation server **108**) may enable troubleshooting known issues, analysis to tune component issues, and machine learning analysis to generate predictive models around operators and various components.

In embodiments, the stage assembly system **100** may include one or more executing programs **106***a*, **106***b* (e.g., "instances") communicatively coupled to the stage automation server **108**. The one or more executing programs **106***a*, **106***b* may be communicatively coupled with the stage automation server **108** and/or other executing programs **106***a*, **106***b* via any network protocol known in the art. For example, the network protocol utilized by the stage automation system **100** may include, but is not limited to, a transmission control protocol (TCP) or internet protocol (IP) network (e.g., ethernet, WiFi), a serial or bus-style network (e.g., RS-485), and the like. By way of another example, the network protocol may include a socketless user datagram protocol (UDP). In embodiments, communicative connections established between components of the stage automation system **100** (e.g., executing program **106***a*, **106***b*, stage automation server **108**, and the like) are performed via the establishment of asymmetric and then symmetric encryption, such as datagram transport layer security (DTLS), transport layer security (TLS), and the like.

The one or more executing programs **106***a*, **106***b* may include software and/or code programs which are configured to control one or more actionable mechanisms **104** (e.g., devices). In embodiments, each executing software may be uniquely identified by a UUID, a UUID of the environment it exists within, and an internet protocol (IP) address. As noted previously herein, hardware and industrial mechanisms produced by various manufacturers may be run/controlled by varying software programs. In this regard, a first executing program **106***a* may include a software and code base of a first manufacturer executed on a first set of computers (e.g., servers, virtual machines, graphical user interfaces (GUI)), and a second executing program **106***b* may include a software and code base of a second manufacturer executed on a second set of computers. Each executing program **106***a*, **106***b* may include an individual runtime configuration. For example, the first executing program **106***a* may include a first software program with a first configuration and the second executing program **106***b* may include a second software program with a second configuration.

In embodiments, each executing program **106** is configured to selectively control one or more actionable mechanisms **104***a*-**104***n* (e.g., "axis" or "axes"). The one or more actionable mechanisms **104***a*-**104***n* may be communicatively coupled to the respective executing programs **106***a*-**106***n* via any wireline or wireless communication protocol known in the art (e.g., TCP network, serial connection, and the like). The one or more actionable mechanisms **104***a*-**104***n* may include any machinery or industrial motion controlling device known in the art including, but not limited to, a servo drive, a motor, a linear motor, a brake, a valve, an encoder, a solenoid, a light, a power source, and the like. For instance, an actionable mechanism **104***a* may include a Control Techniques M700 or a Kollmorgen AKD.

In embodiments, the one or more actionable mechanisms **104***a*-**104***n* are configured to selectively control one or more characteristics of one or more actionable assemblies **102***a*-**102***n*. In this regard, actionable assemblies **102***a*-**102***n* may be regarded as the physical "thing" or "device" which gets moved, and the actionable mechanisms **104** may be regarded as the "controller" or means through which the actionable assemblies **102***a*-**102***n* are moved. For example, the one or more actionable assemblies **102***a*-**102***n* may include, but are not limited to, a stage elevator, a lineset, a screen track, a chain motor, a winch, a turntable, and the like. Operational characteristics of the actionable assemblies **102***a*-**102***n* which may be selectively controlled by the actionable mechanisms **104***a*-**104***n* may include, but are not limited to, a position/location, velocity, acceleration, deceleration, operational state (e.g., "active/on," "inactive/off"), voltage, wattage, current, fault state, and the like.

It is noted herein that each actionable assembly **102***a*-**102***n* and/or actionable mechanism **104***a*-**104***n* may be described as existing within three-dimensional (3D) space of a particular environment of the stage automation system **100**. 3D positional data of various actionable assemblies **102***a*-**102***n* and/or actionable mechanisms **104***a*-**104***n* may be defined by (x, y, z) coordinates. Rotational data may be defined, transmitted, and stored within the stage automation system **100** as quaternions, and presented to users as Euler (x, y, z) values. For simplicity throughout the present disclosure, the position (0, 0, 0) may be defined as down-stage center of a particular venue or location within which an environment of the stage automation system **100** is employed. In a similar manner, for the purposes of simplicity, upstage may correspond to positive z-values, elevations above a ground/floor may correspond to positive y-values, and stage left may correspond to positive x-values. It is further noted, however, that positional data may be shown and described in any manner known in the art and defined in relation to any frame of reference.

In embodiments, the stage automation server **108** may include one or more processors **110** and a memory **112**, the one or more processors **110** configured to execute a set of program instructions stored in memory **112**, the set of program instructions configured to cause the one or more processors **110** to carry out various steps of the present disclosure. For example, the one or more processors **110** of the stage automation server **108** may be configured to: receive, from a first executing program **106***a*, a distributed program object announcement including a set of one or more time-stamped variables associated with a first actionable mechanism **104***a*; generate a distributed program object associated with the first actionable mechanism **104***a*, the distributed program object including the set of one or more time-stamped variables; transmit the distributed program object to at least a second executing program **106***b*; receive one or more data packets from the second executing program **106***b*; adjust at least one time-stamped variable of the distributed program object stored in memory **112** based on the one or more received data packets; and transmit one or more data packets to the first executing program **106***a*. Each of these steps will be addressed in turn.

In embodiments, the stage automation server **108** may be configured to receive, from a first executing program **106***a*, a distributed program object announcement (e.g., "XaObject," "XaValuesAnnouncement") including a set of one or more time-stamped variables (e.g., "XaValues") associated with a first actionable mechanism **104***a*. For example, a first executing program **106***a* associated with a first actionable mechanism **104***a* may transmit a distributed program object announcement including a set of one or more time-stamped variables associated with the first actionable mechanism **104***a*.

Distributing program objects (e.g., "XaObjects") may include objects which read, log, and command functionality within the stage automation system **100**. Each executing program **106***a* may transmit a distributed program object announcement associated with each respective actionable mechanism **104***a*-**104***n* indicating that the respective actionable mechanism **104***a*-**104***n* exist within the environment. A distributed program object may include any number of time-stamped variables associated with a component (e.g., actionable mechanism **104***a*), cue, or command of the stage automation system **100**.

As it is used herein, "distributed program object announcements" may be regarded as initial "announcement" data packets of data which are transmitted by a particular component in order to announce the component's existence within a particular environment. More specifically, a distributed program object announcement (e.g., announcement data packet) may announce to other components of the stage automation system one or more time-stamped variables which the particular executing program **106** and/or actionable mechanism has to publish, or wishes to consume. An announcement data packet may include a list of universally unique identifiers (UUIDs) the sending component includes, and definitions for each UUID. A "definition" within an announcement data packet may be composed of the following byte string illustrated in Table 1. The number of definition byte arrays which may be included within a single announcement data packet may be dependent upon the maximum transmission unit (MTU) and maximum packet length of the stage automation system **100**.

TABLE 1

"Definition" Byte Array Structure of Announcement Data Packet

Starting Byte	Bytes	Type	Content
0	16	UUID	Time-stamped variable UUID
16	16	UUID	Sending component UUID
32	1	Byte Value Type	Value Type enum for the time-stamped variable (e.g., bool, float, double, string, bytes, GeometryPoint)
33	1	Byte BaseUnit	BaseUnit enum for the time-stamped variable
34	1	Byte Variable Mode	Variable Mode for the time-stamped variable (e.g., publish, subscribe, omni-publish)
35	1		Reserved for future use

The one or more time-stamped variables associated with the first actionable mechanism **104***a* may include variables associated with any characteristics (e.g., operational characteristics) of the first actionable mechanism **104***a* including, but not limited to, a position of the first actionable mechanism **104***a* at a point in time, a velocity of the first actionable mechanism **104***a* at a point in time, a brake engagement status of the first actionable mechanism **104***a* at a point in time, a connection status of the first actionable mechanism **104***a* at a point in time, a command or cue associated with the first actionable mechanism **104***a* at a point in time, and the like. By way of another example, time-stamped variables may define other characteristics of a particular component including, but not limited to, start/initiation time, network lag, commands, target position/velocity/acceleration, velocity clamp, position clamp, order in a cue sheet, and the like. The one or more time-stamped variables may include a UUID defining the time-stamped variable, a UUID of the component (e.g., actionable mechanism **104***a*) with which the value is associated, a unit of measurement, and time-stamps indicating when each time-stamped variable was captured.

In embodiments, each distributed program object (e.g., distributed program object announcement) may include a set of one or more value types (e.g., "XaValueType") defining types of values associated with each time-stamped variable. In this regard, each time-stamped variable of a distributed program object may be defined by a value type. Value types may include, but are not limited to, a Boolean value type (true/false), a long integer value type (numeric), a double float value type (numeric), a string value type (text), a byte array value type (data), and the like. For example, a time-stamped variable defining a DC bus voltage feedback of an actionable mechanism **104** may be defined by a value type of a double float value type (e.g., "322.12 V"). By way of another example, a time-stamped variable defining a status of a brake (e.g., actionable mechanism **104**) may be defined by a Boolean value type (e.g., "true" for open, or "false" for closed). By way of another example, a time-stamped variable defining raw encoder feedback from a machine (e.g., actionable mechanism **104**) may be defined by a long integer value type (e.g., "103923"). By way of another example, a time-stamped variable may define a make and/or model reported by a drive (e.g., actionable mechanism **104**) as a string value type (e.g., "Kollmorgen AKD-TBAN0607"). By way of another example, a time-stamped variable defining a UUID of a drive (e.g., actionable mechanism **104**) may be defined by a byte array value type in hex (e.g., "2bd2fb25b76849288fbe49874b610287").

In embodiments, each distributed program object (e.g., distributed program object announcement) may include a set of one or more variable modes defining various modes associated with each time-stamped variable. In this regard, each time-stamped variable of a distributed program object may be defined by a variable mode. Variable modes associated with each time-stamped variable may define how other components within the stage automation system **100** may view and/or interact with each respective time-stamped variable. Variable modes may include, but are not limited to, a "publish" variable mode, a "subscribe" variable mode, and an "omni-publish" variable mode.

For example, a publish variable mode indicates that the component associated with the time-stamped variable is the only component which may adjust/update the value. For instance, if an actionable mechanism **104** include a time-stamped variable indicative of current consumption for the actionable mechanism **104** with a publish variable mode, only the actionable mechanism **104** itself may adjust/update the time-stamped variable, with other components being unable to do so. By way of another example, a subscribe variable mode indicates that components which depend on the time-stamped variable (e.g., "subscribe" to the time-stamped variable) may receive published updates when the time-stamped variable is updated. For instance, if a second executing program **106***b* subscribes to a time-stamped variable with a subscribe value type of the first executing program **106***a*, the stage automation server **108** may transmit data packets/distributed program objects to the second executing program **106***a* each time the time-stamped variable is updated. Components may subscribe or not subscribe to other distributed program objects and/or individual time-stamped variables based on their relevance to their own operations. By way of another example, an omni-publish variable mode indicates that a plurality of components may adjust/update the value, and published updates are sent to subscribed components. For instance, a brake (e.g., actionable mechanism **104**) may include a time-stamped variable indicative of operational status (e.g., on/off) with an omni-publish variable mode such that any component within the stage assembly system **100** may adjust/update the operational status of the brake (e.g., turn it on or off).

As noted previously herein, an announcement data packet (e.g., distributed program object announcement data packet) transmitted by an executing program **106** may include one or more time-stamped variables which the particular executing program **106** and/or actionable mechanism has to publish, or wishes to consume. For example, an announcement data packet transmitted by the first executing program **106***a* may include definitions for a first distributed program object associated with the first actionable mechanism **104***a*, a second distributed program object associated with the second actionable mechanism **104***b*, a third distributed program object associated with the third actionable mechanism **104***c*, a fourth distributed program object associated with the actionable assembly **102***a*, and the like. Announcement data packets may describe the type of each component (e.g., actionable mechanism **104**, actionable assembly **102**), which time-stamped variables are associated with each respective distributed program object, and a value type of each time-stamped variable. For example, an announcement data packet is shown in further detail in Table 2 below:

TABLE 2

Announcement Data Packet Structure

Starting Byte	Bytes	Type	Content
0	16	UUID	Distributed program object UUID
16	16	UUID	Host executing program UUID
32	2	Unit XaObjectType	Value type enum for the distributed program object's type on the host executing program
34	2	Unit XaObjectType	Value type enum for how the distributed program object should be represented on the receiving executing program.
36	10	—	Reserved for future use
46	2	ushort	Number of time-stamped variables that will be described.
48	16	UUID	Time-stamped variable UUID for Value 1
64	2	ushort	Number of following bytes that create the key string for Value 1 (in this example, 27)
66	27	UTF-16string	Key name for the time-stamped variable in the distributed program object for Value 1
93	16	UUID	Time-stamped variable UUID for Value 2
109	2	ushort	Number of following bytes that create the key string for Value 2 (in this example, 491)
111	491	UTF-16string	Key name for the time-stamped variable in the distributed program object for Value 2

As shown in the table above, it is noted herein that time-stamped variables with string value type may include text encoded with UTF-16 such that the number of bytes for each time-stamped variable key may be more than the number of text characters.

In embodiments, the stage automation server **108** is configured to generate a distributed program object associated with the first actionable mechanism **104***a*, the distributed program object including the set of one or more time-stamped variables. In this regard, the stage automation server **108** may be configured to generate and/or store the distributed program object received from the first actionable mechanism **104***a* in memory **112**. Accordingly, time-stamped variables associated with actionable mechanism

104*a* (e.g., position, velocity, operational status) may be stored in memory **112**. Additionally, in some embodiments, executing programs **106***a*-**106***b* may transmit distributed program objects including sets of time-stamped variables and/or individual time-stamped variables at regular and/or irregular intervals. For example, the first executing program **106***a* may be configured to transmit a distributed program object including a set of time-stamped variables associated with actionable mechanism **104***a* every ten milliseconds (10 ms). In this regard, time-stamped variables associated with actionable mechanism **104***a* (e.g., position, velocity, operational status) may be updated and stored in memory **112** every ten milliseconds. The memory **112** may be configured to store all time-stamped variables associated with each component of the stage automation system **100** throughout the existence of an environment such that historical values throughout the environment may be easily searched and retrieved.

In the context of components (e.g., actionable assembly **102**, actionable mechanism **104**, executing program **106**, stage automation server **108**) which subscribe to one or more time-stamped variables of other components, the subscribing components may store time-stamped variables of the subscribing instances in memory. In this regard, components may store their own time-stamped variables, as well as time-stamped variables to which they are subscribed. Components may additionally be configured to transmit request packets via the network protocol in order to request specific time-stamped variables of other components from the stage automation server **108**. In additional embodiments, components of the stage assembly system **100** may be configured to sync recorded/stored time-stamped variables with those recorded/stored in the memory **112** of the stage automation server **108**.

In embodiment, some components may "depend" on other components or time-stamped variables such that the actions of a first component "depend" on a time-stamped variable of a second component. In this example, the time-stamped variable of the second component may be said to be "critical" to the first component. By subscribing to critical time-stamped variables of other components, the stage automation system **100** may enable a distributed, interconnected network of components with built-in fault states and commands for coordinated motion. Furthermore, as will be described in further detail herein, if the stage automation server **108** and/or another component (e.g., executing program **106**, actionable mechanism **104**, actionable assembly **102**) is dependent on a critical packet, distributed program object, or time-stamped variable of another critical component within a predefined window (e.g., within a 20 ms window), and the component does not receive the critical packet, distributed program object, or time-stamped variable within the predefined window, the component may be configured to go into a fault state.

It is further noted herein that storing a complete (or semi-complete) database of time-stamped variables generated by the environment throughout the duration of an event may be particularly useful for troubleshooting. Currently, stage automation systems do not store a complete set of data throughout an event, such as a concert. Thus, when weird anomalies occur during the concert (e.g., stage elevator goes past height clamp, light strays from intended planned course), these anomalies are difficult to troubleshoot, and almost impossible to reproduce. By storing time-stamped variables in a central stage automation server **108**, embodiments of the present disclosure may enable improved troubleshooting and stage automation management.

It is noted herein that components within the stage automation system **100** may each include one or more processors and memory. For example, each executing program **106***a*-**106***n* may include one or more processors and memory, such that each executing program **106***a*-**106***n* may be configured to store time-stamped variables associated with coupled components in memory. In this regard, time-stamped variables associated with a particular actionable mechanism **104** may be stored in a memory of a coupled executing program **106**, the stage automation server **108**, and the like. Similarly, each component within the stage automation system **100** (e.g., stage automation server, executing program **106**) may include a message broker configured to transmit and/or receive time-stamped variable updates and distributed program objects throughout the stage automation system **100**.

In embodiments, acknowledgement (ACK) and non-acknowledgement (NACK) packets may be handled and transmitted within the application layer of the stage automation system **100** such that non-received data packets may be identified by the UUID of the transmitting executing program **106** and an unsigned long integer value for the sequence number, and re-sent.

Data packets including distributed program objects and/or time-stamped variables transmitted via the network protocol of the stage automation system **100** may be used to monitor network lag and to ensure runtime configurations (e.g., timecodes) of each executing program **106** are in sync with that of the stage automation server **108**. For example, upon receiving a distributed program object from the first executing program **106**, the stage automation server **108** may transmit a data packet (ACK) including a timecode of the stage automation server **108** such that the first executing program **106***a* may ensure its timecode is in sync with that of the stage automation server **108**.

In embodiments, the stage automation server **108** is further configured to transmit the distributed program object announcement received from the first executing program **106***a* to at least a second executing program **106***b*. In embodiments, data packets (e.g., data packets of a distributed program object announcement) transmitted via the network protocol of the stage automation server **108** perform time-syncs when passed from one component to another in order to judge network jitter and latency. For example, a time synchronization protocol, such as simple network time protocol (SNTP), a network time protocol (NTP), or a precision time protocol (PTP) may be utilized by the stage automation server **108** in order to keep internal clocks of respective components in sync, and to monitor network congestion. In some embodiments, all communication (e.g., transmitting/receiving data packets) within the stage automation system **100** is performed over two ports, thereby simplifying network security management. In this regard, in some embodiments, functionality within the stage automation system **100** may be conducted over two network ports, instead of breaking different functions of the distributed network protocol into different network ports for different functions. For example, data packets may include byte identifiers associated with a particular functional purpose within the stage automation system **100** that the data packet serves. Additionally, byte identifiers may facilitate efficient decoding of the data packet.

It is contemplated herein that performing communication within the stage automation system **100** over only two network ports may provide a number of advantages. For example, some conventional stage automation systems may utilize a plurality of ports for different functions. For example, a conventional stage automation system may utilize a 4321 port for one type of audio traffic, a 5004 port for another type of audio traffic, 8700-8708 ports for control and monitoring, 319 and 320 ports for time sync, as well as additional ports. Comparatively, by utilizing two network ports, the stage automation system **100** of the present disclosure may allow components within the system to more efficiently and effectively identify traffic originating within the stage automation system **100**, and set up firewall rules to allow (or block) particular traffic.

In some embodiments, all data packets (e.g., data packets of distributed program objects) transmitted throughout the stage automation system **100** may include a common data header. An example data header of data packets transmitted throughout the network protocol of the stage automation system **100** is further shown in Table 3 below. In embodiments, values above the bolded dividing line may be transmitted unencrypted, and fully repeated in the encrypted packet. This may be done in order to validate that a received packet is from another component within the environment (e.g., another component with which the receiving component has established a communicative coupling with). Additionally, transmitting unencrypted values and repeating unencrypted values in the encrypted packet may further serve to verify that the data packet received has not been altered or corrupted following transmission.

TABLE 3

Data Packet Header

Starting Byte	Bytes	Type	Cast	Content	Section
0	2	String		"xA"	Short
2	1	Byte	Packet Type		Short
3	1	Byte	Component Type		Short
4	1	Byte		Version	Short
5	3	n/a		Reserved for future use	Short
8	16	UUID		Sending component's UUID	Short
24	16	UUID		Target component's UUID; left 0 for announcement packets.	Short
40	8	Double		Sender component timecode when packet was sent (this gets updated if the same packet is re-sent because of lack of acknowledgement from the receiving side)	Long
48	8	Double		Sender component timecode at last timecode received from target component (also gets updated on re-send)	Long
56	8	Double		Last timecode received from target component (gets updated on re-send)	Long
64	8	Ulong		Packet index	Long
72	8	n/a		Reserved for future use	Long

In some embodiments, all data packets (e.g., data packets of distributed program objects) transmitted throughout the stage automation system **100** may include the common short data header, as shown above in Table 3. As noted previously herein, the short header may always be transmitted unencrypted to enable debugging, coarse sanity checking before attempting decryption, and to enable traffic shaping, if necessary. Data packets transmitted via the network protocol throughout the stage automation system **100** may be further shown and described with reference to FIGS. **2**A-**2**C.

FIG. **2**A illustrates an announcement data packet **202***a* of a stage automation system **100**, in accordance with one or more embodiments of the present disclosure. The announcement data packet **202***a* may include, but is not limited to, a short header **204** and a data payload **206***a*.

In order to establish communicative couplings between various components (e.g., actionable assemblies **102**, actionable mechanisms **104**, executing programs **106**, stage automation server **108**) a series of announcement and handshake data packets may be exchanged between respective components. For example, FIG. **2**A illustrates an announcement packet which may be transmitted from a first component (e.g., "sending" component) to a second component (e.g., "receiving" or "target" component) in order to establish a communicative coupling between the first and second components. The announcement data packet **202***a* may be transmitted and/or broadcast with a short header **202** and a data payload **206***a* such that one or more receiving/target components may communicatively couple with the sending component. In embodiments, the data payload **206***a* may include an asymmetric public key (RSA) of the sending component such that receiving components may securely respond with an encrypted packet that only the sending component may decrypt.

It is noted herein that various components may transmit/receive data packets (e.g., data packets of distributed program objects) between one another through the stage automation server **108**. For example, a first executing program **106***a* may transmit data packets to the stage automation server **108**, which may then forward/transmit the data packets to a second executing program **106***b*. In additional and/or alternative embodiments, components may transmit/receive data packets directly between one another. For example, as shown in FIG. **1**, the first executing program **106***a* may transmit data packets to directly to the second executing program **106***b*.

FIG. **2**B illustrates a first stage handshake data packet **202***b* of a stage automation system **100**, in accordance with one or more embodiments of the present disclosure. The first stage handshake data packet **202***b* may include, but is not limited to, a short header **204**, a long header **208**, and a data payload **206***b*.

Once two components have received one another's public RSA keys, the two components may exchange first stage handshake data packets **202***b*. As noted previously herein, the short header **204** may be transmitted unencrypted, and repeated fully in the encrypted portion of the data packet. In embodiments, the data payload **206***b* may include an AES symmetric key of the sending component. In this regard, exchanging first stage handshake data packets **202***b* may allow various components to securely handshake and exchange AES symmetric keys with one another.

FIG. **2**C illustrates a second stage handshake data packet **202***c* of a stage automation system **100**, in accordance with one or more embodiments of the present disclosure. The second stage handshake data packet **202***c* may include, but is not limited to, a short header **204**, a long header **208**, and a data payload **206***c*.

Once two components have received one another's AES symmetric keys, the two instances may transmit a series of second stage handshake data packets **202***c*. Second stage handshake data packets **202***c* may be transmitted between components in order to judge network latency and jitter between the respective components. In embodiments, the long header **208** may include timing information (e.g., timecode or runtime configuration of sending component), and may be included within all subsequent data packets in order to continually monitor network latency and other conditions/characteristics of the network protocol. After AES symmetric keys have been exchanged and a communicative coupling has been established between components of the stage automation system **100**, data payloads **206***c* exchanged between the components may be trusted and acted upon.

While FIGS. **2**A-**2**C are shown and described in the context of RSA/AES encryption algorithms, these are not to be regarded as a limitation of the present disclosure, unless noted otherwise herein. In this regard, the stage automation system **100** may be configured to utilize any encryption algorithms or schemes known in the art without departing from the spirit and scope of the present disclosure.

In embodiments, a component may be configured to transmit one or more update data packets (e.g., "XaValuesUpdate" data packets) when a value of one or more time-stamped variables associated with the component is adjusted or updated. For example, if a position of the actionable mechanism **104***a* is updated, the first executing program **106***a* may transmit one or more update data packets to the stage automation server **108**. The stage automation server **108** may then be configured to forward the one or more update packets to one or more additional executing programs **106***b*-**106***n* which subscribe/depend upon the updated value. A database of components (e.g., executing programs **106**) which subscribe/depend on the updated value may be stored in memory **112**. In this example, the one or more executing programs **106** may then update distributed program objects associated with the updated time-stamped variable based on the one or more received update packets. Example update data packets which may be transmitted throughout the stage automation system **100** via the network protocol is shown in Table 4 and Table 5 below.

TABLE 4

Update Data Packet Structure

Starting Byte	Bytes	Value Type	Content
0	16	UUID	Time-stamped variable UUID for example Value 1
16	1	Byte	Value type for Value 1. This is redundant, since the receiving executing program should know what value type that time-stamped variable UUID is, but this additionally helps reduce speed of decoding and prevents accidental buffer overruns that would cause the decoding process to miss other valid data. For this example, we'll assume the value type of Long, which is a signed integer that is stored with 4 bytes per value
17	2	UInt16	Number of time-stamped variable slices that are included after this (e.g., 3 slices)
19	8	Double	Slice 1: Timecode
27	8	Long	Slice 1: Value
35	8	Double	Slice 2: Timecode
43	8	Long	Slice 2: Value
51	8	Double	Slice 3: Timecode
59	8	Long	Slice 3: Value
67	16	UUID	Value type UUID for example Value 2
83	1	Byte	Value type for Value 2
84	2	UInt16	Number of time-stamped variable slices that are included after this (e.g., 3 slices)
86	8	Double	Slice 1: Timecode
94	8	Long	Slice 1: Value
102	8	Double	Slice 2: Timecode
110	8	Long	Slice 2: Value
118	8	Double	Slice 3: Timecode
126	8	Long	Slice 3: Value

TABLE 5

Update Data Packet Structure

Starting Byte	Bytes	Value Type	Content
0	16	UUID	Time-stamped variable UUID for example Value 1
16	1	Byte	Value Type (e.g., String)
17	2	UInt16	Number of time-stamped variable slices that are included after this (e.g., 3 slices)
19	8	Double	Slice 1: Timecode
27	2	UInt16	Slice 1: Value
29	17	Byte	Slice 2: Timecode
46	8	Double	Slice 2: Value
54	2	UInt16	Slice 3: Timecode
56	635	Byte	Slice 3: Value
691	16	UUID	Time-stamped variable UUID for example Value 1
707	1	Byte	Value Type (e.g., Bool)
708	2	UInt16	Number of time-stamped variable slices that are included after this (e.g., 3 slices)
710	8	Double	Slice 1: Timecode
718	1	Bool	Slice 1: Value
719	8	Double	Slice 2: Timecode
727	1	Bool	Slice 2: Value
728	8	Double	Slice 3: Timecode
736	1	Bool	Slice 3: Value

In particular, Table 4 illustrates an update data packet storing multiple time-stamped variable slices (e.g., multiple time-stamped variables for various points in time) for two time-stamped variables including long integer value types, and Table 5 illustrates an update data packet storing multiple time-stamped variable slices for two time-stamped variables including a string value type and a bool value type.

It is noted herein that transmitting update data packets throughout the stage automation system **100** upon updating a time-stamped variable may allow each component of the system to maintain up-to-date information regarding other components, such as a location, velocity, operational status, and the like. It is further noted herein that the size of update data packets may be limited to the maximum transmission unit (MTU) of the network, less the known length of the short header **204** used throughout the stage automation system **100**. Once a byte count of the data payload of an update data packet (or other data packet) reaches a maximum length, the respective data packet may be transmitted, and a new data packet including remaining bytes to be transmitted may be constructed.

FIGS. **3**A-**3**C illustrate a conceptual diagram of a time-stamped variable with a publish variable mode and a time-stamped variable with an omni-publish variable mode, in accordance with one or more embodiments of the present disclosure. In particular, FIGS. **3**A-**3**C illustrate a first component (Component **1**) subscribing to a first time-stamped variable of a second component (Component **2**), and updating a second time-stamped variable of the second component (Component **2**).

For example, as shown in FIG. **3**A, a second component (Component **2**) (e.g., actionable mechanism **104***d*, executing program **106***b*) may transmit a distributed program object announcement to the stage automation server **108** including a first time-stamped variable with a publish variable mode and a second time-stamped variable with an omni-publish variable mode. The stage automation server **108** may receive the distributed program announcement, and generate/store the distributed program announcement of Component **2** in memory. The distributed program object announcement may only include the names of the time-stamped variables, variable modes, and value types. Subsequently, Component **2** may transmit update data packets (e.g., updated time-stamped variables) which include actual values for each time-stamped variable. As shown in FIG. **3**B, a first component (Component **1**) (e.g., user interface **114**, actionable mechanism **104***d*, executing program **106***b*) may subscribe to the distributed program object of Component **2** such that it may view the values of the time-stamped variables. Subsequently, Component **2** may update the value of the second time-stamped variable with an omni-publish variable type, and transmit update data packets to Component **2**, thereby adjusting one or more characteristics of Component **2**. Conversely, in FIG. **3**A, Component **2** may update the first time-stamped variable, and update data packets may be transmitted to Component **1** reflecting the change.

In embodiments, one or more executing programs **106***b*-**106***n* may receive one or more data packets from the second executing program **106***b*. In additional embodiments, one or more executing programs **106***a*-**106***n* may be configured to adjust one or more time-stamped variables with an omni-publish value type. For example, the first executing program **106***a* may include a motor as a first actionable mechanism **104***a*. The motor may include a distributed program object including a time-stamped variable indicative of an output of the motor, and the motor may be characterized by an omni-publish value type such that other components of the system may adjust/update the motor output. In this example, a second executing program **106***b* may subscribe to and/or receive the distributed program object of the motor (e.g., first actionable mechanism **104***a*), adjust the time-stamped variable indicative of motor output to a new output, and transmit one or more data packets including the updated distributed program object and/or updated time stamped variable to the first executing program **106**. The one or more data packets may be configured to cause the first executing program **106***a* to adjust the time-stamped variable of the motor (e.g., first actionable mechanism **104***a*) of the motor's distributed program object in response to the received data packets. In this regard, the first executing program **106***a* is configured to adjust one or more operational characteristics (e.g., motor output) of the first actionable mechanism **104***a* associated with the at least one adjusted time-stamped variable.

Continuing with the same example above, upon receiving the one or more data packets including the updated distributed program object and/or updated time stamped variable, the stage automation server **108** may also be configured to update the time-stamped variable of the motor stored in memory **112** based on the received data packets which may be forwarded to the first executing program **106***a*.

It is noted herein that selectively adjusting and controlling characteristics of components within the stage automation system **100** via distributed program objects may enable much greater flexibility and robustness than is currently possible using conventional systems.

An example may prove to be illustrative. consider a conventional stage automation system which includes a first microwave from Manufacturer A. In order to start the first microwave, a user must select the "Cook" button, then select the "Time Cook" button, enter a time, then press and hold a "Start" button for two seconds. The conventional stage automation system may further include a second microwave from Manufacturer B. In order to start the second microwave, a user must enter a time, select the "Cook" button, then press and release a "Start" button. In order to remotely control the first and second microwaves via a server/controller, the server/controller would have to be programmed specifically for Manufacturer A and Manufacturer B separately such that it performs the right steps, in the right order, in the right format, and the like. This specific programming would be individualized on a manufacturer and product basis. Such bespoke programming may be tedious and time consuming, which leads to a lack of compatibility and communication between components of the conventional stage automation system.

Comparatively, consider the same example with the first and second microwaves in the context of the stage automation system **100**. Upon introduction into the environment of the stage automation system **100**, each microwave may generate and transmit a distributed program object announcement. The distributed program object of each microwave may include a time-stamped variable with value type string indicating the type/name of the microwave (e.g., "CookMeister500Microwave," "GEJES1072SHMicrowave") which will be unique to each microwave. Additionally, the distributed program object of each microwave may allow other components update the cook timer (e.g., time-stamped variable, double, publish), may publish the operational state (on/off) of the respective microwave (e.g., time-stamped variable, boolean, publish), publish the internal temperature (e.g., time-stamped variable, double, publish), and allow other components to turn on the microwave (e.g., time-stamped variable, boolean, omni-publish). Upon receiving the distributed program object of the respective microwaves, interested/dependent components may save/generate the distributed program object which will listen/view to the temperature of the microwaves, and listen/view in order to update/adjust the cook timer and on/off status. For instance, in order to turn on either microwave, all another component would have to do is update the time-stamped variables associated with the cook timer and on/off status and transmit update data packets including an updated distributed program object to the respective microwave. Each respective microwave may then be configured to receive the updated distributed program objects, and adjectively alter the cook time and on/off status of the microwave based on the updated distributed program object and the manufacturer

As illustrated in the example above, controlling/adjusting characteristics of components within the stage automation system **100** via distributed program objects may enable much greater flexibility. In particular, distributed program objects may render manufacturer-specific processes and instructions to be generic for all components/machines of a particular type. For instance, updating an on/off status via a distributed program object may be similar for all microwaves, no matter the manufacturer or manufacturer-specific program instructions. In other words, the stage automation system **100** may utilize generic updates to distributed program objects to implement manufacturer-specific programming. Controlling a component within the stage automation system **100** therefore focuses on the type of the component (as defined by its distributed program object), and not the manufacturer or individual machinery associated with the component. In this regard, embodiments of the present disclosure may enable a stage assembly system **100** which is hardware and protocol agnostic.

By way of another example, the processes used to release the brake and trigger a velocity change in a Nidec M750 servo drive is very different from that of a Kollmorgen AKD-BASIC servo drive. Distributed program objects would include names (value type string) indicating the different servo drives. However, the distributed program objects of both servo drives would include similar and/or identical time-stamped variables (e.g., on/off status, voltage, current, output), making the distributed program objects of each specific servo drive appear to be a generic distributed program object for any servo drive (minus the name within each distributed program object)

It is further noted herein that allowing components of the stage automation system **100** to selectively adjust characteristics of other components within the environment of the stage automation system **100** may enable increasingly automated events and performances (e.g., concerts, plays, and the like). For example, during a concert, three separate actionable mechanisms **104***a*-**104***c* may be required to perform their designated actions before a fourth actionable mechanism **104***d* is to be activated to perform its designated actions. In this example, upon performing its designated actions (e.g., actuating to correct location, producing light, etc.), the first actionable mechanism **104***a* may update a distributed program object of the second actionable mechanism **104***b*, causing the second actionable mechanism **104***b* to perform its designated actions. Upon performing its designated actions (e.g., actuating to correct location, producing light, etc.), the second actionable mechanism **104***b* may update distributed program objects of the third and fourth actionable mechanisms **104***c*-**104***d*, causing the third and fourth actionable mechanisms **104***c*-**104***d* to perform their designated actions.

In embodiments, updates to time-stamped variables and/or distributed program objects are written in real-world units, such that the position, velocity, acceleration, and the like, of each component of the stage automation system **100** is known with respect to 3D space. As noted previously, positional information (e.g., position, velocity, acceleration, deceleration, position clamps, and the like) may be selected and stored within the system with respect to a defined reference point.

In some embodiments, actions performed within the stage automation system **100** may be limited or controlled by "conditions." The term conditions may be used to refer to logic statements which limit actions and/or cause certain actions to occur within an environment. For example, a condition may state that if a certain time-stamped variable or statement of time-stamped variable comparisons is true, limit the velocity of actionable assembly **102***a* to 1 m/s. By way of another example, a condition may be configured to fire a command (e.g., update a time-stamped variable to initiate an action) on the rising edge of a certain time-stamped variable. In embodiments, a database of conditions associated with an environment may be stored in memory **112** of the stage automation server **108** and/or memory of components within the stage automation system **100**.

In some embodiments, a show or event (e.g., concert, play, sporting event) may be programmed in terms of cue sheets, cues, and commands. A cue sheet may be stored in memory **112**, wherein the cue sheet includes a list of one or more cues and/or commands. A command may be used to refer to an action which an actionable mechanism **104** and/or actionable assembly **102** may take. For example, commands may include "move to x position," "sync position with another actionable mechanism **104** and/or actionable assembly **102**," move to x velocity and hold x velocity," move along a predefined series of positions or velocities," "follow the position value passed via Open Sound Control (OSC) or Art-Net," "wait x seconds, then move to y position," and the like. A cue may be used to refer to multiple actions which are executed on one or more actionable mechanisms **104** and/or actionable assemblies **102**. Thus, a cue may refer to a group of one or more commands.

In practice, a memory **112** of the stage automation server **108** may be configured to store a cue sheet including a plurality of cues and/or commands, often the cues/commands for an entire show. For example, the stage automation server **108** may be configured to store a cue sheet in memory, wherein the cue sheet includes a first command executable by the first actionable mechanism **104***a*, and a second command executable by the second actionable mechanism **104***b*. Cues and commands may be linked to the particular component (e.g., actionable assembly **102**, actionable mechanism **104**, executing program **106**) configured to execute them based on the UUID of the respective component.

For instance, the stage automation server **108** may be configured to transmit one or more data packets associated with a first command to the first executing program **106***a*, wherein the first executing program **106***a* is configured to adjust at least one time-stamped variable of a first actuatable mechanism **104***a* in order to execute the first command. The stage automation server **108** may be further configured to transmit one or more data packets associated with the second command to the second executing program **106***b*, wherein the second executing program **106***b* is configured to adjust at least one time-stamped variable of the second actuatable mechanism **104***d* in order to execute the second command.

The stage automation server **108** may include a "CueManager." The CueManager may include a module or a set of program instructions executable by the one or more processors **110**. The CueManager may be configured to hold commands which are currently running in a position denoted "CommandExecutor," and commands which are to be run subsequently in a position denoted "CommandStandby." To initiate a cue or command, the CueManager may receive a UUID of an existing cue or command. The CueManager may then make a copy of the cue/command, and put its contents into CommandStandby position. For each executing program **106** in the environment, the CueManager may be configured to take the first cue/command in the CommandStandby position and move it to the CommandExecutor position in order to initiate/execute the cue/command on its intended executing program. Once the cue/command in the CommandExecutor position completes, the CueManager may remove it from the CommandExecutor position, and move the next cue/command (if any) from the CommandStandby position in to the CueExecutor position.

In embodiments, some cues/commands may keep a component (e.g., executing program **106**, actionable mechanism **104**, actionable assembly **102**) active until certain criteria or conditions are met. For example, a command may keep a component active until a timer has expired, a position of another component is met, or a particular time-stamped variable reaches a particular value. In this regard, the CueManager (e.g., stage automation server **108**) may not deactivate a command in the CommandExecutor position until the command is complete. In other embodiments, some components may be put into a fixture mode to be controlled by a lighting console or other external control. For example, some components may be put into a fixture mode such that they are controlled via the user interface **114**. Commands in a fixture mode may stay active until they are manually cleared. The stage automation server **108** of the environment may listen to the Art-Net/Streaming Architecture of Control Networks (sACN) universe/multiverse for transporting frames in the DMX universe/multiverse, and can make executing programs **106** respond to live commands to chase a position command from Art-Net. Each executing program **106** may include a starting address in the DMX multiverse and the OSC namespace.

As shown in FIG. **1**, the stage automation system **100** may further include a user interface **114**. The user interface **114** may include a user input device **1167** and a display device **118**. The user interface **114** may include any user interface device known in the art including, but not limited to, a desktop computer, a laptop, a mobile device (e.g., smartphone, tablet), a wearable device, and the like. The display device **118** may be configured to display data/information of the stage automation system **100** to a user. In this regard, a user may be able to view characteristics of components within the stage automation system **100** (e.g., distributed program objects, time-stamped variables for position, velocity, operational state, and the like) via the user interface **114**.

In embodiments, the user input device **116** may be configured to receive one or more input commands from a user responsive to data shown on the display device **118**. For example, a user may be able to input various commands, conditions, and the like, via the user interface **114**. By way of another example, a user may be able to run components (e.g., actionable assemblies **102**, actionable mechanisms **104**) of the stage automation system **100** in a "jog mode." The term "jog" may be used to refer to user-controlled actions (e.g., user-input position, user-input velocity, user-input command to move forward or backwards). For example, an executing program **106** may be run in a jog mode, live-chasing either a position, velocity, or torque target with minimum/maximum position clamps, velocity clamps, and acceleration clamps set by settings of the executing program **106**, parameters associated with a command, or a defined condition. Actions of a particular component may be limited to whichever parameters (e.g., executing program **106** settings, parameters associated with a command, a defined condition) are most restrictive.

For instance, the stage automation server **108** may be configured to receive one or more input commands (e.g., data packets) from a user interface **114**, and adjust at least one time-stamped variables of a set of one or more time-stamped variables of a distributed program object stored in memory **112** based on the one or more input commands received from the user interface **114**. Subsequently, the stage automation server **108** may be configured to transmit one or more update data packets to the component associated with the updated distributed program object in order to implement the action, command, or condition based on the received input commands.

In some embodiments, the stage automation system **100** may implement the DMX multiverse which receives data in and transmits data out to a DMX, Art-Net, and/or RDM network. The DMX multiverse may also contain configuration linking UUIDs of various executing programs **106** to starting addresses and fixture styles in the multiverse, a configuration that is static and persistent in the configuration of the environment. In embodiments, an executing program **106** of an environment of the stage automation system **100** may be configured to listen to a DMX512 source (e.g., RS-485, DMX512, Art-Net, DMX over sACN), and transmit the universe values as data packets over the network protocol to the stage automation server **108**. The universe values may be timestamped with the time of capture (similar to time-stamped variables), and marked with the UUID of the physical device (e.g., RS-485, DMX512) or the Art-Net universe the values originated from. The multiverse object will by default consume any incoming DMX data from any executing program **106** that has a DMX source configured on it, and will broadcast out any DMX out from the multiverse any DMX output configured in the environment, either by DMX512 or a configured Art-Net/sACN network endpoint.

In embodiments, the stage automation system **100** may be configured to support OSC in and out of the environment of the stage automation system **100**. By default, the fixture mode for OSC listens to a predefined address prefix for each actionable assembly **102**/actionable mechanism **104**, and transmits status on a predefined address prefix. The address prefix can be the UUID of the actionable assembly **102**/actionable mechanism **104** or another name. In additional embodiments, the stage automation system **100** may be configured to support PosiStageNet input and output. By default, the system will consume any PSN data on the network and create and update single-point geometry object for each unique tracker fount. It will also publish current point data for all actionable assemblies **102**/actionable mechanisms **104** with known 3D positions to the network protocol (e.g., to the stage automation server **108**).

In embodiments, a fixture move command may instruct an actionable assembly **102**/actionable mechanism **104** to make its best effort to hit a requested position target channel, within set clamps and limits in addition to the ones set in the DMX fixture. Each actionable assembly **102**/actionable mechanism **104** has the option to configured a starting DMX multiverse and address, and whether it functions in a "Simplified" or "Complete" mode. This may be further understood with reference to Table 6 and Table 7. Table 6 illustrates a simplified one-dimensional (1D) position jog fixture profile, and Table 7 illustrates a complete one-dimensional (1D) position jog fixture profile.

TABLE 6

Simplified 1D Position Jog Fixture Profile

Channel	Purpose	Value	Note
1	Enable	255 = enable; 0 = disable	Permits motion for component. Fixture Move Command may optionally activate or deactivate the component depending on component type and setting.
2	16-bit position target course	Byte	Combined 16-bit value range is normalized between −32,768 and 32,767, representing mm for linear axes, and tenths of degrees for rotational axes.
3	16-bit position target fine	UInt16	
4	8-bit velocity clamp	Double	Value range between 0 and 255 representing cm/sec for linear axes, and degrees/sec for rotational axes. Applies to positive and negative velocity.
5	8-bit acceleration clamp	Long	Value range between 0 and 255 representing cm/sec for linear axes, and degrees/sec for rotational axes. Applies to positive and negative velocity.
6	8-bit deceleration clamp	Double	

TABLE 6-continued

Simplified 1D Position Jog Fixture Profile			
Channel	Purpose	Value	Note
7	Corruption Check 1	Long	Evaluation of binary 10101011; component will fault if not set.
8	Corruption Check 2	Double	Evaluation of binary 01010100; component will fault if not set.

TABLE 7

Complete 1D Position Jog Fixture Profile			
Channel	Purpose	Value	Note
1	Enable	255 = enable; 0 = disable	Permits motion for component. In this fixture mode, servo & brake activation will follow the jog mode selection.
2	Jog mode select	0 = none; 24 = position; 36 = velocity	None is functionally equivalent to a deactivate request. Position will have the component chase the position target, and velocity will have the component chase the velocity target.
3	16-bit position target course		Combined 16-bit value range is normalized between −32,786 and 32,767, representing mm for linear axes, and tenths of degrees for rotational axes.
4	16-bit position target fine		
5	16-bit velocity target course		Value range between 0 and 255 representing cm/sec for linear axes, and degrees/sec for rotational axes. Applies to positive and negative velocity.
6	16-bit velocity target fine		
7	16-bit velocity clamp coarse		Value range between 0 and 65,534 representing mm/sec for linear axes, and degrees/sec for rotational axes. Applies to positive and negative velocity.
8	16-bit velocity clamp fine		
9	16-bit acceleration clamp coarse		Value range between 0 and 65,534 representing mm/sec^2 for linear axes, and tenths of degrees/sec^2 for rotational axes.
10	16-bit acceleration clamp fine		
11	16-bit deceleration clamp coarse		
12	16-bit deceleration clamp fine		
13	16-bit position limit maximum coarse		Combined 16-bit value range is normalized between −32,786 and 32,767, representing mm for linear axes, and tenths of degrees for rotational axes.
14	16-bit position limit maximum coarse		
15	16-bit position limit minimum coarse		
16	16-bit position limit minimum coarse		
17	Corruption Check 1	171	Evaluation of binary 10101011; component will fault if not set.
18	Corruption Check 2	84	Evaluation of binary 01010100; component will fault if not set.

FIG. **4** illustrates a graph **400** depicting position and velocity limits of a component executing commands, in accordance with one or more embodiments of the present disclosure. As noted previously herein, commands and actions implemented by various components (e.g., actionable mechanism **104**, actionable assembly **102**) may be constrained by various parameters, including condition statements, position clamps, velocity clamps, acceleration/deceleration clamps, and the like.

For example, graph **400** is a visual representation of position of an actionable assembly **102** over time. The actionable assembly **102** may start at an initial start position **301** at an initial time (t_0), and end at an end position **311** at a final time (t_f). The movement of the actionable assembly **102** may be constrained by a maximum position clamp **403***a* and a minimum position clamp **403***b* such that the actionable assembly **102** may not move outside of the respective position clamps **403***a*, **403***b*. Additionally, the movement of the actionable assembly **102** may be constrained by a velocity clamps **413** indicated by the slanted/diagonal lines. In embodiments, the graph **400** may be an example depiction of the position of the actionable assembly **102** displayed to a user via the display device **118**.

A cue/command for the actionable assembly **102** may include a set of position targets **402***a*-**402***n* which the actionable assembly **102** is to hit throughout execution of the cue/command. As shown in FIG. **4**, if one or more command positions **402***d*, **402***e* are outside of the range of values determined by the position clamps **403***a*, **403***b*, the actionable assembly **102** may move until the position clamp limit (e.g., minimum position clamp **403***b*, clamped position target **407**), then continue in sync with the intended command positions once the command positions return to the range of values determined by the position clamps **403***a*, **403***b*. [ono] In embodiments, the stage automation system **100** may include a 3D geometry system which functions similarly to Cinema 4D/Unity style hierarchies, with parent/child relationships, and additional attributes to an object that can link it to other objects' influence. All components within an environment may have an associated geometry tree. In its simplest configuration, this tree has an origin position & orientation, and a childed object for static geometry, a null object driven by the axis' positionMeasured, and a childed object with the driven geometry. For example, a stage elevator (e.g., actionable assembly **102**) may include support structure that would not move, and the orientation for the driven object would be at the top pointing down. Negative position values on the component move the physical platform down to the ground; when the component moves, the 3D object also moves to mimic that. The component also automatically creates representative geometry to show the physical position limits configured on the component. The PSN object has its own associated geometry tree, and it creates & updates child objects for each "show" ID and object ID it receives network data for. Each component (e.g., actionable mechanism **104**, actionable assembly **102**) by default publishes its origin and driven position to the PSN network, and other geometry objects in the environment hierarchy can also be tagged to send their position out to the PSN network. The stage automation server **108** may contain 3D models for each make/model of component, and models can be imported to attach to custom components.

FIGS. **7**A-**7**E illustrate a three-dimensional (3D) tracking system of a stage automation system, in accordance with one or more embodiments of the present disclosure.

It is noted herein that measurement of an environment and understanding of the position and orientation of machinery/components (e.g., actionable mechanism **104**, actionable assembly **102**) in a 3D space is critical to being able to program complex motion and ensure safe operation around other elements and people. In embodiments, the 3D tracking system of the stage automation system **100** may include a plurality of anchor constellations **700** to automatically configure the RF tracking system, as shown in FIG. **7**A. In embodiments, a single anchor constellation **700** may include a plurality of transponders **702***a*-**702***d*. GUIDs and/or UUIDs for the anchor constellation **700** and each transponder **702***a*-**702***n* may be stored in memory **112**. Each anchor constellation **700** and transponder **702***a*-**702***d* may additionally include a distributed program object which may be transmitted throughout the system. In additional and/or alternative embodiments, an anchor constellation **700** including a plurality of transponders **702** may include a single distributed program object such that the anchor constellation **700** is regarded as one single component.

In embodiments, the plurality of transponders **702***a*-**702***n* may be arranged on a stiff, static frame **701** (e.g., 3D pyramid). Locations of each the anchor constellation **700** on an event stage may be known, and the heights and distances between each of the transponders may also be known and stored in memory **112**. For example, as shown in FIGS. **7**B and **7**C, a location/position of each anchor constellation **700***a*-**700***d* of a plurality of anchor constellations **700***a*-**700***d* may be stored in memory **112**. Similarly, the height of each transponder **702** and distances between each transponder **702** may be stored in memory **112**. As shown in FIGS. **7**B-**7**C, an environment may include multiple anchor constellations **700**, multiple actuators (e.g., multiple actionable mechanisms **104***a*-**104***d*), and an object moved by the actuators (e.g., chain motor actuatable mechanisms **102***a*-**102***d*). Trackers **704** may be attached to actuators (e.g., actionable mechanisms **104**) and the objects to be moved (e.g., actionable assemblies **102**), as shown in FIGS. **7**D-**7**E. A database associating each components UUID to the serial number(s) of the associated trackers may be stored in memory **112**. Additionally, the geometry of the actuator in relationship to the trackers may also be stored in memory **112**.

By associating UUIDs with the respective components, and associating commands/cues with the UUIDs, the stage automation system **100** may be configured to link particular cues/commands to the associated components to which they are intended. For example, actuators in FIGS. **7**C-**7**D can be identified and operated by their UUID. In this example, an optical scanner on a control device can be used to read a hoist's UUID encoded in a 2D barcode like a QR code, to select it for immediate operation. Physical actuators may also be identified by a unique, persistent UUID that allows the stage automation server **108** to look up what it is, how to configure the drive it's attached to, and how to display it on the user interface **114**.

In embodiments, by transmitting signals between the respective trackers **704** and the transponders **702** of each anchor constellation **700**, the 3D position of each component of the stage automation system **100** may be triangulated. 3D positional information may be determined using any mathematical formula, algorithm, or technique known in the art including, but not limited to, time-of-flight determinations, inertial measurement units (IMU), and barometric sensors, received signal strength indicator (RSSI) values. For instance, the stage automation system **100** may be configured to utilize RSSI values to reject potentially bad data packets from nodes which may be obstructed, or for signals which have reflected off other surfaces.

In embodiments, movement cues/commands for the sphere (e.g., actionable assembly **102** in FIGS. **7**B-**7**C) may be written in relation to the environment's origin. With knowledge of where all of the hoists and their hooks are, the stage automation server **108** may be configured to selectively move/actuate the suspended sphere anywhere within the area between the hoists. The operator doesn't need to individually program the motion of each hoist, just program the motion the sphere itself. In embodiments, actionable mechanisms **104** and/or actionable assemblies **102** may be configured to receive and/or execute commands in 1D and/or 3D space, dependent upon the physical attributes of the respective actionable mechanisms **104** and/or actionable assemblies **102**. Actionable mechanisms **104** and/or actionable assemblies **102** may continuously identify their physical position by transmitting RF positioning feedback to the stage automation server **108** and/or by measurements entered by an operator via user interface **114**.

For example, as shown in FIG. **7**B, an actionable assembly **102** may be physically actuated (e.g., moved) by four separate actionable mechanisms **104***a*-**104***d* (e.g., hoists **104***a*-**104***d*). In this example, commands to move the actionable assembly **102** may be written in the context of 3D space that move the actionable assembly **102** itself, and the stage automation server **108** may transmit jog commands to each respective hoist **104***a*-**104***d* to achieve the 3D command of the actionable assembly **102**. For instance, a user may input a new position for the actionable assembly **102** via the user interface **114**. The stage automation server **108** may then calculate the required hoist actuation to achieve the new position indicated by the 3D command, and transmit 3D commands (e.g., jog commands) to each of the hoists **104***a*-**10***d*. Upon receiving the 3D commands (e.g., jog commands) for the new position of the actionable assembly **102**, each hoist **104***a*-**104***d* may jog the hoist to the determined position. Such calculations may be carried out many times a second, jogging the position of each hoist **104***a*-**104***d* to achieved desired hoist positions required for the 3D positioning of the actionable assembly **102**. In additional and/or alternative embodiments, the calculations required to achieve the positions associated with the user commands may be carried out by processors of the hoists **104***a*-**104***d* themselves. By writing cues/commands in the context of distributed program objects, the stage assembly system **100** may be configured to command motion of the hoists to match the length that it simulates in 3D space.

It is noted herein grouping a plurality of actionable mechanisms **104** together to move one or more actionable assemblies **102** (e.g., FIG. **7**B) may provide a number of benefits. For example, as shown in FIG. **7**B, if an operator chooses to move a single actuatable mechanism **104***a* (hoist **104***a*) individually instead of as part of the group, the chains of the hoist **104***a* may slack, resulting in the other three hoists **104***b*-**104***d* now holding the majority of the weight of the object. In this example, if the operator then sends a 3D move command to the actionable assembly **102**, the stage automation server **108** may be configured to first bring all the actionable mechanisms **104** (e.g., hoists **104***a*-**104***d*) back into a stable position that it expects it to be in (e.g., remove the slack from the hoist **104***a*), then continue with the move.

It is further herein that writing commands in the context of the actionable assembly **102** being moved, rather than the actionable mechaims **104** used, can be applied in other contexts. For example, a gantry track (e.g., actionable mechanism **104**) and a plurality of hoists (e.g., actionable mechanisms **104**) may be used to operate cooperatively in order to selectively actuate/move an actionable assembly **102**. In this example, commands may be written for the actionable assembly **102** as distributed program objects such that the cues and commands for the actionable assembly **102** are written for the actionable could be written for the actionable assembly **102** itself, instead of the gantry track and/or hoists individually.

It is noted herein that accurate positional measurements by the operator or automatic measurements by a 3D tracking system may be used for several purposes including, but not limited to: accurate visualization of the 3D environment to a user via the user interface **114**; cues can be written to move objects instead of individual actuators; interfaces can show contextual information on objects or devices they're in proximity to or aimed at; augmented reality displays can overlay contextual information on real objects or actuators; actuators can automatically move, level, or tilt objects to predetermined positions without manual human control; objects and actuators can programmatically respond to motion of actors or other non-actuated props; actuators can be automatically assigned to the system via either known proximity of their end effectors; the system can automatically set static or dynamic limits based on known positions of other objects or people.

In another embodiment, physics solver operations may be used to accurately simulate how rigged axes may sway with shifting centers of gravity, compensate for shifts, predict if objects may collide, and transition between control of individual machines and a combined virtual axis representing a 3D object. This is possible through use of a predictive physics engine, which may run frame calculations to look ahead over a given time interval (e.g. a number of seconds in actual time) to predict how 3D objects will move and/or sway based on various inputs. This physics engine is used at several different levels within the system, depending on the need or the complexity of the 3D object itself.

Stage automation systems may work exclusively in either Object-based cueing, or Machine-based cueing. In Machine-based cueing, an operator may manually control individual machines (e.g. lifts, hoists, and the like) to move to one or more desired positions, then record the positions of the machines as a cue in a control system. Such cue automation does not necessarily require any surveying or 3D modeling, and may be used for programming movements of simple objects (e.g. straight trusses), or inherently stable objects (e.g. triangles with three rigging points). However, such a system may have no concept of how individual machines are related, no knowledge that they are connected, and no way to monitor how a rigged object is moving.

Machine-based cueing may be difficult to use with rigged objects that are not inherently stable. For instance, with a square object suspended by four points, if a hoist lifts too far, it could overload itself and slack two of the other lifting lines. Additionally, a Machine-based cueing system may not understand the actual position and orientation of lifted objects, which may be critical for use with modern multi-axis lighting and projection systems. Machine-based automation cueing may require a secondary optical or RF tracking system to be able to interact with other show control systems.

In Object-based cueing, an operator may build a 3D model of a system in software, designating, for example, virtual representations of a mother grid (e.g. a primary support framework to which one or more lifting machines are coupled in order to suspend a 3D object) having rigged points which are attached to a 3D object in 3D space. Then, the operator can move the virtual object in a virtual space and the system may calculate the virtual distance between the various lifting machine virtual attachment points, and program the actual machines to match those movements.

The detected position of each of the actual machines (e.g. the value the encoders are reporting) should match the automation system's virtual position for each of the machines. As such, if an individual machine is moved independently of a supported 3D object (e.g. by error or for utility purposes), the Object-based automation system may need to move all the machines back to match a designated virtual model state before it can move the object again. It will be noted that prior object-based cueing systems may not be able to simulate how actual 3D objects may shift their center of gravity or how an object would move if a single machine was moved, independent of moving the entire 3D object.

In one embodiment, for example, as shown in FIG. **9**A, a mother grid **901** may include a first hoist **902**A and a second hoist **902**B from which a flown object **903** (e.g. a lighting truss, video board, staging component, and the like, which may be "flown" (i.e., lifted/translated by one or more hoists/lifts) may be suspended via a first support line **904**A and a second support line **904**B, respectively. A Machine-based system may implement a first que corresponding to a first position **900**A of the flown object **903** (1 m above a reference floor surface **906**) and a second que corresponding to a second position **900**B where flown object **903** has been rotated. Specifically, an operator may create a first cue with the support line **904**A of the first hoist **902**A and the support line **904**B of the second hoist **902**B at an extension of 3 meters, respectively. A second cue may be configured with the support line **904**A of the first hoist **902**A at an extension of 2 meters and the support line **904**B of the second hoist **902**B remaining at 3 meters.

Referring to FIG. **9**B, in an Object-based system, the flown object **903** may be characterized by the virtual position of a center-of-gravity (COG) reference point **905** and a degree of rotation of the flown object **903** about the reference point **905**. For example, to replicate the Machine-based cues of FIG. **9**A, in an Object-based system, a first que may be created with the reference point **905** at a Y=1 m (relative to a reference floor surface **906**) position and a rotation about the reference point **905** of Z=0°. A second cue may be created where the reference point **905** moves to a Y=1.5 m (relative to the reference floor surface **906**) position and a rotation about the reference point **905** of Z=12°.

The Object-based system may then calculate (e.g. via interpolation) that support line **904**A needs to be retracted to ~2 m by the first hoist **902**A and support line **904**B needs to remain at 3 m in order to tilt the flown object **903** 12° at a Y position of 1.5 m. However, if an operator manually controlled the first hoist **902**A to retract the support line **904**A to 1 m, an Object-based system may be unable able to determine what current Y position and Z rotation of the flown object **903** are, as it was not explicitly moved as an object by the system. As such, in order for it to move the flown object **903** based on position and rotation input again, the system it would need to move support line **904**A back down to its prior 2 m position, before it can move the flown object **903** again according to the previously established virtual cues.

The present stage automation system **100** allows for the movement of 3D objects both in Machine-based cueing mode, or Object-based cueing mode, and seamlessly transition between the two.

For example, as shown in FIG. **9**C, the flown object **903** can be moved from the first position **900**A to the second position **900**B in Object mode where stage automation system **100** virtually moves the flown object **903** in a simulated 3D space and calculates that support line **904**A needs to be at 2 m and support line **904**B needs to be at 3 m to achieve this position, and controls the first hoist **902**A and the second hoist **902**B accordingly. Subsequently, non-Object-based control (e.g. manual operator control) may move the flown object **903** to a third position **900**C where the support line **904**A is retracted to 1 m (equating to Y=2 m and Z=21° in Object mode). As the flown object **903** moves, the stage automation system **100** may monitor and record how the flown object **903** has rotated and/or translated under such manual control to compute its current rotational and position state. With the real-time data of the third position **900**C, the operator can re-enter Object mode to translate from the third position **900**C to another position in Object mode, without having to return support line **904**A to 2 m (as in second position **900**B) to continue programmed Object-based movement of the flown object **903**.

While movement of the flown object **903** shown in FIGS. **9**A-**9**C is shown in two dimensions, it will be understood that the described methodologies may be extended to three-dimensional implementations as well.

Further, referring to FIG. **9**D, if the stage automation system **100** detects that the rigging for the flown object **903** is in an unstable state (e.g. a support line has slack in it) when the stage automation system **100** starts an Object-based move, it may first align any hoists connected to the flown object **903** before continuing an Object-based move. For example, a third hoist **902**C may be coupled to the center of the flown object **903** via a third support line **904**C. An operator may then move the first hoist **902**A to in Machine-based mode to retract the support line **904**A to 2 m, moving the flown object **903** from the first position **900**A to the second position **900**B, but which may leave support line **904**C slack as the flown object **903** tilts and moves up. If the operator then issues an Align command, or initiates a subsequent Object-based command, the stage automation system **100** may automatically align third hoist **902**C to set the support line **904**C to a position it that it determines to be able to keep the flown object **903** stable.

Such functionality offers flexibility and recovery speed for loading and cueing simple and/or complex objects. It will be noted that such functionality is not limited just to rigging automation but could be used with multi-axis industrial robots, platforms lifted by multiple actuators pushing up on them, connected gantry & rotate machines, and the like. For example, at a base level, a movements of a flown object **903** could be programmed entirely in Machine-based cues, and the operator could use the physics engine to pre-visualize how all of the flown object **903** will move in the 3D visualizer.

Additionally, this system can be used to read and translate other automation systems' Machine-based data into 3D information for other show control systems, to eliminate the need to add additional tracking hardware to legacy stage automation systems. For example, at any time, the movements of a flown object **903** represented by Machine-based data (e.g. rotational positioning of a lift winch or chain-motor drum of a hoist **902** and the resulting length of an attached support line **904**) can be translated by the stage automation system **100** to correspond various geometric relationships associated with the resulting movement of the flown object **903** to generate the 3D Object-based parameters for Object-based cueing and programming.

Referring to FIG. **10**, as a flown object **903** moves vertically, it may shift laterally, relative to the hoists **902**, as its center of gravity shifts towards whichever rigging point is holding the most load (e.g. second hoist **902**B). This effect is particularly noticeable as the rigged object gets closer to its high trim position (e.g. first hoist **902**A).

This example shows the difference between a simple rigging calculation (blue) and reality (black) when attempting to tilt an object to 45° virtually. The virtual object is rotated around its center of gravity, the middle of the truss. Even though the calculated and real support line lengths are the same (i.e. 3.6 m and 1.9 m for support line **904**A and support line **904**B, respectively), the real flown object **903** shifts to the right the closer it gets to the second hoist **902**B, because of how the center of gravity behaves with rigging. However, the simple rigging calculator doesn't know that reality doesn't match its virtual 3D calculations; it's only calculating distance between points, without consideration for how gravity will act on the object.

This behavior from a stage automation system **100** is undesirable. A simple rigging calculator will not be fully able to tilt an object to match a desired angle without accounting for gravity. Depending on how the object settles, the lowest point of the object may be lower than the automation system indicates, so a real object may collide with another real object before the stage automation system **100** is able to automatically predict the collision. As shown above, an object will tilt in reality even on just a position move, which is undesirable creatively. If other show control systems depend on the stage automation system **100** to know its position in order to point lights or projection from it, their positions will be inaccurate because of this, too.

There are three levels of physics engine solutions for this situation.

The first, simplest level is to show an operator how the results from the simple rigging calculator will actually settle in reality, like above in FIG. **10**. This alone will at least better inform other show control systems (like the lighting console or projection mapping system) of reality, help the stage automation system **100** know if the object has physically travelled past a limit, and show the operator what will happen so they can compensate for it.

A second, more robust solution is to use the physics engine as part of the rigging calculator, and iteratively adjust either each Machine position (e.g. length of a support line **904**) or each Object parameter (position X/Y/Z, rotation about X/Y/Z) until the physics engine's calculation of reality is within an acceptable margin such that the calculated support line **904** lengths will cause the flown object **903** hit the desired positions and rotations. As shown below, this simple rigged object can be solved for Position Y and Rotation Z by shifting its virtual Position X until the virtual position and the physics engine's output matches within a specified threshold.

Referring to FIG. **11**, an example of a simple rigging problem is shown, with only a single parameter to adjust to find a solution. However, the concept, as a solution, may scale to extremely complex rigging setups. Using a physics engine to verify results, a brute-force iteration solve may be accomplished for any and all parameters. For instance, with six chain hoists on an object that can move in Position Y and rotate around X & Z, a series of values for Positions X & Z and rotation Y may be recursively employed in increasingly smaller step sizes as the physics solve returns a resting position closer to our desired values of Position Y and Rotation X & Z. If a brute-force solve isn't able to find a position and rotation within the desired threshold, the system may simply cancel this move as faulted, because it has determined it's been asked to go to a physically impossible position. As all of these calculations may occur in real time, it allows extremely complex live motion, responding even to how an objects mother grid is moving under control of other machinery.

For example, as shown in FIG. **11**, the physics engine may take as an input "seed" or "target" values for position and rotation of a flown object **903** (defined by a locus at its center of gravity and a defined base rotational value) as shown in blue. As can be seen, the initial position of the flown object **903** is defined as being at a Y position of 2 m, an X position of 0 m, and a rotation of 45° (i.e., support line **904**A is extended to 9.1 m and support line **904**B is extended to 6.3 m). Subsequently, the flown object **903** may simply be moved vertically within the user interface of the stage automation system **100** to a desired position where it has a Y position of 7.5 m, and retains its X position of 0 m, and a rotation of 45° (i.e., support line **904**A is retracted to 3.6 m and support line **904**B is retracted to 1.9 m). These target values are those which are desired to be achieved in the real world. However, as noted herein, gravitational affects may cause a real-world object to sway in the direction of a rigging point supporting the most weight (e.g. to the right in FIG. **11**).

To account for this shift, a center-of-gravity algorithm may be applied to the movement in the target case to yield a naïve solution. The naïve solution may retain the same support line extension/retraction values as the target case but also applies the center-of-gravity algorithm to yield a more accurate position accounting for gravitational affects. For example, as shown in FIG. **11**, the algorithm may yield values, in black, which replicate the support line length values of the target case for the initial position (i.e., support line **904**A is extended to 9.1 m and support line **904**B is extended to 6.3 m) and the final position (i.e., support line **904**A is retracted to 3.6 m and support line **904**B is retracted to 1.9 m). However, as noted in FIG. **11**, under such support line length conditions, when accounting for gravitational effects, the initial state of the flown object **903** may be defined as Y position of 1.97 m, an X position of 0.58 m, and a rotation of 43.1°, and the final state of the flown object **903** may be defined as Y position of 7.2 m, an X position of 0.54 m, and a rotation of 35.6°.

It will be noted that these naïve solution position parameters do not correspond closely to the desired target values. As such, supplemental adjustment may be required to modify the naïve solution to reflect a full-simulation solution that both accounts for gravitational effects and ultimately replicates the desired real-world positioning of the flown object **903**.

To account for this shift, the naïve solution may be provided to the physics engine as an initial condition for an iterative process by which the support line lengths are modified to result in states which replicate the desired real-world behavior of the target case while accounting for those gravitational effects. For example, as shown in FIG. **11**, the algorithm may yield values, in red, which replicate the support line length values of the target case for the initial position (i.e., support line **904**A is extended to 9.1 m and support line **904**B is extended to 6.3 m) and the final position (i.e., support line **904**A is retracted to 3.6 m and support line **904**B is retracted to 1.9 m). However, as noted in FIG. **11**, under such support line length conditions, when accounting for gravitational effects, the initial state of the flown object **903** is be defined as Y position of 1.97 m, an X position of 0.58 m, and a rotation of 43.1°, and the final state of the flown object **903** may be defined as Y position of 7.2 m, an X position of 0.54 m, and a rotation of 35.6°. Such parameters may be iteratively modified until the positional values are within a tolerance level relative to the target case. As shown in FIG. **11**, this iterative algorithm may result in a full-simulation case where the initial state of the flown object **903** is be defined as Y position of 2 m, an X position of 0.58 m, and a rotation of 45°, and the final state of the flown object **903** may be defined as Y position of 7.5 m, an X position of 0.85 m, and a rotation of 45°. As can be seen, the Y position and the rotation of the flown object **903** have been retained relative to the target case and only the X position has been modified to account for the gravitational effects required to achieve the desired target case values.

As shown in FIG. **12**A a target case for a 2D implementation is shown in grey. In FIGS. **12**A-**12**C, the progressive seed (grey) and simulated solution (red) cases are shown. In FIG. **12**D, the target (grey) and corresponding simulated solution (black) are shown.

As shown in FIG. **13**A a target case for a 3D implementation is shown in grey. In FIGS. **13**A-**13**H, the progressive seed (black) and simulated solution (red) cases are shown. In FIG. **13**I, the target (grey) and corresponding simulated solution (black) are shown.

A third, still more robust solution may be to pre-calculate a multidimensional lookup table (LUT) of all controllable parameters on an object, and store the physics engine's result for each set of parameters. This can also be done in conjunction with an RF positioning system and moving the object in reality, to record how the object settles. Then to move to a cued position and rotation, the rigging calculator uses the LUT in reverse, interpolating between closest solved desired positions to get machine position output.

Any or all of the above referenced positional parameters representing an object may be offloaded to a dedicated Exato Instance (usually the Exato Server) via XaStream, using XaValues set in an XaObject in the GeometryManager to request a solve based on specified parameters, and receive the result.

This process can be expanded to include other variables known within the art of rigging, like compensating for catenary weight, solving for time and sheave position to eliminate sway, preventing line slack on unloaded machines at certain places within the flight envelope, calculate and limit load on individual points, and more.

Other systems have attempted to solve these problems by using complex rigging calculators that are specifically formulated for a narrow slice of rigging situations. That solution may be computationally faster, but it is limited to a small subset of problems. Exato's solution is to basically throw CPU cycles at the problem with a brute force solution to find and prove the answer is more flexible, easier to program, and computationally inexpensive with modern technology.

This is an update and clarification on the 3D object-based cueing solver in Exato. This is a subsystem that solves what position various actuators need to get to in order to make what they're attached to go to a certain desired position.

The stage automation system **100** may include four components to implement the 3D path generation described herein: a physics solver module, transformation utilities, a pre-solve trainer, and curve generation utilities.

The physics solver module may simulate how a rigged object will move and settle with constraints that the stage automation system puts on it. For all of its solutions, it uses the same algorithm to iteratively find the 3D position & orientation that will get the simulated coordinate axes to solve for and match the target case (as described above with respect to FIG. **11**). The may be accomplished by running a simulation with an initial seed position (e.g. the target case position or naïve case position referenced above) to the solver, and integrating the non-solved-for-axes from the simulation back into the seed (e.g. using a proportional-integral-derivative (PID) loop). Additionally, as described elsewhere herein, the physics engine may use a pre-generated lookup table (LUT) or a machine learning model that has been trained on a data set output from the physics solver, to first modify the target/seed to a close approximation of the likely output from the solver, to eliminate the number of physics iterations that need to be run.

For example, the physics solver may perform the following calculations:

1) Solve for machine actuator lengths (e.g. support line **904**), given:
 - a target 3D position for a cued object
 - object attachment point positions in 3D space
 - machine attachment points relative to the cued object
 - which coordinate axes to solve for (position X/Y/Z, rotation X/Y/Z, etc.)

2) Solve for machine actuator lengths, given the above, plus:
 - starting 3D object position for the cued object (for instance, where it was 20 milliseconds before it's being requested to be at the target position)
 - starting actuator positions and velocities (for instance, where they were 20 milliseconds ago)
 - time delta from the starting position (for instance, 20 milliseconds)
 - actuator position limits
 - actuator velocity limits
 - actuator acceleration limits
 - Such limits may govern the first function; if the result for the actuators would cause them to exceed any of their position, velocity, or acceleration limits, it will multiply the target (or seed) position & orientation delta by the percentage it exceeded the limit, and attempt to solve again with the limited target.
 - For example, if actuator **1**'s starting velocity was 0.9 m/s and its limit was 1 m/s, and the solve would've caused the actuator to go 1.1 m/s (causing a velocity delta of 0.2 m/s, exceeding the limit by 0.1 m/s), this function would multiply the target position & orientation delta from the starting position & orientation by 0.5, and add the limited delta to the starting position & orientation to create the new target. If a new solve still exceeds an actuator's limits, it'll apply a new multiplier to the delta to limit it, and run the solve again.

3) Solve for 3D object position, given:
 - static attachment points position in 3D space
 - attachment points relative to the cued object
 - machine actuator lengths

4) Solve for resting cued object position, given:
 - initial 3D position for the cued object
 - static attachment points position in 3D space
 - attachment points relative to the cued object

The transformation utilities may address the condition where zero solved coordinate axes are allowed under various limits. In this case, a quaternion-to-quaternion conversion may occur discarding a selected rotational Euler angle to match a target input would be. For example, a rotation coordinate that is modified by the solver (Y, or yaw in airplane terms) may change because of the nature of how physics acts on a fly-rig. To intelligently discard the solved value to convert the fly-rig's rotational position back into a cued X and Z rotation values for the human operator, the following function may be employed:

 - multiply the normalized representation of AxisY (0,1,0) by the input quaternion, returning a Vector3, creating a value (e.g. "targetVectorRotated").
 - generate and return a new quaternion representing the rotation between a Vector3 of zero (0,0,0) and the "targetVectorRotated".

A pre-solve trainer may:

1) Generate a data set of the following:
 - input 3D object position/orientation, number of attachment points, attachment & static points position, and set of coordinate axes to solve for
 - input steps for ever coordinate axis to solve for (for instance, 0.1 m steps for position, 1 degree steps for rotation)
 - output 3D object position/orientation from the solver
 - The output from this pre-solve trainer may be used either directly by the physics solver, either to interpolate between the closest already-solved points, or to feed to a machine learning trainer.

2) Generate a machine learning model based on a pre-generated data set, or a range of data sets including, but not limited to:
 - Angular data must be transformed into a linear representation via sine/cosine encoding or radians
 - Output models are generated to meet the input/output requirements of the solver functions. The same data set is used to generate multiple models.
 - This function is used to generate generic models that ship with Exato, or to create new models when an operator creates a new multi-axis.

3) Generate physics engine tuning parameters, given the same input as the first function in the pre-solve trainer group. This runs the solver multiple times, modifying the physics engine parameters programmatically to test which values (for example—air linear friction, air rotational friction, iterations per step, step time, iterations) solve the fastest while creating a solve that is evaluated as trustworthy (low velocity).

A curve generation utility may perform the following operations:

1) Given 3D position targets, a velocity curve, and limits for the 3D object, create a 3D motion path that smoothly interpolates between each point. The generated curve may use a combination of automatically generated curves and manually adjusted curves. This may include an application of automatically generated Bezier curves, specifically tuned for a performance rigging.

2) Given 3D position targets, a time delta for each position target, and limits for the 3D object, create a 3D motion path that smoothly interpolates between each point. The generated curve may use a combination of automatically generated curves and manually adjusted curves.

Referring to FIGS. **14-17**, various process flow diagrams associated with the solving operations of the physics engine are shown. FIG. **14** shows a process for determining a position of a 3D object. FIG. **15** shows a process for determining support line/actuator lengths resulting a position for a 3D object. FIG. **16** shows a process for determining a position of a 3D object. FIG. **17** shows a process for initiation of solving operations for the physics engine.

Referring to FIGS. **5-6**E-**2** conceptual diagrams of a stage automation system **100**, in accordance with one or more embodiments of the present disclosure are shown.

In particular, the conceptual diagrams illustrated in FIGS. **5-6**E-**2** illustrate how the software components of the stage automation system **100** connect to and control devices (e.g., actionable mechanisms **104**, actionable assemblies **102**), in a standalone configuration. FIGS. **5-6**E-**2** further demonstrate how the distributed program objects within the stage automation system **100** software architecture relate to each other. It is noted herein, however, that FIGS. **5-6**E-**2** are provided solely for example, and are not to be regarded as limiting, unless noted otherwise herein.

The host operating system may include an operating system that contains normal operating system components and functionality including, but not limited to, file system or persistent memory (2), a network stack (3), serial ports (4), RAM, a processor, etc. In some embodiments, the stage automation system **100** may be deployed on some flavor of Linux or Windows on bare metal or a virtual machine, but this environment could also be a Docker container, another generalized container, or an embedded system on a chip with a simple bootloader and operating system-like functionality.

The database file may include persistent storage capable of storing multiple tables of randomly accessible information. This may include a SQLite database file, but could also include a Postgres database managed by a separate Postgres server process, multiple flat CSV files, or key/value pairs stored in EEPROM on an embedded system on a chip. This is not necessarily a requirement for a functioning executing program **106**, but may be required for many core features.

A "network endpoint" may include an IP address that can send and receive normally-switched ethernet packets on a network, or to a single other networked device. The "endpoint" could be a physical RJ-45 ethernet jack with a single IP address, a single IP address amongst many on a physical RJ-45 port, a fiber-optic port, a virtual port on a virtual switch, a WiFi card, a single IP address on multiple bonded ethernet ports, or any other common computer science concept of a network endpoint. Similarly, a "serial port" may include any serial port commonly known in the art. In practice, this serial port and the physical medium (10) may include a physical DB-9 connector on a box, a USB to serial device, or a virtual serial port shared via another process across a network. It could function as half-duplex or full duplex, with any common state signaling that accompanies common serial standards like RS-232, RS-485, CANbus/CANopen, or UART. Typically, this will present itself to the operating system as a/dev/tty* device on Linux, a COM device on Windows, or a serial device object in an embedded system.

The TCP/IP network may include be a wide variety of actual configurations understood as a "network" with a variety of transport types (9) between/amongst devices, from a single ethernet cable direct to another device, a switched network, a connection passed through a VPN or WiFi network, or any other common computer science understanding of a computer network that switches and broadcast packets to devices. In embodiments, the stage automation system **100** may not require any abnormal configurations of a TCP/IP network, and any computer network known in the art may be configured to facilitate communication within the stage automation system **100**.

In embodiments, a "servo drive" in may include any device known in the art configured to provide power and control to machinery (e.g., actionable mechanism **104**, actionable assembly **102**). In the context of the stage automation system **100**, servo drives and/or servo motors may be used to provide predictable closed-loop control. For the purposes of the present disclosure, it is contemplated that predictable, closed-loop control may be applied to any device which is used to achieve a particular location, speed, temperature, or the like. In this regard, any reference to "servo drive" may additionally and/or alternatively be regarded as applying to other mechanisms including, but not limited to, motors (e.g., DC motors, AC motors), hydraulics, pneumatics, and the like. this example, the servo drives may include identical models of a servo drive connected by different communications technologies, or could be entirely different models of servo drive. FIGS. **5-6**E-**2** illustrated the multiple devices (e.g., servomotor, brake, limit switches) connected to a single servo drive, which are required to make an actionable mechanism **104**/actionable assembly **102**. In embodiments, a servo drive may include its own capability to provide power to control motion on the machine (e.g., actionable mechanism **104**, actionable assembly **102**), and the stage automation system **100** provides it configuration and commands to set it up, and to control it from the stage automation system **100**. The stage automation system **100** may regularly poll the drive for information on its status, its configuration, and the status of all its connected peripherals.

The "Instance" may include the executing programs **106**, running on a single operating system (e.g., environment, operating system, a Docker-like container, a virtual machine, or an embedded system). The instance may generate its own unique identifier when it's launched, may keep track of its own internal time since it was launched, and manage the distributed program objects and values within itself. A stage automation system **100** could run off of a single Instance (e.g., executing program) that is able to connect directly to all the machinery, or system could run off of multiple executing programs **106** distributed across the network managing multiple machines. All of the arrows drawn within objects enclosed in the Instance box represents communication directly between software objects.

The concept of the "Environment" in the stage automation system **100** encompasses the configuration of the system, and all of the components able to communicate with each other. A running Instance (executing program **106**) of the software will hold information about the Environment, and multiple Instances across the network will hold shared information about the Environment and be part of the Environment.

The Database object handles communication with the underlying database system as described previously herein. It handles logging many things throughout the Instance's lifecycle, like XaObjects, what XaValues the XaObjects are linked to, the time & value of variables within those XaValues, network packets, debug information, etc. It also reads data for loading saved EnvironmentConfiguration, servo drive configuration for uniquely identified Axes, XaValue history within the Instance or from previous sessions, etc. Everything else in the Instance calls functions within the Database object to request it save or read data, and the Database object abstracts the underlying database file system.

The NetworkManager handles communication on the TCP/IP network on the operating system for the core Exato systems. Anything to and from Exato's core distributed network system passes through it, and it delivers incoming packets to the objects that need it within the Exato Instance, like the XaValueManager. The XaValueManager is a message broker that monitors all active XaValues in the Instance, whether their source is this Instance or others on the network (not pictured in this logical diagram).

The GeometryManager translates position data of axes into 3D information, based on measured or inferred 3D position and orientation of those axes in real space. It also works with the CueManager to translate object-based cueing into jog commands for individual axes, based on where they are in space. For instance, four hoists may be configured to lift a set piece; those four hoists may be at different heights depending on the theater. The operator writes cues for the set piece to go to various heights, and the GeometryManager translates the requested position of the set piece into commands for the hoists to follow, based on their measured height above the stage that day. The Log is a catch-all of messages from software objects within Exato, with various levels of severity that can be filtered by the operator. All of the Log messages are sent to the Database (21), and optionally to the user interface.

The AxisManager holds all of the automation axis objects in the Instance. It monitors the state of all of them, and presents them to be easily accessible and searched by other objects within the Instance. Its configuration (what Axes it contains, what real AxisDevices those contain & their configuration, etc) may be defined by the EnvironmentConfiguration loaded from the Database or saved to the Database.

The CueManager holds the Environment CueSheet(s), and all of the Cues and Commands contained therein. It also holds the CueExecutor, that calls the AxisManager to call functions on individual axes to make them do things described in Commands. It sequences Commands within Cues; a Cue could hold multiple Commands for an axis that need to run sequentially. Its configuration may also be contained in the EnvironmentConfiguration, and may be loaded from or saved to the Database.

An Axis is an object that contains both an AxisSim and optionally an AxisDevice object. The AxisSim virtually simulates how a real device would respond to configurations and cues, for offline programming and setup of a show. This is an XaObject, which in a distributed network environment would share its existence and associated XaValues with other Instances. In a standalone configuration like this, the XaValues would still exist and log their time & values in the Database.

In this diagram, the AxisDevice connects directly to the operating system's network stack to connect to the servo drive it monitors and controls.

FIG. **8** illustrates a flowchart of a method **800** for operating a stage automation system **100**, in accordance with one or more embodiments of the present disclosure. It is noted herein that the steps of method **800** may be implemented all or in part by stage automation system **100**. It is further recognized, however, that the method **800** is not limited to the stage automation system **100** in that additional or alternative system-level embodiments may carry out all or part of the steps of method **800**.

In a step **802**, a distributed program object announcement is received from a first executing program **106***a*. For example, the stage automation server **108** may receive a distributed program object from the first executing program **106***a*. The distributed program object may include a set of one or more time-stamped variables associated with a first actionable mechanism **104***a* selectively controlled by the first executing program **106***a*.

In a step **804**, a distributed program object associated with the first actionable mechanism is generated. For example, the stage automation server **108** may be configured to generate and store a distributed program object associated with the first actionable mechanism **104***a* in memory **112**, wherein the distributed program object includes the set of one or more time-stamped variables.

In a step **806**, the distributed program object is transmitted to a second executing program. For example, the stage automation server **108** may be configured to transmit a distributed program object announcement to the second executing program **106***b*. Subsequently, the one or more data packets are received from the second executing program. For example, the second executing program **106***b* may update/adjust one or more time-stamped variables of the distributed program object, and transmit one or more update data packets to the first executing program **106***a* via the stage automation server **108**.

In a step **808**, at least one time-stamped variable of the distributed program object is adjusted based on the one or more received data packets. For example, upon receiving the one or more update data packets from the second executing program **106***b*, the stage automation server **108** may be configured to adjust the one or more time-stamped variables adjusted by the second executing program **106***b* based on the received update data packets.

In a step **810**, one or more data packets are transmitted to the first executing program indicative of the at least one adjusted time-stamped variable. For example, the stage automation server **108** may be configured to forward the one or more update data packets indicative of the one or more updated time-stamped variables.

In a step **812**, one or more characteristics of the first actionable mechanism associated with the at least one adjusted time-stamped variable are selectively adjusted. For example, the first executing program **106***b* may be configured to generate one or more control signals configured to selectively adjust one or more operational characteristics of the first actionable mechanism **104***a* based on the one or more updated time-stamped variables.

It is believed that the present disclosure and many of its attendant advantages will be understood by the foregoing description, and it will be apparent that various changes may be made in the form, construction, and arrangement of the components without departing from the disclosed subject matter or without sacrificing all of its material advantages. The form described tis merely explanatory, and it is the intention of the following claims to encompass and include such changes.

What is claimed:

1. A system comprising:

a first hoist including a first support line operably couplable to an object suspended from the first hoist;

a second hoist including a second support line operably couplable to the object suspended from the second hoist;

at least one user interface device;

at least one controller device configured to implement one or more instructions for:

receiving one or more inputs defining a first position of a reference point of the object and a second position of the reference point of the object;

calculating one or more support line length adjustments for at least one of the first support line and the second support line corresponding to movement from the first position of the reference point of the object to the second position of the reference point of the object;

controlling at least one of the first hoist and the second hoist to execute one or more support line adjustments to move the reference point of the object from the first position of the reference point of the object to the second position of the reference point of the object;

receiving a third user input, via the at least one user interface, manually controlling at least one of the first hoist and the second hoist to move the object to a manually adjusted position;

recording an amount of support line length adjustment associated with at least one of the first support line and the second support line resulting from the manual control of the at least one of the first hoist and the second hoist to move the object to the manually adjusted position;

computing a third position of the reference point of the object according to the amount of support line length adjustment resulting from the manual control of the at least one of the first hoist and the second hoist.

2. The system of claim **1**, wherein:

the first position of the object includes:

a first position of a reference point of the object, relative to a reference surface, along a vertical axis; and

a first rotational position about the reference point of the object, relative to a reference position,

and the second position of the object includes:

a second position of a reference point of the object, relative to a reference surface, along a vertical axis; and

a second rotational position about the reference point of the object, relative to a reference position.

3. The system of claim **2**, wherein the third position of the reference point of the object includes:

a third position of the reference point of the object, relative to a reference surface, along a vertical axis; and

a third rotational position about the reference point of the object, relative to a reference position.

4. The system of claim **2**, further comprising:

a third hoist including a third support line operably couplable to the object suspended from the third hoist,

wherein the at least one controller device is further configured to implement one or more instructions for:

determining an amount of support line length adjustment associated with the third support line needed to correct for an unstable state resulting from the manual control of at least one of the first hoist and the second hoist of the object to move the object to the manually adjusted position; and

controlling the third hoist to execute one or more support line adjustments to correct for the unstable state.

5. The system of claim **4**, wherein the third hoist including a third support line operably couplable to the object suspended from the third hoist includes:

third hoist including a third support line operably couplable to the object at an intermediate position between a connection point of the first support line and the second support line.

6. The system of claim **4**, wherein the unstable state includes:

a slack state.

7. The system of claim **4**, wherein the controlling the third hoist to execute one or more support line adjustments to correct for the unstable state includes:

controlling the third hoist to execute one or more support line adjustments to controlling the third hoist to execute one or more support line adjustments to correct for the unstable state responsive to a user input command, received via the at least one user interface device, to perform an alignment operation.

8. The system of claim **4**, wherein the controlling the third hoist to execute one or more support line adjustments to correct for the unstable state includes:

controlling the third hoist to execute one or more support line adjustments to controlling the third hoist to execute one or more support line adjustments to correct for the unstable state responsive to controlling at least one of the first hoist and the second hoist to execute one or more support line adjustments to move the reference point of the object.

9. The system of claim **1**, wherein the reference point of the object includes:

a reference point defined by a center-of-gravity of the object.

10. The system of claim **1**, wherein the at least one controller device is further configured to implement one or more instructions for:

receiving an input defining a fourth position of the reference point of the object;

calculating one or more support line length adjustments for at least one of the first support line and the second support line corresponding to movement from the third position of the reference point of the object to the fourth position of the reference point of the object;

controlling at least one of the first hoist and the second hoist to execute one or more support line adjustments to move the reference point of the object from the third position of the reference point of the object to the fourth position of the reference point of the object.

11. The system of claim **1**,

wherein the receiving a first input defining a first position of a reference point of the object includes:

receiving a first user input, via the at least one user interface device, defining the first position of the reference point of the object; and

wherein the receiving a second input defining a second position of a reference point of the object includes:

receiving a second user input, via the at least one user interface device, defining the second position of the reference point of the object.

12. The system of claim **11**, wherein:

the first position of the object includes:

a first position of a reference point of the object, relative to a reference surface, along a vertical axis; and

a first rotational position about the reference point of the object, relative to a reference position,

and the second position of the object includes:

a second position of a reference point of the object, relative to a reference surface, along a vertical axis; and

a second rotational position about the reference point of the object, relative to a reference position.

13. The system of claim **1**, wherein at least one of the receiving a first input defining a first position of a reference point of the object or the receiving a second input defining a second position of the reference point of the object includes:

receiving data indicative of at least one of a rotational position of a hoist or a length of a support line;

translating the data indicative of at least one of a rotational position of a hoist or a length of a support line into a position of a reference point of the object.

14. The system of claim **13**, wherein the position of a reference point of the object includes:

a position of a reference point of the object, relative to a reference surface, along a vertical axis; and

a rotational position about the reference point of the object, relative to a reference position.

15. A system comprising:

a first hoist including a first support line operably couplable to an object suspended from the first hoist;

a second hoist including a second support line operably couplable to the object suspended from the second hoist;

at least one user interface device;

at least one controller device configured to implement one or more instructions for:

receiving one or more inputs defining a first virtual position of a center-of-gravity of the object;

receiving one or more inputs defining a second virtual position of the center-of-gravity of the object;

calculating one or more support line length adjustments for at least one of the first support line and the second support line corresponding to movement from the first position of the center-of-gravity of the object to the second position of the center-of-gravity of the object;

solving for a center-of-gravity adjustment to the second virtual position of the center-of-gravity of the object;

controlling at least one of the first hoist and the second hoist to execute one or more support line adjustments to move center-of-gravity of the object from a position corresponding to the first virtual position of the center-of-gravity of the object to a position corresponding to the center-of-gravity adjustment to the second virtual position of the center-of-gravity of the object.

16. The system of claim 15, wherein the solving for a center-of-gravity adjustment to the second virtual position of the center-of-gravity of the object includes:

solving for a naïve center-of-gravity adjustment to the second virtual position of the center-of-gravity of the object.

* * * * *