

BuddyBot: Controlling a Stepper Motor with an Android Smartphone

By Michael Parks, P.E., for Mouser Electronics

[Licensed under CC BY-SA 4.0](#)

Summary

Sometimes you have an idea on how to solve a problem and look for the electronics to make it happen. Sometimes you have the electronics and must manufacture a problem to solve. This open source project is in the latter category and uses a [stepper motor](#) and the [STM32 Nucleo](#) to have a little fun with our four legged companions. Chasing the infamous red dot of a laser pointer is a favorite hobby for our family's feline friend.

We're engineers and makers, so just waving around a laser pointer with our hand isn't going to be enough; we must add a little electronic magic... This device will let the user control the laser pointer remotely via an Android app. We'll also add an automated "robot mode" to keep them entertained automatically. The schematic is provided in both .PDF and .SCH on the Source Files page.

Of course, this design can be easily hacked for other projects that have similar requirements to:

1. Move a stepper motor remotely using an Android device as a remote control via Bluetooth.
2. Move a stepper motor remotely at random.
3. Use hardware interrupts.
4. Practice basic circuit concepts such as the voltage divider and pull-up resistors.
5. Use additive manufacturing techniques to build the "mechanical" portions of your electronics project.

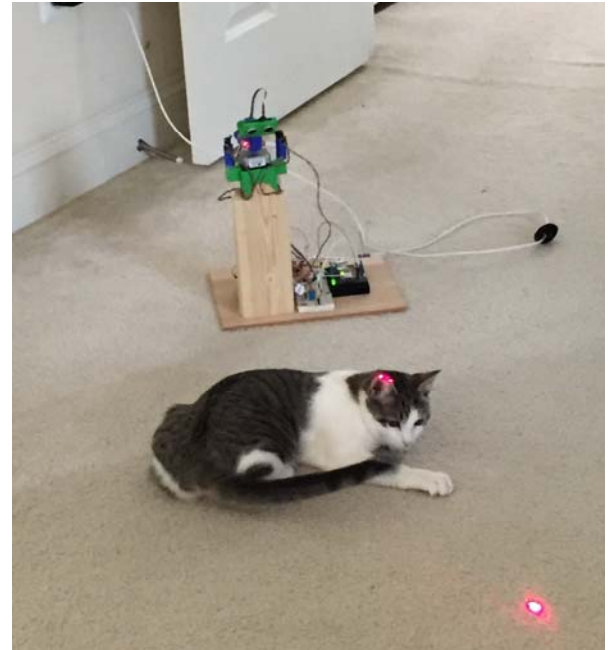


Figure 1: BuddyBot is an Android device-controlled laser pointer mounted on a stepper motor using a \$10 STM32 Nucleo. Actual Bluetooth range is around 15 - 30 ft.

Materials Needed

The Bill of Materials (BOM) for this project is for the electronics, but a bit more is needed build a custom enclosure. If you have access to a 3D printer, you can use the provided **STL files** to print the mechanical hardware holding this project together. Of course, this is not an absolute requirement and you can build the mounting hardware with whatever building materials you have at hand. An Android device running at least the Jelly Bean version is also a requirement if you want to run the provided app.

To make things convenient we have provided a pre-built shopping-cart-full-of-BOM (<https://www.mouser.com/ProjectManager/ProjectDetail.aspx?AccessID=3137ce43e6>) with the parts you will need to get this project built, assuming you already have tools, [heat shrink](#), and [hook-up wire](#) on hand.

1. The electronic components recommended include:

Table 1: Bill of Materials for Buddy Bot

QTY	Mouser Part #	Description
1	511-STM32 NUCLEO-F401RE	STM F401RE STM32 Nucleo
1	834-HT17-268D	834-HT17-268D NEMA 17 stepper motor
1	511-L293D	STM L293D Dual H-Bridge motor driver
1	765-RN42XVP-I/RM	765-RN42XVP-I/RM Bluetooth module
1	647-UVZ2A0R1MDD	0.1uF capacitor
1	647-UVR1H100MDD1TA	10 uF capacitor
1	511-L7805CV	511-L7805CV 5V regulator
4	71-CMF5510K000FKEB	10k-ohm resistors
1	71-RN60D-F-5K	5k-ohm resistor
1	782-A000021	Xbee shield
1	619-28015	Ping ultrasonic sensor
2	785-ZM90G20F01	Micro Lever Switches
1	571-826646-2	Male Header Pins (3-wide)
1	571-2828362	Screw Terminal Block
1		Laser Pointer
Various		Heat shrink
Various		Hook-up wire

2. Tools:

As for tools, you can make do if you have at least these, or add items to the project BOM cart:

- [Wire strippers](#)
- [Digital multimeter](#)
- [Small screwdriver](#) - for tightening down terminal screw blocks, these will do just fine.
- [Soldering iron](#)
- [60/40 solder](#)
- [Heat gun](#)
- [Chip inserter](#) (optional)

Mouser carries the suggested tools and supplies mentioned above. The only exception being a red laser pointer, but you can find one by searching on laser and diode that has a forward voltage of 3V or 5V.

IMPORTANT:

Before we start, please be sure to follow [these instructions](#) to update the firmware of your STM32 Nucleo before you start! Older versions of the firmware do not allow for interrupts to be attached to all of the STM32 Nucleo GPIO pins. That has been remedied in the latest firmware version, and it will be critical for our project.

Overview

- **The Brains:** We will use an [STM32 Nucleo F401RE microcontroller platform](#) to tie all our components together. The STM32 Nucleo is programmed using the C/C++ language.

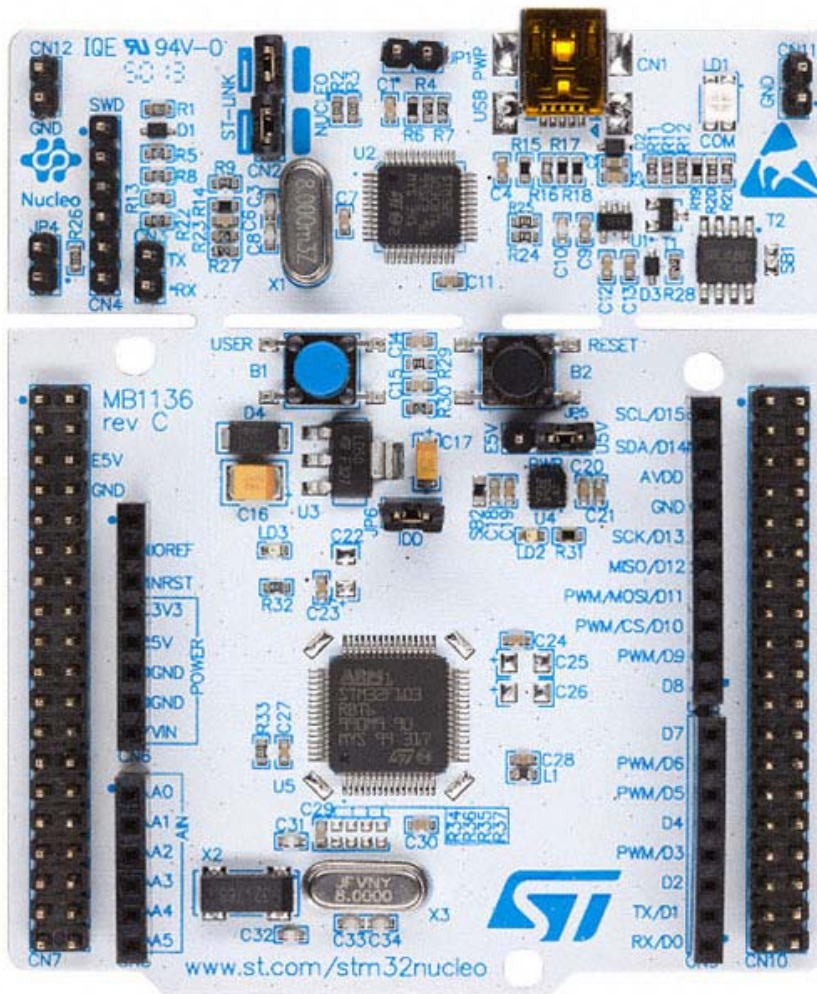


Figure 2: The STM32 Nucleo has a detachable, reusable programming card at the top that is removed once a project is ready for final deployment. <STM32Nucleo-F401RE.jpg>

You will need a free account on [mbed.org](#) to use the STM32 Nucleo since their [Integrated Development Environment](#) (IDE) lives in the cloud. The benefits of having a cloud-based IDE are that:

- you are always running the latest IDE
- your code is backed up on their servers
- it makes adding third-party libraries a breeze.

We will use the latter feature to add a pre-built library to interface with our ultrasonic sensor so we can enable an automatic “robot” mode for our BuddyBot. I have also provided a link to the STM32 Nucleo code [here](#) to help get you started.

- **The Brawn:** A NEMA 17 [stepper motor](#) will provide the rotation of our laser and ultrasonic sensor. We will also use a [L293D dual H-Bridge motor driver](#) microchip to provide the interface and some protection between our microcontroller and the stepper. Important note: the 'D' at the end of L293D signifies that it has the built-in flyback protection diodes. There are also L293 chips that do not have the diodes, so be sure to pick the right ones when ordering if you don't use the ones in the [project cart](#) already. The diodes are key in preventing blowback voltage from the motor coils from frying your microcontroller. Blowback voltage results when the power to the relay coil gets switched off and the magnetic field starts to fade. Current can flow back into your microcontroller if you leave it unprotected.
- **Cutting the Wires:** We will use a [RN42 Bluetooth board](#) to provide the serial communications link between our STM32 Nucleo and our Android app. To make things easy, we will be using [MIT's App Inventor](#) software to develop our app. In addition, I have provided the [.aia file](#) <LINK to Source files page > if you want to upload it into your App Inventor account and tinker with the code. Or if you just want the app, Green Shoe Garage has published it [in the Google Play Store](#).

The Build

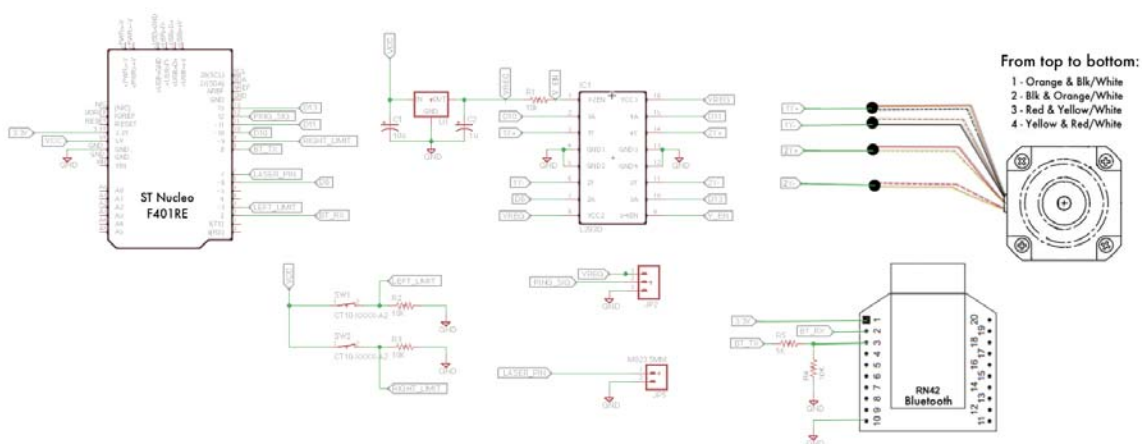


Figure 3: The BuddyBot Schematic. Click to enlarge. [Source files](#) provided. < BuddyBotSchematic.png>

1. To begin, partition your breadboard into different parts. You will need space on the breadboard for the following components:
 - L293D stepper motor interface chip and all the interface wires
 - Terminal screw blocks for the laser
 - 3 male header pins for interfacing to the ultrasonic sensor
 - Voltage divider circuit to interface the STM32 Nucleo with the Bluetooth board
 - Pull-down resistors for the motor limit switches
 - Voltage regulator circuit
2. Let's start with the [L293D H-Bridge Motor Driver Chip](#):
 - a) Gently secure the L293D chip into the breadboard to avoid damaging the pins.
 - b) Hook up the ground rail (GND) to the pins that need a connection to GND; this includes pins 4, 5, 12, and 13.
 - c) Run a 10k-ohm resistor for the Vcc rail to pin 1 which is the enable pin for the chip.

- d) Run a wire from pin 1 to pin 9, which is also an enable pin.
- e) Run Vcc directly into pins 8 and 16 to provide the power to drive the motors.
- f) If we had a large load, we would have to use an external power source for these pins. But since the load is small, we can drive it off the 5V from the STM32 Nucleo with no problem. We will revisit this when we wire it up to the motor and the STM32 Nucleo later on.

3. Hooking up the **laser**:

- a) If needed, extend the wires of the laser by soldering an additional length of wire. Be sure to insulate the solder point with electric tape or heat shrink.
- b) Insert the screw terminal block onto the breadboard.
- c) Connect one terminal of the screw terminal to GND.
- d) Connect the GND wire of the laser to the GND terminal of the screw block.
- e) Connect the Vcc wire of the laser to the other terminal of the screw block.
- f) Connect a wire from the STM32 Nucleo pin PA_8 to the same row on the breadboard as the Vcc laser terminal.
- g) This allows us to toggle the laser on and off in the software we will write later.

4. Hook up the **PING))) ultrasonic sensor**:

- a) Place a strip of 3 male header pins on the breadboard.
- b) Then we provide a connection to Vcc and GND to the header pins.
- c) The signal output pin of the ultrasonic sensor can be ran from the header pin directly into the STM32 Nucleo's PA_6 GPIO pin. We will use the pulseIn function of the STM32 Nucleo to read in the ultrasonic sensor signal.
- d) Ensure that you match the pins on breadboard to the pins on the PING sensor. Looking at the front of the PING sensor the order of the wires is SIG, Vcc, GND.

5. **Limit Switches**: To prevent the motor from turning a full 360-degrees and ripping out the laser and ultrasonic sensor wires, we will need to include two limit switches. Use the schematic to refer to, as well as instructions below. You also have an illustrated breadboard with which to follow along. Let's wire the limit switches up next:

- a) First let's solder some pretty long wires to our micro lever switches.
- b) Using a multimeter, check to see the [normally open \(N.O.\) position for your switches](#). Solder a wire to each of the terminals. I prefer to use a red and green color wire to differentiate terminals.
- c) We can't let our STM32 Nucleo [pins float](#) when the switches aren't depressed, so we will need 10k-ohm pull-down resistors for each switch. Repeat steps 1) through 3) below for both limit switches:
 - 1) Connect one end of the 10k-ohm resistor to GND and the other end into a row on the breadboard.
 - 2) Connect the red wire of the limit switch into the Vcc rail of the breadboard (the row marked +).
 - 3) Connect the green wire of the limit switch to the same row on the breadboard as the 10k-ohm resistor.
- d) Connect the left limit switch to STM32 Nucleo pin **PB_3** by tapping the point between our left limit switch and its 10k-ohm resistor.
- e) The right limit switch will connect to pin STM32 Nucleo **PC_7** by tapping the point between our right limit switch and its 10k-ohm resistor.

6. **Voltage regulator:** To ensure clean power on the breadboard we have a 7805 voltage regulator and some capacitors to filter out any noise. Wire the components as such:
 - 1) Place the 7805 into the breadboard.
 - 2) Connect the positive lead of the 10uF capacitor to Vin pin of the 7805.
 - 3) Connect the negative lead to the GND pin.
 - 4) Connect the positive lead of the .1uF capacitor to Vout pin of the 7805.
 - 5) Connect the negative lead to the GND pin.
 - 6) Connect the Vout pin of the 7805 to the Vcc rail of the breadboard.
 - 7) Connect the 5V pin of the STM32 Nucleo to the Vin pin of the 7805.
 - 8) Connect a wire from the GND pin of the 7805 to the GND rail of the breadboard.
7. **STM32 Nucleo:** Let's next connect the STM32 Nucleo to the RN42 Bluetooth module. Most important in this step is to create a voltage divider circuit so that the STM32 Nucleo can safely talk to the RN42 Bluetooth module:
 - a) The STM32 Nucleo outputs 5V for logic high. However, the Bluetooth board can only tolerate up to 3.3V. A simple way to resolve this mismatch is to make a voltage divider circuit using a 5k-ohm and 10k-ohm resistor.
 - b) Ensure that one end of the 5k-ohm resistor connects to pin **PA_9** on our STM32 Nucleo.
 - c) One end of the 10k-ohm resistor is connected to ground.
 - d) The remaining ends of each resistor will connect at a common node on the breadboard. We will tap off this node and run a wire to the Bluetooth **TX pin**, which is **pin 3** on the RN42 breakout board.
 - e) The Bluetooth module RX pin (RN42, **pin 2**) can safely connect directly to the STM32 Nucleo on pin **PA_10**.
 - f) Connect the RN42 **pin 1** to the STM32 Nucleo's 3.3V pin.
 - g) Connect the RN42 GND pin to the breadboard's GND rail.

When you are done your breadboard should look like this:

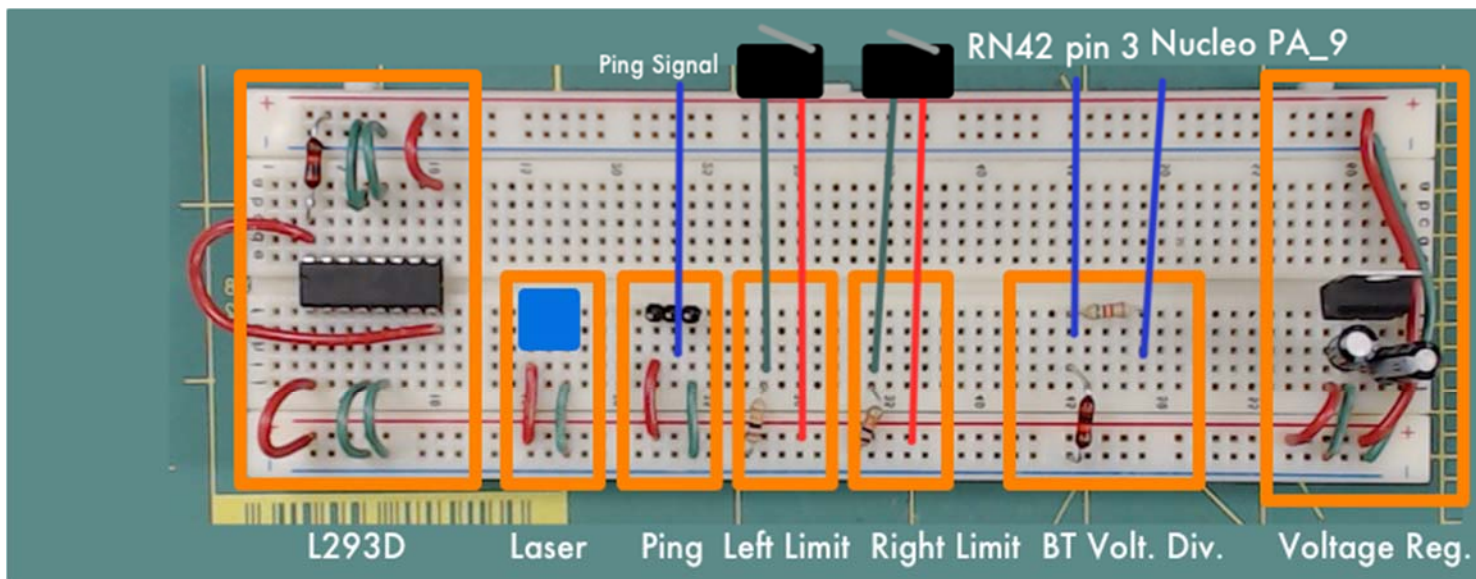
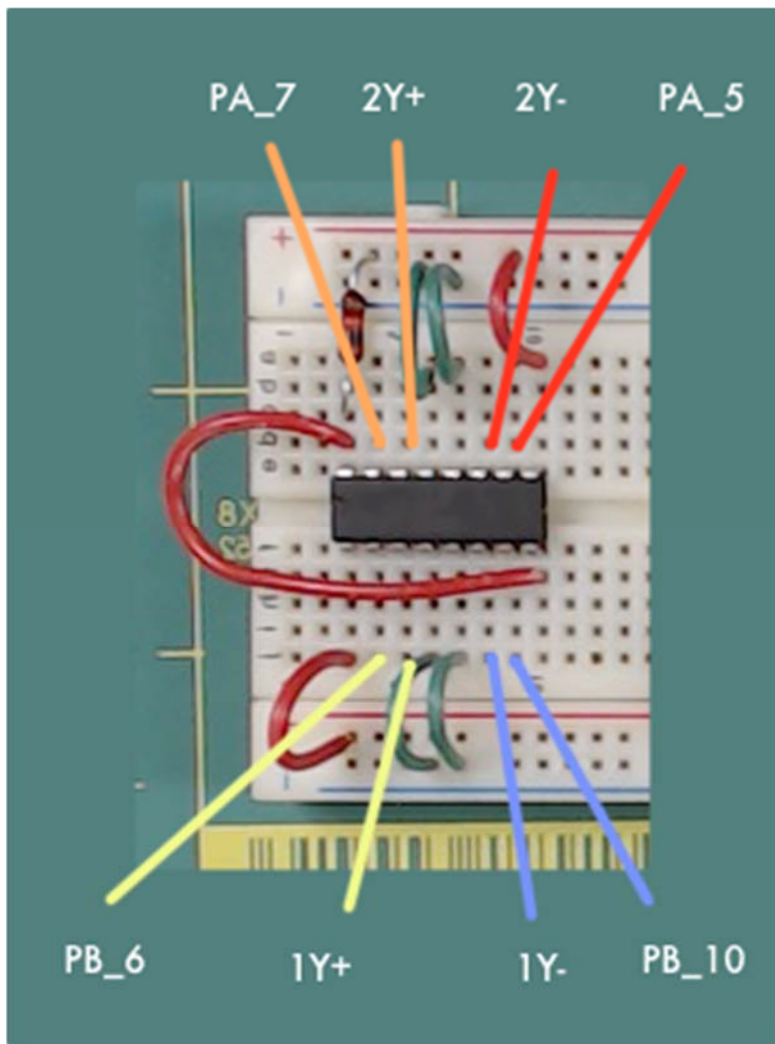


Figure 4: Illustrated BuddyBot Breadboard

8. Now that the breadboard is wired for our components, we can go back in and add the wires that **connect the STM32 Nucleo to the L293D** and the **L293D to the stepper motor**. I chose to wire the motors last since there tends to be a lot of long wires involved and this prevents confusion when wiring up the earlier components.
- a) The particular stepper motors used here happen to split each phase into two separate coils (thus there are four coils in total). For this particular application, we will keep things simple by soldering together the 8 motor wires into 4 pairs of two wires. Specifically, we will solder together the following 8 wires coming from the stepper motor into these 4 pairs:
 - 1) Wire 1Y+ : Solid Orange wire with the Black/White wire
 - 2) Wire 1Y- : Solid Black wire with the Orange/White wire
 - 3) Wire 2Y+ : Solid Red wire with the Yellow/White wire
 - 4) Wire 2Y- : Solid Yellow wire with the Red/White wire
 - b) Next we will hook up these wires to our L293D in the following way:
 - 1) Wire 1Y+ connects to L293D pin 3
 - 2) Wire 1Y- connects to L293D pin 6
 - 3) Wire 2Y+ connects to L293D pin 14
 - 4) Wire 2Y- connects to L293D pin 11
 - c) Last, we need to connect the STM32 Nucleo to the L293D in the following way using some hook up wire:
 - 1) PB_6 to L293D pin 2
 - 2) PB_10 to L293D pin 7
 - 3) PA_7 to L293D pin 15
 - 4) PA_5 to L293D pin 10

This is what the wiring to L293D should look like (Figure 5b below):



That wraps up the hardware portion of the project. Go ahead and plug-in the USB cable to the STM32 Nucleo and see if the LEDs on the STM32 Nucleo and the RN42 Bluetooth module light up. If the little lights are blinking, then it's time to move onto the software part of the project.

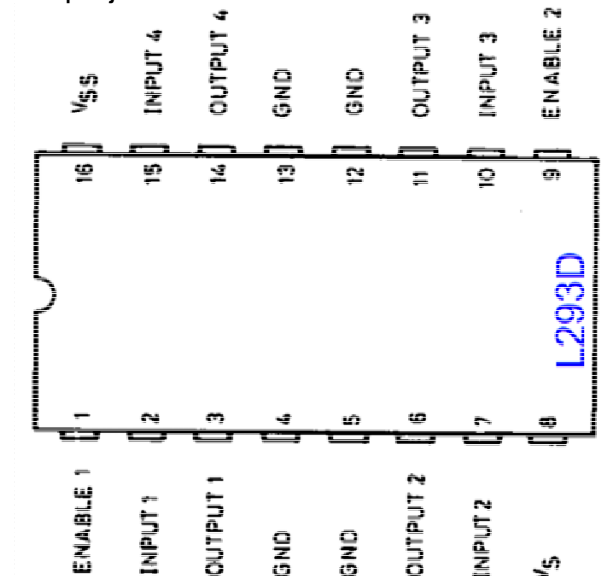


Figure 5a: L293D Pinout <L293D

pinout.jpg>

Figure 5b: Connect the motor and the STM32 Nucleo to the L293D.

Click on image to download. <Breadboard_L293D.png>

Software

The BuddyBot project has two software components:

- Code that runs on the STM32 Nucleo and controls the motor, laser, and ultrasonic sensor.
- User Interface (UI) that runs on an Android device. They communicate with each other over a Bluetooth serial communications link that can be initiated from the Android app.

1. Let's first take a look at the code that runs on the STM32 Nucleo. You can get the code from [GitHub](#). In short, the code makes the STM32 Nucleo sit in a loop waiting for a character to be received from the Bluetooth module. Based on the character received, a variety of functions can be triggered. The first 25 lines of the main code looks something like this, and the filename on GitHub (at the time of this writing) is *BuddyBot_v3_STM32 NUCLEO-f401RE.bin* :

```

1  // -----
2  //File:      /CatLaserProject/main.cpp
3  //Target HW:  ST Nucleo F401RE
4  //Description: This project allows a user to control a laser
5  //cat toy via Bluetooth from a companion Android app.
6  //
7  //Released under Creative Commons License
8  //Pinout:
9  //PA_2:  Computer Serial TX  / D1
10 //PA_3:  Computer Serial RX  / D0
11 //
12 //PA_8:  Laser                / D7
13 //
14 //PA_9:  Bluetooth Serial TX  / D8
15 //PA_10: Bluetooth Serial RX  / D2
16 //
17 //PB_3:  Left Kill Switch    / D3
18 //PB_4:  Right Kill Switch   / D5
19 //
20 //PB_6 /D10:  hMotor Coil 1A + to 293-2 | 1Y+  O B/W  293-3
21 //PA_7 /D11:  hMotor Coil 2A + to 293-15 | 2Y+  R Y/W  293-14
22 //PB_10 /D6:  hMotor Coil 1B - to 293-7  | 1Y-  B O/W  293-6
23 //PA_5 /D13:  hMotor Coil 2B - to 293-10 | 2Y-  Y R/W  293-11
24 //
25 //PA_6:  Ultrasonic Sensor    / D12

```

Figure 6: A section of the Nucleo main code from *BuddyBot_v3_STM32 NUCLEO-f401RE.bin*. You can look at an image file of the entire code in the [Source Files](#) area for this project as *Nucleo_main_cpp.png*, located inside *BuddyBotNucleoCode.zip* or get the code itself inside *BuddyBotNucleoCode.zip* or on github.

2. To drive the particular stepper motors we're using, I wrote a custom library that contains *motor.h* and *motor.cpp* files, for download as images (.png files) in the [Source Files](#) area. (As image files, they are named: *Nucleo_motor_h.png* and *Nucleo_motor_cpp.png* and are located in *BuddyBotPhotos.zip* with other images. The actual source code files *motor.h*, *motor.cpp*, and *main.cpp* are located in *BuddyBotNucleoCode.zip*)

Depending on how you construct your BuddyBot, there are a few key variables you may be interested in tweaking. In the *main.cpp* file:

- 1) **int triggerDistanceCM = 100:** This variable is the minimum distance, measured in centimeters, between the ultrasonic sensor and your pet that will trigger a rotation of the motor when the BuddyBot is in robot mode. This will need to be adjusted depending on the height and angle of your ultrasonic sensor. The taller the sensor, the larger the variable may need to be.

In the *motor.cpp* file you may wish to tweak the following:

- 1) **int _numMotorPulseCycleStandard = 5:** The number of motor rotation cycles that will run per click of the left or right button in the Android app. A smaller number will result in a smaller angle of rotation per each button press.
- 2) **int _numMotorPulseCycleMax = 10:** This variable performs the same function as _numMotorPulseCycleStandard except it controls the number of rotation cycles that occur when an interrupt is triggered by the limit switches.
3. Once you have the code all written, click the “Compile” button to create the necessary .bin file that you will drop onto your STM32 Nucleo. You will be asked where you want to save the file to on your computer. Installing code on the STM32 Nucleo is as simple as dragging the file from wherever you saved it onto the STM32 Nucleo, just as you would click-and-drag any file onto a USB thumb drive.
4. Now let’s take look at the Android application (whose source code is located in the [Source Files](#) page inside *AndroidApp.zip*). The user interface (UI) of the Android application is minimalist and consists of the following elements:
 - 1) “Laser On” checkbox to toggle the laser on and off. If the checkbox is checked, the laser will turn on. If it is unchecked, the laser will turn off.
 - 2) “Robot Mode” checkbox to toggle the automatic mode. If checked, the motor will move at random without user input. The “Left” and “Right” buttons will be disabled while in auto mode.
 - 3) “Left” button will turn the motor to the left when in manual mode.
 - 4) “Right” button will turn the motor to the right when in manual mode.
 - 5) “Pair Device” button will allow the user to pair their Android device to the BuddyBot.
 - 6) “Connect” button will allow the user to connect to the BuddyBot after pairing the two devices.
 - 7) “Disconnect” button will only appear once a BuddyBot is connected. Pressing “Disconnect” will stop the connection between the Android device and the BuddyBot.

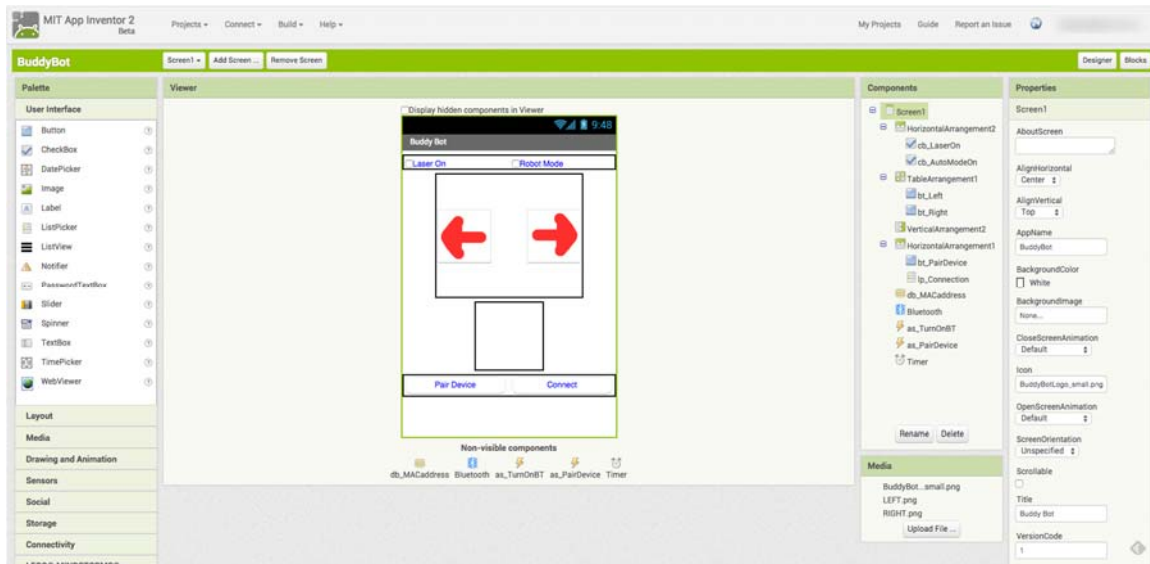


Figure 7: A screenshot of Android Designer on the [MIT Application Inventor](#) where the very fancy BuddyBot UI was created. <Android_Designer.png>

- The serial communications link is a simple, one-way link that passes single characters from the Android app to the BuddyBot hardware. The STM32 Nucleo code interprets each character and runs the appropriate function for each corresponding command character received. The following characters are sent based on the following button presses:

Character	Sent if
l (lower case L)	"Left" arrow button is pressed.
r	"Right" arrow button is pressed.
a	"Laser On" checkbox is checked.
z	"Laser On" checkbox is unchecked .
b	"Robot Mode" checkbox is checked. Puts BuddyBot into auto mode.
y	"Robot Mode" checkbox is unchecked . Puts BuddyBot into manual mode.

- If you are so inclined to write the code yourself or modify the provided BuddyBot.aia file (inside AndroidApp.zip) , then the "Designer" window of your screen should look similar to this:

The "Blocks" window of your screen should look similar to this:

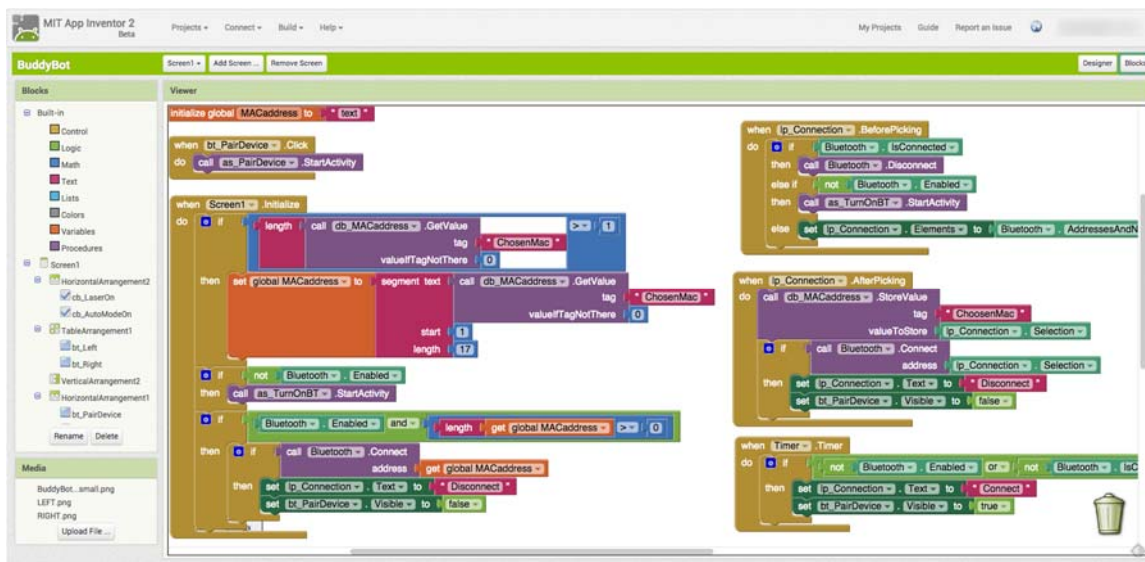


Figure 8a: Blocks window of the Android designer screen. <Android_Blocks01.png>

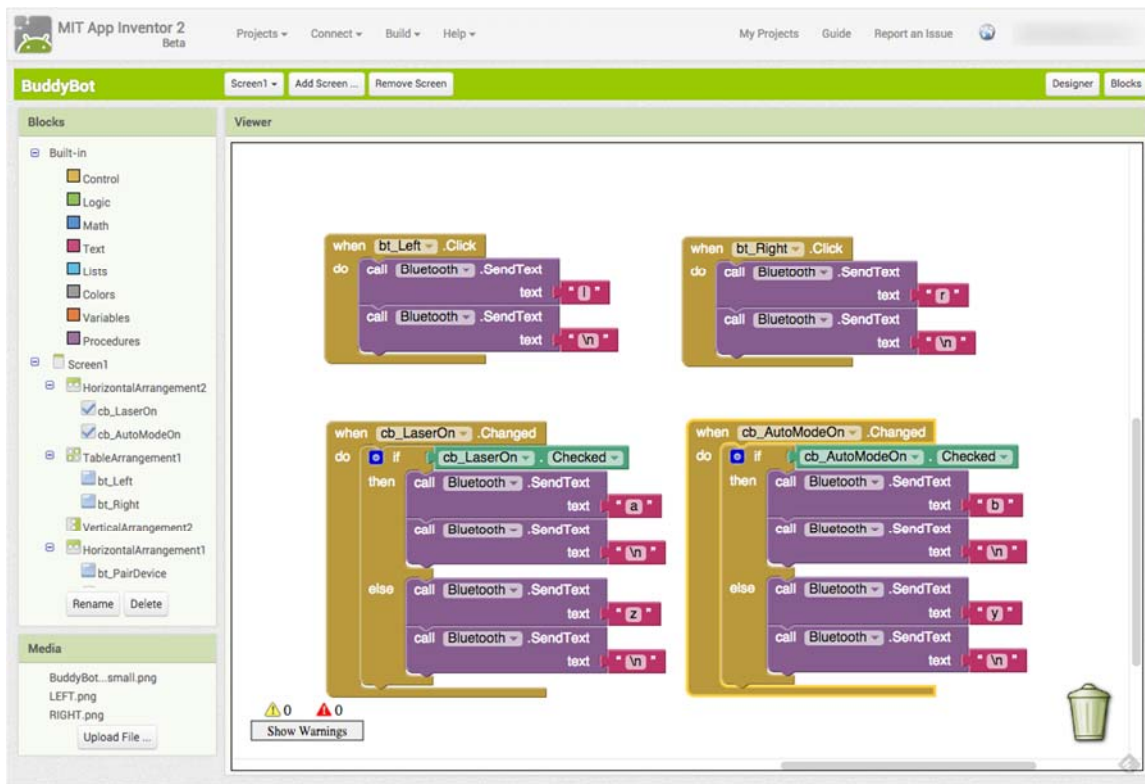


Figure 8b: Blocks window of the Android designer screen. <Android_Blocks01.png>

Assembly: Putting It All Together

As I mentioned in the Project Summary, I used a 3D printer to manufacture some of the mechanical interface components. I also used my amateur woodworking abilities to build the physical structure. Feel free to let your imagination run wild with laying out your BuddyBot. Some things to keep in mind though:

1. Give it sufficient height, and angle the laser accordingly to give yourself ample room for the dot to traverse around your hardware. You don't want Felix the Cat running into the box while chasing the laser.
2. You may need to tweak certain variables in the STM32 Nucleo code such as *triggerDistanceCM* in the *main.cpp* file to account for the height and angle of your laser.
3. Make sure to adjust the location of your limit switches so they are activated before the laser and PING sensor wires get wrapped up around the motor shaft.
4. If you power the device from a [USB wall charger](#) (instead of a computer's USB port), be sure to place a jumper across JP1, or else the STM32 Nucleo will not power the microcontroller.

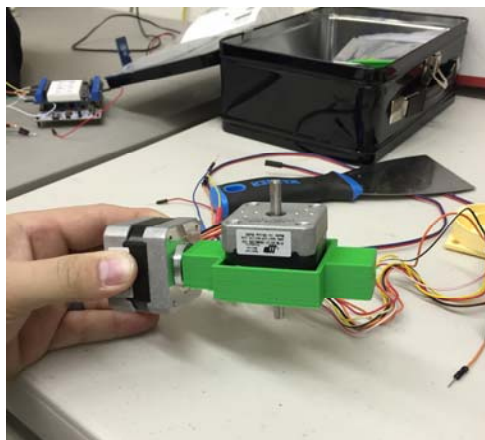


Figure 9: 3D printer files are provided for the enclosure if you have one, but is not necessary, depending on how violent your cat gets when it plays. < 3D-printer.JPG>

Figure 10: Assembly. <IMG_0465_2.jpg>

Time to fire it all up!

1. Plug the USB cable into the STM32 Nucleo and into a wall outlet.
2. Make sure that there are blinking LEDs on both the STM32 Nucleo and the Bluetooth module.
3. Launch the BuddyBot app on your Android device.
4. Click the “Pair” button to pair to the RN42 device. You should only have to ever do this once.
5. Next click the “Connect” button and the blinking LED on the Bluetooth module should become solid.
6. Try turning the laser on and moving the motor!



Figure 11: Testing the Buddy Bot in random mode. I also made a housing to put around it and protect the electronics from jumping cats. < IMG_0471_2.jpg>

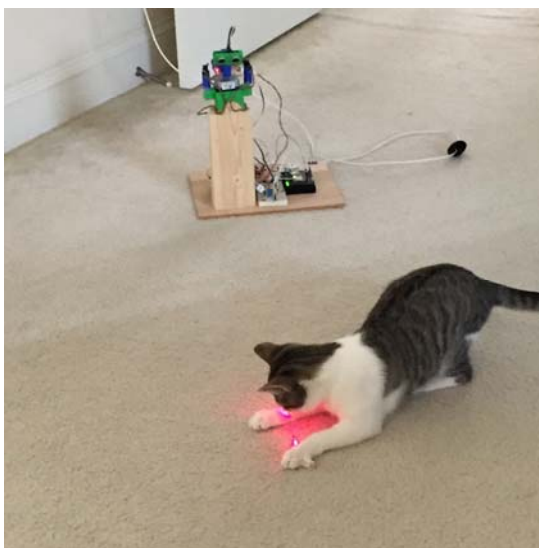


Figure 12: Testing the BuddyBot with my tablet. This is Bluetooth wireless, so you have to be within about 15 feet of BuddyBot to control it, but you could re-program the app to turn on in random mode at certain times of the day, if you’re willing to leave your tablet within 15 feet of Buddy Bot. < exciting_cat_times.jpg>

So there you have it, you’ve built your very own BuddyBot! Please send us pictures of your furry friend enjoying their new toy. If you make any modifications to the design, please share that as well. We can’t wait to see what you do!

Additional Resources:

STM32 Nucleo to Arduino Pin Layout Conversion “Cheat” Sheet

I/O Function	STM32 Nucleo	Arduino Pin
Computer Serial Receive (RX)	PA_3	D0
Computer Serial Transmit (TX)	PA_2	D1
Laser	PA_8	D7
Bluetooth Serial Transmit (TX)	PA_9	D8
Bluetooth Serial Receive (RX)	PA_10	D2
Left Limit Switch	PB_3	D3
Right Limit Switch	PC_7	D9
Ultrasonic Sensor	PA_6	D12

Motor Connection	Wiring from Microcontroller to L293D ¹			Wiring from Stepper Motor to L293D ²		
	STM32 Nucleo Pin #	Arduino Pin #	L293D Pin #	Stepper Wire #1	Stepper Wire #2	L293D Pin #
hMotor Coil 1A +	PB_6	D10	2	Orange	Black/White	3
hMotor Coil 2A +	PA_7	D11	15	Red	Yellow/White	14
hMotor Coil 1B -	PB_10	D6	7	Black	Orange/White	6
hMotor Coil 2B -	PA_5	D13	10	Yellow	Red/White	11

¹ Nucleo header pins are identified by both a Morpho and Arduino pin identifier. STM32 Nucleo Pin # and Arduino Pin # refer to the same physical pin. The Arduino Pin # makes it easier to determine which pin to use since they are sequentially numbered.

² Both Stepper Wire #1 and #2 must be connected to the associated L293D pin. (e.g. Both the solid orange and the black/white striped wires must connect to L293D pin 3)

RN42 Pin Layout “Cheat” Sheet

Function	RN42 Pin Number
Bluetooth 3V	Pin 1
BT_RX	Pin 2
BT_TX	Pin 3
Bluetooth GND	Pin 10

On the [STM32 Nucleo](#), Arduino™ connectivity support and ST Morpho headers make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields. The STM32 Nucleo board does not require any separate probe as it integrates the ST-LINK/V2-1 debugger/programmer. The STM32 Nucleo board comes with the STM32 comprehensive software HAL library together with various packaged software examples, as well as direct access to mbed online resources.

[STM32 Nucleo Resources](#)

The [Microchip RN-42 Bluetooth Module](#) is a small form factor, low power, class 2 Bluetooth radio for designer's who want to add wireless capability to their products. The RN-42 supports multiple interface protocols, is simple to design in, and is fully certified, making it a complete embedded Bluetooth solution. The RN-42 is functionally compatible with the RN 41. With its high-performance, on-chip antenna and support for Bluetooth EDR, the RN-42 delivers up to a 3 Mbps data rate for distances up to 20 meters.

Source Files and Downloads

3D Printer Files: These STL files are used if you have a 3D printer to create the non-electronics portion of the project. 3D printing is not necessary to create the enclosure for project, however. Download them here as: [3DPrinterFiles.zip](#)

Android Application files: Source files for the Android application that is used to control the BuddyBot from an Android device (tablet, smartphone, etc.) You need to be running the JellyBean release or higher on the device to ensure compatibility. Download here as: [AndroidApp.zip](#)

STM32 Nucleo Code: Original source files for programming the main STM32 Nucleo board. However, any subsequent revisions can be found on [Github](#). Mbed.org is the tool used to program the STM32 Nucleo. Download here as: [BuddyBotSTM32 NucleoCode.zip](#)

Images/Photos: Images of the BuddyBot and .PNG images of the original, commented source code are available here for download: [BuddyBotPhotos.zip](#)

Schematic Source files: This project wouldn't be open source hardware without the source files for the schematics. There is no PCB (yet), or we would provide source for that, as well. [SchematicSource.zip](#)

This project is licensed under [CC BY-SA 4.0](#)