March 04, 2019

Reproduction of Captured Imagery via Machine Learning and Device Locomotion

Melissa Daniels

Ryan Bae

Reproduction of Captured Imagery via Machine Learning and Device Locomotion

Inventors: Melissa Daniels and Ryan Bae

Summary

Aspects of the present disclosure are directed to mobile robotic devices that are capable of reproducing captured imagery (e.g., images captured by a camera that is on-board the robotic device) through the use of device locomotion. For example, the robotic device can capture (or be operated to capture) an image (e.g., a self-portrait photograph which is also known as a “selfie”) and can then reproduce the image (e.g., a stylized version thereof) by controlling motion of at least a part of the robotic device to draw at least a portion of the captured scene onto a medium. For example, the robotic device can move a portion of the device that holds a pen, pencil, marker, paintbrush, etching tool, and/or the like to draw the scene onto a piece of paper, wood, canvas, glass, metal, etc. In addition, in some implementations, machine learning techniques such as “style transfer” techniques can be used to stylize the captured imagery into a particular style (e.g., a pencil sketch style portrait) prior to reproduction.

Thus, aspects of the present disclosure are directed to the application of robotics and motors to create a visual portrait using on-device processing. One example of the aspects described herein includes a selfie-taking and selfie-drawing robot which can be referred to as “DrawBot”. DrawBot can take a picture of a user and then draw it on a piece of paper by driving around. For example, commands like “go forward 10” and “turn 90 degrees right” can be used to make complex shapes, such as drawing a full face included in the selfie. DrawBot can be built using an embedded operating system platform such as the Android Things operating system, thereby demonstrating the capabilities of Android Things and leveraging its flexibility as an operating system. One

example implementation of DrawBot was built in two weeks, showing how easy it is to prototype a new idea using Android Things.

Example Figures

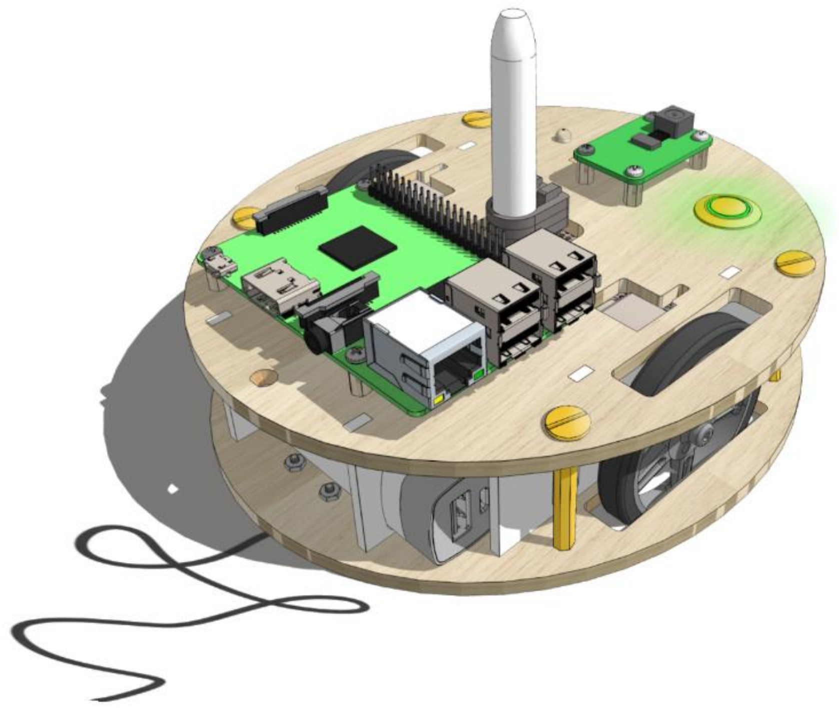


Fig. 3

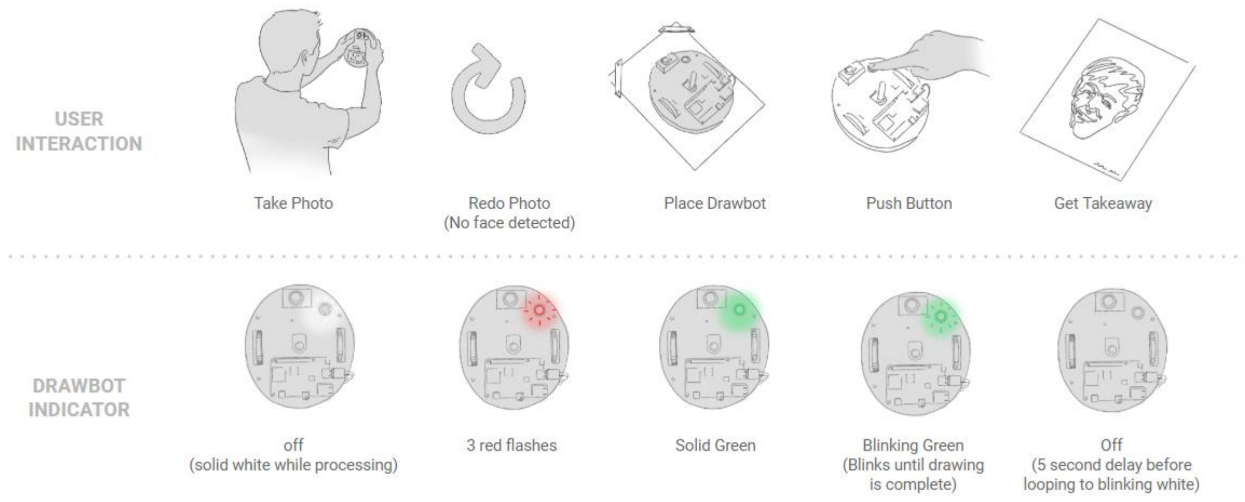

USER INTERACTION
Take Photo
Redo Photo (No face detected)
Place Drawbot
Push Button
Get Takeaway
DRAWBOT INDICATOR
off (solid white while processing)
3 red flashes
Solid Green
Blinking Green (Blinks until drawing is complete)
Off (5 second delay before looping to blinking white)

Fig. 7

Detailed Description

As described above, the present disclosure is directed to mobile robotic devices that are capable of reproducing captured imagery (e.g., images captured by a camera that is on-board the robotic device) through the use of device locomotion. In some implementations, the robotic device can operate as an Internet of Things (IoT) device within an IoT environment.

Referring first to Figures 3-5 Figures 3-5 which are included at the end of this document depict example embodiments of the robotic devices described herein. Figure 6 depicts a schematic diagram of an example robotic device and accompanying system as described herein. As illustrated in Figure 6, in some implementations, the robotic device can include and run on an i.MX7D Pico Development board, running Android Things. In some implementations, the robot device can also use the camera module from the Pico i.MX7 StartKit.

Android Things is an operating system that can run on embedded hardware, like a single board computer. It can run on a Raspberry Pi 3, or on the i.MX7D board. A user (e.g., developer) can write Java code for Android Things in Android Studio, similar to the development process for developing an Android application for a smartphone. Android Things also supports Kotlin.

In some implementations, the robotic device can be powered by two USB power bank batteries. These are off-the-shelf and make it very easy to get started prototyping the portable device. One battery powers the board, and the other powers the motors.

In some implementations, the robotic device can have a chassis that is built from laser-cut acrylic and wood. In some implementations, there are no 3D printed parts. The wheels, tires, motors, casters, and batteries can be all off-the-shelf parts available to hobbyists. The camera, camera cable, and Development board all come in the PICO i.MX7 StartKit.

Figure 7 depicts an example user flow and corresponding indicator feedback for interactions with an example robotic device as described herein. The robotic device can include and use a medium-resolution camera module to captured an image (e.g., a selfie of a user). The robotic device can use a simple button to get user input.

Once the device has taken the picture, it can go through these steps to make it into a drawing: Uses the OpenCV application programming interface (API) to find a face in the camera image; Crops the image to a close up around the face, and resizes it to 30 pixels tall; Each pixel is mapped to a brightness value from 0 - 3 (0 is white, 3 is the darkest); and Generating the drawing based on the brightness values.

In some implementations, the robotic device can include and implement an on-device machine learning library (e.g., TensorFlow or TensorFlow Lite) in order to run certain machine learning techniques on the robotic device. Thus, the robotic device can run machine-learned models "on-device". As one example, the robotic device can run certain machine-learned models (e.g., neural networks) on-device in order to detect a face in the captured image. For example, a face detection model can provide an output that indicates whether a face is detected in the image and, if so, where the face is located (e.g., as a bounding box or as a segmentation mask). As another example, machine learning techniques such as "style transfer" techniques can be used to stylize the captured imagery into a particular style (e.g., a pencil sketch style portrait) prior to reproduction. For example, a machine-learned model can be run to produce brightness values for each pixel or for groups of pixels which cause the drawing to exhibit a certain style (e.g., a user-selected style). As yet another example, machine-learned models can be used to generate a set of motor commands that, when executed by the robotic device, cause the robotic device to produce the drawing.

For example, to generate the drawing, the robotic device can drive across the paper. While driving, the device can press down the marker (or other drawing tool) with different levels of pressure to create wider or thinner lines. These pressure levels can correspond to the brightness values. In some implementations, the robotic device can use a servo motor to move the pen, and two stepper motors to move the wheels. In some implementations, the robotic device can use an LED for feedback. For example, it can have Red, Green, & Blue dies and can show any color.

In some implementations, a user of the robotic device can hold the device at arm's length and direct the camera (e.g., a small square to the left of the pushbutton) towards their face.

One example interaction may be as follows: The user can press the button once. The LED will turn yellow while the image is processed. If the LED goes back to red, the image must be re-taken (no face detected.) If the LED goes to green, the robotic device is ready to draw. The user can place the robotic device in the center of the paper. The user can press the button to start drawing. The LED will turn purple, then blue and the bot will begin moving.

Figure 8 depicts an example captured image of a face on the left-hand side and, on the right-hand side, an example drawing produced by a robotic device based on the example image.

Appendix A of this document provides a bill of materials and instructions for constructing one example robotic device as described herein. The devices and flows shown in Figures 3-8 and Appendix A are only examples of how devices can be made and operate. Many other and different devices can be generated 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

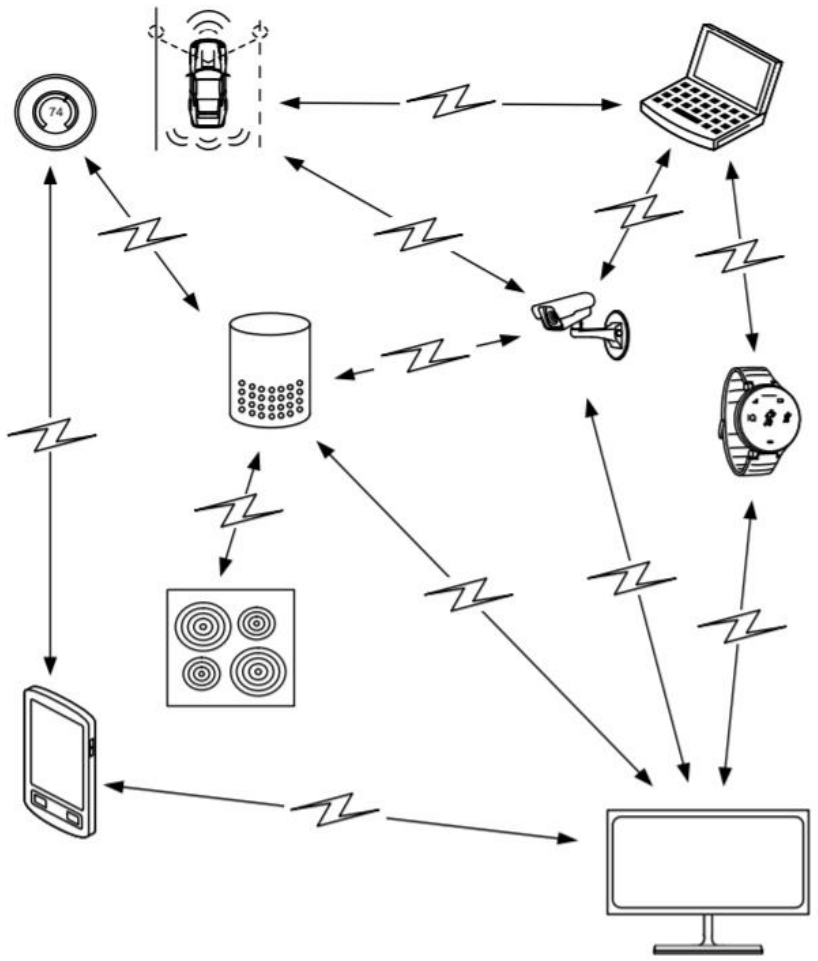


Fig. 1

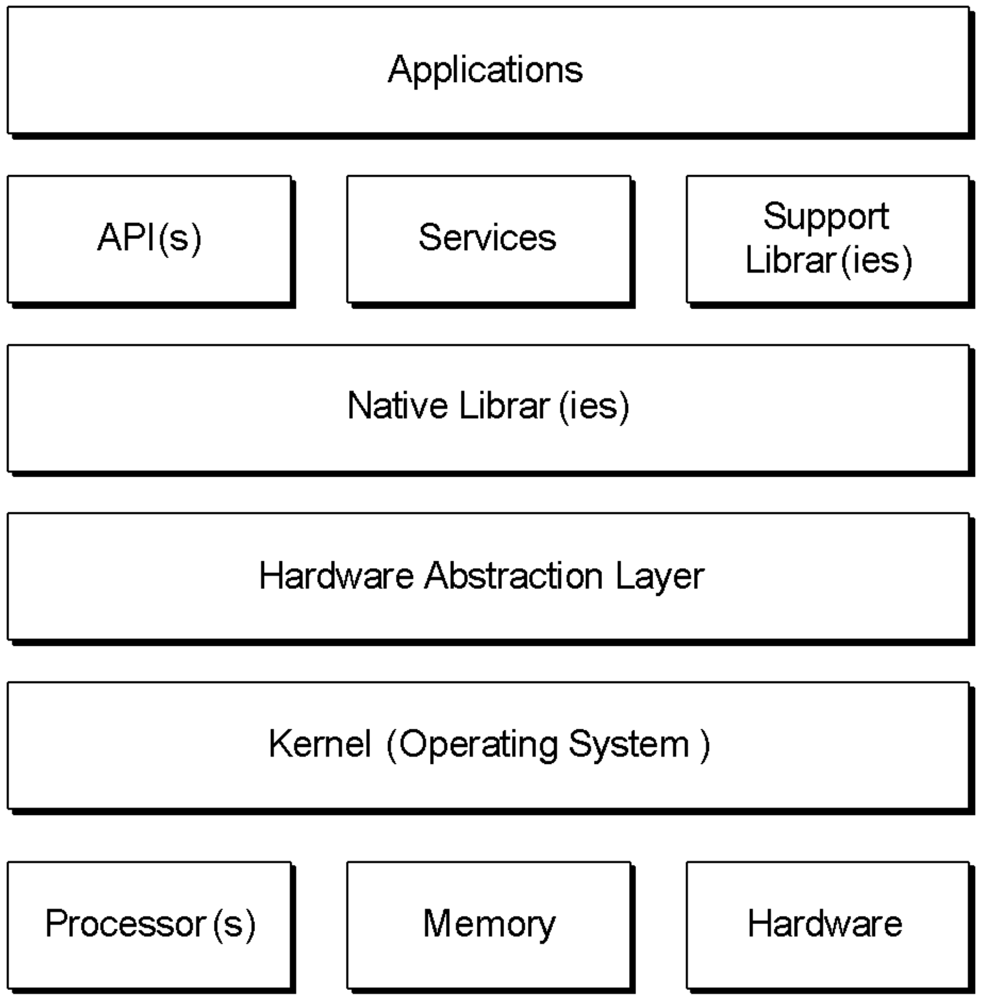

Applications
API(s)
Services
Support Librar(ies)
Native Librar (ies)
Hardware Abstraction Layer
Kernel (Operating System)
Processor(s)
Memory
Hardware

Fig. 2

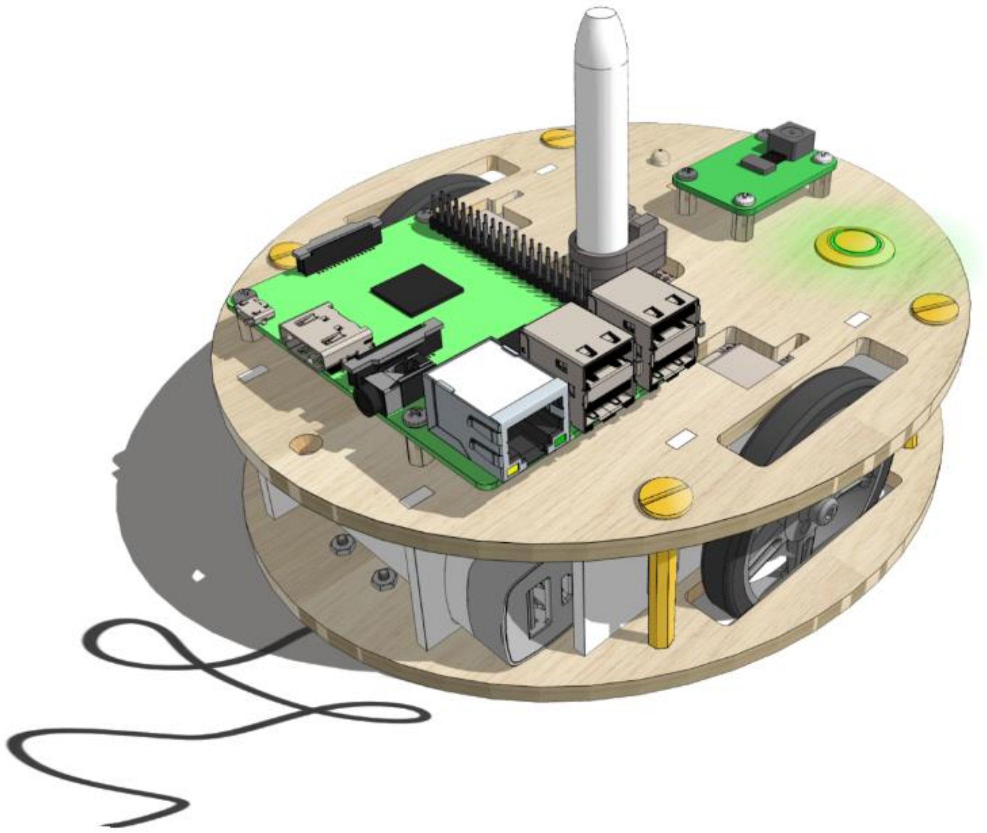


Fig. 3

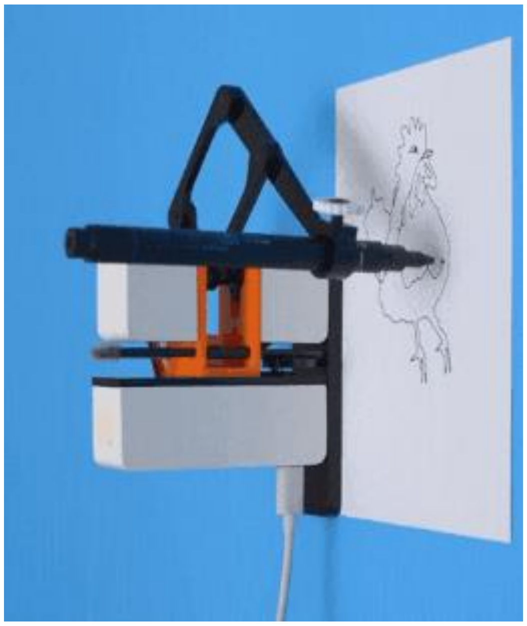


Fig. 4

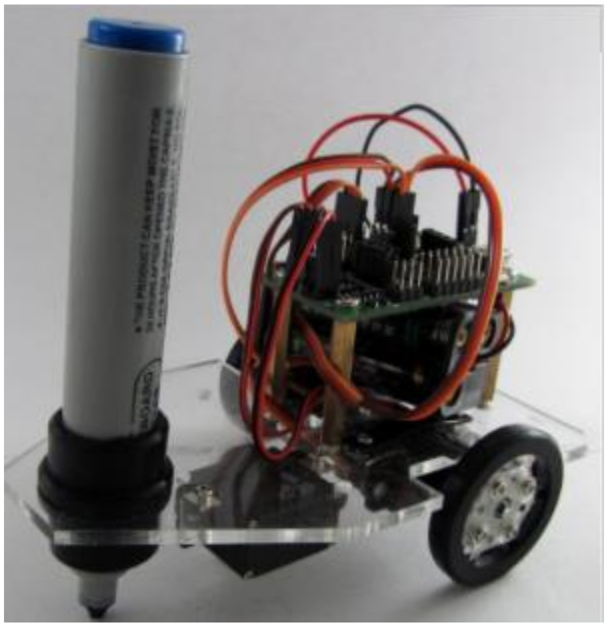


Fig. 5

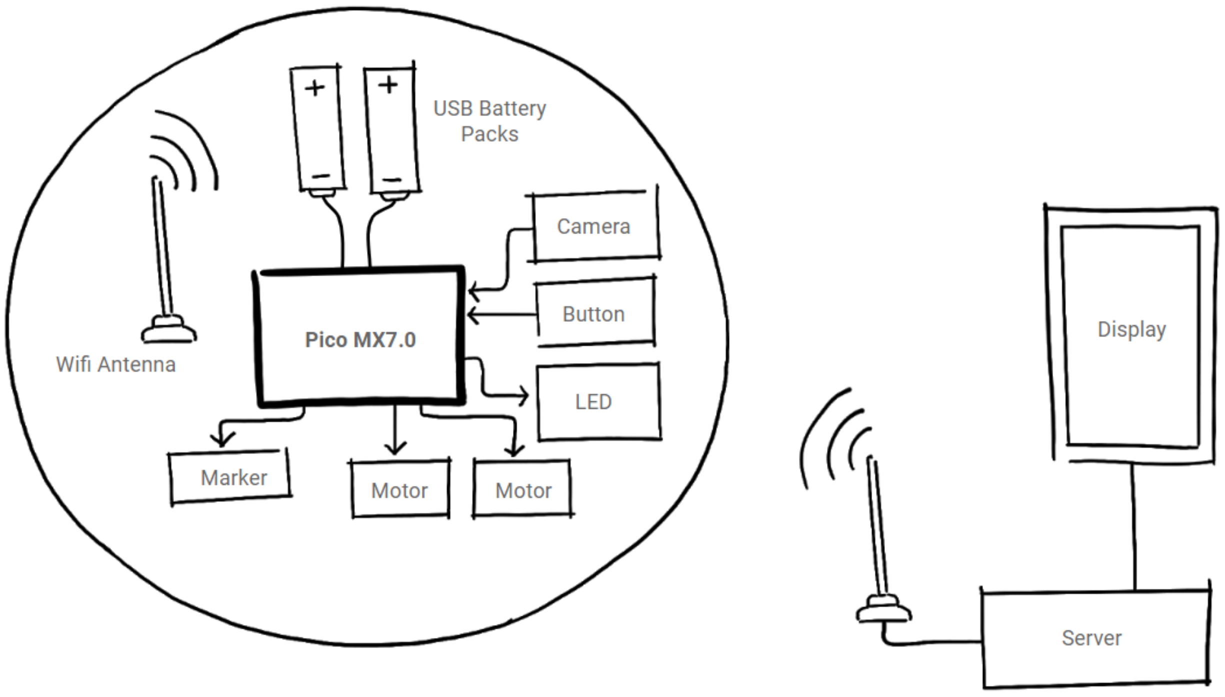

USB Battery Packs
Wifi Antenna
Camera
Button
Pico MX7.0
LED
Marker
Motor
Motor
Display
Server

Fig. 6

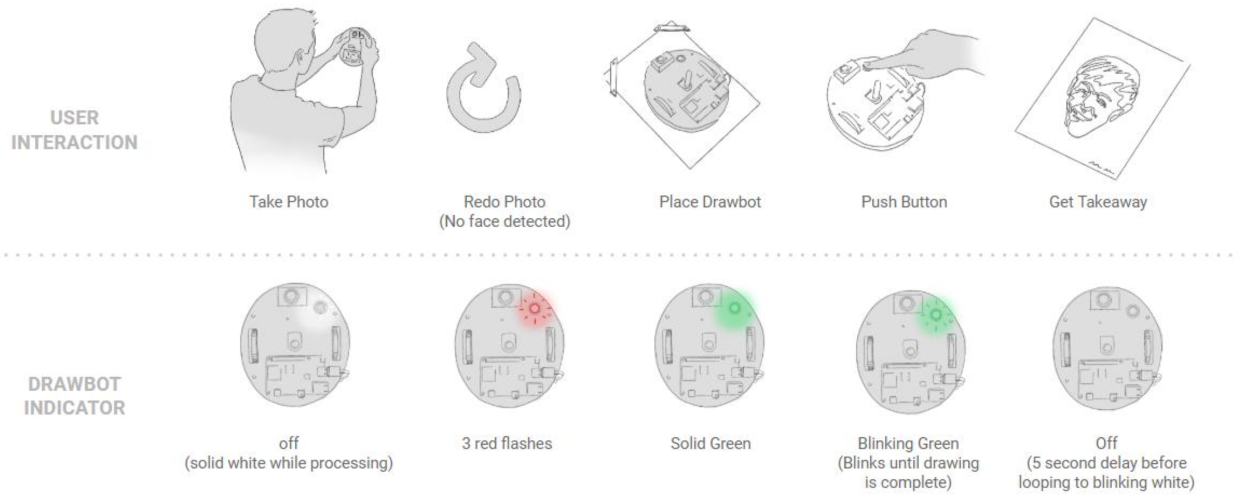

USER INTERACTION
Take Photo
Redo Photo (No face detected)
Place Drawbot
Push Button
Get Takeaway
DRAWBOT INDICATOR
off (solid white while processing)
3 red flashes
Solid Green
Blinking Green (Blinks until drawing is complete)
Off (5 second delay before looping to blinking white)

Fig. 7

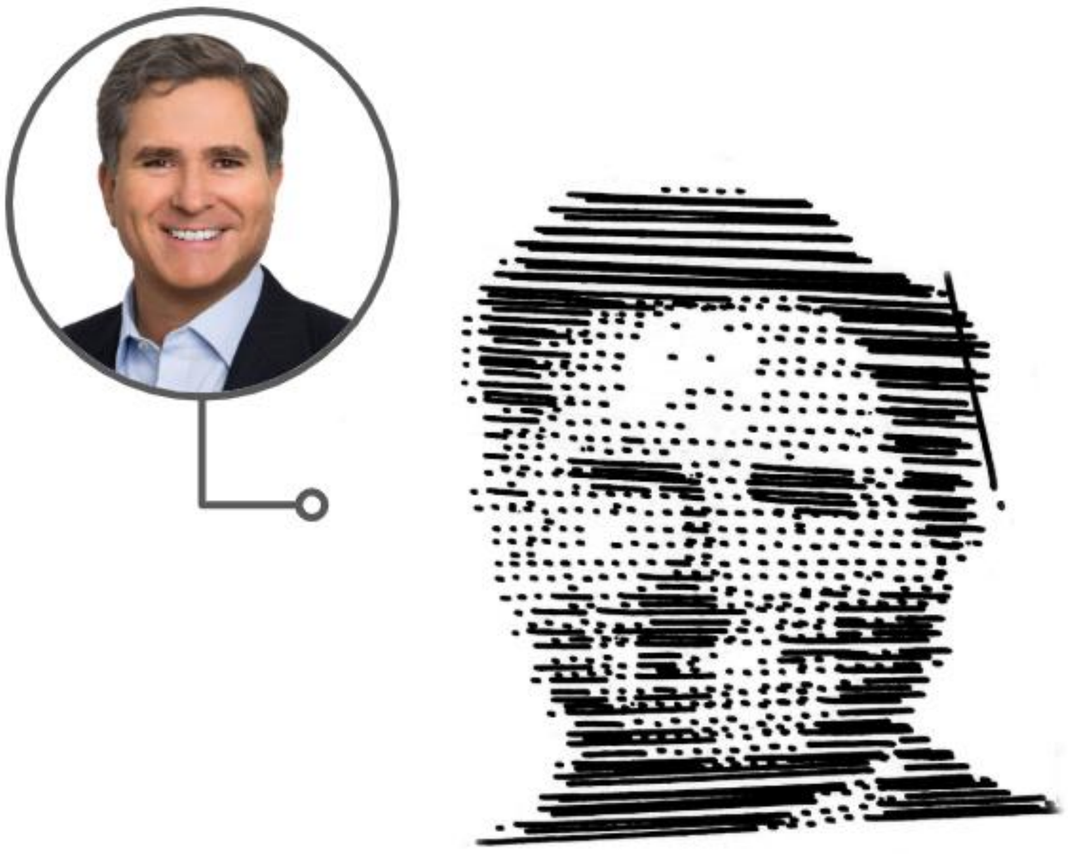


Fig. 8

Appendix A – Example Device

Bill of Materials

- (1x) Pico.iMX7Dual Starter Kit
- (2x) wheels
- (2x) wheel hubs
- (2x) ball casters
- (2x) batteries
- (1x) servo motor
- (2x) stepper motor
- (2x) stepper driver chip
- (1x) breadboard
- (1x) APA102 color LED
- (1x) breakout for LED
- (1x) ribbon cable
- (1x) button
- (1x) 10k resistor
- (2x) right-angle USB-C cable
- (1x) brush tip marker
- (12" x 6") acrylic 0.125" thick
- (14" x 8") plywood 0.196" thick
- Nuts and bolts (see attachment "Hardware BOM")

Estimated Parts Cost: $160 (+ an Android Things starter kit)

Estimated Build Time: 5 hours... or a rainy afternoon.

Wiring schematics and file attachments are located at the bottom of this page.

Overview

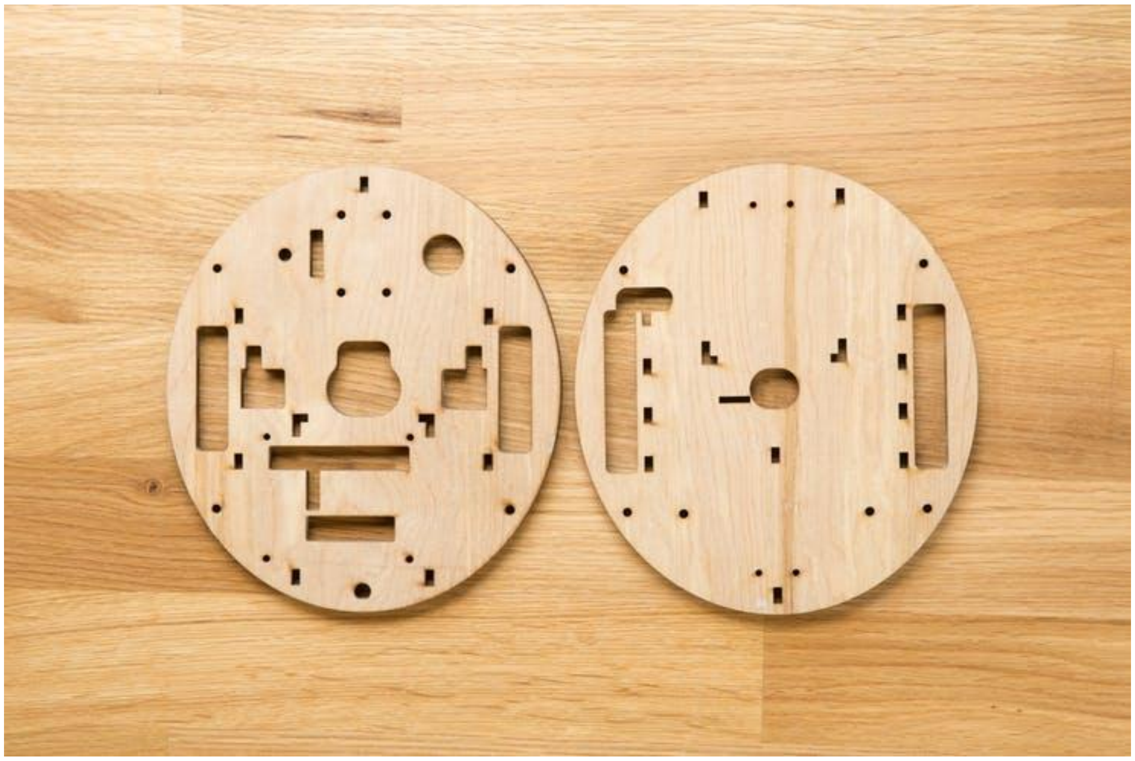
Our DrawBot is built like a sandwich. The top and bottom plates are cut from plywood, and everything else sits inside. Each wheel is driven by it's own stepper motor. Two ball casters under the front and back of the robot keep it balanced.

The DrawBot draws using a brush tip marker. A servo motor raises and lowers the marker to draw lines of different thickness on the paper. The DrawBot can turn in place around the marker, and can drive forward in straight lines, just like the original Logo Turtle programming game! Using only basic commands (TURN and FORWARD), we're able to generate complex drawings.

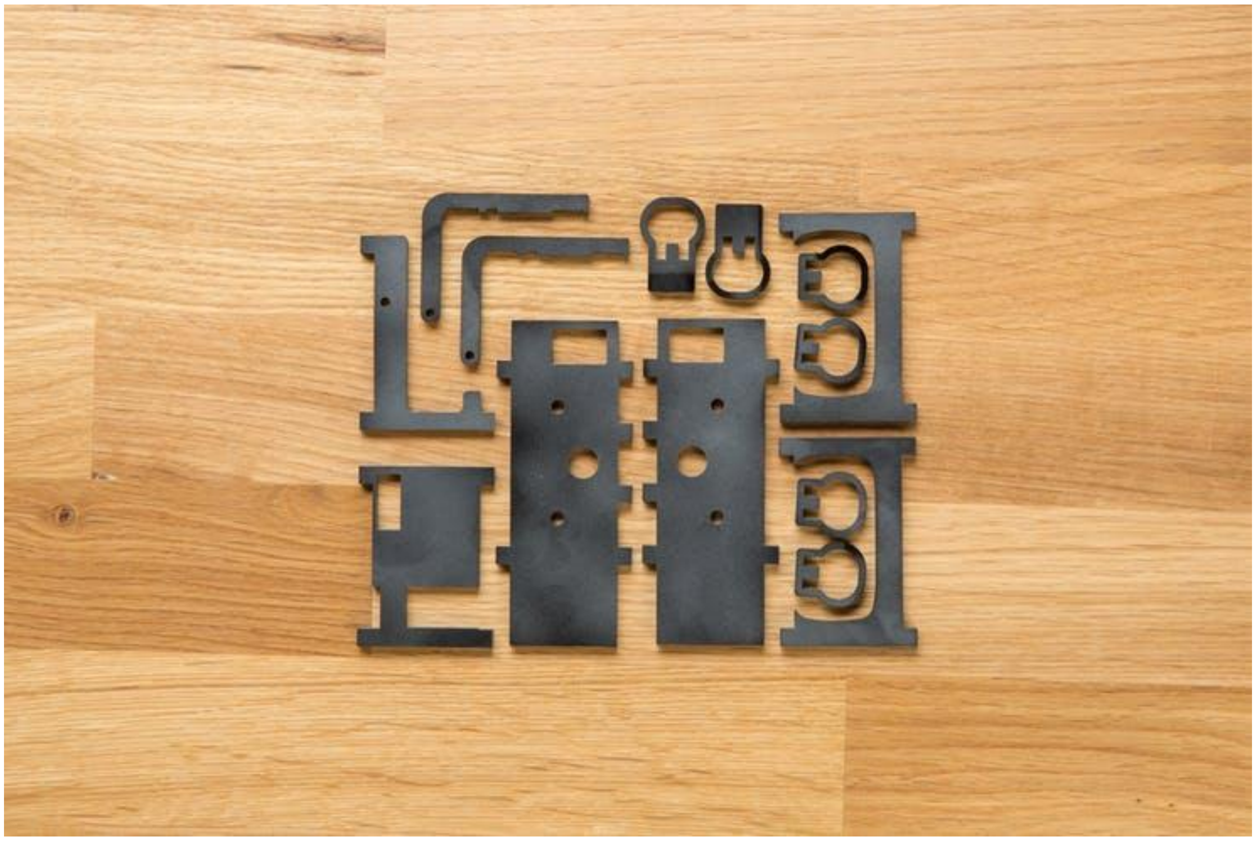
We use two USB power banks to power our DrawBot. One battery powers the Pico.iMX7 board, which runs Android Things. The other battery powers the motors.

Assembly

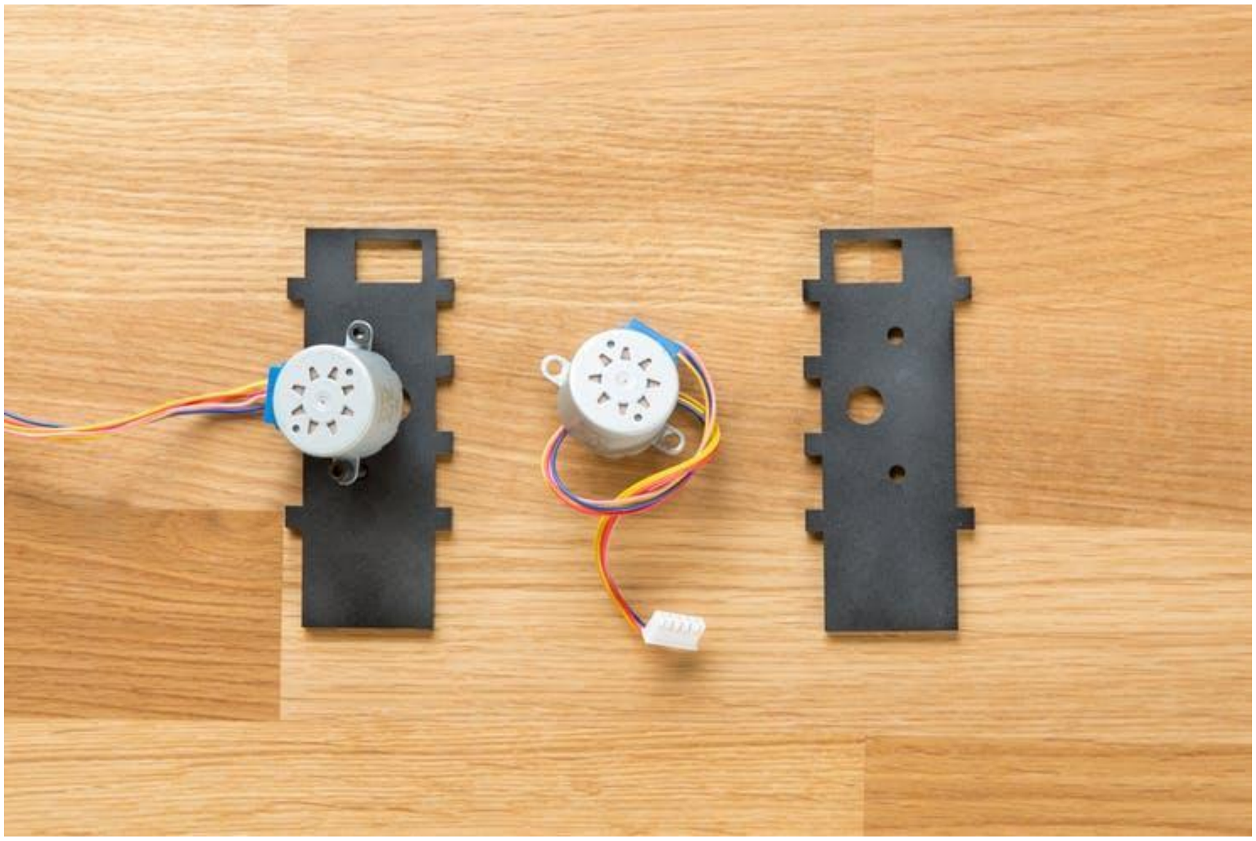
1. Cut the wood plates. We used a laser cutter to cut ours from the attached files. If you don't have a laser cutter handy, check out Ponoko.com - we recommend the 0.204" birch veneer core.



2. Cut the acrylic pieces. We used a laser cutter to cut ours from the attached files. If you're using Ponoko.com, we recommend the 0.125" black delrin.

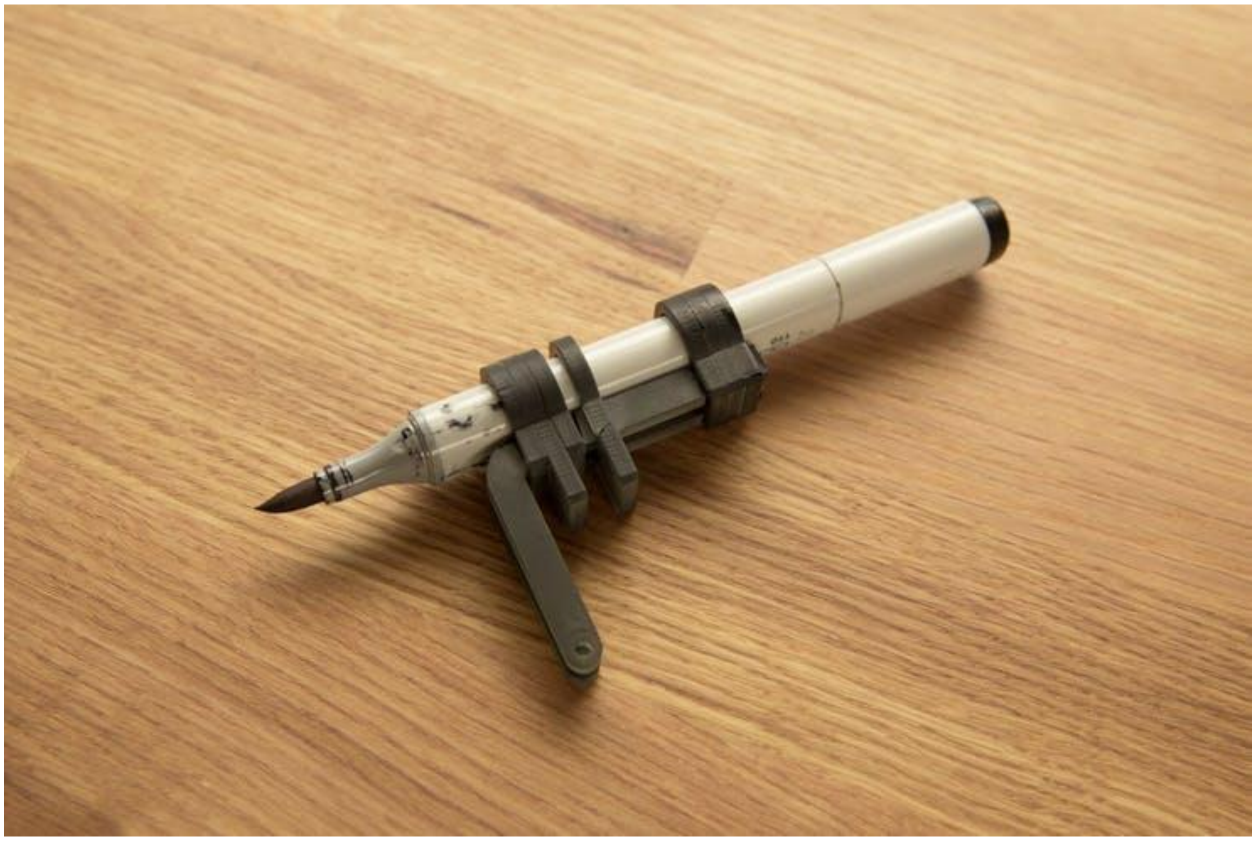


3. Assemble the motor mounts as shown. Use two #6-32 screws and nuts to hold the motor in place. The second assembly should mirror the first.

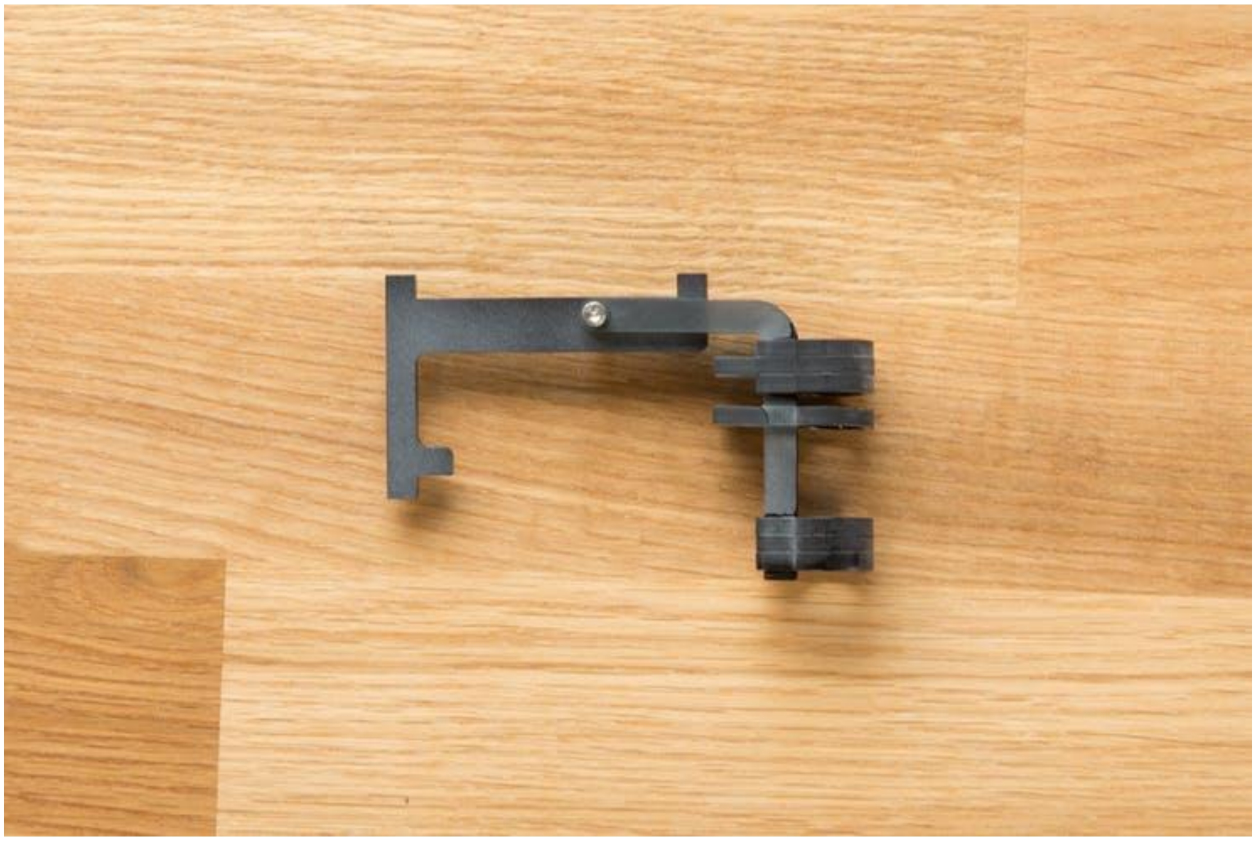


4. Mount the wheel hubs onto the wheels using two #4-40 screws. Slide the hub onto the motor shaft and secure it with a set screw.

5. Assemble the marker holder. Slot each clip into the vertical levers, then push the marker through to test fit. Glue the assembly together and check for straightness. Remove the marker while the glue dries.

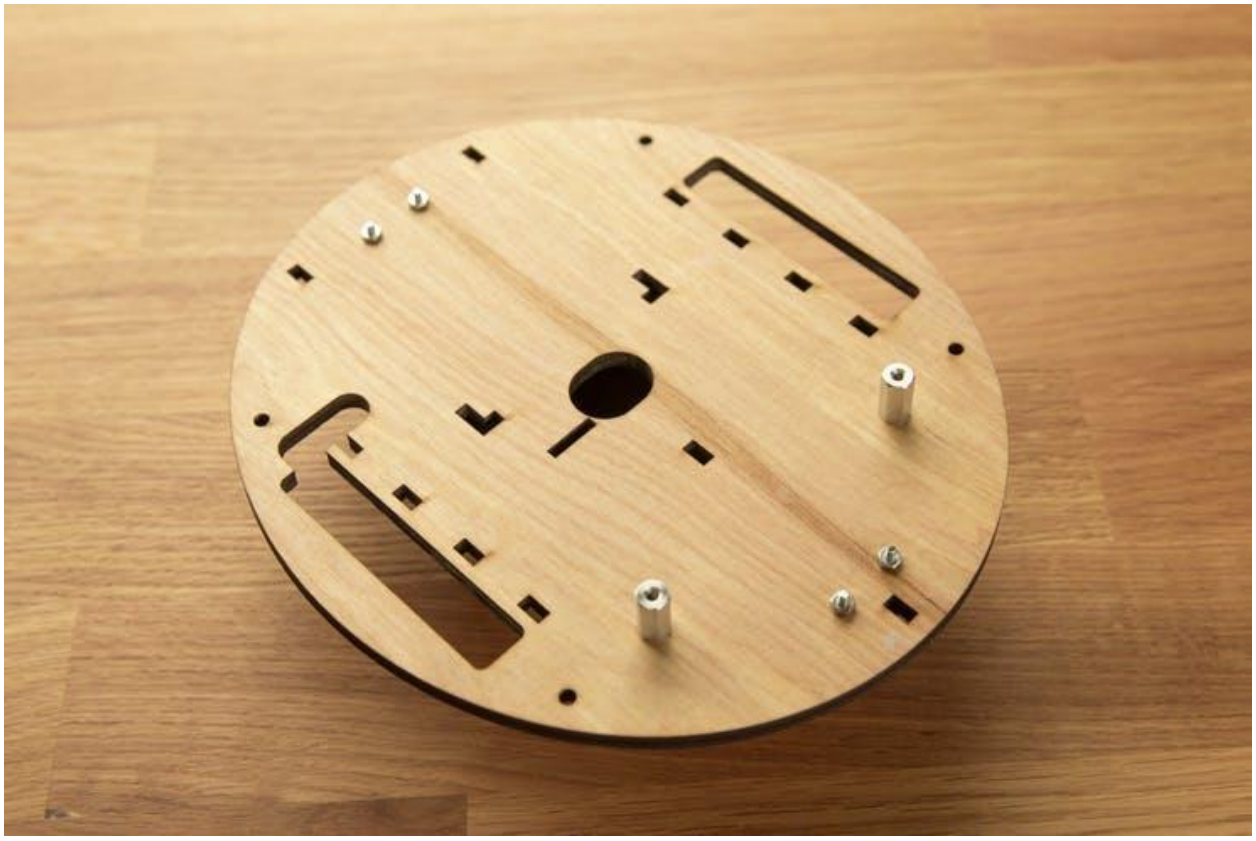


6. Bolt the marker holder to the pivot point with a #4-40 screw and nylock locking nut. Leave the nut slightly loose, allowing the pivot to rotate freely.

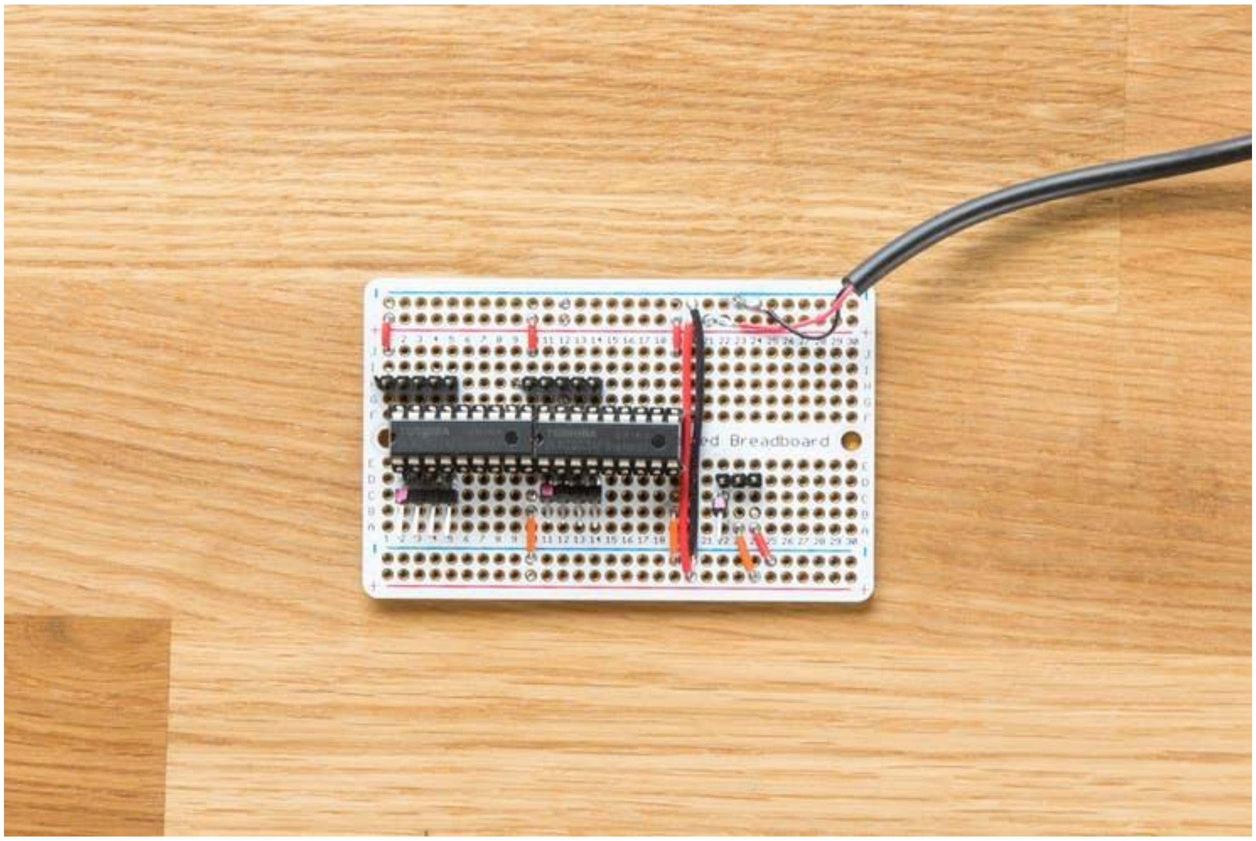


7. Bolt the two caster balls onto the bottom of the lower wood base.

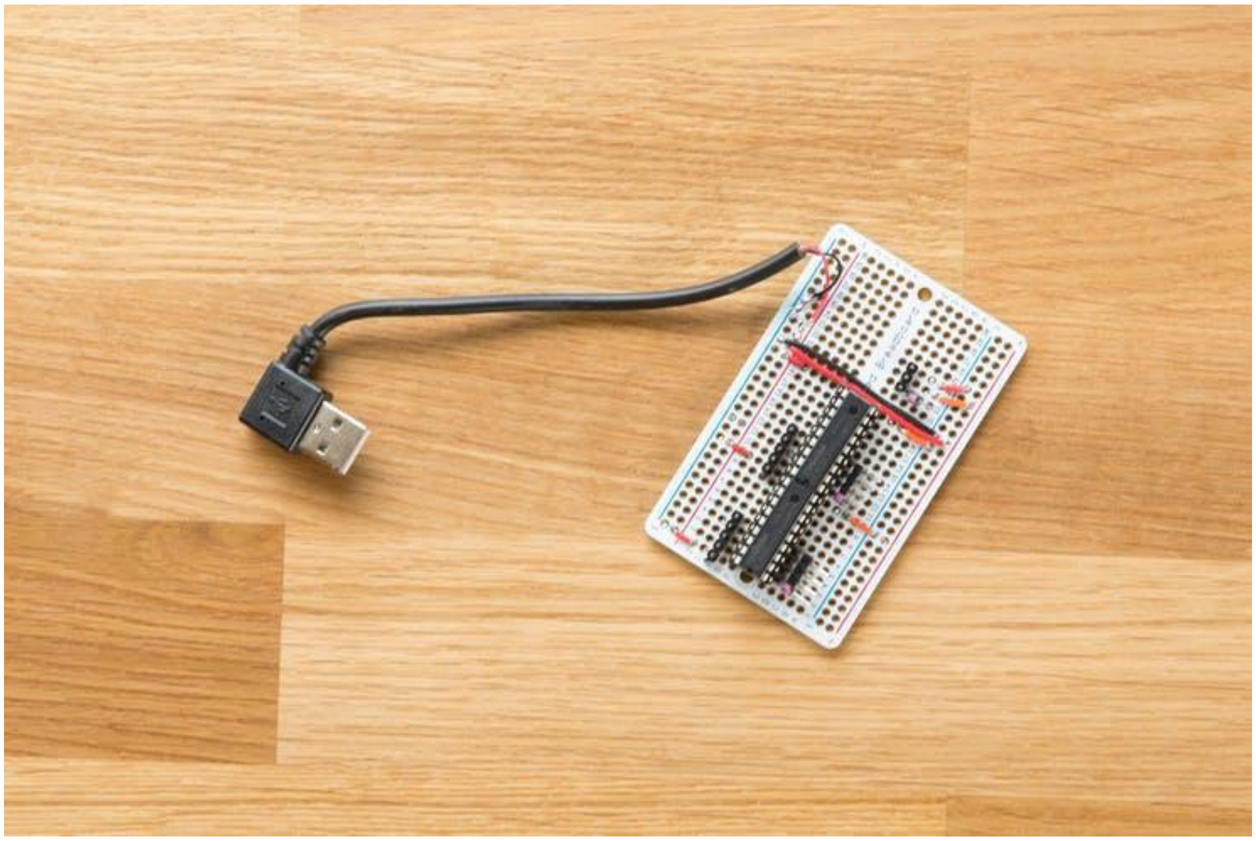
8. Mount the two #4-40 standoffs for the breadboard.



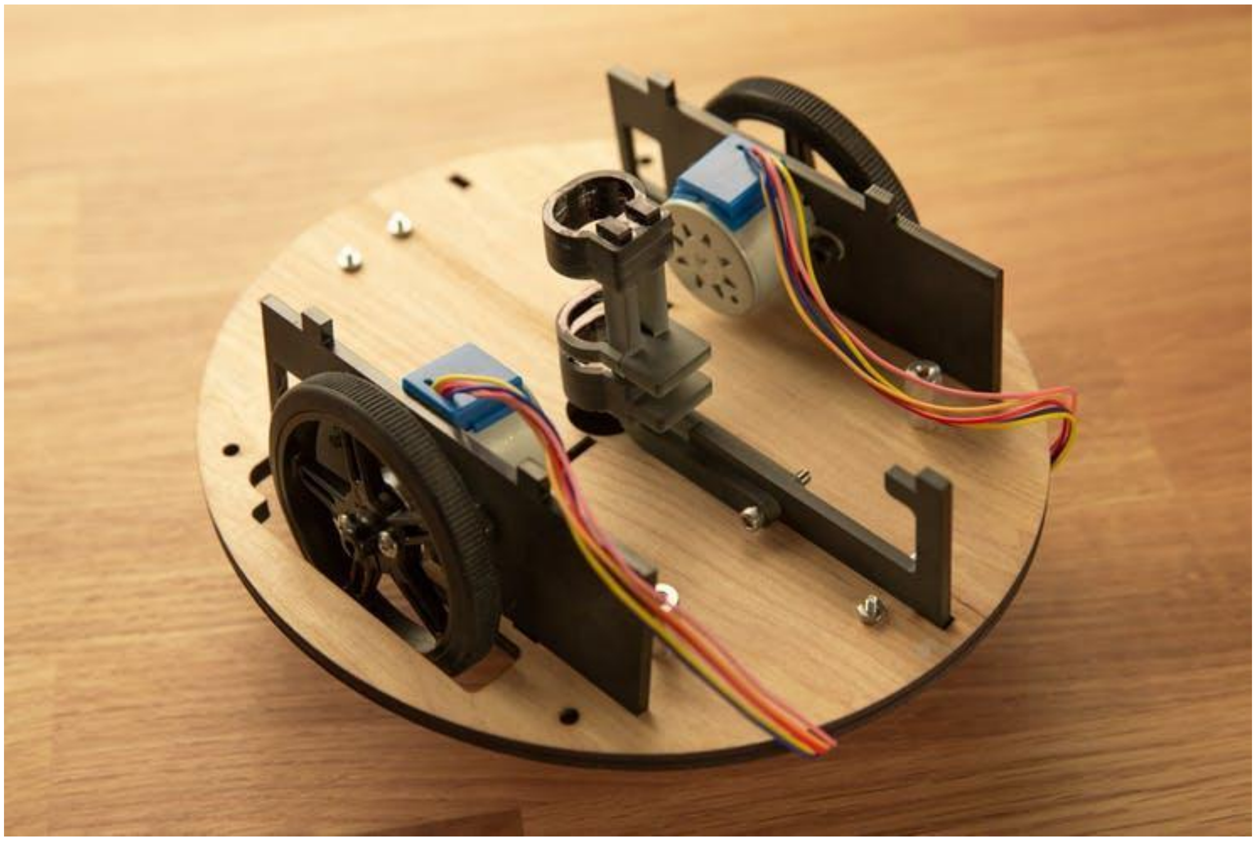
9. Assemble the electronic components following the attached breadboard schematic (at the bottom of this page). We used a prototyping board for ours, but a solderless breadboard can also work.



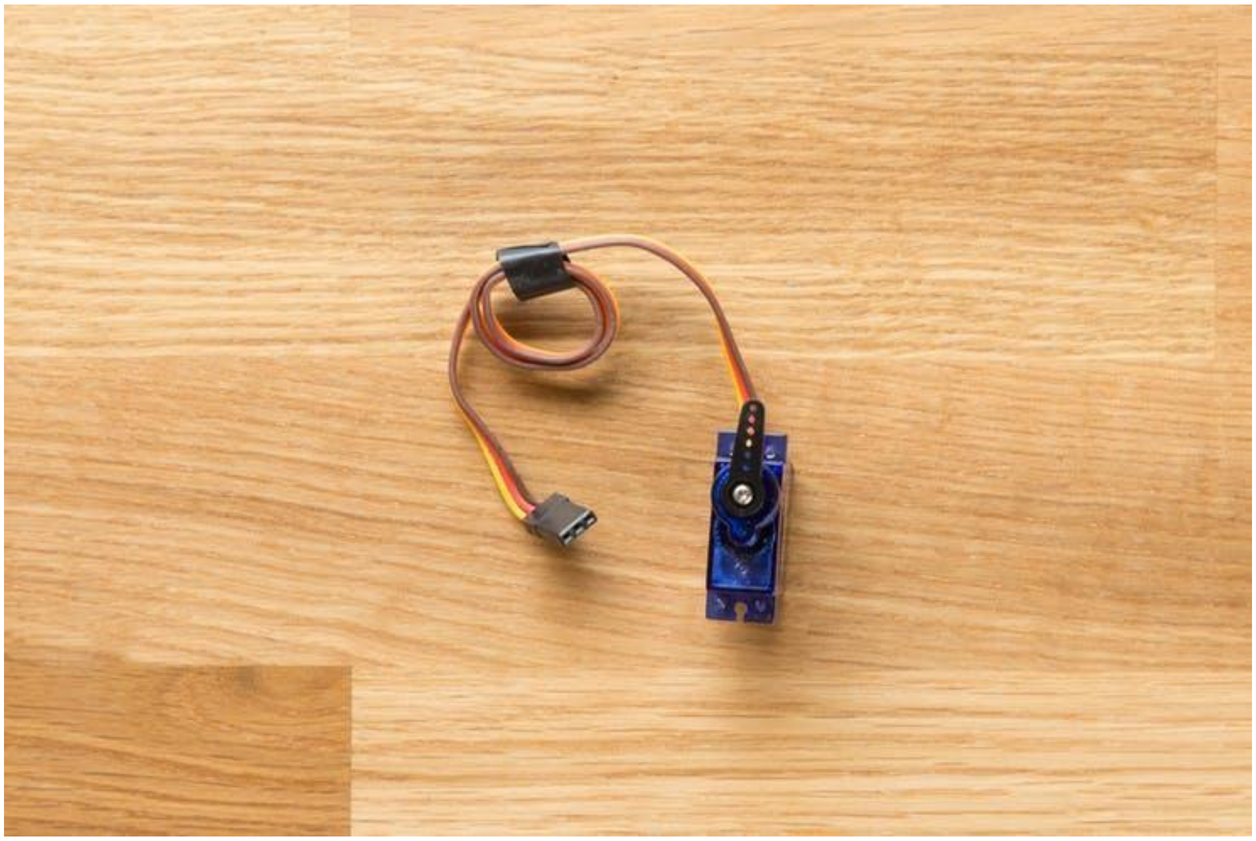
10. Cut and strip back a USB cable. Solder the red and black wires to the (+) and (-) rails of the breadboard. This will supply power to the motors.



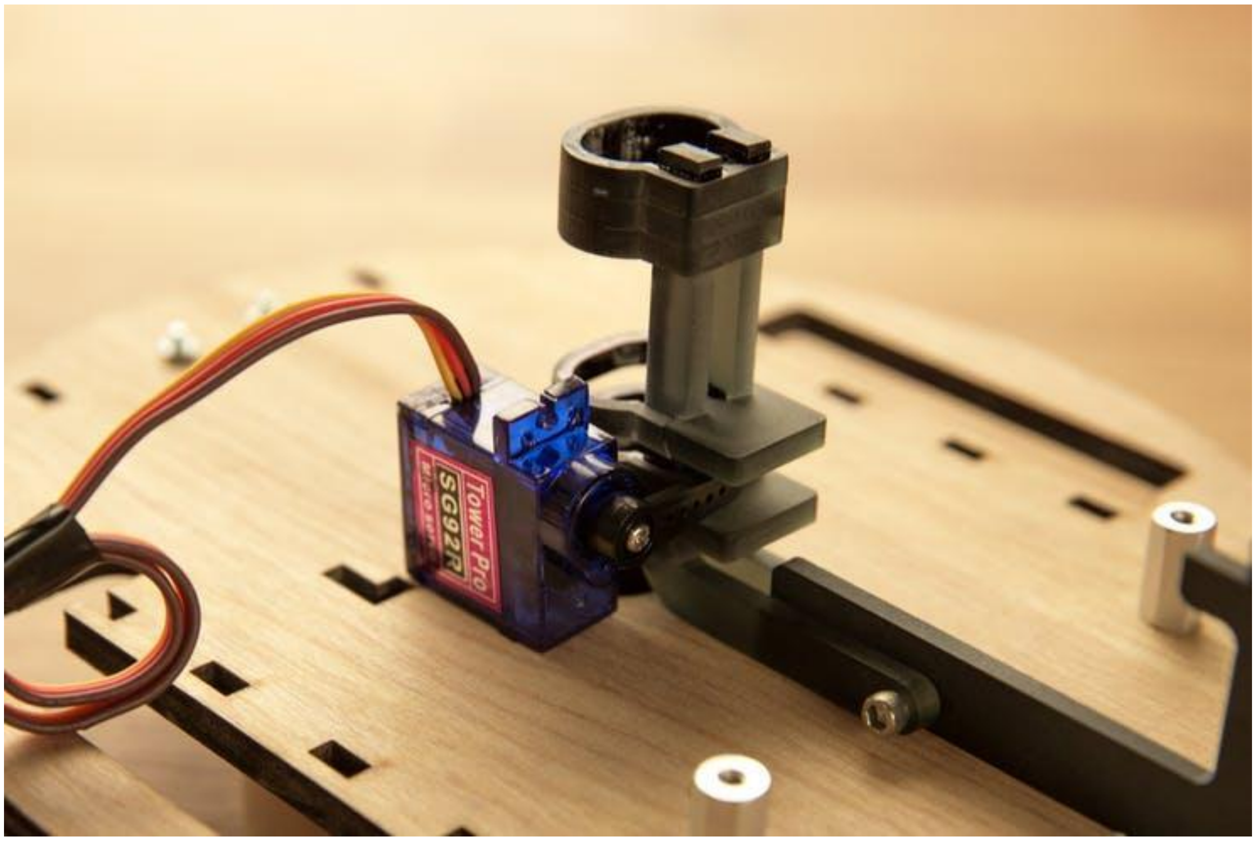
11. Slot the motor mounts and marker holder into the lower wood base. Support the wood base on a roll of tape or a coffee mug to make this step easier.



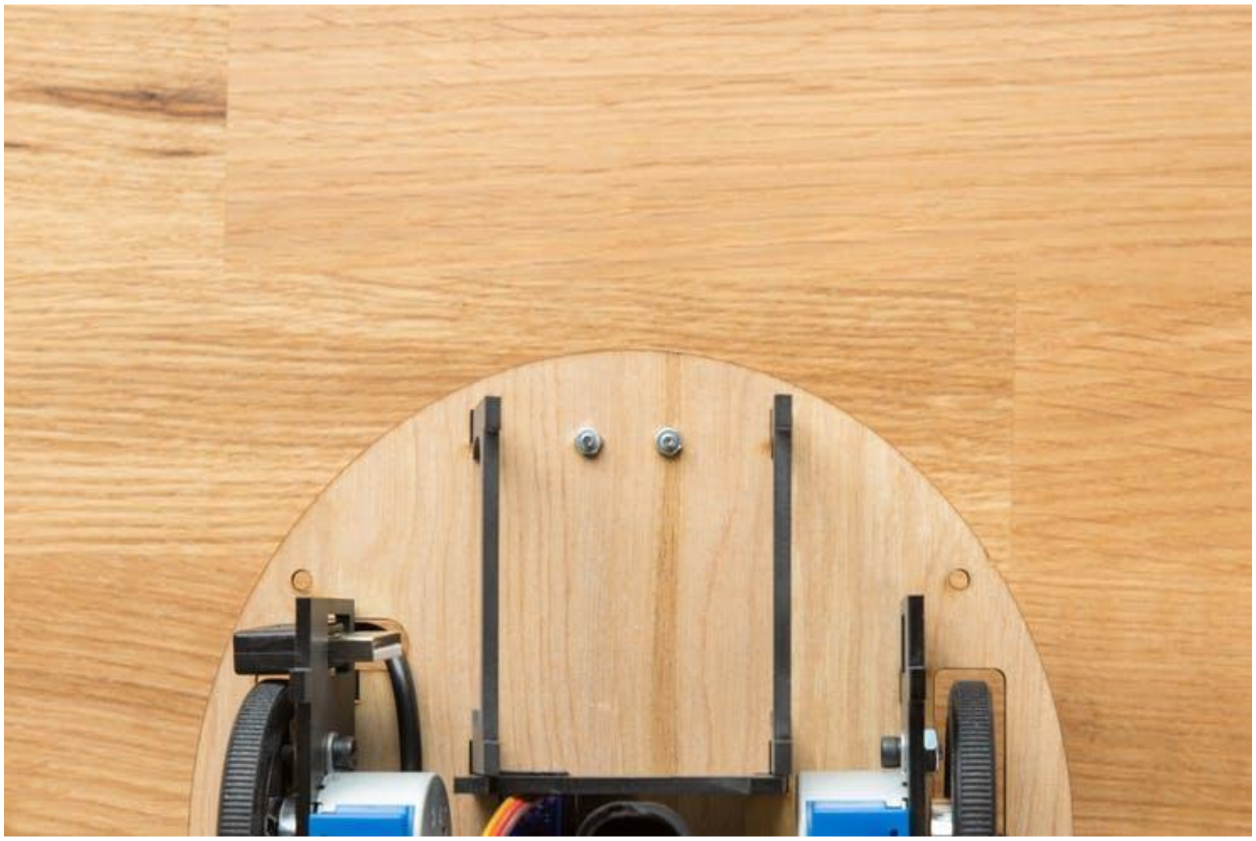
12. Turn the servo motor arm clockwise until it hits the mechanical stop. Without rotating the servo spline, remove the arm and re-mount it onto the spline pointing directly downwards (towards the longer end of the servo). This sets the "zero" position of the servo arm.



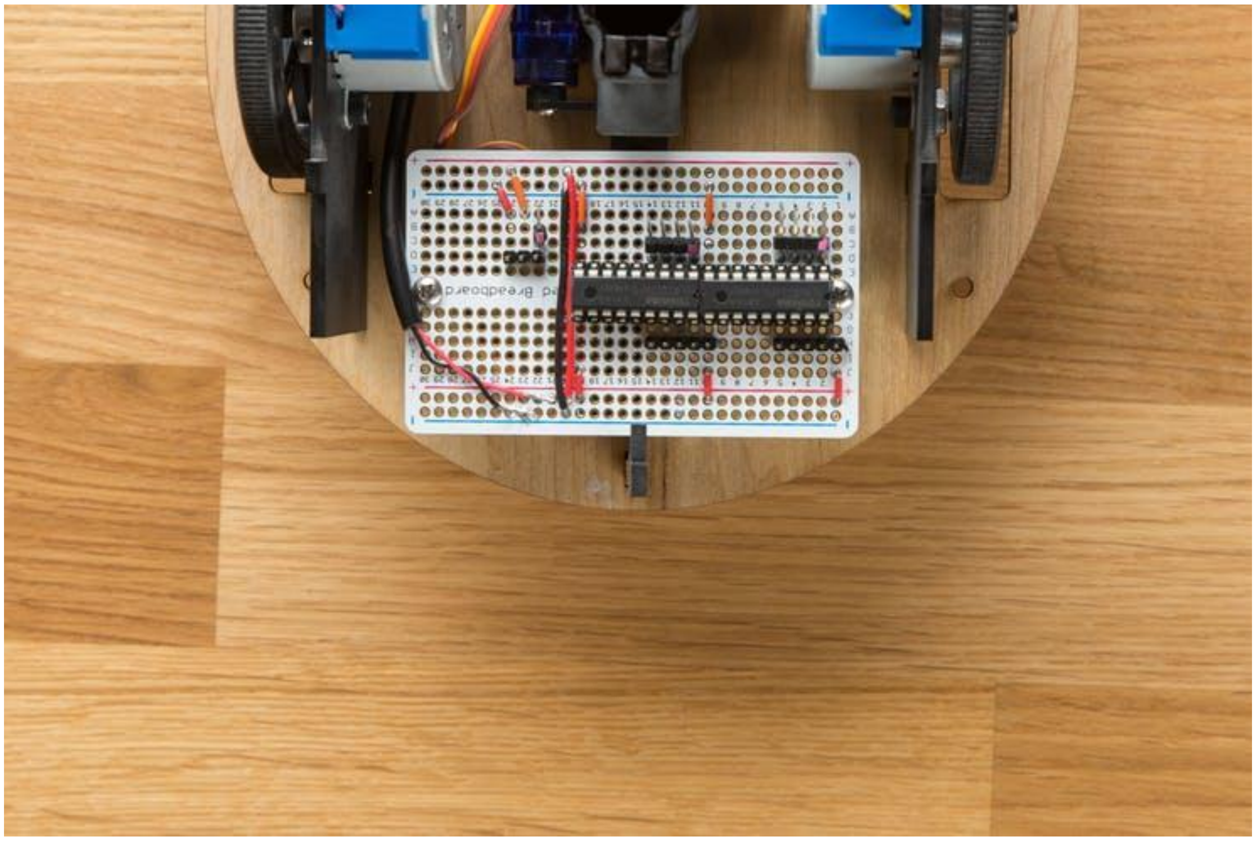
13. Slot the servo into the wood base alongside the marker holder. The servo arm should sit between the two protruding shelves on the marker holder.



14. Slot the servo holder and battery mount acrylic pieces into the wood base. The thicker end of the battery mounts should face towards the inside of the DrawBot.

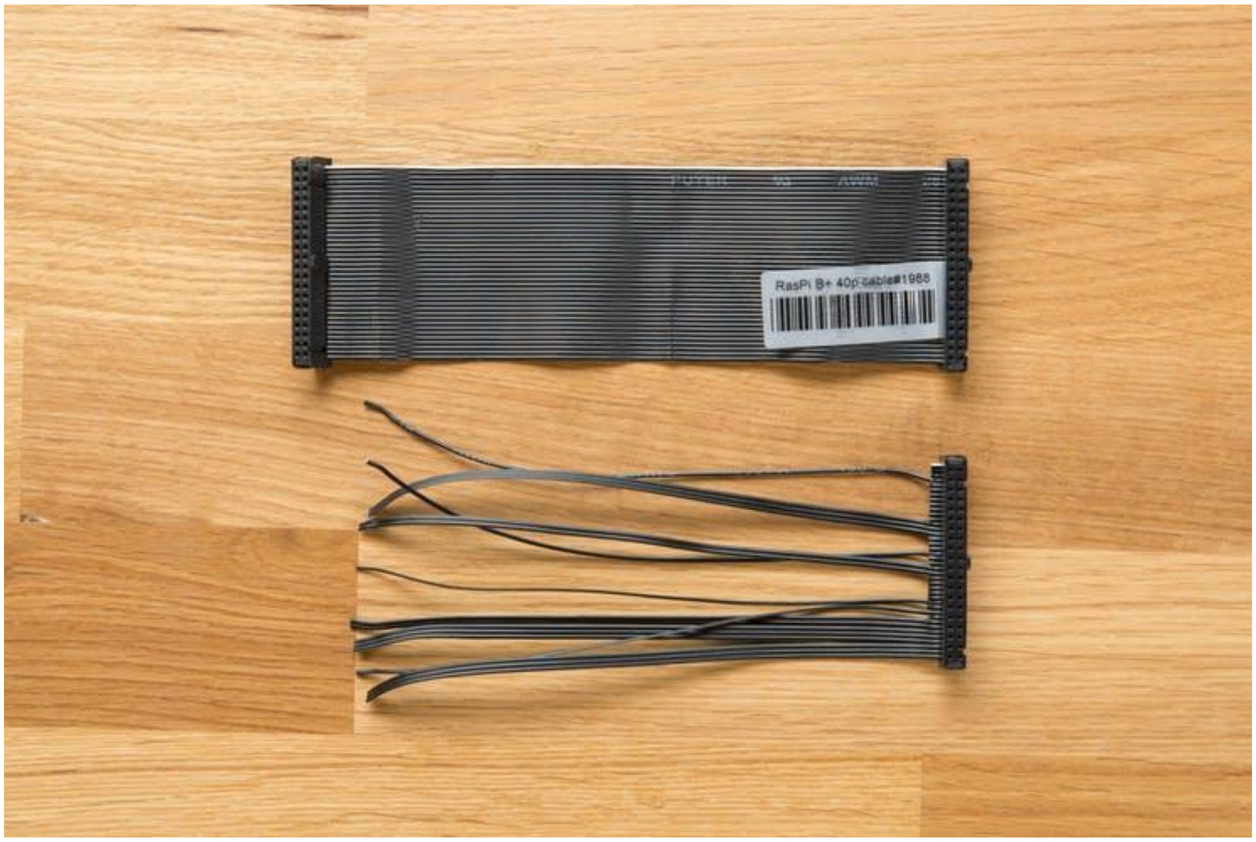


15. Mount the breadboard onto the standoffs using two #4-40 screws. Route the USB cable under the motor and under the acrylic motor mount as shown.

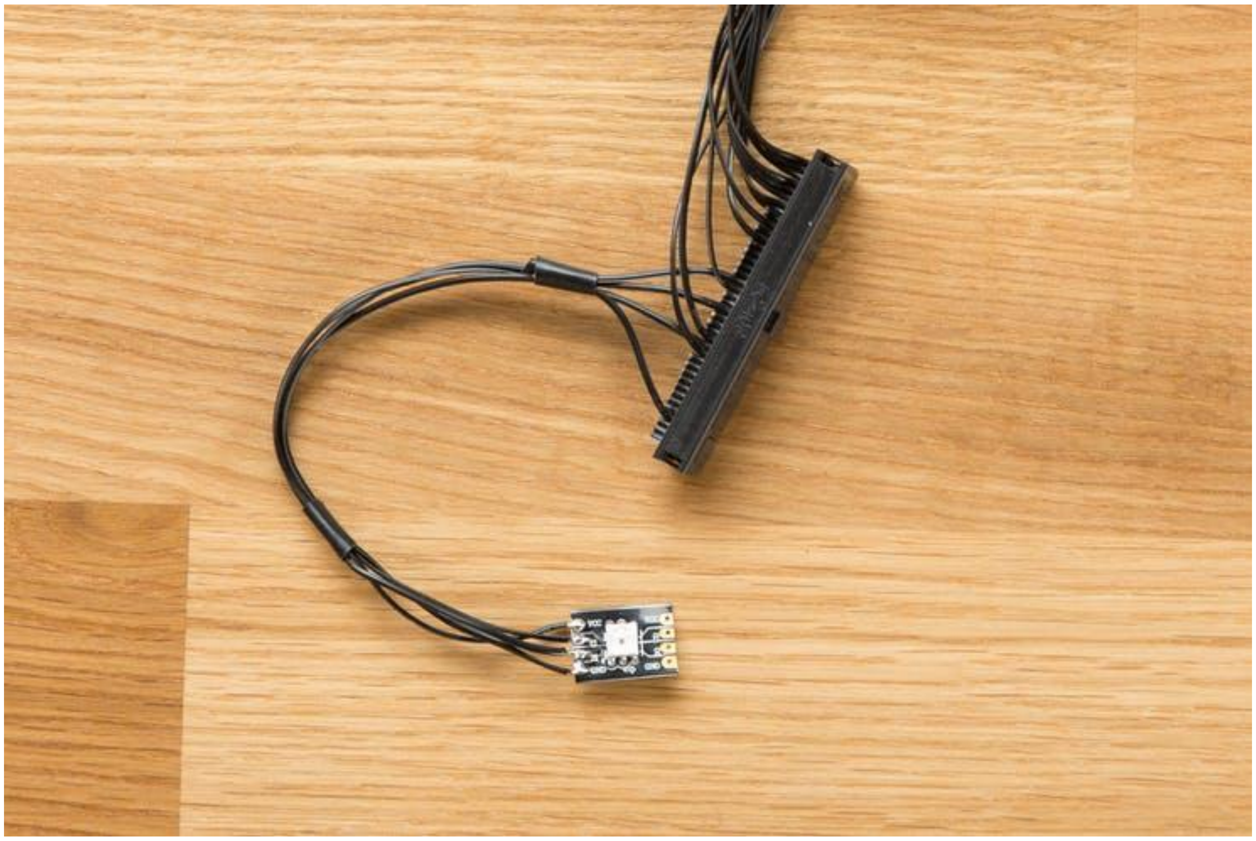


16. Coffee break!

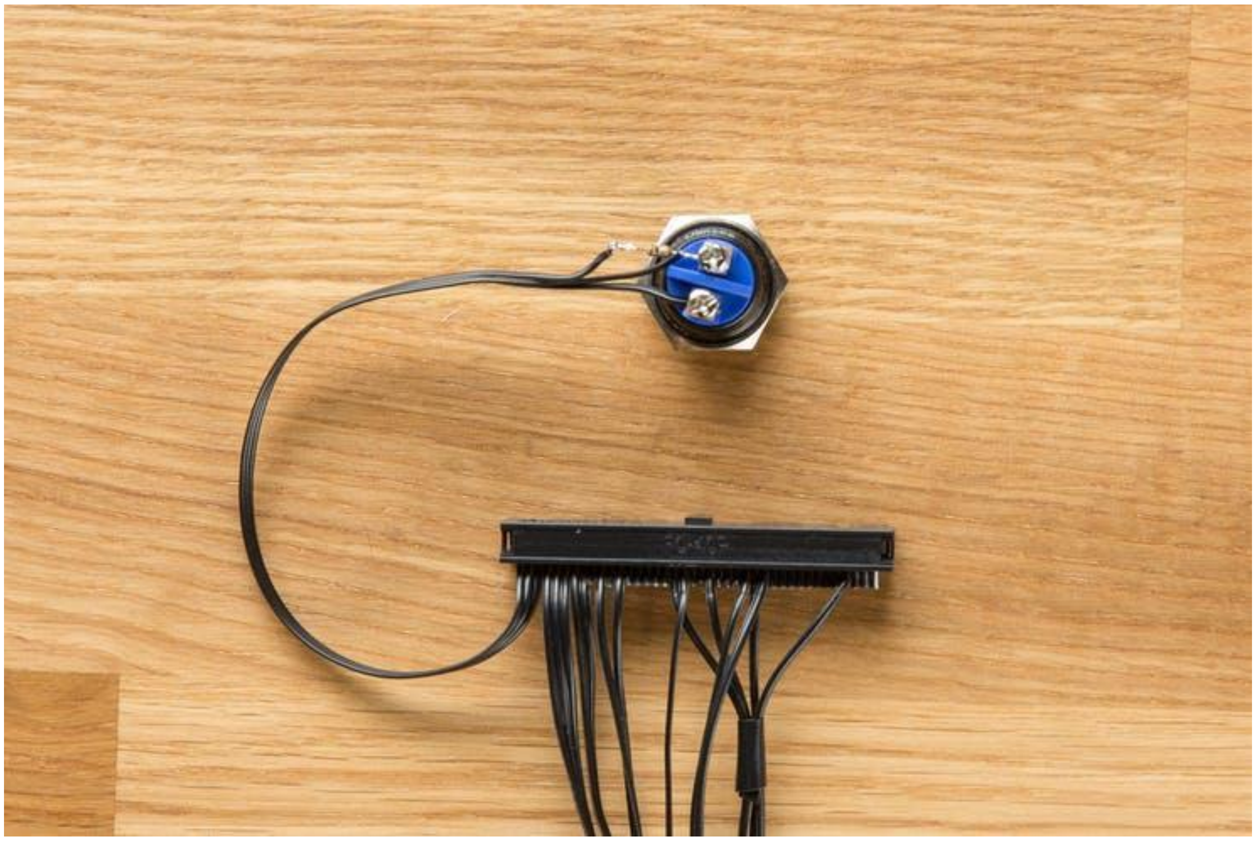
17. Cut the end off the black ribbon cable (see picture). Separate the conductors and strip according to the attached GPIO pinout chart. Peel back and cut off any unused conductors. The white wire corresponds to pin 1.



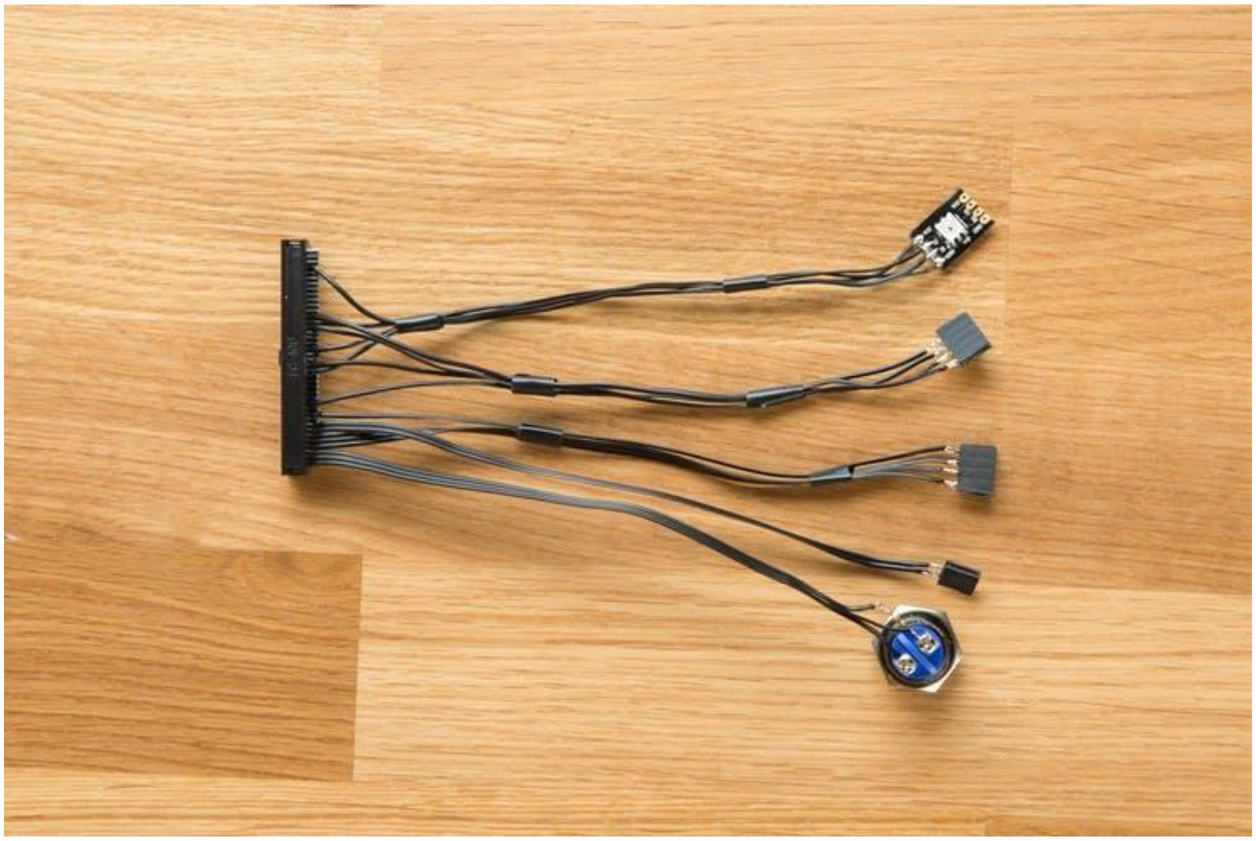
18. Solder the DotStar LED to the ribbon cable as shown. Refer to the attached GPIO pinout chart for pin numbers.



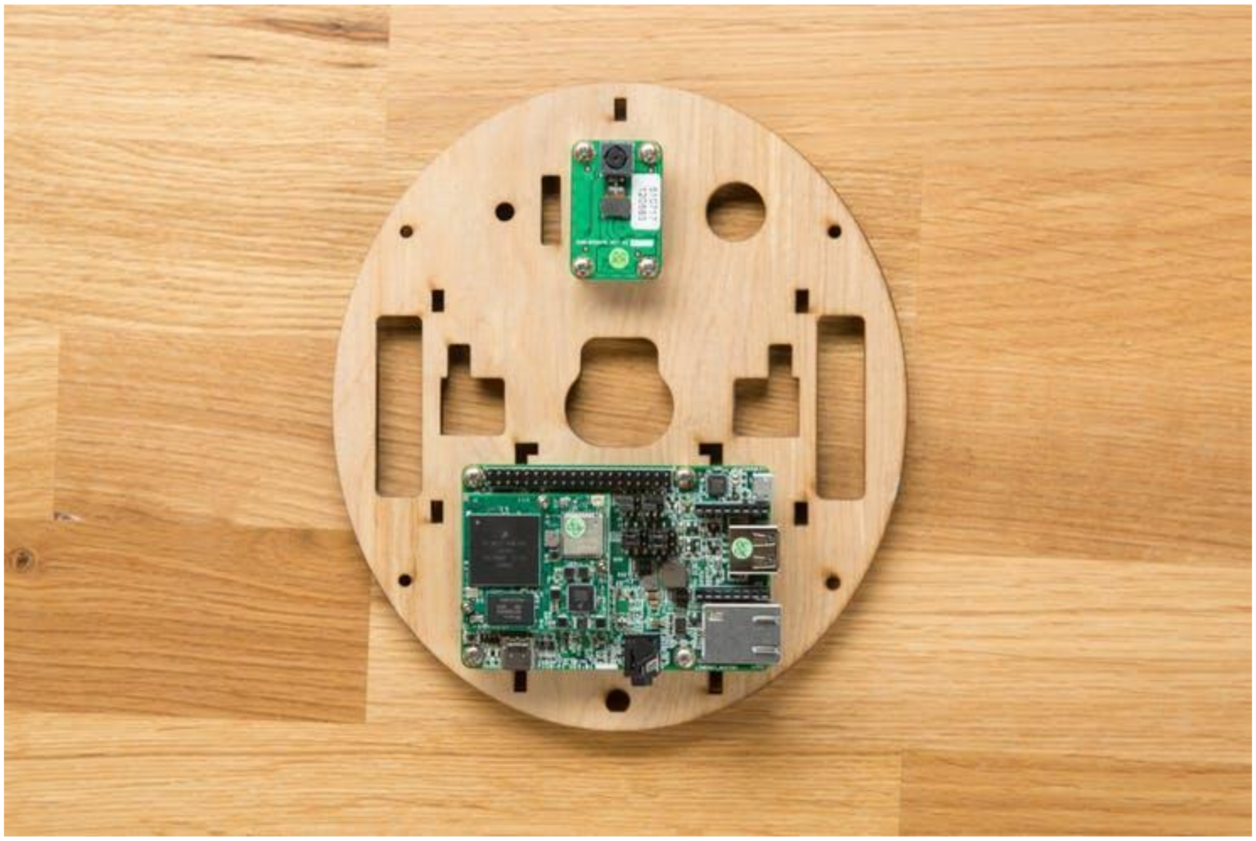
19. Solder the button to the ribbon cable as shown. Refer to the attached chart for pin numbers. Use a 10k pull-up resistor.



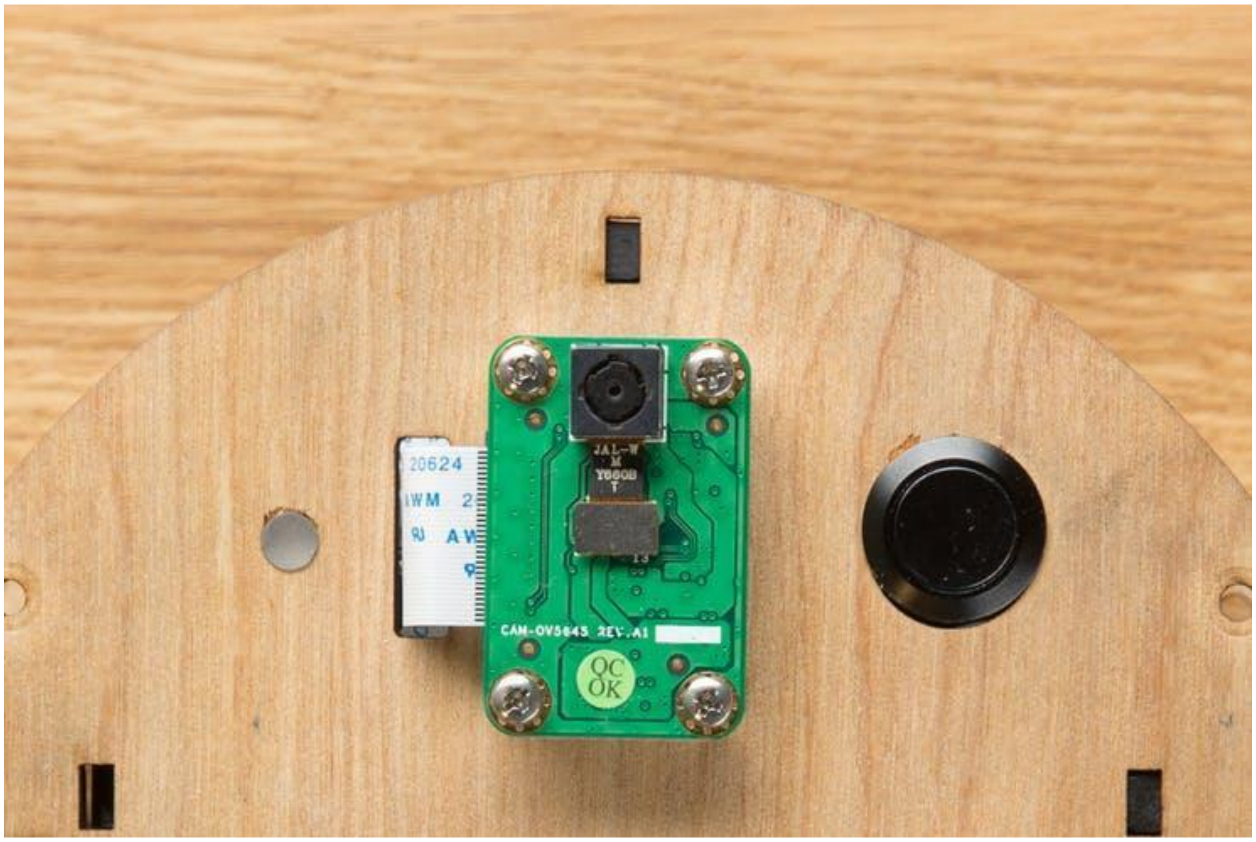
20. Connect the stepper motor and servo motor wires to the breadboard. We used breakaway headers and 0.1" sockets to make ours easily removable.



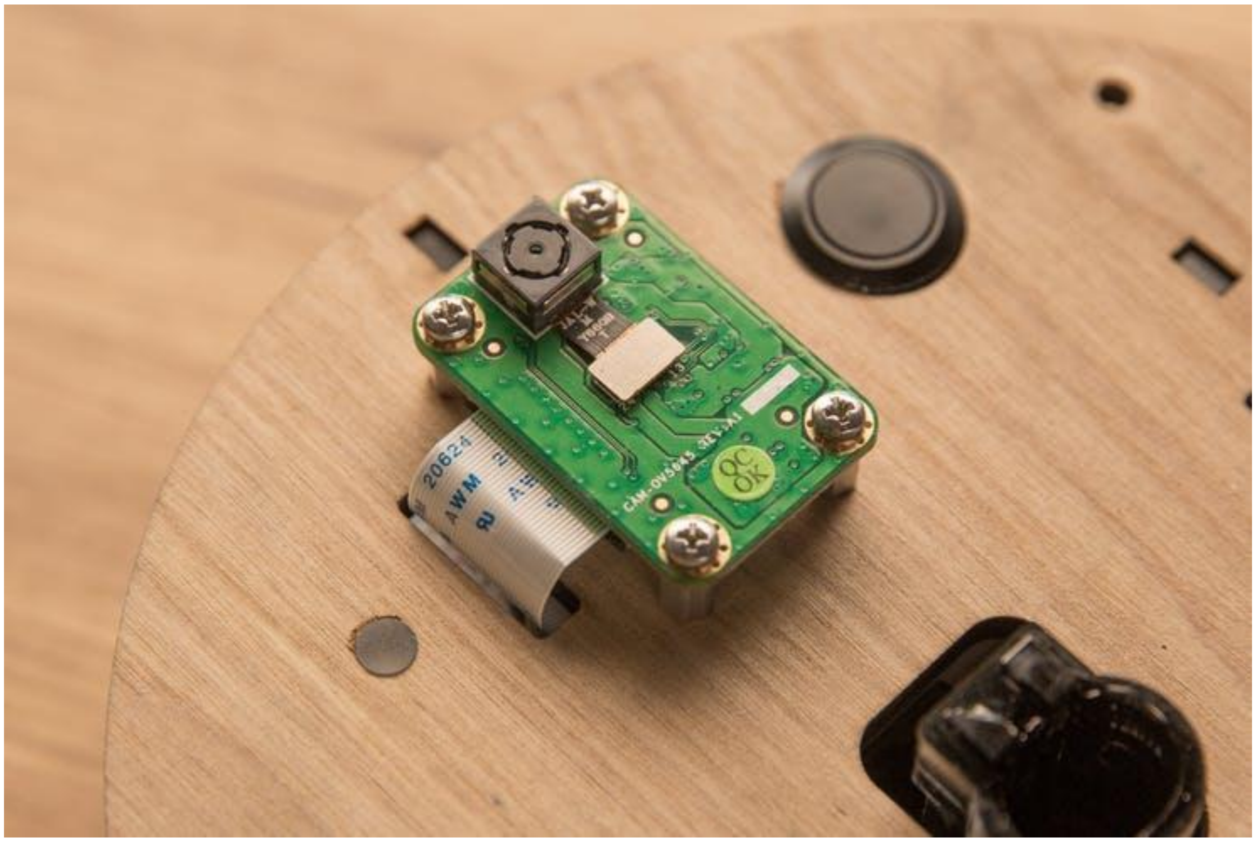
21. Mount the Pico Pro and the camera module to the upper wood deck using the M2.5 standoffs and screws.



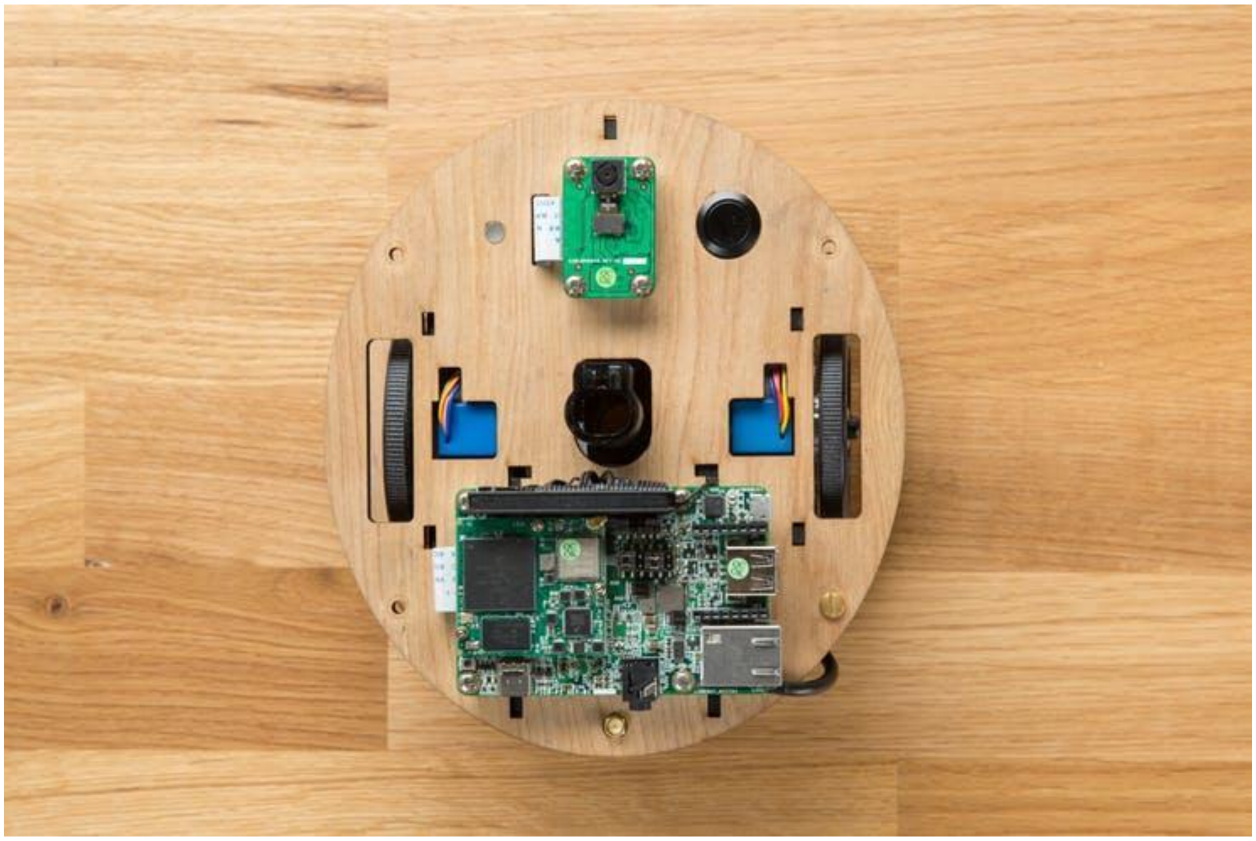
22. Mount the button and LED to the upper wood deck. We used hot-melt glue for the LED.



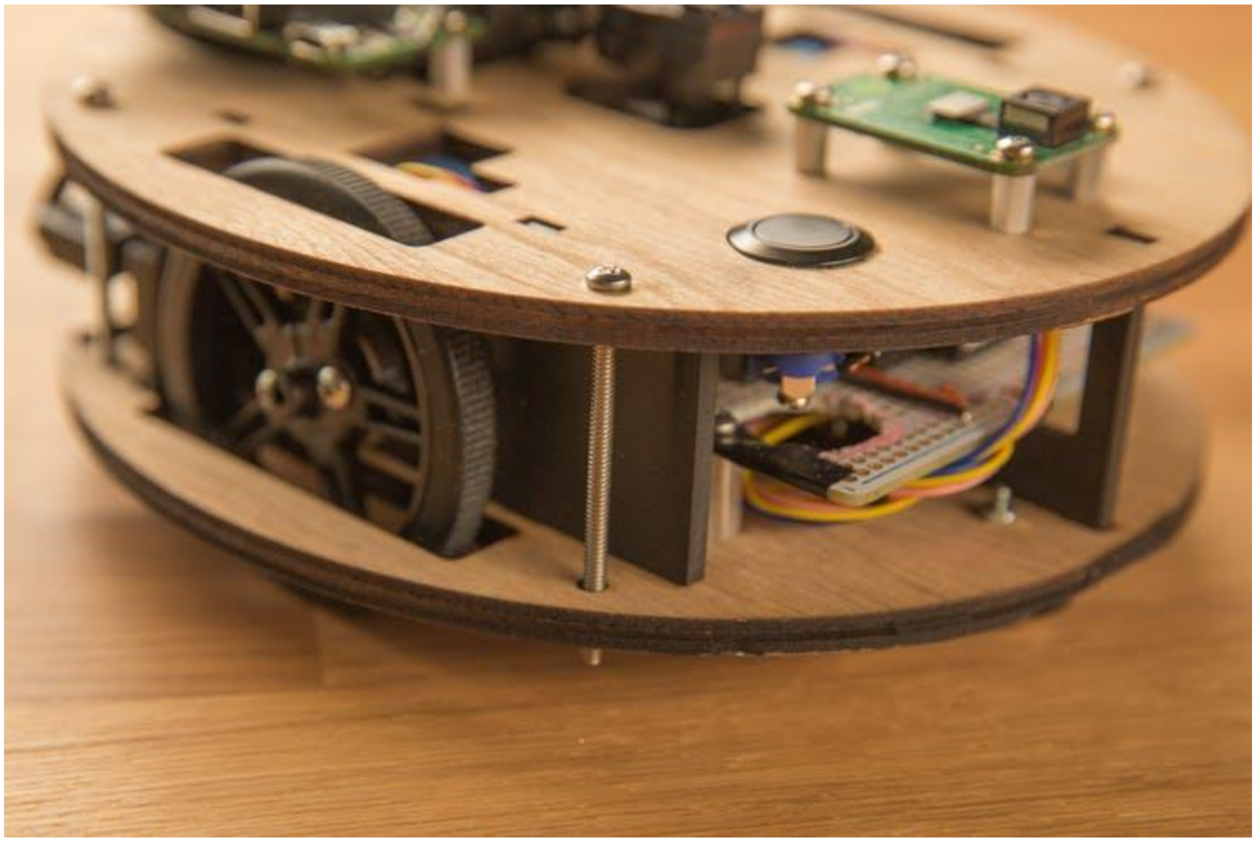
23. Thread the camera cable through the upper wood deck. Plug one end into the camera, and the other end into the bottom connector on the Pico Pro board. Thread the end of the black GPIO ribbon cable up through the slot under the Pico Pro, and connect it to the GPIO header on the Pico Pro.



24. Slot the upper wood deck onto the acrylic tabs. Route the wiring internally down each side, between the marker assembly and the stepper motors.

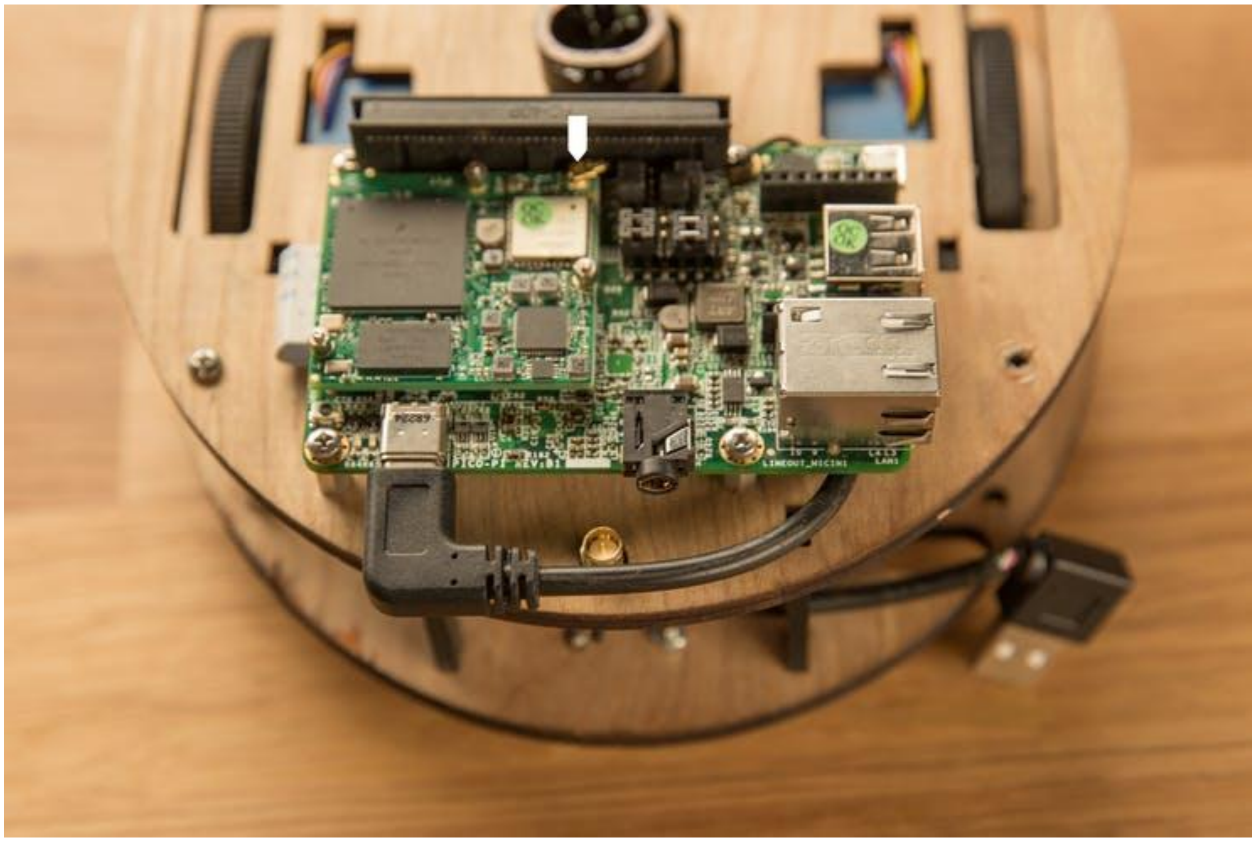


25. Fasten the upper and lower decks together using four long #4-40 screws.

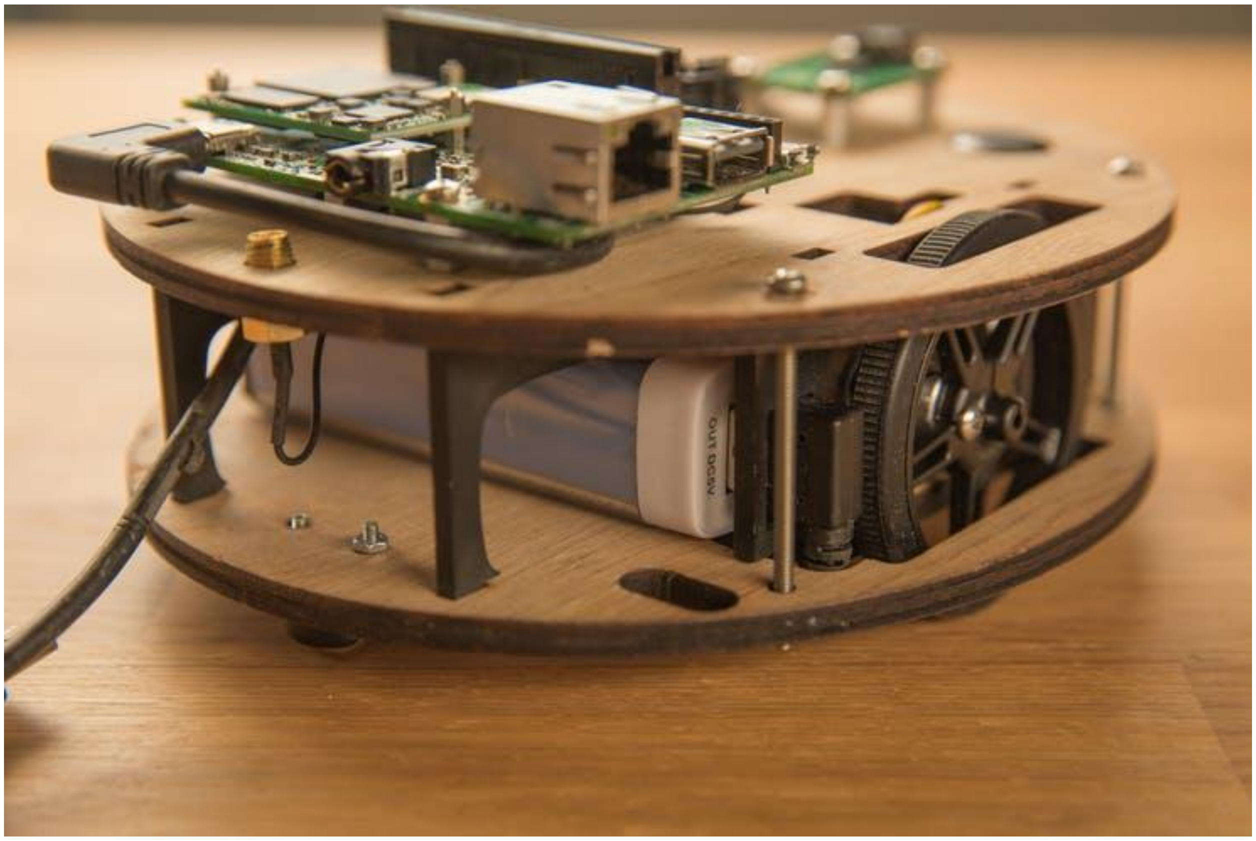


26. Thread the second USB cable through the slot under the Pico Pro board.

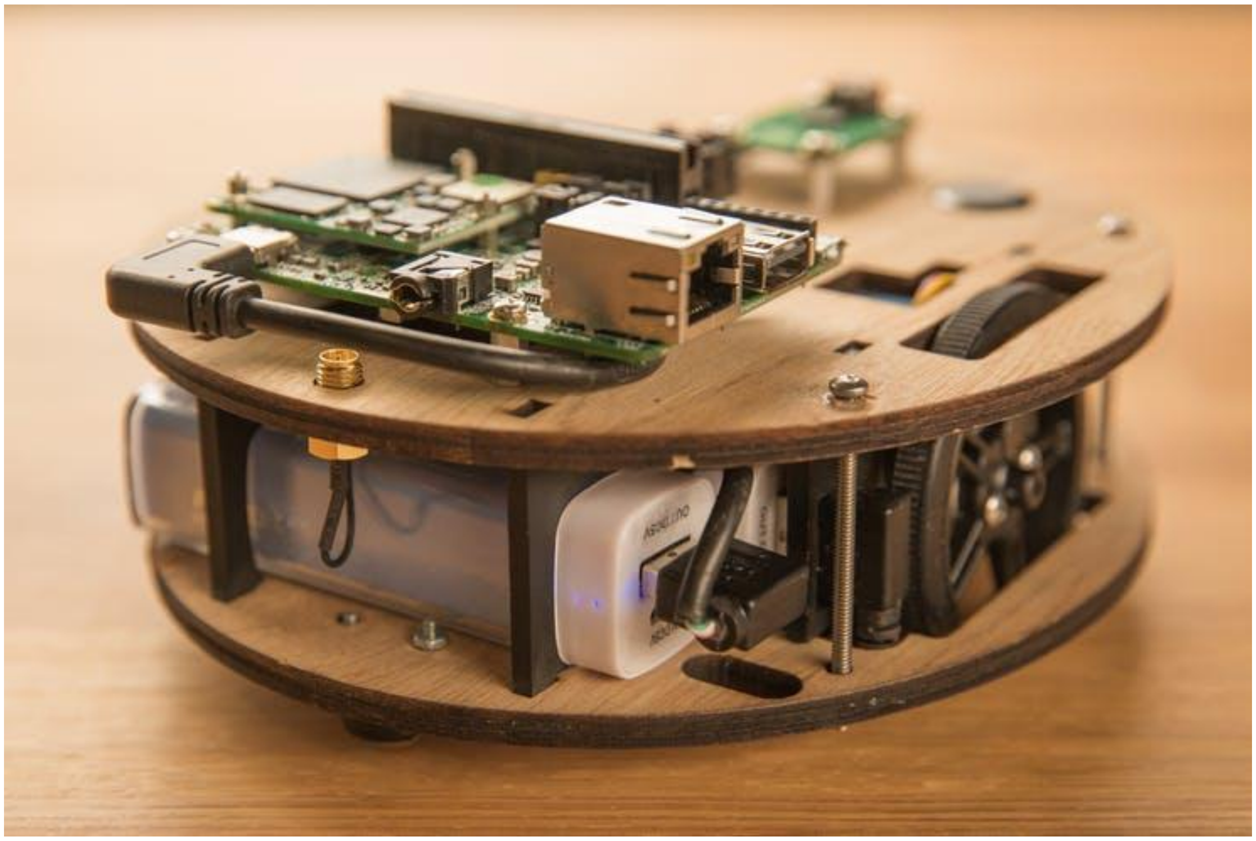
27. Thread the WiFi antenna jack through the slot under the Pico Pro board, and connect it to the Pico Pro. Thread the other end through the D-shaped hole on the upper wood deck and secure with the included nut.



28. Push the first battery into the rear slots as shown, and plug in the USB cable that goes to the breadboard. This battery powers the motors.



29. Push the second battery into place, and plug in the USB cable that goes to the Pico Pro. This battery powers the processor.



Installing the Software

The board's firmware is written and deployed entirely through Android Studio. It uses the Pico i.MX7D camera module to capture photos, OpenCV for facial detection and photo manipulation, and Android Things hardware drivers to control the button, LED, drive motors, and pen servo.

1. Flash Android Things onto the NXP development board using this guide.

2. Clone the software repository onto your computer.

3. Import the project into Android Studio. (File > New > Import Project)

4. Deploy the application. (Run > Run 'app')

Calibrating the DrawBot

Because mechanical systems (like these motors and gearboxes) aren't always perfect, you'll need to calibrate the DrawBot before it can generate pictures.

1. First, test that the stepper motors are wired properly by driving the DrawBot in a square.

On startup, press the button two times (wait for the LED to flash blue after each press). This will set the DrawBot up to run the square test pattern. After five seconds, the LED will flash blue twice to confirm your selection, then turn green. Place the DrawBot on the paper and press the button again to begin the sequence. If the robot doesn't move in a square, skip down to the Troubleshooting section.

2. Tune the servo motor positions with the marker in place.

The servo is able to hold the marker at four different heights: off the page, light pressure, medium pressure, and hard pressure. Turn on the DrawBot and press the button four times to go to the servo test pattern. This will draw line segments at each height, allowing you to adjust the height of the marker in the holder. You may need to adjust the servo positions in RobotConfig.java to get a good result.

3. Tune the stepper parameters until the DrawBot draws straight squares.

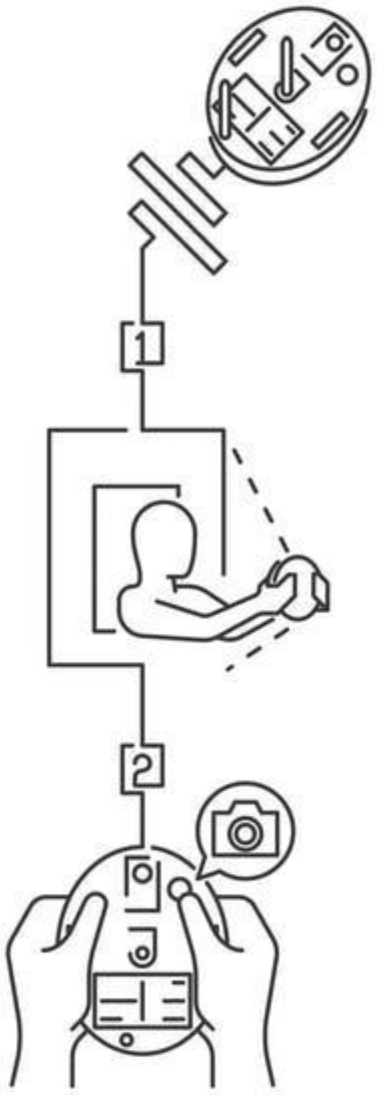
Run the square test pattern from step 1. (Press twice for a right-turn square or three times for a left-turn square). Allow the robot to draw a few overlaid squares. If the squares rotate over time, it means the DrawBot is not turning perfect 90-degree turns. Adjust these parameters in the code until the squares are close to perfect.

Running the DrawBot

Plug the NXP development board into one battery and the external hardware into the other. Once the LED boots to white, press one time to confirm setup. When the LED turns red, the application is ready and the bot is in the "waiting for photo" state.

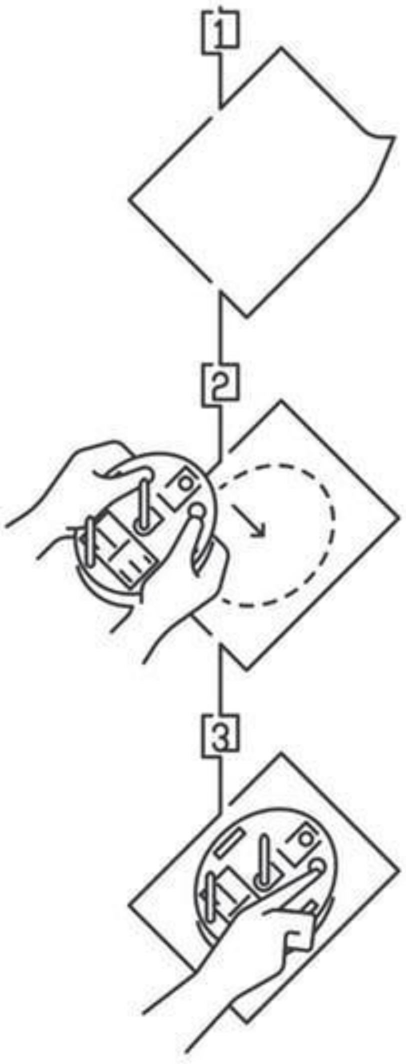
1. Tape a clean sheet of paper on a large, level drawing surface.

2. Hold the bot upright at arm's length with the camera facing you, and press the button. The LED will turn yellow to indicate it is processing.



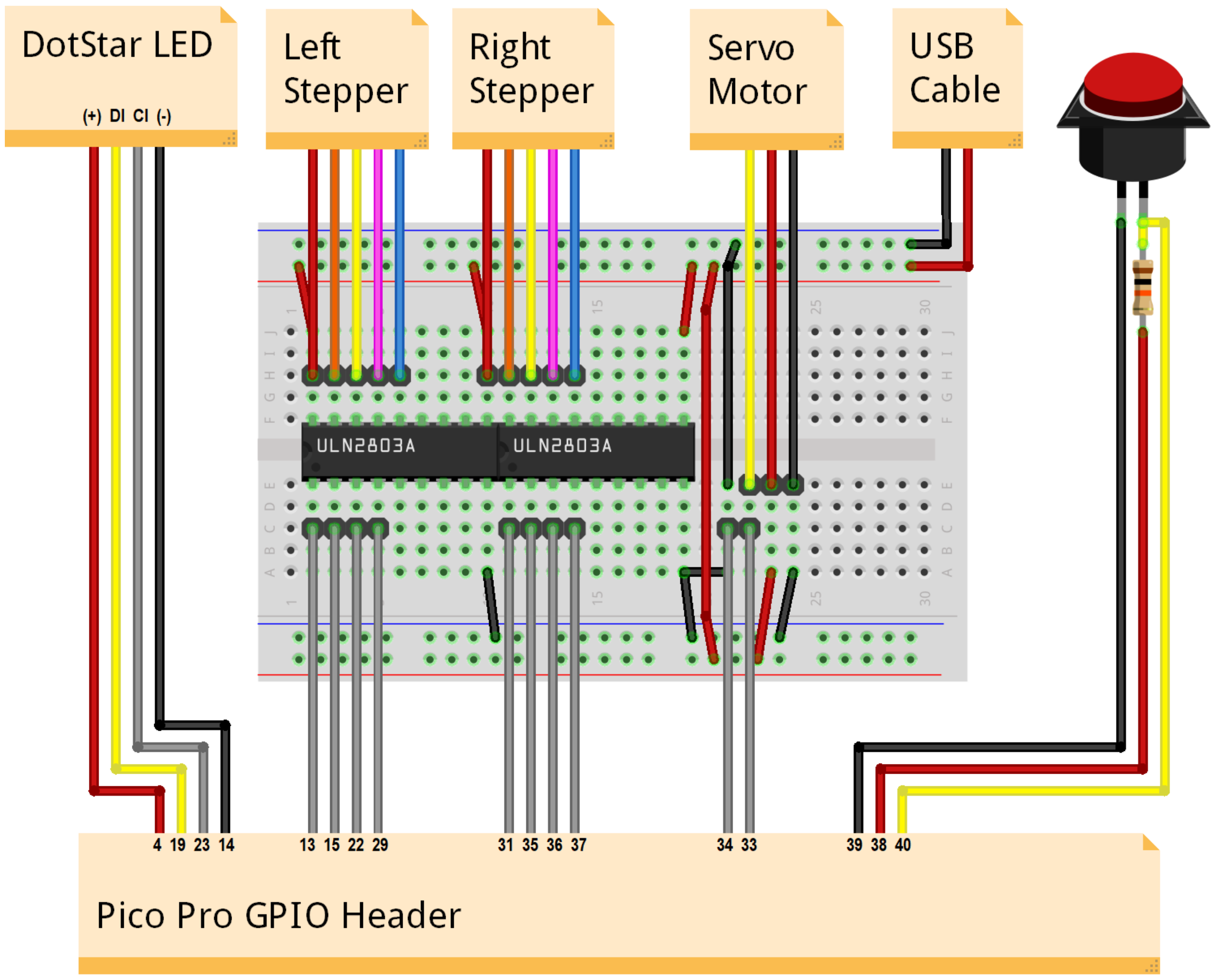
3. If the LED turns green, continue to the next step. If it goes back to red, a face was not detected and you need to re-take the photo (repeat step 2).

4. Place the bot in the center of your drawing surface, facing the top, and press the button to begin drawing.



5. Allow five or so minutes for the bot to draw.

6. Once the bot completes a drawing, it returns to the initial "waiting for photo" state.


DotStar LED
(+) DI CI (-)
Left Stepper
Right Stepper
Servo Motor
USB Cable
ULN2803A
ULN2803A
4 19 23 14
13 15 22 29
31 35 36 37
34 33
39 38 40
Pico Pro GPIO Header