March 04, 2019

Internet of Things Three-Dimensional Printer System

Ed Coyne

Internet of Things Three-Dimensional Printer System

Inventor: Ed Coyne

Summary

Aspects of the present disclosure are generally directed to a custom three-dimensional (3D) printer system that allows for the offloading of timing sensitive operations for performance by a microcontroller. In particular, the present disclosure is directed to a 3D printer system that can include multiple classes of processors that are used to perform operations associated with three-dimensional (3D) printing, thereby showcasing a heterogeneous computing system which uses two classes of processor to collaborate to solve a complex problem. In particular, the 3D printer system can facilitate the process of providing 3D printer instructions to a 3D printer.

3D printing can involve use of a 3D model (e.g., a mathematical representation of the surface of some object) from which a physical, 3D printed object is created. One of the initial steps in 3D printing involves the use of software (e.g., slicer software) that operates on a computing device and converts the 3D model into movement instructions that guide the movement of the printing components of the 3D printer. For example, as part of the 3D printing process, 3D printers can receive movement instructions in the form of g-code, which is an ASCII format code that includes movement instructions (e.g., move the printer head from one set of {x,y,z} coordinates to another set of {x,y,z} coordinates).

Because 3D printers typically use a microcontroller that lacks memory to store a complete set of movement instructions locally, the movement instructions (e.g., g-code) are stored on an external storage device (e.g., a solid state drive connected to the 3D printer) or sent to the microcontroller via a serial connection from an external computing device that transmits the instructions to the 3D printer.

By contrast, the microcontroller system of the present disclosure is able to more effectively perform the task of 3D printing by using multiple different classes of processors in the 3D printing process. In particular, in one example embodiment, a first class of processor in a microcontroller can perform the low-latency motor control, while a second class of processor (e.g., that is implementing the Android Things OS), can handle three-dimensional rendering (e.g., OpenGL rendering). For example, OpenGL (Open Graphics Library) rendering can be used to render the object being printed and transition (e.g., line by line) from translucent to solid color as the print progresses (e.g., the object is a 3D rendered progress bar).

By keeping most of the logic on a high-level platform like Android, the proposed systems and methods make development and debugging much easier, thanks to Android's great tooling and developer features. Furthermore, the 3D printer processing flow (e.g., from text g-code to stepper movement commands) can be implemented as a pipeline to allow easy testability and reconfigurability.

In some embodiments, the multiple processors can be located on a single chip while, in other embodiments, the multiple processors can be located on different chips and/or different computing devices. By having a microcontroller and a full SoC on hand, we are able to provide easy access to networked and cloud based sources of g-code

In some embodiments, the first and/or second class of processor can be implemented as a real-time subsystem that showcases the flexibility of Android Things. For example, a proto file can be used as a live buffer for larger than memory prints (e.g., there can be two open handles - a read head and a write tail).

Thus, aspects of the present disclosure are directed to the operation of a heterogeneous computing system that can improve the performance of time sensitive operations such as 3D printing. Further, the 3D printer system can improve 3D printing performance by interacting with Internet of Things devices.

Example Figures

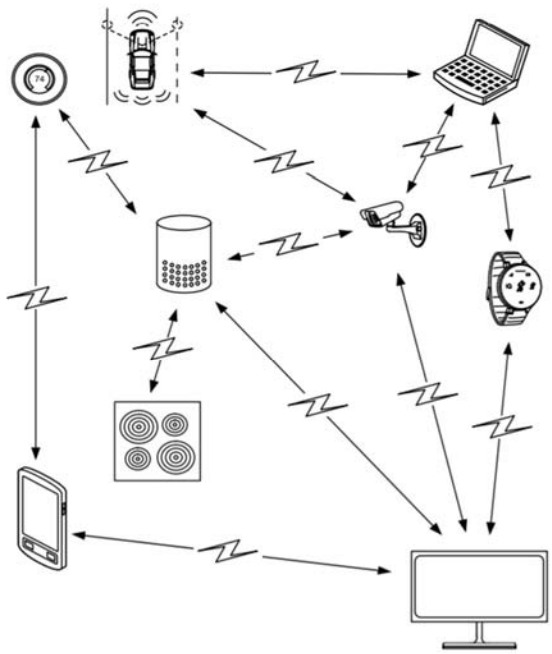

Fig. 1

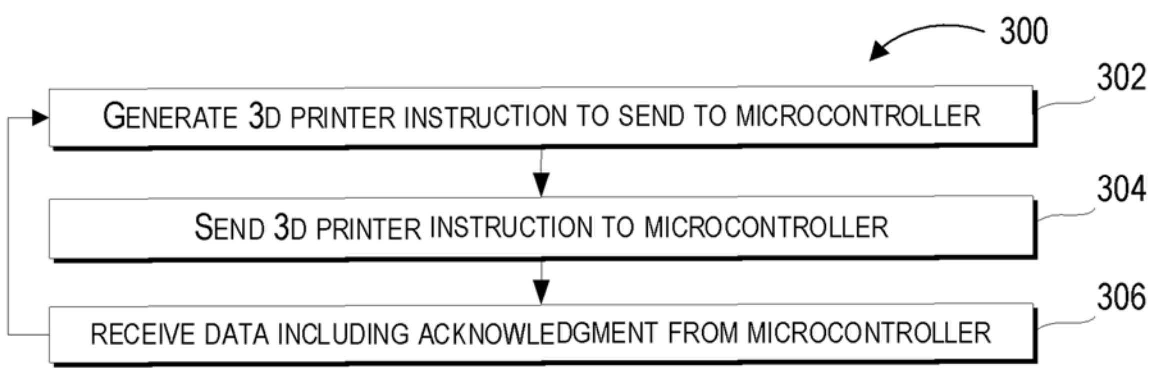

Fig. 3

Detailed Description

Aspects of the present disclosure are directed to a real-time subsystem for Android Things that allows offloading of timing sensitive work to a microcontroller that was integrated as a first class citizen in a SoC (System on a Chip). CNC control and a 3d printer showcase non-trivial use of this system.

Thus, example embodiments of the present disclosure are directed to a three-dimensional printer system that allows for offloading the performance of timing sensitive operations to a separate processor. Further, the device can operate as an Internet of Things (IoT) device within an IoT environment.

Example embodiments of the proposed system are examples of heterogeneous computing, where two classes of processors work together to solve a common goal. This allows the system to do most of the processing on the higher-level system (e.g., inside of an Android Things application). This provides access to a modern development environment with world-class debugging, profiling and testing tools.

This further enables keeping the code running on the microcontroller to a minimum. In one example, the microcontroller code can be a tight loop that simply checks for its next instruction, executes it, then checks again. Instructions for the microcontroller can be simple, such as, for example: "Move stepper-motor A 1000 steps at 100hz, Move stepper-motor B 2000 steps at 10hz, etc.". The microcontroller's job may simply be to schedule these movements simultaneously and perform them on time. This makes the microcontroller code very simple and focused, leaving all of the complexity in the higher-level environment.

The Android application can be responsible for parsing the text-based g-code format, handling high-level machine controls (e.g., turning components on/off, managing temperatures, etc.), as well as tracking the machine position in the 3D coordinate space, translating 3D movements into steps for the stepper motors, and/or managing acceleration ramp up and ramp down. Aside from these fundamental jobs, it is easy to experiment with other functions like mesh-bed leveling, where the system measures the relationship between the print surface and the moving print head and slightly adjusts the 3D coordinate space to account for the machine being out of true.

Working in Android gives access to a large ecosystem of existing libraries. OpenGL enables easy rendering of what the object being printed will look like. Use of Google Drive or Firebase APIs can enable the addition of cloud integration with only trivial amounts of work. Similarly, the existing Camera APIs can be used to capture video or timelapses of the print and share them with the user's phone.

In some embodiments, the Android application itself can have been architected as a pipeline. For example, g-code can be taken from a source, either a file that was uploaded to the printer or from the serial connection on the board. The g-code can then be run through a set of pipeline stages, each responsible for a single focused step of the transformation from text-based g-code to specific machine instructions. This architecture allows each pipeline stage to be easily tested independently so it can be ensured that invariants exist in each stage that other stages can depend on. Furthermore this allows easy experimentation with different approaches including trivially swapping one stage for another.

In some embodiments, at the end of the pipeline the instructions are fed into a buffer of shared memory between the android system and the real-time subsystem running on the microcontroller. This buffer can be implemented without using locks to limit requirements on the hardware itself, which allows this approach to be portable to other platforms easily. As each instruction is processed, the microcontroller can write an acknowledgement back to the Android system.

After the g-code has been processed into specific movements, it is sent both to a buffer where it waits until the microcontroller gets to it, as well as being sent to the front-end for rendering. When the front-end receives the instruction it will render the movement in its OpenGL coordinate space (e.g., it can be rendered in a translucent grey form). This creates a solid-looking 3D rendering of the final print. After the microcontroller has processed a given instruction the front-end will be notified again, at this point it will change the given line from being translucent grey to a solid color to signify that that movement is complete. This process creates a 3D progress-bar effect, where the 3d rendering transitions from translucent grey to solid color line by line as the print proceeds.

Referring to Figure 3, a flow chart illustrating one embodiment of a process 300 for generating instructions for a 3D printer is illustrated in accordance with aspects of the present subject matter. The operations of the process 300 can be performed by a computing system including one or more features of the IoT systems depicted in Figs 1 and 2. Although the operations of the process 300 are shown and described in a particular order, certain operations can be performed in a different order or at the same time.

As indicated in Fig. 3, at 302, the computing system (e.g., the 3D printer system) can generate a 3D printer instruction to send to a microcontroller of a 3D printer. In some embodiments, the computing system can parse a 3D printer instructions (e.g., the text based g-code format), handle high-level 3D printer controls (turning components of the 3D printer on or off and managing temperatures in various components of the 3D printer), track the position of 3D printer components (e.g., mechanical components involved in 3D printing an object) in the three-dimensional coordinate space (e.g., {x,y,z} coordinates), translate 3D printer movements into steps for the stepper motors of the 3D printer, and/or manage acceleration ramp up and ramp down. Aside from these operations, the 3D printer system can also perform operations associated with functions including mesh-bed leveling, in which the relationship between the print surface and the moving print head is measured and the 3D coordinate space is adjusted to account for the 3D printer being out of true.

At 304, the computing system can send one or more 3D printer instructions to the microcontroller. Towards the end of the 3D printing pipeline one or more 3D printer instructions are fed into a buffer of shared memory between the 3D printer system and the real-time subsystem running on the microcontroller. In some embodiments, the buffer can be implemented without using locks to limit requirements on the 3D printer hardware itself thereby allowing this approach to be portable to other platforms. After the g-code has been processed into specific movements, it is sent both to a buffer where it waits until the microcontroller gets to it, as well as being sent to the front-end for rendering. When the front-end receives the instruction it will render the movement in its OpenGL coordinate space, it will render it in a translucent grey form. This creates a solid-looking 3d rendering of the final print.

At 306, the computing system can receive data including an acknowledgement from the microcontroller. As each 3D printer instruction is processed, the microcontroller can write an acknowledgement back to the 3D printer system. After the microcontroller has processed a given instruction the front-end of a computing system used by a user (e.g., a computing device connected to a display device viewable by a user) can be notified again.

In some embodiments, the 3D printer system can access an ecosystem of existing 3D printer libraries. For example, OpenGL can be used to render what an object being 3D printed will look like before the completion of 3D printing.

In some embodiments, existing Camera APIs can be used to capture video or time lapses of a 3D print job and shared with a user on the user's phone. For example, the OpenGL rendering of the amount of the 3D model that has been 3D printed can be sent to a user's smartphone every minute until the 3D print job is complete.

In some embodiments, at this point the image displayed on a display device will change the given line from being translucent grey to a solid color to signify that that movement is complete. This process creates a 3D progress-bar effect in which the 3D rendering transitions from translucent grey to a solid color on a line by line basis as the printing operations proceed.

The devices and flows shown in Figures 1-3 are only examples of how devices including the 3D printer system can be made and operated. Many other devices, including different devices can be created according to aspects of the present disclosure.

Referring now to Figure 1, Figure 1 depicts a block diagram of an example IoT environment according to example implementations of the present disclosure. As illustrated in Figure 1, in some implementations, the IoT environment includes a plurality of different devices, each of which can be referred to as an IoT device. An example IoT device can be an intelligent, environmentally-sensing, and/or network-connected device configured to communicate with a central server or cloud service, a control device, and/or one or more additional IoT devices to perform any number of operations (e.g., in response to received commands). IoT devices can, in some instances, also be referred to as or include "smart" devices and/or "connected" devices.

Each IoT device can be a stand-alone physical device or can, in some instances, be an embedded device that is embedded within a larger device or system. Each IoT device can include electronics, software, sensors, actuators, and/or other components, including various components that sense, measure, control, and/or otherwise interact with the physical world. An IoT device can further include various components (e.g., a network interface, antennas, receivers, and/or the like) that enable the IoT device to send and/or receive data or other information from one or more other IoT devices and/or to a central system.

In particular, the various IoT devices can be communicatively connected to one another via one or more communications networks. The networks can be wide area networks, local area networks, personal area networks, piconets, cellular networks, other forms of networks, and/or combinations thereof. The networks can be wired and/or wireless. The networks can be private and/or public. As examples, two or more of the IoT devices can communicate with one another using a Wi-Fi network (e.g., a home network), Bluetooth, Bluetooth Low Energy, Zigbee, Radio-Frequency Identification (RFID), machine to machine connections, inductive communications, optical communications, infrared communications, other communications techniques or protocols, and/or combinations thereof. For example, an IoT device might communicatively connect with a first nearby device using Bluetooth while also communicatively connecting with a second nearby device using Wi-Fi.

In some implementations, each IoT device can have a unique identifier. For example, the identifier for each IoT device can include and/or be based on an Internet Protocol address associated with such IoT device, a manufacturer associated with such IoT device, a location at which such IoT device is positioned, a model number of such IoT device, a functionality of such IoT device, and/or other device characteristics. In some implementations, a given IoT device can locate and/or communicate with another IoT device based on its unique identifier. In some implementations, the identifier assigned to an IoT device can be modified by a user and/or owner of such IoT device.

In particular, in some implementations, a user can assign one or more identifiers to the IoT devices within a device topology representation. The device topology representation can describe and/or organize a group of IoT devices (e.g., based on location with one or more structures such as one or more homes, offices, vehicles, and/or the like). The identifiers can be chosen by the user and associated with the respective IoT devices within the device topology representation. The identifier(s) can include but are not limited to names, nicknames, and/or aliases selected for the IoT devices by the user. In this manner, the identifiers can be names or aliases of the respective IoT devices that the user is likely to use when identifying the IoT devices for requested control or command operations (e.g., “turn on the kitchen lights”).

An IoT device can be mobile or can be stationary. In some implementations, an IoT device can be capable of autonomous or semi-autonomous motion.

In some implementations, an IoT device can be controlled or perform operations in response to communications received by the IoT device over a network. For example, an IoT device can be controlled by a control device that is communicatively connected to the IoT device. The control device can communicate directly with the IoT device or can communicate indirectly with the IoT device (e.g., over or using a mesh network). The control device can itself be an IoT device or the control device can be a device that is not considered part of the IoT environment. For example, the control device can be a server device that operates as part of cloud computing system. The commands can be in response to or generated based on a user input or can be generated without user input.

Thus, in one example, an IoT device can receive communications from a control device and the IoT device can perform operations in response to receipt of such communications. The performed operations can be internal operations (e.g., changing an internal setting or behavior) or external operations (e.g., interacting with the physical world in some way). The IoT device and the control device can be co-located or can be remotely located from each other.

As an example, the control device can be or include a user device such as a smartphone, tablet, computing device that is able to be worn, laptop, gaming console or device, virtual or augmented reality headset, and/or the like. As another example, the control device can be a server computing device. As another example, the control device can itself be an IoT device. For example, the control device can be a so-called “smart speaker” or other home control or automation device.

In some implementations, a user can interact with a control device (e.g., which can be an IoT device) to input data into or otherwise control the IoT environment. For example, the control device can include and execute a software application and/or other software programs that provide a user interface that enables entry of user input. The software applications can be executed entirely at the control device or can be web applications where some portion of the program or functionality is executed remotely (e.g., by a server connected over the Internet) and, in some implementations, the client-side logic runs in a web browser. Thus, in some implementations, a web server capable of sending, receiving, processing, and storing web pages or other information may be utilized.

In some implementations, a cloud service may be used to provision or administer the IoT devices. For example, a cloud computing system can enable or perform managed and/or integrated services that allow users to easily and securely connect, manage, and ingest IoT data from globally dispersed IoT devices at a large scale, process and analyze/visualize that data in real time, and/or implement operational changes and take actions as needed. In particular, in some implementations, the cloud computing system can employ a publication subscription model and can aggregate dispersed device data into a single global system that integrates seamlessly with data analytics services. An IoT data stream can be used for advanced analytics, visualizations, machine learning, and more to help users improve operational efficiency, anticipate problems, and/or build rich models that better describe and optimize the user's home or business. The cloud system can enable any number of dispersed IoT device to connect through protocol endpoints that use automatic load balancing and horizontal scaling to ensure smooth data ingestion under any condition.

In some implementations, the cloud system can include or implement a device manager. For example, the device manager can allow individual IoT devices to be configured and managed securely in a fine- or coarse-grained way. Management can be done through a console or programmatically. The device manager can establish the identity of a device and can provide the mechanism for authenticating a device when connecting. The device manager can also maintain a logical configuration of each device and can be used to remotely control the device from the cloud.

In some implementations, an IoT device can include an artificial intelligence-based assistant or software agent. A user can interact with the artificial intelligence-based assistant via a control device, directly through the IoT device, or any other method of interaction. The artificial intelligence-based assistant can perform tasks or services based on user input and/or contextual awareness (e.g., location awareness), including acting as a control device to control other IoT devices. In some implementations, an IoT device (e.g., an artificial intelligence-based assistant on such device) can access information from a variety of online sources (such as weather conditions, traffic congestion, news, stock prices, user schedules, retail prices, etc.).

The artificial intelligence-based assistant or software agent can be stored and implemented by a single device (e.g., a single IoT device) or can be spread across multiple devices and implemented by some (e.g., dynamically changing) combination of such multiple devices.

In some implementations, an IoT device can include (e.g., as part of an artificial intelligence-based assistant) one or more machine-learned models that assist in understanding user commands, determining context, and/or other actions. Example machine-learned models include artificial neural networks such as feed-forward neural networks, recurrent neural networks, convolutional neural networks, autoencoders, generative adversarial networks, and/or other forms, structures, or arrangements of neural networks. Additional example machine-learned models include regression models, decision tree-based models (e.g., random forests), Bayesian models, clustering models, linear models, non-linear models, and/or other forms, structures, or arrangements of machine-learned models. Machine-learned models can be trained using supervised learning techniques or unsupervised learning techniques. Machine-learned models can be stored and implemented on the IoT device or can be stored and implemented in the cloud and the IoT device can leverage the models by communicating with cloud devices. Feedback or other forms of observed outcomes can be used to re-train models to improve their performance. Models can be personalized to one or more users or environments by re-training on data specific to such users or environments.

In some implementations, the artificial intelligence-based assistant can perform concierge-type tasks such as, for example, making dinner reservations, purchasing event tickets, making travel arrangements, and/or the like. In some implementations, the artificial intelligence-based assistant can provide information based on voice input or commands (e.g., by accessing information from online sources). In some implementations, the artificial intelligence-based assistant can automatically perform management or data-handling tasks based on online information and events, including, in some instances, without user initiation or interaction.

In some implementations, a control device (e.g., which may be an IoT device) can include components such as a mouse, a keyboard, buttons, knobs, a touch-sensitive component (e.g., touch-sensitive display screen or touch pad), and/or the like to receive input from the user via physical interaction.

In some implementations, the control device can include one or more microphones to capture audio signals and the device can process the audio signals to comprehend and respond to audio commands (e.g., voice commands) provided by a user or by some other device. Thus, in some implementations, the IoT devices can be controlled based on voice commands from a user. For instance, a vocalization from a user can be received by a control device. The vocalization can be a command spoken by a user proximate to the control device. The control device can control itself and/or one or more of the IoT devices based on the vocalization.

In some implementations, one or more vocalization(s) may be used as an interface between a user and an artificial intelligence-based assistant. For example, a user may vocalize a command which the artificial intelligence-based assistant may identify, process, and/or execute or cause to be executed. The vocalized command may be directed at the artificial intelligence-based assistant.

As one example, the vocalization may indicate a user desire to interact with or control another IoT device (e.g., lowering a thermostat setting, locking a door, turning off a light, increasing volume, etc.). The artificial intelligence-based assistant may communicate the command to the desired IoT device which can respond by executing or otherwise effectuating the user command. As another example, the vocalization can include a user commanding the artificial intelligence based assistant to perform a task (e.g., input an event into a calendar, retrieve information, set a reminder, make a list, define a word, read the first result of an internet search, etc.).

In some implementations, speech recognition or processing (e.g., natural language processing) can be performed on the vocalization to comprehend the command provided by the vocalization. For instance, data indicative of the vocalization can be provided to one or more language models (e.g., machine-learned models) to determine a transcription of or otherwise process the vocalization.

In some implementations, processing the vocalization or other user input can include determining one or more IoT devices to control and/or one or more actions to be performed by the selected IoT devices. For instance, a semantic interpretation of the vocalization (e.g., a transcript of the vocalization) can be determined using one or more semantic interpretation techniques (e.g., natural language processing techniques). The semantic interpretation can provide a representation of the conceptual meaning of the vocalization, thereby also providing an interpretation of the intent of the user.

In some implementations, the interpretation of the vocalization can be determined based at least in part on the device topology representation. For instance, the device topology representation can be accessed to determine the one or more selected IoT devices and/or actions to be performed. As one example, the device topology representation can be accessed and compared against a transcription of the vocalization to determine a match between one or more terms included in the transcription and one or more terms associated with the IoT device topology representation (e.g., “kitchen lights”). In some implementation, the identity of the speaker can be ascertained and used to process the vocalization (e.g., such as to process commands that include possessive modifiers: “brew a cup of my favorite roast of coffee”).

In some implementations, the control device (e.g., which may be an IoT device) can include a vision system that includes one or more image sensors (e.g., visible-spectrum cameras, infrared cameras, LIDAR systems, and/or the like) that capture imagery. The device can process the imagery to comprehend and respond to image-based commands or other input such as, for example, gesture commands provided by the user. In some implementations, the vision system may incorporate or perform facial movement identification (e.g. lip reading) capabilities while, in other implementations, the vision system may additionally or alternatively incorporate hand shape (e.g. hand gestures, sign language, etc.) identification capabilities. Facial movement and/or hand shape identification capabilities may allow a user to give commands a control device in addition or alternatively to voice control.

In some implementations, in response to the image data of the facial or hand gesture, the control device can determine one or more IoT devices to control and/or one or more actions to be performed (e.g., by the selected IoT devices). Interpretation of image data that depicts lip reading and/or sign language may be achieved through any method of image data analysis. The interpretation can provide a representation of the conceptual meaning of the image data. In this manner, the interpretation of the image data can provide an interpretation of the intent of the user in performing the gesture(s).

In some implementations, the interpretation can be determined based at least in part on the device topology representation. For instance, the device topology representation can be accessed to determine the one or more selected IoT devices and/or the action to be performed. For example, the device topology representation can be accessed and compared against the image data to determine a match between one or more aspects of the image data and one or more aspects associated with the IoT device topology representation (e.g., the user may be pointing to a specific IoT device when providing a voice command or a gesture command).

In further implementations, gaze recognition can be performed on the captured imagery to identify an object or device that is the subject of a gaze of the user. A user command (e.g., voice or gesture) can be interpreted in light of (e.g., as applied to) the object or device that is the subject of the gaze of the user.

In some implementations, the vision system may be used as an interface between a user and an artificial intelligence-based assistant. The captured image data may be interpreted and/or recognized by the artificial intelligence-based assistant.

In some implementations, the selected IoT devices and/or the actions to be performed can be determined based at least in part on contextual data (e.g., location of user, day of the week, user data history, historical usage or command patterns, user wardrobe, etc.) For instance, in response to receiving a command from a user, a location of the user, a time of day, one or more past commands, and/or other contextual information can be determined. The location can be determined using various suitable location determination techniques. The location determination technique can, for example, be determined based at least in part on the control device to which the user provides the vocalization.

As one example, if the control device is an IoT device that is specified in the device topology representation, the user location can be mapped to the structure and/or room to which the control device is assigned in the device topology representation. As another example, if the control device is a user device not specified in the device topology representation, the user location can be determined using one or more location determination techniques, such as techniques using wireless access points or short range beacon devices associated with one or more IoT devices, and/or other suitable location determination techniques. In some implementations, the user location can be mapped to one or more structures and/or rooms specified in the device topology representation. In some implementations, the control device and/or other IoT devices can also process audio signals and/or imagery to comprehend and respond to contextual information. As examples, triangulation and/or beamforming techniques can be used to determine the location of the user based on receipt of the voice command at multiple different audio sensors. In some implementations, multiple possible user commands or requests can be disambiguated based on the contextual information.

Further to the descriptions above, a user may be provided with controls allowing the user to make an election as to both if and when systems, devices, or features described herein may enable collection of user information (e.g., information about a user's preferences, a user's activities, or a user's current location), and if the user is sent content or communications from a server. In addition, certain data may be treated in one or more ways before it is stored or used, so that personally identifiable information is removed. For example, a user's identity may be treated so that no personally identifiable information can be determined for the user, or a user's geographic location may be generalized where location information is obtained (such as to a city, ZIP code, or state level), so that a particular location of a user cannot be determined. Thus, the user may have control over what information is collected about the user, how that information is used, and what information is provided to the user.

Figure 2 provides a block diagram of an example software stack that can be included on an IoT device. The software stack shown in Figure 2 is provided as one example only. Various different IoT devices can have any number of different software and/or hardware configurations which may be of greater or lesser complexity to that shown in Figure 2.

In some implementations, an IoT device can include and execute one or more computer applications (also known as software applications) or other computing programs. The IoT device can execute the application(s) to perform various functions, including collection of data, communication of data, and/or responding to or fulfilling received commands. In some implementations, the software applications can be native applications.

In some implementations, the software application(s) on an IoT device can be downloaded and installed by or at the direction of the user. In other implementations, the software application(s) can be default applications that come pre-programmed onto the IoT device. In some implementations, the software application(s) can be periodically updated (e.g., via download of update packages). The software application(s) can be closed source applications or can be open source applications. The software applications can be stand-alone applications or can be part of an operating system of the IoT device. The software applications can be embodied in computer-readable code or instructions that are stored in memory and then accessed and executed or followed by one or more processors of the IoT device.

In some implementations, the software application(s) on an IoT device can be user-facing applications such as a launcher or a browser. In other implementations, the IoT device does not include any user-facing applications but, for example, is instead designed to boot directly into applications developed specifically for the device.

More particularly, in some implementations, an IoT device can include or otherwise be implemented upon or in conjunction with an IoT platform that includes a number of elements.

The IoT platform can include an operating system. The operating system can, for example, have been optimized for use in the IoT environments (tuned for faster boot times and/or lower memory footprint). The operating system and other platform elements may be able to receive secure and managed updates from the platform operator. The IoT platform can include hardware that is accessible and easy to integrate.

The IoT platform can also enable application developers to build applications using a rich framework provided by an operating system software development kit (SDK) and platform services, including, for example, the same user interface toolkit, multimedia support, and connectivity application programming interfaces (APIs) used by developers of mobile applications for larger devices such as smartphones. Applications developed for the IoT device can integrate with various services using one or more client libraries. For example, the applications can use the libraries to interact with services such as application deployment and monitoring services, machine learning training and inference services, and/or cloud storage services. The applications can use the APIs and/or support libraries to better integrate with hardware, including, for example, custom hardware. This can include support for peripheral interfaces and device management. The device can also include a number of native libraries, including, for example, C/C++ libraries, runtime libraries, core libraries, and/or the like. Updates to one or more of these components can be deployed over the air and/or automatically when updates are available.

In some implementations, an IoT device (e.g., the software applications executed thereby) can utilize APIs for communicating between a multitude of different software applications, operating systems, databases, libraries, enterprises, graphic interfaces, or any other component of the IoT environment disclosed herein. For instance, a first software application executed on a first IoT device can invoke a second software application via an API call to launch the second software application on a second IoT device.

In some implementations, the applications can run on a single or variety of operating system platforms including but not limited to OS X, WINDOWS, UNIX, IOS, ANDROID, SYMBIAN, LINUX, or embedded operating systems such as VxWorks.

The IoT device can include one or more processors and a memory. The one or more processors can be any suitable processing device (e.g., a processor core, a microprocessor, an application specific integrated circuit (ASIC), a field programmable gate array (FPGA), System on a Chip (SoC), a controller, a microcontroller, etc.) and can be one processor or a plurality of processors that are operatively connected. The memory can include one or more non-transitory computer-readable storage mediums, such as RAM, ROM, EEPROM, EPROM, flash memory devices, magnetic disks, etc., and combinations thereof. The memory can store data and instructions which are executed by the processor to cause the IoT device to perform operations. The IoT devices can, in some instances, include various other hardware components as well, including, for example, a communications interface to enable communication over any number of networks or protocols, sensors, and/or other components.

In some implementations, the IoT device can include or be constructed using one or more System on Module (SoM) architectures. Each SoM can be a fully integrated component that can be dropped directly into a final design. Modules can be manufactured separately and combined to form the device. In some implementations, the device software can include a hardware abstraction layer and kernel which may be packaged as a board support package for the modules. In other implementations, different, non-modular architectures can be used.

Example IoT devices include or can be associated with an air conditioning or HVAC system, lighting device, a television or other home theater or entertainment system, security system, automatic door or window locking system, thermostat device, home energy manager, home automation system, audio speaker, camera device, treadmill, weight scale, smart bed, irrigation system, garage door opener, appliance (e.g., refrigerator, dishwasher, hot water heater, furnace, stove, fireplace, etc.), baby monitor, fire alarm, smoke alarm, medical devices, livestock tracking devices, cameras, beacon devices, a phone (e.g., smartphone), a computerized watch (e.g., a smart watch), a fitness tracker, computerized eyewear, computerized headwear (e.g., a head mounted display such as a virtual reality of augmented reality display), other types of computing devices that are able to be worn, a tablet, a personal digital assistant (PDA), a laptop computer, a desktop computer, a gaming system, console, or controller, a media player, a remote control, utility meter, an electronic book reader, a navigation system, a vehicle (e.g., car, boat, or plane/drone) or embedded vehicle system, an environmental, food, or pathogen monitor, search and rescue devices, a traffic control device (e.g., traffic light), traffic monitor, climate (e.g., temperature, humidity, brightness, etc.) sensor, agricultural machinery and/or sensors, factory controller, GPS receivers, printers, motor (e.g., electric motor), and/or other suitable device or system.

The technology discussed herein makes reference to servers, databases, software applications, and other computer-based systems, as well as actions taken and information sent to and from such systems. One of ordinary skill in the art will recognize that the inherent flexibility of computer-based systems allows for a great variety of possible configurations, combinations, and divisions of tasks and functionality between and among components. For instance, server processes discussed herein may be implemented using a single server or multiple servers working in combination. Databases and applications may be implemented on a single system or distributed across multiple systems. Distributed components may operate sequentially or in parallel.

Figures

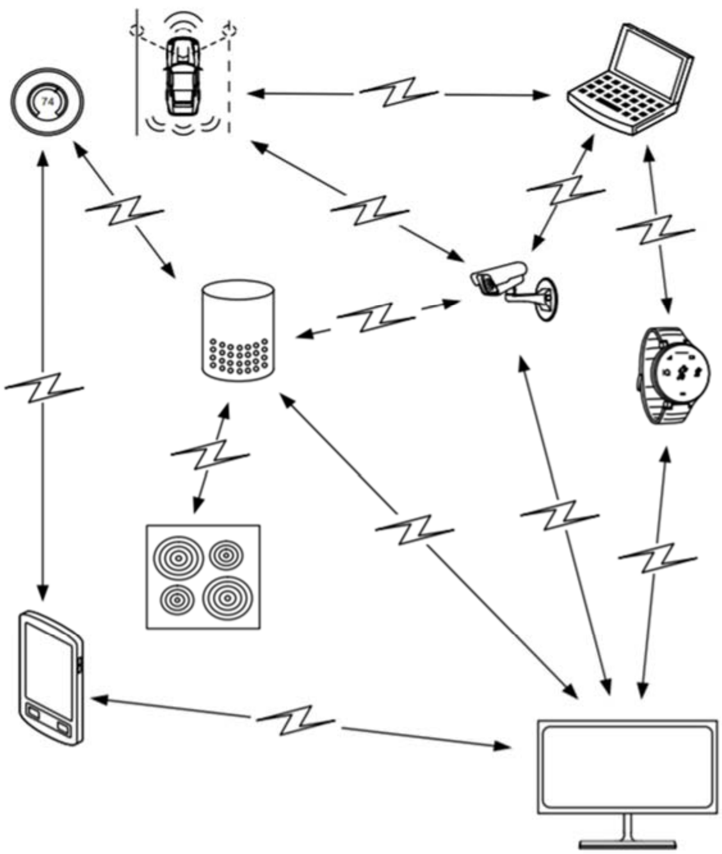

Fig. 1

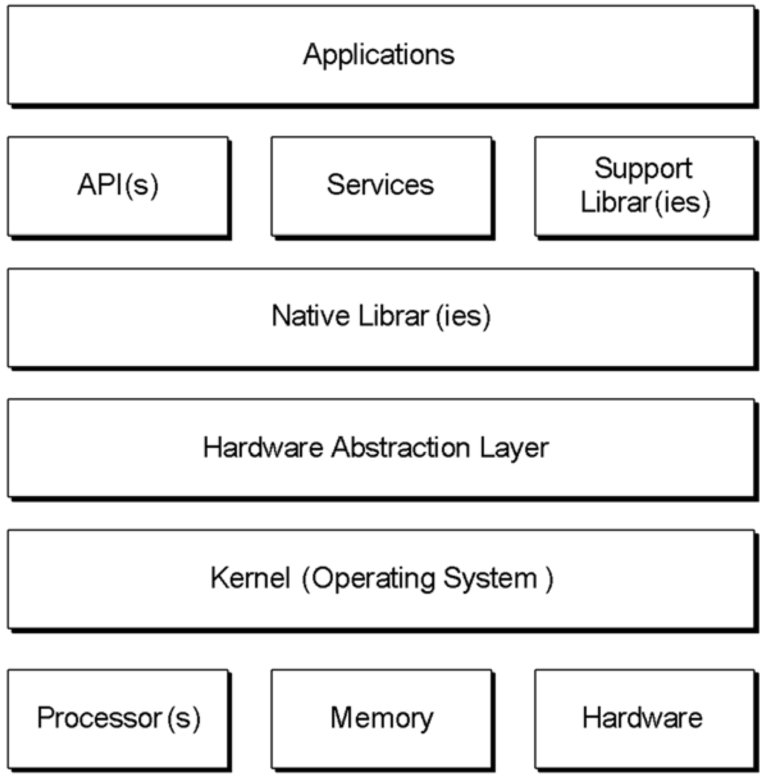
Applications
API(s)
Services
Support Librar(ies)
Native Librar (ies)
Hardware Abstraction Layer
Kernel (Operating System)
Processor (s)
Memory
Hardware

Fig. 2

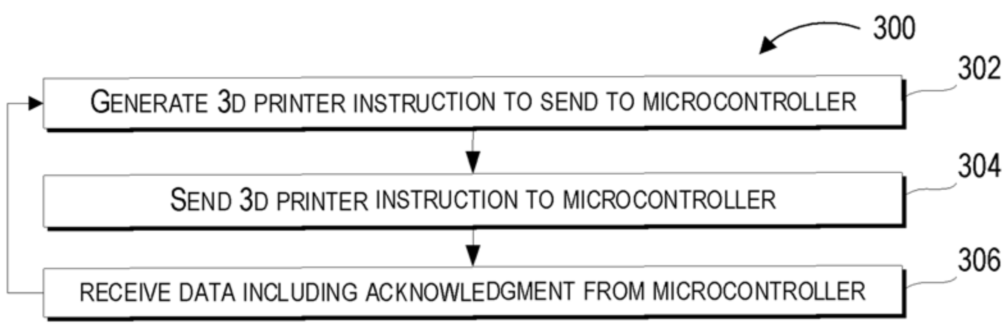
300
302
GENERATE 3D PRINTER INSTRUCTION TO SEND TO MICROCONTROLLER
304
SEND 3D PRINTER INSTRUCTION TO MICROCONTROLLER
306
RECEIVE DATA INCLUDING ACKNOWLEDGMENT FROM MICROCONTROLLER

Fig. 3