21 PROJECTS! Arduino, ESP32, RasPi, Robots, 3D Printing

Make:



Knit This Working **Plushie Breadboard**

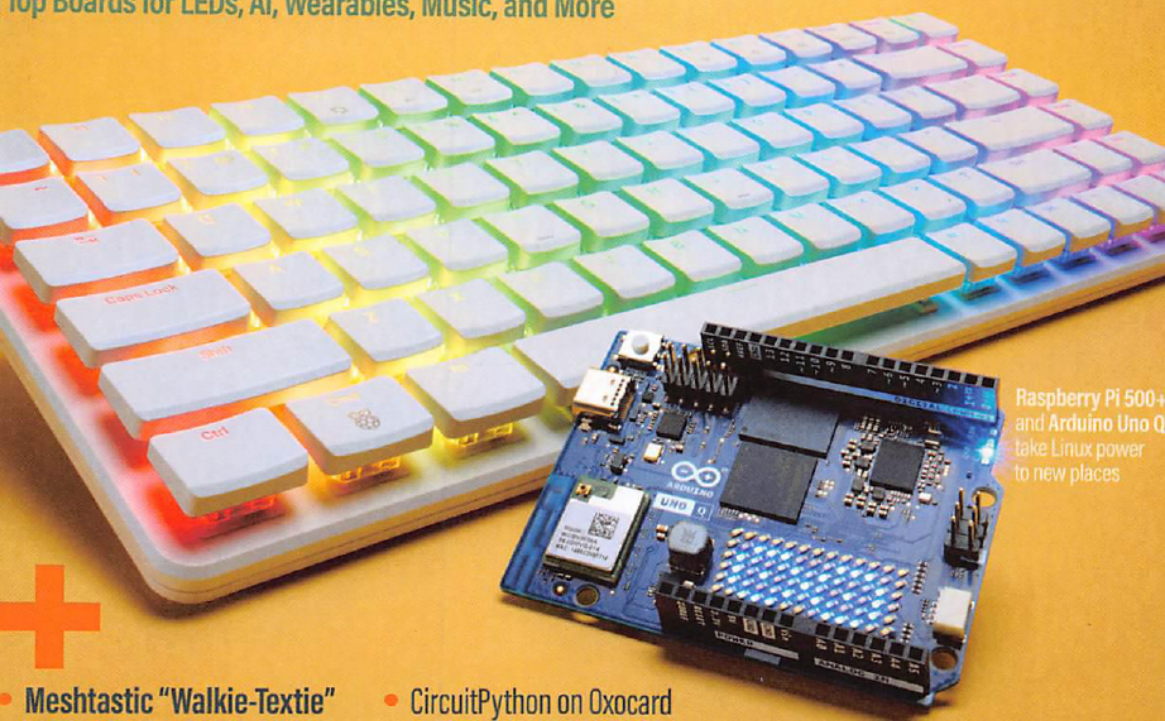
BOARDS GUIDE 2026

SPECIAL INSERT: 39+ SUPERB NEW BOARDS

SUPER [TINY] COMPUTERS

Linux Love: **New Arduino and Raspberry Pi**

Top Boards for LEDs, AI, Wearables, Music, and More



Raspberry Pi 500+ and Arduino Uno Q take Linux power to new places

- **Meshtastic "Walkie-Textie"**
- Voice Control Tools with ESP32
- **3D Print an Origami Handbag**
- Aircraft Tracking with ADSBee
- **Pocket Video Synth**
- CircuitPython on Oxocard
- **Differential-Drive Robot**
- Pocket Pickleball Game
- **Light Sensors & Amplifiers**
- Hands On with ROS 2

US $14.99 CAN $17.99
ISBN: 978-1-68045-889-3
9 781680 458893 51499>

make.co | makezine.com | makerfaire.com

VOLUME 95



Explore simple **AI CODING**, **ENGINEERING**, and **ROBOTICS** mixed with a maker **MYSTERY ADVENTURE**!

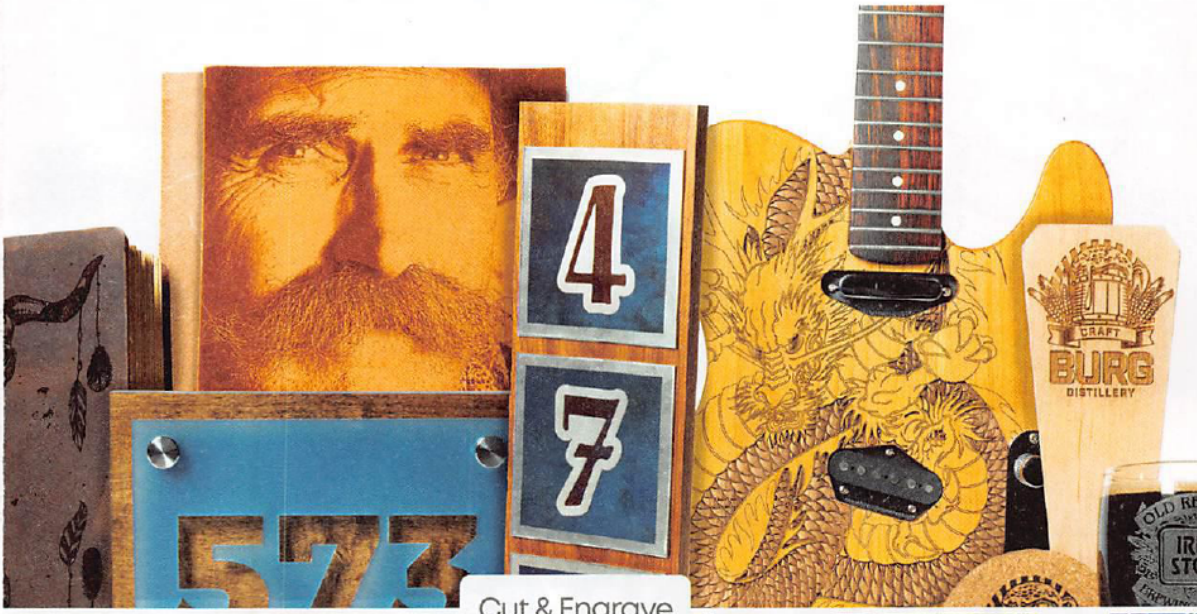
Get started at: veryusefulmonsters.com

Make: VERY USEFUL MONSTERS

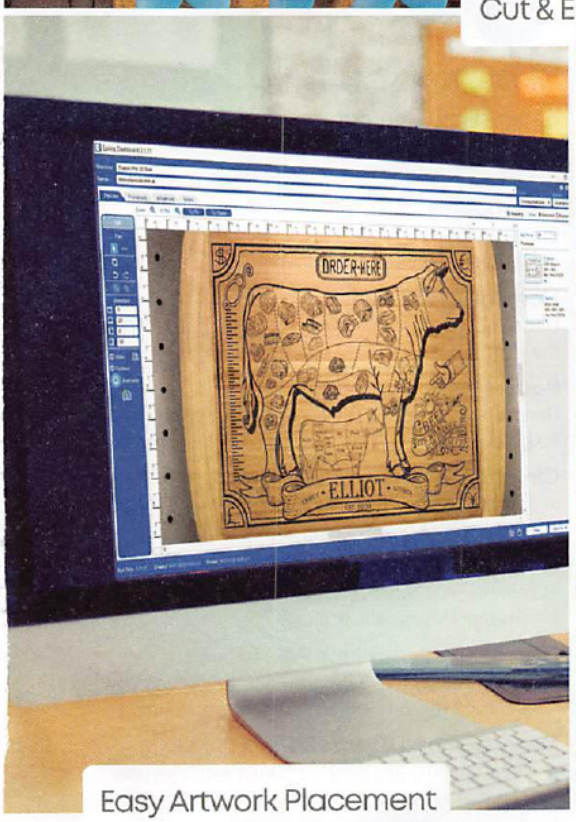


Brought to you by **Make:** and electrify

Elevate Your Side Hustle With EpilogLaser!



Cut & Engrave



Easy Artwork Placement



Higher Throughput

LEARN MORE OR SET UP A DEMO TODAY! | 888.437.4564 | SALES@EPILOGLASER.COM | EPILOGLASER.COM/MAKE



30

CONTENTS

COLUMNS

Welcome: A Maker's Maker 06
Apple cofounder Steve Wozniak held court at Maker Faire to recount the early days of maker community.

From the Editor's Desk 07
Letters and missives from readers like you.

Made on Earth 08
Amazing builds from around the globe.

FEATURES

Open Lab Starter Kit 12
Build your own digital fabrication machines — lasers, 3D printers, CNCs, and more — from this open source hardware toolkit.

A Real Big Band 16
Quentin Thomas-Oliver built a 13-foot, half-ton drum monster to keep the beat for his band Ponytrap.

My Voice-Controlled Workbench 22
"Zoom in and enhance!" How one maker automated his solder station, microscope, and magnifying visor using ESP32s and free home automation tools.

ON THE COVER:
The new Raspberry Pi 500+ and Arduino Uno Q both take a stab at making full-blown tiny computers.

Photos: Mark Madeo (Raspberry Pi 500+ and Arduino Uno Q), Michael Crommet (knitted breadboard)

BEST BOARDS

Super (Tiny) Computers 30
Arduino and Raspberry Pi's new releases put real computing power where we've never seen it before.

Indie Boards 32
Small teams at BeagleBoard and IceWhale use tiny computers to make a big impact.

Homage to the Axe 36
Cheap but mighty, Picaxe microcontrollers still get the job done.

Pocket Video Synthesizer 42
Generate video art on the go with a tiny gadget inspired by Teenage Engineering's Pocket Operators.

CircuitPython on the Oxocard Connect 48
In addition to beginner-friendly NanoPy, our nifty microcontroller now runs CircuitPython.

Meshtastic "Walkie-Textie" 54
Stay in touch without cellular, with this LoRa mesh communications project.

How to Speak Airplane 60
Track aircraft with a low-cost, open source, embedded ADS-B receiver that you can build into anything.

Make: Guide to Boards 2026 Booklet
The board landscape keeps growing with this year's new crop of versatile maker tools: microcontrollers, single-board computers, and beyond.



60


68

PROJECTS

Threadboard 64
Knit a working plushie breadboard from yarn and conductive thread.

3D Printing Folding Geometry 68
Make an elegant 3D-printed fabric bag that bends and folds like origami.

MBot 74
Build a robust wheeled robot platform and program it with MicroPython.

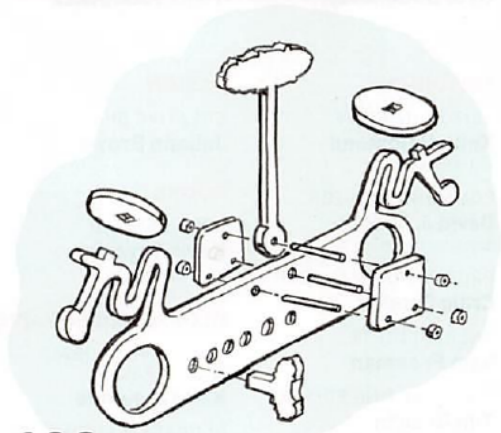
Squishy Tech: Talking Paper PCB 82
Embed electronics in paper circuit boards that communicate directly via ESP-Now, no Wi-Fi needed.

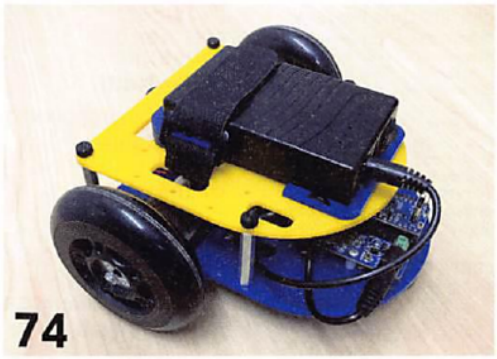
Adventures in Object Photography 90
Shooting your project or product? Low-budget solutions for controlling highlights, shadows, and background.

Amateur Scientist: Semiconductor Light Sensors 96
Learn the basics and make a transimpedance amplifier to boost light signals.

Reaction Race 100
Make a classic reaction game with micro:bit and cardboard.

Toy Inventor's Notebook: Pocket Pickleball 106
Make this pocket-sized, two-player action game that plays with a pickle instead of a ball!


106


74

SKILL BUILDER

Hands On with ROS 2: Nodes, Topics, and Services 110
An introduction to the Robot Operating System for hobbyists and engineers.

Making Makers 120
Inspire discovery in budding makers with gamified skill trees and level-ups.

TOOLBOX

Toolbox 122
Gear up with the latest tools and kits for makers.

OVER THE TOP

A Cube of a Higher Order 128
There's a new record for the most complex Rubik's Cube on the planet.


08

Mark Madeo, John McNelly, Sophy Wang, Bob Knetzger, MBot Ecosystem, Greg Detker

Make:®

"Making is where my success came from." —*Steve Wozniak*

PRESIDENT
Dale Dougherty
dale@make.co

VP, PARTNERSHIPS
Todd Sotkiewicz
todd@make.co

EDITORIAL

EDITOR-IN-CHIEF
Keith Hammond
keith@make.co

COMMUNITY EDITOR
David J. Groom
david@make.co

PRODUCTION MANAGER
Craig Couden

ONLINE EDITOR
Sam Freeman

CONTRIBUTING EDITORS
Tim Deagan
William Gurstelle
Matt Stultz

CONTRIBUTING WRITERS
Debra Ansell, Christopher Biggs, Kathy Ceceri, Jon Davis, Dom Dominici, Peter Gaskell, Shawn Hymel, Daniele Ingrassia, Bob Knetzger, Helen Leigh, John McNelly, Forrest Mims III, Abhishek Narula, Alanna Okun, Steph Piper, Marshall Piros, Charles Platt, Ramona Sharples, Erin St Blaine, Jennifer Strand, Jacques Supcik, Quentin Thomas-Oliver, Lee Wilkins, Sophy Wong

CONTRIBUTING ARTISTS
Josh Ellingson, Mark Madeo

MAKE.CO

WEB APPLICATION DEVELOPER
Rio Roth-Barreiro

DESIGN

CREATIVE DIRECTOR
Juliann Brown

BOOKS

BOOKS EDITOR
Kevin Toyama
books@make.co

GLOBAL MAKER FAIRE

DIRECTOR, GLOBAL MAKER FAIRE
Katie D. Kunde

GLOBAL LICENSING
Jennifer Blakeslee

MARKETING

DIRECTOR OF MARKETING
Gillian Mutti

PROGRAM COORDINATOR
Jamie Agius

OPERATIONS

ADMINISTRATIVE MANAGER
Cathy Shanahan

ACCOUNTING MANAGER
Kelly Marshall

OPERATIONS MANAGER & MAKER SHED
Rob Bullington

LOGISTICS COORDINATOR
Phil Muelrath

PUBLISHED BY

MAKE COMMUNITY, LLC
Dale Dougherty

Copyright © 2025
Make Community, LLC. All rights reserved. Reproduction without permission is prohibited.
Printed in the U.S. by Schumann Printers, Inc.

Comments may be sent to:
editor@makezine.com

Visit us online:
make.co

Follow us:
𝕏 @make
makemagazine
makemagazine
makemagazine
makemagazine

Manage your account online, including change of address:
makezine.com/account
For telephone service call 847-559-7395 between the hours of 8am and 4:30pm CST.
Fax: 847-564-9453.
Email: make@omeda.com

Make: Community

Support for the publication of *Make:* magazine is made possible in part by the members of Make: Community. Join us at make.co.

CONTRIBUTORS

What's on your maker wish list this holiday season?



Alanna Okun
Brooklyn, New York (Threadboard)
All the best pieces of leftover ribbon, wrapping paper, and boxes I can get my hands on!



Ramona Sharples
San Francisco, California (Pocket Video Synthesizer)
I hope Santa brings me a hot air rework station so I can fix my mistakes!

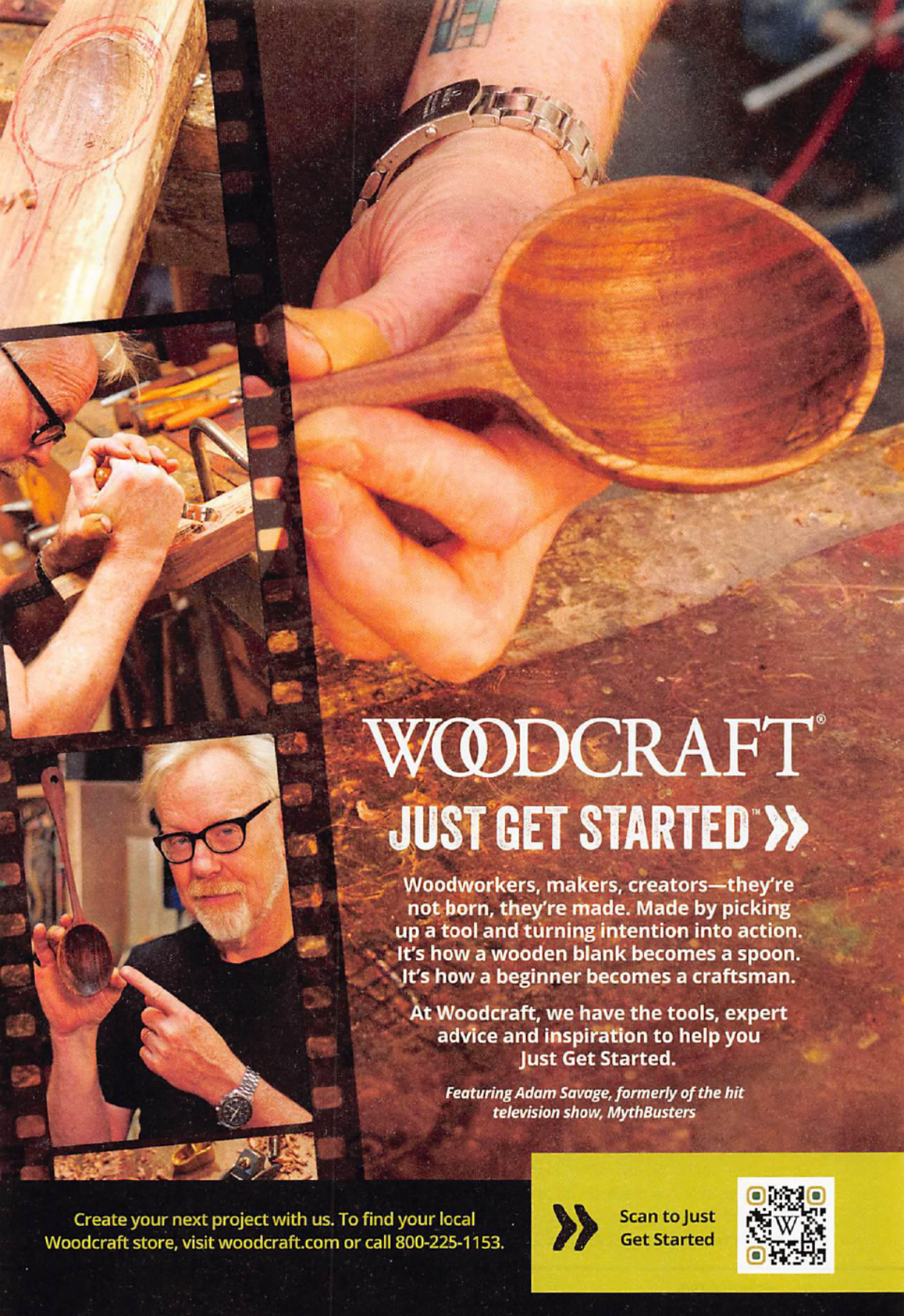


Christopher "Kit" Biggs
Brisbane, Australia (My Voice-Controlled Workbench)
I want to assemble a mechanical wristwatch with parts from AliExpress. Then add an e-ink dial.

Issue No. 95, Winter 2025. *Make:* (ISSN 1556-2336) is published quarterly by Make Community, LLC, in the months of February, May, Aug, and Nov. Make: Community is located at 150 Todd Road, Suite 100, Santa Rosa, CA 95407. SUBSCRIPTIONS: Send all subscription requests to *Make:*, P.O. Box 566, Lincolnshire, IL 60069 or subscribe online at makezine.com/subscribe. Subscriptions are available for $34.99 for 1 year (4 issues) in the United States; in Canada: $43.99 USD; all other countries: $49.99 USD. Periodicals Postage Paid at San Francisco, CA, and at additional mailing offices. POSTMASTER: Send address changes to *Make:*, P.O. Box 566, Lincolnshire, IL 60069. Canada Post Publications Mail Agreement Number 41129568.

PLEASE NOTE: Technology, the laws, and limitations imposed by manufacturers and content owners are constantly changing. Thus, some of the projects described may not work, may be inconsistent with current laws or user agreements, or may damage or adversely affect some equipment. Your safety is your own responsibility, including proper use of equipment and safety gear, and determining whether you have adequate skill and experience. Power tools, electricity, and other resources used for these projects are dangerous, unless used properly and with adequate precautions, including safety gear. Some illustrative photos do not depict safety precautions or equipment, in order to show the project steps more clearly. These projects are not intended for use by children. Use of the instructions and suggestions in *Make:* is at your own risk. Make Community, LLC, disclaims all responsibility for any resulting damage, injury, or expense. It is your responsibility to make sure that your activities comply with applicable laws, including copyright.

PRINTED WITH SOY INK


WOODCRAFT®
JUST GET STARTED™
Woodworkers, makers, creators—they're not born, they're made. Made by picking up a tool and turning intention into action. It's how a wooden blank becomes a spoon. It's how a beginner becomes a craftsman.
At Woodcraft, we have the tools, expert advice and inspiration to help you Just Get Started.
Featuring Adam Savage, formerly of the hit television show, MythBusters
Create your next project with us. To find your local Woodcraft store, visit woodcraft.com or call 800-225-1153.
Scan to Just Get Started

A Maker's Maker: Woz at Maker Faire

by Dale Dougherty, President of Make: Community



In 1975, Steve Wozniak had a design for the Apple I PC board. He was hanging out with Steve Jobs, who was keen to manufacture and sell the boards, but neither of them had any money. Woz was more interested in showing off his design to other engineers at the Homebrew Computer Club.

On stage at Maker Faire Bay Area at Mare Island in September, Woz recounted the story of designing the Apple I computer 50 years ago. When I asked him what he was doing in 1975, he talked non-stop for 30 minutes.

His father introduced him to electronics, but Woz was mostly self-taught. "I was already skilled at building computers by 1975," he said. "Five years before, I built the little 'cream soda' computer," named after his favorite soft drink. It was ugly, with switches and lights on the front panel, but the day he showed it off was the day he met Steve Jobs. "He supposedly had things in common with me, an interest in electronics and pranks. ... He admired the fact that I knew how to design things and actually knew how to use electronics."

"I was so shy. I had no friends," said Woz. "Jobs and I became best technical friends forever." That first day, Woz brought Jobs home. He said Jobs was "searching for what's in the world," driving around in a van with absolutely no money. They listened to Bob Dylan records at Woz's house because Jobs couldn't afford to buy albums.

"I designed processors over and over in high school," said Woz. "You couldn't get any manuals or any magazines — no bookstores had anything about computers. I stumbled on a journal article here and there. I eventually found a small computer manual that taught me the architecture."

In Palo Alto, Woz snuck into the Stanford Linear Accelerator Center (SLAC), where high-level physics research took place. He discovered that "the smartest people in the world don't lock their doors." He would drive to SLAC on a Sunday and visit the library where he could read about computers.

"At first, I designed my computer on paper only, and I was really good at that. I got to the point where I could design the chips that would make a processor, but I didn't see how I could afford input and output devices," said Woz. "Then I saw *Pong*. It's on a TV. Everybody owns a TV."

"The very first meeting [of the Homebrew Computer Club], I found out that people weren't interested in terminals. They were interested in this thing that Altair called a computer. Altair was an 'ugly big panel' with switches and lights and 256 bytes of memory. Well, that's what I had built five years before with my cream soda computer. I wanted (the input) to be like a typewriter. I love typewriters!

"Because I worked at Hewlett Packard, I had access to chips. I took my *Pong*-type design and built a new computer. I typed on a keyboard which would cost about $600 today, the most expensive thing on my way to a computer, and *it put letters on my screen*. I could type to the faraway computer on the ARPANET using a modem."

Woz brought the new design to the club. "Some people built their own from my designs and I helped others build theirs and it started getting popular. I was too shy to ever speak in the club meetings, but I would take my computer and set it up.

"Jobs had never been to the club. ... I took him to the club to show him the excitement and when he saw all the people gathering around me, he knew we were seeing the future. Every computer before that had a front panel, but every computer after the Apple I, after I showed it off at the club, came with a keyboard and a video display.

"That was the paradigm for low-cost computing. Steve saw the interest and that's when he said we should start a company.

"I had to sell my most valuable possession, my Hewlett-Packard HP-65 calculator. I sold it for

500 bucks. Steve sold his van. We paid $500 to get the PC boards made. We were gonna make about a hundred of 'em. Steve Jobs's idea was to sell that PC board blank, and if 50 people buy it, we'll get our money back maybe. And who cares about getting your money back? You get to have a company when you're young, call yourselves a company. At any rate, I did not want to start a company. I did not want to start an industry. I just wanted other engineers to see my work, to tell you the truth."

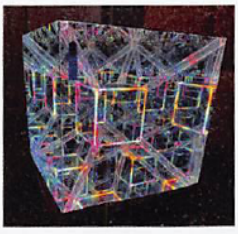
In July 1976, the Apple I went on sale for $666.66, with the Apple II soon to follow. It was the birth of the low-cost computing we celebrate in this issue of *Make:*.

That's a Wrap for Maker Faire Bay Area!

Maker Faire Bay Area took place just as we went to press; see more at youtube.com/@makerfaire!









Juliann Brown, Craig Couden

FROM THE EDITOR'S DESK





VARIETY IS THE SPICE OF LIFE

Thank you for your outstanding *Make:* Volume 94. While some of us are no doubt interested in Arduinos and LEDs to the exclusion of all else, many readers see the *Make:* world much more broadly. This issue includes prop and model making, painting, DIY drones, puzzle rooms, alternative soldering techniques, workshop design and safety, robot costumes, cinematographer techniques, simple rockets, and even squirt bottles! The articles seem extensive and complete, and span a broad range of abilities and skills so we can all get in on the act. Bravo! —*Randall Smith, via email*

Production Manager Craig Couden responds:
Thanks Randall! We try to strike a balance of interests and skills — even in this electronics-heavy issue. We hope you find something to like!

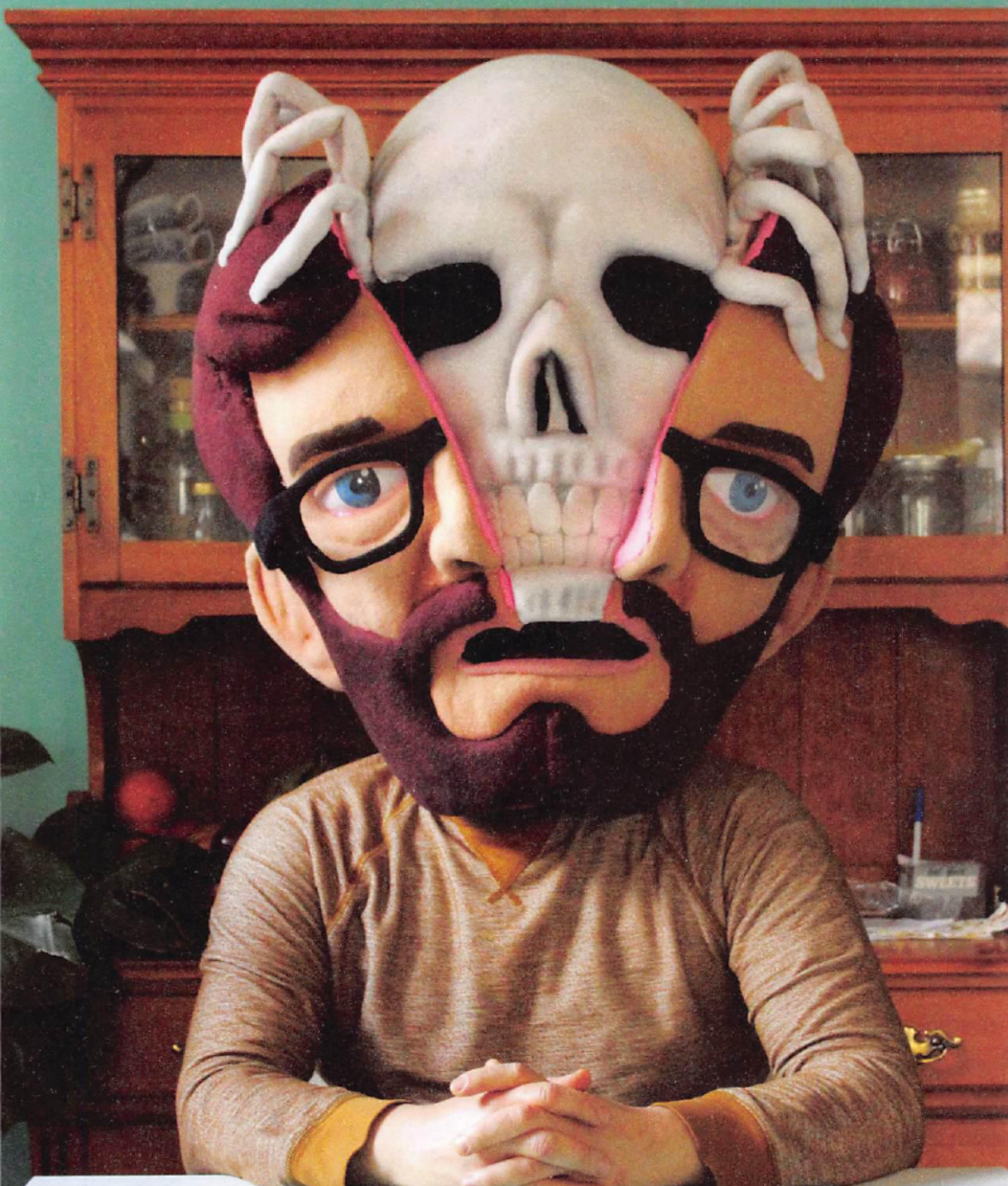
MACGYVER-IN-A-BOX CONTEST

Can you solve one of the planet's biggest problems? Join *Make:* and the MacGyver Foundation in our MacGyver-in-a-Box Home Sanitation Design Challenge, a cash award contest to identify the best ideas for low-cost, indoor toilets that could give over 1 billion people access to basic bathroom sanitation. What would MacGyver do? Submissions open November 1 at macgyver.com/foundation.

MADE ON EARTH

Amazing builds from around the globe

Know a project that would be perfect for Made on Earth? Let us know: *editor@makezine.com*



MASCOT MADNESS

INSTAGRAM.COM/HAND.SEWN.HEADS

Oversized, plush mascot characters are a common feature at entertainment venues like theme parks and sports arenas. **Ian Langohr** takes this concept to a new level, bringing characters and creatures of all sorts to life through his unique hand-sewn, mascot-style heads.

Based in Montreal, Quebec, Langohr's skill is built upon many years of sculpting education. He gained experience in stone and wood carving, bronze casting, and plasma cutting during high school and later majored in sculpture and installation at Toronto's Ontario College of Art and Design. He then spent time working with foam, fabric, and plastics at a mascot fabrication company and now uses all his skills to help companies and artists bring their costumes to life for performances, trade shows, and advertisements.

Langohr draws inspiration for both his personal and professional pieces from many sources. When working for himself, he often turns to online references of his favorite horror films or mythological creatures. "I make crude sketches in pencil and crayon in order to play with color," he says. "This is how I've been comfortable in my own process for my work." That process is usually similar across commissions and personal pieces. He starts with black polyethylene foam for a base, then uses retractable blades and several different types of glue — hot, spray, cement, and super — to shape his creation. After forming the foam, he covers it in fleece or fun fur fabric and sews it together; further details are added with plastic attachments, rubber bits, and airbrushing. "The design process varies in time depending on whether it is a commission or my own personal piece," Langohr says. "For the former, I am often provided with detailed art of the product and can finish a piece in a matter of weeks. For personal pieces, I often work on them between commissions and they can be paused and picked back up for long periods. I do try and make both commissions and personal pieces have as much dynamic personality as I can!"

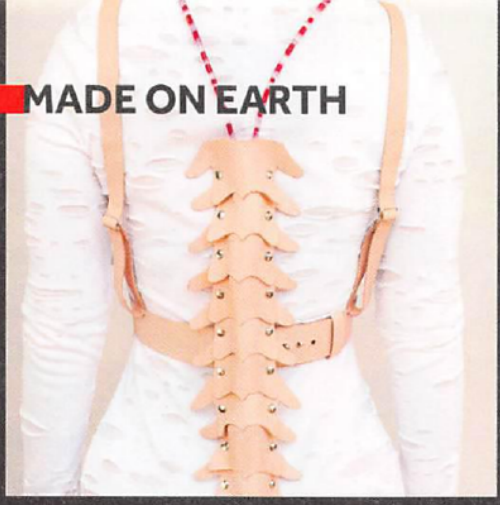
—Marshall Piros







MJ Archambault, Ian Langohr, Greg Detker







Courtesy of Brittany Cohen

BOUND BY (FAKE) BLOOD

INSTAGRAM.COM/BRITTANYANNECOHEN

Brittany Cohen wanted to blend her background in electrical engineering and her creativity to make a wearable piece that was an extension of the body, so she started experimenting with plastic tubing and built the first prototype of her Liquid Corset. She documented her work and posted a video on the internet asking for feedback. People loved what she was doing and made suggestions (i.e. making it more macabre) so Cohen spent the following year incorporating her viewers' advice and testing improvements to the design.

She ran into several challenges along the way. The first resin dye she added to the mixture of mineral oil and water stained the tubing, so she switched to a water-based dye. She also had to experiment with tubing compatibility to find the right material which wouldn't break down over time and result in the oil permeating through. The structure of the corset itself went through iterations using different widths of leather so the tubing didn't bunch, and 3D-printed connectors to keep the tubes from slipping out of place when putting on or taking off the corset.

The final design resembles a ribcage with blood coursing through, pulsing at intervals fueled by a hardware pack that includes a small pump attached to a LiPo battery, a battery protection board, buck converter to ensure the pump is powered consistently, and a charging board, all hidden in a spinal column fashioned from leather. The result of her year of hard work and meticulous experimenting is a beautiful and creative nod to the engineering of the human body.

See Brittany's documentation of how the Liquid Corset evolved on her Instagram or Tiktok (tiktok.com/@brittanyannecohen). *—Jennifer Strand*



Jacs Fishburn, Ulysses Secrest

FREESTYLE METALWORK AERARIUSARTS.COM

True to the spirit of making, **Ulysses Secrest** — known online as Aerarius — builds from the ground up, forgoing many traditional and technological design assistance tools in favor of working directly from their imagination.

Secrest, who was raised in their parents' art gallery in Columbus, Ohio's Short North Arts District, has always felt a special connection to the arts. Notably, their process and philosophy are very much centered on the idea of starting totally from scratch. "Typically," they say, "I don't start with a specific drawing or follow a set plan. I will often just throw myself into the work and figure out where I am going on the fly." They are also firm about not using computer-based or digital aids during the design process — "partly because I enjoy the manual element and the labor involved, but also because I believe that it allows me to produce higher-quality work."

When designing jewelry, which often includes intricate pendants and chains or multi-piece rings with moving pieces, Secrest tends to use wax as a creative base. Without measurements or schematics, they can easily add more heft to an in-progress design or shave down unnecessary details. They follow a similar freewheeling approach to constructing their moving masks: "I envision how I want the movement to work, and build the mask in whatever way it needs to meet the vision, often having to adapt and change as I go. This ends up making the pieces very personal and exaggerated forms that are difficult to fit onto people other than myself." Secrest's masks can also feature flashes of color and spectacle — one mask, the "Night Orchid," is covered by glow-in-the-dark pigmented tissue paper, while the paper covering of another mask is designed to be lit on fire! *—Marshall Piros*

OPEN LAB STARTER KIT

Written by
Daniele Ingrassia

DESIGN FOR REPLICATION

Build your own digital fabrication machines from this open source hardware toolkit

Small CNC



Large CNC



Vinyl Cutter



Small Laser



Large Laser



3D Scanner



Small 3D Printer



Large 3D Printer



THE 8 OLSK MACHINES AT A GLANCE.



An Italian living in Germany, DANIELE INGRASSIA is a Fab Lab guru, a machine builder, and the owner of InMachines Ingrassia GmbH, a small company located in a small town close to Hamburg.

In 2016, I was a fresh graduate of Fab Academy. Just after moving to Germany, I started teaching as a Fab Academy instructor at the Rhine-Waal University of Applied Sciences. Since they were so kind as to give me the key to their massive Fab Lab, I also started building machines.

I began making LaserDuo, a massive dual laser cutter, that year, and the machine building activity continues 9 years later at my company InMachines, with the **Open Lab Starter Kit (OLSK)** — a set of open source designs for all kinds of digital fabrication machines.

LaserDuo was built with two laser sources — a 130W CO_2 and a 75W YAG — and a big cutting area of 1500×1000mm to reduce cutting time for large wood and acrylic sheets (which at that time were done with a CNC mill). Thanks to its size, LaserDuo is still in use after 8 years; it's particularly useful to cut the machine housings of the other OLSK machines (e.g. the windows of the 3D printers). In 2025, LaserDuo was installed in the German Museum of Technology in Berlin as part of their permanent exhibition. (You can find LaserDuo and other older designs at github.com/fab-machines.)

Make Your Own Machines

In 2021, we started the development of the Open Lab Starter Kit in cooperation with the New Production Institute (NPI) of Helmut Schmidt University, for the "Fab City" project funded by the European Union. The goal? To study how distributed and decentralized production modes can foster local value creation. With the NPI we decided to develop eight different open source hardware machines to cover the most common needs of a Fab Lab or makerspace. We embarked on a prototyping marathon, developing three versions for each machine type — a total of 24 machine prototypes in 3 years.

The main goal of the OLSK was to design high-quality machinery that can be replicated using easy-to-source materials and parts — good machines that anyone could build locally anywhere. If you have a small shop, you could make some of the parts at home and outsource the rest, or if you have a Fab Lab or makerspace nearby, you could even make an entire machine with the tools there.

For each machine, we held workshops to prove





OLSK

LaserDuo was built together with Ahmed Belal Abdellatif at Rhine-Waal University of Applied Sciences (Germany).

replicability, reliability, and usability, testing them with wood shop owners, local Fab Labs, school labs, fashion studios, etc.

Each machine comes with open-source documentation including CAD models, a bill of materials, electronic schematics, and online 3D assembly guides. We really push to enable people to build, repair, and modify their machines as needed. To tell the truth, during our journey, my team noticed how long it takes to produce good, clear, and complete documentation for open source hardware. It takes more time than producing and building the prototype itself! So, we are working hard to fully complete the documentation that will be released in the repos for the latest developments.

OLOS: An Operating System for Open-Source Machines

OLOS stands for **Open Lab Operating System**, an open source control system designed to provide a unified interface for operating and monitoring OLSK machines. Managing multiple digital fabrication tools can be complex, so we tackled the challenge of standardizing machine operation, for compatibility across the different tools.

OLOS also improves the usability of the OLSK tools by making them stand-alone machines,

without the need for multiple software packages and a computer to operate them. Features include:

- Wi-Fi file manager
- AI image generator (for the laser only)
- G-code generator (laser only)
- Tool-changer manager
- Job preview
- Camera positioning system (laser only)
- User interface

Online Assembly Manuals

We started with a static manual in the form of a PDF, but soon realized the difficulty of updating it. So we switched to online assembly manuals, created with software we developed in-house (github.com/Open-Lab-Starter-Kit/Online-Documentation). This format improves:

- **Accessibility:** it's web based and open source.
- **Interactivity:** features like zoom, pan, and orbit to details, explosion of parts, camera reset, easy navigation of steps, animated assembly tips, and identification of parts, all enable the assembler to have a customized experience.
- **Ease of replication and updating:** it's based on one 3D model, with automatic step visualization, parts counts, and text input from a shared sheet.

What Users Say

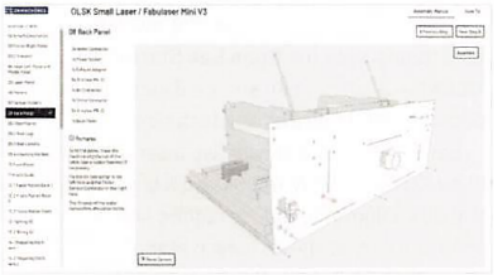
We've built OLSK machines in workshops with customers and makers around the world, and some people have attempted independent replications. The Fabulaser Mini has about 30 machines replicated or sold as a kit in Ukraine, USA, Germany, Portugal, and Indonesia. Other machines have been built in Tunisia and Ghana.

People really enjoy the workshop experience, which is much more than mechanical assembly. There is a lot of knowledge transfer, from zero to everything you need to know to design, make, assemble, and maintain the machine. Some participants started their own business like ours; others applied to work with us. The Tolocar Mobile Fab Lab operators in Ukraine are happily fixing their two Fabulasers without waiting and paying for service, using cheap parts on the local market.

Truth be told, you need some hands-on experience or previous knowledge to make proper use of these machines. Despite our efforts to make them easy to use, some customers gave up and returned the machines for a loss.



OLOS interface.



Online assembly manual.

We also found that not all users are allowed to operate non-certified machines, restricting their use in strictly safe environments like universities.

The Future of Open Source Fabrication

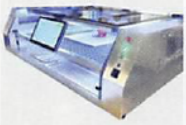
Open source digital fabrication can become a key enabler of self-sufficient, local economies. We are glad to help provide the tools and knowledge to build a more collaborative and sustainable manufacturing future, as in the Fab City project (discover more in the Fab City Playbook at makezine.com/go/fab-city-playbook).

You can find all the OLSK machine data and documentation, the Online Assembly Manual, and the Open Lab Operating System, at our GitHub repo, github.com/Open-Lab-Starter-Kit.

InMachines is currently focused on developing an open source metal 3D printer (Open Industrial Starter Kit, OISK) and an open source, CE-certified commercial milling machine. Stay tuned for our crowdfunding campaign at inmachines.net. Stay open and replicate!

Thanks to the following people for their support:

- Prof. Dr. Karsten Nebe and Ahmed Belal Abdellatif from Rhine-Waal University of Applied Sciences
- InMachines Team: Gaia Di Martino, Liane Honda, Wilhelm Schütze, Sulaiman Tanbari, Marc Kohlen
- Dr. Tobias Redlich, Dr. Manuel Moritz, and the New Production Institute team

Machine		Description	BOM Cost/Build Time
OLSK Small 3D Printer		A compact desktop 3D printer (build volume 235×235×235mm) with a CoreXY motion system and an enclosed aluminum frame. It has a maximum speed of 1,200mm/s and the extruder can reach up to 295°C. It integrates accelerometer, mesh bed leveling, monitoring camera, and a filament sensor which enables auto-pause.	BOM is about $1,400; requires about 1 day for assembly and testing.
OLSK Large 3D Printer		A large-format 3D printer (build volume of 1,000×1,000×1,300mm) equipped with a 1.4mm nozzle and an enclosed housing. With a CoreXY motion system, it features a quad-point self-leveling gantry system, kinetic joints, and eddy current surface scanner for height compensation.	About $12,000. Takes up to 2 weeks to assemble, calibrate, and operate.
OLSK Small CNC		A fully enclosed desktop CNC milling machine (milling area of 600×400×150mm) with a CNC-milled aluminum frame, open-source tool changer, coolant system, Z height tool sensor, and air-cooled spindle. The latest version has a touchscreen with OLOS and a vacuum bed.	About $4,000 and 3 days to assemble and get it operating.
OLSK Large CNC		A fully enclosed large-format CNC milling machine (milling area of 2,500×1,250× 300mm) constructed with aluminum square tubing and 3D-printed connectors. There are ball screws on all the axes, with rolling ball screw nuts on the Y, decoupled 1kW AC servomotors, a pneumatic system to clean the tool and deploy the exhaust brushes, and automatic tool changer. Thanks to OLOS, it features Wi-Fi connectivity and remote management.	About $10,000 and 10 days to assemble.
OLSK Small Laser aka Fabulaser Mini		A compact laser cutter (600×400mm cutting area) powered by a 40W CO_2 laser source, with maximum speed of 1,000mm/s and resolution of 0.05mm. Based on the first desktop laser cutter I designed and prototyped (Fabulaser Mini), it minimizes the amount of parts thanks to the interlocking aluminum plate of the frame and housing. It can cut materials ranging from wood and acrylic to textiles. This is one of our most developed and debugged machines.	About $2,200 and 3 days to assemble.
OLSK Large Laser		A large-format laser cutter (1,000×600mm). We're back to our origins with this dual laser machine — 60W CO_2 RF laser (air cooled) and 40W diode — but there's a new tool changer, which can switch laser heads even during a job, plus a camera positioning system, closed-loop stepper motors, automatic focus, and inductive sensors for smooth homing. Thanks to OLOS, it implements G-code generation, AI image generation, and a dithering algorithm for engraving.	About $9,000 and 4 days to assemble with 60W CO_2 RF laser; about $5,000 with 40W instead; or about $4,000 if you choose the diode laser only.
OLSK Vinyl Cutter		A compact vinyl cutter with a cutting width of 300mm. Aluminum and 3D-printed parts, with widely available off-the-shelf components including an Orange Pi that offers remote management via a hotspot with OLOS.	About $250 and 1 day to assemble — very cost friendly and a good starting point for less experienced makers.
OLSK 3D Scanner		A 3D scanner with turntable and fixed background. One high-quality camera slides on an arc to capture the object from every angle.	About $150 and 1 day for assembly; perfect for medium-experienced makers.

A Real Big Band

By Quentin Thomas-Oliver, as told to Dale Dougherty

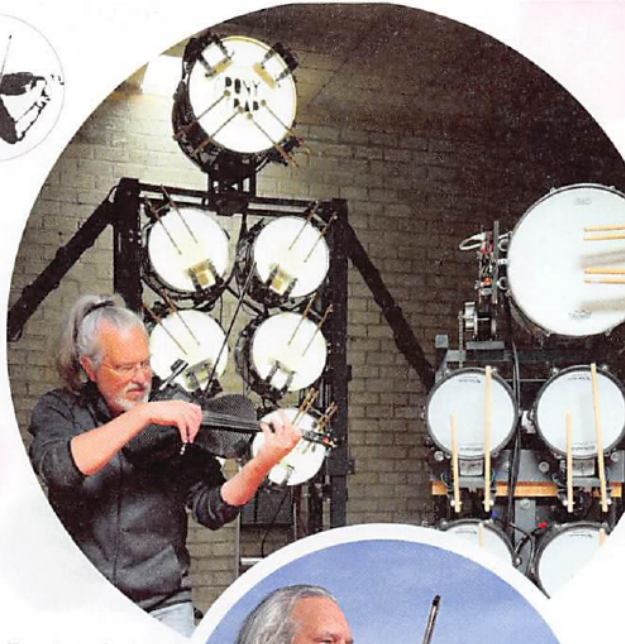


Ponytrap built a 13-foot, half-ton drum monster to keep the beat



The creator of Ponytrap, Quentin Thomas-Oliver, is an American living in the Netherlands. Ponytrap is a band with a 13-foot-tall drum machine called Max, built to be the collaborator Quentin always wanted — reliable and predictable. Quentin and Max play a strange blend of heavy metal and classical strings. Ponytrap is traveling the world from the Netherlands to France, Germany, Prague, and the United States; they came to Maker Faire Bay Area in 2024 and back to the Exploratorium in San Francisco in 2025. Quentin has become a touring musician in part because, along the long and winding way, he became a maker by building a robotic bandmate.

I caught up with Quentin at Maker Faire Lille, where we talked about Ponytrap's history and how he became a Maker.





Top: Quentin, Max, and newest robotic member Nova, in their Maastricht, Netherlands workshop.

Bottom: Quentin is a classically trained violist with a passion for heavy metal.

My university training, my professional life, is all as a musician. I have a degree in classical viola performance. I perform with Ponytrap, and orchestras, and teach private lessons on a variety of instruments. In one way or another, I've been a professional musician almost my entire adult life.

My family wasn't very musical. Neither of my parents played, but I took some perfunctory piano lessons. It was just something you're supposed to do and it didn't really stick. Even so, I loved music from a very young age. Kiss was huge at the time — this sort of kabuki-style theatrics with the flames and costumes and hard-hitting rock and roll. I wanted to be an outrageous performer like Gene Simmons.

I learned to play bass and joined a garage band. I eventually went to the University of Texas, not because I cared about the school, but I had heard that Austin was a fantastic music community — it was, and still is to this day. As soon as I got there, I dropped out and joined a band. I toured around in a couple of hard rock bands for a few years and was lucky to play some famous venues like CBGB and Limelight.

About the age of 23 or so, I started feeling like I was too old to be in a band, which is ridiculous but ... I went to a staged play about Paganini, the virtuoso violinist. It was the first time I'd heard classical music with that kind of intensity and it very much changed my life.

Band relationships are difficult. It's like you're married to five people instead of just one. It's hard, and that life is hard. I was looking for a way to stay in music but get out of being in a band. I saw this Paganini thing and just lost my mind. I'm like, that's it — that's the thing I want to do.

Quentin borrowed a viola that his brother had abandoned and started practicing for hours every day. He found a teacher, Martha Carapetyan, and took private lessons, eventually deciding that he wanted to go back to school to study music.

At the age of 35, I found a place willing to take me, the University of Iowa. The professor, Christine Rutledge, was willing to take me on as a project, having never even been in an orchestra. I was older than several of my professors and very much a fish out of water, but I made it through. After graduation, I moved to Colorado and started working in symphonies and teaching a little bit. It felt like I was finally getting settled at almost 40 years old.

In Colorado, I put together an early version of Ponytrap with some people who seemed like-

Katie Cowden, Kerne Media, Hilary Thomas-Oliver

minded. We even did a bit of making: creating a set of one-string cellos out of rebar. The project went off in directions I hadn't anticipated, but it was creative and fruitful and fun. Then in 2011, we're at the South by Southwest music conference back in Austin and I had the absolute worst gig of my whole entire life. Like, it could not possibly have gone more sideways.

We did two sets and the first one was great. It was all music I was really excited about. But, somehow, for the second set, they convinced me to do a 45-minute free-form improvisation while someone I'd never met read part of *Howl*. I can only describe it as a breakdown. I just started coming unglued. I was thinking, this couldn't be farther from what I'm trying to do.

This is [future wife] Hilary's entrance. We'd worked together years before at a bar in Austin and reconnected through social media. She was living in town and attended this show — watching in real time as I just crumbled to dust. Hilary tells the story of just sort of watching this shift in my demeanor, like I'm almost being born at this moment, like my villain origin story.

As I was grinding my way to the end of that show, I thought, "I'll just build a machine!" Something that would play the same thing twice. Something that would help me have a bigger hand in my own destiny. That moment, March 2011, was the exact moment when this new thing started happening.

The first steps were difficult, because I've just jumped in the ocean. It's a huge open space and I don't know what I'm doing. I'm not an engineer. I'm a musician. I know some smart people and I just started calling them. My original viola teacher's husband, Ken Amstutz, was a physical engineer working on chips. I asked him, "Hey, Ken, what would it take to make a drum that played itself?" Of course, he gave me some NASA-engineered solution that was way out of my league. But I kept having conversations, right? Hilary had a friend, another engineer, Sean Dunn, whose name is important to me because he said one word that changed everything: "You should check out an *Arduino*." I said, "What?"

I got an Arduino and went through all the tutorials. That community is so amazing. I made a couple of lights blink and felt like I'd just built



Hilary at the SXSW Music Festival, 2017.

Mitch McCarthy, Quentin Thomas-Oliver, Hep Svadja

a spaceship. As this happens, Hilary and I are starting to become a thing. So I packed all the stuff that fit in my pickup truck, and drove back to Austin, Texas, to be with Hilary. I show up and say: "Look at these blinking lights." She's like, "Okay, whatever." It doesn't seem to be that big of a deal. But I'm insisting, "No, this is a drum. I can make these lights blink. I know how to do it." But I wasn't as close as I thought.

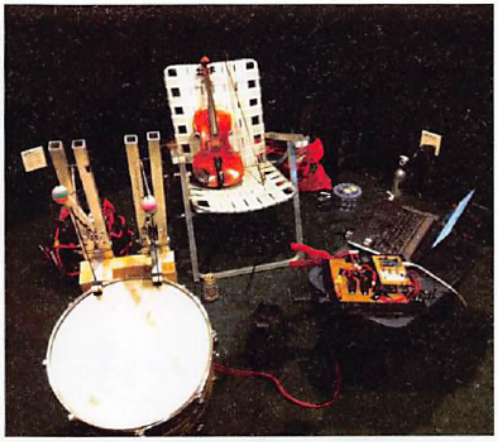
I found The Robot Group in Austin through Facebook. They were all really nice and welcoming — super eager to help me figure out the project. I said to one of the guys that what I really want to do is make my own Chuck E. Cheese band. He said, "Hold on!" and ran out to his car, returning with a set of mechanical eyes. Turns out he'd been for years one of the few official maintenance guys in the U.S. for the actual Chuck E. Cheese band! I became friends with them even more through the years because of course they're all going to Maker Faire, which was about to become a huge part of Ponytrap's life. But at this point, I hadn't yet heard of *Make:* magazine or Maker Faire.

So, I've got blinking lights, but it was trickier to turn that into physical motion than I anticipated. My friend Ken the engineer came to the rescue by teaching me about [ceramic] transistors, which are ceramic switches, and using them a bit like light switches send power to motors. So now, I've got the Arduino flipping the ceramic switches on and off, which then turns on the motors that make the sticks move. (Eventually, I swapped out the transistors for MOSFETS, which can be used

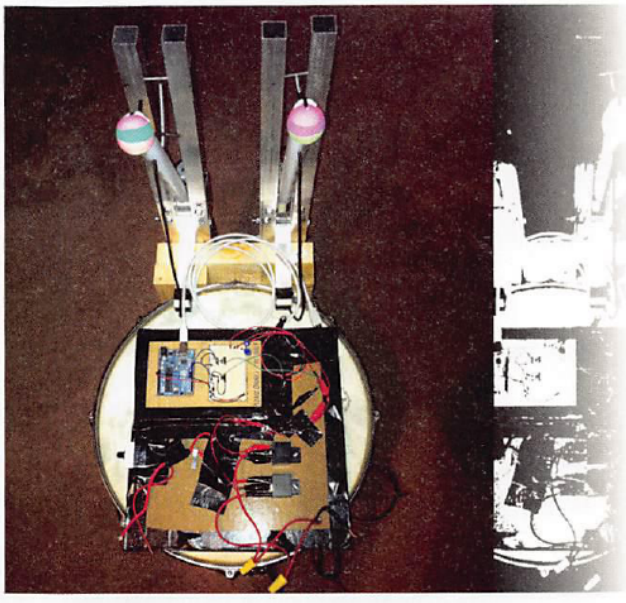
in a similar manner, but are a bit more efficient.) That was really the big moment — the first motor spinning — because now it's just a matter of scale.

On weekends, Hilary and I would go to secondhand stores to buy anything that moved. We would take things apart and try to understand how they functioned. I started working with little DC motors with pinion gears. (I didn't know what that was called at the time, but I do now.) I took a practice pad, a kind of a pretend drum, and taped two DC motors to the side. I drilled holes in two pencils, stuck them onto the pinion gears, and then just had the pencils tap by turning the motors on and off via the transistor. The first beat they played was "Happy Birthday."

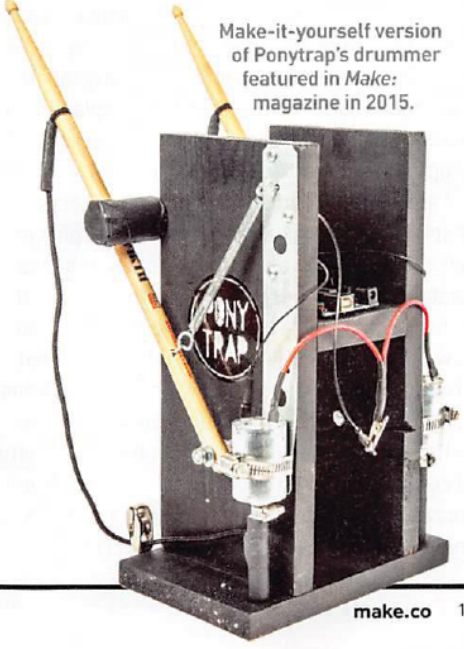
This whole evolution happened in 2011. We had a birthday party for the robot in our apartment, where I ran the first strong version of the robot at full blast. The neighbors weren't happy. I had a 16-inch tom with two arms for beaters. (I wrote a one-pager of a smaller version for *Make:* in 2015, makezine.com/projects/make-robotic-drum-using-arduino-uno.) That first truly functional version used a more powerful motor with a spindle and pulley. I got the idea from a kick drum pedal; using the mechanical advantage of the pulley to bring the beater down with force and then a spring return ... simple machines. I had a performance-worthy robot by early 2012. Then I realized that you can't do very much with just one. So I built another and we did our first show with just those two drums, me on viola, and Hilary playing cello.



Earliest working version of Ponytrap.



Detail of Ponytrap's original circuit.



Make-it-yourself version of Ponytrap's drummer featured in *Make:* magazine in 2015.

The pairing of drums and viola is an unusual combination of classical music with heavy metal rock music.

That's hard for me to explain. The idea was almost immediately in my head after the Paganini play: viola and two drummers doing a kind of dark, tribal music inspired by the hardcore music that I loved and the modern classical stuff I'd learned at school, like Bartók and Hindemith. But I can't really explain *why* it made sense to me, it just did. I never anticipated the robots, though!

In its full glory, Max is four and a half meters tall —13 or 14 feet tall. It's made up of six 16-inch toms, and a full-sized 24-inch kick drum that serves as its head, with cymbals for hands,



Max and Quentin at Maker Faire Bay Area.





Above: Ponytrap performing at the Rusthaven festival outside of Prague.

Right: Ponytrap in Maastricht.

roughly in the shape of a scarecrow, if you will. At least, that's the image in my head.

Max runs off of one Arduino Uno. It turns out you can hit those things pretty hard, they're very durable. I've learned some lessons about keeping it alive because I am absolutely maxing it out. There's 20–30 actuators on Max and I'm running them almost non-stop. The whole system still works like the first generation: Arduino → ceramic switch → motor.

Everything I do is 12-volt battery powered. I grew up as a redneck kid out in the cornfields of Indiana. I know how cars work and I learned how to make power out of 12-volt systems. I use two types of motor actuators. One is the mechanism that operates the lock inside a car's door; it's a small brush motor with a linear gear on it. They're cheap, durable, and pretty much universal. I can find the exact same replacement parts all over the world. These are what drive the drumsticks.

I also hit the drums with a different mechanism that I call a "flail." It's a motor that just spins and hits the drum with a couple of loosely attached beaters. One thing I learned over the years is that if the mechanism is too accurate, it sounds too ... well ... robotic. So I built in a bit of controlled chaos. The flails might hit once, twice, five times? I let it be a bit random and it creates a bit of flex in the sound. The sticks maintain the metronomic accuracy and the flails give it some flavor. The marching drumline sound that Max gets from that combination was just a happy accident.

Quentin talked about how he writes music for the robotic drums.

My robots do not listen. They do not improvise. In fact, that was a huge part of why I built them. Going back to the story about South by Southwest in 2011, the people I was working with, their whole thing is improvisation and they were incredibly talented musicians. But I could never get them to do the exact same thing twice and I found that really frustrating. The whole second set was pure improvisation, no structure allowed, which is definitely not my thing. I have to admit this, and I try not to be a person who becomes angry too much, but I was mad after that show ... very upset. I felt misled. So, as I was building the machines, it was actually a key part of how I built them that they don't improvise even a little.

I write the music out beforehand. It goes in as notation on a traditional five-line staff (the way that beats are written for trap-kits) and it doesn't change. There are all kinds of MIDI interfaces designed specifically to take music and translate it to a computer language, but back in 2011, they didn't do notation very well. It was important to me that my robots read music. When I'm thinking about music, I'm very much thinking in traditional notation as opposed to the more ubiquitous DJ style punch card thing. It's just the way that my brain has been kind of rewired by music school.

I had learned Visual Basic 6 in an office job that I worked barely long enough to realize that I am not the kind of person who is cut out to go into an office every day. But I learned a skill and used it to write the software that runs my robots.

Music notation software is a bit like a word processor for music. Once you've written a piece, it can make a file from a protocol called

Dale Dougherty, Ivo Osvald, Quentin Thomas-Oliver



Max entertaining the crowd at Maker Faire Brno.

ABC, which is an ASCII representation of the music notes. I'll write the robots' music in the software, export it into an ABC/ASCII file and then my software just translates that into serial commands for the Arduino. That's how it works — super simple.

Last year, Quentin received a commission to build a new machine from an arts and technology incubator in the Netherlands called Tetem.

I got together with Tetem in October 2024 and hashed out some ideas for a new machine. Hilary has a full-time, regular job and simply can't come play the cello at all the events as Ponytrap has gotten busier. So, the idea of the new machine is to be able to fill in when Hilary can't be there, and complement her when she is. I have generally been averse to building humanoids that play things. It is just not my thing. I didn't want to build, like, C-3PO with a cello. I ended up settling on something like a smaller version of Max. Instead of physical drums though, it's playing what are called trigger pads or electronic drums. The cool thing about these electronic drums is they don't have to sound like drums. They can be anything, all screams, or whatever — jet engine noises. So imagine it's a synthesizer, but instead of the keyboard, it's drums. I describe the new machine as a robotic DJ and call it Nova.

Nova has already done dozens of events with me and some with Max, but the thing I haven't had time to finish is a unified control system. The way they currently work together is that I hit play on two different laptops at the same time. If I miss by a fraction of a second, it's total chaos! Synchronization is definitely my next project.

During Covid, Quentin and Hilary left Austin and moved to Maastricht in the Netherlands.

It was mostly for the adventure. We just wanted to do something different. We have built up a good network of friends here in Maastricht, and more generally, this part of the world. I'm lucky to have a large shop to keep making, and we very much enjoy our new hometown.

Since moving to Europe, I've essentially been touring full-time and I love it. I have said many times over the years that the only job I ever really wanted was rockstar. I don't even know what that word means anymore but I am playing my own music on my own terms and having a blast.

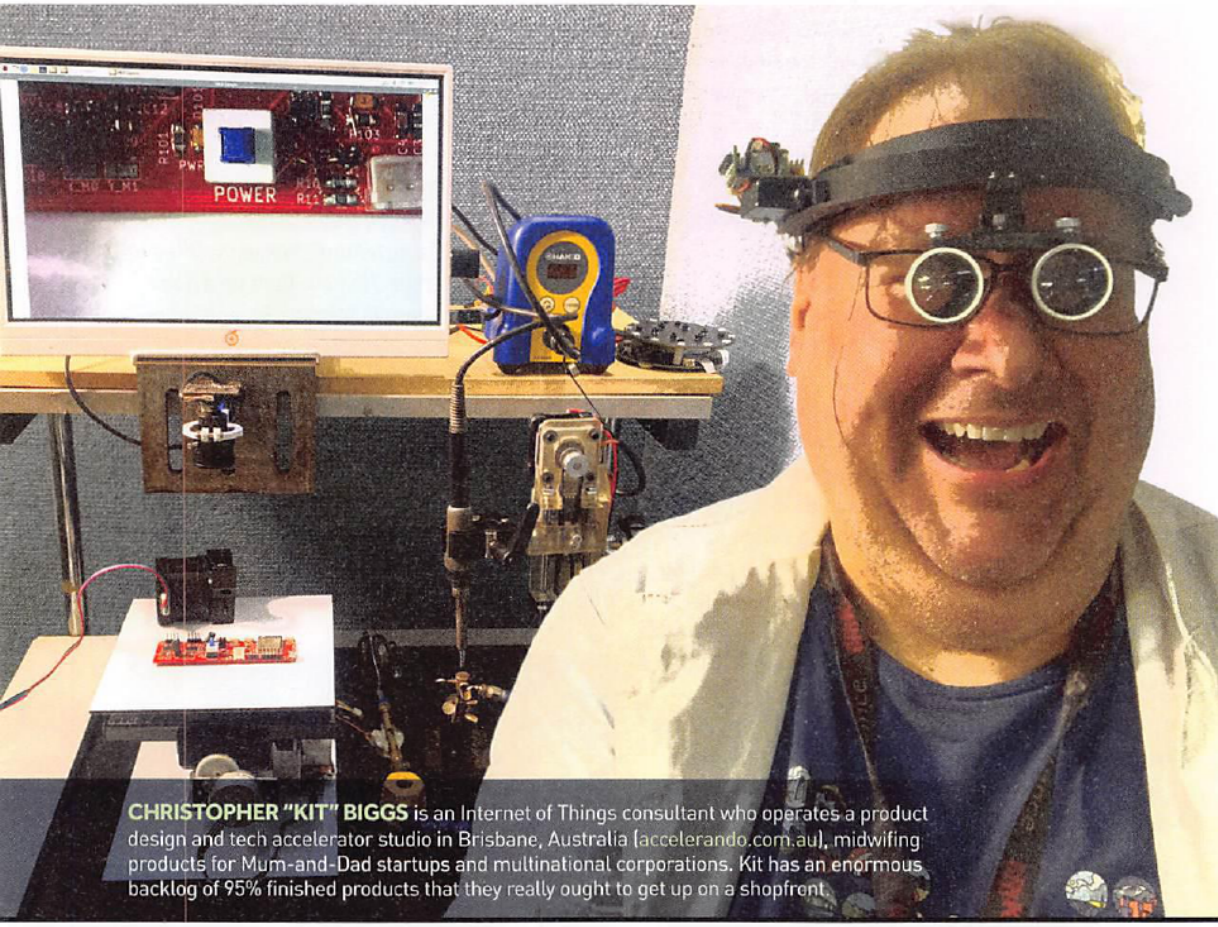
PONYTRAP ON VIDEO:

- Max and Quentin performing at Maker Faire Bay Area: youtu.be/9Be3pePMGtk
- Ponytrap's entry for SXSW Film Festival 2018: youtu.be/ojkIqC3zVKk
- The evolution of Ponytrap's drum machines: youtu.be/kQbHPeVCgpo

MY VOICE-CONTROLLED WORKBENCH

I automated my solder station, microscope, and magnifying visor using ESP32s and free home automation tools

Written and photographed by Kit Biggs



CHRISTOPHER "KIT" BIGGS is an Internet of Things consultant who operates a product design and tech accelerator studio in Brisbane, Australia (accelerando.com.au), midwifing products for Mum-and-Dad startups and multinational corporations. Kit has an enormous backlog of 95% finished products that they really ought to get up on a shopfront.

I think a lot about what kind of future I want to live in.

On the one hand we have *Star Trek*, which says that in the future whenever we want to do anything we have to poke at a screen or command a disembodied computer. "Tea, Earl Grey, hot." In the *Star Trek* future, the computer becomes an unavoidable presence in our lives. Everything we do is mediated through it.

On the other hand we have Frank Herbert's science fiction novel *Dune* and its various motion picture interpretations. The world of *Dune*, 20,000 years in our future, is a society that already had their full-service AI singularity, decided they didn't much like it, and chose another path, outlawing AI. The household items of *Dune* are smart, some of them even hold conversations, but interactions are tactile and subtle, and they do not attempt personhood.

And of course, form follows function. In *Star Trek* everything is a rectangle, devices make piercing beeps, and the computers all sound like they're vaguely annoyed. In the Duneiverse, at least for the aristocratic viewpoint characters, everything is elegant, deferential, and pleasant to hold. This is a future I can live in.

"Technology is stuff that doesn't work yet."
—Bran Ferren (via Douglas Adams)

I love this quote from artist and technologist Bran Ferren, who was making the point that when things just work, we stop calling them technology. As novelist and essayist Douglas Adams noted when riffing on Ferren, nobody calls chairs technology — despite the amount of engineering sophistication that goes into them — because they almost never go rogue for no visible reason.

MEET SERENA

All right, that's enough waffle. What this article is about is technology (or not-technology) that exists to unobtrusively and efficiently do one job, well. My day job is electronic design and troubleshooting, which means I'm never more than six feet from a soldering iron, and my magnifying visor is always in arm's reach. So I set out to build a set of tools that help me with certain tasks, like troubleshooting a circuit board, where I never seem to have enough hands. These tools are:

- **a motorized microscope stage**
- **a hands-free solder-wire feeder**
- **a retractable soldering iron** for soldering and desoldering ribbon connectors, and
- **a headset with adjustable magnification and lighting.**

Oh, and it's all voice activated.

I call it "Serena." This might sound like a lot of work, but really I wrote very little code to achieve it, because I leveraged what is already available. I use the voice control toolkit from Espressif, motor control software from the FluidNC project, and a general-purpose IoT appliance framework from ESPHome, and the coordination and scripting support is provided by Home Assistant.



TALK NERDY TO ME: A corner of my voice-controlled workbench showing (clockwise from front left) servo-actuated magnifying visor, microscope stage, soldering iron arm, and solder feeder.

VOICE CONTROL ON ESP32

For voice control I'm using the **Espressif ESP32** microcontroller and **Espressif's open source voice recognition toolkit**. Why not use the voice recognition from Home Assistant? I like it for home and office automation, but for body-worn or fingertip devices I find the response is not quite fast enough.

At this moment in late 2025, the state of voice recognition is that you can have either:

- **open-ended conversation** with a cloud-hosted chatbot/language model, or
- **continuous speech recognition** with a GPU in a PC or a high-end single-board computer, or
- **standalone phrase recognition** on a microcontroller.

I'm not interested in being tied to the cloud, and there isn't room to properly ridicule LLM chatbots here, so those are off the table. I've tried setting up my own in-house voice assistant using the Home Assistant Voice developer preview device. This device listens for a ***wake word***, then passes whatever you say next to a PC for processing. It works okay for home automation but there's still a noticeable lag from speaking a command to having it carried out; it's not responsive enough to be faster than putting down what you're holding, picking up a different tool, and directly operating that tool.

The thing I like about standalone voice agents is that they are fast. They do achieve this speed by sacrificing flexibility — they can only recognize a list of a hundred or so pre-configured phrases — but a limited palette of command phrases is exactly what I need for a hands-free power tool.

It's absolutely feasible to build a voice-controlled tool with a single ESP microcontroller, furnishing the tool with a microphone and doing the speech processing and the tool operation in a single program. However, when you have a set of tools within earshot, I think it makes more sense to have a single device handle the sound (perhaps a smartwatch, or a *Star Trek* "combadge," or a benchtop puck like those from Amazon, Apple, Google, et al.), and have the voice processor route the recognized command to the tool being commanded. Since all the Espressif microcontrollers have built in Wi-Fi and Bluetooth, they are born communicators. My approach is to have a number of task-based vocabularies that are appropriate to what I'm doing right now. If I'm sitting at a microscope, I want the system to be listening for microscope commands; and similarly when I'm holding a soldering iron or wearing a magnifying visor. It's conceivable I might be doing all of these at once.

Espressif's voice toolkit revels in the



VOICE PROCESSOR: This Espressif Korvo development board has a microphone array, buttons, lights, and battery charger. It can act as a standalone voice interface for "phrase recognition" or cooperate with a more powerful computer for full speech recognition.

catchy name **ESP-Skainet**, which I guess is a portmanteau of *speech kit AI neural network*, and reader, I am embarrassed to say that I only just now got the "SkyNet" joke as I typed this article. A big chunk of the necessary code is actually in a lower-level toolkit called **ESP-SR**. The Git repositories for these two products contain some quite good documentation and some — alas, very poorly structured — examples. The key components of Skainet are:

- **audio processing and microphone array handling;** Espressif calls this the Audio Front End
- **wake word detection** a la "Alexa," "hey Siri," etc.
- **phrase recognition**
- **audio output** for answering commands

Wake word processing is about efficiency and privacy. A cloud-based voice assistant can't be transmitting every word it hears to the cloud, so it waits until it is addressed by name before beginning to burn CPU cycles and network bytes trying to understand speech.

I call my voice agent Serena, after a *Dune* character who instigates the Butlerian Jihad against the enslavement of humanity by artificial intelligence. The phrases that Serena understands break down into three categories:

- **global commands** that are always active,
- **meta-commands** that modify the active task context, and
- **task-specific commands** that are active in the current task context(s).

WHEN SERENA IS WORKING, IT IS IN PHRASE RECOGNITION MODE AT ALL TIMES; THE SYSTEM ESSENTIALLY LISTENS CONSTANTLY FOR CONFIGURED PHRASES.

The always-active global commands include "list commands," "what was I doing?" and "stop listening." Meta-commands are task names like "soldering," "microscope," or "visor." Task-oriented commands (say, for the microscope) resemble "track left," "go back," "zoom and enhance," and if you're thinking of the voice-controlled photo enhancing scene from the classic sci-fi movie *Blade Runner* right now, this is not a coincidence!

If you do want to use a wake word, be aware that this is perhaps the least flexible part of the toolkit. Each wake word is a separate machine learning model; Skainet comes with a limited number of wake models in English and Chinese. Espressif will train a custom wake model for a fee, however in recent times it has become possible to do this yourself via some open source wake word toolkits.

My tools, since they are not cloud-based and need not be concerned about data usage, skip the wake word step entirely. When Serena is working, it is in phrase recognition mode at all times; the system essentially listens constantly for configured phrases. The trick that makes this feasible is that the list of configured phrases changes based on context: if I say "I'm soldering," then the soldering phrases are loaded, and so on. You are limited to 100 or so phrases, and the system doesn't recognize numbers ("two one" won't be understood as 21), so any numbers you need it to understand must be explicity loaded as phrases ("twenty-one").

MQTT AS A COMMAND FABRIC

Having come at this problem via home automation technologies, I'm leveraging a technique from that space: ***message brokers***, which simplify communication by breaking it into the acts of ***publishing*** a message, and ***subscribing*** to particular messages. ***MQTT*** is one message broker protocol that is heavily used in IoT and home automation.

A Raspberry Pi in my lab runs an MQTT message broker. When my voice agent recognizes a phrase, it publishes that phrase to the message broker, without needing to care which devices may act on it. An MQTT publish operation consists of a ***topic*** and a ***payload***. In this case the topic resembles **`serena/phrase/TASKNAME`** and the payload is the phrase recognized. Tool devices simply subscribe to topics (or topic patterns) that relate to their job — the solder feeder only needs to be aware of the "feed solder" commands via subscribing to the **`serena/phrase/soldering`** topic. The microscope only concerns itself with movement commands. Some of the tools know when they are picked up or otherwise activated, so the act of picking up a soldering iron might publish a "tool was activated" message (topic **`serena/task/activate`**), and the voice agent, having subscribed to activation messages, can adjust its table of active phrases.

That's really all there is to Serena: a set of task vocabularies, some housekeeping commands, and a way to load or unload the active phrase tables.

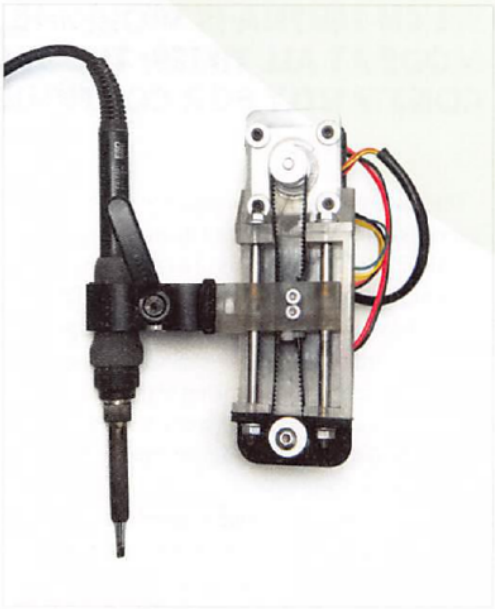
In the source code (*commandset.h*) a data type **`commandset_t`** is defined, which has a name, a parent set name, and a list of commands. Serena maintains a pointer to the active command set, whose "parent" may be another set of task commands, or the global commands. Essentially this is a list of phrase-sets that have been loaded into Skainet. Serena handles some commands internally (loading and unloading command-sets, sleeping and waking), and just passes the rest to MQTT. Adapting Serena for your own purposes simply consists of configuring how to connect to your message broker, and defining the sets of commands for which you want to listen (see the documentation at github.com/unixbigot/serena).

MY VOICE CONTROLLED DEVICES

Let's look at the three benchtop devices that I've designed to be voice controlled.



SOLDERING IRON ELECTRONICS: an off-the-shelf "backpack motor driver" (top) and ESP32 microcontroller board (a TTGO T7, bottom), and my own "Betsy" single-axis CNC module.



My 3D printed linear actuator consists of some rails and bushings, a stepper motor, and a belt. (Typically a screw-type actuator is too slow for a soldering station.) The soldering iron is gripped by a photography clamp.

SOLDERING STATION

Soldering is a task that often needs three hands — one to hold the soldering iron, another to hold a wire or component, and a third to hold the solder wire — and if your circuit board is not held in a vise or other clamp, you maybe need four hands. This is a well-recognized problem in industry, and there are tools used in factories for doing repetitive tasks like soldering wires to connectors (plugs and sockets). Rather than using a vise, these devices have a platform, often shaped to accept a particular connector. The operator holds the connector in place on the platform with one hand and positions the wire using their other hand. The soldering iron and solder wire are robotically controlled, usually triggered by a foot pedal.

My version of this separates the assembly into three appliances — a soldering iron arm, a solder feeder, and an actuator which can be either Serena or a cordless foot pedal, both interacting via MQTT. So there are six ways that any two of these four items can interact, but with the MQTT broker acting as "virtual wire" we need only four interactions: each device with the broker. As the number of devices goes up, this reduction in complexity — N interactions instead of $C(N,2)$ interactions — becomes a huge win.

The problem of moving a soldering iron up and down is just (haha it's never "just") motion control. Moving tools through space to meet workpieces is known as ***computer numerical control (CNC)*** and there are excellent open source CNC implementations out there that we can (ab)use to control our soldering iron. I'm using the **FluidNC** motion control software which runs on ESP32. If you have a spare Arduino, the older GRBL software will run on that.

These CNC firmwares interpret a motion control language called **G-code** which they receive over serial port or network. I'm using some glue logic on my Home Assistant server to listen for MQTT events and pass commands to FluidNC over the network, but I plan to add MQTT support directly to FluidNC to simplify this arrangement. FluidNC supports ***macros***, small canned programs that are stored in flash memory on the device, and a macro can be invoked over the network or via an electrical input. For my soldering iron there's a macro that performs

FOR MY SOLDERING IRON THERE'S A MACRO THAT PERFORMS "LOWER THE IRON, WAIT A SHORT MOMENT, RETRACT THE IRON," AND OTHER MACROS THAT ADJUST THE MOVEMENT DISTANCE AND WAIT TIME.

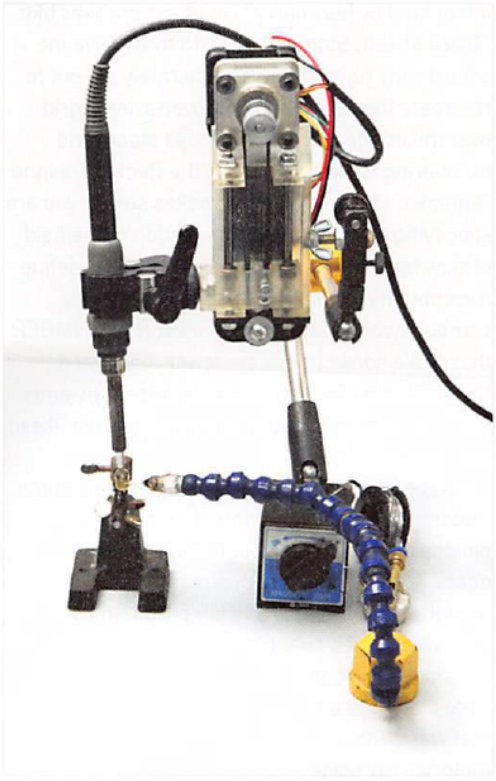
"lower the iron, wait a short moment, retract the iron," and other macros that increase/decrease the movement distance and wait time. All those macros can be triggered by voice events or button presses.

The second part of doing semi-automatic soldering is feeding some solder wire at the same moment as the iron lowers, so that the iron heats the joint, melts the solder, and then retracts, leaving the solder to solidify. FluidNC supports six axes of movement, so a single instance could do everything (iron movement as one axis, solder feed as another) but I've made my devices separate so that I can reuse the solder feeder when doing other hand soldering tasks. My solder is fed down the middle of a flexible arm so I can position the arm over my work, or even clip it to my iron, and have it feed solder when I need it. I have three sizes of solder (0.2, 0.5, and 0.8mm) at my workbench, so there are actually three motors feeding wire into a bundle of tubes.

To run the motion control firmware, you can use a number of Arduino-compatible CNC shields, or your ESP can communicate directly to an off-the-shelf discrete stepper motor controller. I've designed a few different ESP-compatible motor control boards — a two-axis controller for a battery-powered pen plotter, a single-axis controller for lottery-ball barrels (yes, the ones you might see on TV), and finally a stackable unlimited-axis controller that I've used to build several CNC machines. These designs are all open source; you can have them made using a PCB assembly service, or obtain boards from me at tindie.com/stores/accelerando.



Solder feeder made from a small stepper motor and some 3D printer spare parts. A brass pinion drives the solder wire by friction, within a brass tube glued to the motor's mounting plate. The solder is fed through a flexible arm intended for milling machine coolant; the tip has an old 3D printer nozzle drilled out to feed the solder.



Operator's-eye-view of the soldering station. An old "third hand" clamp is repurposed as a saddle to hold connectors while the wire is positioned from left. Solder is fed from in front and the iron descends from above, on command.

REMEMBER THE SCENE FROM *BLADE RUNNER* WHERE DECKARD, THE DETECTIVE, ENHANCES A PHOTOGRAPH BY UTTERING "TRACK 45 LEFT. STOP. ENHANCE 15 TO 23"? I DELIBERATELY SET OUT TO RE-CREATE THIS TECHNOLOGY.

MICROSCOPE STAGE

I designed a 3D-printed microscope stage long ago, large enough to fit a 20cm×20cm circuit board (and customizable to your size requirements). However I'm currently using a compact macrophotography tripod head since I broke my printed stage during a workshop reorg.

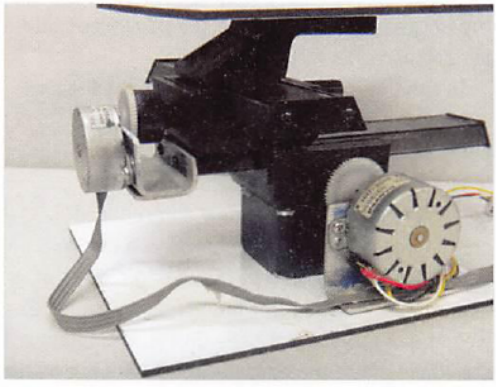
Whether you use a self-built or off-the-shelf stage, once again you can achieve no-code automation by treating your stage as a simple X-and-Y-axis CNC machine.

Remember the scene from *Blade Runner* where Deckard, the detective, enhances a photograph left behind by his quarry? He utters phrases like, "Track 45 left. Stop. Enhance 15 to 23. Give me a hard copy right there." I deliberately set out to re-create this technology. By overlaying a grid over the image of the microscope stage, and numbering each grid square, the Deckard-esque "enhance 15 to 23" actually makes sense: you are specifying a rectangular sub-section of the field of view by nominating two of its corners. I define a vocabulary for Serena that includes all the combinations of "enhance NUMBER to NUMBER" that make sense (there are fewer than you'd think). This makes it possible to voice-pan-and-zoom on a microscope camera image "just like in the movies."

On the hardware side, I'm using some stepper motors salvaged from printers that have gear-pinions on the shaft. I bought bags of plastic gears with the same tooth size (metric "0.5 modulus" in this case) and simply glued two gear wheels of a suitable size to the knobs on the stage. I mounted one of the motors to the base plate and the other to the stage ways, and that was all the modification I needed to do to motorize my stage.



MICROSCOPE STAGE: A macrophotography tripod head has been repurposed, with the addition of a deck plate and some motors salvaged from dead printers, as an X-Y microscope stage.



The Y-axis motor mounts to the base plate, and the X-axis motor rides on the platform carriage, attached to its end plate. Their small 12-tooth pinions mesh with larger gears, from a hobby assortment pack, glued onto the original control knobs.

MAGNIFYING VISOR

The last part of my voice-controlled workbench is a magnifying visor. Now, I could treat this as

another weird CNC machine, but I took a different tack here and used **ESPHome firmware**, which is a no-code/low-code system for building IoT appliances (lighting and appliance control).

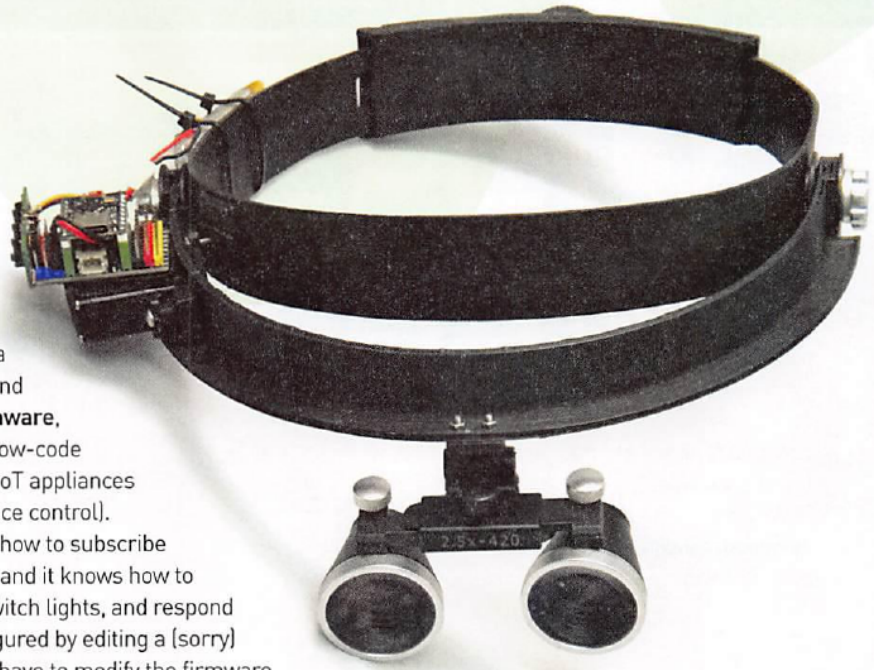
ESPHome knows how to subscribe to MQTT messages, and it knows how to run servomotors, switch lights, and respond to buttons. It's configured by editing a (sorry) YAML file; you don't have to modify the firmware at all to build a custom device. My configuration file exposes a number value (ESPHome's generic output for controls that don't fit the common metaphor of switch/light/motor) that represents the angle of the visor lift servo.

Once again some logic translates voice events (say "visor up") to motor controls (servo position **-50**); this logic could reside on the HomeAssistant server, but with sufficiently frightening YAML-fu it can be implemented on-device. Activating the built-in headlights and monocle lenses is a bread-and-butter task for ESPHome; this is what it was designed for. I don't always make my software do unnatural things.

MOVE IN AND ENHANCE!

I will literally talk all day about these technologies; you can find some of my conference presentations via my website to learn more (christopher.biggs.id.au).

In my day job as a product designer and prototyper, I use open source tooling as much as possible, so explore my personal and consulting GitHub accounts (unixbigot and accelerando-consulting) to find software and hardware that can help you to be commanding your devices to "feed solder" or "give me a hard copy right there" in no time.



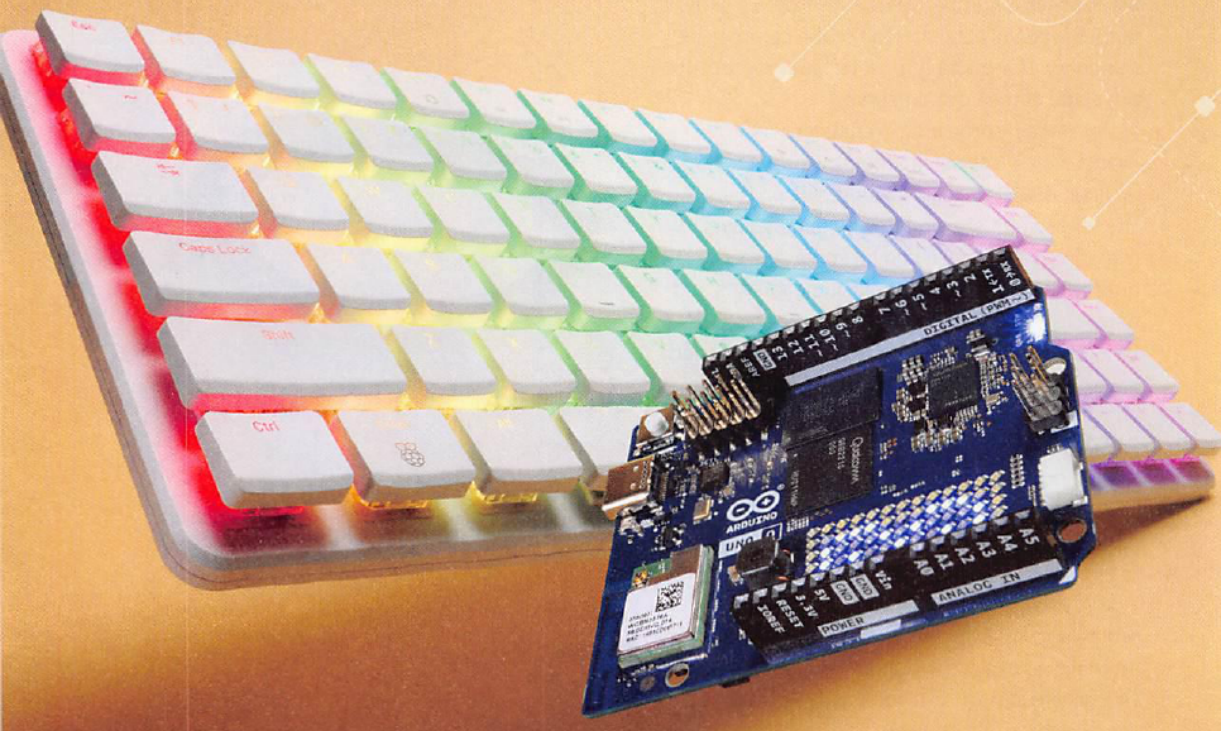


MAGNIFYING VISOR: The head strap is salvaged from a commercial visor that I broke; the lenses are a dental loupe set with a 40cm working distance. The servomotor and ESP32 board on top are powered by a lithium battery and 5V boost circuit. The servo arm has a pin which rides in a slot in a 3D-printed guide plate attached to the strap. The semicircular platform holding servo, loupe, and headlights (not visible here) is also 3D printed.

SUPER [TINY] COMPUTERS

Arduino and Raspberry Pi's surprising new releases put real computing power where we've never seen it before

Written by David J. Groom





DAVID J. GROOM loves writing code that you can touch. If he's not hacking on wearables, he's building a companion bot, growing his extensive collection of dev boards, or hacking on 90s DOS-based palmtops. Find him online at ishjr.com

I remember getting my first Arduino in 2007 (a Diecimila "Prototype Limited Edition"), excitedly plugging it into my computer, and programming it as a brain transplant for my Roomba (thanks, todbot!). I vividly recall the arrival of my first Raspberry Pi (Model B, code 2, with just 256MB of RAM!), and my Automotive EE friends making fun of it for using an SD card for storage, which they estimated would wear out after a year or two. These devices allowed me to harness my computer to write code for devices I could touch, rather than being trapped in a screen or on a server.

The Pi was a single-board computer (SBC) but it wasn't really powerful enough to use as a desktop, versus SSHing in remotely to interact with it (though that interaction would give me the confidence to eventually switch to Linux as my daily driver.) Still, the Pi kept getting better, so I excitedly tested each iteration as a daily driver itself, with the Raspberry Pi 3 Model B+ just starting to tease at viability, then the 4 being reasonably serviceable, and finally the 5 doing an extremely convincing job. Meanwhile, 2020's Pi 400 tickled me with its all-in-one keyboard form factor (reminiscent of my beloved BBC Master breadbin, see "Old Tech, New Thrill," *Make:* Volume 89, page 24).

But it's the new Raspberry Pi 500+, featured on the cover of this issue, that brings together the power and utility, for the true embodiment of that "plug it in the telly and go" experience that made me fall in love with computers in the first place. No flashing SD cards, no faffing about with a million cables, just HDMI, USB-C power, a mouse, and a whole universe of learning and entertainment at your (RGB-bathed) fingertips.

Raspberry Pi might be the big name in SBCs, (though not the only one — see "Indie Boards" on page 32) but out of absolutely nowhere, Arduino, newly acquired by Qualcomm, blew our minds with the new Uno Q, an SBC that asks "What would Xzibit do if he designed a dev board?" The obvious answer: they put an Arduino *in* your Arduino, so you can program your Arduino MCU while you program *with* your Arduino SBC. In short, in 2005 you programmed your Arduino with a computer; in 2025 your Arduino *is* the computer!

Looking back at our boards coverage in recent years, the supply chain disruptions that threatened the whole ecosystem now feel like a distant memory, and a thriving environment of innovative and specialized boards has emerged (see *Make: Guide to Boards* packaged with this issue!). But new threats loom, in the form of unhinged, unpredictable tariffs, which impinge on almost every niche interest, hobby, and small business. While amassing dozens of boards this year to inform our guide, I received a $365 bill from DHL due to products being incorrectly marked (I was able to refuse it and receive the goods duty-free upon re-shipping), and another $56 surprise that I ended up having to pay in the interest of time. And these are boards I receive for free! Small businesses and makers who need to procure items from overseas are hurting and in many cases giving up, at least until things stabilize or become saner. (See my coverage of TariffsAreBullshit.com in the *Make Things* newsletter at makezine.com/go/maker-tariffs.)

We're seeing MicroPython and CircuitPython emerge as first-class citizens with newer boards, rather than an afterthought or hack, with the Arduino Uno Q's new App Lab throwing desktop Python into the mix too, and directly interfacing it with traditional sketches. LoRa, well over a decade old, is having a banner year thanks to Meshtastic (see page 54), which utilizes the long-range radio networking protocol to create an encrypted off-grid messaging system. And two ultra-affordable microcontrollers keep bringing us fantastic projects: the RP2040 (build a pocket video synth, page 42; ADSBee plane tracking, page 60; and MicroPython-powered wheeled robot, page 74) and the ESP32 (try the Oxocard Connect, page 48, and voice controlled workbench, page 22.)

Our Guide to Boards is evolving too, with new music and cosplay board specialists joining some familiar returning authors. As always, I'm keen to hear your feedback on our choice of boards and categories — get in touch at editor@makezine.com and let me know your favorite new and upcoming boards, and your predictions for next year!

Mark Madeo, Adobe Stock-Bibi

INDIE BOARDS

Small teams at **BeagleBoard** and **IceWhale** use tiny computers to make a big impact

Written by David J. Groom



ZimaBoard 2

While the big guns like Arduino and Raspberry Pi may steal all the headlines (see "Super Tiny Computers," page 30), there are a huge variety of lesser known but equally, and sometimes even more, exciting single-board computers (SBCs) cropping up from all over the world. Boards like the Pi-shaped D-Robotics RDK X5 have caught us off guard with their impressive compute-to-price ratio, not to mention the outrageously cost-effective Luckfox Lyra family of boards with their many form factors (see "New & Notable," in the *Guide to Boards* insert). While early Arduinos were essentially ATmega8 breakout boards with the minimum viable circuitry to program them, the 64-bit application processors that power today's SBCs are vastly more complex than those simple 8-bit microcontrollers. And even if creating your own SBC may be a feasible feat, success often requires fostering an ecosystem and community around it, beyond the hardware

Adobe Stock-Bi

itself. I'm fascinated by what some surprisingly tiny teams are accomplishing in this space, so I decided to sit down with two particularly poignant examples to better understand where they came from, how they got where they are today, and what the future holds.



PocketBeagle 2

SMALL TEAM, BIG GOALS

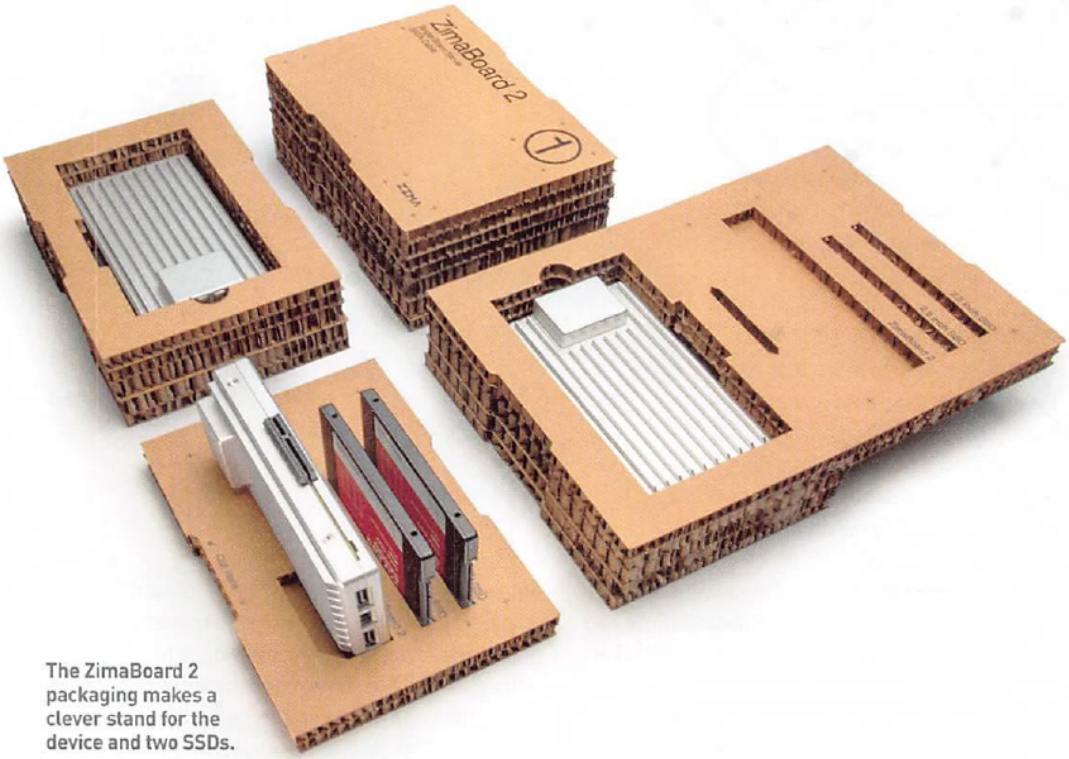
I was astonished to learn that the BeagleBoard.org team has just four full-time staffers, during a call with the Michigan-based nonprofit's president/co-founder/community manager/software cat herder, Jason Kridner. By the time the first Raspberry Pi was released in 2012, multiple generations of BeagleBoards had already found their way into the hands of makers, educators, and professional engineers.

Hatched in 2007 while Kridner and co-founder Gerald Coley were working at Texas Instruments supporting the OMAP family of Arm-based, high-performance application processors, the project began as a way to explore how to engage the community around their latest high-performance OMAP 3 System-on-Chip (SoC). What emerged, and shipped in July 2008, was a low-cost, USB-powered, tiny computer, at a time when such a thing was far more unique than it is today. The original BeagleBoard was succeeded by the more powerful BeagleBoard xM before the stripped-down BeagleBone was released in 2011, with a white PCB and the familiar Altoids-tin shape and "cape" ecosystem, akin to Arduino "shields." A more powerful BeagleBone Black launched in 2013, in a landscape that now included the original Raspberry Pi.

Kridner described the BeagleBone to me as a part of a "21st-century survival kit" — where once you might have stuffed your Altoids tin with matches, Band-Aids, and water purification tablets, our modern-day existence revolves a lot more around technology, and a minimalist computer might find greater utility in your life.

With an open-source product and no for-profit arm (in contrast to Raspberry Pi), BeagleBoard lets you dive into whatever layer you want to explore, from the Linux kernel to maker projects to hardware design. Their mission is not to create cheap computers, but rather computers that are accessible. And if you prototype a project using the PocketBeagle 2 for example, there is nothing to stop you designing your final board around the same TI AM6254 SoC. BeagleBoard sees nothing from sales of those processors — unlike Raspberry Pi, where the only practical way to get that Broadcom chip is by embedding a compute module like the CM5. While Raspberry Pi may no longer be referred to as "BeagleBoard but cheaper," Kridner says that the competitor's popularity is the best thing that happened to their platform, elevating the whole SBC segment.

Building on a large community of makers, kernel hackers, and professional engineers, BeagleBoard also participates in Google Summer of Code internships, attends Linux Foundation conferences, and has an extremely active web forum, as well as, to Jason's open-source-loving chagrin, an active Discord server (he still prefers, and monitors, the foundation's Internet Relay Chat (IRC) channel!). No special strategy was used to stimulate the community; instead, individuals' own enthusiasm created a self-selected network of passionate contributors. An all-volunteer board helps steer the organization, along with Jason, CEO Christi Long, and two developer/documentation author/do-it-alls



The ZimaBoard 2 packaging makes a clever stand for the device and two SSDs.

Deepak Khatri and Ayush Singh; DigiKey's Robert Nelson is the "heartbeat" of their community forums.

BeagleBoard's openness and flexibility have resulted in a number of exciting projects and products being built around the platform. In addition to the Bela high-performance audio systems ("New & Notable," GB2), projects like Kulp Lights (kulplights.com) have evolved from BeagleBone capes to entire ecosystems unto themselves, in this case providing the foundation for immense LED light shows. BeagleBoard's own TechLab Cape for the PocketBeagle helps makers and engineers learn file-based embedded Linux, as well as kernel development using training from The Linux Foundation. In addition to countless capes, MikroBUS support enables the use of hundreds of Click boards (well, two at a time!). By providing a minimalist platform based around standard buses and then "getting out of the way," BeagleBoard have created a supple platform that makers, developers, educators, and professionals can use as a springboard in executing their dreams.

A CLOUD SERVER FOR EVERY HOME

Another emerging leviathan in the SBC segment is IceWhale, creator of the new ZimaBoard 2. I had an incredibly inspiring and delightful call with Lauren Pan and his team, who I discovered are also huge fans of *Make:*! After developing the LattePanda series of x86-based SBCs at DFRobot, Pan left in 2019 to explore new emerging trends. Looking to the homebrew computers of the 70s, then the personal computers of the 80s, Pan mused that perhaps the 2020s could be the era of the personal cloud — taking all your data, media, smart home functionality, and digital life back from giant cloud platforms onto a server that you control.

Thus in 2021, Pan founded IceWhale, the name suggesting purity as well as great size and intelligence, with the goal of creating a community rather than a company — a team that loves to learn, build, and do. Observing that 20% of Raspberry Pis are used for home/media servers, Pan's first inclination was to build a cheaper NAS — but the product fell flat with friends he shared it with. He started reading

thousands of Reddit and other posts, connecting with YouTubers, and building a much deeper understanding of the landscape. Self-hosting was just starting to gain popularity, though homelabs were still uncommon — what if he could build a $100 home server that can function as a media server, storage, Git server, and run scripts that help automate users' daily lives, with copious external ports and a finished appearance rather than a bare board? That first year, 1,880 backers (Pan recites from memory!) pledged almost 2.5MM HKD ($315,000) to bring the "ZimaBoard Single Board Server for Creators" to life.

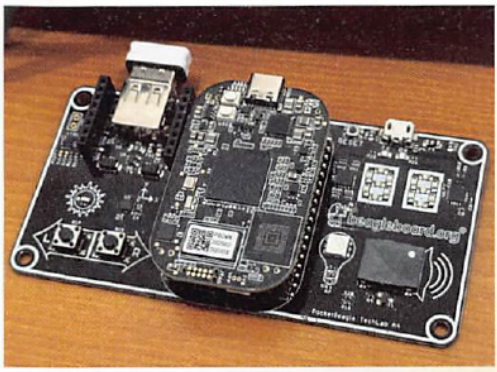
Based on Intel Celeron chips, the ZimaBoard was capable of running Windows, but the team chose to build a new OS for their personal cloud from scratch, the Debian-based CasaOS. It was featured on GitHub's trending repositories list, and now has an incredible 31.9K stars at the time of writing. The one-line install isn't limited to ZimaBoards, and can run on Intel NUCs, the old computer gathering dust in your closet, and even their direct competitor, Raspberry Pi!

Their most recent Kickstarter, centered around the $169 ($199 MSRP) N150-based ZimaBoard 2, was even more successful, with 2,149 backers pledging 5.3MM HKD. The new board is 3x faster, with up to 16GB of DDR5 memory, dual 2.5GbE LAN, SATA and USB 3.1 ports, MiniDisplayport 1.4 for 4K video, and perhaps most excitingly, a PCIe 3.0 interface, allowing "desktop" cards like GPUs to be added to the device. One of my favorite innovations in the advanced preview I received is that the cardboard packaging can be used to hold the board and two 2.5" SSDs, instead of the usual sprawling mess that most SBCs and their accessories end up creating on our desks.

The ZimaBoard is born of the community. Lauren says that 99% of people might not like your idea, but if you connect with the 1% who do, you can form a passionate base — which he experienced the moment their Discord server reached 10,000 members. He realized then that ZimaBoard exists because of them, for them, and that the community embracing what his team built could help grow that passionate 1% of believers into maybe 10% of the home server community.

In addition to CasaOS, IceWhale has launched a more compact standalone ZimaOS, which also runs on third party hardware, including QNAP and Synology NASs. Their latest hardware and software are fulfilling that dream of a fast local server that makes it easy to deploy whatever you need for your life without reliance on the cloud. Longer term, they hope to democratize computing again, releasing it from the clutches of a handful of giant cloud companies. I asked about the name ZimaBoard, joking about the 90s malt beverage, and Lauren shared a beautiful story of being inspired by an episode of *Love, Death & Robots* called "Zima Blue," about a humble pool-cleaning robot that was hacked and improved to eventually become a profound artist and visionary. [Spoiler alert!] In his final performance, he jumps back into the pool, shedding his complex augmentations, and returns to the simple life of pool cleaning.

Today's enshittified cloud-centric computing experience feels a million miles away from the ethos of the 8-bit BBC I grew up with, created by a handful of passionate nerds in a turkey shed in Cambridge. Hopefully small, passionate teams creating tiny computers like the BeagleBoard and ZimaBoard will help unshackle us from the capitalistic clouds that threaten to consume our digital lives and beyond, and bring us back to that pure computing experience.



The PocketBeagle TechLab Cape was designed to help programming novices learn Linux kernel hacking.



DAVID J. GROOM loves writing code that you can touch. If he's not hacking on wearables, he's building companion bots, growing his collection of dev boards, or hacking on 90s DOS-based palmtops. Find him online at ishjr.com

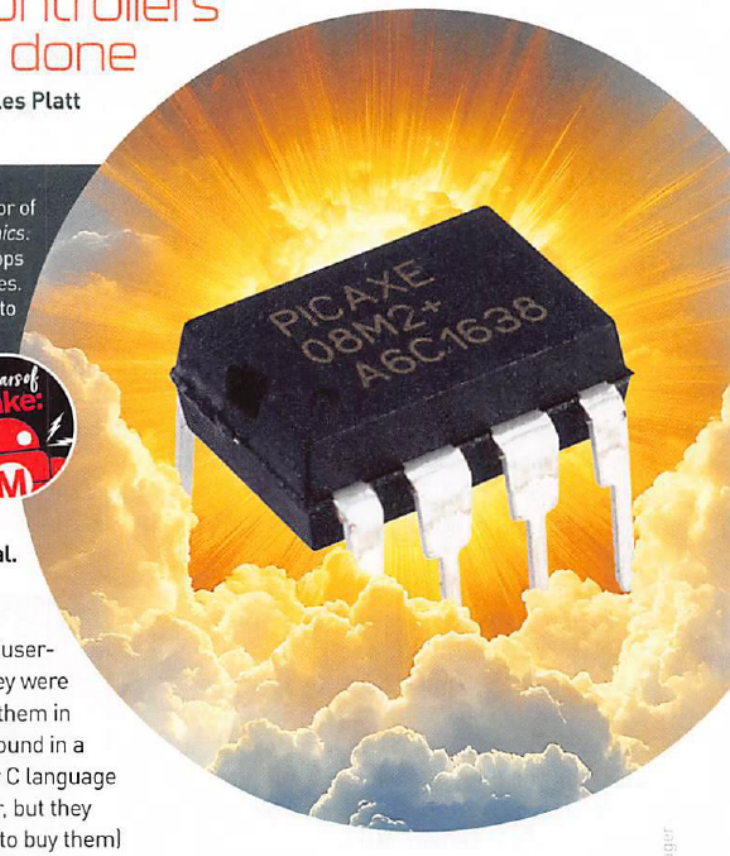
HOMAGE TO THE AXE

Cheap but mighty, Picaxe microcontrollers still get the job done

Written and photographed by Charles Platt



CHARLES PLATT is the author of the bestselling *Make: Electronics*. He lives in Arizona and develops gear for Biostasis Technologies. This is his 100th contribution to *Make:*. makershed.com/platt



Adobe Stock-Bibi and BAIVECTOR, Clive Seager

Normally I prefer to avoid mentioning product names, but in this case I am going to break my own rule, because I cannot remain diplomatically neutral.

First, I need to provide some orientation. There was a time when microcontrollers were not remotely "user-friendly." Back in the 1980s, when they were first introduced, you had to program them in assembly language to move bytes around in a very laborious fashion. Compilers for C language were introduced about a decade later, but they tended to be expensive (yes, you had to buy them) and they were still quite troublesome and time-consuming.

Let's suppose you had a PIC microcontroller, and you wanted it to process an analog input from a temperature sensor. When the voltage from the sensor increased beyond a certain level, you wanted another pin to switch on an LED. And that was all you wanted it to do.

The code that you wrote might look like the sample in Figure A.

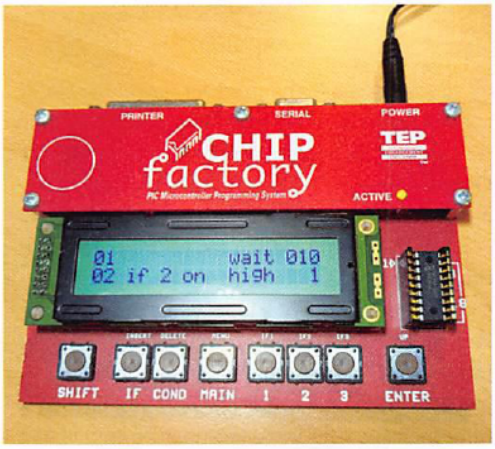
Could you avoid the labor of typing all this text by going online and downloading a routine that someone else had written? Probably not. Microchip (which acquired the rights to PIC microcontrollers) did provide some application notes and code examples, and if you lacked internet access in the mid-1990s, the company could send you the examples on a floppy diskette. Or, if you could get online, you might find something on Usenet, or on privately run bulletin boards — but it would not necessarily be for the particular PIC chip you were using, or for the

```
#include <16F877A.h>
#include <xc.h>
#device ADC=10
#use delay(crystal=8000000)

// CONFIG1
#pragma config FOSC = ECH // External Clock, 4-20 MHz
#pragma config WDTE = OFF // Watchdog Timer (WDT) disabled
#pragma config PWRTE = OFF // Power-up Timer disabled
#pragma config MCLRE = ON // MCLR/VPP pin function is MCLR
#pragma config CP = OFF // Flash Memory Code Protection off
#pragma config BOREN = ON // Brown-out Reset enabled
#pragma config CLKOUTEN = OFF // Clock Out disabled.
#pragma config IESO = ON // Internal/External Switchover on
#pragma config FCMEN = ON // Fail-Safe Clock Monitor enabled
// CONFIG2
#pragma config WRT = OFF // Flash Memory Self-Write Protect off
#pragma config PPS1WAY = ON // PPS one-way control enabled
#pragma config ZCDDIS = ON // Zero-cross detect disabled
#pragma config PLLEN = OFF // Phase Lock Loop disable
#pragma config STVREN = ON // Stack Over/Underflow Reset enabled
#pragma config BORV = LO // Brown-out Reset low trip point
#pragma config LPBOR = OFF // Low-Power Brown Out Reset disabled
#pragma config LVP = OFF // Low-Voltage Programming disabled
// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF

unsigned int16 i;
void main(){
  TRISC1 = 0;                       //  set RC1 pin as a output
  setup_adc(ADC_CLOCK_DIV_32);      // Set ADC conversion time to 32Tosc
  setup_adc_ports(AN0);             // Configure AN0 as analog
  set_adc_channel(0);               // Select channel 0 input
  delay_ms(100);                    // Wait 100ms

  while(TRUE){
      i = read_adc();
      if (i > 100) {
        RC1 = 1;                    // set RC1 pin to logic High & turn on
    } else {
        RC1 = 0;                    // set RC1 pin to logic Low & turn off
    }
    delay_ms(1000);                 // Wait 1000ms
   }
}
```

A A program listing to make a PIC do something quite simple, back in the day.



B A recent photo of Clive Seager.



C Clive Seager's first educational project to burn PIC chips: the Chip Factory.

appropriate compiler, so you'd have to adapt the code, and — well, it might be easier to continue your old-school habit of soldering components onto a piece of perf board.

CLIVE'S REVOLUTION

In 1996 a young Englishman named Clive Seager embarked on a mission that ultimately helped to resolve this sad situation. Clive was an electronics engineer, but had become a schoolteacher in the U.K., where a national curriculum specified that every child in every school had to learn something about electronics. He sent me the recent selfie in Figure **B**.

"There was a big mismatch between what the government was telling the schools they had to do, and what the teachers were capable of doing," he recalled, when I spoke to him earlier this year. "So I started developing resources for other teachers to use. And then I got invited to join the Technology Enhancement Programme, which was funded by one of the owners of a large supermarket chain, Sainsbury's. He was giving millions of pounds to develop technology and education."

This led Seager to start a company that he named Revolution Education (often known as Rev-Ed in the years that followed). And his first step was to make PIC microcontrollers easier to program. At the time, this was a fussy process. "You powered them with 5 volts," he explained, "but to put it into programming mode, you had to apply 12 volts to another leg of the chip."

To program it you needed a chip burner, which contained the complex programming circuit and applied the 12V signals. Clive came up with an educational burner named Chip Factory, which enabled students to enter about ten programming commands by pressing buttons without the need for a computer. The sequence was then programmed into a PIC chip. A vintage Chip Factory is shown in Figure **C**.

In the late 1990s Microchip introduced a new generation of PIC devices like the PIC16F872 microcontroller, which no longer needed the 12-volt signal. It had 2kB of flash memory (in 14-bit words), 128 bytes of RAM, and 64 bytes of EEPROM, creating an opportunity that Clive could not resist. With superhuman ingenuity, he

could just squeeze enough firmware into the tiny memory to include an interpreter and enable serial communication with a computer via a custom-built cable.

Now you could write a program on your computer in a high-level language, using an editor which tokenized the language and uploaded it to the PIC chip.

FACE IT YOU'RE BASIC

The high-level language that Clive Seager chose was BASIC, an acronym for *Beginner's All-purpose Symbolic Instruction Code*, which had existed since 1964. Clive modified it specifically for the PIC. The listing from Figure A has been rewritten in Figure D using Rev-Ed BASIC, and I think you'll agree, this makes things a bit simpler.

In the United States, a company named Parallax did something similar, marketing a microcontroller that they named the BASIC Stamp, because it wasn't much bigger than a postage stamp. An example of the Stamp 2 is shown in Figure E. I respect their product, but I happen to prefer Rev-Ed microcontrollers, because (a) the Stamp requires a miniboard instead of being entirely self-contained on one chip, and (b) a comparable Picaxe costs less than half as much, and I am thrifty by nature — which is to say, I am cheap.

However, Rev-Ed had a couple of problems.

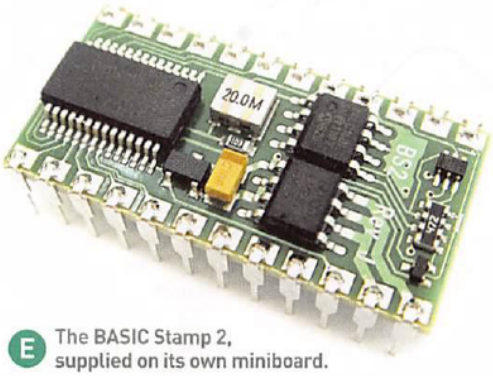
First, "serious" coders had a bad attitude toward BASIC. They complained that it didn't compel you to structure a program properly, but I suspect that the name of the language was just as much of a problem, especially as the *B* stood for *beginners*. It created a lasting stigma among "professionals," although the educational market quickly adopted it.

Not everyone scorned the language. Bill Gates, for instance, seemed to be a fan. Microsoft bundled BASIC with the first version of MS-DOS, and upgraded it to QuickBASIC and then Visual Basic over the years. Personally I still like BASIC because it doesn't impose arbitrary rules or quirky syntax. It also seems well suited to a microcontroller, where most code is not highly structured anyway.

But now, if you want a more socially acceptable alternative, probably you use Python. In Figure

```
do
   readadc A.0,b1      ; read ADC on pin A.0
   if b1 > 100 then    ; test if greater than zero
     high C.1          ; make pin output and switch on
   else
     low C.1           ; else make output and switch off
   end if
   pause 1000
loop
```

D The program from Figure A rewritten in Rev-Ed BASIC.



E The BASIC Stamp 2, supplied on its own miniboard.

```
from machine import Pin
from machine import ADC
from time import sleep

led = Pin(0, Pin.OUT)    # GPIO0 = Pin 1
adc = ADC(0)             # A0    = Pin 31

while True:
    x = adc.read_u16() # full scale is 65535
    if x > 6000:
        led.on()
    else:
        led.off()
    sleep(1)
```

F A MicroPython version of the code in Figure D. Note that the indents are mandatory.

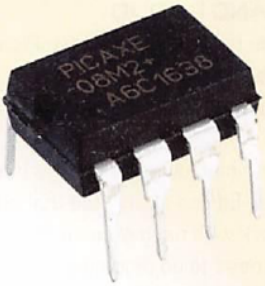
F you will see a Pythonized version of the BASIC code in Figure D. (This particular piece of Python was written for a Raspberry Pi Pico by my friend Fredrik Jansson, whose book *Make: Radio* was published last year.)

Personally, I find Python too willfully weird. Probably its most notorious quirk is that it compels you to indent a loop, while in every other language, indents are merely a stylistic option. But there is no command in Python to close a loop! (What madness is this?) And, you have to remember to include those annoying colons and parentheses. To me, Python is an invitation to create syntax errors.

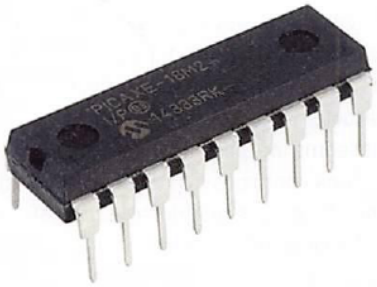
Parallax



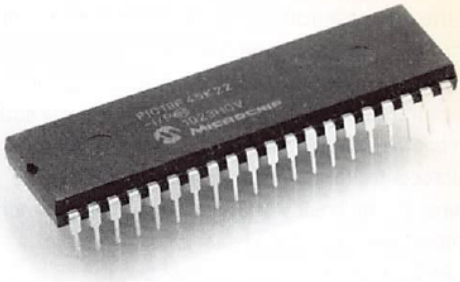
G USB cable needed to upload program code, and a breadboard adapter for the jack plug.



H Yes, this is a microcontroller. The 8-pin Picaxe.



I The Picaxe 18M2.



J The Picaxe 40X2.

BIG TREE SMALL AXE

But I digress. I mentioned two problems afflicting the Rev-Ed microcontroller. What was the second one? Well, Clive named his product the "Picaxe."

The name made sense, in a frivolous way, if you happened to know what a PIC chip was. But for code snobs, there was no way they could say "I do my programming in BASIC on a Picaxe." It was too embarrassing.

I am not easily embarrassed, so I bought a Picaxe soon after its introduction, and discovered its positive attributes. The chip was complete, without any extras. It didn't need its own miniboard. After you programmed it, you could plug it into a circuit on a breadboard, or solder it into some perf board, like any other chip. The documentation, developed by Rev-Ed, was excellent. And, as I mentioned previously, it was amazingly cheap!

You did need a special upload cable (shown in Figure G), but one cable could program any number of chips.

The smallest Picaxe is the same size as a 555 timer, with eight pins, although many of the pins have multiple functions, and an analog-digital converter is built in, along with fun features such touch-sensor recognition and musical ring tones. The current version of this chip, the 08M2, costs less than $3 if you buy it from a U.S. vendor such as RobotShop. A picture of it is shown in Figure H. To program it, all you need are the special cable, a 3.5mm jack socket, and a free download of the Picaxe IDE (integrated development environment software) which includes error checking and a simulator to test your code.

What if you want more I/O than eight pins can provide? The 18M2 has 18 pins (see Figure I), and the last time I checked, it cost less than $5.

Still more pins? The 40X2 has 40 of them, as in Figure J. Probably that should be enough, especially since more than 30 of the pins can be configured either for input or for output.

Picaxe also sells kits from its website (picaxe.com), and ships them to the United States. It offers a free downloadable graphical programming language based on Blockly (which has become a Google product) and add-on modules such as alphanumeric displays. It even sells an ultrasonic range finder, and a single-board oscilloscope.

CLOSED AND PROUD

I have left the best news until last: Picaxe products are not open source.

Wait a minute. *Am I deprecating the concept of open source?*

Yes. By retaining ownership of its intellectual property, Rev-Ed can guarantee that all its products work with each other, out-of-the-box, without any need to go digging around on GitHub for a piece of code, written by some wacky hacker you've never heard of, which may not even work by the time you download it, because the version of the microcontroller you're using may be too old or too new to be entirely compatible.

Of course, open source has an advantage: It encourages rapid development of third-party add-on boards with special features. Plus, modern microcontrollers are more powerful than Picaxe chips. But do I really need extra features and more processing power for a typical maker project?

As Clive Seager phrased it, when I raised the issue: "Unless you want to fly a drone or something like that, execution speed is irrelevant for many projects. The vast majority of people are putting pause commands into their programs, rather than wanting something to go faster. And there are no new 8-pin microcontrollers that have significantly more memory capacity or features than the one that we're using at the moment."

Personally, in every project that I've wanted to develop with a microcontroller, a Picaxe has been sufficient. This was even true for a rapid cooling system controlling six pumps in a laboratory where they thought they needed a real computer to run it. A simplified schematic is shown in Figure K, using a Picaxe 28X1 microcontroller. The program listing contained 480 lines.

IT JUST WORKS

So why I am ranting about this obscure British product when the rest of the maker community embraced Arduino happily so many years ago? Have I become a paid shill for Rev-Ed?

I can guarantee I have received no favors from Rev-Ed, not even a $3 chip. This rant is simply the outcome of two decades of mild irritation with the open source community. I keep waiting for it to evolve, but I don't think it ever will, because the concept itself is flawed.

The Raspberry Pi Pico, for instance, is a formidable achievement with all kinds of clever features. Yet I find the documentation inadequate, and even now, the hardware has omissions that I find inexplicable. For instance, I would think that connecting an alphanumeric display should be fundamental, yet apparently it is not possible until I download and install an LCD library "from a reliable source (e.g., GitHub repositories like Tom's Hardware tutorial or Random Nerd Tutorials)."

Seriously? Well, that was what Grok told me when I asked it for advice, and I think the AI is on-target this time. In which case, the Pico doesn't plug-and-play with an ordinary 2-line 20-character LCD screen.

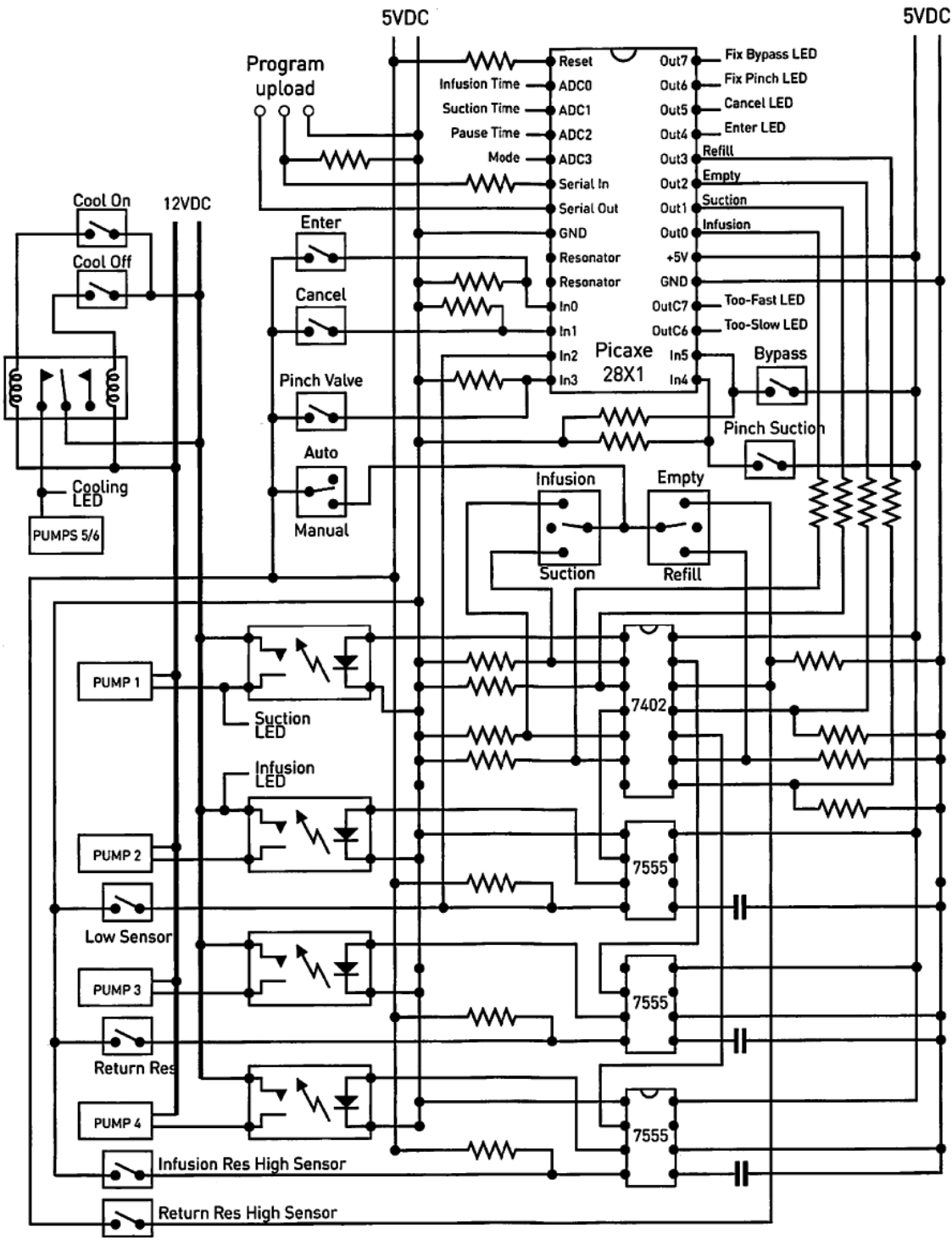
That seems inexplicable to me, but it gets worse. Grok continues: "If the LCD doesn't display text, the I²C address may differ (e.g., 0x3F instead of 0x27). Use an I²C scanner script to find the address." No. Absolutely no. I'm not going to waste my time like this. It's like an excruciating flashback to the 1990s, when incompatibilities were an endless source of exasperation.

The sad part is that Rev-Ed started marketing its simple and dependable products in 1999, six years before the Arduino. The BASIC Stamp was introduced even earlier. Why did the Arduino become so much more popular?

Maybe amusing names like Stamp and Picaxe marginalized the products, while Arduino had an exotic, European ring. Or maybe the early adopters of Arduino already knew how to write C, and loved the Arduino for using a "serious" language. And for them, open source was a fun concept, as they enjoyed swapping downloads with other GitHub fans who didn't mind the chores that I find so annoying, such as using an I²C scanner if I have trouble displaying two lines of text on a screen.

Here's my simple requirement. I just want a product that works, without any messing around, using a computer language that isn't afflicted with funny punctuation, and is properly documented by a company that owns the IP, so I can have confidence that it will continue to work tomorrow in the same way that it works today.

And that is why I cannot be diplomatically neutral about microcontrollers.

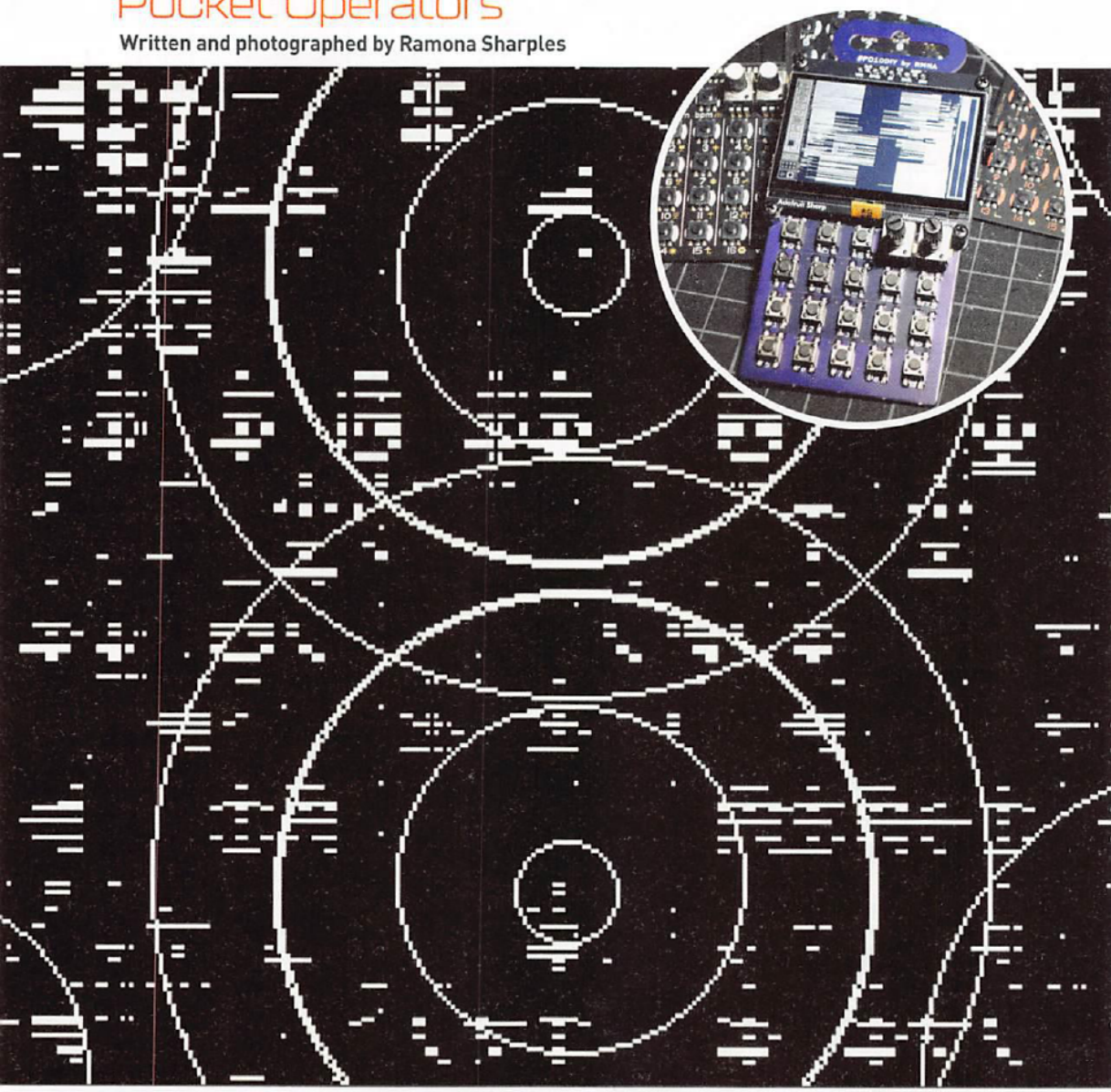


K Yes, a Picaxe microcontroller can handle serious tasks.

POCKET VIDEO SYNTHESIZER

Generate video art on the go with a tiny gadget inspired by Teenage Engineering's Pocket Operators

Written and photographed by Ramona Sharples



In March, Teenage Engineering, the company behind beloved and thoughtfully designed products like the Pocket Operator synthesizers and OP-1, ran a design contest. To celebrate the 10th anniversary of the first Pocket Operators, they invited makers to build hacks, mods, and re-imaginings of their whimsical pocket-sized synthesizers and post them under #PO10DIY.

I love how Pocket Operators pack huge sound into a tiny portable package, so I tried to bring that ethos to a new medium and built a miniature video synthesizer in the P.O. format. This gadget generates programmatic visual patterns by drawing layers of simple shapes like circles, triangles, and lines. You can tweak the parameters of these visual programs and save your patterns into a sequence. It has an onboard two-color Sharp Memory Display, but it also has an HDMI output so you can show your art on a projector or screen. Now you can carry video art in your pocket!

1. DESIGN

I started with the layout of the device. I wanted to fit an onboard screen, and I had a spare Sharp Memory Display that I thought would be perfect. Each pixel can only display two colors (black or white), but it's pretty high-resolution at 400×240px in a 2.7" display, and it uses very little power with its backlight-free reflective technology.

Teenage Engineering published detailed specs and models for the Pocket Operator boards, so I started mocking up layouts in Figma by cobbling screenshots together to see how I might accommodate a screen (Figures A and B on the following page). I considered making it taller, or removing some controls — ultimately I decided to eliminate just one row of buttons and the branding area to keep the height to 105mm. The screen makes it 64mm wide, compared to the original 60mm, but it's almost perfect!

2. KEYPAD FUNCTIONS

So what should all these controls actually do? Pocket Operators generally include a 4×4 button grid for selecting one of 16 sounds and configuring 16-step sequences, but I decided to pare that down to a 4×2 grid — instead, there would be eight "video layers" to play with, and

Ramona Sharples, Adobe Stock-BAIVECTOR

TIME REQUIRED: A Weekend

DIFFICULTY: Advanced

COST: ~$100

MATERIALS

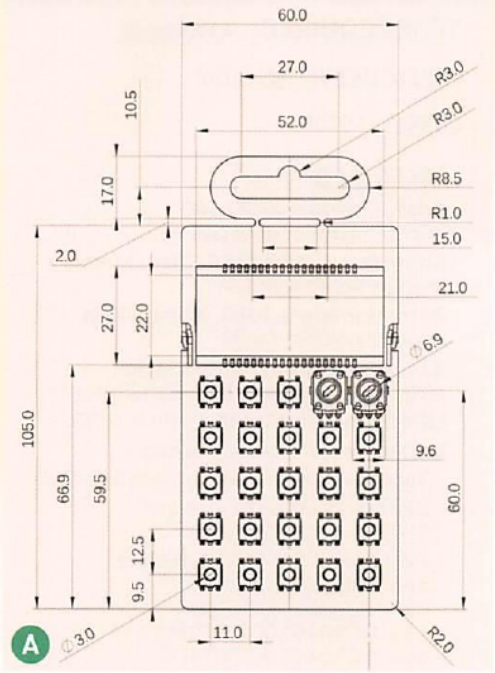
- **Adafruit Feather RP2040 microcontroller with DVI** Adafruit 5710
- **Adafruit 2.7" 400×240 Sharp Memory Display** Adafruit 4694
- **Potentiometers, 10kΩ, with built-in knob (2)** Adafruit 4133
- **3.5mm stereo headphone jacks, breadboard friendly (2)** Adafruit 1699
- **LiPo battery, 400mAh** Adafruit 3898
- **Surface-mount components:**
  - **Tactile switches (18)** Jameco 2334399
  - **Diodes, 1N4148 (18)** Mouser 78-1N4148W-HG3_A-08
  - **Resistors, 1.1kΩ (2), 10.2kΩ (2), and 15kΩ (1)** Mouser 594-MCS04020D1101BE0, 594-MCS04020D1022BE0, and 594-MCS04020C1502FE5
  - **Capacitors, 15pF (2)** Mouser 81-GRM155R60G156ME1D
- **Slide switch** Mouser 612-EGJ1210AAT2
- **Header pins and sockets, standard 0.1"** for the microcontroller and display
- **Custom PCB** I ordered mine from OSH Park! Get the PCB files from my GitHub repo at github.com/ramonaisonline/po10diy.

TOOLS

- **Computer with internet** Grab the project code from the same GitHub repo.
- **Soldering iron**
- **Lead-free solder**
- **Low-temp solder paste** I used Maker Paste, Adafruit 3217.
- **Hot plate** I used a TLBZK 100×100mm 350W hot plate from Amazon.
- **Ventilation fan**
- **Snips**
- **Tweezers** for positioning small components
- **Hobby knife** for cleaning up rough PCB edges

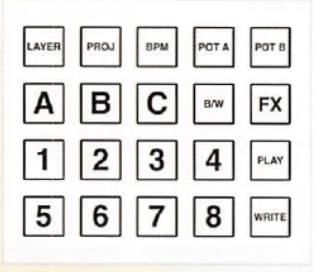


RAMONA SHARPLES aka RMNA is a Bay Area multimedia artist, hacker, and tall girl. ramona.diy





an eight-step sequence to configure. I kept the familiar controls on the right side: record, play, an FX mode button, and two potentiometers. I also kept the buttons at the top for changing the layer you're editing, the sequence you're working on, and the tempo. The remaining four buttons include !, !!, and !!!, which allow you to choose variations of whatever shape layer you've selected, and a black/white color toggle.



3. SCHEMATIC

With my design in mind, my next step was to make a schematic. I've built several video synthesizers and have used this display and Adafruit RP2040 microcontroller (Figure C) before, so I had a pretty good idea of how I wanted this to work.

The keyboard is configured as a matrix of switches that connect rows to columns, like a mechanical keyboard (Figure D). You can scan this grid by pulling a column low and reading each row to see if any switches on that column are pressed, with diodes preventing "ghost presses."

The two potentiometers are set up as voltage dividers, with connections to power, ground, and the signal passing through an RC (resistor-capacitor) low-pass filter set to ~10Hz to clean up some of the noise (Figure E).

Pocket Operators can be daisy-chained with TRS cables, and they can pass audio along the right channel, with a clock signal on the left channel to keep each other in sync. I wanted to be able to tempo-sync the video sequence, so the audio jack feeds the sync signal into a digital input on the Feather, and send the audio to both channels of the output jack.

The display connects via a standard 0.1" header strip and communicates via the Feather's SPI bus (Figure F). I also added a power switch that connects the Feather's Enable pin to ground to switch it off.

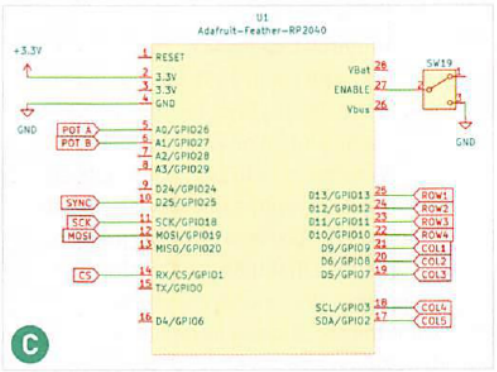
4. CIRCUIT BOARD LAYOUT

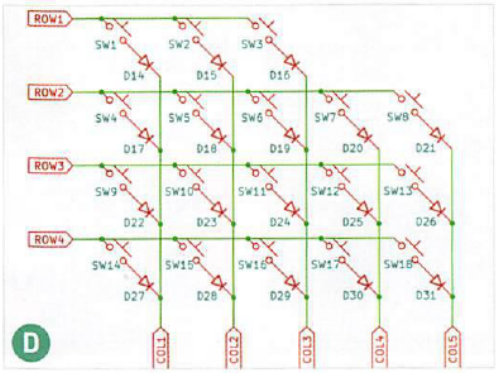
Next I started working on the actual circuit board layout. The combination of small components and specific spacing requirements made it tricky, but I just worked through it step by step.

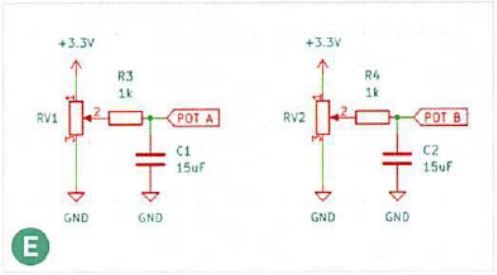
I began by creating the board profile and laying out all the components that needed to be in specific places: the screen, buttons, and knobs. Then I placed the microcontroller at the back and I put the diodes, and most of the resistors and capacitors, underneath the screen to protect them from users' fingers.

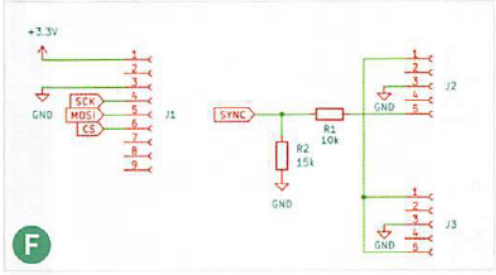
Finally I routed the traces, using the top layer for horizontal connections (Figure G), and the bottom layer for vertical connections (Figure H), with a few exceptions.

After double- and triple-checking everything, I sent the board design files to OSH Park for manufacturing.









5. SURFACE-MOUNT SOLDERING

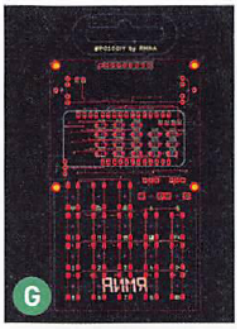
After the boards arrived (Figure I), I started assembly. I had done some hand-soldering of small surface-mount components, and it can be quite tricky! One approach is to add a small solder blob to one pad, then grab your component with tweezers, slide it in, and let the blob cool to glue it down. Then you're free to heat up the other pad and solder that down too without the component moving around too much.

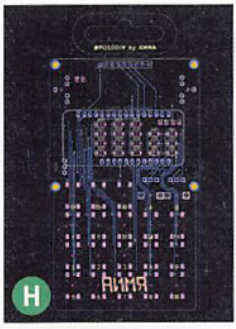
But since my grid of diodes was so closely packed, I figured I'd upgrade my setup a bit and try a hotplate and solder paste.

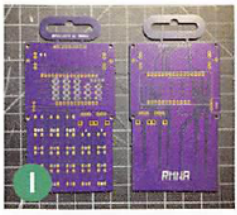
First I carefully organized all my tiny parts so I didn't get them mixed up (Figure J).

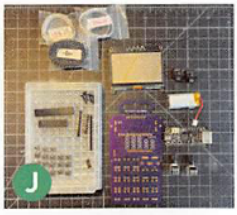
Then I added blobs of solder paste (a gooey mixture of flux and microscopic balls of solid solder) to each pad, and plopped the components on top with tweezers (Figure K). Note that this really only works when all your surface-mount components are on the same side of the board!

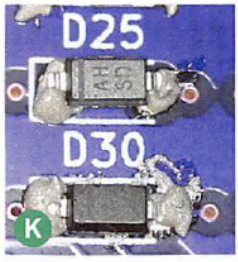
After melting the solder paste for a minute or two, I inspected all the connections with a USB microscope to make sure there was enough solder on every component, and reheated the board with a little more paste here and there to fix any mistakes (Figure L).

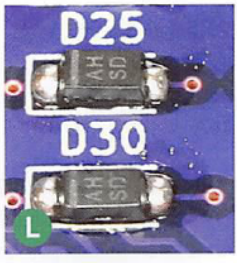


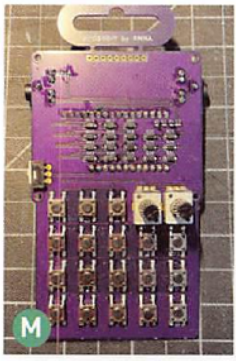


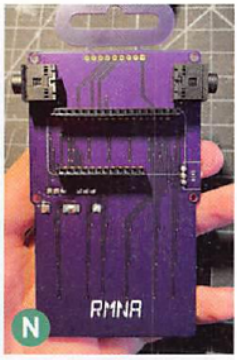




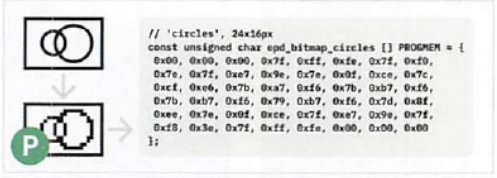












6. THROUGH-HOLE SOLDERING

After the surface-mount components were finished, I hand-soldered the potentiometers, the power switch, and the header sockets for the microcontroller and trimmed the leads with snippers (Figures M and N). I used a socket for the microcontroller to make it removable in case I needed to repair or repurpose it someday, though it also has the benefit of creating a small space to tuck a 400mAh LiPo battery.

Finally I soldered the connection for the display, which was just a strip of 0.1" header pins for a lower profile. The inner corner of the bottom-right mounting point on the display had a small curve that needed to be filed down a smidge to fit around the hard corner of Pot B.

7. INPUT & OUTPUT CODE

With the hardware done, I could finally move on to the really creative and fun part: the code! Adafruit's guide for the Feather RP2040 with DVI has a great tutorial that inspired my first video synth projects, and I built on those foundations.

I started with the displays: the onboard Sharp display has a resolution of 400×240, and the external DVI output is 320×240, which gets scaled up 2x in the output routine to 640×480. I decided to use the extra 80 pixels on the Sharp display for a tiny interface that tells you the state of the sequencer, what drawing layers you're using, what color they're drawing in, and more (Figure O). The visual elements of the UI were designed as PNGs in Figma, then converted to bitmap **char** arrays using a web tool called image2cpp and stored in program memory (Figure P).

The Sharp display is controlled by the *Adafruit_SharpMem* library, and the DVI output by Adafruit's fork of Luke Wren's excellent *PicoDVI* library (learn.adafruit.com/picodvi-arduino-library-video-out-for-rp2040-boards). Both display drivers support drawing with Adafruit's *GFX* graphics library. *GFX* provides some high-level drawing functions that will feel familiar to users of Processing: **`drawCircle(x, y, radius, color)`**, **`fillRect(x, y, width, height, color)`**, etc. For all of those basic operations, I created helper functions that call them once for the DVI display and again for the onboard display (with a 40px horizontal offset to account for the extra UI elements). Drawing everything twice isn't the most efficient approach — I'd love to figure out how to make them share memory someday — but it's simple and it works well enough!

Next I tested the inputs. I scanned the keyboard with the *Adafruit_Keypad* library and tracked things like toggling playback, recording, and switching drawing tools. Then I incorporated values from the potentiometers, cleaning up any remaining noise by reading them a few times and averaging the result. At this point, I could show a simple graphic on screen, like a circle orbiting the center of the screen, and control its speed and size with the potentiometers.

8. VIDEO SYNTH CODE

The final task was to get the sequencer working and to make the visuals interesting. There's no shortage of fun approaches to video art, but I've always liked the effect you get from simple

geometric patterns building on top of each other in an elegant mess. Keeping things visually appealing with only two colors was tricky (it's easy to turn the whole screen white for instance), but with some experimentation I found that making small additions to the canvas, drawing with both colors at once, and occasionally using big shapes to wipe out swaths of the canvas all helped keep the video compelling.

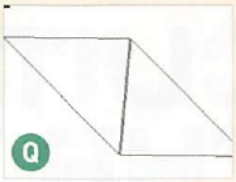
The bigger challenge was having only two knobs. I wanted to help users explore a wide range of visual effects, so I use a single knob to sweep through a "macro" of several parameters at once. For example, I might want a single knob to control the position and size of a circle on screen: the x, y, and radius. I wrote a function that maps the knob's raw value (0 to 1023) into an output range (say, 0 to 319 for x position), with the option to sweep through the output range multiple times. A single knob might sweep through the range of x values five times and the range of radius values twice, providing access to lots of combinations of values.

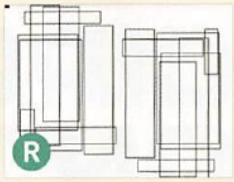
After some tinkering and experimentation, the shape functions I settled on were: opposing triangles, symmetrical rectangles, radiating lines, mirrored circles (Figures Q, R, S, and T), random characters, twirling circles, grids, and sparkly dots.

Each of these layers can be active on each of the eight steps of the sequencer, with independent potentiometer values recorded at each step for each layer, along with the !/!!/!!! variations and the black/white toggle. In the sequencer code, this is modeled as an array of 256 integers (4 values × 8 layers × 8 steps), and a few handy helper functions for retrieving and recording values easily.

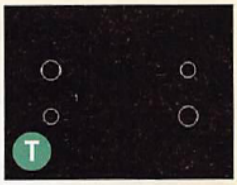
GO WITH THE THROW

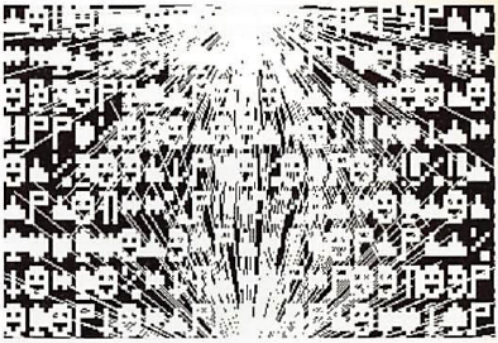
The two-color moving graphics look amazing on a projector, and it's fun to see the contrast between tiny graphics in your hand and gigantic art thrown onto the wall. I'll likely continue to work on the shape functions and swap things out as I have new ideas. The best part about building your own tools for making art is that they can evolve and grow with you. I hope this project inspires you to build your own video art gadgets!



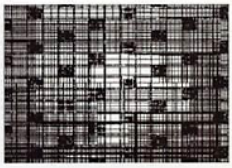


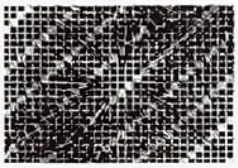














RESOURCES:

- See the Pocket Video Synth in action at: makezine.com/go/po-vid-synth
- Source code, PCB files, and detailed instructions: github.com/ramonaisonline/po10diy
- To get the Arduino IDE and basic code set up correctly, I used tutorials from this guide to Adafruit's DVI video generation library *PicoDVI*: learn.adafruit.com/picodvi-arduino-library-video-out-for-rp2040-boards
- More about programmatic graphics with Adafruit *GFX* library: learn.adafruit.com/adafruit-gfx-graphics-library/overview

CIRCUITPYTHON ON THE OXOCARD CONNECT

Written and photographed by Jacques Supcik

Our nifty microcontroller comes with beginner-friendly NanoPy language – but it also runs super-popular CircuitPython



The Oxocard Connect is a versatile microcontroller board with a color display and plug-in cartridges for making all kinds of electronic projects. It has been designed to be programmed in NanoPy, a beginner-friendly language similar to Python, but you can also program it using CircuitPython, a popular Python implementation for microcontrollers. In this article we'll cover how to install CircuitPython, the basics of programming with it, and some simple experiments to get you started.

TIME REQUIRED: 1–2 Hours

DIFFICULTY: Easy

COST: $80–$90

MATERIALS

- **Oxocard Connect microcontroller** Get the Oxocard Connect and the following components needed for this article in the Make: Oxocard Innovator Kit, makershed.com/products/make-oxocard-innovator-kit. Now includes our *Getting Started with Oxocard Connect* book!
- **Breadboard Cartridge**
- **LED**
- **Resistor, 220Ω**
- **Jumper wires, small**

TOOLS

- **Computer with internet browser and Thonny IDE software** free download at thonny.org

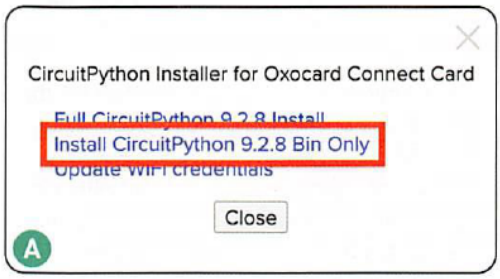


JACQUES SUPCIK is a professor of computer science at the School of Engineering and Architecture of Fribourg, Switzerland (HEIA-FR).

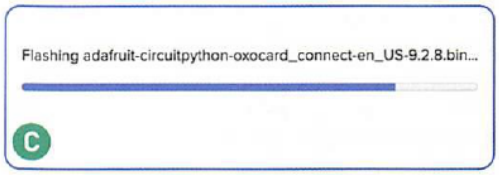
INSTALLING CIRCUITPYTHON

The simplest way to install CircuitPython on the Oxocard Connect is to use the web installer from CircuitPython. It uses WebUSB (developer.mozilla.org/en-US/docs/Web/API/WebUSB_API) and requires a compatible browser; it works best with Google Chrome or Microsoft Edge. (Firefox and Safari do not support WebUSB.)

1. Connect the Oxocard Connect to your computer using a USB cable.
2. Open your browser and go to circuitpython.org/board/oxocard_connect.
3. Choose the language version you want to install (e.g., English).
4. Click on Open Installer to launch the CircuitPython web installer.
5. From here, you can either make a full install, or install the binary only, or update the Wi-Fi credentials. The Full option is tempting because it includes the Wi-Fi configuration, but, at the time of writing, it was not working properly. For now, it is safer to choose the Bin Only option (Figure A).
6. When you see the welcome screen of the installer, click on Next.
7. On the next screen, click on the Connect button. Your browser shows you a list of USB devices. Select the Oxocard Connect from the list and click on Connect.
8. After a few seconds, the installer warns you that the Oxocard will be erased. Click on Continue to proceed (Figure B).
9. The installer erases the flash memory and installs CircuitPython (Figure C).
10. If all goes well, your Oxocard will display some text and your browser will tell you the installation was successful. Click on Close to finish.







Adobe Stock - BAIVECTOR, Thomas Garaio

GETTING STARTED WITH CIRCUITPYTHON

CircuitPython is a beginner-friendly version of Python designed for microcontroller boards, including the Oxocard Connect. It provides a simple way to write Python code that interacts with the hardware.

Some boards (for example, those based on the ESP32-S3 or on Raspberry Pi Pico) act as a USB mass storage device when connected to a computer, so you can simply drag and drop files to the board, like a USB flash drive.

The Oxocard Connect uses a tiny ESP32-PICO chip that does not support this feature, so you'll need to use a serial connection to upload your code. The easiest way to do this is to use Thonny, a popular Python IDE that supports CircuitPython.

BLINKING AN LED

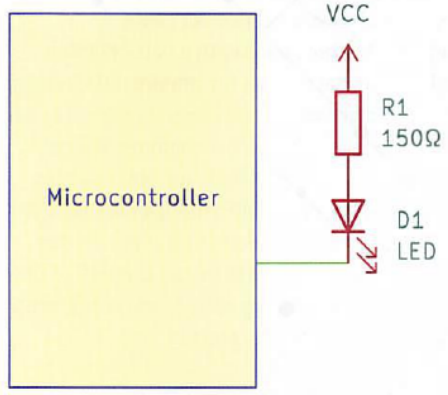
When you learn a new programming language, the first program you usually write is called "Hello World." In the world of microcontrollers, this program is often called "Blinky." It simply turns an LED on and off at regular intervals.

Before coding this program, let's assemble the hardware for it. Note that there are two ways of connecting an LED to a microcontroller: the microcontroller can either ***source*** or ***sink*** the current flowing through the LED. In the first case, the microcontroller pin is connected to the anode of the LED and the cathode is connected to ground. In the second case (Figure D), the cathode is connected to the microcontroller pin and the anode is connected to a positive voltage, usually 3.3V with modern microcontrollers.

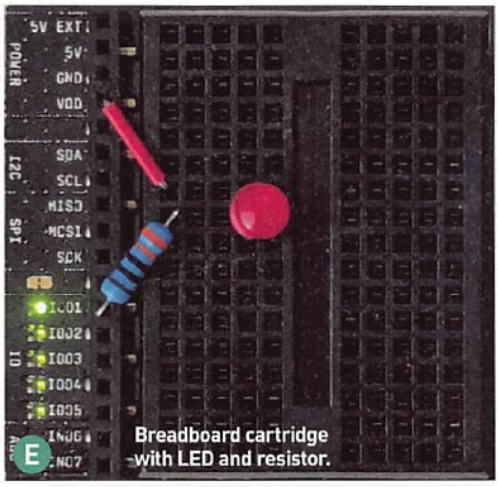
The sinking option is often preferred because it usually allows you to drive the LED with a higher current. The only drawback is that the microcontroller must set the pin LOW to turn the LED on, and set it HIGH to turn it off. This can be a bit counterintuitive, but you get used to it!

As shown in Figure E, connect the LED's longer leg (anode) to VDD using a red wire and its shorter leg (cathode) to pin IO01 through a 220Ω resistor.

Download and install the Thonny IDE from thonny.org. Then connect your Oxocard Connect to your computer with a USB cable, and open Thonny. It should automatically detect the



D Microcontroller sinking current from an LED.



Oxocard Connect and connect to it. If not, you can select the Oxocard Connect from the bottom right corner of the Thonny IDE window.

You can now open the *code.py* file on the Oxocard Connect. This is the main script that will be executed when the Oxocard Connect starts. You can edit this file directly in Thonny (Figure F).

Replace the content of the *code.py* file with the following code:

```
1 # https://github.com/supcik/Oxocard-
Connect-CP-Make-Src/blob/main/src/
blinky.py
2 # Blinky with Oxocard Connect and
CircuitPython
3
4 import time
5
```

```
import board
import digitalio

HALF_PERIOD_S = 0.2 # Half period in seconds
LED_PIN = board.IO01


def main():
    led = digitalio.DigitalInOut(LED_PIN)
    led.switch_to_output(True)
    while True:
        led.value = not led.value # Toggle the LED
        time.sleep(HALF_PERIOD_S)


main() # Run the main function
```

The code above turns the LED on and off every 200 milliseconds. It uses the **digitalio** module to control the GPIO pins of the Oxocard Connect.

Figure G shows the Blinky program in Thonny. To run it, click on the green arrow. You should see the LED blinking on the Oxocard Connect.

NOTE: The external LED you connected on the breadboard is sinking current, but the small green LED on the PCB is sourcing current. This means that both LEDs blink at the same time, but they are in opposite states in the program: when the external LED is "on," the small green LED is "off," and vice versa.

USING A BUTTON TO CONTROL IT

Our next experiment is to to toggle the LED on and off when a button is pressed. For this, we'll use the joystick button on the Oxocard Connect. The button sends a digital signal to the GPIO pin. On the Oxocard, the signal is 0 (or **False**) when the button is not pressed, and 1 (or **True**) when the button is pressed.

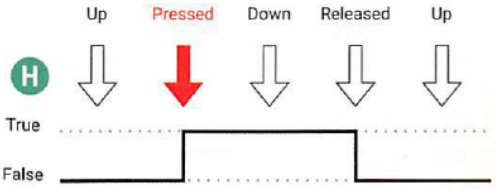
The button has only two positions, but as shown in Figure H, we define four states:

- **Up:** the button is not pressed.
- **Pressed:** the button has just been pressed.
- **Down:** the button is pressed.
- **Released:** the button has just been released.



TIP: You don't need to type the code manually; you can copy and paste it from the source code repository instead. You'll find it on GitHub at github.com/supcik/Oxocard-Connect-CP-Make-Src/tree/main/src.





We are interested in the *Pressed* state, when the button was *Up* but is just now going *Down*. This is the moment when we want to toggle the LED.

We could easily code this behavior ourselves, but CircuitPython provides a convenient way to handle buttons using the Adafruit Debouncer Library (docs.circuitpython.org/projects/debouncer/en/latest). To install it, you'll download the CircuitPython library bundle and use Thonny to copy the library files to the Oxocard Connect:

1. Go to circuitpython.org/libraries and download the latest CircuitPython library bundle for your

version of CircuitPython.

2. Extract (unzip) the downloaded ZIP file.

3. Open the Thonny IDE and connect to your Oxocard Connect.

4. In Thonny, make sure the Files view (the folder icon on the left sidebar) is open.

5. At the top of the Files view ("This computer"), navigate to the folder where you extracted the library bundle, open the */lib* folder, and search for the *adafruit_debouncer.mpy* file (Figure 1).

6. At the bottom of the Files view ("CircuitPython device"), open the */lib* folder.

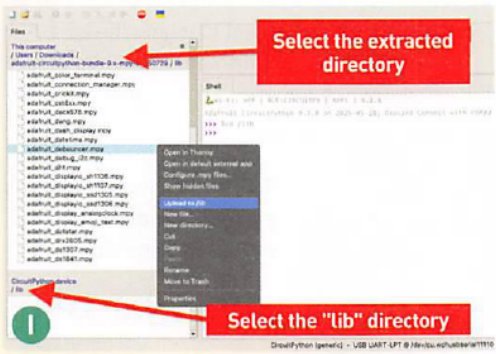
7. Left-click on the *adafruit_debouncer.mpy* file and choose Upload to /lib.

8. The file should appear in the */lib* folder on the CircuitPython device.

9. Repeat this for the *adafruit_ticks.mpy* library, which is a dependency of *adafruit_debouncer*.

Now replace the content of the *code.py* file with the following code:

```
# https://github.com/supcik/Oxocard-Connect-CP-Make-Src/blob/main/src/button.py
# LED control with button press using CircuitPython

import board
import digitalio
from adafruit_debouncer import Button

LED_PIN = board.IO01
BUTTON_PIN = board.BTN5 # The middle button


def main():
    # Configure the LED
    led = digitalio.DigitalInOut(LED_PIN)
    led.switch_to_output(True)

    # Configure the button
    btn = digitalio.DigitalInOut(BUTTON_PIN)
    btn.direction = digitalio.Direction.INPUT
    btn.pull = None # The Oxocard already provides a pulldown
    switch = Button(btn, value_when_pressed=True)

    while True:
        switch.update()
        if switch.pressed:
            led.value = not led.value # Toggle the LED


main()
```



Now a press of the joystick button toggles the LED on and off!

DIMMING THE LED

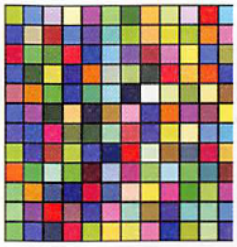
An LED works with a given voltage, and it is not possible to change its brightness by changing the voltage. Instead, we can use a technique called ***pulse-width modulation (PWM)*** to control the brightness of the LED. PWM allows us to control the *average* power delivered to a device by switching the device on and off at a high frequency. By changing the ratio of the time it's on to the time it's off, we can control the average power delivered to the device and thus control its brightness. This ratio is called the ***duty cycle***.

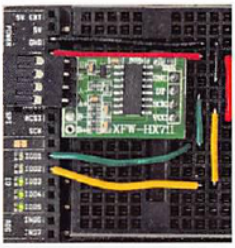
PWM is a very common technique and CircuitPython provides a simple way to use it through the `pwmio` module. This module is already included in the CircuitPython installation.

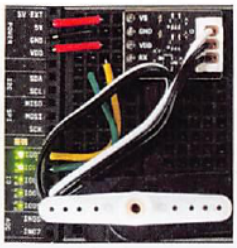
```
# https://github.com/supcik/Oxocard-Connect-CP-Make-Src/blob/main/src/dimmer.py
# LED control with PWM
```

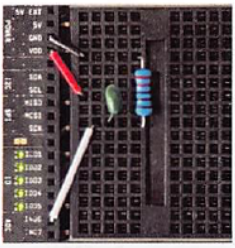
```
3
4 import board
5 import digitalio
6 import pwmio
7 from adafruit_debouncer import Button
8
9 LED_PIN = board.IO01
10 BUTTON_PIN = board.BTN5
11
12 # Duty cycles sequence for the LED
13 DUTY_CYCLES = [0xFFFF, 0xF000,
0x0000, 0xF000]
14
15
16 def main():
17     index: int = 0
18     # Configure the LED with PWM
19     led = led = pwmio.PWMOut(
20         LED_PIN, frequency=50000,
duty_cycle=DUTY_CYCLES[index]
21     )
22
23     # Configure the button
24     btn = digitalio.
DigitalInOut(BUTTON_PIN)
25     btn.direction = digitalio.
Direction.INPUT
26     btn.pull = None # The Oxocard
already provides a pulldown
27     switch = Button(btn, value_when_
pressed=True)
28
29     while True:
30         switch.update()
31         if switch.pressed:
32             # configure duty cycle of
the PWM with the next value
33             index = (index + 1) %
len(DUTY_CYCLES)
34             led.duty_cycle = DUTY_
CYCLES[index]
35
36
37 main()
```

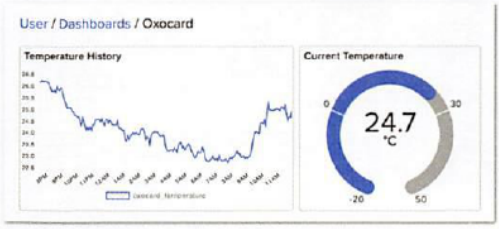
Run this program on the Oxocard Connect and press the button to cycle the LED through four different brightness levels: off, medium brightness, full brightness, and medium again.











SENSORS, SERVOS, SCREENS, AND THE INTERNET

Congratulations, you have installed CircuitPython on the Oxocard Connect and learned how use it to switch an LED, read a button, and dim the LED. In the extended version of this article, which you can find online at makezine.com/go/oxocard-cp, we'll also show how to use your CP-powered Oxocard Connect to play with the built-in OLED display, read values from sensors, control a servomotor, and connect to the internet!

CircuitPython is a powerful and easy-to-use language that allows you to quickly prototype and develop projects with your Oxocard Connect. Feel free to experiment with the project code and add your own features. Learn more about using CP and its libraries at docs.circuitpython.org/en/latest. And please don't hesitate to share your projects and contribute to the Oxocard Connect ecosystem at discord.gg/7zMew7KgaZ, facebook.com/oxocard2, and instagram.com/oxocard.

And when you want to revert to NanoPy, Oxon's web installer at oxocard.github.io/oxocard-firmware makes it easy.

MESHTASTIC WALKIE-TEXTIE

Stay in touch without cellular, with this LoRa mesh communications project

Written and photographed by Jon "ShakataGaNai" Davis





JON DAVIS is a lifelong tinker who loves exploring new technologies. He is a contributor to the Meshtastic open-source project along with being a leader of the San Francisco Bay Area Mesh users group. bayme.sh

Meshtastic is a tool for communicating without the need of cellular service. In simplest terms, think of it like a walkie-talkie for texting. It's a distributed, off-grid, mesh communication system, and just as with the venerable walkie-talkies of old, you don't need a license or special permit to use it. Meshtastic utilizes a low-power, long-range radio technology (LoRa) that can provide much greater ranges than walkies, and it utilizes encryption by default, so you can keep your conversations private and secure — no strange people dropping into your conversation unannounced.

WHEN TO USE IT

Any time you anticipate not having cell service is a good time to make use of Meshtastic, such as while backcountry hiking, backpacking, or camping. But it's also useful back in civilization. In most locations, cellular is ubiquitous to the point where it is taken for granted — until it doesn't work, at which point it can be a major headache. Maybe you're attending a music festival with 50,000 of your closest friends, and despite the best efforts of major telecom companies, smoke signals would be more effective than texting. Or worse yet, a disaster has struck and cellular infrastructure no longer works at all.

Another major difference from old walkies is that Meshtastic, being digital, can send more information in the background. Beyond just text-based messages, Meshtastic can also convey GPS location information, for near-real-time location tracking and telemetry data. This means you could use off-grid weather stations or remote sensors to collect river data, and receive the data without needing to pay for multiple SIM cards.

RANGE AND REPEAT

The range of Meshtastic can be impressively long, with a current record of more than 200 miles, however keep in mind that it uses radio and cannot violate the laws of physics. In a more average situation, a pair of handheld Meshtastic ***nodes*** will get 3–5 miles of range. The distance is controlled entirely by the environment between the two nodes. In downtown San Francisco with its plethora of tall concrete and metal buildings? Less than a mile. On a hilltop with a clear view to the horizon? 50 miles.

But your message can travel much farther. While Meshtastic isn't magic, it does have one trick up its sleeve, and that's the ***mesh network***. A Meshtastic node that can hear a message will then rebroadcast it for other nodes to hear, similar to a repeater. The more nodes you have on your mesh, the more resilient your messaging becomes and the farther your message can go. The mesh allows your Meshtastic communications to do "impossible" feats like cross over a mountain (with a node on top to repeat messages) and travel greater distances than any single node-to-node operation could go.

TIME REQUIRED: 1–2 Hours

DIFFICULTY: Easy to Moderate

COST: $50–$60

MATERIALS

FOR A BARE MINIMUM NODE:

- **RAKwireless WisBlock Mini Meshtastic Starter Kit** with RAK19003 base, about $30, Amazon B0DFMMTQZM or Rokland 115093, store.rokland.com. Rokland. You'll want the 915MHz version for North & South America, 868MHz for Europe. You can check your LoRa region by country at meshtastic.org/docs/configuration.

ADDITIONALLY, FOR A PORTABLE HANDHELD NODE:

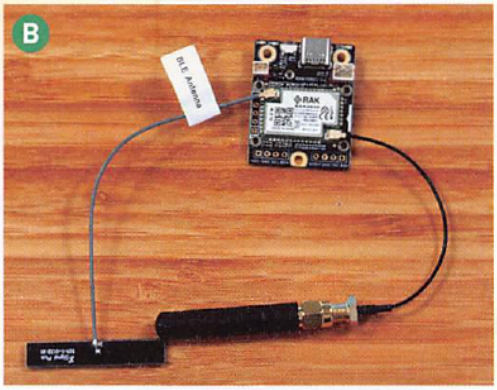
- **LiPo battery, 3.7V 500mAh, with 2.0mm/1.25mm JST connector** such as Amazon B091FKGW8H
- **Machine screws, M3×20mm socket head cap (4)**
- **Nuts, M3 (4)**
- **3D printed case** Download the free 3D files for printing at printables.com/model/286664.
- **Environment sensor (optional)** if you want to experiment with environmental telemetry. Try a BME280 temperature/humidity/pressure sensor, Amazon B07KR24P6P, or the RAK1906 BME680 environment sensor module (store.rokland.com) for a solderless option.

TOOLS

- **3D printer**
- **Computer with Chrome-based browser**
- **Soldering iron and solder (optional)**

Adobe Stock-BAIVECTOR





MESHTASTIC IS OPEN SOURCE

Mesthastic is an open source software project that runs on different types of nodes, some off the shelf and some purpose built. There is node hardware that has a built-in screen and keyboard, like the **LilyGo T-Deck** (Figure A, at right; see "New & Notable" in the Guide to Boards insert), but most users choose to interact with Meshtastic using their phone; the node pairs to your iOS or Android phone via Bluetooth and the companion **Meshtastic phone app**. Anyone who's sent an SMS text or used something like Find My will find themselves quite at home already. As such, the node hardware form factors run the gamut from those retro-Blackberry all-in-ones, to basic nodes like the **LilyGo T-Echo** (left), to waterproof "cards" such as the **Seeed SenseCAP Card Tracker T1000-E** (center).

But it's more fun (and sometimes cheaper) to make your own. There are a number of dev kit options available for those who prefer the more "DIY" route or need something more flexible.

MAKE A MESHTASTIC WALKIE-TEXTIE

If you build it, it will mesh! In this project, we start with the most common of the Meshtastic-compatible dev kits, and then build it up into a usable handheld node including extra sensors and a 3D-printed case. We'll also show how you could use the same hardware in other form factors, such as a solar-powered infrastructure node. Of course, while this demonstration shows just a single Meshtastic node, you should buy two. It's much more fun to give one to a friend/roommate/loved one/random person on the street so you have someone to chat with!

1. BUILD YOUR "BARE MINIMUM" NODE

For this you need only the RAKwireless WisBlock Mini Meshtastic Starter Kit.

1a. Plug in the BLE Antenna into the IPEX port labeled "BLE". Plug the IPEX-to-SMA adapter into the IPEX port labeled "LoRa" and attach the stub antenna to the SMA connector (Figure B).

> **WARNING:** Always make sure you have an antenna plugged in *before* you power up the node, to prevent damage to the radio.

1b. Plug the WisBlock into your computer with the provided USB-C cable.

1c. Use a Chrome-based browser to head over to flash.meshtastic.org (Figure C).

Under the Device heading, hit the rocket ship button to automatically detect your WisBlock (Figure D). A web-to-serial popup will ask you to connect; choose the "WisCore RAK4631 Board" (Figure E). Now the Select Target Device button will be relabeled "RAK WisBlock 4631."

Under Firmware, leave it on the default — the latest Beta version (Figure F).

1d. Click on Flash, then Continue, then Enter DFU Mode and select your serial device again.

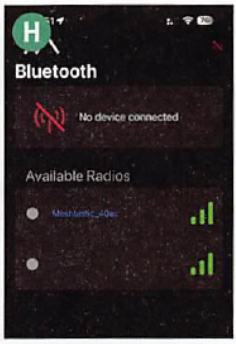






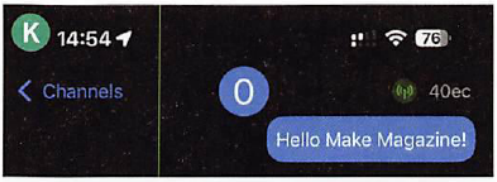












1e. Finally, hit Download UF2 which will download the firmware to your computer.

1f. Open Finder or File Explorer and you should see a new drive named *RAK4631* (Figure G). Copy the UF2 file you downloaded to that new drive. When the copy completes, the WisBlock will disconnect and start updating; it takes a minute.

1g. While you wait, download the Meshtastic app on your phone, from meshtastic.org/downloads.

1h. Open the app on your phone. On the Bluetooth screen, look for "Meshtastic XXXX" (Figure H) — the last four characters will be unique to your node. The pairing PIN is 123456 by default.

1i. The app will ask you to set your LoRa region (Figure I). Choose your country (Figure J) and hit Save. Your node will disconnect and reboot.

1j. Now you can send and receive messages via Meshtastic (Figure K), totally off-grid!

NOTE: Meshtastic nodes can operate almost anywhere on the globe that it's legal to do so, though you may need to switch to an antenna tuned for different frequencies used elsewhere.

2. UPGRADE TO A PORTABLE NODE

USB power is not terribly convenient, so let's make our node a little easier to carry around.

For this handheld node, we're going to add a 3D-printed case and battery, and optionally a BME/BMP280 temperature sensor.

2a. From printables.com/model/286664, download and print the four STL files for the case: the front, back, frame, and reset button (Figure L on the following page). You can also purchase these from vendors listed on the Printables page.

2b. (optional) If you'd like to add the BME/BMP280, now's the time. Solder it to the back of the WisBlock, connecting VCC to VIN, GND to GND, SCL to SCL, and SDA to SDA (Figures M and N). Verify the pin order on your specific BME/BMP280 as they may differ. If you're using the WisBlock BME680, click it in place on the back of the baseboard onto Slot D.

2c. Unplug the two antennas from the WisBlock.

2d. Slip the Bluetooth PCB antenna into the slot on the right side of the frame piece, with the wire sticking down (Figure O).

2e. Put the reset button into the frame, and then press-fit the WisBlock into the frame. Depending on your printer tolerances this might be a tight fit.

2f. Fit the SMA connector into the cutout at top left. This is, again, a tight fit and may require some pushing, shoving, and slight flexing of the PCB to slide the SMA port past the WisBlock.

2g. Loop the SMA wire around and reconnect it to the LoRa port. Fish the Bluetooth wire around the back and plug that into the BLE port (Figure P).

2h. Put the lock washer (from the WisBlock kit) and nut on the SMA port. Add the SMA antenna.

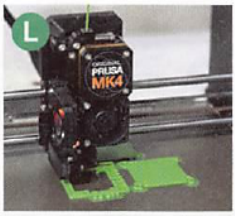
2i. Connect the battery to the bottom left port (closest to the USB), threaded from the backside (Figure Q).

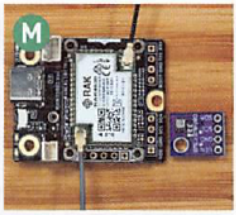
WARNING: Make sure your battery is wired so that the ground wire (black) is on the outside as pictured. If it's wired incorrectly, you can fix it. The JST plugs include small locking tabs; lift these carefully, remove and swap the red and black wires.

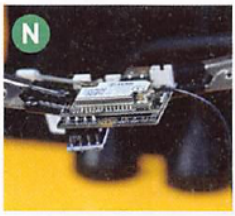
2j. Put the battery into the back case, and fit on the front case (Figure R).

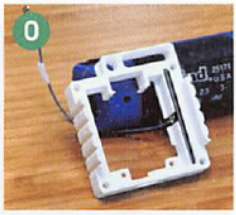
2k. Use the four M3 screws and nuts to secure your new node case together (Figure S).

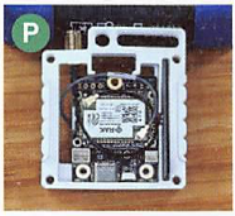
2l. Hit the reset button on the bottom and the node should boot up. The green activity light should begin to blink and flash.

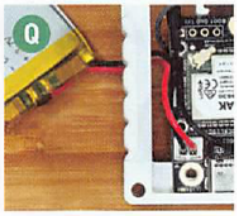












2m. Open the Meshtastic app and enjoy your new portable node!

2n. (optional) If you installed the BME/BMP280, go into Settings → Telemetry. Under the Sensor Options section, turn on the Enabled option and save (your node will reboot) (Figure T). Now the sensor should automatically be detected and broadcast its sensor data, which is visible under Environmental Metrics Log (Figure U).

3. POTENTIAL UPGRADES

One common feature of many Meshtastic nodes is integrated GPS, and this 3D-printed case supports the RAK12500 GNSS module. (It's also got room for the RAK18001 piezo buzzer.)

Meshtastic also supports a number of I²C sensors, including current/voltage sensors, light sensors, and even entire weather stations; explore options at meshtastic.org/docs/configuration/module/telemetry.

Another popular upgrade is solar power. On the front of the RAK WisBlock you might have noticed that there is a second, smaller power port, which will accept any 5V solar panel via a JST ZHR-2 connector (female, 1.5mm pitch).

MESH AROUND AND FIND OUT

Meshtastic puts reliable, license-free messaging, location, and telemetry in your hands when networks go dark. Range is terrain-dependent, but the mesh turns ordinary radio technology into a resilient neighborhood of relays — handy for hikes, festivals, and storm season alike. Because it's open source and runs on everything from ready-made cards to DIY dev kits, you can start small, add a rooftop or solar node to widen coverage, and keep chats private with encryption by default.

There are hardware options to suit most needs, including pre-made, ready-to-go nodes for friends and family who are less excited about wiring and soldering. and lots of DIY projects out there for the more adventurous.

INFRASTRUCTURE NODES

Remember, each Meshtastic node extends the total range of your mesh by acting as a repeater. Not willing to stand on a mountaintop? A simple solar node acting as fixed "infrastructure" might be just the thing you need to bridge the gaps of a mesh. The unit shown in Figures V and W is based on the RAKwireless WisBlock Starter Kit (non-mini), with a plastic box, solar panel, 10,000mAh battery, antenna, and magnet — all hot-glued together for under $100.

Creative Meshtastic users who were not so concerned about waterproofing have even used the plastic shipping cases a node comes in!

DIVING DEEPER

Take your Meshtastic node out for a weekend, turn off your cell service, and see how it works. Make sure to walk through the Getting Started Guide to learn about the other settings and options available (meshtastic.org/docs/getting-started). You'll especially want to set up your own private channels with encryption keys, to keep your conversations private and secure. Check out the Meshtastic Community (meshtastic.org/docs/community) and see if there's a local user group in your region.

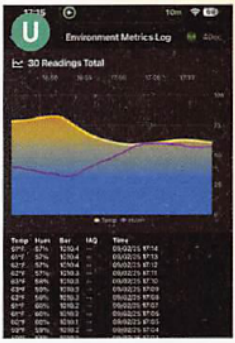
If you enjoyed this project, come join us on the Meshtastic Discord (discord.gg/meshtastic), say hi, and let us know how the build went.

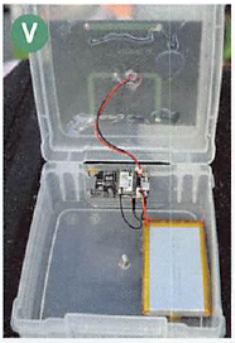
Welcome to the Mesh!













LESSONS ABOUT MESHIN'

SKILL BUILDER: GO LONG WITH LORA
Learn how this low-power, wide-area network IoT protocol works. makezine.com/article/technology/go-long-with-lora-radio

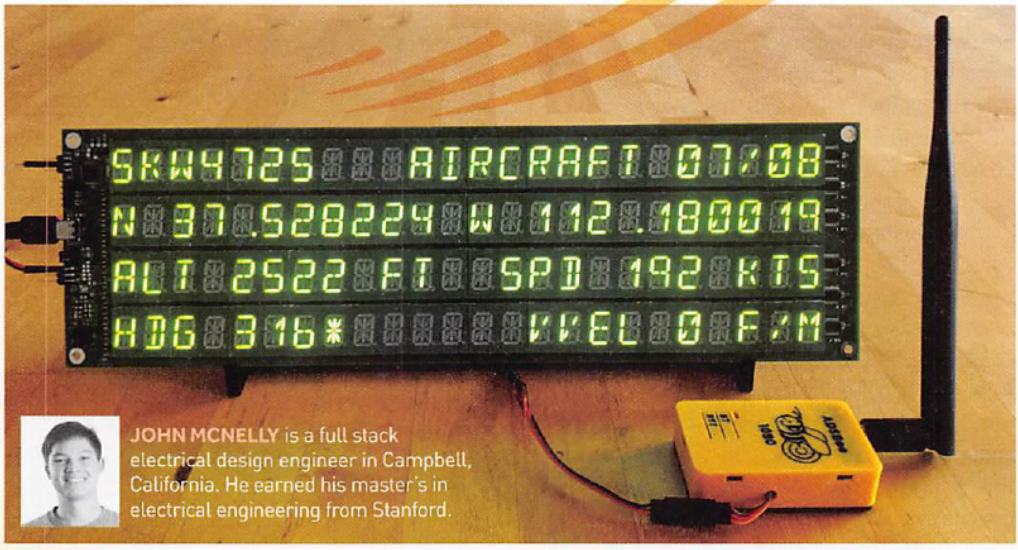
DIY SWARMBOTS
Use Particle's mesh microcontrollers to build a swarmbot network of Thunder Tumblers! makezine.com/projects/diy-swarmbots

"ARMACHAT" LORA COMMUNICATOR
A DIY, Armageddon-ready, extremely low-power LoRa text-messager. makezine.com/projects/armachat-lora-communicator

HOW TO SPEAK AIRPLANE

Track aircraft with a low-cost, open source, embedded ADS-B receiver that you can build into anything

Written by John McNelly



JOHN MCNELLY is a full stack electrical design engineer in Campbell, California. He earned his master's in electrical engineering from Stanford.

Adobe Stock-BAIVECTOR and Saifcreator, John McNelly

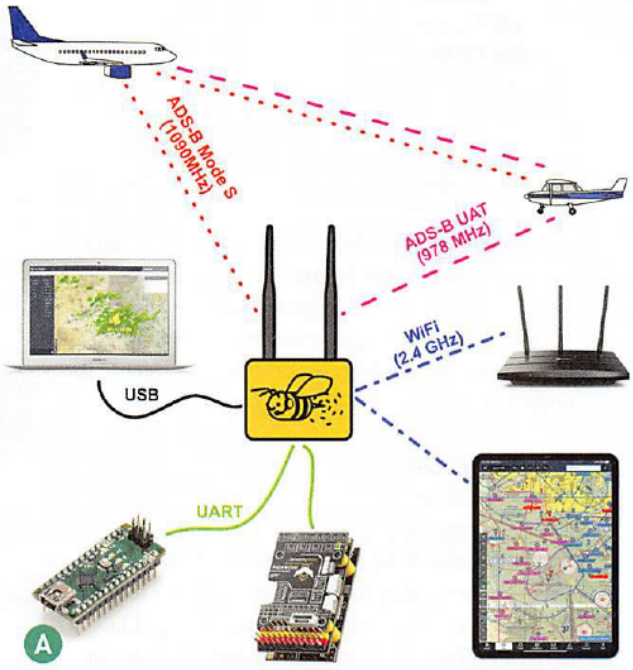
What is ADS-B? It's what the airplanes speak!

Automatic Dependent Surveillance — Broadcast (ADS-B) is a protocol used by aircraft for transmitting their location and other information to other aircraft and to air traffic control. ADS-B is transmitted on 1090MHz via a pulse position modulation protocol called ***Mode S***, and has become an internationally adopted standard over the decades since it was originally introduced.

Because the protocol is unencrypted, aviation buffs have created numerous hardware and software solutions for decoding, saving, and sharing ADS-B data from around the world. Aggregators for this data include community websites like airplanes.live and businesses like FlightAware and FlightRadar24. ADS-B data gets used for everything from airline route planning to air crash investigations to tracking Elon Musk's

jet. It's also used for academic studies, like validating global weather models using aircraft wind data (obrhubr.org/adsb-weather-model) or mapping GPS spoofing and jamming incidents around the world (waas-nas.stanford.edu).

Most of the ADS-B data on community exchanges is received and uploaded to the internet by a network of thousands of community-maintained ***ground stations***. These are usually based on a Raspberry Pi single-board computer connected to one or more software defined radio (SDR) dongles that are used to receive the 1090MHz ADS-B signal. Decoding on these stations is done with an open source software decoder called **dump1090**, which extracts ADS-B messages from the raw I/Q data stream provided by the SDR.



MEET ADSBee

OK, so what is ADSBee? I started the ADSBee project to make small, low-power ADS-B receivers more accessible and open source. Most existing open designs use the aforementioned SDR dongles and external compute device, making them rather bulky and power hungry, and less well suited to embedding into drones, portable battery-powered devices, or other embedded projects. Commercial off-the-shelf ADS-B receivers exist, but most are based on rather expensive FPGAs and cost $350 or more. ADSBee is an open source project that uses a new demodulator that I designed to enable ADS-B decoding in a small, low-power, and low-cost device.

With a novel demodulator based on the RP2040 microcontroller and its PIO peripherals, the **ADSBee 1090 board** provides commercial-grade ADS-B decoding without the need for an FPGA or external compute. This makes it affordable enough to integrate into a wide variety of projects, while still having low power draw (about 1W) and a small footprint. ADSBee can provide aircraft data over a variety of interfaces (Figure A), including USB, UART, Wi-Fi, and Ethernet (with an accessory board). This makes it easy to send aircraft data to anything from tablets to online databases to drone autopilot flight computers.

OPEN SKIES

The best part about the ADSBee project is that it's open source! Unlike ADS-B aircraft transponders, which are subject to a strict certification regimen, there are no certification standards for portable ADS-B receivers (those not permanently installed in full-size aircraft). This means that commercial receivers, despite their high price tag, use proprietary closed source firmware and hardware that can be slow to evolve and sometimes contain bugs that can lay hidden for years.

By open-sourcing the receiver schematic and firmware, ADSBee has drawn upon a supportive community of beta testers to squish bugs fast and incorporate everyone's best ideas. ADSBee users have proposed improvements ranging from helpful new configuration commands to improved aircraft position filtering algorithms, and it seems like every time ADSBee is baked into a new application, a flurry of new ideas is quick to follow.

BEE BETA

The ADSBee project is 2 years old and has been in a beta phase since late 2024. Since the beta units were released, firmware features have been added and improved based on user feedback.

Most recently, in July 2025 we rolled out new

dual-band capable hardware in the form of the **ADSBee 1090U**, which includes an additional radio and microcontroller in the same form factor PCB (Figure B). While the original ADSBee 1090 decodes ADS-B packets sent on the standard 1090MHz frequency, about 5% of aircraft in the United States transmit ADS-B on a separate frequency band of 978MHz, using a protocol called ***UAT (Universal Access Transceiver).*** The new dual-band ADSBee can receive messages from aircraft operating on both frequencies simultaneously. It could even be used to decode other aircraft protocols in the future.

PANTS DROP!

More new ADSBee-powered hardware is in the works! Accessory boards for the ADSBee were originally going to be called "hats," but beta testers insisted they be called "pants" after the name of my small business, Pants for Birds LLC.

So, a **PoE pant** (for connecting and powering an ADSBee over Ethernet) is already in production, and plans are being made for a **GNSS pant** (adds a GPS receiver for real-time receiver positioning data) and a **battery pant** (for portable operation without a USB cable).

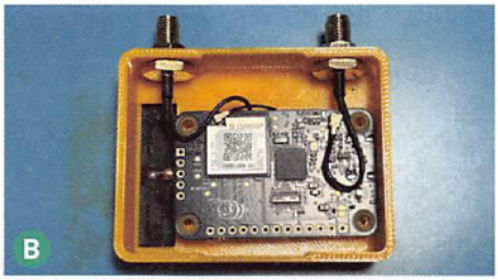
The ADSBee connects to pants via standard 0.1"/2.54mm pin headers, so it's easy to create a custom pant using simple protoboard. Future firmware updates will include custom GPIO functions to enable custom pants for activities like battery voltage sensing or charge control. Figure C shows an ADSBee 1090U with a PoE pant in a 3D-printed enclosure, ready to be mounted in an attic for optimal reception!

And for projects with tighter requirements, the single-band **ADSBee m1090 solder down module** (Figure D) can be directly integrated into PCBs. We're excited to see what people will build with it!

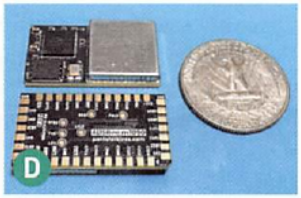
BUILDING WITH THE BEE

I am continuously surprised and delighted by the projects that people are cooking up with their ADSBees. I've seen PoE and solar-powered ground stations, beautiful LED displays of aircraft in the area, portable handheld receivers, and ADSBees directly integrated into drone autopilots, for traffic alerting and miniature cockpit displays.

Here are a few projects from our beta testers:









John McNelly, Petr Čada, Ryan Null

ADSBee on an FPV Airplane

Petr Čada, Czech Republic

youtube.com/watch?v=_bxTrD90Kmw

Petr integrated an ADSBee into his T1 Ranger first-person-view (FPV) R/C aircraft. Air traffic data from the ADSBee is reported to the flight controller over the MAVLINK protocol, and gets added to the pilot video feed as part of the On-Screen-Display (OSD) overlay. Figure E shows an aircraft being detected by the ADSBee while it flies a few thousand feet overhead (both the UAV and the passenger aircraft are operating in approved areas). The airliner can be seen in the center top of the frame, below the distance to home indication.

14-Segment Flight Display

Steve Mo, California

Steve put together a beautiful 14-segment display that reads aircraft data from the ADSBee and shows information about nearby aircraft (see it on page 60). The display cycles through each of the aircraft within range and displays latitude/

longitude, altitude, vertical rate, airspeed, and heading. This device was lovingly crafted with some SMT work and a *lot* of through-hole hand soldering. The display is powered by an STM32F103 microcontroller running custom firmware that reads aircraft data from the ADSBee via MAVLINK1 and a UART connection.

Solar/Battery ADS-B Ground Station

Ryan Null, California

Ryan wanted to mount an ADSBee on the roof without a PoE cable run, so he designed a solar- and battery-powered base station (Figure F). The ADSBee's low power draw allows it to be powered with a relatively small solar panel and LiPo battery bank, even in partial shade. Ryan is now working on adding a larger panel, MPPT tracker, and larger battery to enable his ADSBee to last through the dark winter months.

Ryan's ADSBee connects to a Wi-Fi network to feed air traffic data from the surrounding area. Beta testers are excited about using larger external Wi-Fi antennas with improved directional gain to enable solar/battery receivers to be mounted further from their Wi-Fi networks, in spots with better ADS-B reception or longer hours of sunlight.

FPVToys Cockpit Display with ADSBee

Jeff Hendrix, Colorado & John McNelly, California

Jeff makes miniature cockpit flight displays for R/C aircraft that are fully functional! He recently added an ADS-B moving map display that renders live air traffic and runways of nearby airports onto a miniature instrument panel. This was so cool that I had to put together a miniature cockpit to show it off at Open Sauce (Figure G), with Jeff's help on the electronics. The shell is 3D printed, with the screens mounted from the inside. An ADSBee sits in the back and is connected to 1090MHz and 978MHz antennas. The map can be scrolled around using a custom keypad. Open Sauce attendees were shocked to see the gauge cluster come to life when they picked up the instrument panel from the display table!

The instrument panels and controller are from Jeff's business, FPVToys.net. He recently released a firmware for the display controller that is compatible with ADS-B input from an attached



F



G

flight controller. I'm excited to see ADSBee-equipped mini cockpit displays taking to the sky!

NOW BOARDING

Makers are coming up with new ways to use ADSBee every week. On our Discord server, beta testers can share projects and ask questions. Our monthly email newsletter brings you firmware updates, cool new projects, and adventures in manufacturing open source hardware. We'd love to have you along for the ride.

RESOURCES

- pantsforbirds.com/adsbee-1090
- github.com/CoolNamesAllTaken/adsbee
- discord.com/invite/KRqVT9sSVW

Threadboard

Knit a working breadboard from yarn and conductive thread!

Written and photographed by Alanna Okun



ALANNA OKUN is a writer, textile artist, and video game designer living in New York since 2008. She is currently a master's candidate at NYU ITP.

Michael Dornnett

The solderless breadboard is a cornerstone of electronics education and prototyping — a rectangular waffle that conducts electricity in configurations of columns and rows, for testing and determining where wires and components should connect to one another in a circuit, before more permanent measures like soldering. They're grids, the bare canvas of so many kinds of patterns. Generally they're made of plastic, but I wanted to see if I, a lifelong knitter, could make a functional one out of my personal favorite material: yarn.

Textiles-as-electronics have a rich history in their own right. As I've learned more about programming and circuitry, what made it all start to gel was when I began to see technical schematics as patterns, like in knitting or needlepoint. Row by row, stitch by stitch, the pitter-patter rhythm of the knits and the purls — the 1s and the 0s — adding up to the whole. The breadboard itself, the location of all of these experiments, felt less like a hostile piece of equipment to be tamed and more like an inviting canvas.

This project is appropriate for anyone who is passingly familiar with knitting, comfortable enough picking up a sewing needle, and interested in electronics (no experience necessary in that arena).

TIME REQUIRED: 10 Hours

DIFFICULTY: Easy–Intermediate

COST: $30–$40

MATERIALS

- **Yarn, DK weight, in a natural fiber like wool or cotton (~50 yards/45 meters)**
- **Two contrast color yarns, DK weight (several yards/meters each)**
- **Conductive thread (~50 yards/45 meters)** e.g. two spools of stainless steel thread, Adafruit 640, adafruit.com
- **Sewable coin cell battery holder with appropriate battery, 1.5V–3V**
- **LEDs (1–3) or DC hobby motor** rated for appropriate battery voltage

TOOLS

- **Knitting needles, size 6** or whatever size matches your yarn. Gauge doesn't matter a ton here but shouldn't be too tight or loose.
- **Alligator clips**
- **Tapestry needle**
- **Scissors**
- **Multimeter (optional)** You don't *need* a multimeter, especially if you've never used one before, but it can be very helpful for troubleshooting.

CAUTION: Whether you're new to circuitry or otherwise, be sure to take common-sense safety precautions: ensure your components are rated correctly for the battery you use, turn the power supply off while you're working and when the finished piece isn't in use, and don't get it wet. There is a hypothetical risk of flame (there are reasons we don't typically make breadboards from cloth), but it's more or less that of leaving a sweater near an extremely weak lightbulb, so do with that what you will.

MAKE YOUR SOLDERLESS THREADBOARD

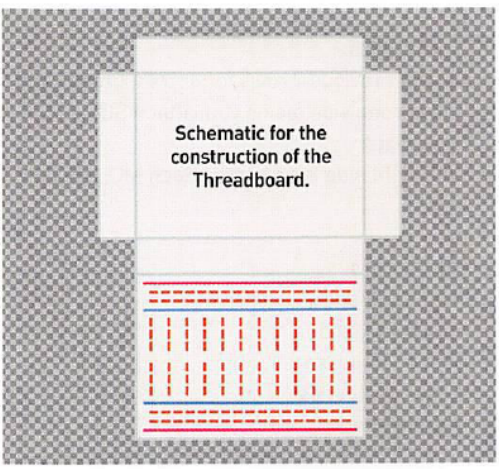
1. KNIT THE THREADBOARD

Your approach to this section will depend on your comfort level with knitting. If you're new to reading patterns or otherwise intimidated by knitting jargon, all you really need to come away with is a knitted white rectangle — the construction and even the stitch counts are less important than just getting a base for your conductive embroidery.

If you'd like to simplify this project further, you can skip knitting and stuffing the body and just focus on knitting the face.

BODY

- Cast on (CO) 45 stitches (st) with main color (MC)
- Knit in stockinette for 10 rows



Schematic for the construction of the Threadboard.

- With wrong side facing you, knit 1 row
- At the end of the row, CO 12 st — 57 st total
- Knit 12 st (k12); slip 1 st knitwise (sl1k); k45; sl1k; CO 12 st — 69 st total
- Purl across (p69)
- k12, sl1k, k45, sl1k, k12
- p69
- Repeat previous 2 rows for 30 more rows total
- Cast off 12st knitwise; sl1k; p45; sl1k; cast off 12st knitwise; break yarn
- With right side facing you, reattach yarn to rightmost knit stitch and k45 across
- p45
- p45
- Repeat these 2 rows for 10 more row
- With wrong side facing you, k45

FACE

- k45 (MC). Break yarn.
- With wrong side facing you, attach secondary color 1 (SC1); p45; break yarn
- With right side facing you, attach MC; p1k1 across
- p45
- p1k1 across
- With wrong side facing you, attach secondary color 2 (SC2); p45; break yarn
- With right side facing you, attach MC; p1k1 across
- p45
- p1k1 across
- Repeat previous 2 rows for 6 rows
- p45
- p45
- p45
- p1k1 across
- Repeat previous 2 rows for 6 rows; break yarn
- With wrong side facing you, attach SC2; p45; break yarn
- With right side facing you, attach MC; p1k1 across
- p45
- p1k1 across
- Attach SC1; p45, break yarn
- Attach MC; k45
- Bind off knitwise
- Weave in all ends
- Block if needed

2. EMBROIDER WITH CONDUCTIVE THREAD

Like the last section, it's less important that you follow these directions precisely and more important that you get some horizontal and vertical lines of thread sewn onto your board in some configuration.

The most crucial thing is to make sure rows of conductive thread don't unintentionally overlap; that could cause a short or otherwise uncooperation from your circuit, and it's why we're tying all these knots.

- Thread tapestry needle with about 48"/120cm of your conductive thread. Double it and secure the end with a double knot — this stuff doesn't always love to stay tightly bound to itself, and it's fairly easy to tangle, so take extra care.
- Starting at the bottom-left corner of the face of the breadboard, whip stitch up the leftmost rail using the purl ridges as a guide — go over, around, and under the purls in order to create a firm path that's neither too tight nor too loose (Figure A).
- When you reach the end of the rail, turn and whip stitch back over the stitches you just made to reinforce them.
- Tie the end of the working thread to the knotted tail on the wrong side, and secure with a double knot. Do not pull too tight.
- Trim ends, being careful not to sever the knot nor leave any loose strands that could cause a short circuit.
- Repeat up and down the near left rail (Figure B).
- Repeat up and down the near right rail.





- Repeat up and down the far right rail.
- Thread the tapestry needle with about 16"/40cm of conductive thread. Double it and secure the end with a double knot.
- Starting at the top left corner of the face of the breadboard, whip stitch in the same manner from the leftmost edge of the inner body until the center gap.
- Turn and whip stitch back over the stitches you just made to reinforce them.
- Tie the end of the working thread to the knotted tail on the wrong side, and secure with a double knot.
- Trim ends.
- Repeat across the gap to the rightward rail and all the way down the inner body on both sides, tying a new knot and trimming any excess from each row (Figure C).

3. TEST AND FINISH

To test, attach one end of an alligator clip to the positive terminal of your coin cell battery holder (with functioning appropriate battery) and the other to the "power" rail of the threadboard.

Attach one end of an alligator clip to the negative terminal of your coin cell battery holder and the other to the "ground" rail of the threadboard (Figure D).

Appropriately place the two legs of an LED, or the two terminals of a DC hobby motor, to test conductivity (or use a multimeter).

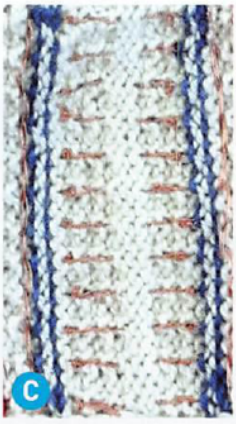
If the rails are functional, use another pair of alligator clips to test each row — run power to one side and ground to the other, from each of the rails (Figure E), and test your components there.

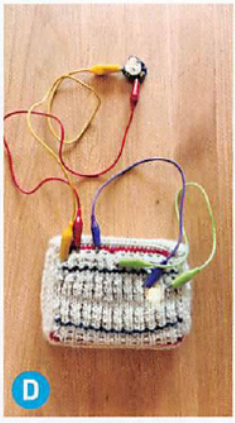
To finish, stuff the piece with cotton batting, fabric scraps, or other natural fiber fill. Seam up all loose edges with single crochet or mattress stitch (Figure F). Weave in and trim all ends.

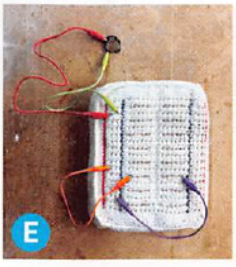
Your Threadboard is complete.

PLUG AND PLAY

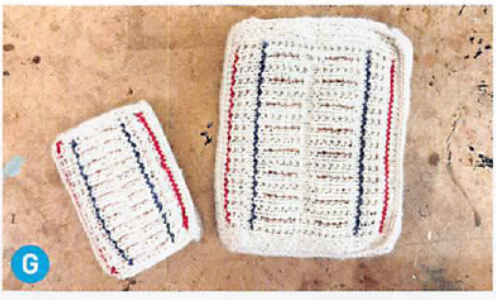
The funny thing about making a breadboard — out of any medium — is that when you're done, you're actually at the beginning. By nature, it's an item meant to be a blank slate, to be inscribed with circuit pathways. It's a tool in itself, and what you do with it next is entirely up to you. I'd be interested to see one of these featuring a circuit beyond a simple switch and load, utilizing resistors, capacitors, ICs, and all the other bits and bobs that make up electronics. (As always, practice due caution.)











I've made several threadboards now, at varying scales. The original is closer to a lumbar pillow in size (Figure G, at right), and I mostly use it as such! A smaller board, based on the measurements here, is more portable and easier to use for classroom demonstrations. I like it as a teaching tool — for making both electronics and textiles more legible to people already interested in one or the other — and also as a celebration of the absurd. Nobody asked for me to knit a breadboard, but I did anyway, and you can too.

3D Printing Folding Geometry

Make an elegant 3D-printed fabric bag that folds like origami

Written and photographed by Sophy Wong





SOPHY WONG is a designer, costumer, and maker creating artistic wearable technology. Using digital fabrication techniques like 3D printing and laser cutting, her work highlights the intersection of technology and design for the human body.

One of the most powerful tools in the studio of any designer today is the 3D printer. Whether making prototypes or finished products, 3D printing supercharges a designer's power to create new things with speed and complexity. As a designer of costumes and fashion, I am always interested in new ways to create wearable design objects, and learning how to 3D model for 3D printing has been a game changer for my projects.

Recently, I've been exploring the technique of 3D printing directly onto fabric. This process involves printing small, repeating shapes onto a piece of thin mesh fabric which is inserted between layers of the print. A quick internet search will reveal many examples of this innovative and experimental technique being used to create flexible pieces of 3D-printed materials, often with complex surface patterns like dragon scales. In my projects, I have used the process to create flexible headpieces, an LED glove, and even a fully wearable garment: my 3D-printed dress. I've also made flexible 3D-printed appliqués to sew onto the surfaces of costumes and fabric garments.

FOLDABLE GEOMETRY

While working on these projects, I became enchanted by how 3D-printed fabric pieces transform when they bend and move. I began to wonder what could be produced by combining 3D printing on fabric with concepts of folding geometry, like origami. After many experiments and prototypes, I've learned that 3D-printed fabric is a perfect match for working with foldable geometry. The fabric element holds up well to repeated bending, and the printed designs allow for precise fold shapes. Also, with every print there is a delightful transformation to experience: the finished 3D print is completely flat when it comes off of the printer, and the magic happens when you fold the print into its final shape.

In this article, we'll look at a design I created for a foldable 3D-printed fabric bag. I'll show you how I designed it, walk through the process of printing it onto fabric, and share some tips and tricks that will help you make your own bag at home!

TIME REQUIRED: A Weekend

DIFFICULTY: Intermediate

COST: $1–$2

MATERIALS

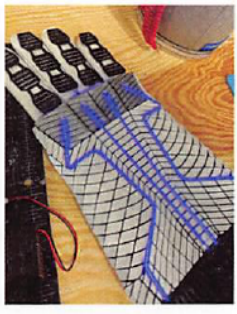
- **Thin nylon mesh fabric, aka "tulle"** You'll need a 10" square for the project, plus more for tests.
- **Satin cord**
- **3D printer filament** of your choice. I print in PLA.

TOOLS

- **3D printer**
- **Computer** Download the free 3D models for this project at printables.com/model/1332875.
- **Scissors**
- **Sanding stick, pointed**
- **Painter's masking tape**











THE DESIGN

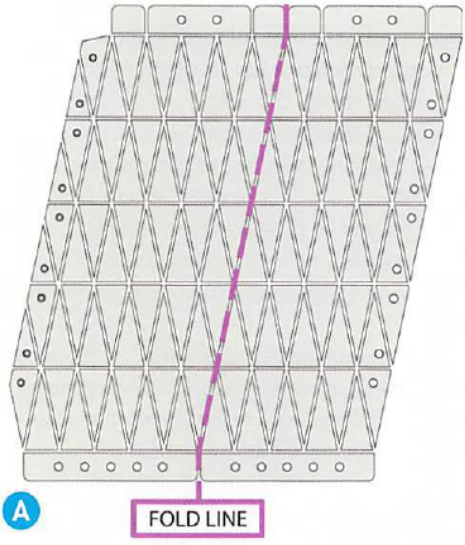
In this design, there are two aspects that create the form of the finished piece: the repeating triangle shapes that make up the 3D-printed fabric, and the larger triangular shape of the bag itself.

First, let's look at the overall shape (Figure A). When I had the idea to design a folding bag, I started by looking at product packaging examples for inspiration. I was instantly drawn to the classic tetrahedron-shaped carton, an iconic piece of packaging design from the 1950s (Figure B). When flat, the shape is a series of connected triangles, and when folded, it becomes a four-sided pyramid. In my version, I've extended the shape to create a longer, more pointed pyramid shape. This shape makes better use of typical square-shaped 3D printer beds, increases the bag's capacity, and has a more elegant proportion.

Once I had decided on the finished shape of the bag, I prototyped it with paper to work out the actual size and shape of each folded panel.

To create the repeating triangle shapes that make up the printed fabric, I based the triangles on the proportions of the bag panels themselves, rotating them to make a tessellating pattern.

To fasten the folded sides of the bag together, I added printed pegs and matching holes that snap together along the side seam. The bottom of the bag is held closed with a printed tab that's inserted into holes at the bottom edge. Finally, a piece of cord is laced through holes at the top of the bag, and a 3D-printed slide is added to the cord for opening and closing the bag.

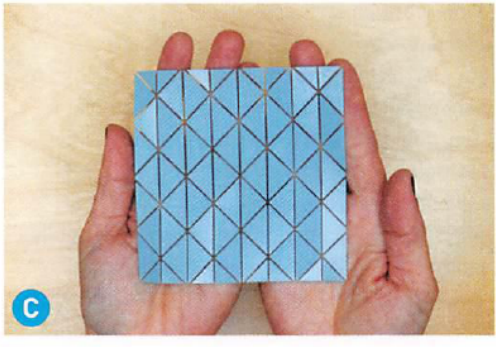


A

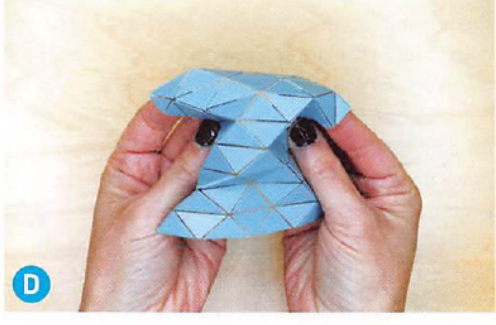


B

Tetra Pak developed their iconic tetrahedral milk carton in the 1950s.



C



D

HOW TO 3D PRINT ON FABRIC

Here's the process I use for printing onto fabric. This is an experimental technique that goes "beyond the manual" for any off-the-shelf 3D printer. Before trying it, you should already be very comfortable with basic 3D printing, and slicing and preparing 3D models for print. You should know how to stop your printer safely, and how to pause and resume a print. When printing on fabric, always monitor your printer continuously and be ready to stop the print if anything unexpected happens.

Start with a few test prints to get comfortable with the process and dial in your printer. Practice with the *fabric test.stl* model (Figures C and D).

Tetra Pak CC BY-SA 2.0

1. PREPARE YOUR FABRIC

Cut a piece of the nylon mesh fabric. If necessary, press it completely flat with a cool iron to remove all creases. Use a 6" square of fabric for the Fabric Test, and an 8" square for the Bag model.

2. LOAD 3D MODEL AND CREATE A PAUSE

In your slicer, create a pause after your first layer. Here's how to do it in UltiMaker Cura:

Load the provided 3D model into Cura, and place it in the center of the print bed. Click on the Extensions menu, then go to Post Processing and click Modify G-code to open the Post Processing Scripts window.

Click on Add a Script and select Pause at Height. In the Pause at Height options, set Pause At to Layer Number (instead of Height). Then set the Pause Layer to 1. Use the Park Print Head coordinates to move the printer head to the back left corner during the pause. Finally, make sure to set Standby Temperature to match your printing temperature. When finished, close the window.

NOTE: This script will stay active in Cura until you remove it in the Post Processing Scripts window.

3. SLICE THE MODEL

The prints shown here were all printed in PLA filament with the following basic print settings:

- Layer Height: 0.2mm
- Printing Temperature: 205°C
- Build Plate Temperature: 60°C
- Print Speed: 60mm/s
- Top Layers/Bottom Layers: 4 (no infill required)

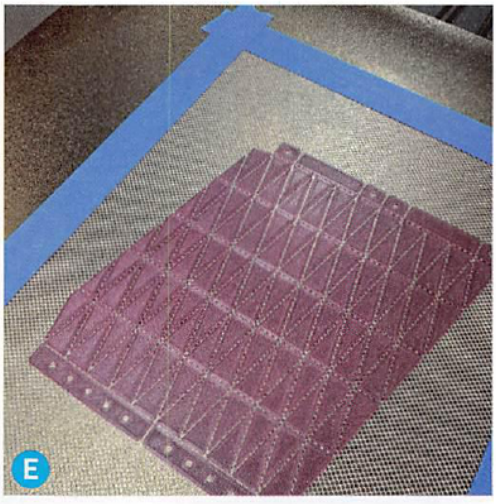
4. START PRINT AND WAIT FOR PAUSE

Keep your prepared fabric and painter's tape nearby while you watch the first layer to make sure it prints well and shows good adhesion.

5. INSERT THE FABRIC

Be careful here — the printer bed and extruder are both hot during this step! While the printer is paused, gently lay the mesh fabric over the print, centering it so there is at least an inch of fabric extending around all sides of the print.

Carefully tape down the edges of the fabric to hold it in place (Figure E). Be sure to cover all four edges of the fabric while ensuring that no tape covers any of the 3D-printed shapes. Press the tape down gently but firmly so that it is completely flat, being careful not to move the printer bed or the extruder. Then resume the print using your printer's controls.

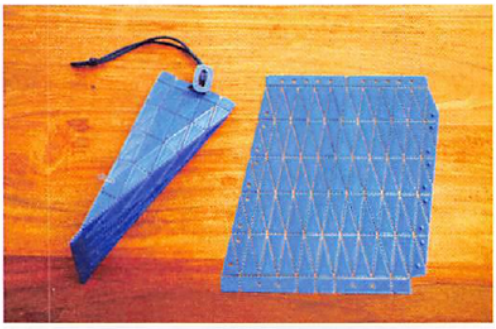


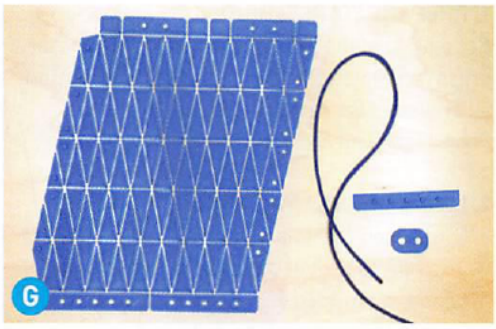


6. WATCH THE PRINT FINISH

Monitor the rest of your print to make sure the fabric is printed on smoothly and does not snag on the moving parts of your printer.

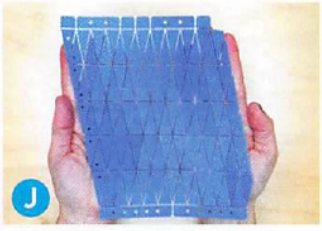
When the print is complete, let the printer bed cool down completely before you remove the print (Figure F). If it is difficult to remove, mist with isopropyl alcohol to encourage separation. Try not to yank the fabric against the printed shapes, as the delicate mesh may tear.























MAKE THE BAG

1. PRINT

Print the *3D Printed Bag* model onto nylon mesh fabric, using the method described above.

Print one each of the Bottom Tab Closure and Slide models separately, not on fabric (Figure G).

2. TRIM

Trim away the mesh fabric along all outer edges of the print (Figure H).

Use a pointed sanding stick to remove any mesh fabric from the inside of each circular hole in the print (Figure I). Try not to enlarge the holes too much as you do this.

3. FOLD

Place the print flat in front of you with the bottom side facing up, and the holes on the long edge to your left (Figure J). Fold the long edges together, placing the holes on the left side over the pegs on the right side (Figures K, L, and M). The five holes on the bottom edge should line up.

4. SNAP THE SIDES

Starting from the bottom edge and working your way up, snap each peg into its corresponding hole (Figures N, O, and P). If needed, use the pointed sanding stick to widen holes so that the peg snaps in securely.

5. CLOSE THE BOTTOM

Snap the Bottom Closure piece into place to hold the bottom edge closed. Again, use the sanding stick as needed to ensure the pegs fit securely (Figures Q, R, S, and T).

6. STRING THE CORD

Cut a 16" length of cord and lace it through the holes at the top edge of the bag (Figure U).

String both ends of the cord through the slide and tie a knot to make it into a loop (Figure V).

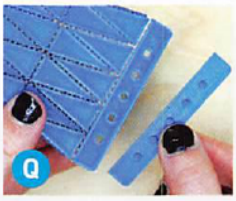
Your bag is complete (Figure W)!

FOLDED FASHION

To open the bag, pull the slide toward the knot and pop the sides in gently (Figure X). To close the bag, pull the slide away from the knot (Figure Y). It's great for stashing small valuables or a few essentials for a night out.

I designed this bag for a workshop at Emerald City Comic Con 2025. It was pretty challenging to work it out, even though it seemed like a "simple idea" in my head (as usual, Sophy!). I refined this design over quite a few prototypes and I'm really happy with how it came together. I'm feeling energized to explore this concept further and make more folding designs like this.



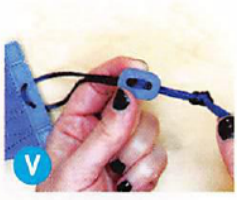


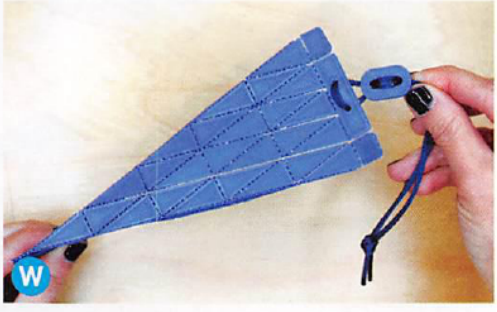


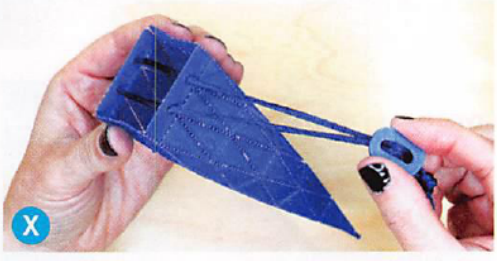










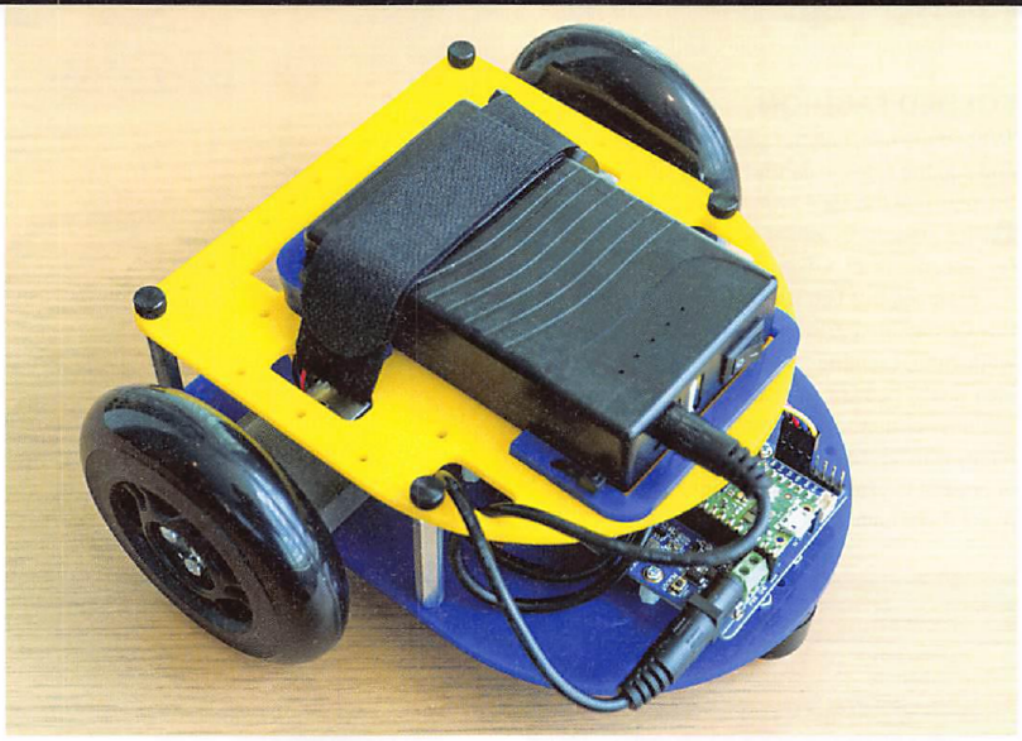


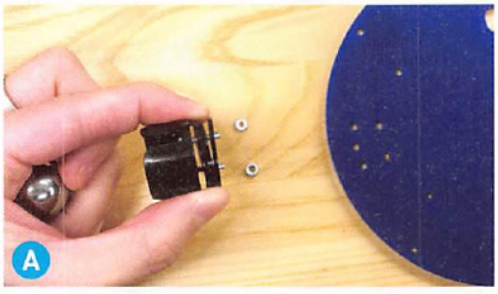


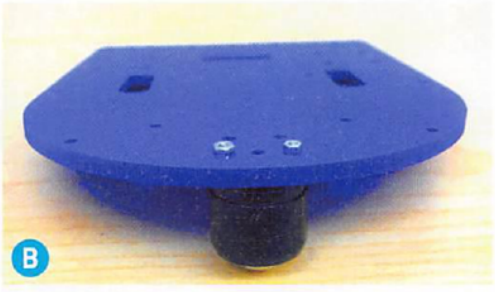
MBot

Build a robust wheeled robot platform and program it with MicroPython

Written by Peter Gaskell and Abhishek Narula







Abhishek Narula



PETER GASKELL is a faculty lecturer in the Robotics Department at the University of Michigan. His research focuses on robotics platforms for engineering education, as well as mapping, localization, navigation, and safe autonomous systems.



ABHISHEK NARULA is an engineer and designer in the Robotics Department at the University of Michigan. He creates robotics platforms for engineering education that are both engaging and expressive.

TIME REQUIRED: 4–5 Hours

DIFFICULTY: Moderate

COST: $240

MATERIALS

COMPONENTS:

- **MBot Robotics Control Board** $43 from mbot-dev.engin.umich.edu/mbot-robotics-control-board. Includes MBot Pico Board and pin jumper.
- **Power bank, 12V 12000mAh** Amazon B01M7Z9Z1N
- **USB-C cable** Amazon B07DC5PPFV
- **Gearmotors, 12V, 78:1 ratio, with extended motor shaft (2)** Pololu 3489, pololu.com
- **Magnetic encoders (2-pack)** Pololu 3499
- **Scooter wheels, 84×24mm (2)** Polulu 3275
- **Scooter wheel adapters for 4mm shaft (2)** Polulu 2672
- **Ball caster with ¾" metal ball** Polulu 955
- **Jumper wires, female to female, 6" (12)** Amazon B07GJG25DJ

FASTENERS:

- **Velcro cinch strap, 1"×12"** Amazon B088FH1289
- **Nylon cable ties, 2.5mm×200mm (4)** aka zip ties, Amazon B015HAV2NG
- **Nylon standoffs, M2.5×8mm (4)** Amazon B07LF8R399
- **Machine screws, M2×16mm (2) and nuts (2)** McMaster-Carr 92005A037 and 0591A111, mcmaster.com
- **Thread-locking screws, nylon patch, M2.5×6mm (4)** aka Nylok patch screws, McMaster 95911A164
- **Machine screws, M2.5: 6mm (8) and 8mm (8)** McMaster 90116A110 and 92005A069
- **Aluminum hex standoffs, 1½" (4)** McMaster 91780A171
- **Machine screws, 4-40 × ⅜" (4)** McMaster 90272A108
- **Thumb screws, 4-40 × ⅜" (4)** McMaster 91185A237
- **Heat-set inserts, brass, M2.5 (8)** McMaster 94180A321
- **Machine screws, M3×20mm (6)** McMaster 92005A128
- **Laser cut parts, from 3/16" acrylic sheet** Download the DXF files from mbot.robotics.umich.edu/docs/hardware/classic/building:
  - **Bottom plate** *Bottom-Plate.dxf*
  - **Top plate** *Classic-Middle-Plate.dxf*
  - **Battery clips (4)** *Battery-Clip.dxf*
- **3D-printed motor mounts (2)** STL from mbot.robotics.umich.edu/docs/hardware/classic/building

TOOLS

- **Computer with Thonny IDE software** free download from thonny.org
- **Soldering iron**
- **Screwdriver, #2**
- **Allen key, M2** aka hex key
- **Flush cutters**

In this tutorial, you'll build and operate the MBot Basic — a small, wheeled robot designed for hands-on learning. The MBot Basic is part of the MBot ecosystem, an open-source robotics platform created at the University of Michigan to make robotics affordable and easy to use.

Since its launch in 2014, MBot has become a cornerstone of hands-on robotics education at Michigan. Hundreds of MBots have been built and used to teach a wide variety of courses across both undergraduate and graduate levels, helping students explore core concepts in robotics, programming, control systems, and artificial intelligence.

By following this guide, you'll assemble your own MBot Basic and then learn how to program its main controller, the MBot Robotics Control Board (MRCB), using MicroPython. It's based on the RP2040 microcontroller — the same chip found in the Raspberry Pi Pico family.

Along the way, you'll discover how different sensors and features of the MRCB allow you to control your robot's movements with precision. Building the MBot Basic is a great way to explore the basics of robotics, programming, and electronics — all while creating something fun and functional. You can make it in an afternoon, not counting the laser cutting and 3D printing.

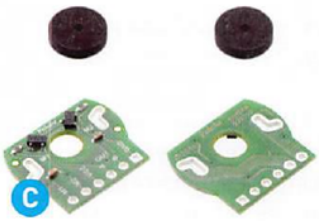
ASSEMBLE THE MBOT BASIC

1. ATTACH CASTER TO BOTTOM PLATE

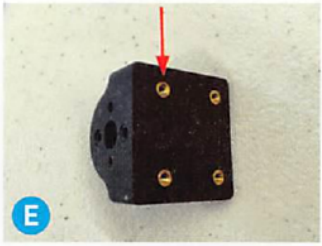
Take the caster with the ¾" metal ball (from Pololu) and the two included spacers. Stack both spacers on the caster, then use two M2×16mm screws to assemble the pieces together as shown in Figure A. Secure the caster to the acrylic bottom plate by tightening it with two M2 hex nuts (Figure B).

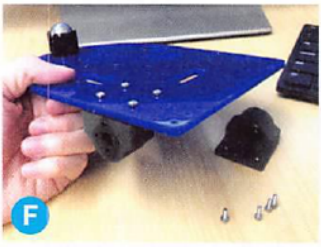
2. SOLDER ENCODERS TO THE MOTOR

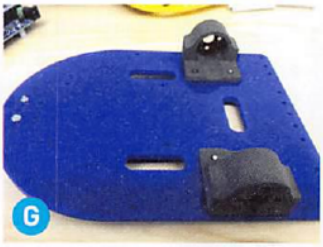
Gather the encoder kit (Figure C on the following page) and solder the encoders to the ends of the motors, as shown in Figure D on the following page. Make sure to solder the headers so that they face toward the motor shaft. Finally, place the magnet on the end of the motor. Be sure to leave a small gap between the motor and the encoder PCB so they do not touch each other.

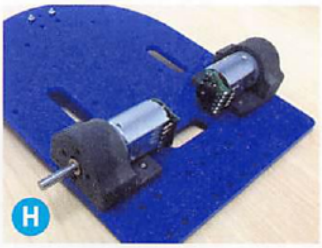


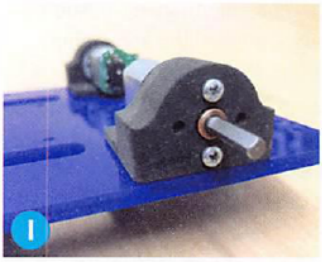


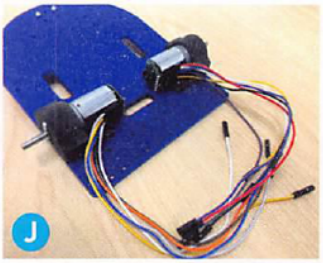


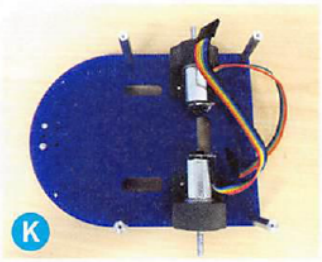












Abhishek Narula

3. INSTALL HEAT-SET INSERTS

Using a soldering iron or a heat insert tool, heat the M2.5 inserts until they're hot enough to melt into the plastic, then press the heated inserts into their designated holes in the 3D-printed motor mounts until they're flush with the surface (Figure E). Allow the inserts to cool and solidify before moving on to the next step.

4. ATTACH MOTOR MOUNTS TO BOTTOM PLATE

Use two M2.5×8mm screws to secure each motor mount to the bottom plate, making sure they're oriented as shown (Figures F and G).

5. MOUNT THE MOTORS

Place the motors into their mounts, making sure to align the holes so that the encoder headers are facing away from the caster (Figure H). Secure the motors to the mounts using M2.5×6mm Nylok screws (Figures I).

Finally, connect the 12 female-to-female jumper wires to each of the encoder headers' pins (Figure J).

6. ATTACH ALUMINUM STANDOFFS

Secure the 1½" aluminum standoffs to the bottom plate using four 4-40 screws, attaching them to the holes shown in Figure K.

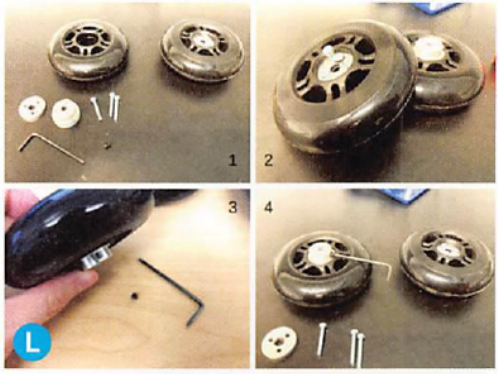
7. ATTACH ADAPTERS TO WHEELS

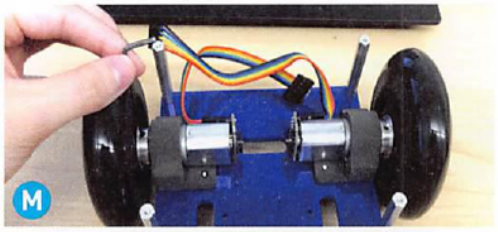
Press the machined wheel adapter into one side of the wheel, making sure part of it extends out. On the opposite side, press the flat metal piece into place, aligning the three holes.

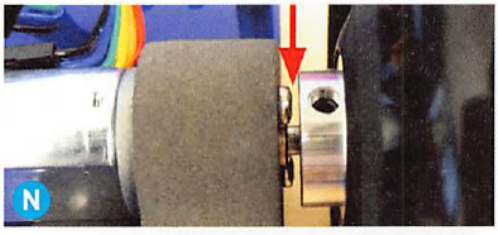
Secure the assembly by inserting and tightening three M3×20mm screws as shown in Figure L. Using a 2mm hex wrench, tighten the setscrews from the wheel adapter kit partway into the adapters.

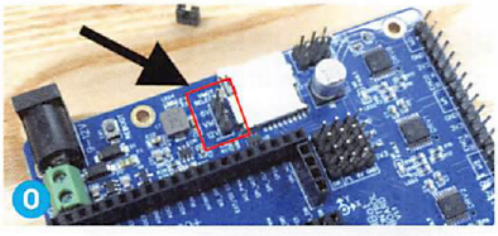
8. ATTACH WHEELS TO MOTORS

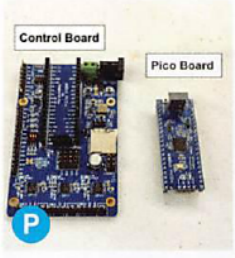
Put each wheel on its motor shaft and use the hex key to tighten the setscrews fully (Figure M). Be sure to leave a bit of a gap between the wheel and the motor screws (Figure N), otherwise the wheel hub will rub against these screws, causing excess friction or jamming the motors.

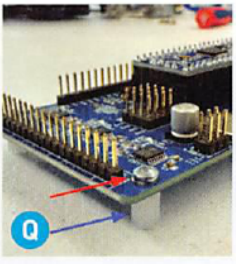


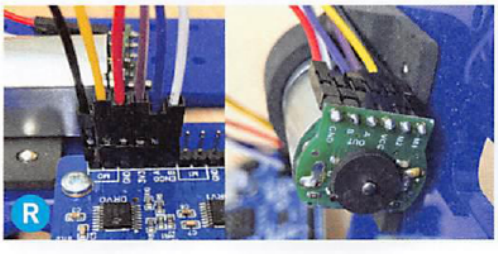












9. ASSEMBLE THE CONTROL BOARD

If it isn't already in place, connect the jumper on the MBot Robotics Control Board to the VM and 12V pins to match the 12V motors (Figure O).

Now mount the Pico board on top (Figure P). Finally, attach the 8mm nylon standoffs using four M2.5×6mm screws in the designated holes on the control board (Figure Q).

10. CONNECT MOTORS TO CONTROL BOARD

You can connect up to three motors to the Robotics Control Board, indicated by labels M0, M1, and M2 on the board. We will be using M0 and M1 channels for this build. Connect the left and right encoder wires to the control board as shown in Figure R, using the following tables:

Left Encoder	Robotics Control Board Motor Channel
GND	GND
B	B
A	A
VCC	3V3
M2	M0
M1	M0

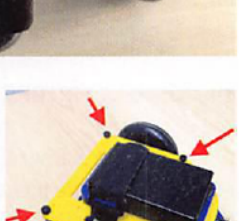
Right Encoder	Robotics Control Board Motor Channel
GND	GND
B	B
A	A
VCC	3V3
M2	M1
M1	M1















Abhishek Narula

11. MOUNT THE CONTROL BOARD

Attach the control board to the bottom plate using four M2.5×6mm screws (Figure S).

12. ATTACH THE BATTERY

Secure the four battery clips to the top plate with four zip ties as shown: three stacked at the front, one at the rear. Trim any excess zip tie using a flush cutter (Figures T and U).

Now secure the battery with the velcro strap (Figure V).

13. ATTACH TOP AND BOTTOM PLATES

Secure the top and bottom plates together using four 4-40 thumbscrews as shown (Figure W).

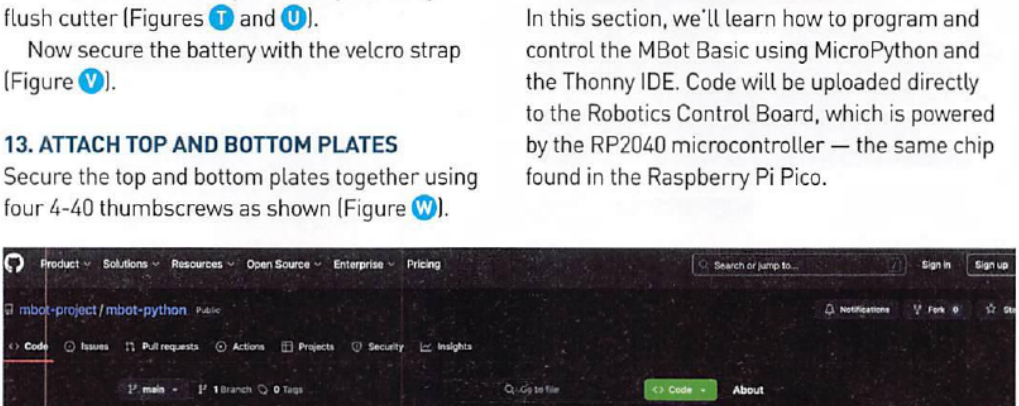
14. ATTACH POWER CABLE

Use the Y cable provided with the battery to connect the MBot Robotics Control Board to the battery (Figure X), then tuck the power cable neatly between the plates (Figure Y).

That's it, you've built your MBot!

PROGRAM YOUR MBOT

In this section, we'll learn how to program and control the MBot Basic using MicroPython and the Thonny IDE. Code will be uploaded directly to the Robotics Control Board, which is powered by the RP2040 microcontroller — the same chip found in the Raspberry Pi Pico.



INSTALLING MICROPYTHON ON THE MBOT

1. INSTALL THONNY

Download and install the Thonny IDE on your computer. Make sure to select the version appropriate for your operating system.

2. DOWNLOAD MBOT MICROPYTHON FILES

Go to the MBot GitHub repository at github.com/mbot-project/mbot-python. Click the green Code button and choose Download ZIP (Figure Z).

Unzip the file and locate the *mbot* directory — we'll upload this to the board later.

3. CONNECT THE MBOT BOARD

Plug the MBot Robotics Control Board into your computer using a USB-C cable, and open Thonny.

4. ENTER BOOTLOADER MODE

To install MicroPython, you need to put the board into ***bootloader mode***:

- Locate the BOOTSEL (Boot Select) and RST (Reset) buttons on the board (Figure Aa).
- Press and hold both buttons.
- While still holding BOOTSEL, release RST, then release BOOTSEL.

If done correctly, the RP2040 should appear on your computer as a USB drive — just like a flash drive.

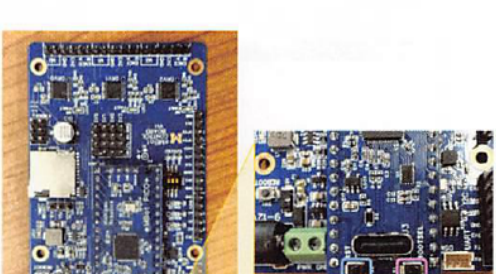
5. INSTALL MICROPYTHON

In Thonny, click the interpreter selector in the bottom-right corner, then choose Install MicroPython (Figure Bb).

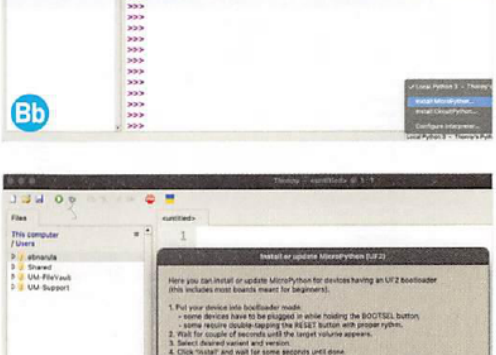
Select the RP2040 and the appropriate options as shown: family RP2, variant Raspberry Pi Pico / Pico H, and version 1.25.0. Click Install to complete the process (Figure Cc).

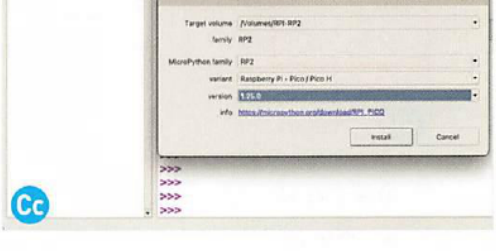
6. SELECT THE CORRECT INTERPRETER

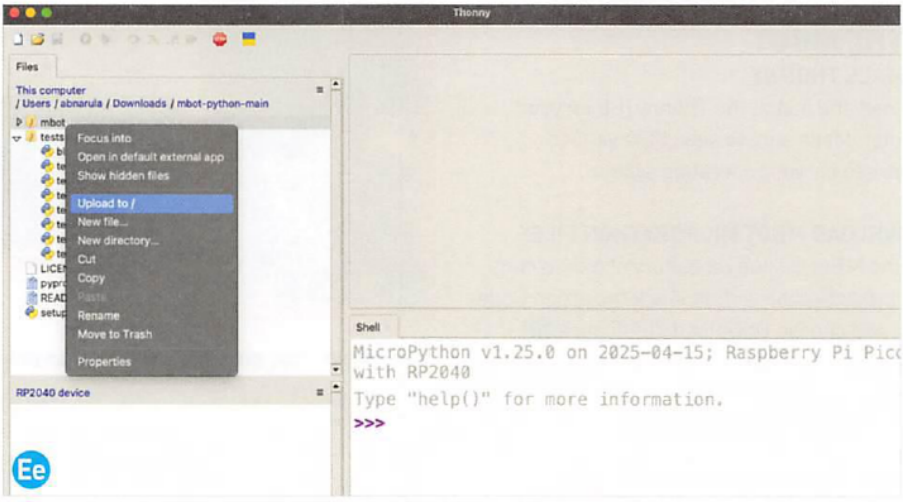
After installation, click the interpreter area again and choose MicroPython (RP2040). This will configure Thonny to run code directly on the MBot's control board (Figure Dd).



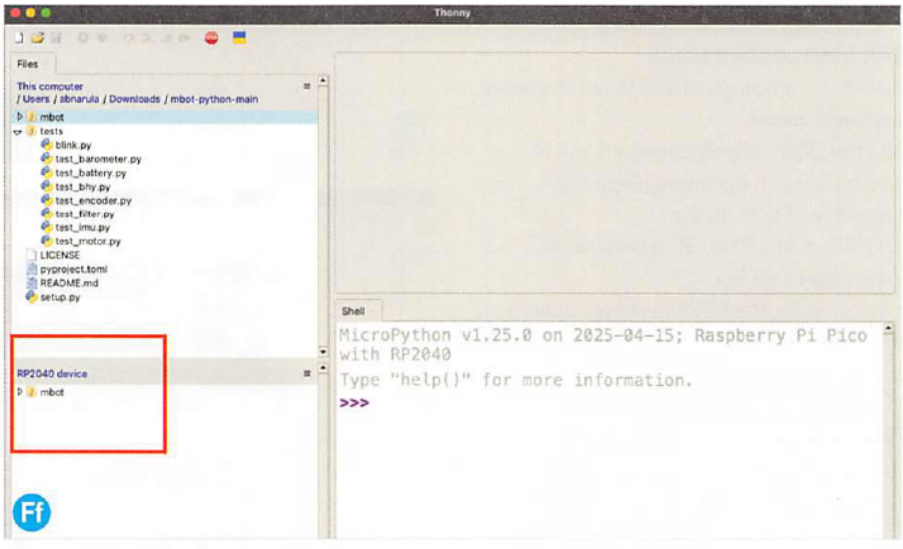








Ee



Ff

```
from mbot.motor import Motor

m = Motor(0, polarity=-1)  # Initialize motor 0 (or 1), with direction set by polarity
m.set(0.5)                 # Set motor speed to half (0.5 = 50%)
```

Gg

```
from mbot.pins import ENC0_A, ENC0_B, ENC1_A, ENC1_B
from mbot.encoder import Encoder

enc0 = Encoder(ENC0_A, ENC0_B, polarity=1)
enc1 = Encoder(ENC1_A, ENC1_B, polarity=-1)

count0 = enc0.read()
delta0 = enc0.read_delta()
count1 = enc1.read()
delta1 = enc1.read_delta()
```

Hh

LOADING AND TESTING YOUR MBOT CODE

1. UPLOAD THE MBOT LIBRARY

In Thonny's file browser, navigate to the folder you unzipped from GitHub. Locate the mbot folder, right-click on it, and select Upload to / (Figure Ee).

If successful, you'll see the mbot directory appear under the RP2040's file system in Thonny (Figure Ff).

2. TEST THE MOTORS

In Thonny, open the file test_motor.py by double-clicking it. Click the Run button (green play icon). Your MBot's motors should move one at a time, confirming the setup is working properly.

3. USE THE MBOT MICROPYTHON LIBRARY

The MBot MicroPython library provides various functions to control the MBot Basic. Try them out! Below are examples for controlling motors and reading encoder values.

- **Motor Control:** Use the `Motor` class to control the MBot's motors (Figure Gg).
- **Reading Encoders:** Use the `Encoder` class to read values from the MBot's magnetic encoders. These sensor values can be used to track movement and adjust control logic accordingly (Figure Hh).

For more examples, see the extended version of this article at makezine.com/go/michigan-mbot. We'll show how to use these two classes to drive the robot, and explain more cool features of the MBot MicroPython library.

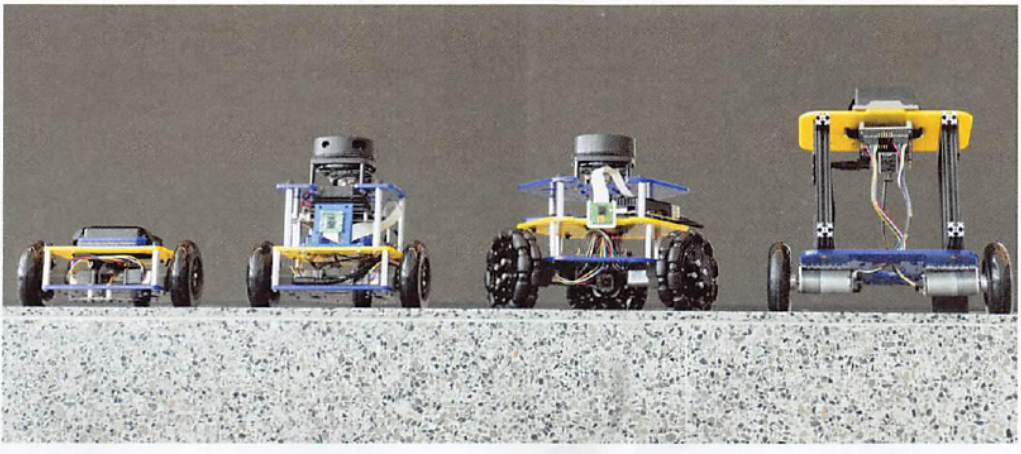
MBOTS IN THE WILD

Now that you can precisely control a mobile robot, a whole world of possibilities opens up. You can use your robot to complete playful tasks like delivering a can of Coke to a friend or surprising a colleague with a balloon down the hall. But that's just the beginning. Here are a few ideas to take your robot further:

- **Remote delivery tasks:** Add a camera and send items across rooms or even buildings.
- **Interactive art projects:** Use the robot as part of a moving sculpture or performance.
- **Autonomous navigation:** Integrate sensors and write code to navigate around obstacles.
- **Swarm behaviors:** Connect multiple robots and experiment with collective behaviors.
- **Voice or app control:** Add Bluetooth or Wi-Fi modules to control the robot with your phone or voice assistant.
- **Sensor-based actions:** Use light, sound, or touch sensors to trigger movement or responses.

Use your imagination — robots like this can be part of games, experiments, installations, or practical tasks. The key is that you now have a reliable base to build upon. Try combining precise movement with other inputs or outputs, and see where it leads!

Learn more about the MBot Ecosystem: mbot.robotics.umich.edu



Dan Newman, MBot Ecosystem

Talking Paper PCBs

Embed electronics in paper circuit boards that communicate directly via ESP-Now

Written and photographed by Lee Wilkins





LEE WILKINS is an artist, cyborg, technologist, and educator based in Montreal, Quebec, a board member of the Open Source Hardware Association, and the author of this column on technology and the body and how they intertwine. Follow them on Instagram @leeborg_

Collaborative work by Lee Wilkins and Sidney Drmay. Thanks to Dinacon and Sea Communities!

In July I spent 4 weeks co-organizing the Digital Naturalism Conference (Dinacon). Every few years we bring together about 80 artists, designers, scientists, and technologists to unique places around the world to explore biodiversity and human-computer interaction. One of the unique restrictions of being in semi-remote places is very limited access to resources and infrastructure. For example, unpredictable Wi-Fi or scarce supplies can shape what one might create.

This year, I worked with Sid Drmay to use the natural environment in Les Village in the north of Bali, Indonesia, to make Paper PCBs: electronics embedded in paper made from local plants. We wanted to experiment with networked devices that didn't rely on the internet, and were easy to implement and cheap. I decided to embed some Espressif ESP32 modules in the paper Sid was making out of pandan leaves and local flowers. We used the paper as a substrate — like the fiberglass of a PCB — to embed wires, contacts, and all kinds of electronics in a single sheet of paper! The goal was to have a modular set of organic, PCB-like objects that could communicate in any environment.

The great thing about using Espressif's ESP32 microcontrollers is their **ESP-Now** protocol that lets them talk directly to each other, which means we can bring this project anywhere — from the ocean to the jungle — without needing to change any network settings or infrastructure. We also decided to power these paper PCBs using solar panels, which we embedded in paper alongside the ESP32 modules.

ESP-NOW

For the last few years, everyone has been obsessed with ESP32s. As someone who has been working with electronics for a long time, sometimes it's hard for me to accept that new things are better, but I'm really into the ESP32 lately. Not only is it small, cheap, and powerful, it's got a built-in wireless protocol, ESP-Now, that lets you talk to other ESP32 modules with no additional hardware or cost. The protocol is



developed by Espressif and enables low-power control without the use of a router. It is also very fast and can send up to 250 bytes per message between ESP32 or ESP8266 boards. ESP-Now supports two-way communication, either one-to-one, or one-to-many devices. It's great for remote sensors, IoT projects, and private networks.

In the C code for each ESP device, you'll find a ***broadcast address*** (where you're sending data), a piece of data to send (a ***variable*** or a ***data structure***, which we'll talk about below), and a few ***callback functions*** for on-send and on-receiving data. I really liked this tutorial from DroneBot (dronebotworkshop.com/esp-now) but I've noticed that almost all ESP-Now tutorials use very similar base code that has a few tricks that aren't super intuitive for beginners. So I'm going to break it down and explain them.

ESP-Now uses ordinary MAC addresses to send and receive information, similar to Wi-Fi, Ethernet, and Bluetooth. So your first step is to plug in each of your devices and figure out its unique MAC address. You'll need to import the *WiFi.h* library to be able to do this:

```
#include "WiFi.h"
void setup(){
  Serial.begin(115200);
  WiFi.mode(WIFI_MODE_STA);
  Serial.println(WiFi.macAddress());
  }
```

The address will print in your serial output, so write it down. I usually tape the address physically to each ESP32. It's a series of six two-digit hexadecimal numbers, so it will usually look something like this: **00-1A-2B-3C-4D-5E**.

Once you've determined this for each device, you'll be able to send to that particular address. You can establish this communication by first making an array to hold the parts of the address:

```
uint8_t broadcastAddress[] = {0x94,
0xB5, 0x55, 0x26, 0x27, 0x34};
```

If you're new to microcontroller programming, you may be more familiar with data types like **int**, **float**, **char**, or **String**. One of the reasons ESP-Now is so efficient is that it is using a lot of optimizations that make things run smoothly. I'm going to go over a couple of things you may see in ESP-Now examples that can be confusing to Arduino programmers. One example is this variable I've seen in many tutorials that uses the data type **uint8_t** to make an array. This basically means that we are using the smallest possible size of integer to store the parts of the broadcast address.

Another great thing about ESP-Now is that it can send a range of different types of data. A common example you'll see is the use of a ***data structure*** or **struct** to send multiple pieces of data in a single structured chunk. This is similar to what you might see in JavaScript as an **Object**, but it's a bit more basic:

```
typedef struct struct_message {
  char a[32];
  int b;
  float c;
  bool d;
} struct_message;
```

Structs are basically a way of defining your own data type. In this example we're making a new data type called **struct_message**, but you could easily call it whatever you want. Inside **struct_message** we have several other pieces of data you can access by using the dot operator (.); for example, if we make a new instance of **struct_message** called **my_data**, then I can access pieces of it by using **my_data.a** or **my_data.b**. It's pretty useful, because here we can send a lot of different data points by only sending one variable. You could have many instances of **struct_message**, each with their own name and data. Pretty cool!

Next, we'll need to register ***callback functions***. These are very useful because they happen outside of the flow of a normal program. For example, we could have a function that checks if data is received at the end of every loop, but if our loop happens to be very long we might miss something. A callback function is triggered as soon the data is received, regardless of where we are in the loop. For ESP-Now, we need to register the callback using the following code for send and receive. You can register any function name you want, but these are common. This is done in the **setup** of your program. In this example, I am telling ESP-Now that the functions called **OnDataSent** and **OnDataRecv** should be called when that action occurs, no matter what else is happening.

```
esp_now_register_send_cb(OnDataSent);
esp_now_register_recv_cb(OnDataRecv);
```

Next, I'll register the address I want to communicate with. You can add multiple ***peers*** using this method too. Many ESP-Now examples include a few checks to verify whether the peer has been added, using an **if** statement with the returned value of the **esp_now_add_peer** function. By doing this, we can make sure that the peer has been successfully added and get ahead of any errors. Using this type of check allows us to pinpoint exactly where something might have gone wrong in our program.

```
memcpy(peerInfo.peer_addr,
broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
   // Add peer
  if (esp_now_add_peer(&peerInfo) !=
ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
}
```

Now, to actually send the data we are using the **esp_now_send** function — another line I didn't see explained very well in many tutorials. The function takes three parameters: an address, a pointer to the data location, and the size of the data. We might be used to passing the variable **myData** directly to the function, but that's not exactly how this function works! We've already registered the first parameter with **esp_now_add_peer**, so we can pass the **broadcastAddress** variable.

The next parameter is a ***dereferenced pointer***. To keep it simple, a pointer indicates a location in memory, and is provided by the address operator **&** in C. The **(uint8_t*)** indicates that we are casting the pointer to data bytes, meaning instead of providing a pointer to our **struct_message** type, we're making it look like a pointer to the start of an array of bytes. You can think of this like an address on a street, rather than the contents of a building.

Finally, our last parameter is **sizeof(myData)**, which indicates how big the data is. This is measured in bytes, and **uint8_t** is equal to one byte. This is telling our program to look for information in bytes rather than a **struct** so that this function can take data in many shapes and sizes. This function works by finding the location of the data in memory, then telling us how big of a chunk of data we want to send.

```
esp_err_t result = esp_now_
send(broadcastAddress, (uint8_t *)
&myData, sizeof(myData));
```

Esp_now_send returns an object with the data type **esp_err_t**, which indicates if the data was sent successfully or not. Again, we are checking if this happened successfully with an **if** statement in order to handle errors. Maybe your device is offline, or your address is incorrect.

```
if (result == ESP_OK) {
    Serial.println("Sending confirmed");
  }
  else {
    Serial.println("Sending error");
  }
```

On the receiving device, we have a matching set of data to decode what we've been sent. Our receiving function is passed the MAC address, a pointer to the incoming data, and its length. We then use **memcpy()** to put the data we received into the matching **myData** variable on the receiver. Ta-da! Our data has been transferred.

```
void OnDataRecv(const uint8_t * mac,
const uint8_t *incomingData, int len) {
 memcpy(&myData, incomingData,
sizeof(myData));
  ...
```

If you want to check whether the correct amount of data is received, you can check that the incoming data size is the same as the size of the struct **myData**:

```
if (len != sizeof(myData)) {
    Serial.println("Oops this is isn't
the right amount of data!");
}
```

You can also set your ESP32 to broadcast, which is really useful for a situation where you may have many receivers that you might not know. In this instance we are going to send out a special MAC address **FF:FF:FF:FF:FF:FF** and register all responders as peers using **esp_now_add_peer**. That way, any ESP32 devices with ESP-Now within range will be added to the broadcast.

```
uint8_t broadcastAddress[] = {0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
 esp_now_peer_info_t peerInfo = {};
 memcpy(peerInfo.peer_addr,
broadcastAddress, 6);
 if (!esp_now_is_peer_
exist(broadcastAddress))
 {
    esp_now_add_peer(&peerInfo);
 }
```

ESP-Now is very powerful, and super simple to use. I chose it because it didn't require any additional infrastructure and could be used in our mobile explorations.

If this all seems a bit overwhelming, you can also use **CircuitPython** to control ESP-Now. CircuitPython helps us move past some of this more complex C code and focus more on the direct interaction. CircuitPython can be slower, but it's certainly easier to use. I really liked the tutorial from Adafruit at learn.adafruit.com/esp-now-in-circuitpython.



MAKE YOUR PAPER PCBS

TIME REQUIRED: A Weekend
DIFFICULTY: Moderate
COST: $25–$50

MATERIALS

FOR PAPERMAKING:

- **Leaves from your environment** We used pandan leaves.
- **Soda ash**
- **Recycled paper**
- **Felt or cardboard**
- **Blender**
- **Pot or saucepan**
- **Papermaking mold and deckle** or sieve or other mesh

FOR THE ELECTRONICS:

- **ESP32 microcontroller module** I used the Xiao ESP32-C3.
- **Masking tape**
- **Copper tape**
- **Enamel wire** aka magnet wire
- **Electronic components** to embed in the paper; I used solar panels, LED "noodles," and buttons.
- **Soldering iron and solder**

The next step was embedding the electronics into paper! Sid and I embedded our ESP32s and other low-voltage components into paper made from the local environment. We wanted to recreate PCBs by using natural materials, so we produced some solar-powered ESP32 modules using conductive tape and pandan leaves.

1. FORAGE FOR YOUR MATERIALS

You can make paper out of almost anything fibrous in your environment that you can break down into pulp, including irises, day lilies, grasses and nettles, and more. We chose to use pandan leaves, because of the brilliant green color they created. So many baked goods in Indonesia had a range of green tones, all derived from pandan leaves! Our hope was to make a small nod to the green of traditional PCBs too. Sid also collected some banana leaves and bark because they were plentiful (Figure A). Just gather as much as you can, and be careful of spiders!

2. BOIL AND PULP

Once you have all the materials you want to turn into paper, you'll need to soak and boil them (Figure B). Sid boiled the leaves for about 4½ hours, then put them in the blender to make sure they were pulping correctly. You may want to mix in about half a cup of soda ash in order to help break down the fibers (Figure C).

Sid also mixed in a bit of regular shredded paper to help keep it together, but this may not be necessary with a longer boil time or a stronger blender that can cut through the fibers easier.

3. PREPARE ELECTRONICS

While Sid was pulping, I prepared the electronics. I wanted everything to be embedded in the paper except for contacts that would need to connect to other elements. For example, I pre-soldered the solar panel circuit and left exposed wires to connect to the rest of the circuit. I also soldered wires to motors, ESP32 pins, and buttons so that they could be easily embedded in the paper. I made sure these wires reached well beyond the edges of the paper, so I could trim them later. I used magnet wire, which is super thin, so that the traces would sit inside the paper easily, but you could use any wire!

When preparing the ESP32s I took care to put masking tape over the USB port so that no pulp got in the contacts. This is probably a good idea for any connector that's not a pin pad, such as a JST connector, battery, or anything you might need to access.

4. PULL PAPER, ARRANGE ELECTRONICS

In order to pull paper, the pulp mixture should be put into water at a 2:1 water to pulp ratio, in a bucket or Tupperware. This makes a watery soup that can be scooped up with the mold (Figures D and E).

Next we arranged the electronics on top (Figures F and G on the following page). It took manually dripping some pulp onto the electronics to effectively embed them. In order for the electronics to be actually inside the paper, they need to be at least partially covered. It's a bit of a fine balance and depends on the individual components, for example the solar panels wouldn't work if they were fully covered by pulp!


A


B

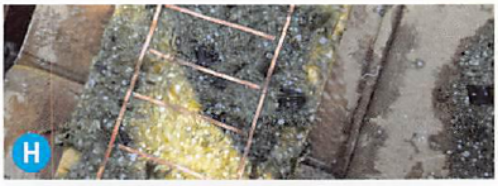

C


D


E









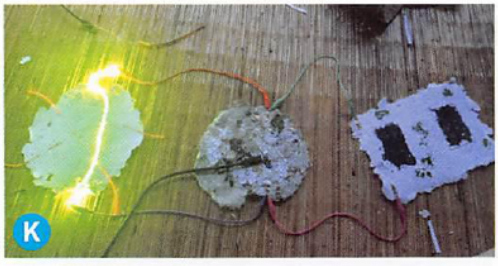


Make sure to keep track of any wires you want to connect to other things (Figure H). It can help to label them, as the connections won't be visible once they go into the paper!

Once we were happy with the arrangement, Sid couched the mixture on a piece of felt by flipping the deckle over so that the paper could dry (Figure I).

5. SOLDER CONTACTS

I soldered each wire to a piece of conductive tape at the edge of the paper. This let me use alligator clips to connect the paper PCBs together and make a range of circuits (Figures J and K).



6. CODE!

Once the paper is dry, you're ready to code! The two solar panel circuits I made were able to each power an ESP32 (Figure L), so I set up two-way communication between them.

Try out my ESP-Now code (you can download it at github.com/LeeCyborg/ESP32_talking_paper or scan this QR code), or make a paper PCB of your own imagination! I'd love to see what you produce.



INNOVATION IS WITHIN YOUR GRASP!





LEARN MORE

make.co/innovate

Grab your Oxocard Connect Innovator Kit and **FREE** *Getting Started with Oxocard Connect* book Today!

Make: OXOCARD

Adventures in Object Photography

My low-budget solutions for controlling highlights, shadows, and background

Written and photographed by Charles Platt



CHARLES PLATT is the author of the bestselling *Make: Electronics*, the *Encyclopedia of Electronic Components*, and other great books. He also wrote "Homage to the Axe" on page 36 of this issue. makershed.com/platt

I like to photograph the things I build, and I think many makers feel a similar impulse to keep a visual record. The challenge is to make a picture look reasonably professional.

Photographing objects is certainly a lot easier than photographing people, but even an object that just sits there and waits for me will entail three challenges: ***highlights***, ***shadows***, and ***background***. Here I'll describe my ongoing struggle to keep these variables under control.

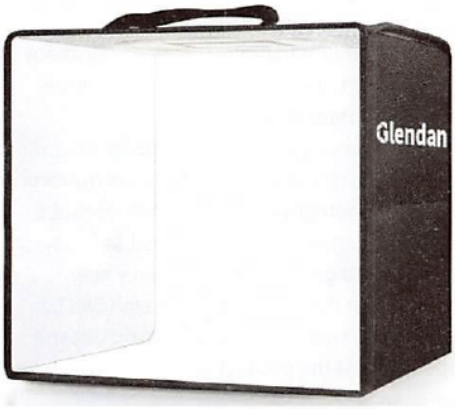
THE UNSATISFACTORY TENT

I'm fussy about lighting. I want a soft gradation of tones from bright to dark, but I also want highlights that are clearly defined.

As for shadows, I like to eliminate them completely. Many people don't seem to mind if a photograph shows an object sitting on a flat surface and casting a shadow, but I want the object to be isolated, as if it is floating in space, with not a shadow in sight.

As for the background, it should be uniform and undistracting.

You might think a photographic ***light tent*** would satisfy these requirements, and tents such as the one in Figure A seem to be popular for photographing objects to sell on eBay. LED lighting is arrayed inside the top of the tent, and is supposed to eliminate shadows — but in my experience, it doesn't quite succeed. Also, you can't make significant adjustments to the lighting, and you certainly can't tweak the highlights. Add it all up, and for me, tents are not acceptable.



A Often described as a *light tent* and sometimes as a *light box*, this type of enclosure has limitations.



B A soft box with its lights switched on.



C The soft box switched off.



D LED lights inside the soft box.

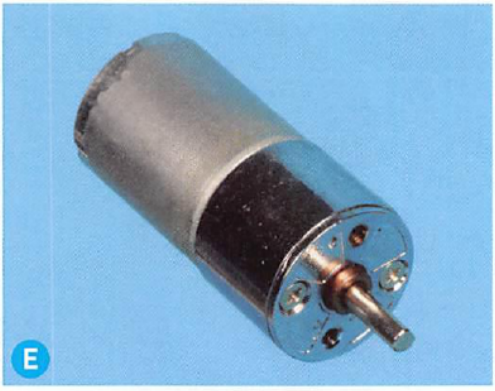
SOFT BOXES

Since an off-the-shelf item doesn't work for me, I had to go back to basics — and the most basic piece of equipment for a photographer is a light source. Fortunately, this part is relatively easy.

A diffuse source is essential to create the smooth gradation in tone that I like to see. I had dabbled previously in portrait photography, so I already owned a pair of ***soft boxes*** such as the one in Figure B. This example is much larger than necessary for pictures of small objects, but still, it works. If you have any interest in indoor photography, you may already own soft boxes. They are not very expensive.

Figure C shows the soft box switched off, and you can see that the front cover is translucent, to diffuse the light. In Figure D the front panel is removed to reveal four LED light bulbs inside. (I don't use electronic flash, because it's too expensive.)

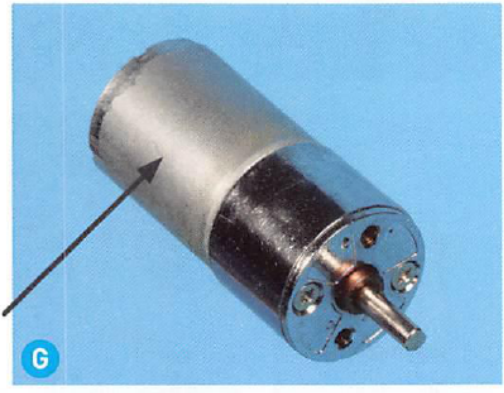
I think you can never have too much light,



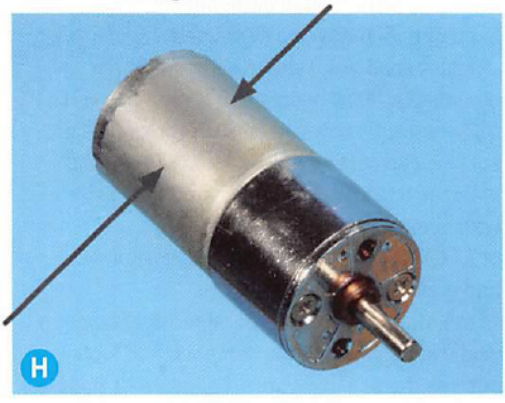
A small 6VDC motor illuminated by soft boxes only.



A mini LED video light. It stands about 6" tall.



A highlight adds to the tonal range and makes the object look more three-dimensional.



Are two highlights better than one? It's a matter of taste.

especially if you want to use a small aperture to get depth of field. Therefore, I bought the most powerful ***daylight-spectrum LED bulbs*** that I could find. Each consumes 60W, creating a light output that would require more than 300W with old-school incandescent lamps. Put four of the LED bulbs together, and you have almost 1,500 watts of incandescent equivalent. Use two soft boxes, and you're close to 3,000 watts equivalent. Just the thing!

The LED bulbs get hot. In fact, they have little internal cooling fans. Soft boxes are not really intended for that much wattage, so I cut holes in the top and bottom of each box to allow ventilation.

So much for the easy part.

HIGHLIGHTS

Figure E shows a picture of a little 6VDC motor, lit by soft boxes. The gentle gradation of tone looks nice, but is a bit flat, which is why I want the highlights that I've been talking about.

Recently I discovered ***miniature LED video lights*** of the type shown in Figure F. Each has its own little tripod, with legs about 4" long, and there are no annoying wires, because each LED cube contains a lithium battery. This provides about 1 hour of light, depending which of the three options for intensity you choose.

I found these lights at B&H Photo (bhphotovideo.com/c/product/1877061-REG). This is my favorite source for photographic equipment, because they have economical items as well as crazy-expensive pro gear, and they actually seem to know what they're selling. Amazon can be cheaper, but I get the feeling Amazon doesn't inspect or test the products it sells, so you take your chances.

Figure G shows the same motor as before, but with a highlight added. Then I decided maybe

I would like two highlights, so I added one more, as in Figure H. Was that an improvement? You be the judge.

In Figure I, you see the setup I used. Two of the LED lights have been elevated by inserting them in some white-painted steel tube mounted in wooden blocks. I happened to have the tube lying around, but electrical conduit or PVC pipe would work just as well.

In each picture, the background is ***photographic paper*** that you can buy in a wide variety of sizes online.

Now for the difficult part.

ELIMINATING SHADOWS

One of the great advantages of diffuse lighting is that if a small object is, say, 10" away from a flat surface, it doesn't cast a visible shadow. So, I needed a way to suspend an object with some space around it.

A friend of mine used to be a studio photographer in London for Sotheby's, the auction house, where he took pictures of antique jewelry that had to look perfect in every way. His trick was to place the object on a glass plate, with a gray background (out of focus) below it. But when I tried this, I often had trouble avoiding reflections in the glass.

So, I tried hanging things on nylon fishing line. The problem with this arrangement was that the object will twist and sway in response to the tiniest air currents, and any motion is unacceptable if you're using an exposure longer than 1/100 second.

This is why I ended up using the rather primitive setup in Figure I, in which the motor is sitting on some poster putty, at the top of a piece of stiff wire embedded in a wooden block. Of course I had to hide the wire in the finished photograph, but this was easily done in Adobe Photoshop.

Incidentally, I prefer not to pay a monthly fee for software, so I am in prehistoric mode here, using Photoshop CS2. An alternative would be an imitation such as PhotoPad from NCH Software, which has a free trial period. If you decide to continue using it, you'll find it's quite affordable, with no monthly fee.



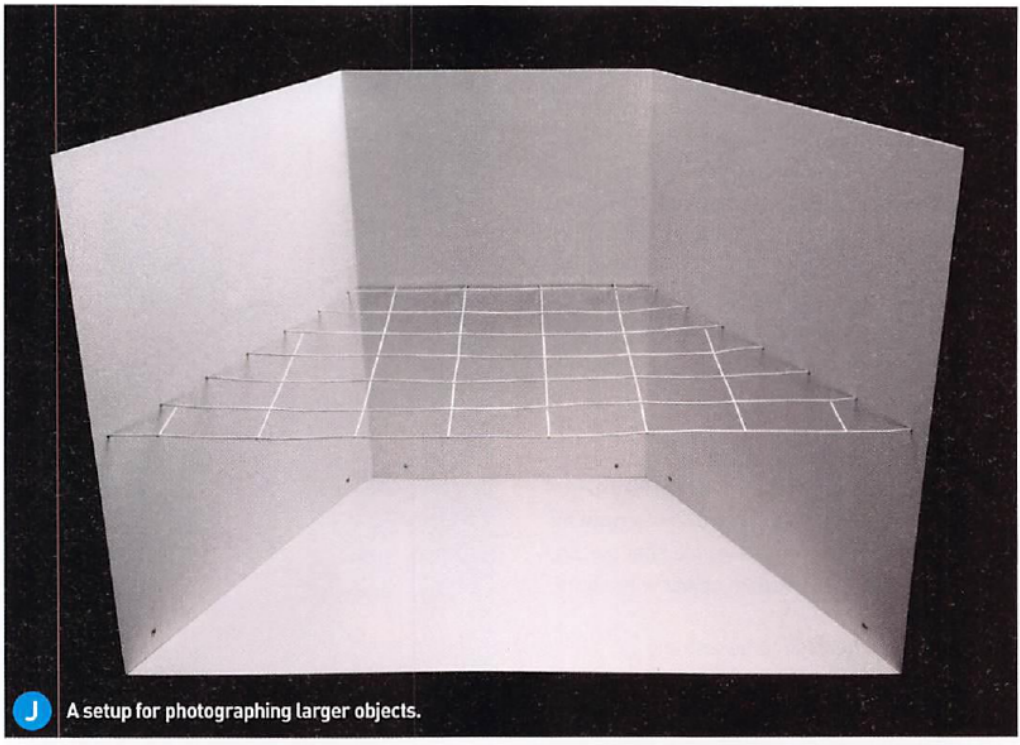
How the photographs in figures E, G, and H were taken. Primary illumination is from soft boxes, behind the camera.

TRANSPARENCY

Elevating an object does solve the problem of shadows, and photographic paper is a simple way to get a plain background, but there is another consideration. While a solid color may be acceptable in many situations, Juliann Brown, the creative director of *Make:* magazine, really likes to have no background at all. In other words, I should knock out the background color to make it transparent. This gives Juliann more creative freedom, and since I admire her work, I want to enable as much creativity as possible.

In the world of movies, a green screen gets rid of a background. The shade of green is intense and distinctive, making it easy to select in post-production. After it is selected, it is erased, and any other background — such as an alien landscape, or an abandoned city — can be inserted instead. For my purposes, I might like to insert a graph-paper background to show the size of the object in front of it.

Unfortunately, a green screen doesn't work so well on a small scale, because it's too close



J A setup for photographing larger objects.

to the object. Light bounces off the background, and the object picks up a sickly green tint around the edges. (You can see the same effect in the motor in Figures E, G, and H, where the body of it reflects the bright blue background. If I knocked out that background, the blue tint on the motor would be very obvious to the eye.)

To eliminate this problem, a white background is needed. My camera will want to "balance" the tonal range by making the white background gray, so I have to adjust the exposure to compensate. Now I am likely to have another problem: White is a more common color than intense green, so Photoshop's Magic Wand tool may select some pale areas at the edges of the object in addition to the background. A bit of manual touch-up will be required to take care of that. (I realize Adobe has developed clever algorithms to select outlines, but as previously noted, I have the limitations of Photoshop CS2.)

So far, so good, but there is yet another problem. The motor I photographed in Figure E is about the largest, heaviest object that will stick reliably to a dab of poster putty. Anything larger will gradually tilt and fall off. So — how can I photograph a big, heavy, complicated item? This was of interest to me recently, as I had to take a picture of a soldering station, which was not only large and heavy, but was attached to a soldering tool on a length of cable. There was no way I could imagine suspending these items with some space around them.

Or was there?

THE GRID SYSTEM

I came up with the idea of using a horizontal wire grid surrounded with background panels of white ABS plastic, as in Figure J. (This picture has been underexposed and lit from above, to make the grid easier to see. In reality, everything is equally bright white.)

You might expect the grid to cast a shadow, but as you can see, it doesn't. This is because light is coming from a very wide area, while the wire is thin.

Figure K shows a photograph taken with this device, after I knocked out the background — which was quite easily done, because it was almost uniformly bright.

I fabricated the background panels from

a sheet of ⅛" white ABS plastic measuring 24"×48", although a more economical option would be to use white-painted masonite, or foam board from an art supply store. To attach the background to the floor of the assembly, I used 1"×2" strips of hardwood behind it, as in Figure L.

I have a plastic bender, so I made a couple of 45-degree bends in the ABS. If you don't have a bender, you can try using a heat gun, or just saw the background into three pieces, and after you take a photograph, paint over the joins using image-editing software.

For the wire grid, I used a piece of field fence that I happened to have, shown in Figure M. (I live in a rural area, where we need to fence out cows and stray dogs.) After I trimmed and spray-painted it, I pushed the ends through holes in the panels. An alternative is chicken wire, which Amazon sells cheaply in short pieces.

What if you still want sharp highlights? Using another piece of ABS plastic, I fabricated the hanger in Figure N, which clips onto the background to hold my mini LED lights, as in Figure O.

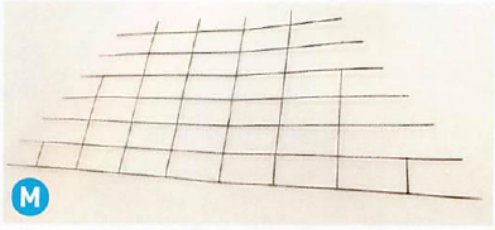
Now, instead of trying to balance an object on some poster putty, or suspending it and trying to prevent it from swinging around, I just lay the object on the grid.

IMPROVISATION

At this point, I feel reasonably happy with my answers to the problems of highlights, shadows, and background, yet I suspect that these problems must have been solved by professional photographers long ago, and their methods probably look — well, more professional. On the other hand, I also suspect that their solutions are expensive, because everything in professional photography seems to be expensive.

In any case, part of the fun of being a maker is to improvise. So, as someone who never gets tired of doing things in his own low-cost, low-tech way, these are my object-photography improvisations so far.







A hanger to hold the tripod of a mini LED light.



A mini LED light, upside down, with its tripod legs pushed into the hanger.

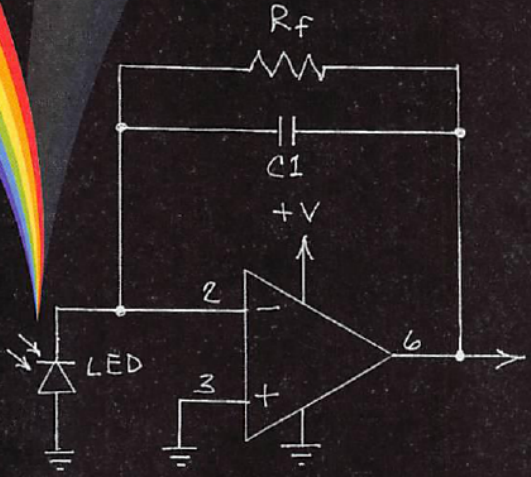
Semiconductor Light Sensors

Learn the basics and make a transimpedance amplifier to boost light signals

Written and photographed by Forrest M. Mims III



FORREST M. MIMS III (forrestmims.org) is an amateur scientist and Rolex Award winner whose books have sold more than 7 million copies, including *Maverick Scientist* and Forrest Mims' *Science Experiments*, both available at makershed.com.



When I first began designing simple analog computers as a high school senior in 1962, very few light-sensitive electronic components were available. One was a miniature solar cell — technically a ***photodiode*** — that I used to detect the position of a meter needle in my homemade language translating analog computer. The Smithsonian Institution describes that computer, which I donated 35 years ago, as an early example of a hobbyist computer.

By 1966, the desktop in my Texas A&M dorm room included a dozen surplus silicon solar cells left over from government satellite projects; also present were several ***phototransistors*** and cadmium sulfide (CdS) ***photoresistors***. Semiconductor light sensors were making progress! Today my electronics shop includes hundreds of semiconductor light sensors.

Light-sensitive semiconductors either produce an electrical current or alter their resistance when illuminated by light. Depending on their composition, the wavelength response of light-sensitive semiconductors can range from a few tens of nanometers to the entire visible spectrum. For example, most silicon photodiodes (as in Figure A) have a wide spectral response of some 300nm, from the violet to the near infrared.

Many semiconductor light detectors can function entirely on their own. For example, connect a silicon solar cell to an earphone. Then point a TV remote control at the solar cell and press the button. You will likely hear a tone from the earphone corresponding to the pulsating near-infrared emitted by the controller.

During my senior year at Texas A&M, I substituted a CdS photocell for the resistor that controls the tone of a simple two-transistor oscillator. I then asked a freshman to stand at one end of the dorm hallway and point the circuit toward me at the opposite end of the hall more than 250 feet away. When we switched off the lights, the oscillator was silent. When I struck a match, the CdS cell's resistance changed enough for the formerly silent tone generator to emit clicks that everyone could hear. Those clicks became a hum when a flashlight was pointed toward the circuit.

While this ultra-simple experiment demonstrated the extremely sensitive nature of CdS photocells, adding a single IC makes them much more versatile. You can easily build such circuits, but the hobby electronics market offers various light-sensitive, miniature solid-state circuits that can be purchased for as little as a few dollars. Consider Figure B on the following page, a tiny CdS photoresistor light sensor board smaller than a pair of postage stamps that features an LM393 op-amp, two LED status indicators, and six SMD components. It even includes a sensitivity control, and it can be connected to an external analog or digital circuit. One Chinese source sells this tiny board for only $0.79. It is also available with a photodiode light sensor instead of the photoresistor.

TIME REQUIRED: 1–2 Hours

DIFFICULTY: Easy

COST: $5–$10

MATERIALS

- **Op-amp IC chip** such as OPA1655
- **LED** You'll use it in reverse as a light-sensing photodiode, not as a light emitter.
- **Resistors, various** (see text)
- **Capacitor** (see text)
- **Perf board**

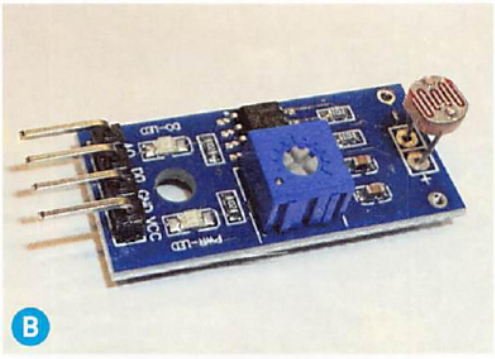
TOOLS

- **Soldering iron and solder**



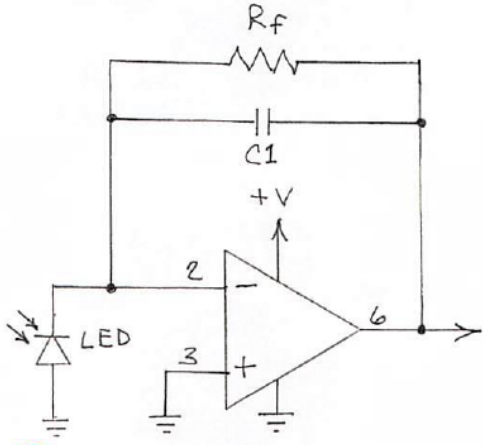
A

These photodiodes are essentially miniature solar cells.



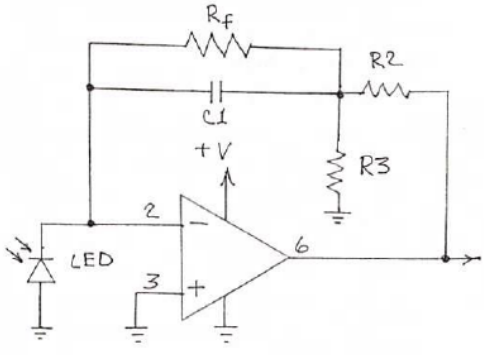
B

This miniature circuit amplifies the signal generated by light received by a CdS photoresistor.



C

This ultrasensitive transimpedance amplifier (TIA) transforms the current generated by a photodiode or LED into an output voltage having a gain equal to the resistance of R_f.



D

Adding two resistors to the circuit in Figure C forms a T-network amplifier that greatly reduces the resistance of R_f necessary to achieve very high gain.

DO-IT-YOURSELF PHOTODIODE AMPLIFIERS

The circuit in the sensor board shown in Figure B includes an op-amp that significantly improves the circuit's sensitivity. The op-amp converts the change in resistance of the photoresistor into a voltage. Op-amp circuits that amplify the photocurrent generated by a photoresistor or photodiode are called ***transimpedance amplifiers (TIAs)***.

Figure C shows a basic TIA that amplifies the photocurrent from a photodiode. Resistor R_f controls the circuit's amplification factor or ***gain***, which in this basic circuit is the resistance of R_f. Therefore, when R1 is 10,000 ohms, the circuit amplifies the photocurrent by 10,000. The amplification can be changed simply by altering the resistance of R1.

If you plan to transmit audio over a light beam, many op-amps will work fine in the receiver. I have had excellent results with the OPA1655 for very high-quality audio applications, for which this chip was designed. This op-amp sells for less than $2 and provides excellent quality audio. The OPA1656 is a dual version of this superb amplifier chip. These chips are only available as SMD components, but you can also get good results with many different through-hole ICs.

For very weak signals, the resistance of the feedback resistor in an op-amp amplifier circuit may need to be millions of ohms. Figure D shows a modified TIA amplifier that uses a ***T-network*** of two resistors to significantly amplify a signal without resorting to a very high resistance for R_f in Figure C.

You can learn how to select values for the components in a TIA amplifier with a T-network from an excellent Texas Instruments publication at ti.com/lit/an/sboa284/sboa284.pdf. Another good source is "High-Gain Transimpedance Amplifier (TIA) for Night Airglow Photometer" by P. T. Patil and colleagues at the Indian Institute of Geomagnetism, at researchgate.net/publication/237580062.

LEDS AS PHOTODIODES

As I discovered and published in 1972, most LEDs can detect light slightly below their emission wavelength. For example, the LEDs used in

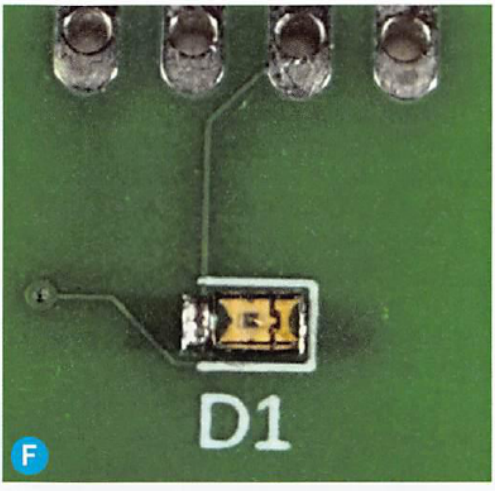
automobile taillights emit red light peaking around 650nm and detect light with a peak around 620nm. Most near-infrared LEDs used in remote controls emit a peak wavelength of 880nm and detect at 820nm.

While experimenting with homemade travel aids for the blind in 1971, I learned that LEDs that emit near infrared at 950nm can also detect infrared emitted by a similar LED. I wrote about this in one of the optoelectronics books I wrote back then (*Light Emitting Diodes*, Howard W. Sams & Co., 1972). I have since designed many projects in which LEDs serve as wavelength-dependent photodiodes. For example, Figure E shows a 1150nm LED mounted on an SMA connector. This was one of the LEDs evaluated for use as a highly sensitive, wavelength- dependent photodiode in a NASA assignment to develop twilight photometers to measure the height of volcanic aerosols in the stratosphere.

Conventional silicon photodiodes respond to light from the blue to the near-infrared. LEDs used as photodiodes respond to a 50–100nm band of light that usually peaks around 60 to 80nm below the LED's peak emission. Thus, a typical red LED with a peak emission wavelength of around 670nm detects light with a peak response near 620nm and a half-power width of around 40nm. The peak spectral response of the LED in Figure E is 970nm.



While the component connected to an SMA connector resembles a standard photodiode, it is actually an LED used as a narrow-wavelength photodiode.



This tiny SMD photodiode is soldered to the backside of a circuit board on which a TIA amplifier circuit is assembled.

SURFACE MOUNT PHOTODIODES

Surface-mount technology has greatly expanded our photodiode packaging options. For example, I am building a miniature light sensor for a new twilight project. In the past, I would have used a standard epoxy-encapsulated photodiode or LED. Instead, I am using a tiny SMD photodiode carefully soldered to the back of the amplifier board in Figure F, designed by my friend Dr. Don Wilcher.

Soldering a tiny 8305 SMD part requires both patience and experience. The most important aspect of soldering the photodiode in Figure F was making sure the anode and cathode sides of the photodiode were properly aligned. This required a 10X hand lens or the head-mounted lens and light source I use for SMD projects.

Soldering SMD parts requires practice and a steady hand. In this case, I first applied a small amount of solder over both SMD pads. I then applied a droplet of solder paste over both pads. I then used tweezers to carefully place the photodiode over the two pads.

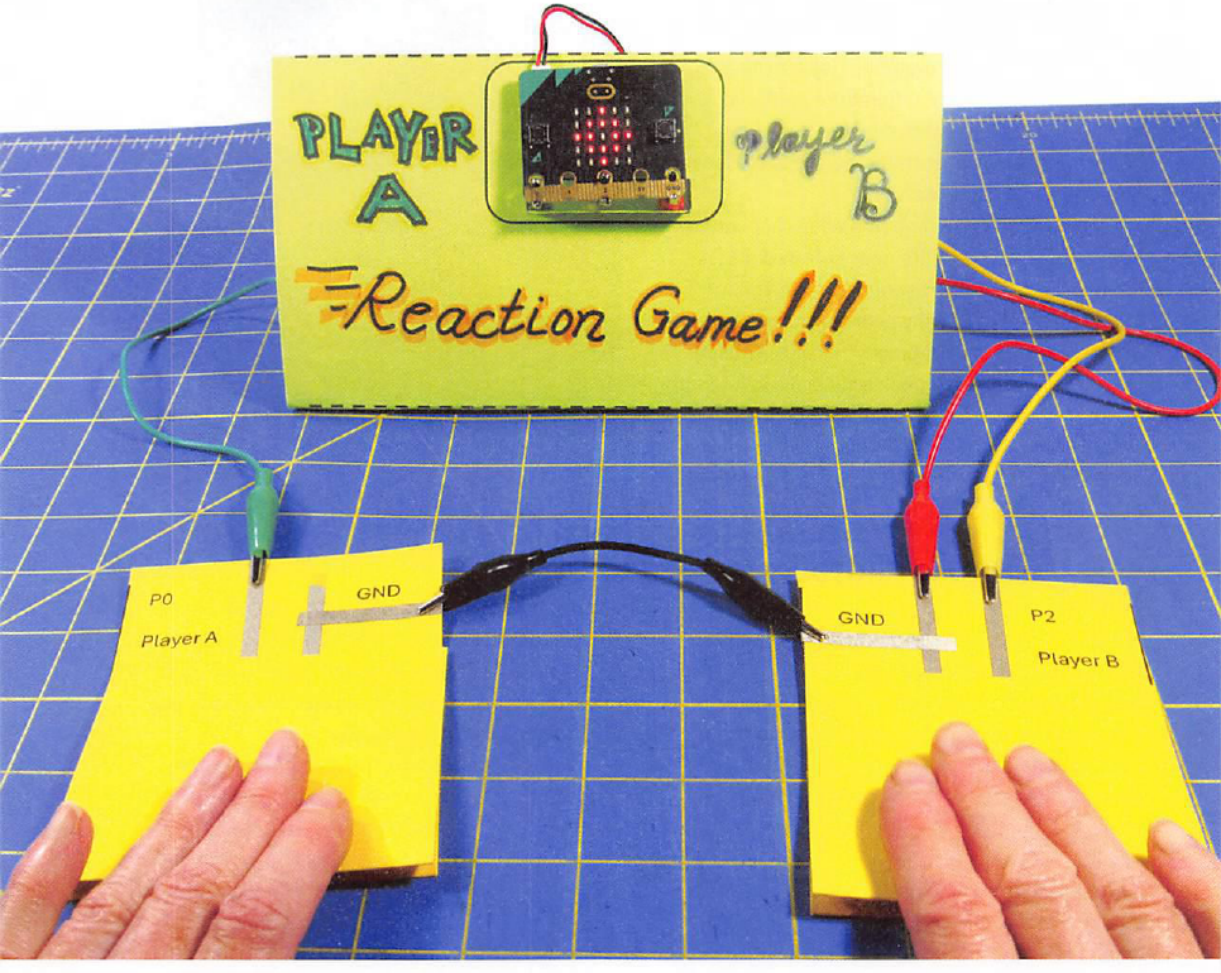
After double checking the orientation of the photodiode, I carefully held the photodiode in place with the tweezers and then touched the clean, sharp tip of a soldering iron to where one end of the photodiode touched the solder atop the adjacent pad. As soon as the solder melted, I removed the iron and repeated this for the opposite side of the photodiode. After both ends were soldered I added a bit of solder to each connection.

It's tricky, but worth it when you need to fit light sensing into very small spaces!

Reaction Race

Make a classic reaction game with micro:bit and cardboard

Written and photographed by Kathy Ceceri





KATHY CECERI is the author of over a dozen books for kids and other beginners featuring hands-on STEAM activities, including *Making Simple Robots, 2nd Edition* and *BOTS!*. She teaches enrichment classes for students and workshops for parents and educators at schools, museums, libraries, and Maker Faires around her home in upstate New York and across the Northeast. Check out her work at kathyceceri.com.

A lot has changed in the 10 years since I wrote my book *Make: Paper Inventions*, a guide to learning about STEAM (science, technology, engineering, art, and math) through paper projects — especially light-up, motorized, and programmable paper circuits! I've been itching to update the original book with new materials like fabric conductive tape, and beginner-friendly microcontrollers lightweight enough for paper models. At the same time, I wanted to make the book accessible to young makers who may be more comfortable playing with screens than with paper crafts.

That's why *Paper Inventions, 2nd Edition*, contains tons of new and updated projects, templates that are easier to cut out by hand, models you can put together quickly, and standard shapes and mechanisms you can use in multiple projects. You'll still learn the molecular science that makes paper so strong and flexible it's used to engineer structures taller than a fifth-grader. You'll still find instructions for making spinning entertainment machines, and wearables, toys, and knickknacks to keep or share with friends. But now you'll also get an intro to basic coding concepts that let you create projects like a motorized paper plate tilt ball maze and a light-up, pop-up card that plays "Happy Birthday" when you blow out the LED candles.

For a taste of what the new edition is like, here's some "bonus content" that didn't fit in the book: a programmable Reaction Game powered by a BBC micro:bit. It's fun to play and it works great with my middle-school-age students. A Reaction Game tests how fast players can hit a button when a signal is given, and there are a zillion variations. I'll show you how to build the button and display panel, then walk you through programming the simplest playable version with Microsoft MakeCode, and adding upgrades to make the game more challenging and more fun.

HOW THE BUTTONS WORK

The players' buttons in the Reaction Game are basically external on/off switches, with conductive tape serving as part of the wiring for the circuit. When the button is open, no electricity can flow through the circuit because of the gap between

TIME REQUIRED: 1–2 Hours

DIFFICULTY: Easy

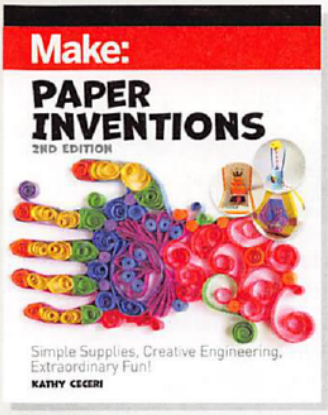
COST: $25–$35

MATERIALS

- **Micro:bit board with USB to micro B cable** available in the Go Pack starter kit along with battery holder and batteries, which are not required for this project
- **Conductive fabric or copper tape**
- **Alligator clip wires (4)**
- **Cardstock, 8½"×11" (2 sheets)** or similar, with the template printed or copied onto them
- **Tape** You can use clear, masking, and/or duct tape.

TOOLS

- **Computer with printer**
- **Scissors**

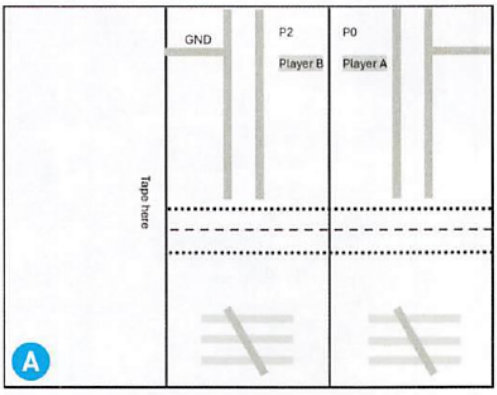


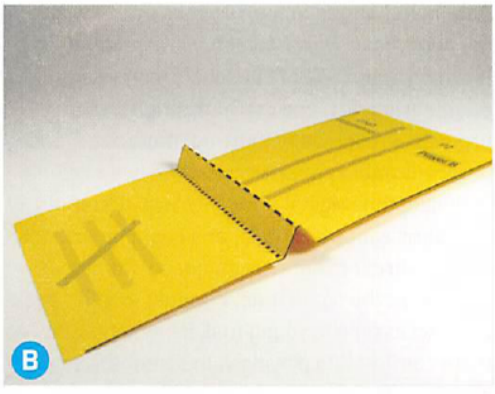
NEWLY REVISED AND UPDATED, *Make: Paper Inventions,* **2nd Edition is available now at Maker Shed (makershed.com) and other booksellers.**

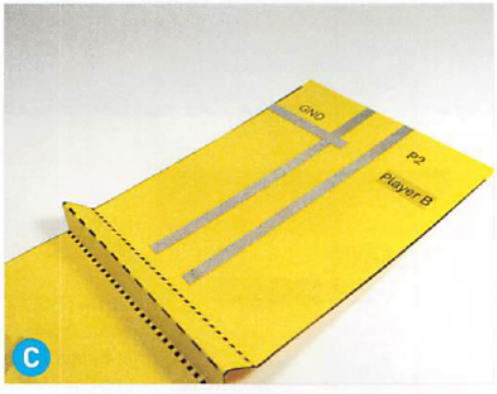
Reaction Videos!

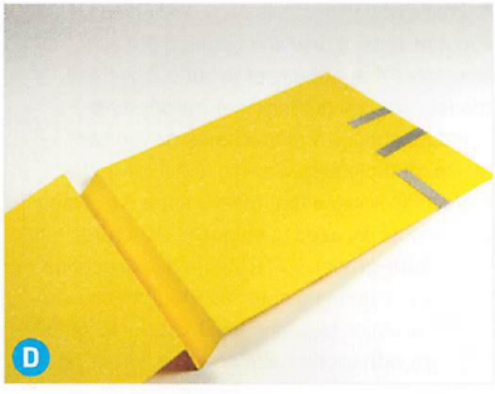
Here are a few of the Reaction Game tutorials that inspired my version. Thanks to Micro:bit Educational Foundation, Peli de Halleux of Microsoft, Brown Dog Gadgets, and other creators who generously shared their work online.

- microbit.org/projects/make-it-code-it/reaction-game
- makecode.microbit.org/projects/reaction-time
- youtube.com/watch?v=S_hzUenqAM0
- learn.browndoggadgets.com/Guide/Reaction+Game+(Paper+Version)/299









the lines of conductive tape. When you press a button, it connects the lines of tape and closes the circuit. This allows some electricity to flow between the micro:bit pin that the button is attached to (pin P0, P1, or P2) and the Ground pin (GND). The micro:bit registers the increase in voltage as input — the same way it would if you pushed one of the on-board buttons — which triggers a bit of code that shows who hit their button first and won the round!

MAKE YOUR REACTION GAME

1. MAKE THE BUTTONS

Download the template from the project page at makezine.com/go/reaction-game and print it on two sheets of cardstock. Cut two buttons out of the sheet shown in Figure A. Save the blank section for later.

Fold along the broken lines — the line in the middle folds down (mountain fold) and the outer lines fold up (valley fold) to make the spring that holds the button open (Figure B).

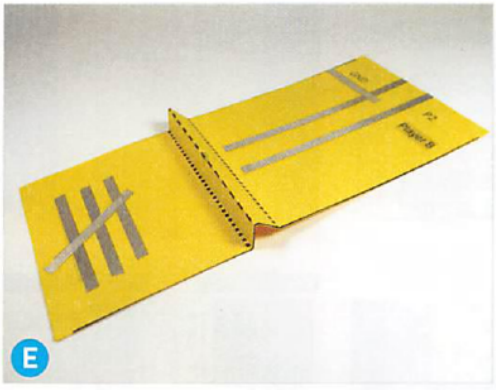
2. ADD THE CIRCUIT

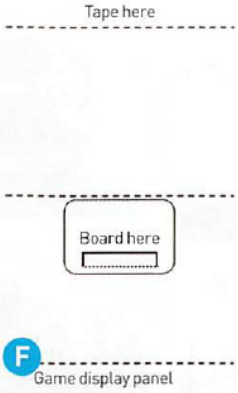
On the bottom layer of each button are markings for strips of conductive tape. Cover these lines with conductive tape (Figure C). One is marked with the pin number on the micro:bit that it connects to. The other strip connects to the ground pin (GND). The gap between them prevents short circuits. An extra piece of tape overlaps the GND line and goes off to the side of the button. This lets you connect the two buttons so they can use the same ground wire.

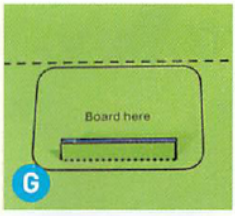
Where the tape gets to the edge of the button, bring the end over to the back for about half an inch (Figure D). This gives the alligator clips

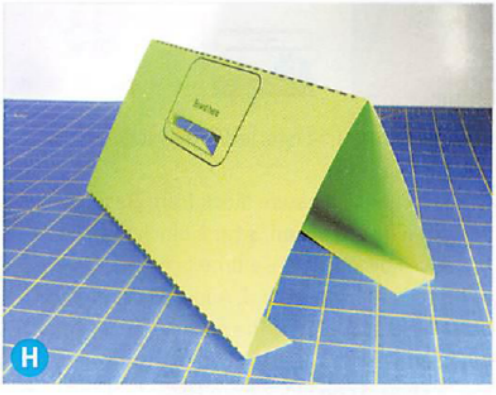
Tape Tips

- Cut pieces on a slight angle.
- Use the pad of your finger to rub the end of the tape off the backing. Then fold the backing away from the tape.
- Press the sticky end of the tape onto the paper and slowly peel the backing away as you go along the line you're covering.











more conductive material to grab onto.

On the top layer of each button, apply shorter strips of conductive tape and connect them with a cross piece, as shown in Figure E. Make sure the top and bottom line up so when you press the top down, the top strips connect the bottom strips (pin and ground) and close the circuit.

3: MAKE THE GAME DISPLAY PANEL

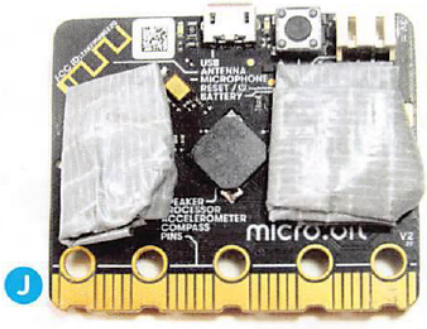
The second sheet of the template (Figure F) is the game display panel, which holds the micro:bit where the players can see and hear it, and helps keep the wires organized. To make it, first cut out and fold out the flap in the spot for the micro:bit, as shown on the template (Figure G). This creates a little shelf to hold the board, and a little window to let the alligator clips attach to the connector holes from the back of the board.

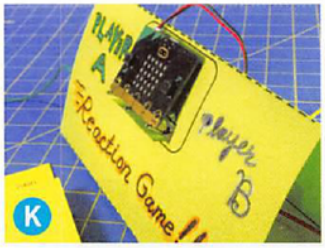
Next, fold the cardstock in half, short edges together. Fold the bottom edges up about 1". Then open the sheet partway to form a tent (Figure H). Fold both bottom flaps in.

If you want to decorate your game display panel, flatten the paper again and do it now. You might want to label each player's side with its own color and personality. When you're done, tape the extra strip of cardstock from the button template to the back flap to make a "floor" for the tent (Figure I). Leave the other side loose for now.

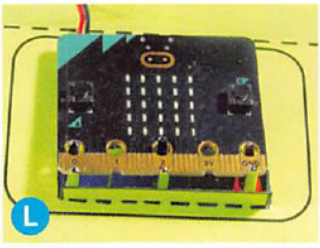
4. CONNECT THE MICRO:BIT

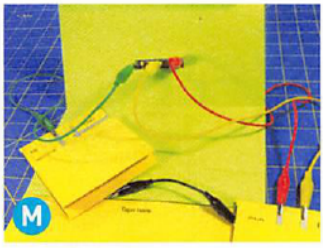
Roll up some loops of masking or duct tape, sticky side out, and press them onto the back of the micro:bit board (Figure J). Make them fat enough to stick out past the components.

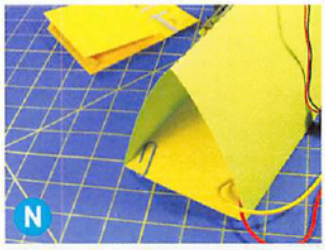


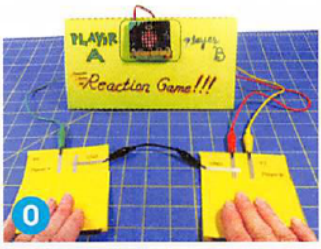


K

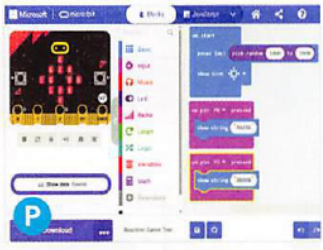

L


M


N



O


P

Next, press the micro:bit board onto the display panel so it rests on the little shelf. Attach alligator clip wires to the proper connector holes on the micro:bit (Pins 0, 2, and GND) by poking them through the window in the game display panel from the back (Figures K and L). If possible, use different color wires to help make sure you are connecting the correct parts.

Connect three wires as shown in Figure M:

- The green wire connects Pin P0 to Player A's button where shown on the template.
- The yellow wire connects Pin P2 to Player B's button where shown.
- The red wire connects the GND pin to the top of the conductive tape strip labeled GND on Player B's button.

Now pull the buttons to the proper side and close the panel by clipping the side of the tent to the floor with paper clips (Figure N). Finally, connect the two buttons by attaching the black alligator clip to the GND conductive tape strips that go off to the side on each button (Figure O).

5. CODE THE BASIC GAME

The Reaction Game Test program (Figure P) displays a "go" signal at unexpected times, and scrolls text on the LED screen to show which player pressed their button first. These instructions will show you how to make it; you can also open and edit the code at makecode.microbit.org/_CcK423PL4Kdo. (If you're new to the micro:bit and MakeCode, go to microbit.org/get-started/getting-started/introduction for help getting started.)

First, choose a **pause** block from Basic blocks and put it inside the **on start** block to make the micro:bit wait. Pause time is measured in milliseconds (written ms). Add a **pick random** block from Math to set the number of seconds between 1 and 5 (1000 to 5000 milliseconds). Use a **show icon** block from Basic to display an image on the LED grid that tells the players to go. (I chose a diamond-shaped target icon.)

Next, get an **on pin pressed** block from Input to trigger a **show string** block from Basic that scrolls a series of characters. Leave pin number at 0. Type in the message "AAAAA" to show that Player A has won.

For Player B, duplicate the **on pin pressed** block, but this time change the pin number to 2 and the message to "BBBBB".

Download the code to the micro:bit board to check if everything is working. Does the icon show up when the program starts to run on the micro:bit? Do the messages scroll on the LED grid when you press one of the buttons? If not, try these troubleshooting tips:

- If none of the code runs, make sure the top conductive tape strips on your button connect with the strips on the bottom when the button is pressed.
- If that's not the problem, check that the right parts are connected by the alligator clip wires, and that the wires are clipped on securely.

- If part of the code runs but not all of it, check it against Figure P to make sure everything is correct.

6. AMP UP YOUR GAME

The basic reaction test is OK, but it can be improved. For one thing, there's no way to prevent a false start if a player hits their button before the go signal. And if both players hit their buttons, both get a "win" message. So this Enhanced version of the game (Figure Q) waits until the go signal before registering a button press, and only counts the first time someone hits a button for each round. This game also resets automatically, so you don't have to restart the micro:bit to play again. You can open and edit the code at makecode.microbit.org/_P9xCM2TTrADy. Here's how to make it:

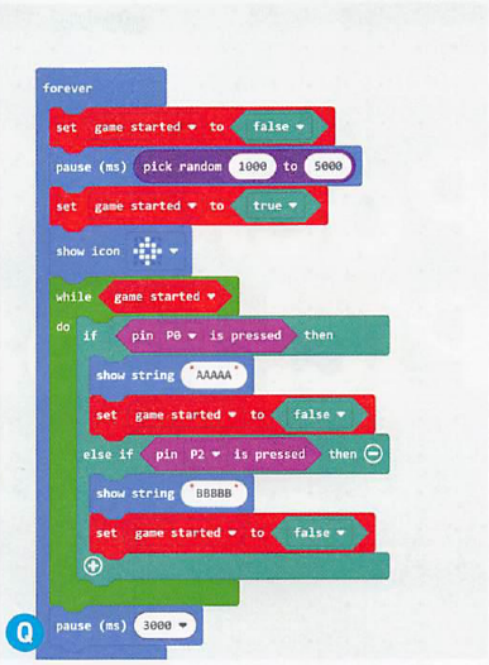
Move the **on start** code into the **forever** loop. Then create a **variable** — a block that can be set to different values — and give it a name like "game started." Insert a pointy-ended **true** block from Logic inside a **set [game started] to** block by snapping it into the space where a number normally goes. It's like turning the **game started** variable on.

Next, insert a **while** block from the Loops menu inside the **forever** block at the bottom of the stack. Drag and drop a pointy-ended **game started** variable block into the space on the **while** block so the program keeps repeating as long as the **game started** variable is set to **true**.

Put another Logic block called **if-then-else** inside the **while** loop. Insert a pointy-sided **pin [0] is pressed** block from Input into the **if** space to show what happens when Player A hit their button first. Underneath, add a "AAAAA" winning message and change the **game started** variable to **false**. This stops the **while** block from continuing to run, because the condition is no longer **true**.

Click the plus sign on the **if-then-else** block to get the **else if** option. Duplicate the blocks for Player A and adjust the condition to show what happens when Player B hits their button.

Now you have a game that lets players hit their buttons only after the starting image shows, and only displays a message for the person who wins the reaction race!





GOING FURTHER

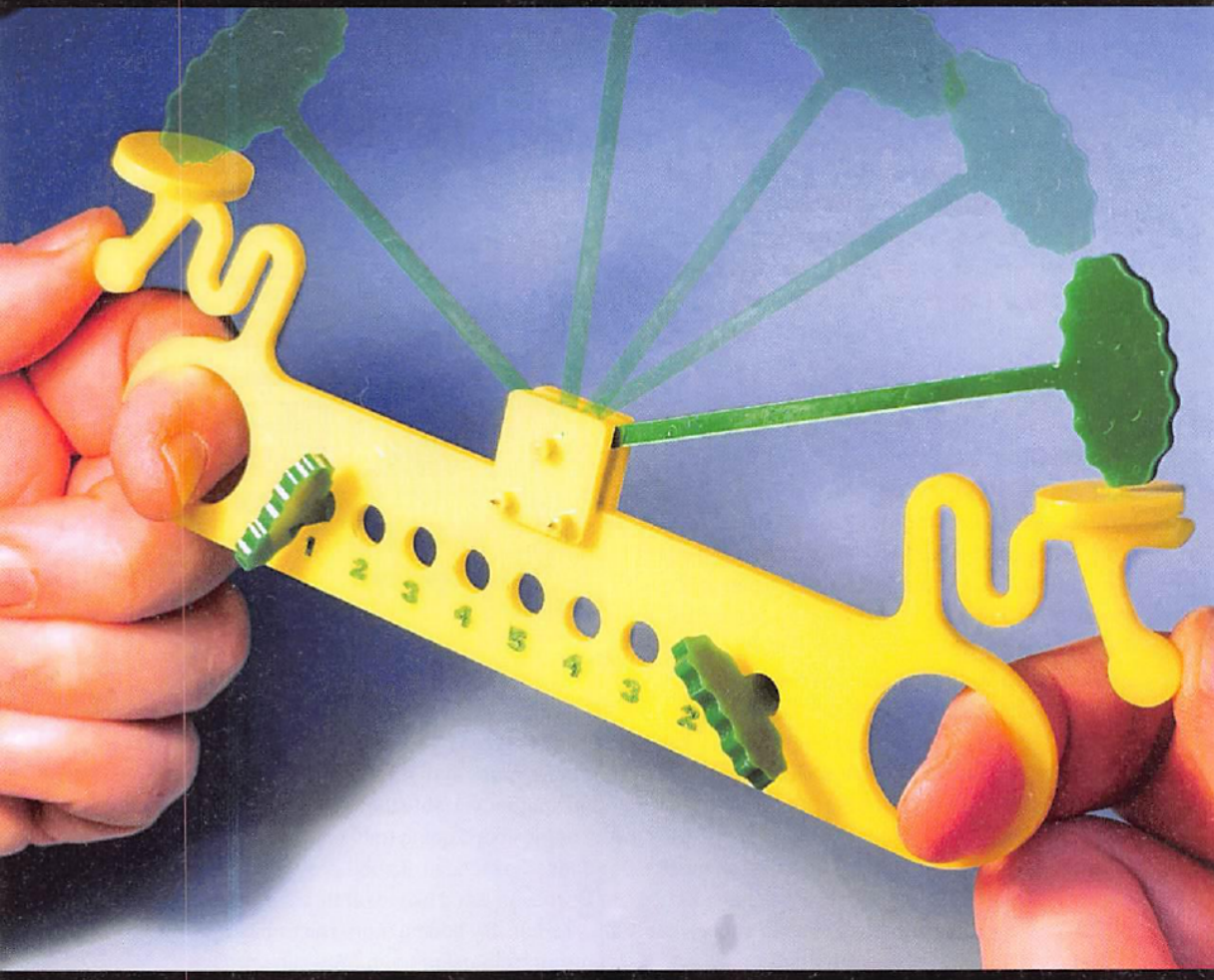
Want to keep going? This 3-Round version (Figure R) plays musical themes for each player and adds another variable that acts as a counter, so the game keeps going until one player wins three rounds: makecode.microbit.org/_aV35udi3CaMT

Or maybe you'd like to let three or four people play? (Hint: Try adding more micro:bits that communicate using radio.) You can probably find endless ways to make the Reaction Game your own. I hope it entices you to check out more games and fun in *Paper Inventions*, 2nd Edition. Enjoy!

Pocket Pickleball

Make this pocket-sized, two-player action game — that plays with a pickle instead of a ball!

Written and photographed by Bob Knetzger





BOB KNETZGER is a designer/inventor/musician whose award-winning toys have been featured on *The Tonight Show*, *Nightline*, and *Good Morning America*. He is the author of *Make: Fun!*, available at makershed.com and fine bookstores.

Pickleball is sweeping the nation and getting court time can be tricky — so here's a pocket-sized version you can play anywhere, anytime. It's easy to make from laser-cut acrylic with a few bits of metal from a paper clip.

In a time-honored tradition of the toy business, this version is a knock-off of a cereal premium from over 40 years ago: Cap'n Crunch Thumb Tennis, produced by Leon Levy of Taico Design Products. The free-inside-the-cereal-box toy was molded in polypropylene, which is tough and flexible enough to work as springy snappers. Here's the original TV ad: youtu.be/8sZlMnVave0

This new version is made from brittle acrylic, but with a wave-shaped design for stress-free springiness. Instead of a ball, you flick a swinging pickle! Metal pins hold the parts together and serve as a low-friction axle.

TIME REQUIRED: 1 Hour

DIFFICULTY: Easy

COST: $2

MATERIALS

- **Acrylic sheet, ⅛" thick** All the parts can easily be cut from one 4"×8" piece of plex. Use separate, smaller pieces for different colors as shown.
- **Small paper clip**
- **Cyanoacrylate (CA) glue** aka super glue
- **Acrylic paint**

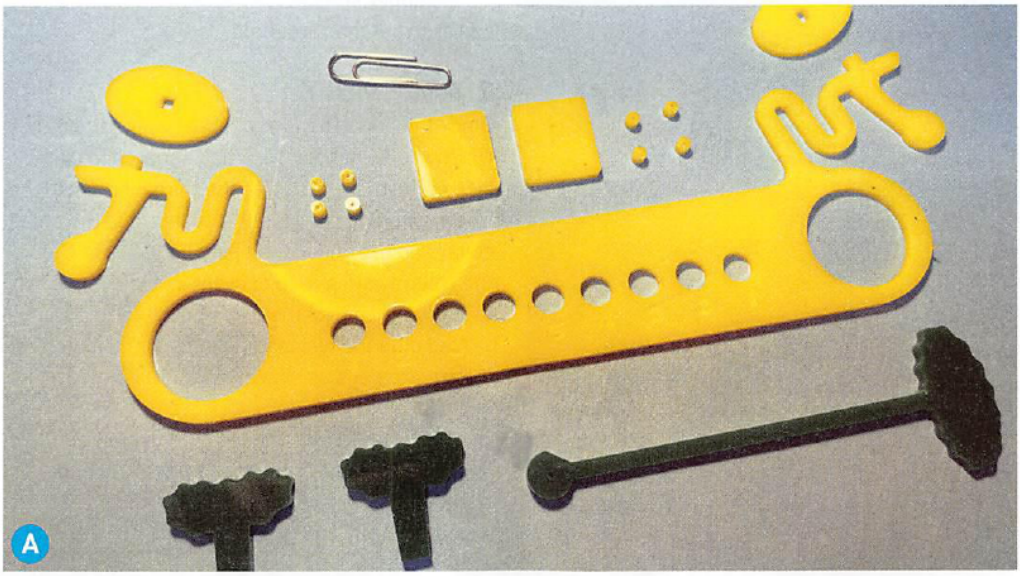
TOOLS

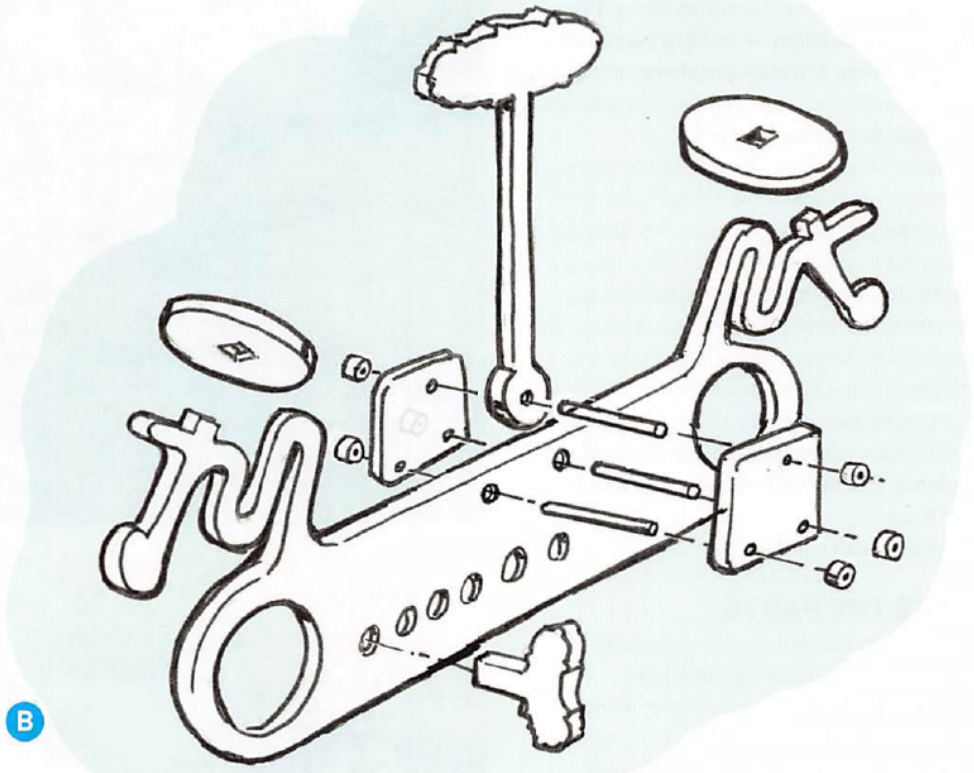
- **Laser cutter**
- **Wire cutters**



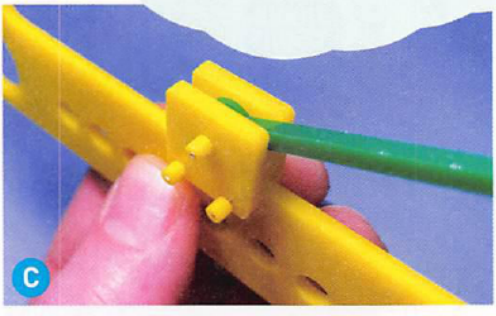
1. MAKE THE PARTS

Go to makezine.com/go/pocket-pickleball to download the *.svg* files for cutting the parts. I used a Glowforge laser cutter, which automatically identifies "cut" and "etch" paths and also sets the proper speeds and power settings for the material. I used Glowforge Proof Grade materials in fun colors: yellow for the game parts and green for the pickle parts (Figure A). Adjust the settings for your particular laser cutter and materials.


A



B



C



D

Snip three 1" straight pieces from a small paper clip for the pins. The polished 0.030" metal wire is super smooth for a low-friction axle and is easier to cut than steel rod.

2. ASSEMBLE

Stack the parts to make a three-layer sandwich. Use two pins to hold the small squares to either side of the main part. Use the third pin to hold the freely swinging pickle in the center. Add the small collars on the ends of the pins (Figure B). Glue the collars with a drop of cyanoacrylate, and then trim off the ends of the pins (Figure C).

Also glue the two landing pads (or "paddles") to the square pegs as shown in Figure D.

3. DECORATE

I did a quick paint-and-wipe to fill the etched scoring hole numbers (Figure E). Acrylic paint wipes off nicely and is easy to clean up with water.

PLAY IT!

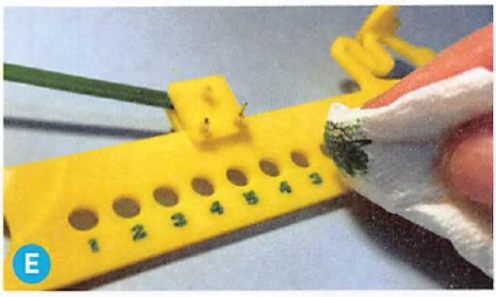
Go solo to hone your Pocket Pickleball skills: hold with both hands and use your thumbs to flick the pickle back and forth (Figure F). Time your flick so that the pickle hits the paddle — and then instantly flies off (Figure G). If you flick too early or too late, or too weakly, the pickle lands on your paddle — that's a fault. How many flicks can you do in a row without missing?

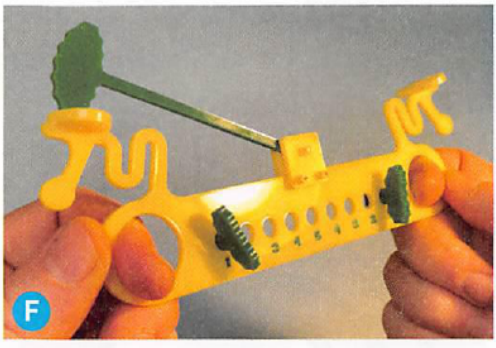
Two-player Pocket Pickleball is scored in a simplified way. One player "serves" and the other player flicks the pickle back to return the volley. If you fault, the other player scores a point: move their scoring pickle peg to the next numbered hole (Figure H). Then the faulting player serves and play resumes. First player to five points wins!

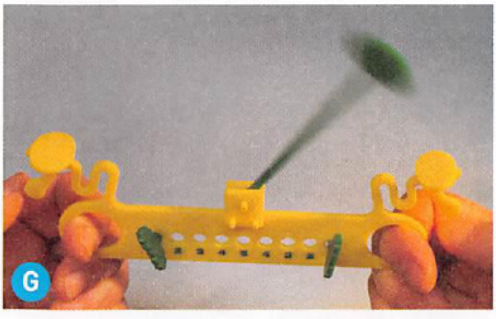
CREATE FUN VARIATIONS!

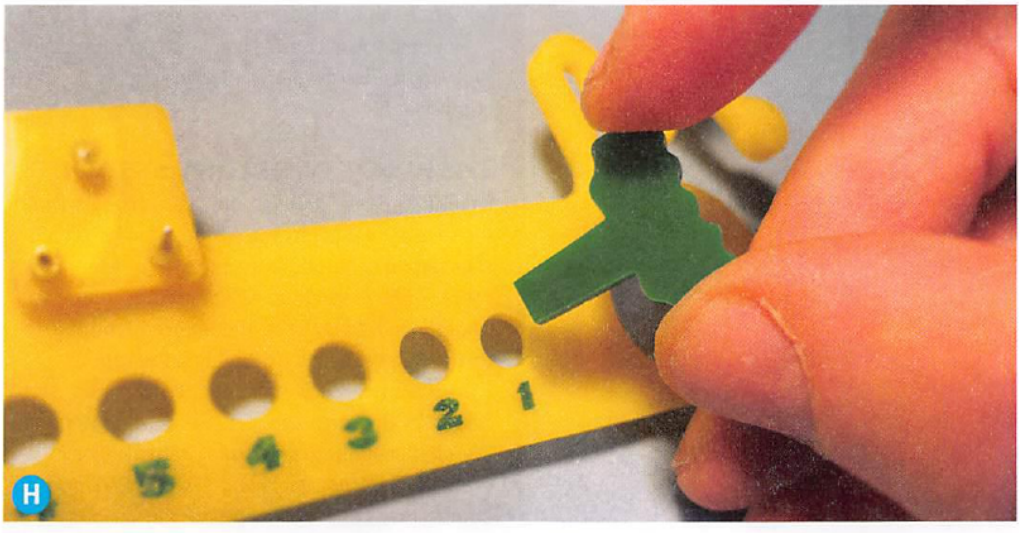
Make your own version of Pocket Pickleball:

- **Doubles:** Design a four-handed version by adding two more thumb flickers, in pairs mounted at a gentle angle (redesign with wedge-shaped connectors, or heat and bend the thumb flickers for the needed angle.) A sloppy, slightly oversized hole in the swinging pickle arm allows some random action, landing the pickle on the left or the right paddle. Flick the correct thumb to send the pickle back!
- **Mega:** Or try scaling the whole thing up for a giant, tabletop-sized version — that would be a "big dill!"





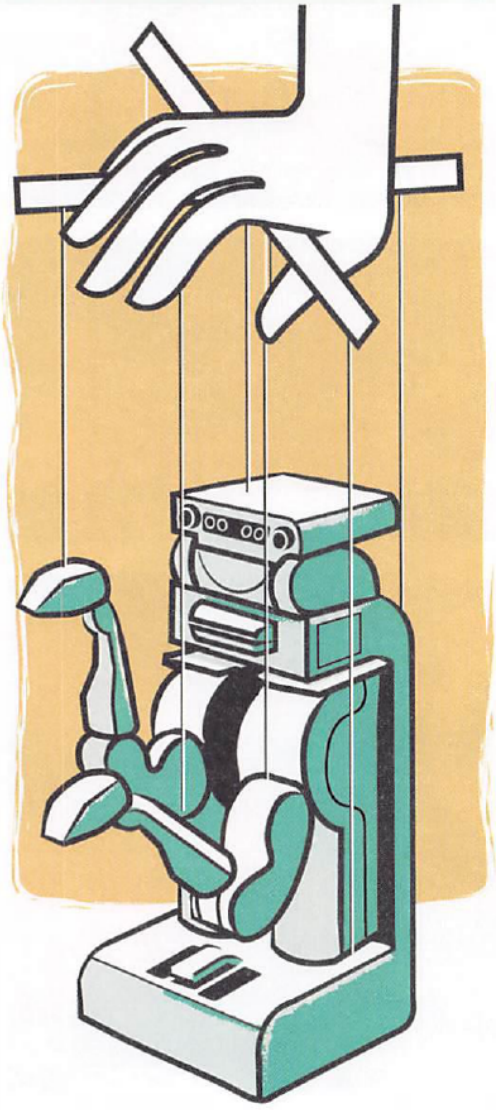




HANDS ON WITH ROS 2
NODES, TOPICS, AND SERVICES

An introduction to the Robot Operating System for hobbyists and engineers

Written by Shawn Hymel



You've been hired to work on a large humanoid robot along with a dozen other engineers. Your job is to develop the arms and hands: moving them to specific locations, picking up objects, shaking human hands, and so on. You do not have direct sensor input, as that's another engineer's job. You also need to work with the locomotion team, as the arms need to move to help balance the robot. Everyone meets one day to figure out how all this will work together.

In the conference room, on a giant whiteboard, you map out all the components and how they'll interact. You need the cameras and motion sensors in the head to send raw data to the main processing core, which computes motions for the arms and legs. But those positions also affect the stability of the robot, so positional data needs to be sent back to the processing code.

How will you accomplish all this communication? Will you need to develop a unique scheme for each component?

THE ROBOT OPERATING SYSTEM

Up until the late 2000s, roboticists struggled with these exact concepts and would often re-create messaging schemes from scratch for every new robot. "Reinventing the wheel" of messaging and underlying frameworks ended up consuming more time than actually building the robot!

In 2006, two Ph.D. students at Stanford's Salisbury Robotics Lab, Eric Berger and Keenan Wyrobek, set out to solve this problem. They created the ***Robot Operating System (ROS)*** to standardize communication among various robot

Josh Ellingson CC BY-NC 2.0, Timothy Vollmer CC BY 2.0

components. Scott Hassan, founder of the Willow Garage incubator, took notice and invited Berger and Wyrobek to continue their work in Willow Garage's program.

Over the next 3 years, the team built the PR2 robot, a successor to the PR1 begun at Stanford, and fleshed out ROS to act as the underlying software framework for the PR2.

ROS is an open-source robotics ***middleware*** framework and collection of libraries. It is not a true "operating system" like Windows, macOS, or Linux, as it cannot control hardware directly and does not have a kernel for handling processes and memory allocation. Rather, it is built on top of an operating system (usually Linux), handles multiprocessing communication, and offers a collection of computational libraries, such as the Transform Library 2 (TF2) for handling coordinate frame transformations.

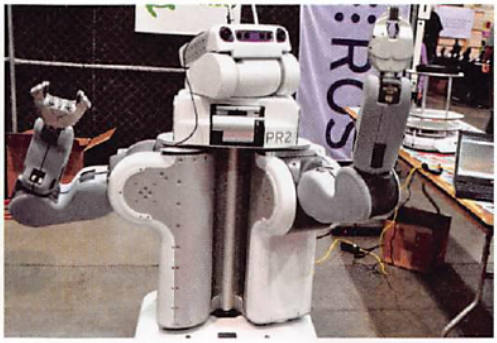
Because ROS requires a full operating system and assumes you're working in a multiprocessing environment, it is not well suited for simple, single-purpose robotics, like basic vacuum robots or maze solvers. Rather, scalability is the deciding factor: when you have multiple, complex components that need to operate together, ROS can save you hundreds of hours of work and frustration.

While ROS is used across academia for research, it also has been adopted by industry for real, commercial robots. Examples include some of the Amazon warehouse robots, Avidbots commercial-grade cleaners, and Omron's TM manipulator arms.

If you're looking to build a large, complex helper bot for household chores, improve your robotic programming skills for a job, or simply see what the hype is about, we'll walk you through installing ROS and creating simple communication examples using topics and services.

INSTALL ROS DOCKER IMAGE

The first iteration of ROS had some technical limitations in the underlying messaging layers, so the team created ROS 2, which began life in 2014. ROS 1 reached end-of-life status on May 31, 2025, which means it will no longer receive updates or support. ROS 2 has fully replaced it.



The PR2 is an advanced research robot capable of navigating human environments and interacting with objects (seen here at Maker Faire Bay Area in 2011).

About once a year the ROS team releases a new ***distribution***, which is a versioned set of ROS packages, much like Linux distributions. Each release is given a whimsical, alliterative name featuring a turtle and progressing through the alphabet. The latest release, Kilted Kaiju, came out in May 2025, but we will stick to Jazzy Jalisco, which has long-term support until 2029.

Each ROS distribution is pinned to a very particular version of an operating system to ensure that all of its underlying libraries work properly. Jazzy Jalisco's officially supported operating systems are Ubuntu 24.04 and Windows 10 (with Visual Studio 2019).

For a real robot that communicates with motors and sensors, you likely want Ubuntu installed on a small laptop or single-board computer (e.g. Raspberry Pi). For this tutorial, I will demonstrate a few ROS principles using a premade Docker image. This image pins various package versions and works across all the major operating systems (macOS, Windows, and most Linux distributions).

If you do not have it already installed on your host computer, head to docker.com, download Docker Desktop, and run the installer. Accept all the defaults.

Next, get the ROS Docker image and example repository. Navigate to github.com/ShawnHymel/introduction-to-ros, click Code, and click Download ZIP. Unzip the archive somewhere on your computer.

Open a command line terminal (e.g. zsh, bash, PowerShell), navigate to the project directory, and build the image:




```
cd introduction-to-ros/
docker build -t env-ros2 .
```

Wait while the Docker image builds (Figure A). It is rather large, as it contains a full instance of Ubuntu 24.04 with a graphical interface.

Once that finishes, run the image with one of the following commands, depending on your operating system.

For macOS or Linux:

```
docker run --rm -it -e PUID=$(id
-u) -e PGID=$(id -g) -p 22002:22
-p 3000:3000 -v "${PWD}/workspace:/
config/workspace" env-ros2
```

For Windows (PowerShell):

```
docker run --rm -it -e PUID=$(wsl id
-u) -e PGID=$(wsl id -g) -p 22002:22
-p 3000:3000 -v "${PWD}\workspace:/
config/workspace" env-ros2
```

If everything works, you should see the Xvnc KasmVNC welcome message (Figure B). You can ignore the **keysym** and **mieq** warnings as well as the **xkbcomp** error message.

Open a browser on your host computer and navigate to https://localhost:3000. You should be presented with a full Ubuntu desktop (Figure C).

TOPICS: PUBLISH AND SUBSCRIBE

In ROS 2, applications are divided up into a series of ***nodes***, which are independent processes that handle specific tasks, such as reading sensor data, processing algorithms, or driving motors. Each node runs separately in its own runtime environment and can communicate with other nodes using a few basic techniques.

The first communication method is the ***topic***, which relies on a publish/subscribe messaging model. A publisher can send data to a named topic, and the underlying ROS system will handle delivering that message to any node subscribed to that topic (Figure D).

Nodes in ROS are independent runtime processes, which essentially means they're separate programs that can be written in one of several supported programming languages. Out of the box, ROS 2 supports Python and C++, but you can write Nodes in other community-supported languages like Ada, C, Java, .NET (e.g. C#), Node.js (JavaScript), Rust, and Flutter (Dart). The beauty of ROS is that nodes written in one language can communicate with nodes written in other languages!

In general, you'll find C++ used for low-level drivers and processes that require fast execution. Python nodes, on the other hand, offer faster development time with some runtime overhead, which makes them great for prototyping and working with complex vision processing (e.g. OpenCV) and machine learning frameworks (e.g. PyTorch, TensorFlow).

ROS relies heavily on object-oriented programming principles. Individual nodes are written as subclasses of the **Node** class, inheriting properties from it as defined by ROS. Publishers and subscribers are object instances within these nodes. As a result, your custom node can have any number of publishers and subscribers.

In our example, we will create a simple subscriber in a node that listens on the **my_topic** topic channel and prints to the console whatever it hears over that channel. We will then create a simple publisher in another node that transmits

"Hello world" followed by a count value to that topic twice per second.

CREATE A ROS PACKAGE

Double-click the VS Code ROS2 icon on the left of the desktop. This opens an instance of VS Code in the Docker container preconfigured with various extensions, and it automatically enables the ROS 2 development environment.

Click View → Terminal to open a terminal pane at the bottom of VS Code.

Navigate into the *src/* directory in the *workspace/* folder. Note that we mounted the *workspace/* directory from the host computer's copy of the *introduction-to-ros* repository. That gives you access to all the code from the repository, and any changes you make to files in the *workspace/* directory will be saved on your host computer. If you change anything in the container outside of that directory, it will be lost, as the container is completely deleted when you exit!

In ROS 2, a ***workspace*** is a directory where you store and build ROS packages. The *workspace/* folder is considered a ROS 2 workspace. A ***package*** is a fundamental unit of code organization in ROS 2. You will write one or more nodes (parts of your robot application) in a package in the *src/* directory in the workspace. When you build your nodes (in a package), any required libraries, artifacts, and executables end up in the *install/* directory in the workspace. ROS 2 uses the *build/* and *log/* directories to put intermediate artifacts and log files, respectively.

From the *workspace/* directory, navigate into the *src/* folder and create a package. Note that the Docker image mounts the *workspace/* directory (on your host computer) to */config/workspace/* in the container.

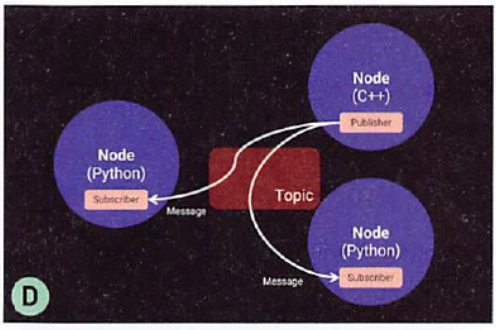
```
cd /config/workspace/src
ros2 pkg create --build-type ament_
python my_first_pkg
```

This will create a directory named *my_first_pkg/* in *workspace/src/*. You might need to click the Refresh Explorer button at the top of VS Code to see the folder show up (Figure E). The *my_first_pkg/* folder will contain a basic template for

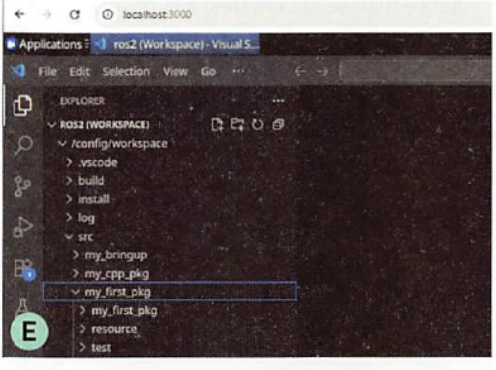




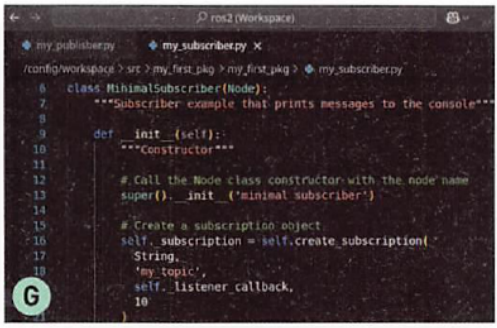
This *env-ros2* Docker image is based on the XFCE Ubuntu webtop image maintained by the **LinuxServer.io** group.



Each node in ROS is a separate process written in one of several supported programming languages. Nodes can contain one or more publishers, which push messages out over a named topic. They can also contain one or more subscribers, which, if subscribed to the same topic, will receive a copy of the message.







creating nodes. Note that we used **ament_python** as the build type, which tells ROS 2 that we intend to use Python to create nodes in this package.

We will write our source code in *my_first_pkg/my_first_pkg/*. The other files in *my_first_pkg/* help ROS understand how to build and install our package.

The workspace contains a number of other example packages from the GitHub repository, such as *my_bringup* and *my_cpp_pkg*. You are welcome to explore those to see how to implement other nodes and features in ROS.

CREATE PUBLISHER AND SUBSCRIBER NODES

Create a new file named *my_publisher.py* in *my_first_pkg/my_first_pkg/* and open it in VS Code:

```
code my_first_pkg/my_first_pkg/my_
publisher.py
```

Open a web browser and navigate to bit.ly/41TlYuj to get the publisher Python code. Copy the code from that GitHub page into your *my_publisher.py* document. Feel free to read through the comments.

Notice that we are creating a subclass named **MinimalPublisher** from the ROS-provided **Node** class, which gives us access to all the data and methods in **Node**. Inside our **MinimalPublisher**, we create a new publisher object with **self.create_publisher()**, which is a method in the **Node** class (Figure F).

We also create a new timer object and set it to call the **_timer_callback()** method every 0.5 seconds. In that method, we construct a "Hello world: " string followed by a counter number that we increase each time **_timer_callback()** executes.

In our **main()** entrypoint, we initialize **rclpy**, which is the ROS Client Library (RCL) for Python. This gives us access to the **Node** class and other ROS functionality in our code. We create an instance of our **MinimalPublisher()** class and then tell it to **spin()**, which just lets our node run endlessly.

If our program crashes or we manually exit (e.g. with Ctrl+C), we perform some cleanup to ensure that our node is gracefully removed and **rclpy** is shut down. The last two lines are common Python practice: if the current file is run as the main application, Python assigns the String **'__main__'** to the internal variable **__name__**. If this is the case, then we tell Python to run our **main()** function.

Save your code. Create a new file named *my_subscriber.py* in *my_first_pkg/my_first_pkg/* and open it in VS Code:

```
code my_first_pkg/my_first_pkg/my_
subscriber.py
```

Open a web browser and navigate to bit.ly/3JqL0ut to get the subscriber Python code. Copy the code from that GitHub page into your *my_subscriber.py* document.

Similar to our publisher, we create a subclass from **Node** named **MinimalSubscriber** (Figure G). In that node, we instantiate a subscription object with a callback. The callback is the method **_listener_callback()** that gets called whenever a message appears on the topic we subscribe to (**'my_topic'**). The message is passed into the method as the **msg** parameter. In our callback, we simply print that message to the screen.

As with our previous program's **main()**, we initialize **rclpy**, instantiate our node subclass, and let it run. We catch any crashes or exit conditions and gracefully shut everything down.

Don't forget to save your work!

BUILD THE PACKAGE

Before we can build our package, we need to tell ROS about our nodes and list any dependencies. Open *my_first_pkg/package.xml*, which was created when we generated the package template. This file is the ***package manifest*** that lists any metadata and dependencies for the ROS package. You can give your package a unique name and keep track of the version here. The only line we need to add tells ROS that we are using the **rclpy** package as a dependency in our package. Just after the **<license>** line, add the following:

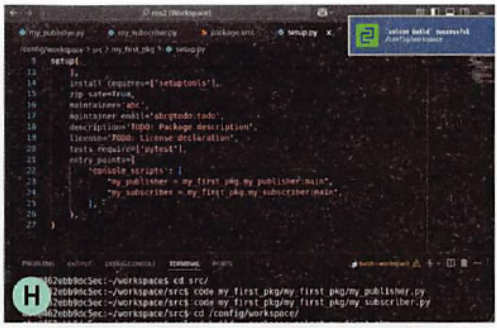
```
<license>TODO: License declaration</
license>

<depend>rclpy</depend>

<test_depend>ament_copyright</test_
depend>
```

We add this line as our package requires **rclpy** to build and run, as noted by the **import rclpy** line in our publisher and subscriber code. While you can import any Python packages or libraries you might have installed on your system in your node source code, you usually want to list imported ROS packages in *package.xml*. This helps the ROS build systems and runtime know what ROS packages to use with your package. Save this file.

Next, we need to tell the ROS build system which source files it should build and install. Open *my_first_pkg/setup.py*, which was also created on package template generation. The **entry_points** parameter lists all of the possible entry points for the executables in the package. It allows us to list functions in our code that act as entry points for new applications or processes.

Add the following to the **console_scripts** key:

```
entry_points={
    'console_scripts': [
        "my_publisher = my_first_pkg.
my_publisher:main",
        "my_subscriber = my_first_pkg.
my_subscriber:main",
    ],
},
```



H



I

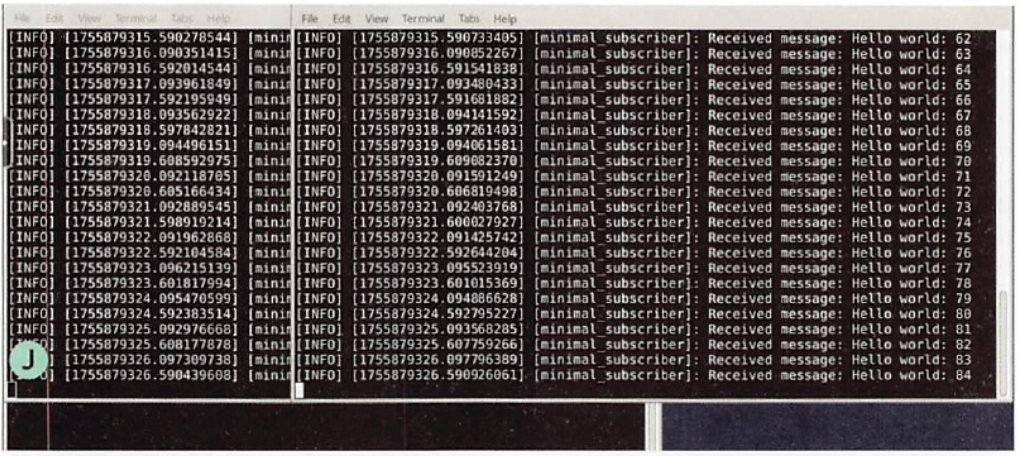
Save this file. We're finally ready to build! In the terminal, navigate back to the workspace directory and use the **colcon build** command to build just our package, which we select with the **--packages-select** parameter. **Colcon** is the build tool for ROS 2 and helps to manage the workspace.

```
cd /config/workspace/
colcon build --packages-select my_
first_pkg
```

Your package should build without any errors (Figure H).

RUN YOUR PUBLISHER AND SUBSCRIBER

Now it's time to test your code. Click on the Applications menu in the top-left of the container window, and click Terminal Emulator. Repeat this two more times to get three terminal windows (Figure I). You are welcome to use the desktop icons in the top-right of the container window to work on a clean desktop.



We need two of the terminal windows to run our publisher and subscriber as two separate processes. We'll use the third window to examine a graph of how our nodes are communicating.

In the first terminal, source the new package environment and run the node. The Docker image is configured to initialize the global ROS environment, so ROS commands and built-in packages work in these terminal windows. However, ROS does not know about our new package yet, so we need to tell it where to find that package. So, every time you open a new terminal window, you need to source the environment for the workspace you wish to use, which includes all the packages built in that workspace. We do this by running an automatically generated bash script in our workspace.

From there, run our custom publisher node using the **ros2 run** command:

```
cd /config/workspace/
source install/setup.bash
ros2 run my_first_pkg my_publisher
```

This should start running the publisher in the first terminal.

In the second terminal, source the workspace **setup.bash** script again and run the subscriber:

```
cd /config/workspace/
source install/setup.bash
ros2 run my_first_pkg my_subscriber
```

You should see the **"Hello world: "** string followed by a counter appear in both terminals. The publisher prints its message to the console for debugging, and the subscriber prints out any messages it receives on the **my_topic** topic (Figure J).

Finally, in the third terminal, run the RQt graph application, which gives you a visualization of how your two nodes are communicating:

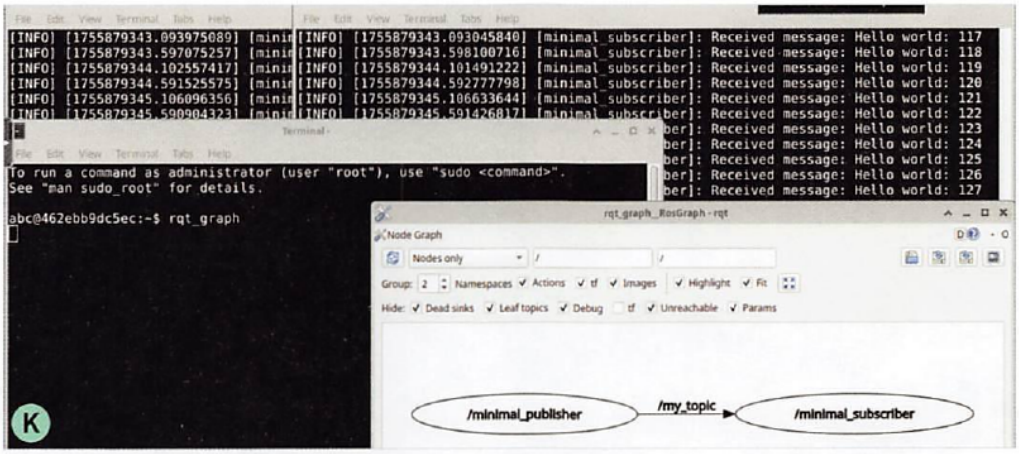
```
rqt_graph
```

RQt is a graphical interface that helps users visualize and debug their ROS 2 applications. The RQt graph shows nodes and topics (Figure K), which can be extremely helpful as you start running dozens or hundreds of nodes in your robot application. RQt contains many other useful tools, but we'll stick to the RQt graph for now. Note that you might need to press the Refresh button in the top-left of the window to get the nodes to appear.

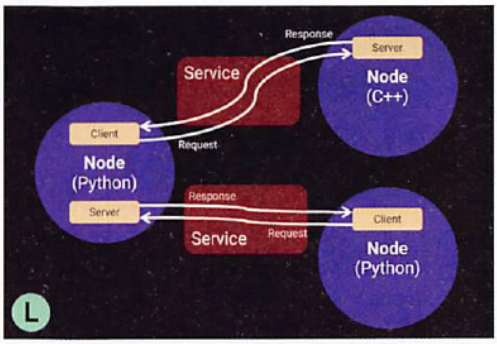
Press Ctrl+C in each of the terminals or close them all to stop the nodes.

CLIENTS AND SERVERS, REQUESTS AND RESPONSES

The publish/subscribe model works well when you need to regularly transmit data out on a particular subject, such as sensor readings, but it doesn't work well when you need one node to make a request to another node. For this, ROS has another method for handling messaging between nodes: ***services***.



A service follows a client/server model where one node acts as a server, waiting to receive incoming requests. Another node acts as a client and sends a request to that particular server and waits for a response (Figure L). This pattern works well in instances where you want to set a parameter, trigger an action, or receive a one-time update from another node. For example, your computation node might request that the motion node moves the robot forward by some amount.



As with topics, nodes can contain one or more servers, which broadcast their ability to respond to requests via a particular service. Nodes can also instantiate clients, which are used to send requests to the servers.

CREATE SUBSCRIBER AND CLIENT NODES

In *my_first_pkg/my_first_pkg/*, create a new file, *my_server.py*:

```
cd /config/workspace/src/
code my_first_pkg/my_first_pkg/my_
server.py
```

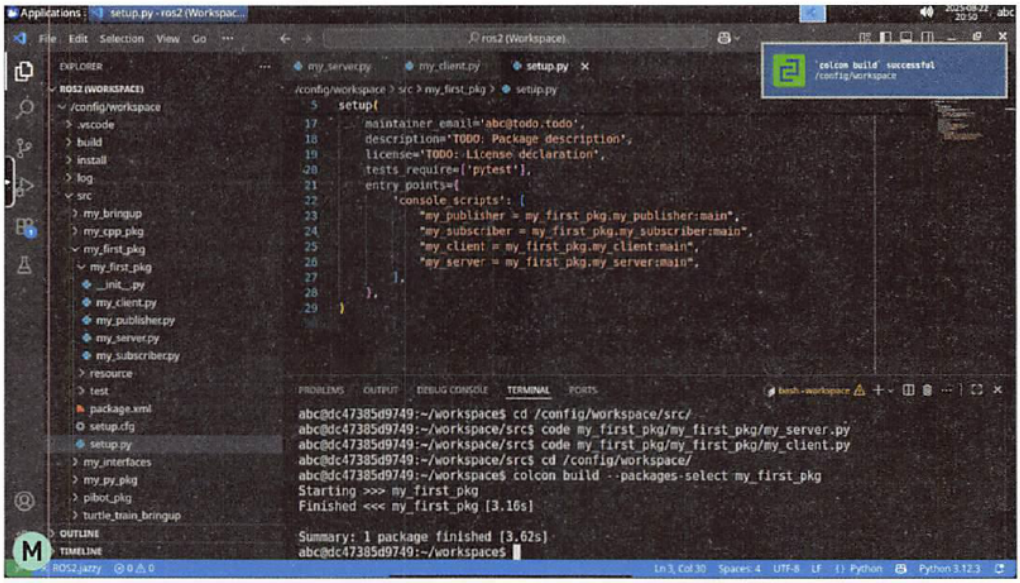
Copy the code found at bit.ly/4fSWUcT and paste it into that file.

Here, we create another subclass node named **MinimalServer**. In that node, we instantiate a service named **add_ints**, which turns the node into a server. We must specify an interface for the service so that we know what kind of data it contains. In this case, we specify **AddTwoInts** as the interface, which was imported at the top of the file in the from **example_interfaces.srv...** line. ROS 2 contains a number of example interfaces, but you can also define your own. Finally, we attach the callback method **_server_callback()** to the service.

Whenever a request comes in for that named service, **_server_callback()** is called. The request is stored in the **req** parameter. This service works only with the **AddTwoInts** interface, so we know that the request will have two fields: **a** and **b**, each containing an integer. We add the two integers together, store the sum in the **sum** field of the response, print a message to the console for debugging, and then return the response object. The ROS 2 framework will handle the underlying details of delivering the response message back to the client. Save your file.

Now let's create our client. In *my_first_pkg/my_first_pkg/*, create *my_client.py*:

```
code my_first_pkg/my_first_pkg/my_
client.py
```



Copy the code found at bit.ly/47bipmK and paste it into that file.

In our client node, we create a client object and give it the interface, **AddTwoInts**, and the name of the service, **'add_ints'**. We also create a timer object, much like we did for the publisher node. In the callback method for the timer, we fill out the request, which adheres to the **AddTwoInts** interface. We then send the request message to the server and assign the result to a **future**. We repeat this process every 2 seconds.

A ***future*** is an object that acts as a placeholder for a result that will be available later. We add a callback to that future, which gets executed when this node receives the response from the server. At that point, the future is complete, and we can access the value within. The response callback simply prints the resulting sum to the console. Don't forget to save your work!

As we did with the publisher and subscriber examples, we need to tell the ROS 2 build system about our new nodes. Add the following to *my_first_pkg/setup.py*:

```
entry_points={
    'console_scripts': [
        "my_publisher = my_first_pkg.
my_publisher:main",
        "my_subscriber = my_first_pkg.
my_subscriber:main",
        "my_client = my_first_pkg.
my_client:main",
        "my_server = my_first_pkg.
my_server:main",
    ],
},
```

Rebuild your package:

```
cd /config/workspace/
colcon build --packages-select my_first_
pkg
```

The package should build without any errors (Figure M).

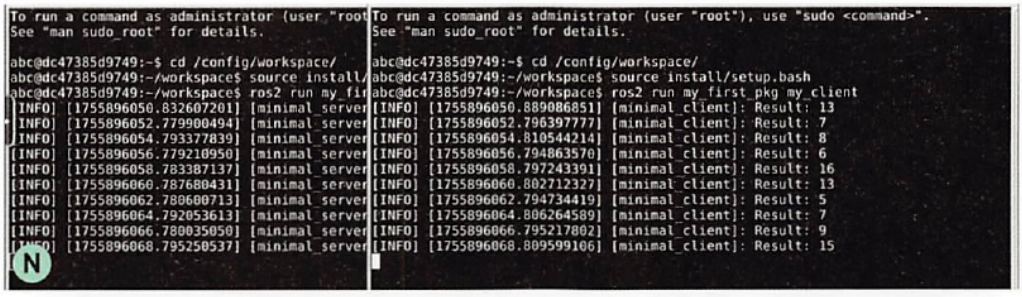
RUN YOUR SERVER AND CLIENT

As you did with the publisher and subscriber demo, open three terminal windows. In the first window, source your workspace environment and run the server node:

```
cd /config/workspace/
source install/setup.bash
ros2 run my_first_pkg my_server
```

In the second window, source the workspace environment again and run the client node:

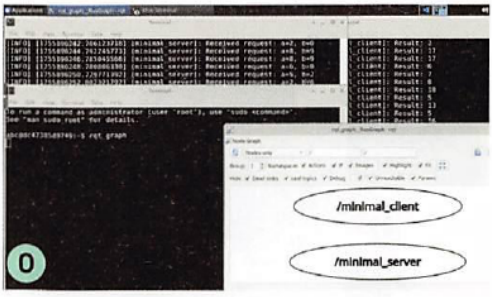
```
cd /config/workspace/
```




```
source install/setup.bash
ros2 run my_first_pkg my_client
```

You should see the server terminal receive the request from the client containing two random integers for **a** and **b**, each between 0 and 10. The server sends the response back to the client, which prints the sum to the terminal (Figure N).

You are welcome to run **rqt_graph** in the third terminal (Figure O), but you will only see the nodes — no service interface or lines connecting them as you saw with topics.



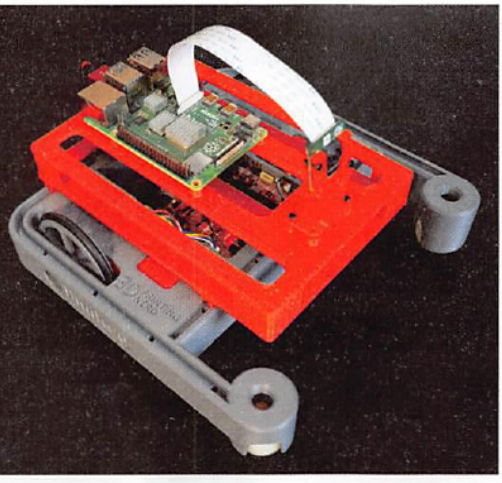
MIGHTY MIDDLEWARE

It might seem odd that we spent all this time just getting some pieces of software to talk to each other, but these concepts form the basis for ROS. Remember, ROS is a middleware messaging layer at its core, not a collection of robot drivers or sensor libraries. ROS solves an important challenge by helping developers scale software projects in large, complex robotics. It contains far too much overhead to be useful for smaller robotics projects.

That being said, I hope this brief introduction satisfied your curiosity about ROS. If you'd like to learn more, check out the examples and getting started video series at github.com/ShawnHymel/introduction-to-ros. Beyond the messaging system we just looked at, ROS ships with a number of libraries, like TF2, diagnostic tools, and visualizers to help you build, test, and deploy your robot software. Topics and services are just the beginning; ROS scales up to handle extremely complex designs and is currently used in many commercial robots found around the world!



SHAWN HYMEL is a course creator and instructor for edge AI and embedded systems that inspire and teach developers of all skill levels. Based in Lafayette, Colorado, he can be found giving talks, running workshops, and swing dancing in his free time.

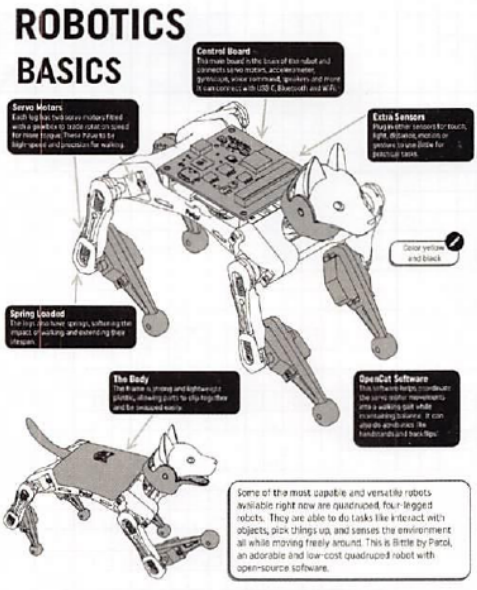


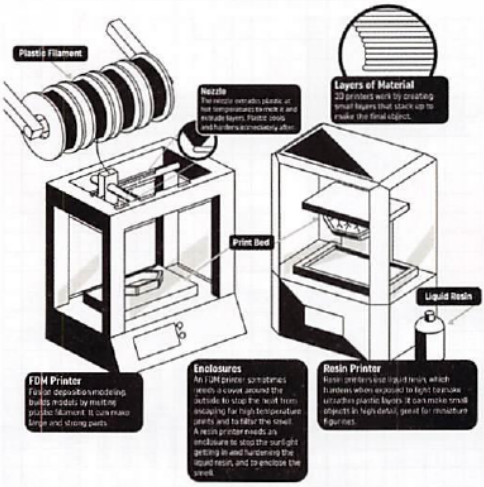


Digi-Key, Josh Ellingson CC BY-NC 2.0

MAKING MAKERS

Inspire discovery in budding makers!





Skill Seeker: Young Maker Edition by Steph Piper is an activity and guide book that gamifies the learning process of key maker skills, putting beginner makers on a STEAM path where they'll want to level up!

Designed for makers ages 8–12, special "skill trees" provide a structured path of skills to learn, leading young makers from one interconnected accomplishment to the next, gently challenging without overwhelming. Twelve foundational skill trees are included, covering such subjects as 3D printing, vinyl cutting, electronics, crafting, and robotics.

To start readers down their paths, special "Learn & Do" activities explain how seemingly complex things work, like simple guidelines to successfully design 3D models for printing; using layers to efficiently create drawings, paintings, and digital art; and even "kidpreneurship" concepts to develop, test, and bring ideas to life.

To help engage and personalize the learning experience, readers create character avatars and track their progress by coloring in achievement badges that reveal their progress and unlock their next challenges.

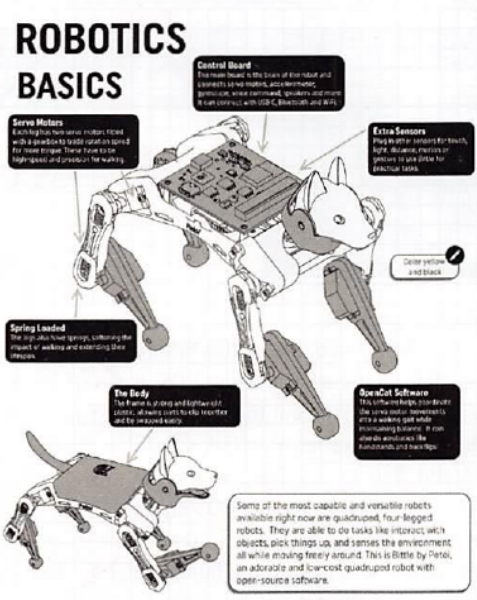
Cultivate the young makers in your life with *Skill Seeker: Young Maker Edition*! —*Kevin Toyama*

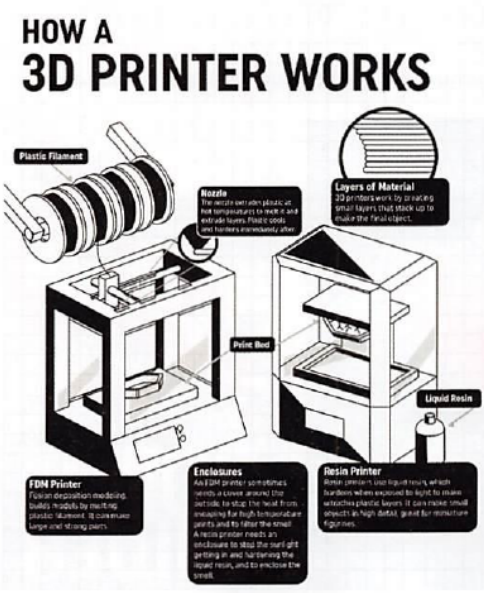


Steph Piper helped develop adult makers with *Skill Seeker: Maker Edition*, and has now created ***Skill Seeker: Young Maker Edition*** to support the next generation of makers. Available now at makershed.com and bookstores everywhere.

MAKING MAKERS

Inspire discovery in budding makers!





Skill Seeker: Young Maker Edition by Steph Piper is an activity and guide book that gamifies the learning process of key maker skills, putting beginner makers on a STEAM path where they'll want to level up!

Designed for makers ages 8–12, special "skill trees" provide a structured path of skills to learn, leading young makers from one interconnected accomplishment to the next, gently challenging without overwhelming. Twelve foundational skill trees are included, covering such subjects as 3D printing, vinyl cutting, electronics, crafting, and robotics.

To start readers down their paths, special "Learn & Do" activities explain how seemingly complex things work, like simple guidelines to successfully design 3D models for printing; using layers to efficiently create drawings, paintings, and digital art; and even "kidpreneurship" concepts to develop, test, and bring ideas to life.

To help engage and personalize the learning experience, readers create character avatars and track their progress by coloring in achievement badges that reveal their progress and unlock their next challenges.

Cultivate the young makers in your life with *Skill Seeker: Young Maker Edition*! —*Kevin Toyama*



Steph Piper helped develop adult makers with *Skill Seeker: Maker Edition*, and has now created ***Skill Seeker: Young Maker Edition*** to support the next generation of makers. Available now at makershed.com and bookstores everywhere.

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

abc@dc47385d9749:~$ cd /config/workspace/
abc@dc47385d9749:~/workspace$ source install/setup.bash
abc@dc47385d9749:~/workspace$ ros2 run my_first_pkg my_client
[INFO] [1755896050.832607201] [minimal_server]: Received request: 2, 9
[INFO] [1755896052.779900494] [minimal_server]: Received request: 7, 0
[INFO] [1755896054.793377839] [minimal_server]: Received request: 8, 0
[INFO] [1755896056.779210950] [minimal_server]: Received request: 6, 0
[INFO] [1755896058.783387137] [minimal_server]: Received request: 8, 8
[INFO] [1755896060.787680431] [minimal_server]: Received request: 5, 8
[INFO] [1755896062.780600713] [minimal_server]: Received request: 5, 0
[INFO] [1755896064.792053613] [minimal_server]: Received request: 7, 0
[INFO] [1755896066.780035050] [minimal_server]: Received request: 9, 0
[INFO] [1755896068.795250537] [minimal_server]: Received request: 9, 6
```

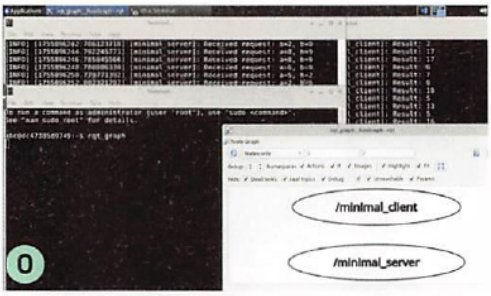
N

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

abc@dc47385d9749:~$ cd /config/workspace/
abc@dc47385d9749:~/workspace$ source install/setup.bash
abc@dc47385d9749:~/workspace$ ros2 run my_first_pkg my_client
[INFO] [1755896050.889086851] [minimal_client]: Result: 13
[INFO] [1755896052.796397777] [minimal_client]: Result: 7
[INFO] [1755896054.810544214] [minimal_client]: Result: 8
[INFO] [1755896056.794863570] [minimal_client]: Result: 6
[INFO] [1755896058.797243391] [minimal_client]: Result: 16
[INFO] [1755896060.802712327] [minimal_client]: Result: 13
[INFO] [1755896062.794734419] [minimal_client]: Result: 5
[INFO] [1755896064.806264589] [minimal_client]: Result: 7
[INFO] [1755896066.795217802] [minimal_client]: Result: 9
[INFO] [1755896068.809599106] [minimal_client]: Result: 15
```

```
source install/setup.bash
ros2 run my_first_pkg my_client
```

You should see the server terminal receive the request from the client containing two random integers for **a** and **b**, each between 0 and 10. The server sends the response back to the client, which prints the sum to the terminal (Figure N).

You are welcome to run **rqt_graph** in the third terminal (Figure O), but you will only see the nodes — no service interface or lines connecting them as you saw with topics.



O

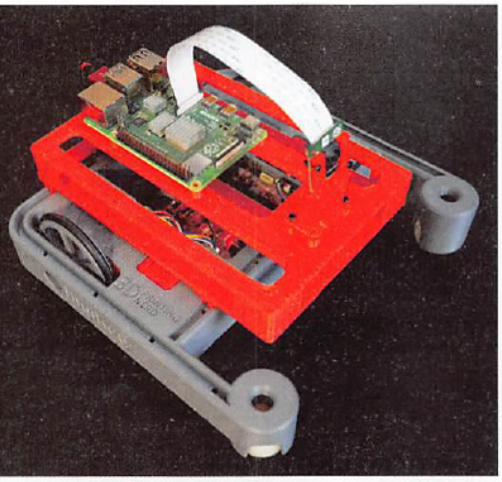
MIGHTY MIDDLEWARE

It might seem odd that we spent all this time just getting some pieces of software to talk to each other, but these concepts form the basis for ROS. Remember, ROS is a middleware messaging layer at its core, not a collection of robot drivers or sensor libraries. ROS solves an important challenge by helping developers scale software projects in large, complex robotics. It contains far too much overhead to be useful for smaller robotics projects.

That being said, I hope this brief introduction satisfied your curiosity about ROS. If you'd like to learn more, check out the examples and getting started video series at github.com/ShawnHymel/introduction-to-ros. Beyond the messaging system we just looked at, ROS ships with a number of libraries, like TF2, diagnostic tools, and visualizers to help you build, test, and deploy your robot software. Topics and services are just the beginning; ROS scales up to handle extremely complex designs and is currently used in many commercial robots found around the world!

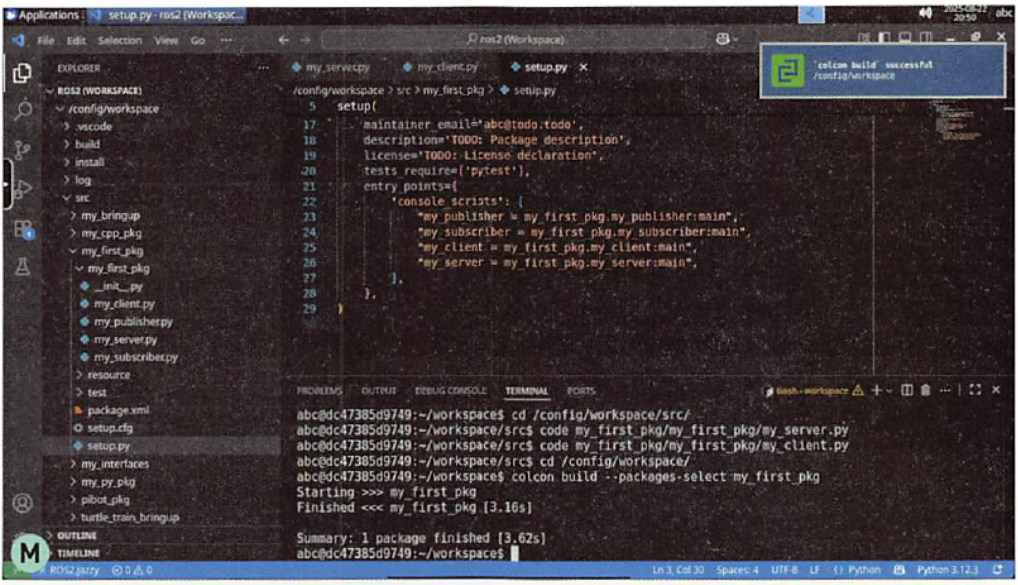


SHAWN HYMEL is a course creator and instructor for edge AI and embedded systems that inspire and teach developers of all skill levels. Based in Lafayette, Colorado, he can be found giving talks, running workshops, and swing dancing in his free time.





Digi-Key, Josh Ellingson CC BY-NC 2.0



Copy the code found at bit.ly/47bipmK and paste it into that file.

In our client node, we create a client object and give it the interface, **AddTwoInts**, and the name of the service, **'add_ints'**. We also create a timer object, much like we did for the publisher node. In the callback method for the timer, we fill out the request, which adheres to the **AddTwoInts** interface. We then send the request message to the server and assign the result to a **future**. We repeat this process every 2 seconds.

A ***future*** is an object that acts as a placeholder for a result that will be available later. We add a callback to that future, which gets executed when this node receives the response from the server. At that point, the future is complete, and we can access the value within. The response callback simply prints the resulting sum to the console. Don't forget to save your work!

As we did with the publisher and subscriber examples, we need to tell the ROS 2 build system about our new nodes. Add the following to *my_first_pkg/setup.py*:

```
entry_points={
    'console_scripts': [
        "my_publisher = my_first_pkg.
my_publisher:main",
        "my_subscriber = my_first_pkg.
my_subscriber:main",
        "my_client = my_first_pkg.
my_client:main",
        "my_server = my_first_pkg.
my_server:main",
    ],
},
```

Rebuild your package:

```
cd /config/workspace/
colcon build --packages-select my_first_
pkg
```

The package should build without any errors (Figure M).

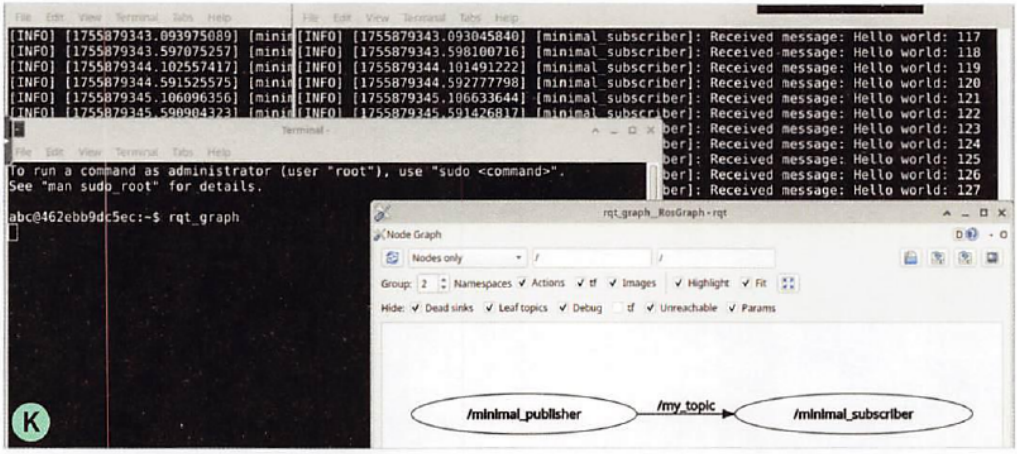
RUN YOUR SERVER AND CLIENT

As you did with the publisher and subscriber demo, open three terminal windows. In the first window, source your workspace environment and run the server node:

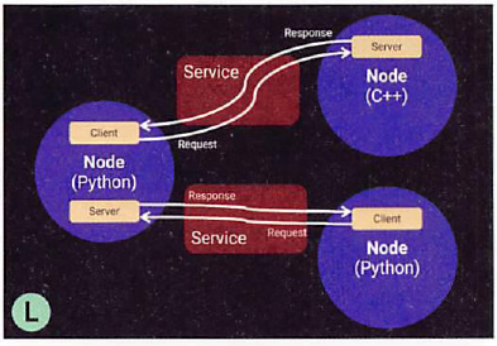
```
cd /config/workspace/
source install/setup.bash
ros2 run my_first_pkg my_server
```

In the second window, source the workspace environment again and run the client node:

```
cd /config/workspace/
```



A service follows a client/server model where one node acts as a server, waiting to receive incoming requests. Another node acts as a client and sends a request to that particular server and waits for a response (Figure L). This pattern works well in instances where you want to set a parameter, trigger an action, or receive a one-time update from another node. For example, your computation node might request that the motion node moves the robot forward by some amount.



As with topics, nodes can contain one or more servers, which broadcast their ability to respond to requests via a particular service. Nodes can also instantiate clients, which are used to send requests to the servers.

CREATE SUBSCRIBER AND CLIENT NODES

In *my_first_pkg/my_first_pkg/*, create a new file, *my_server.py*:

```
cd /config/workspace/src/
code my_first_pkg/my_first_pkg/my_
server.py
```

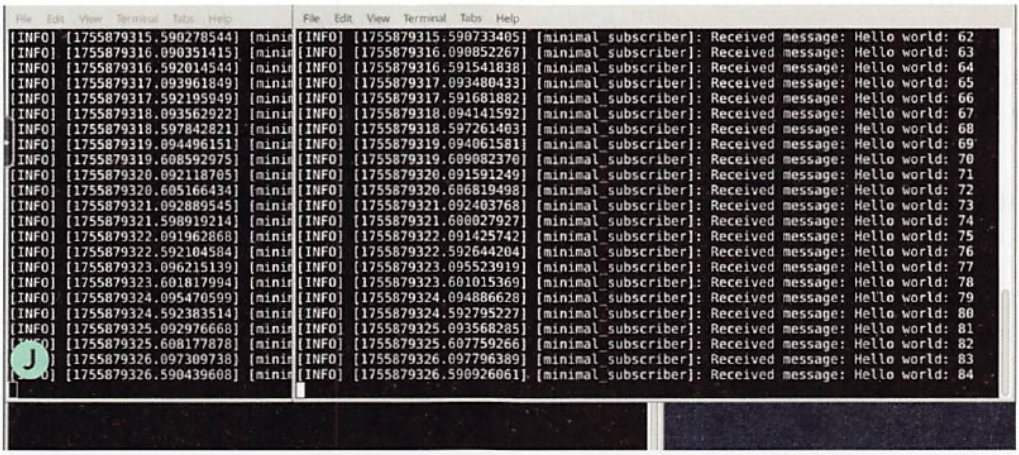
Copy the code found at bit.ly/4fSWUcT and paste it into that file.

Here, we create another subclass node named **MinimalServer**. In that node, we instantiate a service named **add_ints**, which turns the node into a server. We must specify an interface for the service so that we know what kind of data it contains. In this case, we specify **AddTwoInts** as the interface, which was imported at the top of the file in the from **example_interfaces.srv...** line. ROS 2 contains a number of example interfaces, but you can also define your own. Finally, we attach the callback method **_server_callback()** to the service.

Whenever a request comes in for that named service, **_server_callback()** is called. The request is stored in the **req** parameter. This service works only with the **AddTwoInts** interface, so we know that the request will have two fields: **a** and **b**, each containing an integer. We add the two integers together, store the sum in the **sum** field of the response, print a message to the console for debugging, and then return the response object. The ROS 2 framework will handle the underlying details of delivering the response message back to the client. Save your file.

Now let's create our client. In *my_first_pkg/my_first_pkg/*, create *my_client.py*:

```
code my_first_pkg/my_first_pkg/my_
client.py
```



We need two of the terminal windows to run our publisher and subscriber as two separate processes. We'll use the third window to examine a graph of how our nodes are communicating.

In the first terminal, source the new package environment and run the node. The Docker image is configured to initialize the global ROS environment, so ROS commands and built-in packages work in these terminal windows. However, ROS does not know about our new package yet, so we need to tell it where to find that package. So, every time you open a new terminal window, you need to source the environment for the workspace you wish to use, which includes all the packages built in that workspace. We do this by running an automatically generated bash script in our workspace.

From there, run our custom publisher node using the **ros2 run** command:

```
cd /config/workspace/
source install/setup.bash
ros2 run my_first_pkg my_publisher
```

This should start running the publisher in the first terminal.

In the second terminal, source the workspace **setup.bash** script again and run the subscriber:

```
cd /config/workspace/
source install/setup.bash
ros2 run my_first_pkg my_subscriber
```

You should see the **"Hello world: "** string followed by a counter appear in both terminals. The publisher prints its message to the console for debugging, and the subscriber prints out any messages it receives on the **my_topic** topic (Figure J).

Finally, in the third terminal, run the RQt graph application, which gives you a visualization of how your two nodes are communicating:

```
rqt_graph
```

RQt is a graphical interface that helps users visualize and debug their ROS 2 applications. The RQt graph shows nodes and topics (Figure K), which can be extremely helpful as you start running dozens or hundreds of nodes in your robot application. RQt contains many other useful tools, but we'll stick to the RQt graph for now. Note that you might need to press the Refresh button in the top-left of the window to get the nodes to appear.

Press Ctrl+C in each of the terminals or close them all to stop the nodes.

CLIENTS AND SERVERS, REQUESTS AND RESPONSES

The publish/subscribe model works well when you need to regularly transmit data out on a particular subject, such as sensor readings, but it doesn't work well when you need one node to make a request to another node. For this, ROS has another method for handling messaging between nodes: ***services***.

As with our previous program's `main()`, we initialize `rclpy`, instantiate our node subclass, and let it run. We catch any crashes or exit conditions and gracefully shut everything down.

Don't forget to save your work!

BUILD THE PACKAGE

Before we can build our package, we need to tell ROS about our nodes and list any dependencies. Open *my_first_pkg/package.xml*, which was created when we generated the package template. This file is the ***package manifest*** that lists any metadata and dependencies for the ROS package. You can give your package a unique name and keep track of the version here. The only line we need to add tells ROS that we are using the `rclpy` package as a dependency in our package. Just after the `<license>` line, add the following:

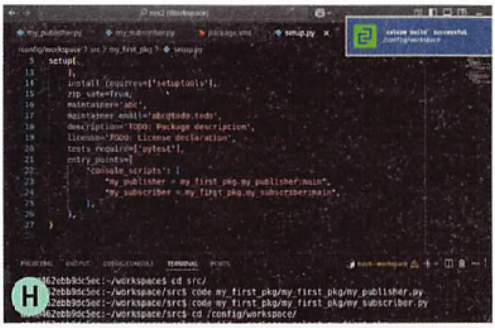
```
<license>TODO: License declaration</
license>

<depend>rclpy</depend>

<test_depend>ament_copyright</test_
depend>
```

We add this line as our package requires `rclpy` to build and run, as noted by the `import rclpy` line in our publisher and subscriber code. While you can import any Python packages or libraries you might have installed on your system in your node source code, you usually want to list imported ROS packages in *package.xml*. This helps the ROS build systems and runtime know what ROS packages to use with your package. Save this file.

Next, we need to tell the ROS build system which source files it should build and install. Open *my_first_pkg/setup.py*, which was also created on package template generation. The `entry_points` parameter lists all of the possible entry points for the executables in the package. It allows us to list functions in our code that act as entry points for new applications or processes.

Add the following to the `console_scripts` key:

```
entry_points={
    'console_scripts': [
        "my_publisher = my_first_pkg.
my_publisher:main",
        "my_subscriber = my_first_pkg.
my_subscriber:main",
    ],
},
```





Save this file. We're finally ready to build! In the terminal, navigate back to the workspace directory and use the `colcon build` command to build just our package, which we select with the `--packages-select` parameter. `Colcon` is the build tool for ROS 2 and helps to manage the workspace.

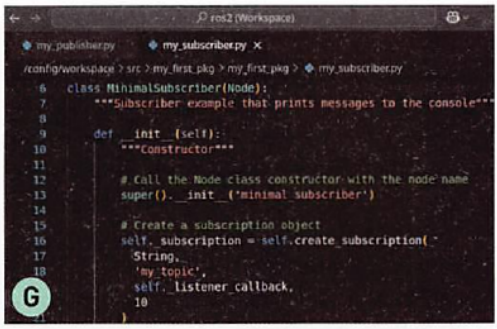
```
cd /config/workspace/
colcon build --packages-select my_
first_pkg
```

Your package should build without any errors (Figure H).

RUN YOUR PUBLISHER AND SUBSCRIBER

Now it's time to test your code. Click on the Applications menu in the top-left of the container window, and click Terminal Emulator. Repeat this two more times to get three terminal windows (Figure I). You are welcome to use the desktop icons in the top-right of the container window to work on a clean desktop.





creating nodes. Note that we used **ament_python** as the build type, which tells ROS 2 that we intend to use Python to create nodes in this package.

We will write our source code in *my_first_pkg/my_first_pkg/*. The other files in *my_first_pkg/* help ROS understand how to build and install our package.

The workspace contains a number of other example packages from the GitHub repository, such as *my_bringup* and *my_cpp_pkg*. You are welcome to explore those to see how to implement other nodes and features in ROS.

CREATE PUBLISHER AND SUBSCRIBER NODES

Create a new file named *my_publisher.py* in *my_first_pkg/my_first_pkg/* and open it in VS Code:

```
code my_first_pkg/my_first_pkg/my_
publisher.py
```

Open a web browser and navigate to bit.ly/41TlYuj to get the publisher Python code. Copy the code from that GitHub page into your *my_publisher.py* document. Feel free to read through the comments.

Notice that we are creating a subclass named **MinimalPublisher** from the ROS-provided **Node** class, which gives us access to all the data and methods in **Node**. Inside our **MinimalPublisher**, we create a new publisher object with **self.create_publisher()**, which is a method in the **Node** class (Figure F).

We also create a new timer object and set it to call the **_timer_callback()** method every 0.5 seconds. In that method, we construct a "Hello world: " string followed by a counter number that we increase each time **_timer_callback()** executes.

In our **main()** entrypoint, we initialize **rclpy**, which is the ROS Client Library (RCL) for Python. This gives us access to the **Node** class and other ROS functionality in our code. We create an instance of our **MinimalPublisher()** class and then tell it to **spin()**, which just lets our node run endlessly.

If our program crashes or we manually exit (e.g. with Ctrl+C), we perform some cleanup to ensure that our node is gracefully removed and **rclpy** is shut down. The last two lines are common Python practice: if the current file is run as the main application, Python assigns the String **'__main__'** to the internal variable **__name__**. If this is the case, then we tell Python to run our **main()** function.

Save your code. Create a new file named *my_subscriber.py* in *my_first_pkg/my_first_pkg/* and open it in VS Code:

```
code my_first_pkg/my_first_pkg/my_
subscriber.py
```

Open a web browser and navigate to bit.ly/3JqL0ut to get the subscriber Python code. Copy the code from that GitHub page into your *my_subscriber.py* document.

Similar to our publisher, we create a subclass from **Node** named **MinimalSubscriber** (Figure G). In that node, we instantiate a subscription object with a callback. The callback is the method **_listener_callback()** that gets called whenever a message appears on the topic we subscribe to (**'my_topic'**). The message is passed into the method as the **msg** parameter. In our callback, we simply print that message to the screen.

"Hello world" followed by a count value to that topic twice per second.

CREATE A ROS PACKAGE

Double-click the VS Code ROS2 icon on the left of the desktop. This opens an instance of VS Code in the Docker container preconfigured with various extensions, and it automatically enables the ROS 2 development environment.

Click View → Terminal to open a terminal pane at the bottom of VS Code.

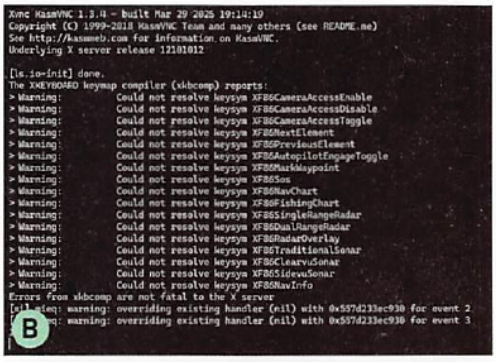
Navigate into the *src/* directory in the *workspace/* folder. Note that we mounted the *workspace/* directory from the host computer's copy of the *introduction-to-ros* repository. That gives you access to all the code from the repository, and any changes you make to files in the *workspace/* directory will be saved on your host computer. If you change anything in the container outside of that directory, it will be lost, as the container is completely deleted when you exit!

In ROS 2, a ***workspace*** is a directory where you store and build ROS packages. The *workspace/* folder is considered a ROS 2 workspace. A ***package*** is a fundamental unit of code organization in ROS 2. You will write one or more nodes (parts of your robot application) in a package in the *src/* directory in the workspace. When you build your nodes (in a package), any required libraries, artifacts, and executables end up in the *install/* directory in the workspace. ROS 2 uses the *build/* and *log/* directories to put intermediate artifacts and log files, respectively.

From the *workspace/* directory, navigate into the *src/* folder and create a package. Note that the Docker image mounts the *workspace/* directory (on your host computer) to */config/workspace/* in the container.

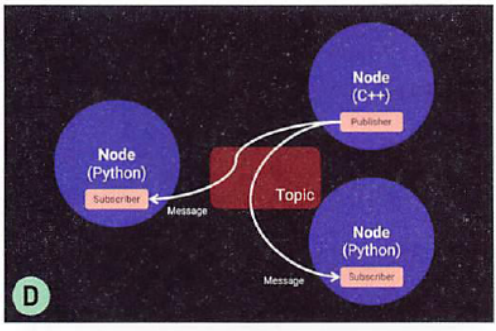
```
cd /config/workspace/src
ros2 pkg create --build-type ament_
python my_first_pkg
```

This will create a directory named *my_first_pkg/* in *workspace/src/*. You might need to click the Refresh Explorer button at the top of VS Code to see the folder show up (Figure E). The *my_first_pkg/* folder will contain a basic template for

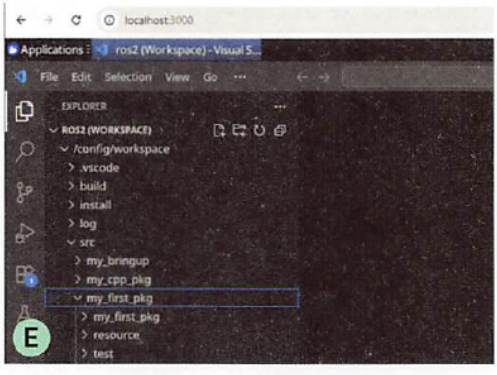




This *env-ros2* Docker image is based on the XFCE Ubuntu webtop image maintained by the **LinuxServer.io** group.



Each node in ROS is a separate process written in one of several supported programming languages. Nodes can contain one or more publishers, which push messages out over a named topic. They can also contain one or more subscribers, which, if subscribed to the same topic, will receive a copy of the message.






```
cd introduction-to-ros/
docker build -t env-ros2 .
```

Wait while the Docker image builds (Figure A). It is rather large, as it contains a full instance of Ubuntu 24.04 with a graphical interface.

Once that finishes, run the image with one of the following commands, depending on your operating system.

For macOS or Linux:

```
docker run --rm -it -e PUID=$(id
-u) -e PGID=$(id -g) -p 22002:22
-p 3000:3000 -v "${PWD}/workspace:/
config/workspace" env-ros2
```

For Windows (PowerShell):

```
docker run --rm -it -e PUID=$(wsl id
-u) -e PGID=$(wsl id -g) -p 22002:22
-p 3000:3000 -v "${PWD}\workspace:/
config/workspace" env-ros2
```

If everything works, you should see the Xvnc KasmVNC welcome message (Figure B). You can ignore the **keysym** and **mieq** warnings as well as the **xkbcomp** error message.

Open a browser on your host computer and navigate to https://localhost:3000. You should be presented with a full Ubuntu desktop (Figure C).

TOPICS: PUBLISH AND SUBSCRIBE

In ROS 2, applications are divided up into a series of ***nodes***, which are independent processes that handle specific tasks, such as reading sensor data, processing algorithms, or driving motors. Each node runs separately in its own runtime environment and can communicate with other nodes using a few basic techniques.

The first communication method is the ***topic***, which relies on a publish/subscribe messaging model. A publisher can send data to a named topic, and the underlying ROS system will handle delivering that message to any node subscribed to that topic (Figure D).

Nodes in ROS are independent runtime processes, which essentially means they're separate programs that can be written in one of several supported programming languages. Out of the box, ROS 2 supports Python and C++, but you can write Nodes in other community-supported languages like Ada, C, Java, .NET (e.g. C#), Node.js (JavaScript), Rust, and Flutter (Dart). The beauty of ROS is that nodes written in one language can communicate with nodes written in other languages!

In general, you'll find C++ used for low-level drivers and processes that require fast execution. Python nodes, on the other hand, offer faster development time with some runtime overhead, which makes them great for prototyping and working with complex vision processing (e.g. OpenCV) and machine learning frameworks (e.g. PyTorch, TensorFlow).

ROS relies heavily on object-oriented programming principles. Individual nodes are written as subclasses of the **Node** class, inheriting properties from it as defined by ROS. Publishers and subscribers are object instances within these nodes. As a result, your custom node can have any number of publishers and subscribers.

In our example, we will create a simple subscriber in a node that listens on the **my_topic** topic channel and prints to the console whatever it hears over that channel. We will then create a simple publisher in another node that transmits

components. Scott Hassan, founder of the Willow Garage incubator, took notice and invited Berger and Wyrobek to continue their work in Willow Garage's program.

Over the next 3 years, the team built the PR2 robot, a successor to the PR1 begun at Stanford, and fleshed out ROS to act as the underlying software framework for the PR2.

ROS is an open-source robotics ***middleware*** framework and collection of libraries. It is not a true "operating system" like Windows, macOS, or Linux, as it cannot control hardware directly and does not have a kernel for handling processes and memory allocation. Rather, it is built on top of an operating system (usually Linux), handles multiprocessing communication, and offers a collection of computational libraries, such as the Transform Library 2 (TF2) for handling coordinate frame transformations.

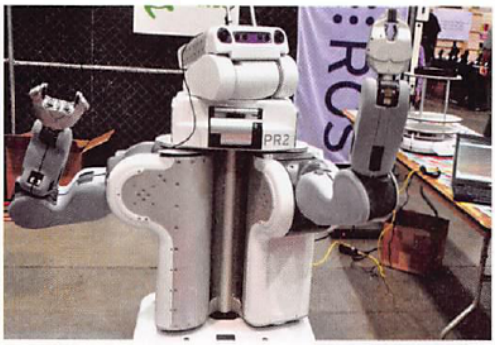
Because ROS requires a full operating system and assumes you're working in a multiprocessing environment, it is not well suited for simple, single-purpose robotics, like basic vacuum robots or maze solvers. Rather, scalability is the deciding factor: when you have multiple, complex components that need to operate together, ROS can save you hundreds of hours of work and frustration.

While ROS is used across academia for research, it also has been adopted by industry for real, commercial robots. Examples include some of the Amazon warehouse robots, Avidbots commercial-grade cleaners, and Omron's TM manipulator arms.

If you're looking to build a large, complex helper bot for household chores, improve your robotic programming skills for a job, or simply see what the hype is about, we'll walk you through installing ROS and creating simple communication examples using topics and services.

INSTALL ROS DOCKER IMAGE

The first iteration of ROS had some technical limitations in the underlying messaging layers, so the team created ROS 2, which began life in 2014. ROS 1 reached end-of-life status on May 31, 2025, which means it will no longer receive updates or support. ROS 2 has fully replaced it.



The PR2 is an advanced research robot capable of navigating human environments and interacting with objects (seen here at Maker Faire Bay Area in 2011).

About once a year the ROS team releases a new ***distribution***, which is a versioned set of ROS packages, much like Linux distributions. Each release is given a whimsical, alliterative name featuring a turtle and progressing through the alphabet. The latest release, Kilted Kaiju, came out in May 2025, but we will stick to Jazzy Jalisco, which has long-term support until 2029.

Each ROS distribution is pinned to a very particular version of an operating system to ensure that all of its underlying libraries work properly. Jazzy Jalisco's officially supported operating systems are Ubuntu 24.04 and Windows 10 (with Visual Studio 2019).

For a real robot that communicates with motors and sensors, you likely want Ubuntu installed on a small laptop or single-board computer (e.g. Raspberry Pi). For this tutorial, I will demonstrate a few ROS principles using a premade Docker image. This image pins various package versions and works across all the major operating systems (macOS, Windows, and most Linux distributions).

If you do not have it already installed on your host computer, head to docker.com, download Docker Desktop, and run the installer. Accept all the defaults.

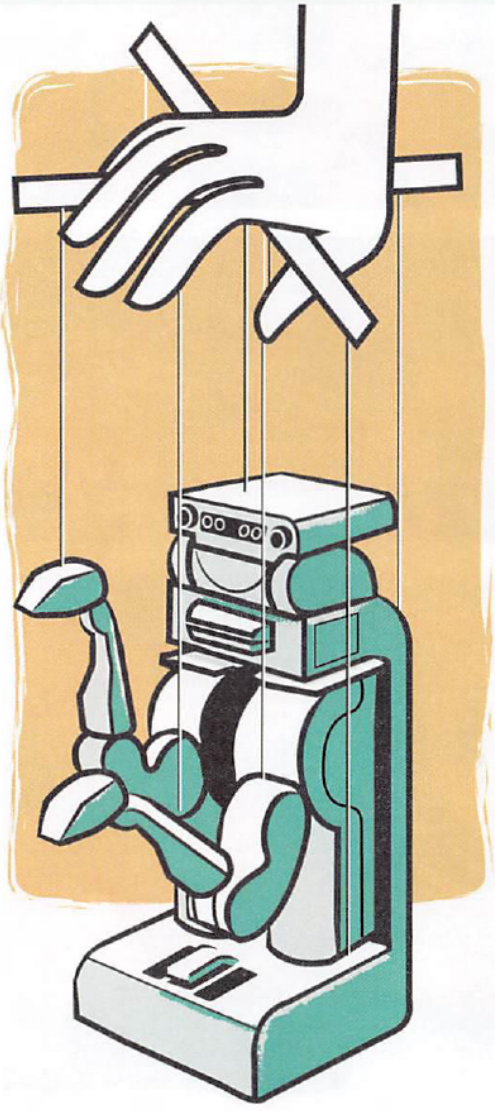
Next, get the ROS Docker image and example repository. Navigate to github.com/ShawnHymel/introduction-to-ros, click Code, and click Download ZIP. Unzip the archive somewhere on your computer.

Open a command line terminal (e.g. zsh, bash, PowerShell), navigate to the project directory, and build the image:

HANDS ON WITH ROS 2
NODES, TOPICS, AND SERVICES

An introduction to the Robot Operating System for hobbyists and engineers

Written by Shawn Hymel



You've been hired to work on a large humanoid robot along with a dozen other engineers. Your job is to develop the arms and hands: moving them to specific locations, picking up objects, shaking human hands, and so on. You do not have direct sensor input, as that's another engineer's job. You also need to work with the locomotion team, as the arms need to move to help balance the robot. Everyone meets one day to figure out how all this will work together.

In the conference room, on a giant whiteboard, you map out all the components and how they'll interact. You need the cameras and motion sensors in the head to send raw data to the main processing core, which computes motions for the arms and legs. But those positions also affect the stability of the robot, so positional data needs to be sent back to the processing code.

How will you accomplish all this communication? Will you need to develop a unique scheme for each component?

THE ROBOT OPERATING SYSTEM

Up until the late 2000s, roboticists struggled with these exact concepts and would often re-create messaging schemes from scratch for every new robot. "Reinventing the wheel" of messaging and underlying frameworks ended up consuming more time than actually building the robot!

In 2006, two Ph.D. students at Stanford's Salisbury Robotics Lab, Eric Berger and Keenan Wyrobek, set out to solve this problem. They created the ***Robot Operating System (ROS)*** to standardize communication among various robot

Josh Elurgson CC BY-NC 2.0, Timothy Vollmer CC BY 2.0

PLAY IT!

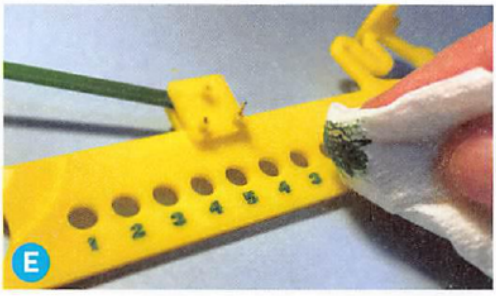
Go solo to hone your Pocket Pickleball skills: hold with both hands and use your thumbs to flick the pickle back and forth (Figure F). Time your flick so that the pickle hits the paddle — and then instantly flies off (Figure G). If you flick too early or too late, or too weakly, the pickle lands on your paddle — that's a fault. How many flicks can you do in a row without missing?

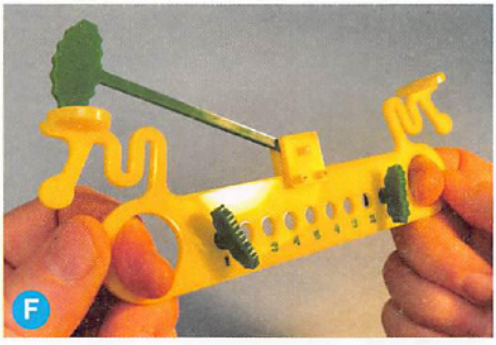
Two-player Pocket Pickleball is scored in a simplified way. One player "serves" and the other player flicks the pickle back to return the volley. If you fault, the other player scores a point: move their scoring pickle peg to the next numbered hole (Figure H). Then the faulting player serves and play resumes. First player to five points wins!

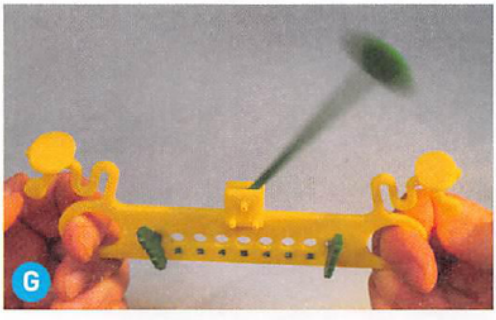
CREATE FUN VARIATIONS!

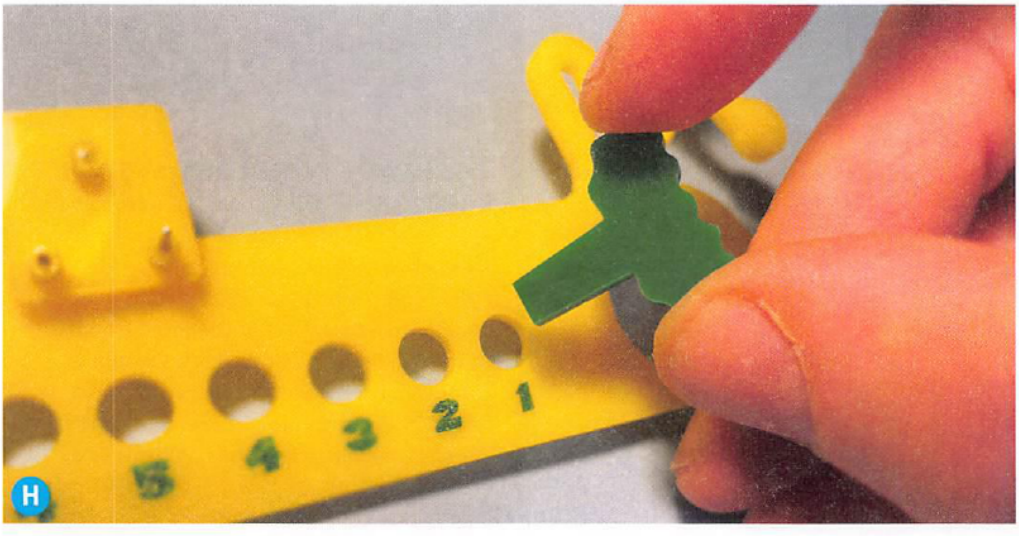
Make your own version of Pocket Pickleball:

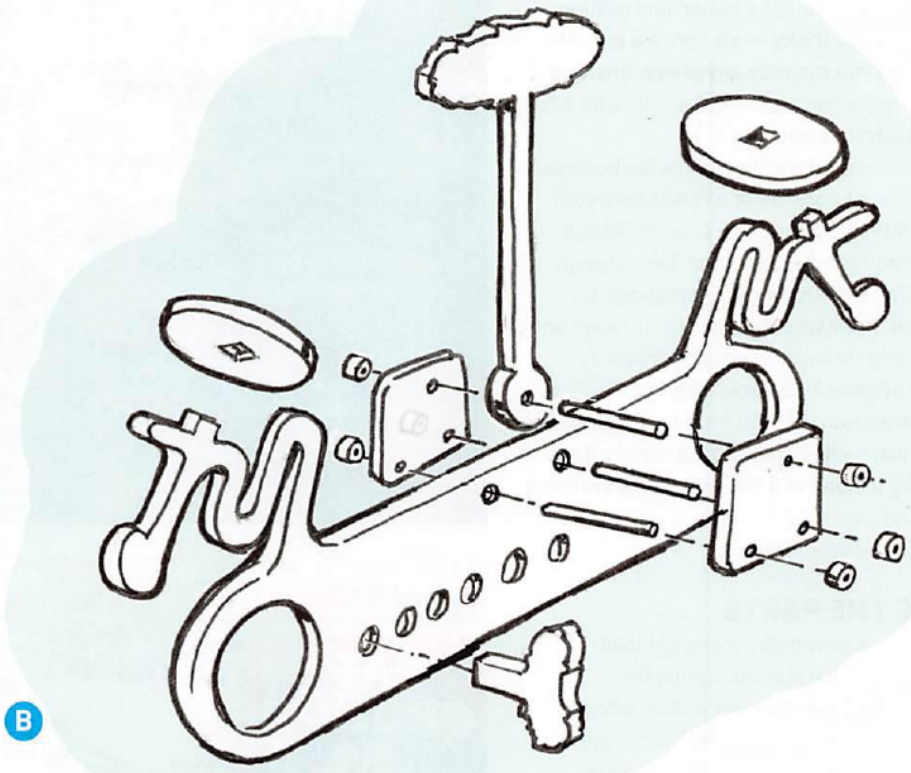
- **Doubles:** Design a four-handed version by adding two more thumb flickers, in pairs mounted at a gentle angle (redesign with wedge-shaped connectors, or heat and bend the thumb flickers for the needed angle.) A sloppy, slightly oversized hole in the swinging pickle arm allows some random action, landing the pickle on the left or the right paddle. Flick the correct thumb to send the pickle back!
- **Mega:** Or try scaling the whole thing up for a giant, tabletop-sized version — that would be a "big dill!"



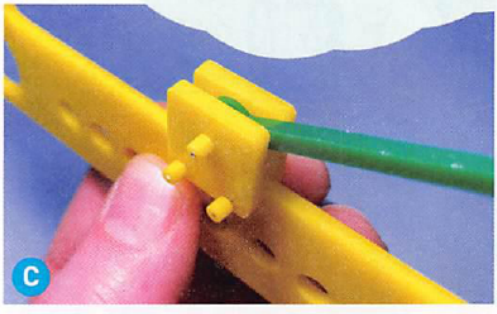




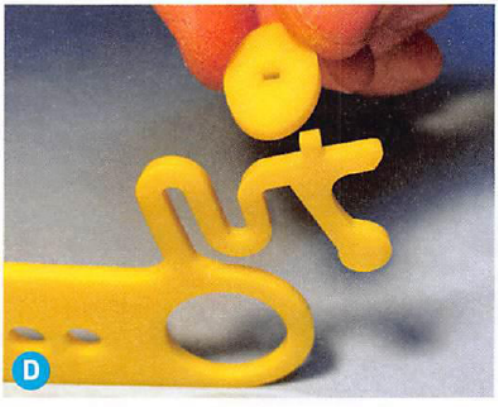




B



C



D

Snip three 1" straight pieces from a small paper clip for the pins. The polished 0.030" metal wire is super smooth for a low-friction axle and is easier to cut than steel rod.

2. ASSEMBLE

Stack the parts to make a three-layer sandwich. Use two pins to hold the small squares to either side of the main part. Use the third pin to hold the freely swinging pickle in the center. Add the small collars on the ends of the pins (Figure B). Glue the collars with a drop of cyanoacrylate, and then trim off the ends of the pins (Figure C).

Also glue the two landing pads (or "paddles") to the square pegs as shown in Figure D.

3. DECORATE

I did a quick paint-and-wipe to fill the etched scoring hole numbers (Figure E). Acrylic paint wipes off nicely and is easy to clean up with water.

Pickleball is sweeping the nation and getting court time can be tricky — so here's a pocket-sized version you can play anywhere, anytime. It's easy to make from laser-cut acrylic with a few bits of metal from a paper clip.

In a time-honored tradition of the toy business, this version is a knock-off of a cereal premium from over 40 years ago: Cap'n Crunch Thumb Tennis, produced by Leon Levy of Taico Design Products. The free-inside-the-cereal-box toy was molded in polypropylene, which is tough and flexible enough to work as springy snappers. Here's the original TV ad: youtu.be/8sZlMnVave0

This new version is made from brittle acrylic, but with a wave-shaped design for stress-free springiness. Instead of a ball, you flick a swinging pickle! Metal pins hold the parts together and serve as a low-friction axle.

TIME REQUIRED: 1 Hour

DIFFICULTY: Easy

COST: $2

MATERIALS

- **Acrylic sheet, ⅛" thick** All the parts can easily be cut from one 4"×8" piece of plex. Use separate, smaller pieces for different colors as shown.
- **Small paper clip**
- **Cyanoacrylate (CA) glue** aka super glue
- **Acrylic paint**

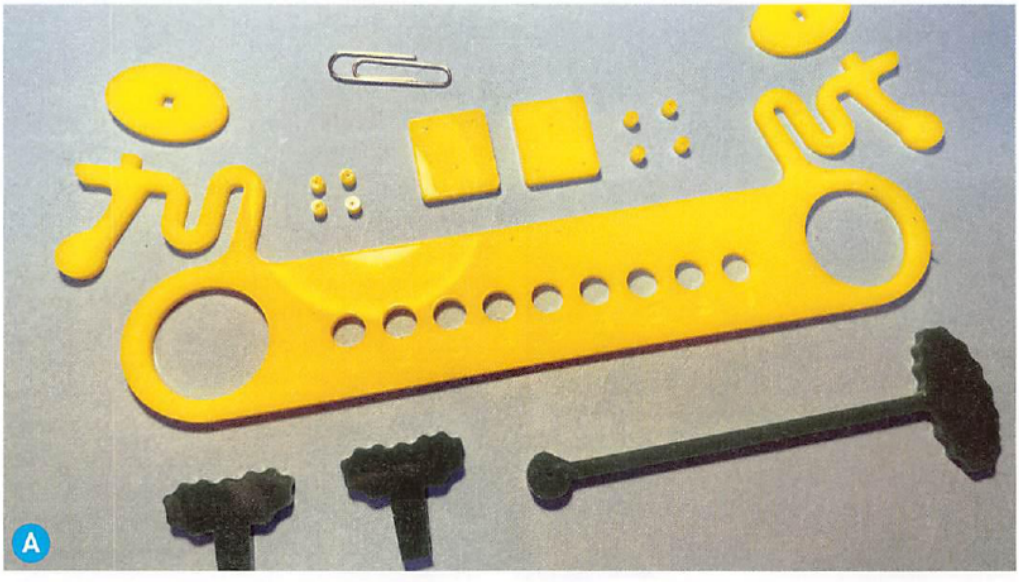
TOOLS

- **Laser cutter**
- **Wire cutters**



1. MAKE THE PARTS

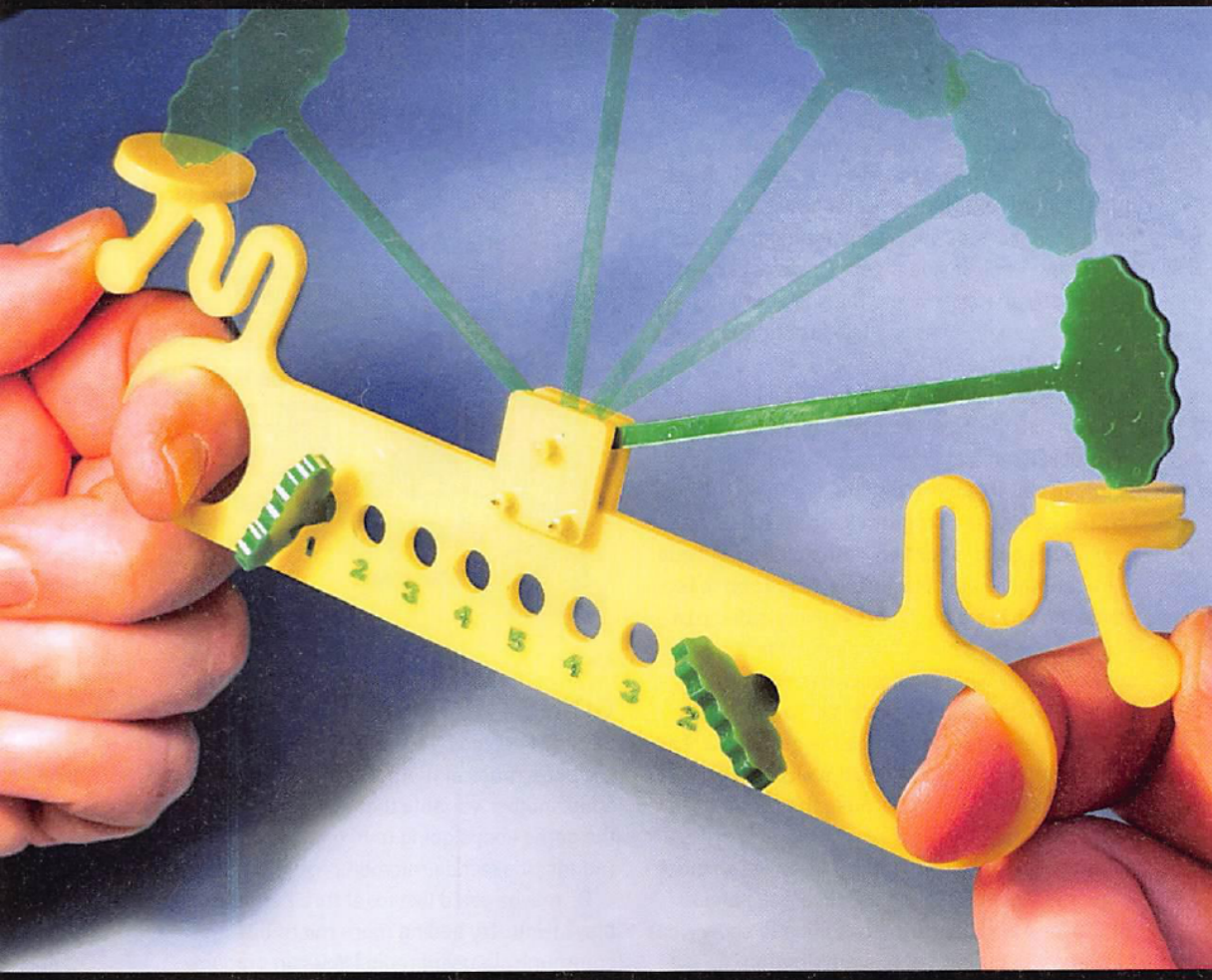
Go to makezine.com/go/pocket-pickleball to download the *.svg* files for cutting the parts. I used a Glowforge laser cutter, which automatically identifies "cut" and "etch" paths and also sets the proper speeds and power settings for the material. I used Glowforge Proof Grade materials in fun colors: yellow for the game parts and green for the pickle parts (Figure A). Adjust the settings for your particular laser cutter and materials.



Pocket Pickleball

Make this pocket-sized, two-player action game — that plays with a pickle instead of a ball!

Written and photographed by Bob Knetzger





BOB KNETZGER is a designer/inventor/musician whose award-winning toys have been featured on *The Tonight Show*, *Nightline*, and *Good Morning America*. He is the author of *Make: Fun!*, available at makershed.com and fine bookstores.

- If part of the code runs but not all of it, check it against Figure P to make sure everything is correct.

6. AMP UP YOUR GAME

The basic reaction test is OK, but it can be improved. For one thing, there's no way to prevent a false start if a player hits their button before the go signal. And if both players hit their buttons, both get a "win" message. So this Enhanced version of the game (Figure Q) waits until the go signal before registering a button press, and only counts the first time someone hits a button for each round. This game also resets automatically, so you don't have to restart the micro:bit to play again. You can open and edit the code at makecode.microbit.org/_P9xCM2TTrADy. Here's how to make it:

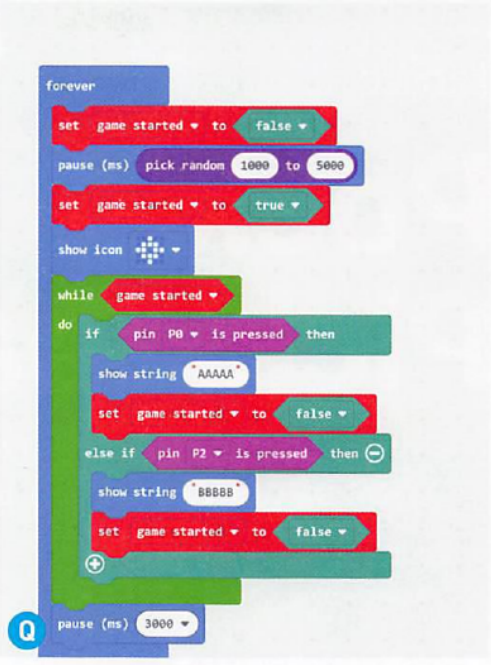
Move the **on start** code into the **forever** loop. Then create a **variable** — a block that can be set to different values — and give it a name like "game started." Insert a pointy-ended **true** block from Logic inside a **set [game started] to** block by snapping it into the space where a number normally goes. It's like turning the **game started** variable on.

Next, insert a **while** block from the Loops menu inside the **forever** block at the bottom of the stack. Drag and drop a pointy-ended **game started** variable block into the space on the **while** block so the program keeps repeating as long as the **game started** variable is set to **true**.

Put another Logic block called **if-then-else** inside the **while** loop. Insert a pointy-sided **pin [0] is pressed** block from Input into the **if** space to show what happens when Player A hit their button first. Underneath, add a "AAAAA" winning message and change the **game started** variable to **false**. This stops the **while** block from continuing to run, because the condition is no longer **true**.

Click the plus sign on the **if-then-else** block to get the **else if** option. Duplicate the blocks for Player A and adjust the condition to show what happens when Player B hits their button.

Now you have a game that lets players hit their buttons only after the starting image shows, and only displays a message for the person who wins the reaction race!

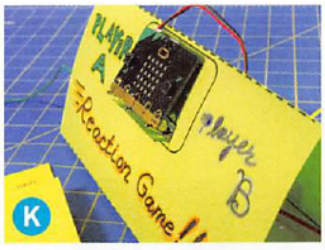




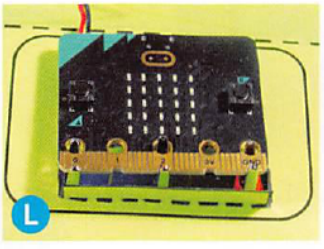
GOING FURTHER

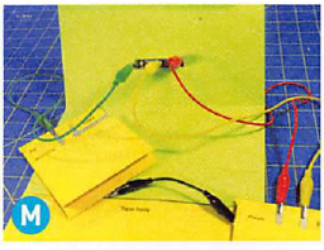
Want to keep going? This 3-Round version (Figure R) plays musical themes for each player and adds another variable that acts as a counter, so the game keeps going until one player wins three rounds: makecode.microbit.org/_aV35udi3CaMT

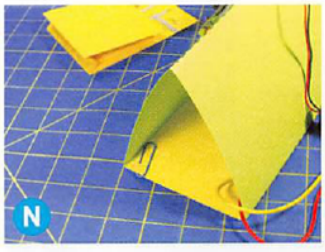
Or maybe you'd like to let three or four people play? (Hint: Try adding more micro:bits that communicate using radio.) You can probably find endless ways to make the Reaction Game your own. I hope it entices you to check out more games and fun in *Paper Inventions*, 2nd Edition. Enjoy!



K


L


M


N



O


P

Next, press the micro:bit board onto the display panel so it rests on the little shelf. Attach alligator clip wires to the proper connector holes on the micro:bit (Pins 0, 2, and GND) by poking them through the window in the game display panel from the back (Figures K and L). If possible, use different color wires to help make sure you are connecting the correct parts.

Connect three wires as shown in Figure M:

- The green wire connects Pin P0 to Player A's button where shown on the template.
- The yellow wire connects Pin P2 to Player B's button where shown.
- The red wire connects the GND pin to the top of the conductive tape strip labeled GND on Player B's button.

Now pull the buttons to the proper side and close the panel by clipping the side of the tent to the floor with paper clips (Figure N). Finally, connect the two buttons by attaching the black alligator clip to the GND conductive tape strips that go off to the side on each button (Figure O).

5. CODE THE BASIC GAME

The Reaction Game Test program (Figure P) displays a "go" signal at unexpected times, and scrolls text on the LED screen to show which player pressed their button first. These instructions will show you how to make it; you can also open and edit the code at makecode.microbit.org/_CcK423PL4Kdo. (If you're new to the micro:bit and MakeCode, go to microbit.org/get-started/getting-started/introduction for help getting started.)

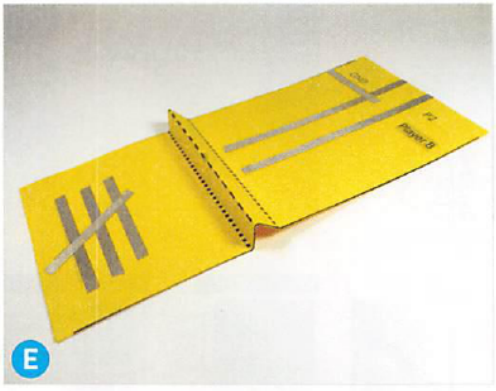
First, choose a **pause** block from Basic blocks and put it inside the **on start** block to make the micro:bit wait. Pause time is measured in milliseconds (written ms). Add a **pick random** block from Math to set the number of seconds between 1 and 5 (1000 to 5000 milliseconds). Use a **show icon** block from Basic to display an image on the LED grid that tells the players to go. (I chose a diamond-shaped target icon.)

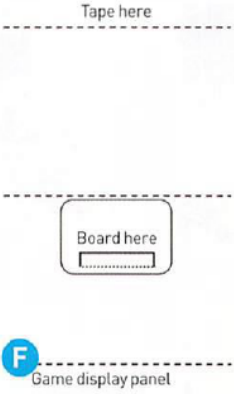
Next, get an **on pin pressed** block from Input to trigger a **show string** block from Basic that scrolls a series of characters. Leave pin number at 0. Type in the message "AAAAA" to show that Player A has won.

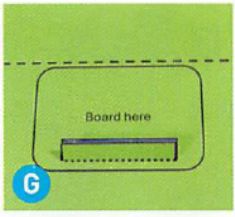
For Player B, duplicate the **on pin pressed** block, but this time change the pin number to 2 and the message to "BBBBB".

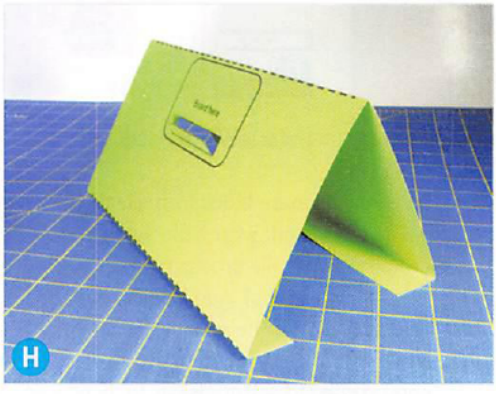
Download the code to the micro:bit board to check if everything is working. Does the icon show up when the program starts to run on the micro:bit? Do the messages scroll on the LED grid when you press one of the buttons? If not, try these troubleshooting tips:

- If none of the code runs, make sure the top conductive tape strips on your button connect with the strips on the bottom when the button is pressed.
- If that's not the problem, check that the right parts are connected by the alligator clip wires, and that the wires are clipped on securely.











more conductive material to grab onto.

On the top layer of each button, apply shorter strips of conductive tape and connect them with a cross piece, as shown in Figure E. Make sure the top and bottom line up so when you press the top down, the top strips connect the bottom strips (pin and ground) and close the circuit.

3: MAKE THE GAME DISPLAY PANEL

The second sheet of the template (Figure F) is the game display panel, which holds the micro:bit where the players can see and hear it, and helps keep the wires organized. To make it, first cut out and fold out the flap in the spot for the micro:bit, as shown on the template (Figure G). This creates a little shelf to hold the board, and a little window to let the alligator clips attach to the connector holes from the back of the board.

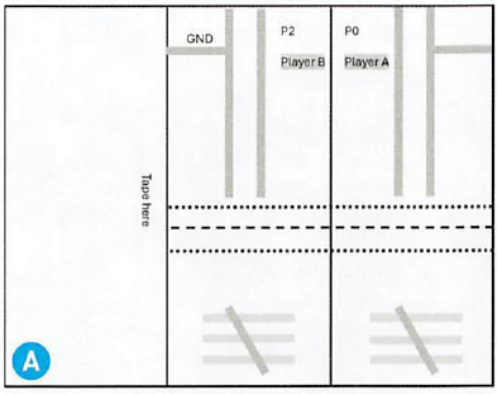
Next, fold the cardstock in half, short edges together. Fold the bottom edges up about 1". Then open the sheet partway to form a tent (Figure H). Fold both bottom flaps in.

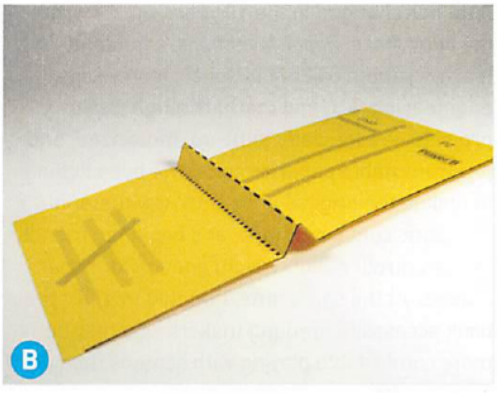
If you want to decorate your game display panel, flatten the paper again and do it now. You might want to label each player's side with its own color and personality. When you're done, tape the extra strip of cardstock from the button template to the back flap to make a "floor" for the tent (Figure I). Leave the other side loose for now.

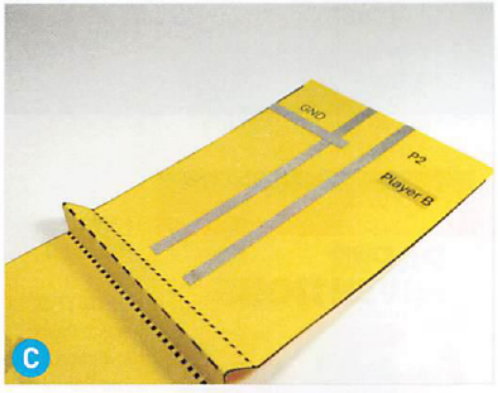
4. CONNECT THE MICRO:BIT

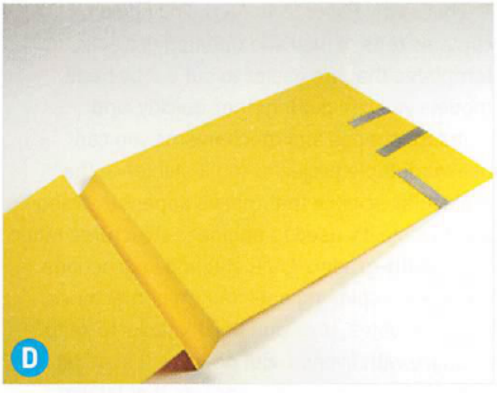
Roll up some loops of masking or duct tape, sticky side out, and press them onto the back of the micro:bit board (Figure J). Make them fat enough to stick out past the components.











the lines of conductive tape. When you press a button, it connects the lines of tape and closes the circuit. This allows some electricity to flow between the micro:bit pin that the button is attached to (pin P0, P1, or P2) and the Ground pin (GND). The micro:bit registers the increase in voltage as input — the same way it would if you pushed one of the on-board buttons — which triggers a bit of code that shows who hit their button first and won the round!

MAKE YOUR REACTION GAME

1. MAKE THE BUTTONS

Download the template from the project page at makezine.com/go/reaction-game and print it on two sheets of cardstock. Cut two buttons out of the sheet shown in Figure A. Save the blank section for later.

Fold along the broken lines — the line in the middle folds down (mountain fold) and the outer lines fold up (valley fold) to make the spring that holds the button open (Figure B).

2. ADD THE CIRCUIT

On the bottom layer of each button are markings for strips of conductive tape. Cover these lines with conductive tape (Figure C). One is marked with the pin number on the micro:bit that it connects to. The other strip connects to the ground pin (GND). The gap between them prevents short circuits. An extra piece of tape overlaps the GND line and goes off to the side of the button. This lets you connect the two buttons so they can use the same ground wire.

Where the tape gets to the edge of the button, bring the end over to the back for about half an inch (Figure D). This gives the alligator clips

Tape Tips

- Cut pieces on a slight angle.
- Use the pad of your finger to rub the end of the tape off the backing. Then fold the backing away from the tape.
- Press the sticky end of the tape onto the paper and slowly peel the backing away as you go along the line you're covering.

A lot has changed in the 10 years since I wrote my book *Make: Paper Inventions*, a guide to learning about STEAM (science, technology, engineering, art, and math) through paper projects — especially light-up, motorized, and programmable paper circuits! I've been itching to update the original book with new materials like fabric conductive tape, and beginner-friendly microcontrollers lightweight enough for paper models. At the same time, I wanted to make the book accessible to young makers who may be more comfortable playing with screens than with paper crafts.

That's why *Paper Inventions, 2nd Edition*, contains tons of new and updated projects, templates that are easier to cut out by hand, models you can put together quickly, and standard shapes and mechanisms you can use in multiple projects. You'll still learn the molecular science that makes paper so strong and flexible it's used to engineer structures taller than a fifth-grader. You'll still find instructions for making spinning entertainment machines, and wearables, toys, and knickknacks to keep or share with friends. But now you'll also get an intro to basic coding concepts that let you create projects like a motorized paper plate tilt ball maze and a light-up, pop-up card that plays "Happy Birthday" when you blow out the LED candles.

For a taste of what the new edition is like, here's some "bonus content" that didn't fit in the book: a programmable Reaction Game powered by a BBC micro:bit. It's fun to play and it works great with my middle-school-age students. A Reaction Game tests how fast players can hit a button when a signal is given, and there are a zillion variations. I'll show you how to build the button and display panel, then walk you through programming the simplest playable version with Microsoft MakeCode, and adding upgrades to make the game more challenging and more fun.

HOW THE BUTTONS WORK

The players' buttons in the Reaction Game are basically external on/off switches, with conductive tape serving as part of the wiring for the circuit. When the button is open, no electricity can flow through the circuit because of the gap between

TIME REQUIRED: 1–2 Hours

DIFFICULTY: Easy

COST: $25–$35

MATERIALS

- **Micro:bit board with USB to micro B cable** available in the Go Pack starter kit along with battery holder and batteries, which are not required for this project
- **Conductive fabric or copper tape**
- **Alligator clip wires (4)**
- **Cardstock, 8½"×11" (2 sheets)** or similar, with the template printed or copied onto them
- **Tape** You can use clear, masking, and/or duct tape.

TOOLS

- **Computer with printer**
- **Scissors**



NEWLY REVISED AND UPDATED, *Make: Paper Inventions*, 2nd Edition is available now at Maker Shed (makershed.com) and other booksellers.

Reaction Videos!

Here are a few of the Reaction Game tutorials that inspired my version. Thanks to Micro:bit Educational Foundation, Peli de Halleux of Microsoft, Brown Dog Gadgets, and other creators who generously shared their work online.

- microbit.org/projects/make-it-code-it/reaction-game
- makecode.microbit.org/projects/reaction-time
- youtube.com/watch?v=S_hzUenqAM0
- learn.browndoggadgets.com/Guide/Reaction+Game+(Paper+Version)/299

ELECTRONICS

Skill Tree: Color in the Boxes

Color in the boxes of anything you've completed.

1 tile = 1 point

Total Score

ADVANCED

BASIC

- Build a robot
- Make an Internet of Things (IoT) project
- Use renewable energy in a project
- Make a project with a camera
- Use an Arduino shield
- Learn to desolder a through-hole component
- Teach a friend an electronics skill
- Make a project that moves
- Fix something that's broken
- Measure voltage with a multimeter
- Use environmental sensors in a project
- Learn the difference between Arduino and Raspberry Pi
- Build your own enclosure for a project
- Follow a project tutorial
- Measure continuity with a multimeter
- Identify parts inside e-waste and their purpose
- Complete a soldering kit
- Use a servo motor in a project
- Use addressable RGB LEDs in a project
- Alter example Arduino code
- Take apart e-waste and see what's inside
- Learn to solder and soldering safety
- Use a button or a switch in a circuit
- Learn about batteries and battery safety
- Upload code to an Arduino or micro:bit
- Use Tinkercad to code and simulate an Arduino project
- Program a message to play on micro:bit LED matrix
- Learn to read a schematic
- Learn about Ohm's law
- Blink an LED with Arduino
- Learn about different electronics parts and what they do
- Learn about voltage, current, and resistance
- Learn to use wire strippers
- Make something with Arduino, micro:bit or a dev board
- Light up an LED with a battery
- Do a squishy circuits activity
- Do a paper circuits activity
- Learn to use a breadboard
- Complete an electronics kit
- Design your own invention idea

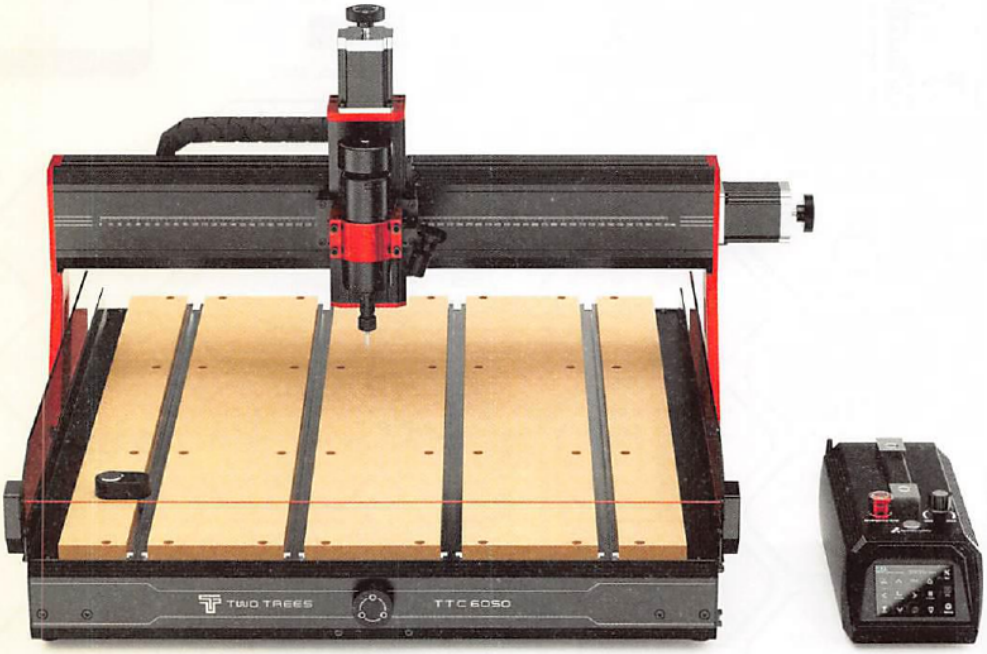
START HERE



STEPH PIPER is a creative technologist who is passionate about making it easier to learn new skills. She's a makerspace manager at a university library in Queensland, Australia, and has created electronics kits that are now sold globally under the brand name Maker Queen.

TOOLBOX

GADGETS AND GEAR FOR MAKERS Tell us about your faves: editor@makezine.com



TwoTrees TTC-6050 CNC Router

$1,799 twotrees3d.com

Nothing excites a maker like a new tool in the workshop, and that especially rings true for anybody fortunate enough to welcome the TwoTrees TT-C6050 CNC. The machine feels like a steal with a retail price of $1,799. You'll be hard pressed to find many competitors in the 2-foot size class under $2,000.

The TTC-6050 features a 600mm×500mm×100mm (23.6"×19.7"× 3.9") work envelope with a 500W spindle that is capable of around 12,000 RPM. The build quality is solid. Between the heavy-duty ball lead screws and NEMA 23 motors, this machine can move quickly and accurately with minimal backlash. The design hides the lead screws and linear rails, keeping most of the dust and debris at bay. For safety, there's an IR sensor that will stop the machine within 1 second of the operator reaching into the work envelope. Optionally, a variety of endmill bits come bundled with the machine so you're ready to start making things right out of the box.

The unboxing and build took me maybe 2 hours despite multiple distractions. Setup in VCarve was easy but required manual configuring — Vectric provides a library of machines online, but TwoTrees isn't even listed as a manufacturer option yet.

Pockets are easy and tested just fine, but what I really wanted to see carved was a relief. What happens when you give a 3D printing fan a CNC machine to play with? A wooden 2.5D Benchy of course! The carve took a long time, but I blame myself for setting speeds so slow.

The interface on the machine allows you to adjust the feed speed on the fly up to 200%. It's on the small size but was usable for my tests.

The TTC-6050 is a lot of machine for the price and performed well with the included endmills. If you don't have dust collection already, the TwoTrees CNC Vacuum Cleaner Monster (M1) works well, and I prefer the light on the provided boot over the light provided with the CNC. Overall, it's a great machine for new and experienced users alike. *—Dom Dominici*



Make It Yourself, by Node

$0 makeityourself.org

Every once in a while someone comes along and creates something that should have already existed, but no one else realized it. Hacker/curator Node put together this incredible PDF encyclopedia of 1,000+ maker projects in late 2024. Each piece is rendered in beautifully minimalist linework, equalizing the presentation of builds from hundreds of creators. Although an all-encompassing tome would be impossible, the range is exceptional, covering everything from electronics to furniture to fashion.

Though it'd be at home on a coffee table, it's an e-book for a reason — each image is a link to the project's site, so you can learn to build all 1,000+ projects yourself. Maybe it's the implied simplicity that gives the reader a strong sense of I-want-to-make-that, but I've found it a good tool for ramping up my motivation. I wouldn't be surprised by a premium charge to cover the effort behind it, yet for reasons I don't understand (but still appreciate), it's free. *—Sam Freeman*



Microjig Grr-Ripper 2 Go

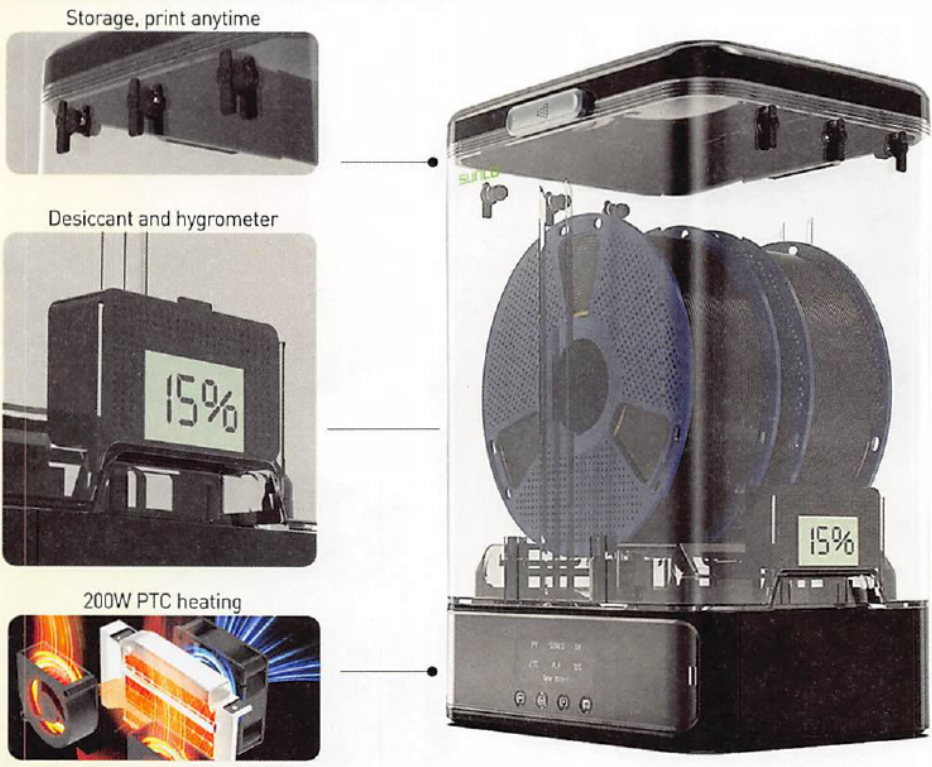
$30 microjig.com

If you have a table saw, you need a push block. It can be as simple as a scrap stick or as fancy as the Grr-Ripper Pro that's all over YouTube. There's a middle ground in the Grr-Ripper 2 Go.

It's magically grippy, far more than my generic store-bought block or homemade plywood board. And compared to a narrow stick, it's easier to push inward and down with one hand.

The 2 Go handles narrow cuts down to 5/16" as long as you don't use a blade guard. I wish it came with a heel on the back, but there are slots to add your own. I appreciate how the high handle keeps my hand far from the blade.

If I could only have one pusher, I'd stick with a homemade rabbit. But I'm a grown-up who's allowed to own multiple tools, and for the right jobs the Grr-Ripper 2 Go feels safer. It doesn't make me a better woodworker, but it does help keep my cuts steady. *—Sam Freeman*



Sunlu SP2 Filament Dryer

$110 sunlu.com

If you live in a humid region, all that moisture might be playing havoc with your filament. It's time for a filament dryer. Filament dryers aren't anything new, but I think the SP2 solves one of the big problems found with most other dryers on the market: what do you do with your filament after you've dried it?

The SP2 comes in two main parts: the dryer base and the storage top box. There are multiple ways to hold the filament in the box, and it can hold two 1kg spools or one 3kg spool. When the base is turned on, the user can select what type of filament they want to dry from a list of popular types. If the default settings don't work for your needs, you can adjust the temperature and drying time. When the job is done, the filament can be fed directly from the dryer into the printer.

With most other filament dryers, the fun ends there. When you want to store the filament or switch to others, you must remove them from the dryer. With the SP2, the storage box can be removed and covers can be placed on, sealing the box from air and moisture. Another box can then be placed onto the dryer base, and another drying job can be run. If you have a bunch of spools you want to keep dry, you don't need to keep buying the expensive electronics over and over, you just buy new storage cases — brilliant. Each case also comes with a built-in hygrometer so you can easily make sure you are keeping it at a proper moisture level.

I think the only thing missing now is for one of you to come up with some kind of cool wall bracket to store the cases. I can't wait to see what you all come up with for this one. *—Matt Stultz*

Make:

THE BEST MAGAZINE FOR **MAKERS, DIY ENTHUSIASTS, HACKERS,** AND **TINKERERS.** SUBSCRIBE NOW!

makezine.com/subscribe



Grip Blades™

MULTI-USE UTILITY BLADE.

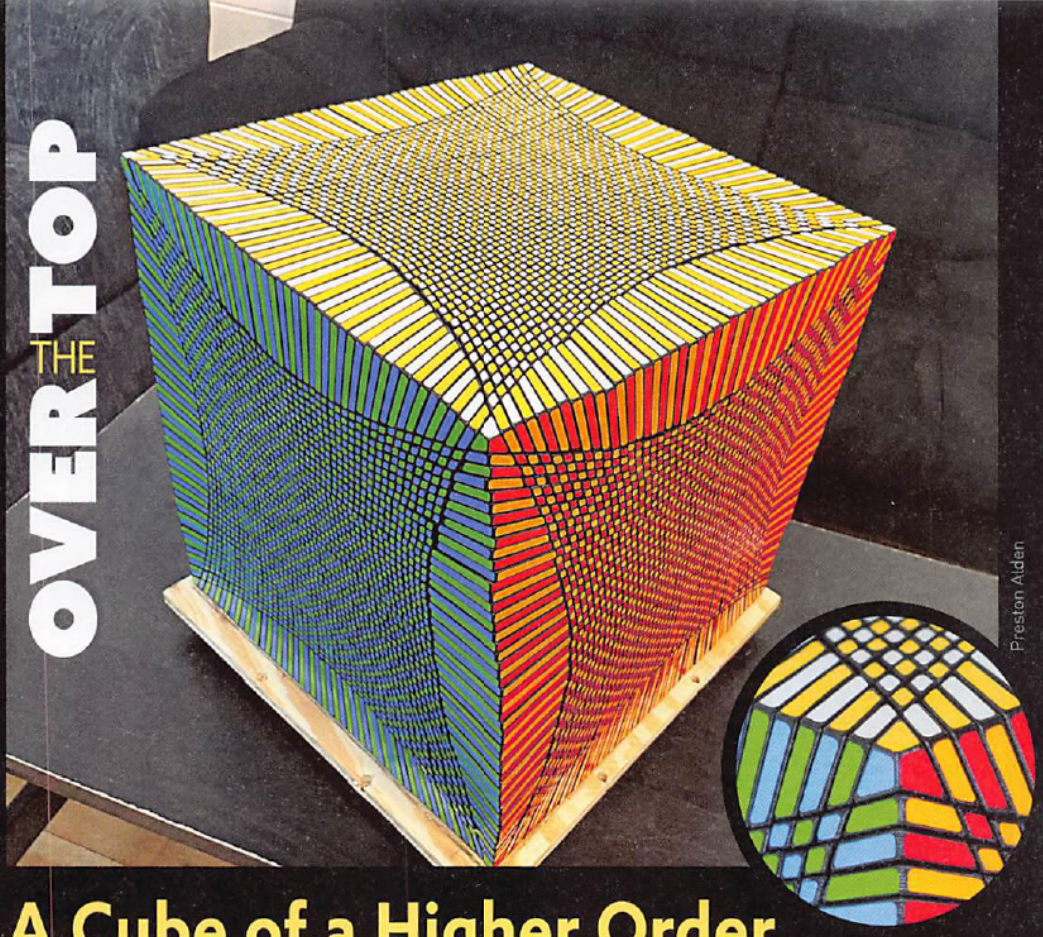
• STRENGTH
• COMFORT
• PRECISION

GET A GRIP!
GripBlades.com



INFO DEMO BUY!

STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION (required by Act of August 12, 1970: Section 3685, Title 39, United States Code). 1. MAKE Magazine 2. (ISSN: 1556-2336) 3. Filing date: 10/1/2025. 4. Issue frequency: Quarterly. 5. Number of issues published annually:4. 6. The annual subscription price is 34.99. 7. Complete mailing address of known office of publication: Make Community, LLC 150 Todd Road Ste. 100, Santa Rosa, CA 95407. 8. Complete mailing address of headquarters or general business office of publisher: Make Community, LLC 150 Todd Road Ste. 100, Santa Rosa, CA 95407. 9. Full names and complete mailing addresses of publisher, editor, and managing editor. Publisher, Dale Dougherty , Make Community, LLC, 150 Todd Road Ste. 100, Editor, Keith Hammond, Make Community, LLC, 150 Todd Road Ste. 100, Santa Rosa, CA 95407, Managing Editor, N/A, Make Community, LLC, 150 Todd Road Ste. 100, Santa Rosa, CA 95407. 10. Owner: Make Community, LLC; 150 Todd Road Ste. 100, Santa Rosa, CA 95407. 11. Known bondholders, mortgages, and other security holders owning or holding 1 percent of more of total amount of bonds, mortgages or other securities: None. 12. Tax status: Has Not Changed During Preceding 12 Months. 13. Publisher title: MAKE Magazine. 14. Issue date for circulation data below: Fall 2025. 15. The extent and nature of circulation: A. Total number of copies printed (Net press run). Average number of copies each issue during preceding 12 months:44,858. Actual number of copies of single issue published nearest to filing date: 45,285. B. Paid circulation. 1. Mailed outside-county paid subscriptions. Average number of copies each issue during the preceding 12 months: 30,911. Actual number of copies of single issue published nearest to filing date: 30,297. 2. Mailed in-county paid subscriptions. Average number of copies each issue during the preceding 12 months: 0. Actual number of copies of single issue published nearest to filing date:0. 3. Sales through dealers and carriers, street vendors and counter sales. Average number of copies each issue during the preceding 12 months: 4,070. Actual number of copies of single issue published nearest to filing date: 3,700. 4. Paid distribution through other classes mailed through the USPS. Average number of copies each issue dur ng the preceding 12 months: 0 . Actual number of copies of single issue published nearest to filing date: 0. C. Total paid distribution. Average number of copies each issue during preceding 12 months: 34,981. Actual number of copies of single issue published nearest to filing date;33,997. D. Free or nominal rate distribution (by mail and outside mail). 1. Free or nominal Outside-County. Average number of copies each issue during the preceding 12 months:751. Number of copies of single issue published nearest to filing date: 793 . 2. Free or nominal rate in-county copies. Average number of copies each issue during the preceding 12 months: 0 . Number of copies of single issue published nearest to filing date: 0. 3. Free or nominal rate copies mailed at other Classes through the USPS. Average number of copies each issue during preceding 12 months: 0. Number of copies of single issue published nearest to filing date: 0 . 4. Free or nominal rate distribution outside the mail. Average number of copies each issue during preceding 12 months: 680 . Number of copies of single issue published nearest to filing date: 911 . E. Total free or nominal rate distribution. Average number of copies each issue during preceding 12 months: 1,431 . Actual number of copies of single issue published nearest to filing date: 1,704 . F. Total free distribution (sum of 15c and 15e). Average number of copies each issue during preceding 12 months: 36,412 . Actual number of copies of single issue published nearest to filing date: 35,701 . G. Copies not Distributed. Average number of copies each issue during preceding 12 months: 8,446 . Actual number of copies of single issue published nearest to filing date: 9,584 . H. Total (sum of 15f and 15g). Average number of copies each issue during preceding 12 months: 44,858 . Actual number of copies of single issue published nearest to filing: 45,285. I. Percent paid. Average percent of copies paid for the preceding 12 months: 96.07% Actual percent of copies paid for the preceding 12 months: 95.23% 16. Electronic Copy Circulation: A. Paid Electronic Copies. Average number of copies each issue during preceding 12 months: 9,823. Actual number of copies of single issue published nearest to filing date: 9,765. B. Total Paid Print Copies (Line 15c) + Paid Electronic Copies (Line 16a). Average number of copies each issue during preceding 12 months: 44,804. Actual number of copies of single issue published nearest to filing date: 43,762. C. Total Print Distribution (Line 15f) + Paid Electronic Copies (Line 16a). Average number of copies each issue during preceding 12 months: 46,235. Actual number of copies of single issue published nearest to filing date: 45,466. D. Percent Paid (Both Print & Electronic Copies) (16b divided by 16c x 100). Average percent of copies paid during preceding 12 months: 96.90%. Actual percentage of copies paid for single issue published nearest to filing date: 96.25%. I certify that 50% of all distributed copies (electronic and print) are paid above nominal price: Yes. Report circulation on PS Form 3526-X worksheet 17. Publication of statement of ownership will be printed in the Winter 2025 issue of the publication. 18. Signature and title of editor, publisher, business manager, or owner: Todd Sotkiewicz - Business Manager. I certify that all information furnished on this form is true and complete. I understand that anyone who furnishes false or misleading information on this form or who omits material or information requested on the form may be subject to criminal sanction and civil actions.



Preston Alden

A Cube of a Higher Order

The Rubik's Cube was invented in 1974 by Hungarian artist Erno Rubik with its now familiar 3×3×3 structure, but with Péter Sebestény's 1981 4×4×4 remix, Rubik's Revenge, the arms race for more complicated puzzles began. Cut to late 2024, when **Preston Alden** announced he had broken the world record for building the "highest order *n*×*n*×*n* twisty puzzle," which is a fancy way of saying he made the most complex Rubik's Cube on the planet.

At 49×49×49 moving pieces, Alden surpassed Grégoire Pfennig's previous record of 33×33×33 by a wide margin. Alden's cube is over 13 inches tall and weighs 65 pounds. It has total of 13,827 pieces 3D printed in PETG and a whopping 14,406 colored stickers superglued by hand (to make sure they *never* peel off). That's twice as many pieces and stickers than the previous record.

Alden designed the cube first in Fusion360 and later in Solidworks. He started with a blank cube shape and extruded a series of cut lines to form cut planes. Using those planes, he could cut out all the pieces at once. He had four 3D printers running 24/7 for 3 months to print them all out. Alden estimates it took 3,000 hours to build the cube, and roughly $3,800 for materials and tools. Sanding was the hardest part: "This step was intense and required more dedication than I had given to pretty much anything else in my life up until that point," Alden said in his announcement.

Alden caught the Rubik's Cube bug around age 12 and from then on kept looking for new and interesting "twisty puzzles," the general term for such puzzles. They can be cubes, pyramids, circles, and more. Alden has about 120 puzzles currently in his collection.

But how big can these cubes really get? Weight is a big factor: "I suspect that a 100×100×100 probably going to be done by someone," Alden predicts, "and it's going to weigh as much as a small car — and cost as much as a car."

—Craig Couden

For a video overview and build process, visit makezine.com/go/49x49x49.

PCBWay

FULL-SERVICE ELECTRONICS MANUFACTURER

PCB Fabrication / CNC | 3D Printing / PCB Assembly / OEM | EMS



CUSTOMIZED TECHNICAL PARTS & PCB MANUFACTURED

WHY CHOOSE PCBWAY?

PCBWay offers a wide range of services including PCB fabrication, PCB assembly and even CNC machining for over a decade, and has earned a distinguished reputation globally in the industry.

Hassle-free ordering

The digital quote-to-order platform puts you in the back seat. Upload a Gerber file and receive feedback soon.

On-time shipping

We maintain a 99% on-time delivery rate, working in three shifts to ensure your packages arrive fast.

Quality assurance

Our technicians have been working strictly to high standards. All the boards will go through the most stringent tests.

Customer support

The customer service teams work inshifts to provide 24-hour support. You can always contact a live customer service person.

Scan the QR code

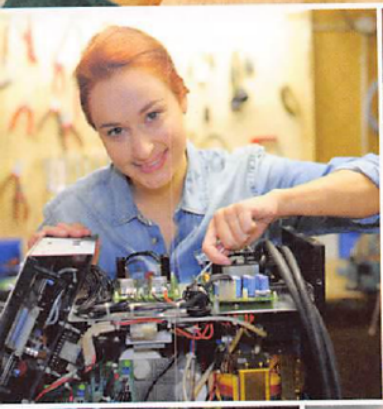
CONTACT US:

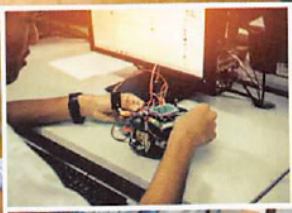
www.pcbway.com

service@pcbway.com

From Garage Projects to Global Innovations: We're Makers at Heart













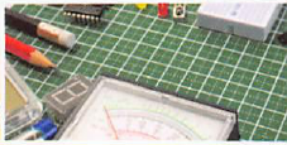












MICROCHIP
microchip.com
The Microchip name and logo and the