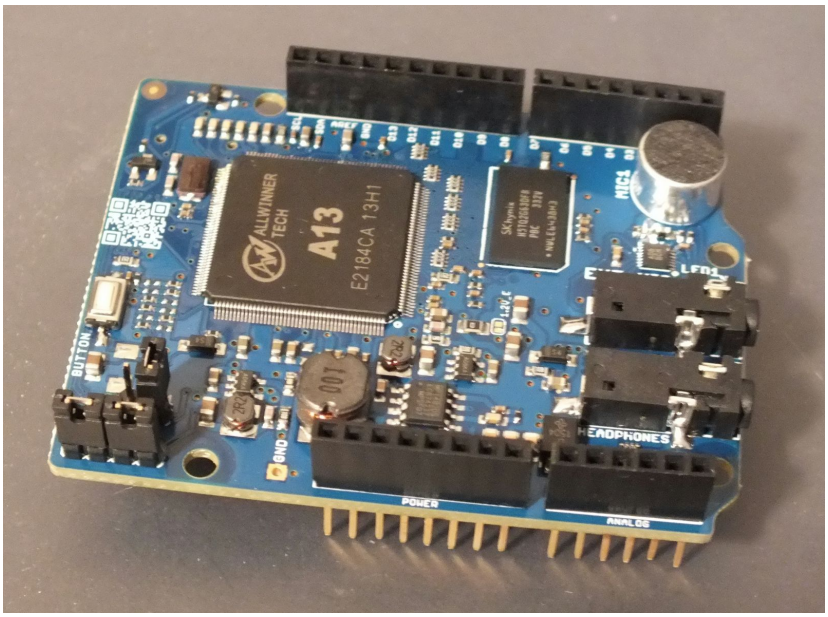
MOVI™ Voice Control Shield for Arduino® boards

User's Manual

covers MOVI™ v1.00



Acknowledgements

MOVI™ would not have been possible without the enduring feedback and assistance of many people. First, we'd like to thank the 334 Kickstarter backers who, through their financial contribution, questions and comments allowed us to make this project happen. A list of the backers is available here: http://www.audeme.com/kickstarter-backers.html

Moreover, we are grateful for the voluntary help by our beta testers Jared Peters from Origami Robotics and Lars Knipping from University of Technology Berlin.

We are also indebted to the open source community. Without the many people creating open source tools, we wouldn't been able to put together MOVI. MOVI uses the following open source packages:

- The Advanced Linux Sound Architecture (ALSA) project (http://www.alsa-project.org)
- eSpeak text to speech (http://espeak.sourceforge.net/)
- CMU Sphinx and PocketSphinx, Speech Recognition Toolkit (http://sourceforge.net/projects/cmusphinx/)
- The OpenFst Library (http://www.openfst.org/twiki/bin/view/FST/WebHome)
- The Many-to-Many alignment model (https://code.google.com/p/m2m-aligner/)
- The CMUCLMTK Development Toolkit (http://cmusphinx.sourceforge.net/wiki/cmuclmtkdevelopment)
- The Phonetisaurus package (https://github.com/AdolfVonKleist/Phonetisaurus)
- The MIT Language Modeling Toolkit (https://code.google.com/p/mitlm/)

MOVI is booting a standard Linux Kernel using u-boot and is relying on many of the GNU standard tools and libraries such as GLIBC, bash, etc. We also make extensive use of Python (https://www.python.org/).

The SDCard shipped with MOVI has a mountable ext3 partition "MOVI Root" which contains a directory *src/* with instructions how to obtain the source code for any of the open source packages we used, independently of the any license requirement for us to do that.

Ultimately though, we need to thank you, the reader of this document, for you interest and commitment. Users are what makes a product.

Thank you so much for all your support!

Table of Contents

1. Introduction

Welcome to MOVI!

MOVI stands for *My Own Voice Interface* and is the first standalone speech recognizer and voice synthesizer for Arduino with full English sentence capability:

- Lot's of space for customizable English sentences (we tested up to 200)
- Speaker independent
- Standalone, cloudless and private
- Easy to program
- Male and female speech synthesizer included

MOVI provides an alternative to buttons, remote controls, or cell phones by letting you use full-sentence voice commands for tasks such as turning devices on and off, entering alarm codes, and carrying on programmed conversations with projects.

This manual will guide you through the first steps, provides compatibility information and serves as a programming reference. It will also give you some tips on how to get the best speech recognitions results with MOVI. We know how boring it seems to have to read a manual when a shiny device just arrived in the mail and all we want is to get our hands on it… However, the world of speech recognition is not only fascinating but also sometimes tricky and things that should be easy aren't while things that should be hard just miraculously work. So we strongly recommend to not only read this manual but to also keep it handy.

Yours,
Bertrand Irissou and Gerald Friedland
Makers of MOVI

2. Board Description

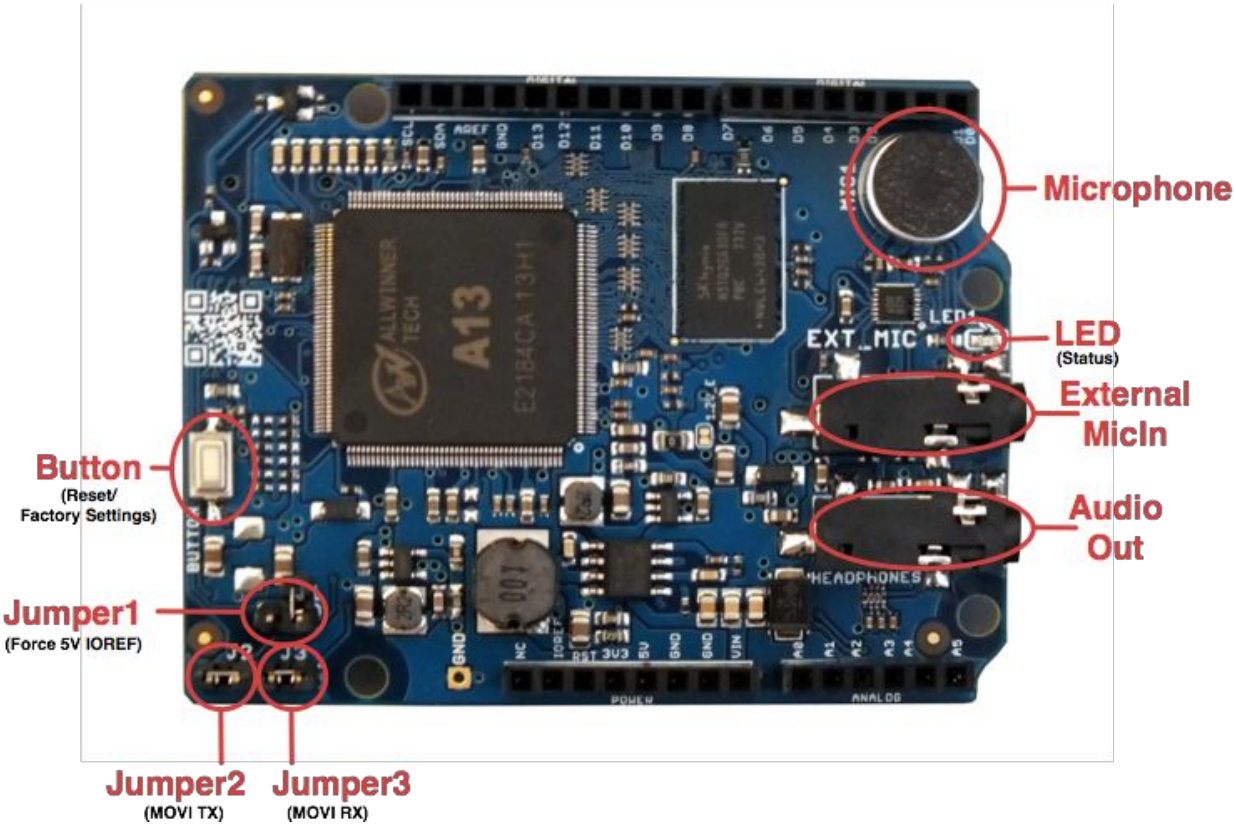


Figure 1. Birds-eye view of the MOVI(TM) board.

Figure 1 shows your MOVI board from the top with a legend of the most important components.

Power Supply

What you don't see in Figure 1 is a power supply jack. MOVI is powered through the Arduino board that needs to be powered using an external power supply. The external power supply should provide between 7V and 16V and at least 500mA current. During tests we usually used either 9V or 12V. Battery packs with this specification work as well.

Please note: MOVI cannot be powered through a USB power supply or the USB cable of the Arduino board as the voltage provided is less than 6V.

Speakers

For proper operation, MOVI requires a speaker connected to Audio Out. The speaker can be mono or stereo but the signal provided by MOVI is mono. The speaker impedance should be 32 ohms, which is the standard impedance for headphones. The output volume can be controlled in software using the MOVI library (see Appendix B and C). For convenience, we recommend active speakers with an amplifier and volume control.

Note: Do not connect 4 ohm or 8 ohm speakers. These require an amplifier and might damage the board!

Note: Only connect 2-conductor (mono) and 3-conductor (stereo) headphone jacks to MOVI. 4-conductor jacks, as used by many smartphones (stereo plus microphone), require an adapter.

Audio Input

By default, the integrated electret microphone (see Microphone in Figure 1) is used. This microphone is internally connected to an Automatic Gain Control that will amplify incoming sounds to standard level independent of the distance. This will work up to about 15 feet (5 meters), under good conditions sometimes even for wider distances. Under bad conditions (noise, room echo) the distance will be shorter. For usability reasons or in difficult environmental conditions, a headset microphone should be used. A headset microphone or an alternative electret microphone can be connected to External MicIn. This audio jack is a stereo jack but only accepts a mono signal. Connecting a device to External MicIn disables the integrated microphone. Also, the signal that comes through External MicIn is not amplified.

Note: Do not connect a Line-In signal or any other signal that is pre-amplified to the microphone jack. Also, microphones that require phantom power will not work.

Note: Only connect 2-conductor (mono) and 3-conductor (stereo) headphone jacks to MOVI. 4-conductor jacks, as used by many smartphones (stereo plus microphone), require an adapter.

LED

MOVI uses the LED as an indicator for the state that MOVI is in. The following states are signalled:

LED off: LED constantly off means MOVI is turned off, there is not enough power to operate, and/or the SD card is not plugged in.

LED blinking faster and faster: MOVI is booting.

LED blinking randomly: MOVI is writing to the SD-Card. This happens during an update, training, or resetting to defaults. MOVI should not be powered off while the LED is blinking randomly.

LED blinking with constant frequency: If MOVI's LED is blinking with constant frequency, there is a serious issue with the SD-Card e.g., MOVI's file system check failed permanently.

LED constantly on: MOVI's LED constantly on indicates MOVI is ready to operate or is operating normally. Only in this state, MOVI will recognize the call sign and the programmed sentences.

Reset Button

The button is programmed as a **reset button** on a short press. MOVI will reboot (not the underlying Arduino board though). Please do not press the reset button while the LED blinks with randomly (see above).

A **long press** will revert MOVI's callsign, trained sentences and other configuration parameters to factory default. Please note, that the board can't be reset and shouldn't be powered off during the restoration process.

Jumper 1 (5V_REF)

On most boards Jumper 1 must be open. Jumper 1 only needs to be closed when a 5 V board is used that does not have the IOREF pin. This is explained in detail in Appendix A.

Caution: Jumper 1 hard wires the 5 V pin to IOREF. Therefore, setting Jumper 1 on a 3.3V board will destroy the board!

Jumper 2 and Jumper 3 (Alternative Serial Communication)

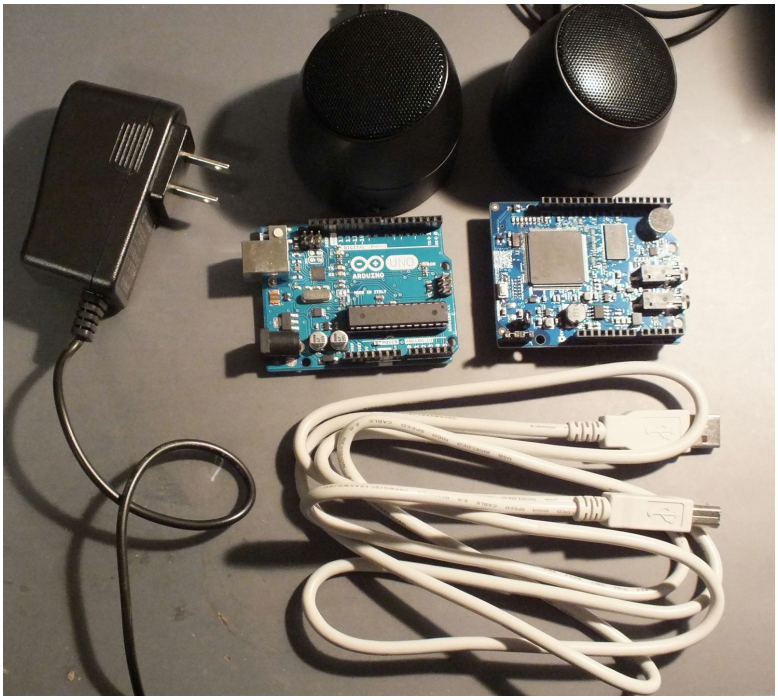
In most cases, Jumper 2 and Jumper 3 should be closed. By default, MOVI is using Arduino pins D10 and D11 for communication between shield and board. If these pins are used for other purposes or by another shield, Jumper 2 and Jumper 3 can be used to rewire the communication. Also, some boards, such as the Arduino Due, are not able to have serial communication on D10 and D11 and therefore need to be rewired for MOVI to operate. Refer Appendix A for details.

To rewire MOVI's communication to different pins, open Jumper 2 and Jumper 3 and connect the right side of MOVI's TX jumper (Jumper 2) and the right side of MOVI's RX jumper (Jumper 3) to two other connectors on the Arduino headers using jumper wires. The right side is the pin that is closer to the power and USB plugs.

3. Getting Started

If you use a new Arduino UNO R3, an Arduino MEGA R3 or an Arduino Leonardo R3 (with an IOREF pin) go right ahead through this Section. If you are not sure or use any other board, including older versions of the UNO, MEGA and LEONARDO and "compatibles" please read Appendix A first.

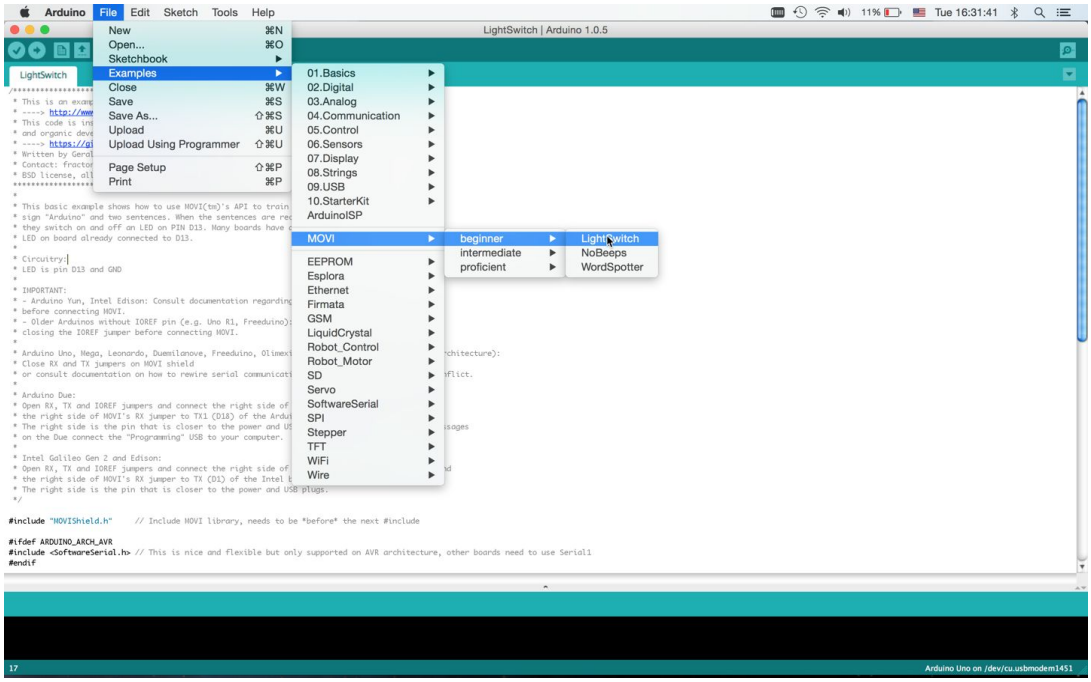
You need: A computer that can run the Arduino IDE, your MOVI board, your Arduino board, your Arduino programming cable, an external power supply, and active speakers that can be driven by a headphone jack. Optional: A green and a red LED for some of the example sketches.



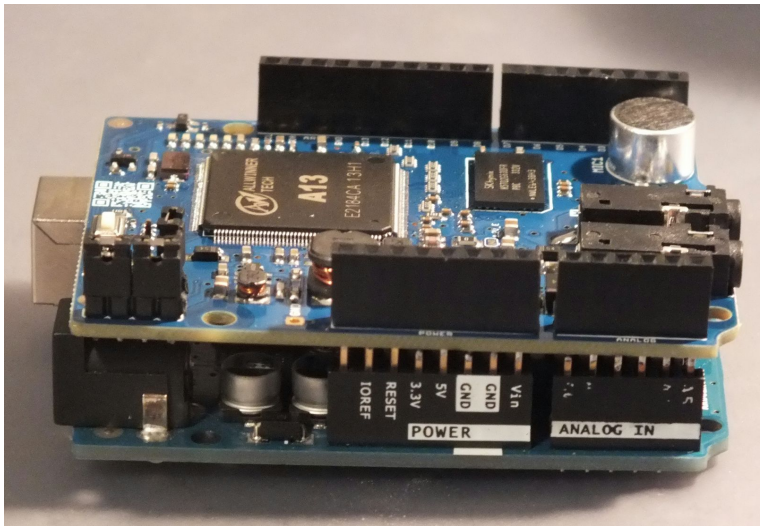
1. Download and install Arduino IDE recommended for your board.
   To do that, follow the instructions in your Arduino documentation or on this website: https://www.arduino.cc/en/Main/Software

2. Download the MOVI library as a zip file from http://www.audeme.com/MOVI/.
   Users familiar with both MOVI and open source programming may also check out the latest source code from https://github.com/audeme/MOVIArduinoAPI.

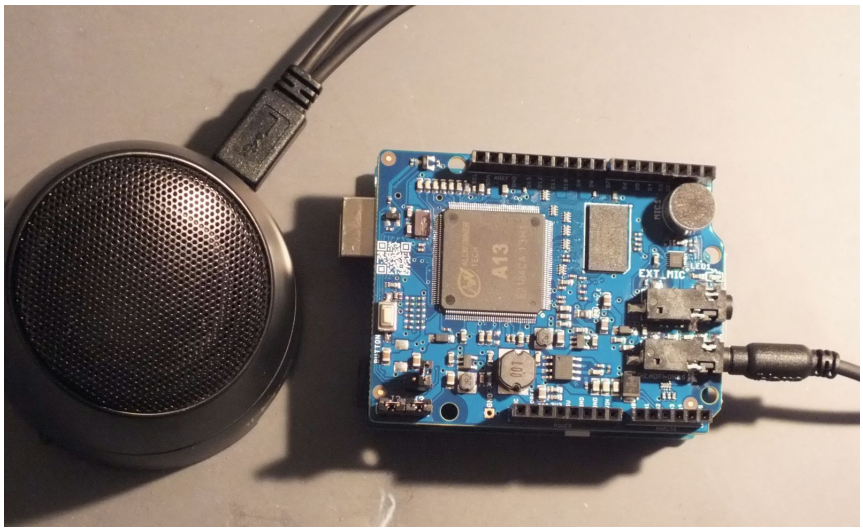
3. Install the MOVI library into the Arduino IDE. Instructions on how to install a library can be found here: https://www.arduino.cc/en/Guide/Libraries
4. Load the *LightSwitch* example by opening the *File* menu under *Examples*. Choose *MOVI* or *MOVI(tm) Voice Control Shield*, depending on the version of the IDE. The result should look similar to this:



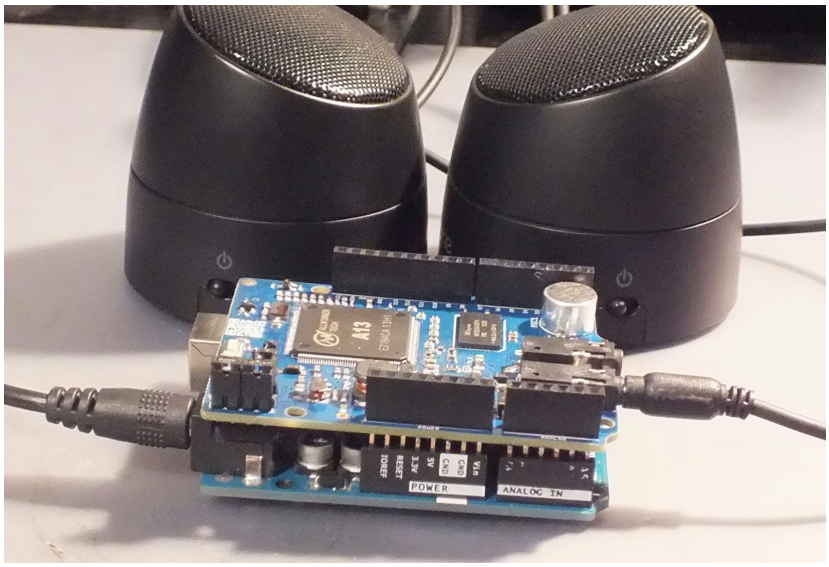
5. Disconnect all power and USB cables from your Arduino board and connect the MOVI shield onto it:



6. Connect an external speaker or a headset to the Audio Out (see Section 2). Audio Out is labeled “HEADPHONES” and is the audio jack further away from the integrated microphone, closer to the Arduino headers.



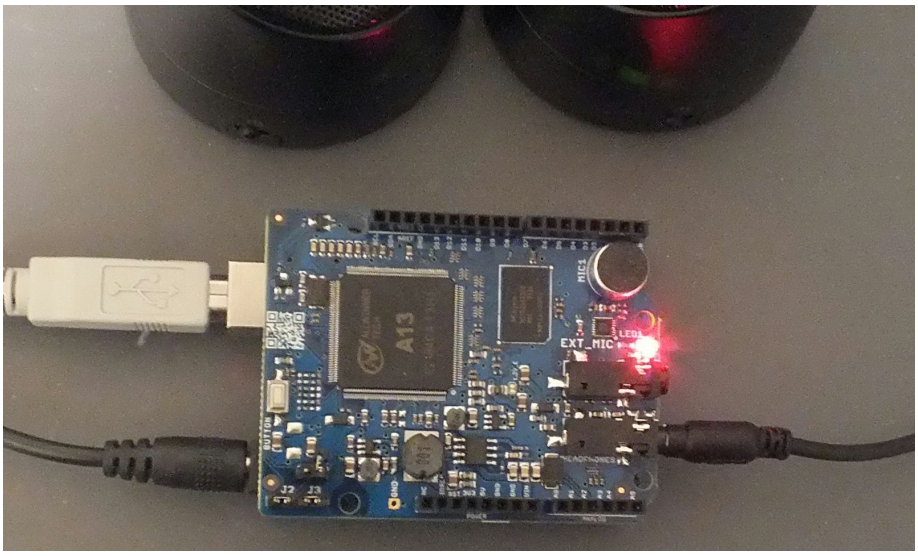
7. On an Arduino board with IOREF pin, make sure Jumper 1 is open (unset). Otherwise, check Appendix A about the best setting for Jumper 1. Connect the external power supply to the Arduino board and switch it on.



8. After about 2 seconds, you should see MOVI’s LED (close to microphone) blinking with increasing frequency. The speakers will say “MOVI is booting”. Eventually the

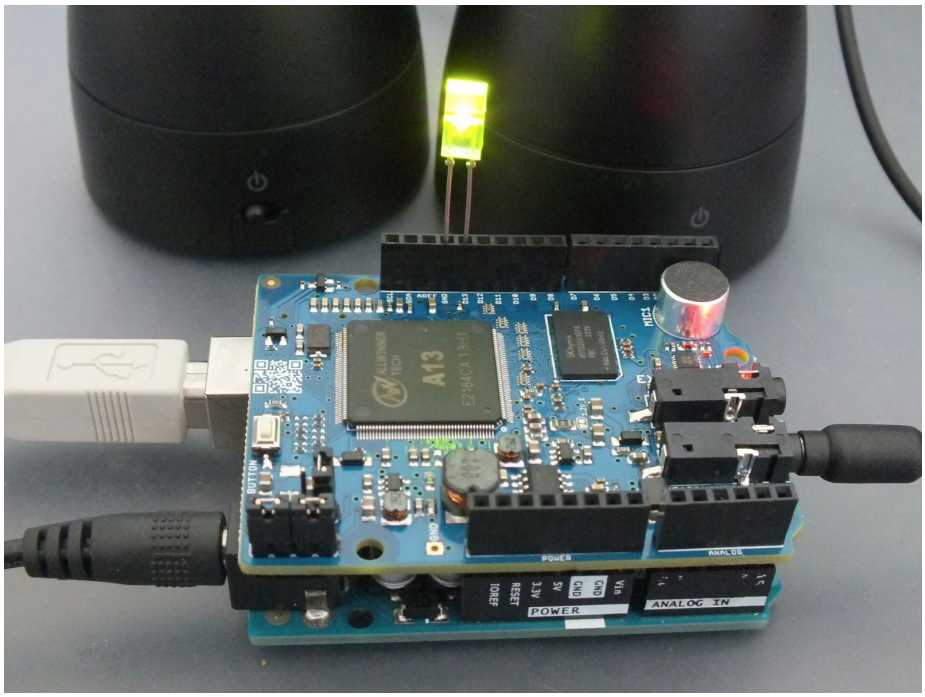
LED will stop blinking and just be constantly on. This indicates MOVI is ready. If the LED does not go on at all, please turn off the power and read Appendix A. If, by the time, the LED has become steady red, you didn't hear anything, please check your speakers/headset and the connection.

9. Connect the USB programming cable to the Arduino board



Important: Always, connect the USB cable **after** you have connected the external power supply. It is safe to disconnect the USB while the power is on. With the exception of MOVI updating, learning a new call sign, learning new sentences, or resetting to factory settings, you can always unplug the power safely. MOVI's LED (close to the microphone, see image) will blink randomly while it is not safe to unplug. **However, please do not disconnect the external power while the USB cable is plugged to the Arduino. Powering MOVI from USB will not supply enough voltage to the board and will therefore leave MOVI in an unstable state where the LED might be on or blinking but MOVI not work properly.**

10. In the Arduino IDE, compile and upload the *LightSwitch* Example.
11. Get close to the microphone capsule and say "Arduino" in a normal voice, wait for <beep beep>. Say "Let there be light". Wait for <beep>.
12. Speakers should say "and there was light" and LED on Arduino board turns on. Please note that Arduino's onboard LED might be a bit hidden below the MOVI shield. For better visibility, connect an LED to Arduino port D13 (+) and GND (-).



13. Say “Arduino”, wait for <beep beep>. Say “Go dark”. Wait for <beep>
14. LED on Arduino board turns off.
15. Congrats! Now play around with the code. Load other examples. Read ahead in this manual, especially Section 4.

4. Getting the Best Speech Recognition Results

Read this section when everything is setup and you have had your first experiences with MOVI but before you are about to plan out your first bigger project. Speech Recognition can be tricky and sometimes things just need a little `magic` even when everything was setup correctly. After all, speech recognition is still a field of active research and many problems haven't even been nearly solved (http://en.wikipedia.org/wiki/Speech_recognition). Having said that, knowing how MOVI principally works will help tinkering with issues that come up.

Operation Modes

MOVI's speech recognizer has two basic modes of operation, training and recognition, which are described as follows.

Training: MOVI's Arduino library sends the training sentences in textual form over the serial connection to the shield. The shield phonetizes the words in each sentences using a 2 GB English dictionary that knows spelling rules and approximates even for proper names. The phoneme sequences are used to create a temporal model that makes sure that only words are recognized that have been part of the training sentences. A second temporal model favors word sequences that are part of the sentences over sequences that are not by assigning higher probabilities to phoneme sequences that occurred in the trained sentences over those that didn't.

Recognition: During recognition, a waveform comes in over the microphone and is broken down into speech and non-speech regions. This is done by an algorithm that monitors the energy of the incoming signal over a short time period and compares it to a threshold. If the pattern looks like speech and speech pauses, it assumes speech, otherwise the signal is ignored. The speech regions of the signal are then passed to a classifier that has been trained on hundreds of adult speakers. It breaks down the waveform into possible phonemes sequences. Using the temporal model created in training, the phoneme sequences are matched to the pre-trained words and also word sequences that are part of the training sentences are favored. A last correction step

maps the words to the most likely sentence in the training set (result from *poll()*). This second step can be omitted in the library by using *getResult()*.

Now that we've got that out of the way, let's discuss some common issues.

Training Sentences vs Words

Let's assume you want to recognize any combination of numbers between one and three. Is it better to train one sentence "one two three" or three 'sentences' "one", "two", and "three"? If the goal is to recognize any combination of the numbers one, two, and three and each of them are equally likely to appear, it's best to train three sentences and use the *getResult()* method. Training one sentences will work too but there is a high likelihood that the recognizer will favor "one two three".

If it's really always a combination of three different numbers between one and three, it is preferable to train all six combinations of "one two three", "one three two", "two three one", "three two one", "two one three", "three one two". This way, *poll()* can be used and MOVI's algorithm can correct errors.

What if the combination of numbers was between 1 and 10? We have tested MOVI successfully with about 150 short sentences and we are pretty sure there can be some more but we also know training 10! = 3628800 sentences will not work. So obviously 10 sentences need to be trained and *getResult()* needs to be used.

What if only one number between one and ten was to be recognized? In this, case it is fine to train one sentence of ("one two three four five six seven eight nine ten") since it saves memory and training speed and the temporality isn't used anyways as there is only one word to be recognized. However, training ten sentences will not harm the recognition accuracy.

What it there was some known word sequencing but not for the entire sentence? Let's say you want to recognize '0.x' and '1.x' with x being a number between 0 and 9. The best way to do this is to train twenty sentences "zero point zero", "zero point one", "zero point two", ... "one point nine". However, if the acoustic conditions in the room are good, it's feasible to break the sequences up into less sentences, for example: "zero point", "one

point", and 8 single word sentences "two", "three", "four", etc… (the words zero and one have already been trained). This may be further reduced to three sentences by making the numbers 2-9 one sentence "two three four five six seven eight nine ten". Splitting up this task in less than twenty sentences, however, requires to use the *getResult()* method.

The overall rule of thumb is: Favor training all known word sequences as sentences. Otherwise, train words as sentences individually.

Sentences do not get higher priority if they are defined twice as, in fact, the system will remove duplicate sentences. However, if one can give a certain sequence (out of many possible) a higher priority by first defining individual word sentences and then the actual sequence. For example, defining the sentence "one", "two", "three" and the sentences "three two one" will give a higher probability to the sequence "three two one" than any other sequence. This does play a role in noisy room conditions.

If you want to create a keyword spotter, e.g. recognize a particular word out of many, it's best to train a lot of other words as well. For example, if you want to recognize whenever the word "Simon" appears in a sentence, you would train the word "simon" as a sentence along with a set of other words, for example words that appear very frequently in English such as "the", "be", "to", "off", (for a more comprehensive list checkout this link: https://en.wikipedia.org/wiki/Most_common_words_in_English) as well as words that are similar to Simon (e.g, "assignment"). This way, these words are recognized and it lowers the false alarm rate of MOVI detecting "Simon" when other words are spoken.

Please also note that for sentence matching (as part of the *poll()* function) it is best for all trained sentences to have about equal length. A single very long sentence will always be favored when a lot of words are spoken.

Saving Arduino Memory

Some Arduino models, especially the Uno, come with very limited memory. Using the *addSentence()* commands in the *init()* method is convenient but it does mean that the sentences are stored in Arduino's memory even when MOVI has already learned them. More on Arduino's memory restrictions can be found here: https://www.arduino.cc/en/Tutorial/Memory

The first solution is to uncomment the *addSentence()* and *train()* calls and re-compile and upload after MOVI has learned the sentences. Please note, however, that the MOVI API itself as well as another other libraries potentially included in a sketch also occupy some SRAM. Another solution therefore is to use the so-called *PROGMEM* method and the *F()* macro to store variables in flash memory. The concept is described here: https://www.arduino.cc/en/Reference/PROGMEM

MOVI's API allows to use *F()* macro strings to be used with *addSentence, say, ask,* and *password*. This means *addSentence("Let there be light")* works as well as *addSentence(F("Let there be light")* but the second options saves critical SRAM.

If the above tricks does not provide enough memory savings, then the best way is to use the low level interface. Compile and upload the LowLevelInterface example (see *Examples/MOVI/proficient/LowLevelInterface* in the Arduino IDE menu) or make sure the MOVI object is constructed using 'true' as first argument. Open the Serial Console and then use the manual *TRAIN* command as described in Appendix C. The *TRAIN* command will ask for one sentence line by line (enter ends a sentence) until "#" is used to finish the input and will then automatically learn all the sentences given. Then switch back to your project and use *poll()* and/or *getResult()* normally. Just make sure, no *addSentence()* or *train()* call is used in your sketch, as this would overwrite your trained sentences. Sentences are stored even after MOVI is reset or powered down until either retrained or a factory reset is initiated, regardless of the method used for training.

Good Call Signs

Since MOVI maps any input to the trained words, regardless of how far off it is, MOVI uses a keyword spotting algorithm to make sure the sound registered is going to be intended for recognition. This cuts down on false alarm rates and allows MOVI to run in the background, e.g. as a light switch that doesn't go erratic while a TV is on. Good call signs are call signs that are pretty distinctive from other words. Obviously, choosing the word "the" or "to" as a call sign works pretty terribly. We found that names that are easy to pronounce and spell and contain two to three syllables work the best. The default, "Arduino" has more syllables but definitely works. So does "MOVI" or "computer". Call

signs from other personal assistant speech recognition products, unsurprisingly, work well too.

The Role of the Acoustic Environment

Perhaps one of the most counterintuitive flaws of state-of-the-art speech recognizers is that they can't cope with the influence that the environment has on the acoustic properties of the speech signal. At the same time, the human ear is incredibly good at it. The human ear can ignore overlapping sounds, echo, reverberation, noise induced by wind, etc. Speech recognizers are not able to do that or only to a limit extend. Room influence is usually a bigger factor than individual speech variance, such as accent. Having said that, speaking rate is a factor as well. Very slow or very fast speech is hard to recognize as well.

While testing MOVI, we have observed situations where the room was completely silent, yet MOVI had serious problems picking up the call sign. Analyzing the situation, we found that there was too much reverberation in the room. It was not audible but clearly visible on the oscilloscope. Short of installing carpet just to make MOVI work, we found different locations in the room worked with different accuracy. Also, of course, the closer we moved to MOVI's integrated microphone, the better it work as this cuts down on the room influence. Using a headset microphone worked perfectly.

The rule of thumb here is: Try to shorten the distance to the microphone as much as possible. If in doubt, use a headset microphone connected to External MicIn.

Operating MOVI under Noisy Conditions

By default MOVI, will give a one-time spoken warning about a too noisy room environment. Moreover, if you find that MOVI takes very long to acknowledge your spoken sentence with beeps after you are finished, the noise level is too high. Needless to say, with the presence of any kind of noise, recognition accuracy will go down, especially when using the *getResult()* method.

The short version is: If there is significant noise in the room, use a headset microphone connected to External MicIn.

If the noise isn't too heavy and a headset is not in question, MOVI provides the *THRESHOLD* command (or setT*hreshold()* call in the MOVI library). The call sets the noise threshold of the speech/non-speech detector component of the recognizer. The possible values the command takes can be between 2 and 95 (percent). The factory default is 5 percent. We found that typically a value of 15 percent is good for somewhat noisy environments and a value of 30 percent for very noisy environments. Ultimately, however, experimentation in the concrete environment is the only way to determine a good noise threshold.

In order to do this, use the low level interface with the *MICDEBUG* command. To access the low level interface, compile and upload the LowLevelInterface example (see *Examples/MOVI/proficient/LowLevelInterface* in the Arduino IDE menu) or make sure the MOVI object is constructed using 'true' as first argument. Open the Serial Console in the Ardunio IDE. Then issue the command *MICDEBUG ON* followed by *RESTART*. The recognizer will restart and operate normally, with the exception that every detected input is played back through the speakers, which allows to listen to the speech signal (and other noise) as captured through the microphone (or connected headset). In the Serial Console, now slowly increase the value of the energy threshold by issuing *THRESHOLD* commands (e.g. *THRESHOLD 5*, *THRESHOLD 10*, *THRESHOLD 15*) and calling MOVI and speaking a sentence until in between each *THRESHOLD* call until MOVI's reaction time seems normal again. *MICDEBUG OFF* followed by a *RESTART* will set MOVI back to normal. Setting the threshold too high, however, will make MOVI insensitive and you will probably have to yell at it. ;-)

5. MOVI Firmware Updates

MOVI is a supported product and Audeme will from time to time provide updates to its firmware to improve performance and add features. To update MOVI's firmware, you need an SD-Card writer and, most likely, a micro SD to SD-Card adapter. Follow these steps:

1. Download the update file provided by Audeme, LLC to a computer that has an SD-Card writer.
2. Unplug all power connections, as well as the USB cable from the Arduino and remove MOVI's SD-Card by pressing on it gently before pulling it out.
3. Put the SD-Card into the computer. One or two file systems should appear automatically: *MOVI UPDATE* and *MOVI Root.* The latter is an ext3 (Linux) filesystem that will only appear if your operating system is able to mount ext3 filesystems. It can be used to perform manual changes on MOVI, e.g. by accessing and compiling source code. On Windows and most Mac computers, only *MOVI UPDATE* will appear.
4. Copy the update file downloaded in step 1 onto the the *MOVI UPDATE* partition.
5. Eject (or unmount) the SD-Card correctly. If the *MOVI Root* partition is visible on your computer, make sure to cleanly unmount this partition as MOVI's file system check capabilities are limited.
6. Re-insert the SD-Card into MOVI by gently pressing the card in until it locks.
7. Connect speakers and power up the Arduino/MOVI combination using an external power supply and listen to the status messages on the speakers. The update should be performed automatically followed by a restart.

6. Further Information

Further information about MOVI is available on Audeme's website:
http://www.audeme.com/MOVI

which also includes a forum. Furthermore, we have been maintaining a Facebook page:
https://www.facebook.com/asrshield

and MOVI's Arduino library is maintained on GitHub:
https://github.com/audeme/MOVIArduinoAPI.

The original Kickstarter page is here:
https://www.kickstarter.com/projects/310865303/movi-a-standalone-speech-recognizer-shield-for-ard

7. FAQ

This FAQ is most likely outdated. The online version of the FAQ can be found on Audeme's website.

- I uploaded my sketch, now MOVI doesn't react anymore when I type commands on the Serial Monitor.

  As with any device there are a number of ways to make MOVI or the underlying Arduino freeze. The three most common causes are these:

  - MOVI is not connected to an external power supply
  - The sketch uses too much SRAM due to too many and too long sentences. This can be fixed by putting as many strings as possible into flash memory, as is described in Section 4.
  - A command is sent to MOVI in every *loop()* cycle.
    Without exception, sending a command to MOVI inside the *loop()* function should only happen after catching a certain event with *poll().* The only function that is save to call without an *if*-statement in front is *poll().* Also note, that *poll()* returns 0 for no event. Sending any command to MOVI after no-event has the same effect.

  In any case, if MOVI is hung up in this way, the easiest way to get MOVI back to life is to load one of the example programs onto the Arduino and then reset MOVI while connected to an external power supply.

- Is MOVI available for Rasberry PI?
  Not at the time of writing this document. However, the MOVI library is open source and Appendix A describes the principal hardware requirements for operating MOVI. We therefore encourage the Maker community to figure it out and publish about it. ;-)

- MOVI doesn't seem to work with my children, what's wrong?

MOVI's acoustic models have been trained only on adults as training with children is inherently difficult. Therefore, MOVI works best with people over 12 years old.

- How do I train other languages and accents?
  At the moment, only US English is supported.

- Does MOVI really need an external power supply? Sometimes it works over USB.
  Lucky you! Don't rely on it. See first question.

- Can I operate MOVI on batteries instead of an AC adapter?
  Yes. We have successfully operated MOVI on an Arduino Uno using a 9V block battery (or two in parallel for longer operation). AAs and AAAs work too, as long as enough of them are being used. Depending on the type, the voltages of AA and AAA batteries range from 1.2V to 1.5V per battery. Also, voltage might drop over time. Therefore we recommend using at least 5 batteries of this type, better 6.

Appendix

A. Compatibility

We developed the MOVI shield using an original Arduino Uno R3 and an Arduino Mega2560 R3. Consequently, we recommend you get one of these boards if you don't already have an Arduino. However, we also tested MOVI with other Arduino and compatible boards.

General Compatibility

With exceptions, MOVI is generally compatible with any board that has an Arduino UNO-compatible header. This sounds easier than it is, as the Arduino UNO already comes with two different set of headers. Figure 1 below shows the original UNO on the left and the R3 version on the right.

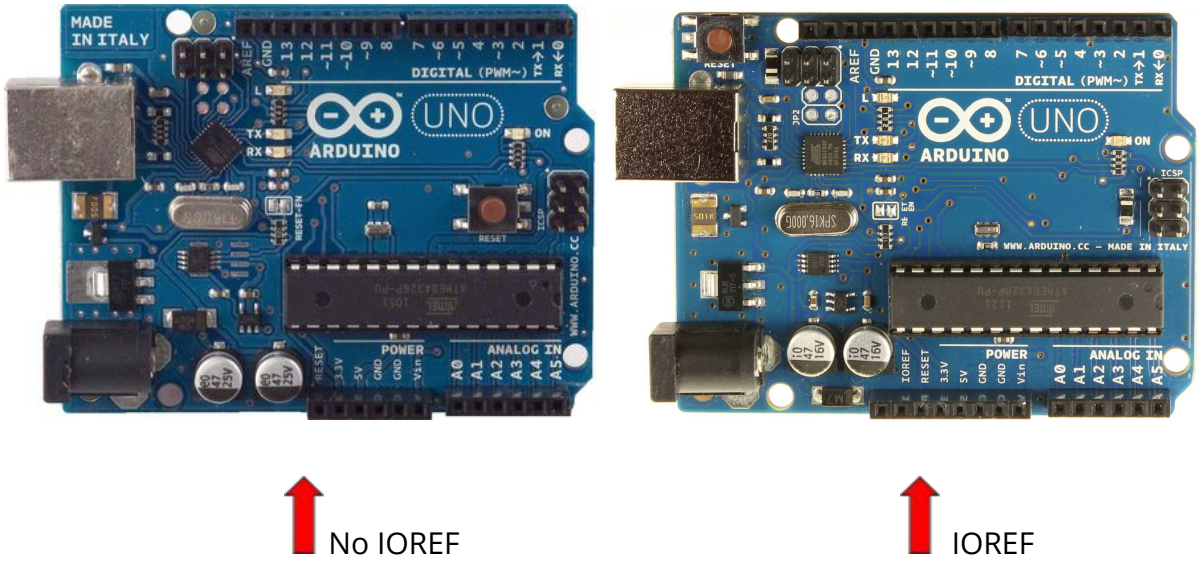


Figure 2. Arduino UNO header without (left) and with (right) IOREF PIN.

Older versions of the Arduino UNO contain no IOREF pin (compare bottom header to the right of the RESET line). In order to be compatible with both versions, we added Jumper 1

(see Figure 1). More on it later. For now it is important to know that by default, on an Arduino UNO R3 or an Arduino MEGA2560 R3, the MOVI shield will use the following pins:

- VIN and GND — to draw power
- D10 and D11 — for serial communication
- IOREF — to determine the voltage for serial communication

Making MOVI compatible with other boards completely depends on the location and functionality provided by these 5 pins.

In detail:

- Power supply: There must be enough voltage and current on VIN to power the MOVI shield (5.20–16 V). In most cases, this requires an external power supply as USB power easily drops below 5V.
- D10 and D11 must be available for serial communication. On boards with ATMEGA processor (e.g. UNO, MEGA, Leonardo, Duemilanova) this is the case. On boards with Intel (e.g. Edison, Galileo) or ARM processor (e.g. Due) , serial communication needs to be rewired using Jumper 2 and 3 (see Figure 1).
- The IOREF pin indicates the voltage used for serial communication. In older boards, this voltage was always 5 V and there was no need for IOREF. Therefore, we recommend to set J1 on an older 5 V board without IOREF pin. **Caution: J1 hardwires the 5 V pin to IOREF. Therefore, setting J1 on a 3.3 V board will destroy the board!** If a board has gray headers it is a 3.3 V board. If it has black headers it might be either a 3.3 V board or a 5 V board.

In the following, we will detail the setting of the jumpers and the power supply needs for the boards that we tested.

Uno R1 and R2, MEGA2560 R1 and R2, Leonardo R1 and R2

Close Jumper 1, 2 and 3. Use external power supply within the Arduino-allowed range but at least 5.20 V.

Uno R3, Mega2560 R3, Leonardo R3

Keep Jumper 1 open. Close Jumper 2 and 3. Use external power supply within the Arduino-allowed range but at least 5.20 V.

Freeduino

Freeduino is a free clone and is compatible to Arduino Duemilanova. Close Jumper 1, 2, and 3. Set Freeduino's power supply jumper to use the external power supply (that you need to connect). Use external power supply within the Freeduino-allowed range but at least 5.20 V.

Olimexino-328

Olimexino-328 is compatible to Arduino Duemilanova. Switch Olimexino's voltage selection switch to 5V and close MOVI's Jumper 1, 2, and 3. Since the Olimexino does not have an IOREF pin, we do not recommend the 3.3 V operation. **Also, be warned that setting the switch to 3.3 V and closing Jumper 1 may destroy the Olimexino board.** Therefore set the switch to 5 V before powering the board. Use external power supply within the Olimexino-allowed range but at least 5.20 V.

Diavolino

Diavolino is essentially a Freeduino. However, by default the board comes without external power supply support. Solder the external power supply support on the board and use it to power the board. Close MOVI Jumpers 1, 2, and 3. Use external power supply within the Diavolino-allowed range but at least 5.20 V.

Arduino Yun

The Arduino Yun does not support an external power supply. Also, the Ethernet plug makes it hard to mount MOVI in a mechanically stable fashion. To solve the latter problem, we therefore recommend to use a (smaller) blank shield between the Arduino Yun and the MOVI shield. To solve the power supply problem, connect an external power supply with 5.10V–6 V to a GND header pin (-) and the VIN pin (+). Leave Jumper 1 open but close Jumper 2 and 3.

Arduino Due

The Arduino Due is a 3.3 V board with an ARM chip that does not support serial communication on pins D10 and D11. In order to operate MOVI with it is important to **keep Jumper 1 open** and also open Jumpers 2 and 3. **Closing Jumper 1 will destroy the Due board!** Connect the right side of MOVI's TX jumper (Jumper 2) to RX1 (D19) and the right side of MOVI's RX jumper (Jumper 3) to TX1 (D18) of the Arduino Due board using jumper wires. The right side is the pin that is closer to the power and USB plugs. Use an external power supply within the Arduino Due allowed range but at least 5.20 V. To see Serial Console messages on the Due connect the "Programming" USB to your computer while in the Arduino IDE.

Intel Galileo Gen 2

The Intel Galileo Gen 2 is a 5 V board with IOREF pin. Unfortunately, Intel does not support serial communication on pins D10 and D11. Keep Jumpers 1, 2 and 3 open. Connect the right side of the MOVI's TX jumper (Jumper 2) to RX (D0) and the right side of MOVI's RX jumper (Jumper 3) to TX (D1) of the Intel Galileo Gen 2 board using jumper wires. The right side is the pin that is closer to the power and USB plugs. Use an external power supply within the Intel Galileo allowed range but at least 5.20 V.

Intel Edison

The Intel Edison can operate shields at 3.3 V and 5 V, depending on a jumper setting on the Edison board. Since the boards supports the IOREF pin, both settings are equally valid as long as MOVI Jumper 1 is open. **Warning: Setting the Edison voltage selection pin to 5 V and closing MOVI's Jumper 1 might destroy the Edison board!** Unfortunately, Intel does not support serial communication on pins D10 and D11. Therefore keep Jumpers 2 and 3 open as well. Connect the right side of the MOVI's TX jumper (Jumper 2) to RX (D0) and the right side of MOVI's RX jumper (Jumper 3) to TX (D1) of the Intel Edison board using jumper wires. The right side is the pin that is closer to the power and USB plugs. Additionally, when we tested the Intel Edison, the VIN pin did not provide enough power to the MOVI shield. If you connected MOVI and you see the shield is not booting (no light on red LED after a couple seconds), connect an external power supply with 5.10V–6 V to a GND header pin (-) and the VIN pin (+).

Boards we were not able to get to work with MOVI

Unfortunately not all Arduino-compatibles are the same. Therefore, some boards might never work with MOVI and some others might work in the future. In general, boards that do not have any shield headers or where the headers are not compatible with the UNO will not work. For example, the Arduino Pro (and clones) has headers that are not compatible with MOVI, although the board might work with rewiring. The Arduino Zero, Zero Pro, M0 and M0 Pro are software-incompatible as they currently do neither support a software UART nor a second hardware UART. This might change in the future. The Intel Galileo Gen1 uses a 5 V power supply that does not supply enough voltage for MOVI.

B. MOVIShield Library Reference

The following describes the functions made available through the MOVIShield library, sometimes referred to as MOVI API (Application Programming Interface). The library and the examples are open source and hoped to be most self explanatory. Also, individual parameters or method names may change be added or become obsolete based on the open-source community process. We therefore provide this section only as a starting point.

The methods in the library are divided into several categories, depending on when they can be called in the Arduino execution process and also their potential frequency of use.

Methods that must be used in setup()

MOVI constructors:

`MOVI()`

Construct a MOVI object with default configuration.

`MOVI(bool debugonoff)`

Construct a MOVI object with optional Serial Console interaction.

`MOVI(bool debugonoff, int rx, int tx)`

Construct a MOVI object with different communication pins and optional Serial Console interaction. This constructor only works on AVR architecture CPU. Other architectures ignore the `rx` and `tx` parameters and use whatever pins are designated to `Serial1` (see also explanation in Appendix A).

Initialization methods:

`init()`

This init method waits for MOVI to be booted and resets some settings. If the recognizer had been stopped with stopDialog() it is restarted.

`init(bool waitformovi)`

This init method only initializes the API and doesn't wait for MOVI to be ready if the parameter is false.

`bool isReady()`

This method can be used to determine if MOVI is ready to receive commands, e. g. when MOVI has been initialized with init(false).

`bool addSentence(String sentence)`

This method adds a sentence to the training set. Sentences must not contain any punctuation or numbers. Everything must be spelled out. No special characters, umlauts or accents. Uppercase or lowercase does not matter. This function can also be used with the F() macro. See: http://playground.arduino.cc/Learning/Memory

`bool train()`

This method checks if the training set contains new sentences since the last training. If so, it trains all sentences added in this MOVI instance. Once training is performed, no more sentences can be added and training cannot be invoked again in the same instance.

`callSign(String callsign)`

This method sets the callsign to the parameter given. If the callsign has previously been set to the same value, nothing happens. Only one call sign can be trained per MOVI instance and `callsign` must be one word and cannot contain any special characters or numbers. The callsign can be the empty string, however. In this case, MOVI will react to any noise above the threshold (clapper mode).

Methods that are typically used in setup() but can also be used in loop()

`setVolume(int volume)`

Sets the output volume for the speaker port. The value for `volume` is expected to be between 0 (mute) and 100 (full). The default is 100. Values between 0 and 100 may be set to approximated values due to internal technical details of the sound card. For example, `setVolume(20)` may result in a volume of 16.

`setVoiceGender(bool female)`

This method sets the gender of the speech synthesizer. `True` being female. `False` being male.

`setThreshold(int threshold)`

Sets the noise threshold of the recognizer. Values vary between 2 and 95. Factory default is 5. Depending on the noise of the environment MOVI may have difficulty distinguishing between noise and speech and may wait very long for a sentence to end. Increasing the noise threshold will help. Typically a value of 15 is good for somewhat noisy environments and a value of 30 for very noisy environments. Ultimately, experimentation is the only way to determine a good noise threshold.

`responses(bool on)`

Turns the spoken responses as a result of recognition events (e.g. silence or noise) on or off.

`welcomeMessage(bool on)`

Turns off the spoken welcome message indicating the call sign.

`beeps(bool on)`

Turns the recognition beeps on or off.

Methods that are typically used in loop()

`signed int poll()`

This method is the most important method. It is called in `loop()` to get an event from the recognizer. 0 stands for no event. A positive number denotes a sentence number. A negative value defines an event number. Event numbers are the negatives of the numbers displayed on the serial monitor. For example: `MOVIEvent[200]` would return -200. The possible events are listed in Appendix D.

`String getResult()`

Gets the result string of an event. For example: `MOVIEvent[201]: LET THERE LIGHT` results in "LET THERE BE LIGHT\n". The resulting string might need trimming for

comparison to other strings. The resulting string is uppercase and does not contain any numbers, punctuation or special characters.

`say(String sentence)`

Uses the internal synthesizer to make MOVI speak the sentence given as parameter using the speech synthesizer. This function can also be used with the F() macro. See: http://playground.arduino.cc/Learning/Memory

`ask(String question)`

This method instructs MOVI to speak the sentence given as first parameter using the synthesizer and then directly listen without requiring a callsign. A programming demo of how this function is used can be found in the Eliza library example. This function can also be used with the F() macro. See: http://playground.arduino.cc/Learning/Memory

`password(String question, String passkey)`

Similar to ask, this methods makes MOVI speak the sentence given as first parameter using the synthesizer. Then MOVI's password function is used to query for a password. The API compares the passkey with the password and returns either `PASSWORD_REJECT` or `PASSWORD_ACCEPT` as an event. The passkey is not transferred to or saved on the MOVI board. While all password attempts are passed over the serial communication, the only board that knows the right answer is the Arduino. It compares the password attempts and sends the appropriate event. IMPORTANT: The passkey must consist only of words contained in the trained sentences and must not contain digits or other non-letter characters except one space between the words. A demo of how this function is used can be found in the AlarmPassword library example. The first parameter of this function can also be used with the F() macro. See: http://playground.arduino.cc/Learning/Memory

Infrequently used advanced commands

`sendCommand(String command, String parameter)`

Sends a command manually to MOVI. This allows to send any command to MOVI that is defined in the low level interface (see also subsequent section).

`float getFirmwareVersion()`

Returns MOVI's firmware version.

`float getHardwareVersion()`

Returns MOVI's board revision.

`float getAPIVersion()`

Returns the version of the library.

`stopDialog()`

Stops the recognizer and synthesizer without powering down the MOVI hardware.

`restartDialog()`

Restarts the recognizer and synthesizer manually (e.g. after a `stopDialog`).

`factoryDefault()`

Resets MOVI to factory default. This method should only be used in `setup()` and only if needed. All trained sentences and callsigns are untrained. The preferable method for a factory reset is to use the serial monitor or a long press on the MOVI's reset button.

`~MOVI()`

Destructs the MOVI object. As of today, frankly, we haven't found any scenario in which this method would ever be called.

C. The Low Level Interface

The MOVIShield library is an easier way to program MOVI, especially when used by modifying the examples that come with it. However, in order to have access to the full functionality of the board, including some useful debugging features, it will be necessary to use the low level interface from time to time. The low level interface is accessible through the Serial Console of the Arduino IDE when a *MOVI* object is constructed using the value true for the debug parameter. Optionally, we included a low level interface sketch in the library examples. The sketch can be run without the MOVI library. Figure 3 shows a screenshot.

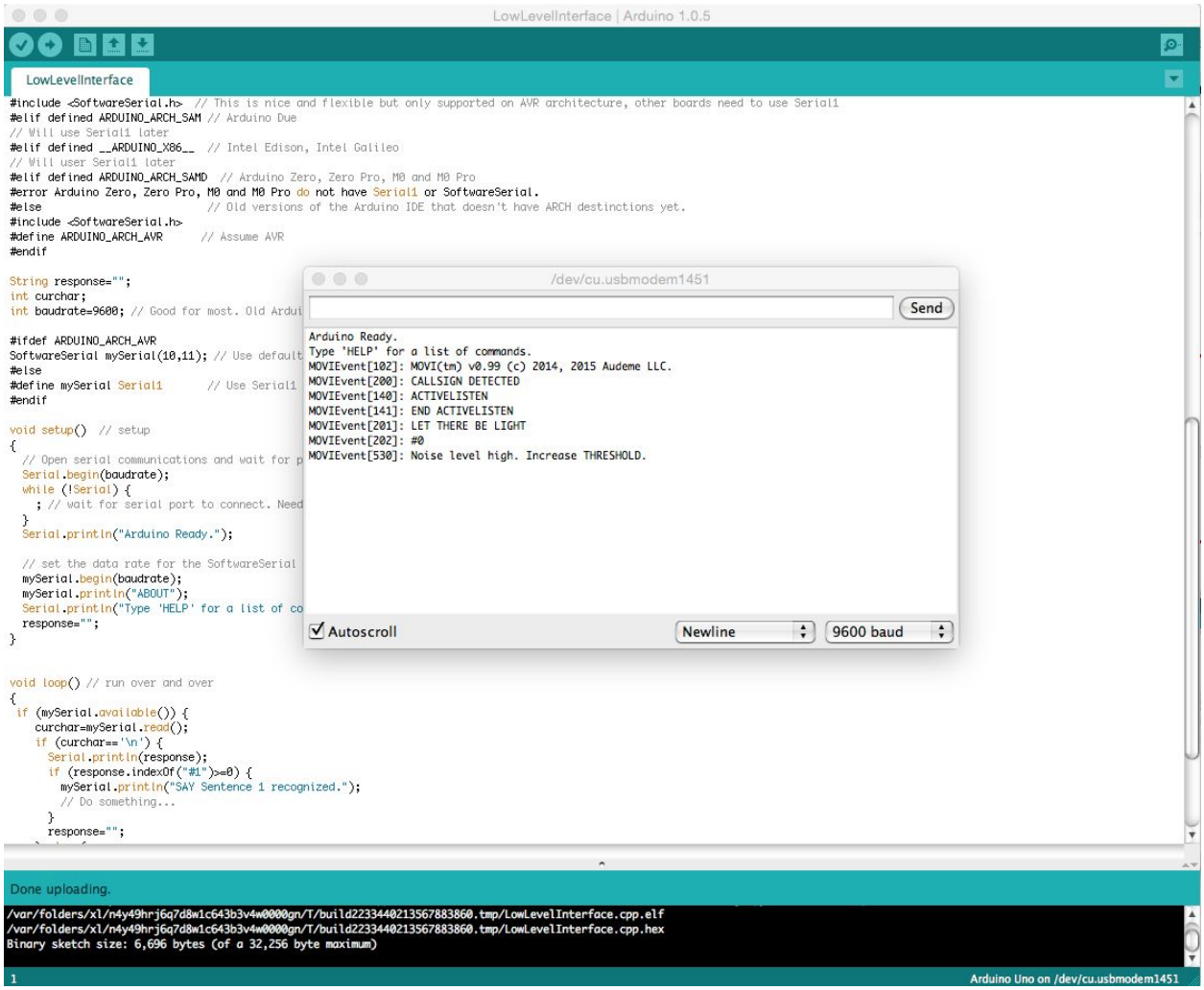


Figure 3. Using the Low Level Interface.

Low level commands are all uppercase and the responses are MOVIEvents. Most of the commands are self descriptive.

Note: When using the USB host capabilities of the Arduino Leonardo, invoking the Serial Console may take some extra seconds as the board needs to reconfigure its serial interface. To see Serial Console messages on the Due connect the "Programming" USB to your computer while in the Arduino IDE.

`HELP`

Shows the list and usage of the manually-usable low-level commands.

`SAY <sentence>`

Speak the sentence through the synthesizer. Sentences can include numbers and punctuation. Corresponds to the method of the same name in the API.

`SHUTDOWN`

Shuts the underlying Linux system on the shield down.

`ABOUT`

Returns copyright messages as shield event.

`VERSION`

Returns the version of this software.

`HWVERSION`

Returns the version of board circuit.

`PING`

Returns "PONG" as a shield event (or not, if the shield is not properly working)

`VOLUME <percentage>`

Sets the output volume between 0-100 and returns the new volume as shield event. Corresponds to the method in the API.

`STOP`

Disables all recognition and ignores all SAY commands until "RESTART". Corresponds to `stopDialog()` in the API.

`RESTART`

Can be used anytime to reset the speech recognition and the speech synthesizer. No retraining is performed. Corresponds to `restartDialog()` in the API.

`FACTORY`

Reset the shield to factory settings. Trained vocabulary, call-signs and settings are reset as well. Corresponds to the API method as well as the long-press of the reset button.

`ASK`

Perform a single recognition cycle without call sign.

`PASSWORD`

Perform a single recognition cycle without call sign. DO NOT correct the raw results.

`FEMALE`

Switch the synthesizer to female voice

`MALE`

Switch the synthesizer to male voice (default)

`VOCABULARY`

Output the trained sentences to the serial console

`CALLSIGN [<word>]`

Change the call sign to a new word. If the word is empty, any sound activity will trigger a recognition cycle.

`TRAIN`

Manually train the recognizer to recognize the sentences. System will prompt. Sentences are separated by \n and end with '#'. '@' aborts. This method is inherently not thread safe and should only be used manually. This command is intended to be used for debugging and to save memory when training a large set of sentences.

`SYSTEMMESSAGES <"ON"|"OFF">`

Toggle synthesizer messages like "System is being shutdown" or "System is booting".

There is no corresponding API method as this method will not reset with a new MOVI object.

`RESPONSES <"ON"|"OFF">`

Toggle synthesizer responses like "I did not understand you". Corresponds to the API method.

`BEEPS <"ON"|"OFF">`

Toggle recognition cycle beeps. Corresponds to the API method.

`WELCOMEMESSAGE <"ON"|"OFF">`

Toggle synthesized welcome message and call sign. Corresponds to the API method.

`THRESHOLD <percentage>`

Set the sound activity threshold. Valid percentages are between 2 and 95. Factory default is 5. This method exactly corresponds with the API method. See description there for further explanation.

`MICDEBUG <"ON"|"OFF">`

Toggles microphone debug mode. `RESTART` required for change to take effect. In this mode all microphone activity above threshold is echoed. This methods is very valuable for debugging environmental noise or other microphone issues, especially in connection with `THRESHOLD`.

`MEM`

Display memory usage. Useful for debugging potential memory overflows in extensive training scenarios with hundreds of sentences.

`INIT`

This function is used in the MOVI API when a new MOVI object is instantiated. It resets certain settings and restarts the recognizer if stopped. It returns shield event 101 containing versioning information.

`NEWSENTENCES`

This command is used in the MOVI API to declare a new set of to-be-trained sentences.

`ADDSENTENCE <sentence>`

This command is used in the MOVI API to add a sentence to the current set of to-be-trained sentences.

`TRAINSENTENCES`

This command is used in the MOVI API to invoke training the set of to-be-trained sentences. This command does nothing if the set of trained sentences and the set of to-be-trained sentences are equal.

D. MOVI Event Categories

MOVI returns events over the serial communication line with 9600 bps as a result of either the execution of a command or extrinsic or intrinsic board events (e. g.shutdown or callsign detected).

The format of the events is

```
MOVIEvent[<eventno>]: <textual description>
```

The textual description is user readable and can change with version to version. The eventno is meant to be interpreted by the machine. *poll()* will return 0 for no event, a positive number when a sentence was recognized and -eventno for an event.

The events itself are grouped into the following categories:
Events in the 0–99 range are to be ignored by the library or programs using MOVI as they constitute debugging output readable only to the user.
Event 100 is pong.
Events 101–110 are defined for versioning checks of the device.
Events 111–199 are defined for other status messages.
Event 200 is callsign detected.
Events in the 201–299 are responses to commands.
Events in the 400 range denote errors.
Events in the 500 range denote non-speech states.

The most frequently used events are defined with *#define* macros in the MOVI library for easy use with the *poll()* command. These are:
0 (`SHIELD_IDLE`) Not an actual MOVI event, returned by *poll()* when nothing happened.
140 (`BEGIN_LISTEN`) MOVI starts to listen (after call sign)
141 (`END_LISTEN`) MOVI stops listening (after timeout or as per energy detector)
150 (`BEGIN_SAY`) MOVI starts speaking in the synthesizer
151 (`END_SAY`) MOVI stops speaking
200 (`CALLSIGN_DETECTED`) Call sign was detected
201 (`RAW_WORDS`) This event contains the raw words (to be returned with *getResult()*)

204 (`PASSWORD_ACCEPT`) Password accepted (generated by library after *password()* call)

404 (`PASSWORD REJECT`) Password rejected (generated by library after *password()* call)

530 (`NOISE_ALARM`) Too much noise.

501 (`SILENCE`) Empty sentence (silence or noise).

502 (`UNKNOWN_SENTENCE`) Unmatchable result. This happens when two or more trained sentences could equally by matched to the recognition result.

E. Terms and Conditions

These Terms and Conditions (the "Agreement") shall govern the rights and obligations of the parties with respect to the sale of products from Audeme LLC, a California limited liability company (the "Seller"), to a party placing an order from Seller (the "Buyer"), as set forth in the attached invoice. Buyer, by accepting delivery of the products, accepts and agrees to abide by this Agreement.

No contractual relationship between Seller and Buyer shall arise until such time as Buyer has placed an order that has been accepted by Seller.

1. Taxes. Prices on the products are exclusive of all city, state, and federal excise taxes, including, without limitation, taxes on manufacture, sales, receipts, gross income, occupation, use and similar taxes. Wherever applicable, any tax or taxes will be added to the invoice as a separate charge to be paid by the Buyer.

2. LIMITED WARRANTY. SELLER WARRANTS THE PRODUCTS FOR A PERIOD OF THIRTY (30) DAYS FOLLOWING DELIVERY. SELLER'S RESPONSIBILITY UNDER THIS WARRANTY IS LIMITED TO REPLACING DEFECTIVE PRODUCTS. IF SELLER IS UNABLE TO REPLACE THE PRODUCTS, BUYER WILL BE ENTITLED TO A REFUND OF THE AMOUNT PAID BY BUYER FOR THE DEFECTIVE PRODUCTS. SELLER'S WARRANTY LIABILITY SHALL IN NO CASE EXCEED THE ORIGINAL COST OF THE DEFECTIVE PRODUCTS. THIS WARRANTY SHALL BE THE ONLY WARRANTY MADE BY SELLER, AND IS IN LIEU OF ALL OTHER WARRANTIES, EXPESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ALL OF WHICH OTHER WARRANTIES ARE HEREBY EXPRESSLY SELLER DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF OR THE RESULTS OF THE USE OF THE PRODUCTS IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE AND DOES NOT WARRANT THAT THE OPERATION OF THE PRODUCTS WILL BE UNINTERRUPTED OR ERROR FREE.

3. CLAIMS. BUYER MUST NOTIFY SELLER OF ANY CLAIM OF DEFECT OR ANY OTHER CAUSE WITHIN THIRTY (30) DAYS AFTER DELIVERY OF THE PRODUCTS. IF BUYER FAILS TO GIVE NOTICE OF A CLAIM WITHIN SUCH TIME PERIOD, BUYER EXPRESSLY WAIVES ANY SUCH CLAIMS. BUYER UNDERSTANDS THAT ANY FAILURE TO NOTIFY SELLER OF A CLAIM WITHIN SUCH THIRTY (30) DAY PERIOD SHALL BE DEEMED A COMPLETE DISCHARGE OF SELLER'S OBLIGATIONS AND THAT BUYER SHALL THEREAFTER HAVE NO REMEDY AGAINST SELLER.

4. Security Interest. Seller shall retain a purchase money security interest in the products (as defined in the Uniform Commercial Code) until Buyer has made complete payment for the products,

notwithstanding any prior delivery of the products by Seller to Buyer. Buyer hereby agrees, upon the request of Seller, to join with Seller in executing one or more financing statements pursuant to the Uniform Commercial Code in form satisfactory to Seller.

5. Liability Limitation. Seller's liability under this Agreement shall not exceed the amounts paid by Buyer for the products. SELLER SHALL IN NO EVENT BE LIABLE FOR ANY LOSS OF PROFITS; LOSS OF USE OF FACILITIES, EQUIPMENT, OR SOFTWARE; OR OTHER INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES.

6. Use of Products. It is understood and agreed that use of Seller's products shall be fully at the risk of Buyer. Without limiting the foregoing, Buyer represents and agrees that the products will be used solely for personal purposes, and not for any business, industrial, or commercial purpose, or other applications involving potential risks of death, personal injury, or property or environmental damage, including but not limited to medical or life support applications, or use in aircraft, aircraft devices, or aircraft systems.

7. No License. Products or any parts thereof sold hereunder may be protected by intellectual property rights of Seller, including, but not limited to, rights under issued and pending patents, mask work rights, copyright rights, trademark rights and trade secret rights. Neither the sale of products or any parts thereof hereunder nor the provision by Seller of any supporting or related documentation, technical information or advice shall confer on Buyer any license, express or implied, under any intellectual property rights of Seller.

8. Entire Agreement. This Agreement contains the entire agreement of the parties and supersedes any prior written or oral agreements between them concerning the subject matter contained herein. There are no representations, agreements, arrangements, or understandings, oral or written, between the parties, relating to the subject matter contained in this Agreement, which are not fully expressed herein.

9. Amendments. This Agreement may be amended only by a written amendment signed by each of the parties. The terms and conditions of any purchase order or similar document issued by Buyer shall not be binding against Seller unless signed by Seller.

10. Binding Effect. All terms and provisions of this Agreement shall be binding upon and shall inure to the benefit of, and be enforceable by, the respective assigns and successors of the parties; provided, however that Buyer may not assign this Agreement or any part thereof without the prior written consent of Seller.

11. Notices. All notices required or permitted by this Agreement shall be in writing and shall be addressed to the parties at the respective addresses specified on the foregoing invoice form. Notice shall be sufficiently given for all purposes as follows: (a) when personally delivered to the recipient, in which case notice is effective on delivery; (b) when mailed by certified mail, postage prepaid, return receipt requested, in which case notice is effective on receipt if delivery is confirmed by a return receipt; or (c) when delivered by overnight delivery service, charges prepaid or charged to the sender's account, in which case notice is effective on delivery if delivery is confirmed by the delivery service. Either party may change its address by giving written notice thereof to the other party in accordance with the provisions of this paragraph.

12. Governing Law. This Agreement shall be governed by and construed in accordance with the laws of the State of California.

13. Waiver. The waiver by Seller of any provision of this Agreement shall not be deemed to be a waiver of any other provisions hereof or of any subsequent breach by Buyer of the same or other provisions. The consent or approval by Seller of any act taken by Buyer shall not be deemed to render unnecessary the obtaining of Seller's consent to or approval of any subsequent similar act by Buyer.

14. Severable. Any provision of this Agreement that shall prove to be invalid, void, or illegal shall in no way affect, impair, or invalidate any other provision hereof, and such remaining provisions shall remain in full force and effect.